# Self-repair during Continuous Motion with Modular Robots

Robert Peck

Ph.D.

University of York
Electronic Engineering

September 2021

# Abstract

Through the use of multiple modules with the ability to reconfigure to form different morphologies, modular robots provide a potential method to develop more adaptable and resilient robots. Robots operating in challenging and hard-to-reach environments such as infrastructure inspection, post-disaster search-and-rescue under rubble and planetary surface exploration, could benefit from the capabilities modularity offers, especially the inherent fault tolerance which reconfigurability can provide. With self-reconfigurable modular robots self-repair, removing failed modules from a larger structure to replace them with operating modules, allows the functionality of the multi-robot organism as a whole to be recovered when modules are damaged.

Previous self-repair work has, for the duration of self-repair procedures, paused group tasks in which the multi-robot organism was engaged, this thesis investigates Self-repair during continuous motion, “Dynamic Self-repair", as a way to allow repair and group tasks to proceed concurrently. In this thesis a new modular robotic platform, Omni-Pi-tent, with capabilities for Dynamic Self-repair is developed. This platform provides a unique combination of genderless docking, omnidirectional locomotion, 3D reconfiguration possibilities and onboard sensing and autonomy. The platform is used in a series of simulated experiments to compare the performance of newly developed dynamic strategies for self-repair and self-assembly to adaptations of previous work, and in hardware demonstrations to explore their practical feasibility. Novel data structures for defining modular robotic structures, and the algorithms to process them for self-repair, are explained.

It is concluded that self-repair during continuous motion can allow modular robots to complete tasks faster, and more effectively, than self-repair strategies which require collective tasks to be halted. The hardware and strategies developed in this thesis should provide valuable lessons for bringing modular robots closer to real-world applications.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

A number of acronyms for newly developed algorithms, concepts and systems are used regularly throughout this thesis. Some acronyms from existing literature are also used.

| | |
|---|---|
| DMS | Dynamic Master Switching |
| DSR | Dynamic Self-repair |
| LM | Lone Module |
| LW+ | Liu and Winfield derived self-assembly |
| LW+MNS | Liu and Winfield derived self-assembly with Mergeable Nervous Systems features |
| MAS | Master of Another Substructure |
| MFM | Master to the Failed Module |
| MLR | Multi-Layered Recruitment |
| MRS | Master of the Removing Substructure |
| MNS | Mergeable Nervous Systems |
| nDSR | naive Dynamic Self-repair |
| nSSR | naive Static Self-repair |
| RM | Replacement Module |
| SSR | Static Self-repair |

# List of Supplementary Materials

Several videos are provided as supplementary material alongside this thesis, they can be found at the following URLs as well as attached with this thesis.

- Self-repair_concepts.mp4, also at [211]
- Early_hinge_test.mp4, also at [212]
- Docking_port_actuation_clips.mp4, also at [213]
- Hinge_actuation_tests.mp4, also at [214]
- Hardware_prototype_conference_demo.mp4, also at [215]
- Initial_MLR_hardware_tests.mp4, also at [216]
- MLR_hardware_demo.mp4, also at [217]
- Example_DSR_scenario.mp4, also at [218]
- Example_nDSR_scenario.mp4, also at [219]
- Example_SSR_scenario.mp4, also at [220]
- Example_nSSR_scenario.mp4, also at [221]
- Example_DMS_scenario.mp4, also at [222]
- Example_multi-failure_scenario.mp4, also at [223]
- DMS_testing_with_hardware.mp4, also at [224]

# Acknowledgements

I wish to thank my supervisors, Andy M. Tyrrell and Jon Timmis for help throughout my PhD, especially in the early planning stages.
I thank Mark Hough for introducing me to PCB design, providing crucial advice on unexpected practical bugs, and great work on producing machined metal hinge hubs to a tight schedule.
I thank Mark A. Post for providing an independent eye separate from my project to whom I could turn for debugging advice.
I thank Richard Redpath for improving my software development skills, and showing me structures to keep my code organised, especially in regards to using the TCP/IP networking suite from C.
I thank Andy White, Mike Angus, Dave Hunter and the rest of the Fourth Floor technicians for tirelessly keeping the 3D printers going throughout my marathons of plastic part production, and getting the department re-opened after the governmental over-reach of Spring 2020.
I thank my office colleagues in the Robot Lab group for their comradeship in the trials and tribulations of robotics research, and making a chilly portacabin the place I most longed to be.
I thank the various authors of open-source software projects mentioned in the bibliography for providing some of the incredibly useful tools which made this work feasible.
I also thank my family and close friends.
Finally I thank all those people across the world who opposed the lockdowns and the authoritarianism of 2020/21 to demand more precedented and proportionate policies, for giving me hope throughout the crisis by staying calm, standing firm and reminding us that the highest duty of a truly civilised society is the protection of liberties.

# Declaration

I declare that this thesis is a presentation of original work and that I am the sole author. This work has not previously been presented for an award at this, or any other, university. All sources are acknowledged in the bibliography. All images in this thesis are by the author except where stated otherwise. Some of the materials on which this thesis is based, and some of the figures within it, have been published in the following papers of which I am lead author:

- R.H.Peck,J.Timmis,A.M.Tyrrell, *Towards Self-repair with Modular Robots during Continuous Motion*, Proceedings. 2018 Towards Autonomous Robotic Systems, Bristol
- R.H.Peck, J.Timmis, A.M.Tyrrell, *Omni-Pi-tent: An Omnidirectional Modular Robot With Genderless Docking*, Proceedings. 2019 Towards Autonomous Robotic Systems, London, doi.org/10.1007/978-3-030-25332-5_27
- R.H.Peck, J.Timmis, A.M.Tyrrell, *Self-assembly and Self-repair during Motion with Modular Robots*, In progress

Some elements in Chapter 2 are also related to a further paper on which I provided assistance:

- M.A.Post, J.Li, J.White, R.H.Peck, X.-T.Yan, P.Letier, *Modularity for the Future in Space Robotics, a Holistic Review*, MOSAR project technical report

Chapter 1

# Introduction

Robots today generally find themselves of most use in tasks described by “the 3 D’s", “Dull", “Dirty" and “Dangerous". Robots serve to perform tasks which would be repetitive, risky or unpleasant for a human to do, often in locations which are entirely inaccessible to humans. Specialised robots can enter locations where humans cannot, or dare not, go and can accomplish useful tasks within them, for some situations robotic inspection can proceed while the infrastructure continues to operate whereas human inspection would cause disruption by requiring infrastructure to be shut down for the duration of the inspection [174]. Sewers, water and oil pipelines, bridges, ships, railway lines, power stations and aero-engines, all require periodic checking. This vital infrastructure, lurking unseen beneath streets, inside pressure vessels, and in remote locations, can be reached by robotic systems to perform periodic inspections and detect wear and tear faults before they develop into catastrophic failures [184].

In those cases where catastrophic failure does occur robots can take on more dull, dirty and dangerous tasks during post-disaster rescue and cleanup. Further specialised robots can keep rescue workers out of harms way, and, for example, improve the chances of trapped survivors being reached before it is too late, by penetrating the rubble [51]. With an increasingly urbanised world [183] even natural disasters which do not result from infrastructure failure are increasingly likely to affect built-up areas where this kind of urban search-and-rescue mission would be necessary. Getting where no human can makes such robots a vital lifeline.

Continuing with the theme of inaccessible places we also see specialised robots finding uses in space missions, where the lower launch mass [170], minimal consumable requirements and lack of a need for a return journey [172], make robots very attractive as planetary explorers. Robots like the Mars Science Laboratory Curiosity rover [173] bring us insights into the secrets of distant worlds, and robots too are likely to play a vital role as humanity expands its reach towards industrialising the solar system. Robots for servicing and decommissioning of low-earth-orbit satellites are already undergoing tests [175] [177] [178].

Specialised robots can play a vital role in keeping the modern world running, pre-empting and preventing disaster, recovering from disasters and expanding human frontiers. Presently robots for these tasks are each highly specialised for very specific jobs [185]. Generally, a new design must be developed for each application, and any surprises found in the field often mean the specialised robot can no longer perform as expected. Specialised robots are expensive to develop and quickly become useless if the mission scenario differs too far from the expected parameters [33].

## 1.1 Autonomous Robots for Tough Environments

Robots, though widely used today, are often static, remote controlled, or both. Whilst robotic arms have populated factories since Unimate in 1961 [139], industrial use of mobile robots, outside of centrally-controlled material handlers in highly controlled environments, remains rarer. Autonomous robots remain rarer still with the majority of mobile robots in use for tasks such as planetary exploration, bomb disposal, mobile material handling, underwater imaging, and infrastructure inspection being tele-operated [181].

Teleoperation, however, can be a seriously limiting factor in what tasks a robot can perform. A human operator can struggle to judge sensor data correctly so as to, for example, apply the correct force to turn a handle without tearing it off [182]. Even with ongoing haptics research into more intuitive ways to provide operators with sensor feedback, time-lag between an operator's commands being sent and the robot performing the intended actions can make controlling robots difficult. Problems become apparent above a 400ms delay [153] [154], time-lags of the multi-minute levels experienced in space exploration applications greatly limit the tasks a tele-operated robot can attempt and slow the execution of those tasks such a robot can still be used for. Challenging environments often prove difficult for standard methods of communication for remote-controlled robots, the kinds of industrial locations where infrastructure monitoring takes place often feature metal walls shielding a robot from the radio signals a remote operator would need to supply. Tether cables for back and forth data-links limit a robot's manoeuvrability [44], and where wireless communication is possible over long enough ranges there is often enough interference to disrupt high data-rate communications, such as video feeds, an operator may require. On top of all this, teleoperation of robots requires the attention of a human operator, the extent to which such robots can be deployed is limited by the number of operators available and the hours of painstaking work they can remain focused for.

Equipping robots with the capabilities to make decisions for themselves, free of dependence on human operators and systems external to the robots, mitigates these difficulties in tele-operation, at the price of adding new challenges for the roboticist to solve. Challenges for autonomous mobile robotics are heavily connected to the ways in which autonomous robots respond to unexpected situations [156] and unstructured environments.

Faults in subsystems within robots often provide those kind of unexpected situations. For robots operating autonomously, or even under human control in an inaccessible environment, faults can develop in to failures. And failures can prove fatal, both to the robot, the mission it is undertaking, and sometimes to lives depending on the successful completion of the task. A fault tolerant system can continue operating properly in the presence of, at least certain types of, faults without failing. Ensuring that degradation of performance is graceful and gradual rather than sudden is also a much sought-after property for engineered systems performing critical tasks. Studies [164] of even tele-operated and ruggedised military ground robots find a mean time between terminal (mission ending) failures of only 7.7 hours, of those failures which occured between 80% and 94% were terminal depending on the task being undertaken. There is a desperate need for more reliable robots, especially when they are to be trusted with autonomy too.

Active [162], or explicit [163], fault tolerance handles faults by making changes to the system, hardware or software, to account for the presence of faults. This contrasts with passive, or implicit, fault tolerance in which a system is designed with redundancy inbuilt so that operation can continue even for predicted types of failure. In its fullest form, although less detailed methods still have practical uses, active fault tolerance can be broken into stages of [158] [161] fault detection, identifying that a fault exists, fault isolation and identification, diagnosing which subsystem is faulty and the nature of the fault, and fault recovery, making the system changes necessary to continue operating, although perhaps at reduced levels of performance than normal, despite the fault. Implicit fault tolerance is limited in what forms of fault in can be useful for [163] and could often require significantly over-engineered systems to parallelise backup systems in ways which do not conflict with one-another if one fails. For example, every motorised actuator would require at least a complex, bulky, and expensive dual drive actuator, such as [167], fitted within it.

Multi-robot systems such as swarms have the possibility to provide both passive and active fault tolerance, being able to use distributed methods to detect, diagnose and recover from faults. Multi-robot teams were initially considered fully fault tolerant simply by passive means [148] as it was expected that all tasks attempted would be entirely parallel, work such as ALLIANCE [108] considered distributed methods by which robots could be assigned tasks and switch roles in cases of failure. Later work [149] found that even a single failed robot could cause problems throughout a swarm, unless a detection and diagnosis strategy was used [150] [151] [152]. Although the chance of a fault occurring somewhere in a group of multiple robots rises as $1 - Probability(Failure)^n$ with the number of interacting robots, the probability of all robots in the group developing faults drops as $Probability(Failure)^n$. Hence *if* the consequences of a fault in one, or several, robots can be mitigated, multi-robot systems can be highly fault tolerant. Swarm robots, under their strict definition [168], do not connect and use only entirely distributed control strategies, so may not be appropriate for the wide variety of tasks that autonomous mobile robots of the future will be desired for. However, the promise of fault tolerance through multi-robot systems of similarly manufactured units is still an idea worth pursuing by other routes [156].

## 1.2 Modular Robots

Modular robots are robotic systems formed from multiple discrete modules, rather than as single specialised monolithic devices. They have some similarities to swarm robots, in that single modules are usually incapable of the overall task being performed [145], but differ in that modular robots can physically connect together when operating together. The multi-module structures which can be assembled can be considered as “organisms" with each module analogous to a biological cell. Once formed, these organisms will typically have abilities beyond those of a single lone module [33], especially in terms of the forces they can exert on objects in the environment [160]. Compared to ordinary monolithic robots, modular robots will, once a matured technology, enjoy a variety of advantages all stemming from the fact that extending the robotic organism becomes a matter of adding additional modules, not always homogeneous units and possibly specialised sensor or tool modules, rather than significant redesign and rebuilding as would be necessary to add functionality to a monolithic robot. This also simplifies production, many copies of common units need manufacturing rather than large specialised robots for different purposes [136]. This should reduce costs for modular robotic systems, making them not only a promising replacement for specialised monolithic robots, but also enabling them to be financially viable in scenarios where the cost of a specially designed robot would be prohibitive.

Self-reconfigurable modular robots are able to reconfigure between morphologies for their group organism without human intervention, platforms such as SMORES [35], SYMBRION [36], Ubot [37] and HyMod [42] have individually mobile modules with docking equipment which are able to shift between discrete lattice positions or relative locations in continuous space, depending on the architecture of the system, to form different shapes.

An especially notable capability of self-reconfigurable modular robots is their potential to self-repair [185], that is to say eject failed modules and bring in spares to replace them, this could prove to be a game-changing feature of such systems in contrast to existing robots. Modularity therefore provides resilience to faults as well as adaptability to unstructured environments, as a modular robotic organism will usually contain a large number of redundant elements [18], when in any given configuration, and these can be harnessed to replace faulty or failed modules. In some circumstances perhaps swapping modules so one with damaged sensors can still serve as an actuator and one with damaged actuation systems can still provide sensor data to the rest of the organism. As robot failures often prove field repairable [164], that is to say that simply replacing faulty subsystems is often adequate to get failed robots back into operational condition, the concept of self-repair by replacing failed modules with working ones appears particularly promising for improving real-world robot reliability.

This thesis develops new strategies and capabilities for modular robots to recover from faults and failures within individual modules, without compromising the entire system.

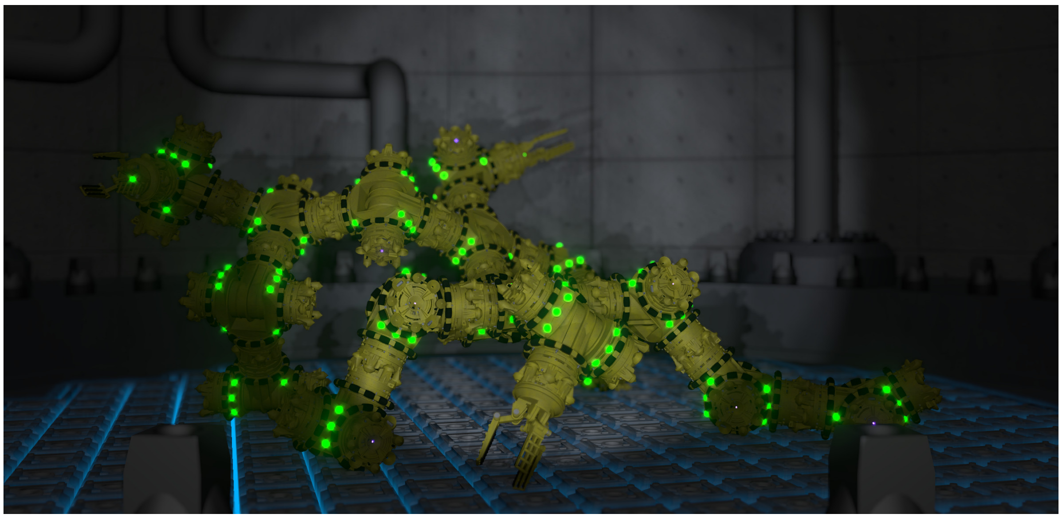


**Fig. 1.1:** R.H.Peck's artist's impression of a modular robotic organism inspecting a nuclear reactor.

In plausible near-future applications one would expect tens or hundreds of macro-scale modules to form structures with motion and reconfiguration abilities which can be exploited to act like existing robot designs, for example: quadrupeds, hexapods, snakes, or arms on wheeled bases. Unlike such existing robot designs modular robotic structures will also have the ability to transform between such forms, and to adapt to complications such as failures or needs for on-the-spot addition of extra sensory or actuation capabilities. An arm on a mobile base constructed from modular robots could shift robots from the base to form a counterbalancing weight, or an extra arm with a camera turret at its tip to help guide the use of an end-effector tool module docked to the tip of the main “arm". These kinds of modular robots can form a snake-like morphology to gain access to a location of interest by passing: along a pipe, in an infrastructure monitoring scenario, under rubble, in a post-disaster situation, or in to narrow cave entrances, in a planetary exploration scenario. Once inside they may reconfigure to a many-legged walker to cross uneven surfaces [30], before reconfiguring again to a mobile manipulator with arms to hold tools and sensors once at the location of interest. Groups of modules could detach from the whole to: allow inspections where forces applied to one part of a structure are measured at other points [30] [176], follow other routes through rubble while the main group tries to dig out a discovered survivor, or remain in place to act as a communication relay back to a rover waiting on the Martian surface while the rest of the modules scout a lava tube. Detached modules can later reattach if larger morphologies are required. Fig.(1.1) provides a depiction of how modular robots in an infrastructure monitoring scenario may appear.

This thesis concerns itself with modular self-reconfigurable robots of this macro-scale variety, and does not expect the findings to be particularly applicable to more hypothetical modular systems consisting of tens of thousands of micro-scale modules and acting as programmable matter [33]. Such programmable matter systems would likely be subject to both constraints (perhaps on per-module processing power) and freedoms (perhaps in terms of modules transported by capillary forces [7], or on air currents) the like of which cannot be accurately conceived at present. For the types of modular robotic applications we can expect in the near future the question of what is the right number of modules may have a lot more importance than a focus on working with the largest number of modules possible [93]. This thesis assumes that modular robots in the near future will have levels of onboard processing power comparable to those which we have come to expect from lightweight battery powered consumer devices, and will not be subject to some of the more stringent conditions which some roboticists [135] in the swarms discipline imagine, such as modules being unable to explicitly communicate or present any unique identifying numbers. It is thought that by considering the near future requirements of modular robotic technology, and working to assumptions driven largely by the practicalities involved in modular robotic hardware development, that the results of this thesis should be applicable to the field of modular robotics at the present time and its advance towards field-ready modular self-reconfigurable systems.

## 1.3 Hypothesis

Although further developed in Chapter(2) the overall question throughout this thesis is introduced at this point. Modular robots, especially of the self-reconfigurable kind considered throughout this thesis, will, once a mature technology, find themselves undertaking difficult tasks in challenging environments. Failures are inevitable, but we hope that, for the system as a whole, they will not be impossible to recover from. We hope that though modules may be lost, the task they are set with achieving can still be completed. The ability to recover from failures, at least failures of some forms, will be of critical importance. Failures which put a system out of action for a period can, should that period be at a critical time, be just as harmful to task completion as fatal failures. A robot inspecting a high radiation area in a nuclear powerplant does not benefit from spending excessive time there [165] should it need to self-repair before retreating if one module succumbs to damage, a robot searching for survivors under a collapsed building should not find itself paralysed and unable to dodge falling debris when it is reconfiguring. The worst of such poorly timed attempts to recover from failures can themselves lead to fatal failures. It can therefore be hypothesised that:

**Hypothesis 1.1:** *Modular robots using self-repair strategies which can operate while the group maintains collective motion will be able to complete tasks faster, and more effectively, than if using a self-repair strategy which requires the system to halt collective tasks so as to perform self-repair.*

This thesis and the work described within it introduces in full the reasoning behind that hypothesis, develops the hardware and simulation infrastructure to test it, then performs a series of simulated and hardware experiments, collectively the tasks in this thesis are referred to as the Dynamic Self-repair Project. Starting from the outlines formulated in [31] this project has featured a wide variety of work, conceptualising, designing, simulating, developing, building and testing both robotic software and hardware, at all levels of the "stack" from mechanical elements up to multi-robot behaviour algorithms. The work in this thesis is presented thematically, not chronologically, however explanations are made where sections of the work are underpinned by other elements previously completed. While some aspects of this project have followed on from where SYMBRION [146] left off, in effect as successors to it within a subset of the research topics it considered, almost all the hardware and software infrastructure involved here has been produced from scratch. As much of SYMBRION's legacy has been lost since that project ended, the Dynamic Self-repair Project has, in the course of 4 years, with most work done by a single person and a budget on the order of £9.8K, produced a hardware platform and simulation infrastructure comparable to SYMBRION's atop which the experimental contributions to knowledge have been made. For comparison, SYMBRION's achievements took >19 researchers, 14 universities, 5 years and €7.4M [147] to get similar preparatory steps completed, including the development of new hardware platforms, simulation environments, and data structures to describe modular organisms. A key element, and technical contribution, of this work was the development of a new modular robot platform, called Omni-Pi-tent.

## 1.4 The Omni-Pi-tent Platform

As part of the Dynamic Self-repair Project a major element of the work in this thesis has been the development of a new self-reconfigurable modular robotic platform. Chapter 3 and Chapter 5 cover the details of this platform, however a brief introduction to it is provided here for clarity.

No previous platform has combined the feature set which Omni-Pi-tent offers [41], especially in terms of actuation and of docking and group motion capabilities. Having:

- an Omni-directional drive,

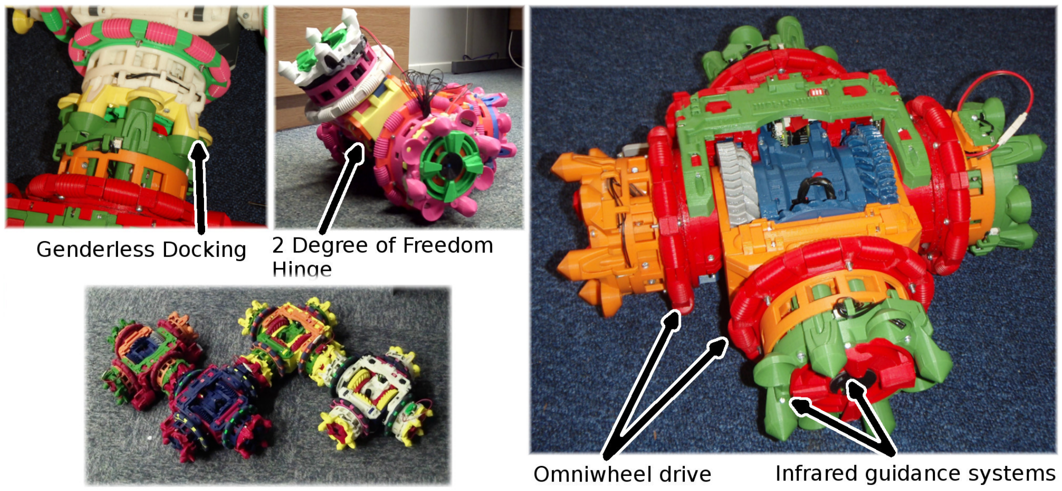


**Fig. 1.2:** Photographs of an Omni-Pi-tent module. Note: the four omniwheels allowing driving in any direction; the four genderless docking ports all capable of single-sided disconnection; and the 2 DoF hinge for 3D reconfiguration. The modules use Wi-Fi for global communication and 38KHz IR for line of sight communication. A 5KHz modulated IR system is used for docking guidance without interfering with the 38KHz communications.

- genderless docking,
- hinge systems for 3D reconfiguration,
- a combination of inter-module port-to-port and global communication methods,
- and an internal electronic architecture combining real-time AVR microcontrollers with a Raspberry Pi Linux single-board computer,

makes this a unique robot design. This is coupled with the use of 3D printing and entirely hand-solderable two-layer PCBs which allow production of modules with only moderately complex equipment. A total of 4 fully working modules, plus an initial prototype, were produced throughout the Dynamic Self-repair Project, providing a hardware reality to anchor the simulations against, and enabling real-world confirmation of the feasibility of some of the multi-robot strategies developed. Fig.(1.2) illustrates the design with key features highlighted.

Beyond the hardware developments the Dynamic Self-Repair Project has made contributions to the modular robotics field in both development of algorithms and in evaluating their relative effectiveness.

## 1.5 Key Contributions

The key contribution of this thesis is proof that self-repair during continuous motion is possible, no previous work has attempted to self-repair whilst another task is ongoing. Other key contributions include:

- The new “Quadruplet" data structure for describing modular robotic structures, see Sec.(6.2.4).
- A series of algorithms designed to enable physically feasible self-repair and self-assembly by processing these “Quadruplet" data structures, see see Sec.(6.2) and Sec.(7.4).
- Recognition of the hardware feature set required to enable self-repair during continuous motion.

Overall this thesis opens up a new field within modular robotics, the study of performing reconfiguration tasks within a modular robotic organism whilst external tasks in which the organism interacts with the environment are also in progress. This thesis also establishes axioms for this new field, a set of “6 laws of dynamic modular robotics", as described below:

- **Parallelisation:** Modular robotic tasks can be made faster and more effective by parallelising them such that different parts of the modular robotic structure can fulfil different parts of an assembly or repair task at the same time.
- **Human readable, physically inspired, data structures:** The use of the new quadruplet data structure provides an easy to interpret method of describing modular robotic organisms which can be read and written by humans and used directly by assembly and repair algorithms. Data structures and algorithms should naturally provide constraints comparable to those enforced by physics and geometry rather than being overly abstracted.
- **Not “just" a module:** Modules within a useful modular robotic system must be fairly capable independently and have reasonable capabilities for independent locomotion, sensing, communication and computation.
- **The embodiment principle:** “Where do I put the screwdriver", and other fabrication considerations, are questions only asked during hardware development but which have profound effects on module designs. Hardware constraints provide a way to limit the search space in which simulations can occur, and simulations can inform what features hardware would ideally have. Simulation and hardware development should therefore feed in to one-another.
- **Mixed global and local communications:** Modular systems can be more versatile when modules can both communicate with the swarm as a whole, or identified others within it, and via shorter ranged, directional methods. Local communication carries implicit information, beyond that supplied within the message’s contents, which can convey important environmental and positional information.
- **The hierarchy of challenges:** Operations such as self-repair which require repeated docking and undocking prove far more complex to implement in practice than operations

such as self-assembly which only require docking, abilities to perform these operations could be used as benchmarks within the field to assess system reliability.

Interesting accomplishments also achieved as part of this project include:

- Development of a new self-reconfigurable modular robot platform, Omni-Pi-tent, with genderless docking, omnidirectional motion and 3D reconfiguration abilities, see Chapter 3 and Chapter 5.
- A detailed example of production and verification of simulation assets, for use in V-REP, to accompany the hardware, see Chapter 4. This includes the novel, within the swarm and modular robotic fields, process of tightening the reality gap using test data from hardware.
- Demonstration of docking with modular robots without need for external infrastructure, a key challenge and relatively rare feat in the modular robotics field accomplished by only a few platforms, including CKbot [9] [166], SYMBRION [36] [204], Sambot[99] and Ubot [37] [38], and considered the “ability that enables all reconfigurable actions"[187], see Sec.(5.18), Sec.(6.5) and Sec.(6.6).
- Development and demonstration of docking between moving modular robots, an achievement not attempted with any previous modular robot platform and beyond such platforms’ capabilities, see Sec.(6.5) and Sec.(6.6).
- Providing proof that Dynamic Self-repair and other interesting multi-robot behaviours with modular robots can be achieved using a relatively sensor-poor platform [1].
- Development of a self-repair method able to change the location of the master unit within a modular robot structure, and modify the hierarchy across the organism, so as to perform self-repair in the optimal way, see Sec.(7.9).

## 1.6 Thesis Outline

The remainder of this thesis is arranged as follows, with each chapter’s objectives taking us closer to the proof of the hypothesis:

- **Chapter 2** examines previous work across the field of modular robotics, existing self-reconfigurable modular robotic hardware platforms are reviewed, with mobile platforms with comparatively capable individual actuation systems noted with particular detail. Real-world applications for modular robots are considered and the lack of existing robot

[1]Although sensor poor by standards of many robotic fields, Omni-Pi-tent has substantially more sensor capabilities than many previous modular platforms

designs with sets of capabilities representative of those needed in real-world applications found. Prior work on self-assembly with modular robots is considered, as is the, surprisingly small, body of literature examining existing concepts and implementations of self-repair. The potential usefulness of a self-repair strategy able to act while robots undertake other tasks, such as motion, is explained and the reasoning leading to the Dynamic Self-repair Hypothesis of Sec.(1.3) is introduced. This chapter concludes by highlighting deficiencies in the capabilities of existing modular robots which limit their ability to perform such a dynamic form of self-repair, and introduces the need for a new platform.
**Objectives:** Identify the difficulties impeding progress in the modular robotics field, Place the Dynamic Self-repair concept in the context of the wider field

- **Chapter 3** develops a list of requirements a modular robotic platform must meet to be able to perform self-repair activities which existing platforms cannot. These lead to the concept of the novel Omni-Pi-tent platform. The rationale behind its design is explained, in terms of meeting the desired requirements, and the early stages of development are discussed.
**Objectives:** Analyse the requirements for a modular robot capable of Dynamic Self-repair, Introduce the Omni-Pi-tent concept, Explain how the lessons of hardware development inform simulations and aid with embodiment

- **Chapter 4** briefly diverges from the previous chapter to discuss the use of simulation in robotics. The need for and use of simulation in the Omni-Pi-tent project are then discussed, details are given on some of the work undertaken whilst development of hardware and simulation proceeded as parallel strands. Notable amongst this is the use of real world experiments on partially developed sensor and actuator subsystems to provide data which was used to better inform the simulation environment and reduce the reality gap found between simulation and hardware.
**Objectives:** Establish the simulation infrastructure for the Dynamic Self-repair project, Discuss sensitivity and limitations of simulation models

- **Chapter 5** returns to where the third chapter left off. The practical implementation of the Omni-Pi-tent hardware is discussed and the subsystems examined in detail, and each related to how they enable novel capabilities for the platform. The software operating on the robots is examined, and compared to the software environment in simulations. The iteration of design features between the subsequent incarnations of prototypes is explained, the testing of certain sub-systems is described, and key performance parameters of the finalised module design briefly compared to other modular platforms.
**Objectives:** Explain the details of hardware development to provide aid and inspiration to future practitioners in the field

- **Chapter 6** relates the developed hardware capabilities back to the problem of self-repair, by considering the related task of self-assembly. The data structures used in this thesis for defining modular robotic organisms are explained, the algorithms for self-assembling free wandering robots into defined forms are examined. The implementation of multi-robot and Mergeable Nervous System [26] like capabilities are explained. This chapter then includes a series of experiments to compare novel dynamic forms of self-assembly to earlier methods. Verification of the underlying principles behind these are performed with limited hardware, followed by a more complete example with a larger number of hardware modules.
  **Objectives:** Develop new data structures and algorithms for describing, building and controlling multi-robot structures, Test the effects of continuous motion on docking and self-assembly operations, Compare Dynamic Self-assembly against existing strategies

- **Chapter 7** develops the self-assembly algorithms into controllers for self-repair. The details of this are discussed, as are the new multi-robot behaviours developed to enable them. Experiments are performed in simulation to compare the novel dynamic, continuous motion, self-repair strategy against implementations of static and structure breakup self-repair methods inspired by earlier work. Self-repair strategies for dealing with a failed master robot are developed, and tested as extensions of dynamic self-repair against the earlier form. Attempts at hardware demonstration discover some of the difficulties still to be overcome before it is possible to operate the new self-repair strategy on real robots. The effects of multiple failures are briefly considered and limitations of the strategy are identified.
  **Objectives:** Develop new algorithms for breaking apart and repairing multi-robot structures, Prove the possibility of Dynamic Self-repair, Compare Dynamic Self-repair against existing strategies, Test Dynamic Self-repair's ability to handle a wider range of scenarios

- **Chapter 8** concludes by summarising the chapters of this thesis. These summaries are related back to the original hypothesis from Sec.(1.3) and its corollaries in other chapters. Areas of future research developing upon this thesis are suggested, and contributions placed in the context of the wider modular robotics field.
  **Objectives:** Summarise chapters and lessons learnt, Link the chapter's results back to the state of the modular robotics field

# Chapter 2

# Background and Related Work

First proposed by Fukuda and Nakagawa in 1988 [32], and first tested in the form of the CEBOT system in 1990 [179], modular robots use small modules with general reconfiguration capabilities to form geometric structures which can reconfigure for the task in hand. While monolithic robots (large single purpose systems) may work well in ordered and predictable environments, modular systems, due to this reconfiguration ability, are more likely to succeed in varying and unknown environments [157]. This concept shares many parallels with the cells in living creatures. By grouping together they can form a vast array of different structures yet the underlying unit is the same. Fukuda and Nakagawa also stated that these robot “cells" will need to have some individual computational ability and need to be able to detach and combine automatically. As is the case for all tasks involving groups of robots [156] advantages include: distributed sensing and the potential for increased fault tolerance, while challenges include robots interfering with one-another, and communication among the robots. Modular robots may be homogeneous or heterogeneous [51] [157], the former using a single type of module and being more commonly developed [171], the latter using multiple module types working together. Modules are generally simple but being too simple can constrain functionality [157], the hardware challenge is often focused on how to ensure that the necessary linkages and functions can fit within the module. Except in some lattice systems [18], modules should be able to move independently as well as in groups, although in many chain systems, for example [137] [186] or [13], this independent motion is a form of wriggling which is relatively limited in speed, direction and accuracy.

| Robot | Architecture | Docking method |
|---|---|---|
| ATRON [18] | Lattice | Mechanical, gendered |
| CEBOT [9] | Chain | Magnetic |
| CKBots [9] | Chain | Magnetic |
| CONRO [186] | Chain | Mechanical, shape memory wire latch |
| HyMod [42] | Mobile hybrid | Mechanical, genderless |
| FireAnt3D [6] | Freeform lattice | Melted plastic |
| FreeBOT [75] | Freeform lattice | Magnetic |
| M-Blocks [117] | Lattice | Magnetic |
| ModRED [50] | Chain | Mechanical, genderless |
| Molecubes [10] | Lattice | Mechanical, genderless snap-in |
| M-Tran [13] | Hybrid | Mechanical, gendered or Magnetic |
| Odin [180] | Heterogeneous lattice | Mechanical, hand-assembled |
| Polybot [137] | Chain | Mechanical, shape memory wire latch |
| Roombots [118] | Lattice | Mechanical, gendered |
| Sambot [99] | Mobile chain | Mechanical, gendered |
| SMORES [35] | Mobile lattice | Magnetic |
| Soldercubes [107] | Lattice | Self-soldering |
| Superbot [94] | Hybrid | Mechanical, genderless with negotiation |
| Swarmanoid [14] | Mobile chain | Mechanical, gendered gripper |
| SYMBRION [36] | Mobile chain | Mechanical, bigendered |
| Trimobot [62] | Mobile chain | Mechanical, gendered |
| Ubot [37] | Hybrid | Mechanical, gendered |

**Table 2.1:** Table listing previous modular robotic platforms along with their architectures and forms of docking.

## 2.1 Modular Robot Platforms

Table.(2.1) notes many of the existing modular robotic platforms and their architectures. Chain, lattice and hybrid architectures [1] are all found; as are architectures with highly mobile individual modules. Docking methods are also highly varied, magnetic connectors, including in some cases electromagnets or electropermanent magnets, occur as do mechanical ones which are divided between gendered, bigendered and genderless systems [21]. Gendered docking allows connections only between differing types of docking port, one port involved typically works in a passive fashion and the other one actively holds it. Bigendered docking gives a very secure dock between two ports but prevents single-sided disconnection as each module involved is held in place subject to the other ones' actions. In effect the docking interface contains a passive section and an active section and that active section of each port

[1]Chain architectures focus on robots with joints and other reconfiguration elements able to reach a continuous range of angles and connect and disconnect from arbitrary initial positions, structures formed by chain modular robots tend to be snake-like or branching shapes [45]. Lattice architectures consider reconfigurations between discrete points in a lattice [33], actions only take place between robots in known relative positions, lattice modular robots often form dense shapes containing loops and blocks of connected robots. Hybrid architectures combine features of both chain and lattice robots [43], typically with the aim of forming lattice structures whilst also being able to start from arbitrary positions or construct jointed appendages.

holds the passive part of the other. In genderless docking systems active sections on each port hold against active sections on the other, ensuring single-sided disconnection is possible. Early modular robots [186] [137] often used the contraction of a shape memory alloy wire to actuate mechanical docking systems whereas DC motor power is found in many of the more modern mechanically connecting designs such as [36] or [42]. Unusual methods of docking also exist: Superbot [94] makes use of a connector [92] based on positioning hooks along rails where negotiation between the connecting modules beforehand selects the gender of each, Soldercubes [107] uses a low melting point Field's metal alloy to connect PCB pads on modules to one-another whilst FireAnt3D [6] melts plastic parts together to form bonds between arbitrary points. In some cases these more unusual methods of connection have also been applied beyond the field of self-reconfigurable modular robots to allow robots to adapt themselves by adding passive parts [200].

Many review papers [185] [202] [157] [203] on modular platforms focus on some of the earliest platforms, which were not particularly reconfigurable, and appear rather obsolete given the development of microcontrollers, batteries, mass produced electronic sensors and small cheap computing boards after those designs were developed. One review paper which does provide some interesting aspects of an overview is Liu et al.'s [45]. As well as cataloguing a wide variety of contemporary systems, old ones, and some systems which might not meet the usual definition of modular robots, they provide an intriguing measure by which to clarify the "autonomy level" of a modular platform. A platform is ranked on five scales relating to its: degrees of freedom, 2d or 3d workspace, ability to undock and dock without human aid, forms of independent mobility and locomotion modes. From this, a cobweb-like diagram can be plotted, see Fig.(2.1). The area enclosed in this plotted shape can be compared to that of a maximally capable robot which would score the maximum on all scales and produce a regular pentagon cobweb. Liu and Zhang consider modules which can fill a larger percentage of this pentagon to be more capable. SMORES [35] comes out as the best from the modules they apply this measure of capability to. The independent mobility and locomotion mode scales both provide counts, respectively, of how many modes of independent motion an individual module is capable of, and how many modes of locomotion a system of multiple modules can perform.

Tan et al. [125] provides another similar "cobweb" framework for evaluating modular robots, referred to as an "Autonomy Index". This framework considers a wider variety of parameters than Liu and Zhang's [45] but places perhaps overly great weighting on parameters which will be influenced more by the controller software running on modules than by the platform's own capabilities, with planning and decision making being some of those software related axes of evaluation.

In the interest of providing a better overview of the hardware involved in modular robotics a number of specific modular platform designs are described in more detail. These platforms

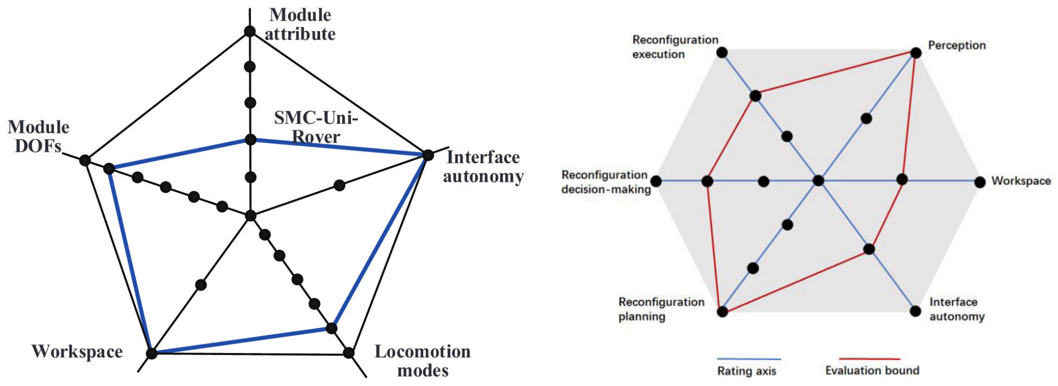


**Fig. 2.1:** A comparison of Liu et al.'s 5 scale "cobweb" figure and Tan et al.'s 6 scale scheme. On Liu's graph a plot is shown for an example robot, the SMC-Uni-Rover [46]. Liu et al's workspace scale determines whether a collection of the specific type of modular robots operates in 2 or 3 dimensions. Interface autonomy handles whether modules can dock and undock without human help, scoring 2, only undock and score 1, or can only maintain a fixed morphology without help and score 0. Degrees of freedom per module are counted by the Module DoFs scale. The Module attribute and Locomotion modes scales relate to how many roles a module can take and how many ways a group of docked modules can move together. Tan et al.'s version is also influenced by the robot's perception, sensory, capabilities and by considerations of how autonomous reconfigurations are. Images from the papers [45] and [125] respectively.

were chosen as a mixture of good representatives of their architecture (for example: ATRON for lattice, ModRED for chain, SYMBRION for mobile) and platforms with particularly interesting capabilities in their modules, hence earlier modular robotic platforms are not discussed despite them being popular in review papers.

### 2.1.1 ATRON

ATRON [18] is a lattice modular robot platform, units contain a continuously rotatable joint between two hemispheres, each hemisphere includes active and passive gendered connection points. The modules have one degree of freedom, the relative rotation of the hemispheres, but are nonetheless able to make impressive structures in 3D, see Fig.(2.2) for a view of one module and of a multi-module structure. This single degree of freedom was a deliberate choice to make the mechanics of a module as simple as possible and was done at the expense [18] of ease of control. A series of rules were calculated for the ways in which modules could attach to one-another based on lattice structures in crystals, due to having only one degree of freedom per module a face-centred-cubic crystal-like lattice was chosen which lead to much less intuitive lattice rearrangements than a cubic lattice would offer. Careful considerations were also necessary to avoid situations where the gendered forms of connector would lead to the need for two populations of robots mixed with one another, having female bars at $90^{\circ}$ to male hooks was in the end chosen to avoid issues of that kind. Individual modules cannot move across the ground alone but small groups can by using tyres placed around the equator

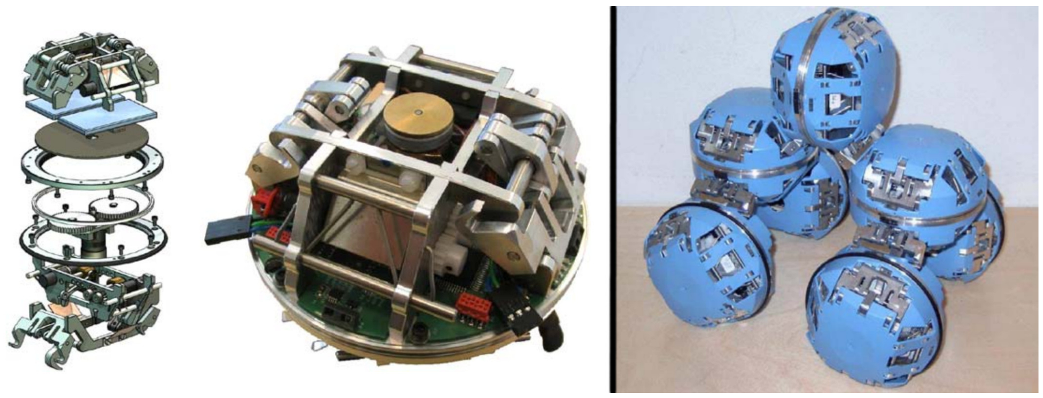


**Fig. 2.2:** The ATRON modular robot design. An exploded view (left) of the components stacked together, is shown beside a photograph of a production module with the outer shell removed (centre). Note the docking hooks, shown extended in the bottom hemisphere of the exploded module and retracted on its top hemisphere and on the photograph. $90^{\circ}$ round from the hooks, bars are visible which other modules can hook onto. One hemisphere has a rubber tyre extending slightly outward from the equator allowing the module to act as a wheel for multi-robot structures. Also shown is a group of ATRON modules in a “car" configuration (right). Single module images from the paper [18], with multi-module image from [119].

of some hemispheres as wheels. The modules contain a processor in each hemisphere to reduce the number of connections necessary across the slip-ring between the hemispheres and communicate locally via infrared, modules always know the location of the neighbour which is messaging them. ATRON appears to be the most capable design amongst those of a pure lattice architecture, but in the paper Ostergaard and Kassow comment that “future module designs should have capabilities to self-align and connect without relying on a lattice for positioning" [18]. This was partly addressed in later work [193] with the ATRON platform but without hardware changes which could have allowed single modules to dock independently.

### 2.1.2 M-Tran

M-Tran [13] modular robots are constructed around a central link with two ends which can dock both lengthways and sideways. The designers describe it as a double-cube, quite different from the space-filling polyhedra which many modular robot modules are designed around, they consider this useful as it allows groups of M-Tran units to dock together with less proportion of empty space between them than space-filling polyhedra designs usually give. Hence M-Tran offers greater mechanical strength to multi-robot structures than some other platforms do. Each “cube" can rotate by $\pm 90^{\circ}$ about its axis, see Fig.(2.3). One end of the robot has male connectors on three faces of the active “cube", the passive cube on the other end of the link has three female connectors. In magnetic versions of M-Tran the active box carried magnets and equipment to detach docked modules while the passive box had only magnets. Self-reconfiguration and group motion has been shown with M-Tran, during reconfiguration the “cubes" only use $90^{\circ}$ multiples for their orientations but continuous angles

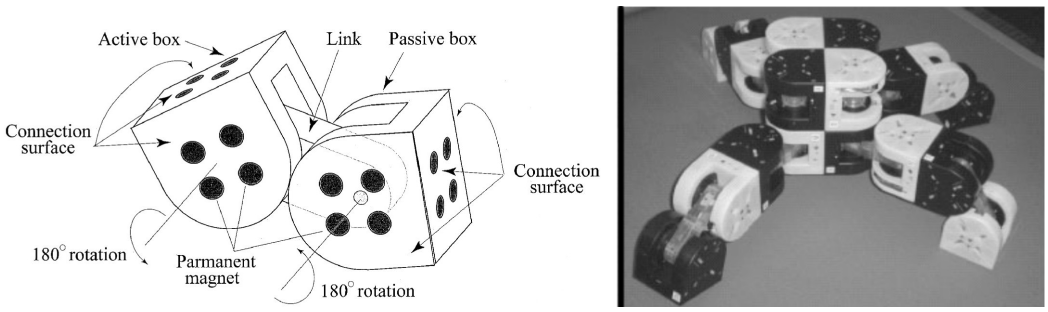


**Fig. 2.3:** A diagram of a single M-Tran module labelling the key components and the rotation axes of the actuators (left) is shown alongside a multi-module M-Tran structure. Images from the papers [13] and [16].

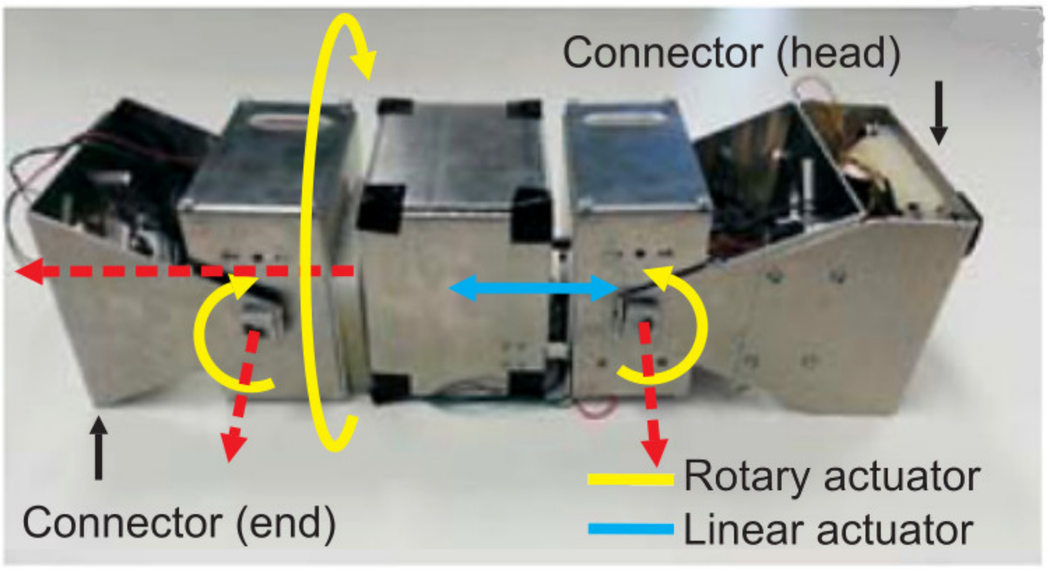


**Fig. 2.4:** A ModRED robot with the degrees of freedom marked, the ends are fitted with RoGenSid connectors, not visible from this angle. Image from the paper [50].

are used when walking as a group. Individual modules alone can move in a straight line by flexing their joints but any other motions such as turning to change direction require multiple modules docked together. M-Tran has been through three generations [16] of design iterations varying the sensors, the initial generation lacked them, and docking mechanism, some use magnets while the latest generation use gendered hooks.

As single modules have limited abilities to move, except by inchworm type wriggling, it is often very slow to perform reconfiguration with M-Tran.

### 2.1.3 ModRED

ModRED [50] is a chain type modular robot designed as a prototype for planetary exploration tasks. The designers state in their paper that modular robots which have more degrees of freedom per module generally have better dexterity and autonomy, hence it has four degrees of freedom per module, a pitch joint, a yaw joint, a roll joint and a length extension ability. See Fig.(2.4) for a labelled image. Constructed from metal the modules are heavy, any joint on one module can only lift about 40 % of a module's weight, as the design is intended for eventual use on other planets where the gravitational acceleration is weaker this is considered acceptable [50], but does limit the platform's usefulness as a laboratory testbed. Two docking ports are fitted to a module in the form of a rotary-plate, single-sided, genderless docking port known as RoGenSid [73], this takes inspiration from earlier connector designs but acts faster and is simpler, using, in principle, just one turning part. If a module fails another module will always be able to rotate its own plate and unhook from the failed module. Similarly, if a failed module needed to be carried somewhere, a working module would still be able to connect to it. The spikes, referred to in the paper as pegs, positioned outside the plate's circular area maintain the docking when the roll degree of freedom is used. The ModRED units use a pair of Arduino boards for computation, communications are done with long-range XBee wireless modems, sensors include infrared proximity sensors, bump switches and a combined accelerometer/gyroscope/magnetometer. A variety of single and double module motion gaits have been demonstrated.

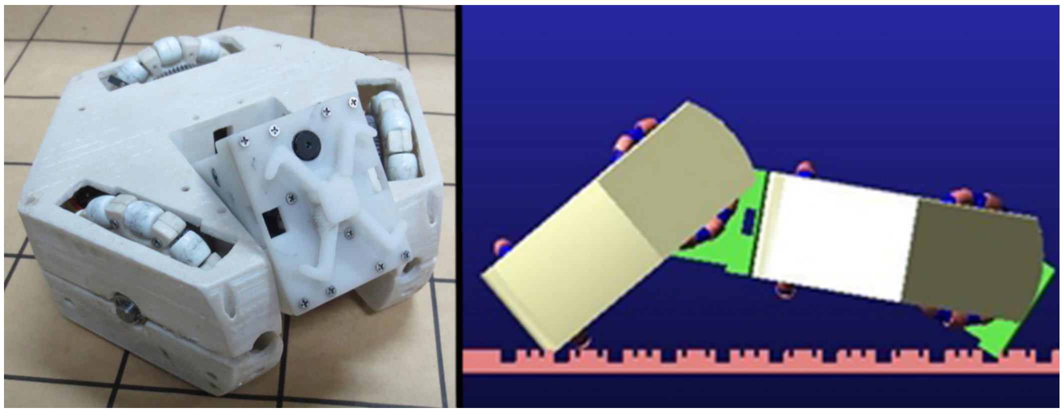


**Fig. 2.5:** A Trimobot robot module (left), note the 3 omniwheels and the gendered active docking hook at the front. A simulated pair of modules forming a chain and performing inchworm locomotion is also shown (right). Images from the paper [62].

### 2.1.4 Trimobot

Trimobot [62], see Fig.(2.5), is one of very few omnidirectional modular platforms, shaped as a hexagonal block with 3D printed omniwheels on three of the sides it can move across ground in all directions and rotate on the spot. Experiments showed these to be a very effective drive system. When following a straight line, it experienced only a 1.25% deflection [62]. The design is up-down symmetric, so can operate when flipped upside-down. Also, it can join to other units to perform inchworm-like group locomotion. The design appears to be able to form some chain morphologies, but would not be able to manage lattice forms and may struggle with branching morphologies. Also, the hinge is positioned such that rotation axis is off-centre, this means any 3D structures formed would not be able to have convenient (for reconfiguration) symmetry. An ATMEGA128A microcontroller provides computing power, but lacks the versatility that a more powerful device running a full embedded operating system could bring. Some of the sensor calculations had to be done remotely on a computer at the end of a Wi-Fi link with results transmitted back to the robot. The docking mechanism is gendered, each module has one male and five female connectors, so modules cannot escape the clutches of their failed companions if the failed companion has taken the active role in docking. This platform therefore provides an indication of the benefits of versatile omni-directional locomotion, but is let down by the limitations of gendered docking and off-centre hinge actuation which reduce the range of tasks it can perform once docked.

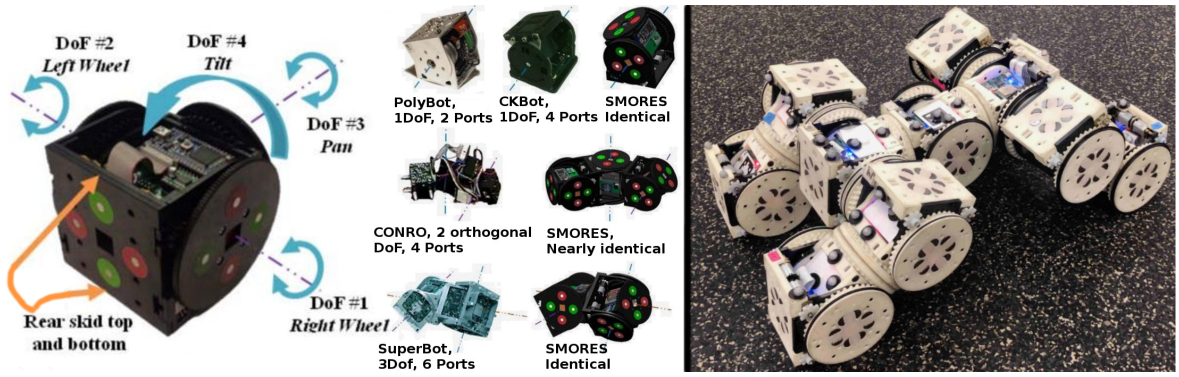


**Fig. 2.6:** A SMORES robot with the degrees of freedom marked (left), is shown alongside a comparison to other modular platforms (centre) showing how one or a small number of SMORES modules can represent the geometry of those designs and copy their degrees of freedom. Also shown is a mobile multi-robot structure using the slightly later SMORES-EP module design. Images from the papers [35] and [5].

### 2.1.5 SMORES

SMORES [35] was designed to be a platform able to emulate most other modular robots, either acting alone or in larger groups it can replicate the geometries, docking port positions and degrees of freedom of most other types of module, see Fig.(2.6). The designers say that the geometric shape and docking port positions on robot modules should enable the largest range of motions and configurations for the number of actuators, SMORES excels in this respect. Overall the module has 4 degrees of rotational freedom, not including its ability to drive across the ground, these are the rotation of two side docking ports and the hinging and rotation, tilt and pan, of the front docking port. The rear docking port cannot be turned or otherwise moved. The motion of the hinge is $\pm 90^\circ$ and the docking ports are all capable of continuous rotation, these rotations are measured to $\pm 5^\circ$ by a number of continuous rotation potentiometers. A very capably designed gearing system enables two motors to control both the lifting of the hinge and the turning of the front docking port depending on whether they run in the same or opposite directions. This provides SMORES modules with sufficient torque to lift chains of 2 modules [35]. The designers consider that docking actuators should only consume energy when changing between docked and undocked states, this leaves mechanical and magnetic options but forbids electromagnets. On SMORES the docking and undocking is magnetic, a docking key is used to detach modules by pushing them away and to give shear strength to the magnetic connection.This magnetic docking is limited in terms of disconnection when one of the units has failed, if a failed unit is docked to the back port of a surviving robot the survivor cannot free itself as the docking key only acts on the front and side ports. Later work fitted SMORES modules with EP-Face connectors [103] where electropermanent magnets allowed switching between states without long term power consumption when actuated, with the docking key omitted in this design “upgrade" SMORES modules with the EP-Face are unable to perform single-sided disconnection for any port on a module. SMORES units use two wheels to skid steer, this means that when docked together in groups it will not always be possible for them to efficiently drive together.

SMORES modules are lacking in sensor capabilities, having only the potentiometer-based hinge angle sensors for proprioceptive purposes [91]. External sensors are required for any reconfiguration involving detached modules joining to form a larger structure, a specialised tool module in the form of a camera on a tower [101] has been developed but this module does not conform to the size convention of regular SMORES modules and has only passive docking, which limits the ability of other modules to reconfigure when this tool module is required within a morphology.

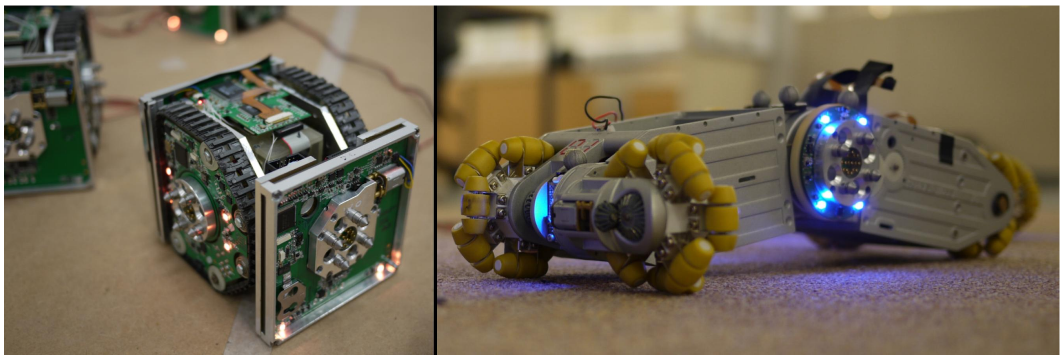


**Fig. 2.7:** SYMBRION modules. A SYMBRION scout unit (left), note both the tracks and the very narrow pins of the gendered docking connectors, and an active-wheel module (right), note the mecanum wheels and the central hinge which allowed the docking port to lift up or down. The backbone modules, not shown, were similar to the scout modules. Image from Murray’s thesis [51].

### 2.1.6 SYMBRION

SYMBRION [146], see Fig.(2.7), was part of a project to develop bio-inspired multi-robot organisms [147] with heterogeneous mobile modules. The plan was for modules of 3 types: scout robots, cubic tracked robots with active docking ports ahead and behind and passive ports to their sides [189]; backbone units, screw-driven modules of the same size as scout robots, but with more of their internal space dedicated to battery and hinge subsystems [83]; and active-wheels, omni-wheeled robots not compatible with the cubic lattice of scout or backbone units but able to lift scout and backbone robots to transfer them between locations [63]. SYMBRION also aimed to make modules which were independent robots in their own right [116] as well as modules within larger structures. Fitted with relatively small docking equipment, SYMBRION units struggled to dock to such an extent that they had to be arranged by hand in many cases, for example in Murray's experiments [51]. One of the major problems with the concept of this system, beyond some serious implementation issues reportedly involving metal mechanical structures becoming electrically live and shorting the modules' power supplies, was that the scout units [189], which made up the bulk of the heterogeneous modules, were only able to move backwards and forwards. This greatly limited the ability of docked formations to move and would have prevented a formation solely using scout units from moving except for a very small number of formations where all units were aligned. It is notable that self-repair work with SYMBRION modules was forced to let formations break into the one largest piece and then the rest had to shatter back to individual units [51] before self-repair could occur, this was a consequence of SYMBRION units struggling to move together. The system was heterogeneous so the active-wheel and backbone robots should have been able to help here. However these had major problems with their mecanum wheel and screw-drive propulsion systems as well as with matching their heights to dock with scout units. Despite their flaws the SYMBRION units carried an impressive array of sensors, rare among modular robots, this can perhaps be ascribed to the project's philosophy of "[not] 'just' modules" [116].

### 2.1.7 HyMod

HyMod [42] is a modular robot platform designed as a hybrid between both chain and lattice architectures. This modular platform was primarily developed as a way to showcase the novel HiGen [90] genderless and retractable docking connector, Fig.(2.8) shows the design. The modules are both individually mobile, and able to operate from arbitrary starting positions as chain reconfiguration can, whilst also having significant capabilities for reconfiguration within a cubic lattice structure. The modules fit within spherical volume and have the unique ability of being able to rotate in place within the lattice structure due to retractable docking ports. The design, however, has limitations [21]. The sensor capabilities are extremely limited, having little except proximity sensing capabilities and an inertial measurement unit, which means that, despite impressive abilities once in a lattice, modules cannot in practice dock autonomously unless some external infrastructure is provided to guide them. The wheels provide only differential drive, non-holonomic, motion capabilities, meaning that, unless lattice transforms are later used, a wide variety of structures cannot be formed. Differential drive wheels also limit the manoeuvrability of multi-module structures. Due to the complications of fitting both batteries and wiring within the tight spherical volume, battery life is short, < 30 minutes [21], limiting the length of experiments which can be performed. Only two HyMod modules were ever constructed [21], hence multi-module experiments, beyond verifying the mechanical capabilities, have not been performed.

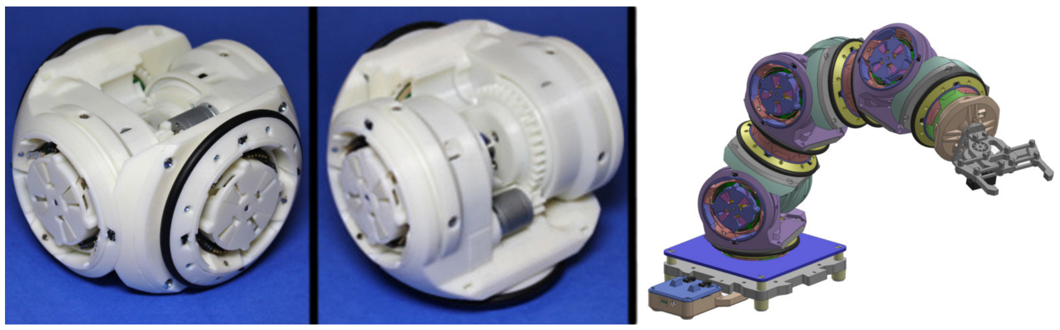


**Fig. 2.8:** The HyMod modular robot. Photographs of a single module in driving position (left) and with the hinge folded (centre) alongside simulated images of a multi-module arm (right). Images from [42] and [21].

To summarise this section, while not the case for all modular platforms, it becomes apparent from reviewing that amongst those with the more capable mechanical designs, and hence better representations of the actuation capabilities expected in an industrial modular robotic system, there is a considerable lack of sensor capability. The fact that review papers on modular robotics, for example [43] [125] and [45], rarely make explicit comparison of modular platforms' sensor capabilities, shows how rarely attention is given to this aspect of design. This lack of comparison of sensor capabilities also shows how the modular robotics field has not yet evolved to the stage at which easy benchmarks exist to allow sensor comparisons to be made between platforms. In many cases this lack of sensors is severe enough that autonomous docking has not been attempted, atleast not without reliance on external tracking cameras which would not be readily available in harsh environments. Future platforms would benefit from improving sensing capabilities, atleast to the point where they no longer undermine the platforms' abilities to dock and make use of their impressive actuation subsystems.

Appendix(B) includes a brief summary of the features of the robot platforms discussed in detail above.

## 2.2 Towards Field-Ready Modular Robots

Modular robotic research has as yet not moved beyond the confines of the laboratory [96], many platforms have exposed wiring [186] [137] [36], or even rely on tethers for power [43], none so far publicised are waterproofed, dirt-proofed, easily maintainable or user-friendly. These laboratory-grade robots have often focused on the development of one capability or feature without great consideration for others, all too often they have been, in-effect, purely mechanical prototypes with little sensing or computation capabilities and remotely controlled from a centralised computer [101]. Researchers have previously suggested that a lack of “killer applications" has held back the field [11][33], with more applications now suggested, one can still plausibly argue that the lack of use of modular robots in the applications for which they are expected to be helpful causes a lack of incentive to develop more field-ready systems, a potential “chicken and egg" situation delaying modular robotics research. The situation may also be worsened by a lack of inter-platform comparison studies. The very few papers which do try to use multiple platforms, such as [106] or [205], even then only try multiple platforms as severely simplified kinematic simulations and do not try much comparison between the platform’s actual sensing and actuation capabilities. The difficulties which this lack of inter-platform comparisons cause for the modular robotics field may be addressable by standard benchmark tests which different platforms could attempt, yet devising such benchmark tests may prove particularly difficult until real-world applications for modular robotic systems exist from which inspiration can be drawn.

Modular robotic research usually suggests modular robots would be useful in up to three types of scenario: infrastructure monitoring[30], planetary exploration [50] and post-disaster urban search and rescue[51]. One or more of these applications are usually suggested within modular robotics papers. However, there are very few papers focused entirely on modular robotics applications. Those papers which are, Jahanshahi et al. on infrastructure monitoring [30] and Yim on urban search and rescue [190], provide a fascinating insight into which specific features of modular robots could prove critical in real-world applications.

### 2.2.1 Modular Robots for Infrastructure Monitoring

Jahanshahi et al.’s paper [30] considers the application of modular robots to structural health monitoring. The gradual shift from maintenance being schedule based to condition based allows resources to be better targeted at those pieces of infrastructure which need attention, but requires much more extensive monitoring. Permanently fixed sensors have many limitations, for example the need for long-term power supplies, but also an inability to move from where they were sited to positions of interest, such as close to growing surface cracks to determine whether they are structurally dangerous or merely cosmetic. Jahanshahi makes a very strong case for robots which can split into multiple units to provide sensor coverage of tasks from different locations than the tasks are being conducted, perhaps measuring vibrations in a beam while a cable some distance away is re-tensioned, and points to how self-reconfigurable modular robots are unique in providing this capability while being able to reach and leave locations as whole structures better suited for mobility. They also provide suggestions as to why, so long as trickier reconfiguration processes can be overcome, chain, or hybrid modules with chain capabilities, are advantageous over pure lattice architectures.

Reasons include the inefficiency of purely lattice based architectures for motion and the requirement that in lattices many modules must collaborate for even simple motions[30]. Connectors are discussed and criticisms made of mechanisms which could either come loose or remain stuck if one end fails, the authors advise on the need for genderless docking when few solutions for it had been developed [92]. Having at least two, preferably more, such connectors per-module is suggested, as is the ability to share power and data across docked connections. Distributed, non-centralised, control is also recommended, along with ensuring that modules which somehow end up docked in an unknown state can discover the topology of the structure around them. Papers on a variety of conventional, specialised, robots for infrastructure inspection purposes [17] [140] [141] [142] focus on the usefulness of features such as adjustable camera and lighting mounts, as well as multi-DoF arms and task-specific tools. Careful swinging actions with counterweights [144] have been considered for robots crossing between power transmission lines when inspecting them, modular robots could find much less convoluted ways to handle these physically tricky operations. There is also an interest in using the robot's sensors for self-inspection work [17] to check the robot for damage to itself. Modular robots will naturally provide this capability if fitted with appropriate sensors. Limitations of highly specialised robots are also noted [140] and given as a reason why tele-operation has been favoured over autonomy, modular robots can overcome the problems of over-specialisation. The desirability of high dexterity systems to enter piping are also raised [141][140], a modular robot designer could apply solutions here by ensuring modules contain joints for reconfiguration which provide multiple degrees of freedom per module. Analyses of present robotic systems [142] find that increases in redundancy of sensors and actuators make full system failures less likely, modular robots provide this redundancy in abundance.

### 2.2.2 Modular Robots for Urban Search and Rescue

Yim et al.'s paper [190] considers modular robots for penetrating rubble to locate survivors under collapsed buildings, modularity is considered useful because of the versatility it offers for passing through "cracks, holes and pipes" [190] and then the possibilities for using modules to shore-up cavities within the rubble to make them stable enough for further activity to be conducted, and finds reconfigurability especially useful as the form of collapsed rubble is not known in advance so cannot drive specialised robot designs. Yim also considers the replaceability of modules a desirable characteristic. The importance of onboard computing capability to ensure that the computing power needed for planning and control of multiple modules scales up, albeit linearly, as the challenges rise, perhaps exponentially, with the number of modules present. Yim recommends that modules should have sufficient actuators to be able to perform distributed manipulation of objects, such as repurposing a rolling-track [138] locomotion gait as a conveyor belt for debris removal. They explain that with a sufficiently effective modular robotic system, the main challenge becomes that of programming the right actions rather than developing the right hardware. However this can only be applicable once hardware is developed to a point where modules are more capable than some lower bound which would be determined by application considerations. For example, in terms of ensuring that mechanical systems are sufficiently compact not to protrude and interfere with reconfiguration. The paper fails to recognise the possibilities for genderless docking to allow single-sided disconnection of failed modules, and suggests, as if inevitable, that multiple modules must be sacrificed to remove failed units. Papers on conventional, hence specialised, designs for Urban Search and Rescue robots [8] imagine a need for separate systems for “peeking" tasks (rapid viewing of inaccessible or dangerous locations), collapsed structure surveying, non-collapsed structure surveying, wall climbing and confined space retrieval, a modular system could perform all those roles, possibly as single modules for some of the tasks. As well as reducing costs the reconfigurability of modular robots could ensure that the correct type of robot is available when and where it is needed, rather than emergency responders having to maintain whole fleets of robots only some of which will ever be of the right sort, in the right place, at the right time, to be useful. It should be noted that, while drones and other flying robotic systems heavily occupy the public consciousness, they cannot be a plausible replacement for all of those ground robot roles [143] in the way that modular robots can, primarily due to the difficulties that UAVs, machines designed to be as lightweight as possible, are sure to have in exerting forces to manipulate objects (for example debris or survivors).

### 2.2.3 Hardware Capabilities for Field Readiness

It would therefore appear that much of the work in modular robotics has been undermined by the fact that the modular robots widely used within it tend to each lack certain capabilities and therefore are not representative of the kind of modular robots we would hope to see in real-world use. Making field-testing a more integral part of modular robotics research has been recommended by some researchers [96]. I, for one, am not criticising the fact that prototype robots are not ruggedised to industrial or disaster-zone [191] specifications, that is entirely understandable given time and resource constraints on the projects, but rather the fact that, for the most part, none have been developed which attempt to represent each key capability of an industrial system, at least to some degree. Combining what can be concluded from [30] and [190] then one would expect an industrial system for any of the usually suggested (infrastructure monitoring, disaster rescue or planetary exploration) applications to require: full onboard sensory capabilities for docking and self-reconfiguration operations, significant levels of mobility for individual modules, a convenient, probably cubic, lattice structure when docked, some level of 3D reconfiguration abilities and a robust docking interface not likely to either become trapped or fall apart immediately if either robot involved in a connection fails.

With that in mind this thesis now moves to review the state-of-the-art in multi-module research, focused particularly on self-repair. This starts, however, with a discussion of the means by which multi-module structures are initially formed: self-assembly.

## 2.3 Self-assembly with Modular Robots

Prior work on self-assembly in modular robotic systems has covered a number of different approaches, self-assembly can occur with lattice modular robots [193], where modules move along well defined discrete transforms between points in a crystal-like lattice structure, or with chain [187], hybrid [5] and mobile [27] modular robots. While in the former scenario pre-planned precision actuator movements can handle most of the motions necessary, self-assembly in continuous space scenarios typically requires some form of sensor feedback for guidance purposes [30]. Self-assembly can also be divided in to systems [7] which use modules which are self-propelled along deliberate paths to perform assembly, and those which rely on stochastic means [185] where modules are randomly agitated by, for example, a shaking surface where many collisions occur but docking only occurs during collisions which “grow" the structure out at desired points. It is self-assembly using self-mobile, not stochastic, modules, from scattered, not lattice-bound initial positions, which will be considered here as it is more relevant to the unstructured real-world scenarios in which modular robots would be expected to find uses, see Sec.(1.2). This review focuses on self-assembly strategies designed for practical use on realistic modular robots, rather than on papers, such as [155], which contemplate self-assembly as a solely mathematical problem involving thousands of units and unusual constraints on lattice position transforms.

### 2.3.1 Early Work on Modular Robotic Docking

Rubenstein et al. [187] provided the first proof of autonomous docking between separated modular robots, they distinguish their work as inter-robot docking, between separated groups of modules, as opposed to intra-robot docking, docking operations related to robots moving between locations in a lattice along precisely known discrete transforms. Previous work [195] on autonomous inter-robot docking had only considered a docking operation between two heterogeneous robots designed solely for docking experiments and running different behavioural controllers on each robot of the system. Rubenstein [187] focused on connecting two multi-module robot snakes head-to-head and the work consisted largely of considerations on how to sense misalignment and correct it. Even today, leaders in the modular robotics field still consider docking to be a “usually difficult task" [5]. At this time no consideration was given to how larger more complex structures could be defined in a “language" which could be interpreted by robots to guide the formation of arbitrary and complex structures.

Stoy [193] provided another of the early practical examples of self-assembly using the ATRON modular platform, in this scenario two groups of 3 modules came together to form a 6 module group, as ATRON units are not individually mobile starting from single modules would not have been possible. This provided one of the earlier examples of multi-module behaviours being used as part of self-assembly, with the 3 module clusters each consisting of a “head" robot which gave commands to the other two which acted as “wheels". The work mainly focused on demonstrating how morphologies could be exploited to ease docking if the shapes were such that, upon pushing against one-another, they would tend to guide the docking ports together. This work did not include any convenient description of the structure to be formed and would likely have needed major rewriting of all controller code to adapt to producing different structures; it was only applicable to structures which could be formed from substructures with the convenient property of being able to self-align when pushed together.

This early work largely lacked consideration of how self-assembly procedures could be designed to autonomously find which operations individual robots would need to complete so as to assemble an arbitrary defined structure.

### 2.3.2 Gradient Self-assembly of Shapes

Other work from Rubenstein et al. [52] developed a self-assembly method for Kilobots, large numbers of extremely physically limited swarm robots. The first four robots define a seed location and orientation after which they define a gradient spreading out from the seed. Further robots are then attracted and use edge following around the partially completed structure to reach the sites where they are required. These robots stop and hold position at the moment they are about to exit the boundary which defines the desired shape. The robots here were not modular, no docking connections were formed between modules, so the completed structure was not able to act as a cohesive whole. This use of a non-modular platform also meant that the defined structure to form was simply a shape on the ground filling a certain boundary region, not a list of docked connections between ports, hence the shapes formed were not always reliably the shape defined. Similar area based self-assembly and self-reconfiguration methods have been developed in other contexts, for example [53] [54] [58] or [59]. These inaccuracies and variabilities mean that such a self-assembly method cannot be useful for smaller numbers of more complex robots needing to form precise chain or branching structures focused around hardware features such as joints.

### 2.3.3 Self-reassembly with Module Clusters

Yim’s Self-reassembly After Explosion work [166] considers how to assemble a modular robotic structure again after a violent force has separated it into discrete modules and scattered them randomly, it is therefore a form of self-repair as well as self-assembly. The goal is that a modular robot separated in to constituent modules can, once it has assembled again, resume a task it was performing, pre-explosion. CKbot modules were used which are not individually mobile, the structures being “exploded" were therefore designed with weak connections to fail between pre-defined clusters of modules, each cluster being shaped so it was able to propel itself. The work used a highly planned method of assembly, with robots scanning the environment around them to recognise others, then finding the shortest routes to come together again. It appears that centralised planning is used to decide which cluster will go where, especially for optimisation of the reassembly trajectories, but module clusters handle obstacle avoidance and docking autonomously. Minimal details of the algorithms underlying the self-assembly method are given. However they clearly are able to handle structures defined by port connections, and can allow different robots to take specific roles within the structure depending on how the assembly is most easily accomplished.

### 2.3.4 Languages for Defining Structures

Christensen’s SWARMORPH [60] script allows distributed formation of a structure according to defined rules. For SWARMORPH, by controlling the colours of a ring of LEDs around their periphery s-bot robots are able to recruit randomly wandering units into 8 approximate locations around their circumference to which an attracted robot can connect a gripper. LED colours also acted as a very slow communication method to pass information in the script to other robots. Defining structures must be considered in terms of rules to be obeyed by each robot in the forming structure rather than a list of connections, and as each robot has only one active docking gripper there are limits on the possible morphologies to be formed. The gripper and ring method of docking makes the formed structures relatively non-rigid, limiting their ability in forming load bearing or force applying structures of the kinds an industrial system would need to cross obstacles or interact with environmental objects. It was mentioned in [60] that SWARMORPH could be used as an underlying practical architecture to control gradient based or hormonally inspired assembly methods. Related work with the same s-bot platform [2] has shown how self-assembly can be triggered by environmental circumstances. SWARMORPH has also been adapted to use an overhead drone to guide the reconfiguration [3] [159], this supervised morphogenesis provides a robotic swarm with the ability to reconfigure into forms informed by the long range views of a flying robot, however also makes them more dependent on external infrastructure.

Baca et al. [15] does not address the task of self-assembly but provides some interesting insights into graph theory based representations of multi-module configurations, unlike SWARMORPH these work with explicitly defined connection lists for the modular structures rather than defined rules from which a structure emerges. Structures are considered as a graph formed from nodes, representing the modules, and edges representing docked connections. All elements of a modular robotic morphology are, under this data structure, described in terms of edges or nodes. A matrix representation allows these graphs to be written in a form which can be processed within robot controller’s algorithms; this is known as an assembly incidence matrix. Each entry $a_{ij}$ describes the link between a particular connection within the structure, the $j$th connection, and a particular module, the $i$th module in effect representing the state of a given docking port. A zero value implies no connection, other values can show which docking port on a module is used for a given connection. They find that the properties of a matrix give useful clues about the properties of a structure, matrices dominated by zeroes give highly flexible shapes with many DoF. Whilst a useful representation, and particularly helpful when considering complex 3D transformations of structures, there are other methods also available for defining multi-robot structural configurations on a port-by-port basis.

Liu and Winfield [27] developed a morphogenesis controller for the SYMBRION platform. The self-assembly method worked by allowing robots within the structure to, where required by a data structure representing the desired morphology, recruit others to them. Information from a seed robot which first began the recruitment was used to tell its recruits what recruitment actions they in turn should perform. The controller in question had limitations due to hardware difficulties and design choices which SYMBRION suffered from, for example robots only ever used their “front" port to dock to a recruiting port because SYMBRION Scout robots could not perform sideways movement. Also relatively short beacon sensor ranges meant robots often moved blindly past each other just out of reach so took considerable wandering time before making structures. The lack of sideways movement available in SYMBRION robots meant that the act of docking became difficult where robots needed to steer when they wanted to move across a beacon to improve alignment then turn again to try to drive towards the beacon, often missing and having to retreat and retry. Liu and Winfield first used a Single Entry Recruitment method which required the swarm to share both an array defining the structure and another array defining the order of recruitment. This was quite a slow strategy to form structures as many of the close passes of robots near to ports which could have led to a docking action did not because a different specific port was recruiting at the time. A Multiple Entry Recruitment method in which all ports on the organism which still required extra modules performed recruitment at once was also tried and found to be faster, especially when plentiful spare modules were available in the environment. A version of this strategy went on to be adapted by Murray et al.’s self-repair work [51] and provides a conceptually convenient starting point for the new self-assembly strategies developed later in this thesis.

Wei et al.’s work with the Sambot platform [99] [192], a two-wheeled differential drive modular robot, worked on achieving autonomous docking between modules, and on the self-assembly of multi-module structures. Behaviour based controllers run on each module which may be in one of three states, seed, docking or connected. Multi-robot structures are defined as a table, derived from a node and edge graph, in which each robot gets a list reflecting whether each docking port on it is to act in active mode, passive mode, or not be connected to anything at all. The seed robot starts with this connections table and recruits robots to dock to it, they switch in to the docking state as they detect recruitment signals, then in to a connected state once attached. The relevant part of the connections table is transferred to the newly attached robot which recruits others to itself. This proceeds one docking connection at a time until docked robots report back to the seed that they have recruited the expected number of modules to them, at which point the seed stops the assembly procedure. This work was limited to some extent by the hardware used, whilst Sambot appears a fairly capable platform, it has gendered docking with only a single active docking port on each module, the other ports being passive ones. As with Trimobot [62] this type of arrangement of ports means only certain types of chain structures can be formed. A connection table only representing what state a robot's ports are in rather than explicitly describing what connections are made, can be sufficient for robots with a single active gendered docking port but would not be applicable to more advanced hardware capable of assembling more complex morphologies. This work did however benefit from Sambot being, it would appear, designed with a "not 'just' a module" philosophy similar to SYMBRION's and therefore containing a worthwhile suite of sensors. Having onboard sensing of this form ensured the work remained focused on what modules could actually achieve, and exactly how this would be best implemented, rather than some of the more speculative work seen elsewhere.

### 2.3.5 Recent Self-assembly Developments

Work by Mathews et al. on Mergeable Nervous Systems [26] used Swarmanoid Footbots to follow on from Murray's work and, inspired by the way in which biological nervous systems are morphology dependent, builds upon it to create organisms which can display "sensorimotor co-ordination equivalent to that observed in monolithic robots" [26]. Morphology is stored as a string describing which robots are connected to which. One robot acts as a brain and master, sensor data is received by body robots and fed as messages towards the brain, at each robot they pass through they are combined together so that no individual robot, including the brain, ever has to deal with inputs from more than the number of robots which can dock around its perimeter. Actuator commands from the brain unit are fed to other units, each of which acts locally in its own reference frame in the appropriate way to produce its part of the group motion ordered by the brain. As they combine, all robots in the sections of the structure which will not contain the brain of the completed structure have their internal representation of the structure updated to account for the new layout. No time for self-discovery is needed, each module understands its location and the form of the structure around it as soon as a connection occurs. The brain unit is then updated by its children as to the state of the newly formed structure. Relatively little detail was given about the self-assembly of the structures but it was mentioned that the communications were mostly over globally shared Wi-Fi and that messages to robots were addressed based on a branching representation of what angle each body unit docks to its brainward immediate master[2] with. The Mergeable Nervous

[2]The robot's direct parent when descending a hierarchy of connections beginning at the brain

Systems project provides very useful concepts which can be implemented in a modular robotic system to enable co-ordinated group sensing and actuation.

The team behind the SMORES modules has recently produced work on “parallel self-assembly" [5], a graph description similar to that of [15] is used to define kinematic structures for desired morphologies, and assemble them from separated modules. Assembly happens sequentially according to a plan based on the initial locations of separated modules, paths are determined for each module to follow and are ensured to be collision-free. This use of computed collision-free paths is likely because SMORES modules lack any sensors, beyond internal proprioceptive measures of joint angles, with which to detect and avoid other moving modules. This requires significant computational power, and global sensor data of the kind likely infeasible in a real-world scenario, which self-assembly strategies relying on recruitment of wandering modules do not need to trouble themselves with. The planner guides modules along paths in a grid so as to reach the correct locations and dock, the parallel aspect of this self-assembly method comes from the fact, that while there is sequential ordering involved, multiple modules can be driving towards docking at once except where one must be delayed to prevent collisions with another. The use of graphs to consider the connections as a topology for a given morphology means that structures can be assembled in flat forms on the surface and then fold up into 3D structures by using joint rotations and with no need for changes of connections. The authors of [5] explained that their graph theory based method used forces this work, in its present state, to only operate on branching structures which lack loops, despite the design of SMORES modules, especially in regard to the flat connector faces, meaning loop formation will often be physically feasible. This work, unlike that of [27] and [26], has a reliance in some elements on a centralised planner for some high-level aspects of reconfiguration control, low-level decisions are however delegated to the modules. It might be argued that this is a scalability problem, however that concern is of relatively little relevance for the small numbers of modules, tens or even a hundred, which can plausibly be expected for near-future real-world modular robotic applications, the real issue with this centralisation is dependence on external hardware which would not typically be available in the modular robotic use-cases expected in the coming decades.

In most cases papers on self-assembly either: address the specific challenges of mechanically achieving the alignment necessary for docking with a certain modular platform, including in one impressive example docking in flight of modular drones [76]; concern themselves with high-level and heavily abstracted descriptions of reconfiguration within a 3D lattice but with little consideration for how modules actually implement these actions and communicate [155]; or describe multi-module self-assembly with a platform but do not provide sufficient detail on the algorithms running within the robot’s controllers. Whilst self-assembly work has been developed in new directions since the end of the SYMBRION project, little of that work, except [26], has actually built on the scenarios which were considered in the likes of [27] with autonomous modules moving independently, and from separated arbitrary initial positions, to construct precisely defined forms. This sort of self-assembly appears likely to be most appropriate for near-future real-world systems, due to its lack of complex centralised decision making, part of the application of self-assembly to near-future real-world uses will also require improvements to the level of computing power, sensory capabilities and communications contained within robotic modules [7]. Self-assembly will be discussed again in Chapter 6 where a new format for describing multi-module structures is developed, based on [27], and the algorithms used to execute self-assembly by using this type of data structure are explained.

Self-assembly provides a crucial ability for modular robotic platforms, it also provides a last-ditch strategy for self-repair. Should a modular robotic system experience severe enough failures, it could resort to breaking back up into individual modules and then repeat an earlier self-assembly procedure, with new modules present, to restore itself to full working order. This is effectively the scenario used in [166] and in Murray's “naive" strategy [51]. However self-repair can be performed in more sophisticated ways too.

## 2.4 Self-repair with Modular Robots

Self-repair with modular robots is considered one of the grand challenges with modular robots [33], and the earliest detailed discussion of it dates back to Murata et al.'s 1999 [47] and 2001 papers [49] and Fitch et al.'s 2000 paper [48]. Despite the usefulness, and the age of the concept, papers on self-repair with modular robotics are remarkably rare, some of them describe themselves as morphogenesis and self-assembly. While it is often described as a desirable capability of modular robots, self-repair has not been achieved in many scenarios or with many of the platforms. Much of the research that does exist into modular robotic self-repair is purely in simulation and, at that, using what might be considered to be “magic cubes" which can climb over each other in any way that the algorithm controlling them desires and are able to support any loads they want [39]. Much of this work also focuses on general reconfiguration, an important capability for modular robots, but is either: non-autonomous and without use of sensor feedback, for some hardware demonstrations of reconfiguration [16]; or is highly abstracted, for example much of the purely-in-simulation 3D lattice work, with aims more focused to micro-scale programmable matter type “robots" rather than near-future industrial macro-scale systems. Lattice reconfiguration is popular among researchers holding more computational views of the field due to the way that motions can be represented as a discrete set [93], however, the complications of chain and hybrid systems are not so computationally difficult when working with smaller search spaces due to smaller numbers of modules. Self-repair has also taken place with robots which are not self-reconfigurable modular systems, such as Khosla and Bererton's robotic team [195] or Hex-DMR [196], while those studies provide useful practical hints on docking operations they consider how repairs of robot internals can occur in encounters between robots rather than for self-repair in structures. Other studies have considered the concept of failed robots being dragged away [197] but not the mechanics of structure reconfiguration as this happens.

### 2.4.1 Modular Robotic Fault Tolerance

This discussion starts with a brief foray into a paper describing fault tolerance in a modular robotic organism without using self-repair, Christensen et al.'s paper [106] shows a way for a 3D modular robot (ATRON [18] and M-Tran [13] are used in this study), using joint actuators in each module, to optimise its walking gait to its morphology. Each module learns independently and within minutes on a real robot can produce effective gaits.

A gait control table is used, a 2D table containing actuator actions in the cells where columns identify the specific actuator in the overall robot and rows are labelled by sensor or time conditions which trigger the transition to the next row. As the table is cycled through a walking motion should occur. Work had already been done by Bongard [133] which allowed a robot to model its own configuration in simulation and co-evolve a motion gait, Christensen and colleagues considered that by learning in real-time and the real world instead, reality gap issues could be avoided. Learning in this way is also less computationally demanding than self-modelling simulation methods.

During the evolution process modules select random actions to perform from a predefined list available to them, a form of temporal difference learning where not every possible order will be tested. Modules then learn "set points" of their own column of a gait control table, these columns vary as robots are added or removed. A reward was given to modules based on how fast the group locomotion was, as tracked by a camera looking at the arena. They were able to find optimum peak positions in the fitness landscape of possible options for actuator motions and gait elements. An "acceleration heuristic" ensured the learning was focused on improvement at the expense of widely exploring the landscape.

Modules learnt methods of walking, though in configurations where modules could be used like wheels, wheeled locomotion strategies were quicker to be learnt to the point of convergence. Learning did not work for structures where modules could collide with one-another or for those which could fall over during the learning stage. The learning was also limited in that only a fairly small set of predefined actions were available for actuators to pick, they used a number of discrete angles each could turn too rather than a realistic continuum of possible values, a system in more complex circumstances would surely have to work from a much larger set of options.

The most significant element of this paper, in the context of self-repair and comparing it to fault tolerance, was what happened when the experimenters reconfigured the robots, caused a module failure, removed a module or inserted an extra one. The locomotion speed initially

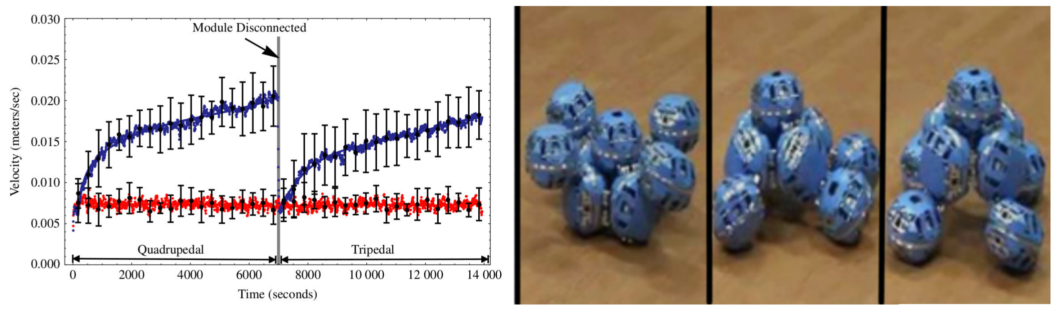


**Fig. 2.9:** A graph (left) with the blue line displaying the velocity that a quadrupedal modular robotic organism achieved when newly constructed, while learning, when damaged by removal of a module turning it into a tripod, and after time as it learnt a new controller setup to move with three legs. The red line represents the velocities achieved in these situations by a robot wiggling in a random fashion rather than using an evolving controller. Other forms of reconfiguration lead to less severe initial performance drops and the re-learnt gait matched the pre-change gait even more closely in performance terms. Some of the motions involved in one of the learnt gaits are also shown (right). Images from the paper [106].

dropped, though not to as poor a speed as would result from random wiggling, but then over time returned to near the initial speed before the change. One example of this is shown in Fig.(2.9). Other work on adapting the way a modular organism behaves, to try to mitigate for failures has been accomplished by another team using SYMBRION backbone (CosMO) modules [105]. Methods like these could prove useful in self-repair scenarios as a potential way to allow the remains of a structure to keep moving while the repair is in progress.

In summary, Christensen's work here shows an interesting capability which can be added to modular robots, however it is highly plausible that modular robots could encounter faults of forms which cannot be overcome by adapting the controller in this fashion. Useful as behavioural changes can be for fault tolerance [12], there are some forms of module fault and failure which simply cannot be overcome by them. Explicit self-repair work is considered next, prior studies in which considerations have been made regarding the removal of failed modules and the addition of replacements. A number of notable pieces of work stand out amongst self-repair research, these are considered in more depth in the following section.

### 2.4.2 Simulated Self-repair Studies

Some pure simulation papers have, alongside self-reconfiguration and self-assembly, considered self-repair processes, for example [57], [55] and [39].

Stoy and Nagpal [55] use a gradient system to fill the volume of a provided 3D CAD model with blocks representing modular robots and extend it to handle reconfiguration between shapes. Wandering cubes, how real modules could wander across all 3 dimensions is not discussed, can be attracted to any face of a docked cube. Docked cubes continue to attract others until all spaces around them are filled, or rather, given their knowledge of their relative location to the structure's seed, until all positions around them which are within the structure's volume have been filled. This is described as providing a self-repair capability as any cubes which were to disappear would leave spaces which would recruit other cubes in to them. This so-called self-repair does not consider how failed modules would actually be removed, and the structures defined are given as shaped volumes not lists of connections, likely making them inappropriate for use with the small numbers of modules to be expected in near-future modular robotic systems. Rubenstein and Shen conducted work involving similar scenarios and gradient based strategies [56]

Arbuckle and Requicha [57] consider another scenario with defined shapes and very large numbers of very small "robots". Shapes are formed by defining a boundary and when complete using a diffusion-like process to bring extra robots from outside onto the boundary wall while those previously forming the boundary wall enter the interior to fill the shape. Again the description of this work as self-repairing simply implies that if robots are removed others can refill their places, not that robots are able to detect, remove and **then** replace failed comrades.

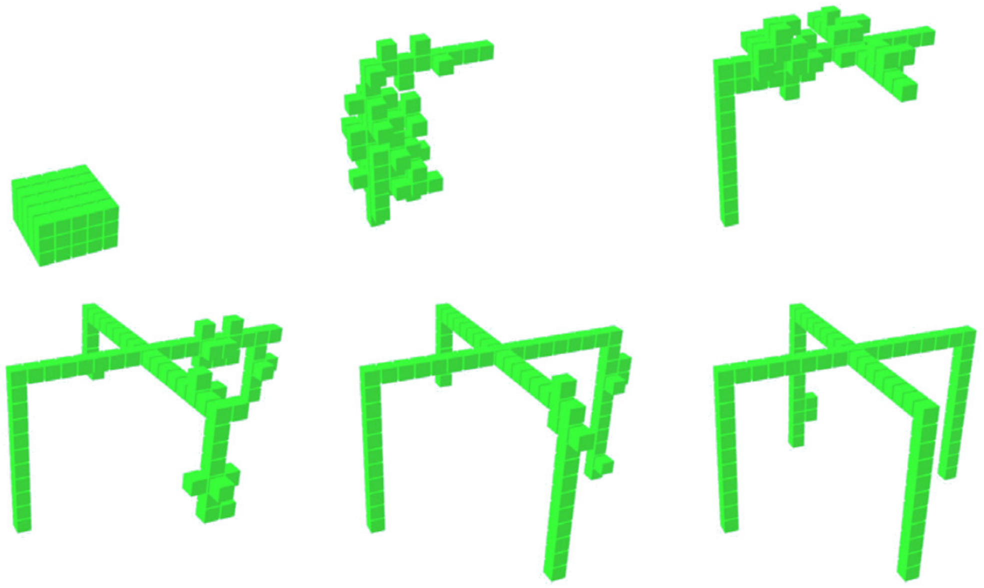


**Fig. 2.10:** An example of the flowing motions of "robots" appearing in the L-systems based study. Image from the paper [39].

Zhu and Bie [39] provides an example of a strategy to generate morphologies and repair them, assuming that failed elements have already been removed. L-systems [40] derived from a theory of plant growth are used to generate morphologies into which modules best described as "magic cubes" can "flow", specifically an L-system of rules is used to generate a character string which is then read in a "turtle" fashion to produce a branching structure, somewhat similarly to Liu and Winfield's [27] method for multiple entry recruitment. A combination of cellular automata rules, L-system effects and gradients are used to guide modules around. The method in [39] used local communication only yet still managed to create structures in an interesting way, video from the simulation shows circumstances where modules flowing towards one site would re-route elsewhere upon seeing construction at that location finished. Fig.(2.10) displays an example of this flowing motion shown with some snapshots over time. A limiting aspect of this paper, is the level of detail regarding the implementation of the method. Again this paper is limited by considering only simulations without much consideration for how modules will actually be moved across one-another in a physical system.

Given the limitations found so far, the effect these limitations have on practical use of self-repair strategies, and the inappropriateness of these simulations for structures with smaller numbers of modules, the focus now moves to self-repair work using real modules or realistic simulations.

### 2.4.3 Early Practical Work on Self-Repair

Work from Murata et al.'s team [49] [47] and Fitch et al.'s team [48] provided early demonstrations of the self-repair concept, using a mixture of hardware and simulations designed to reflect the actions that available hardware could undertake. Both considered lattice modular robotics, Fitch worked with robots which used translational degrees of freedom, a rare choice in modular robot design, Murata's robots used rotational degrees of freedom. The work explored self-repair as a novel concept and how modules other than the failed one could carry-out the removal of a failed module, and the passing of spare modules, scattered around the structure at strategic spots, back to the location where a replacement was needed. Unlike in the Self-reassembly After Explosion project [166] these pieces of work performed removal of a damaged robot and a restoration of the initial structure, via an intermediate structural configuration, rather than a total rebuild. The importance of having robots around the failed unit to detect failure is noted by Murata [47] and the concept of using a lack of communications from a module as evidence of its failure is introduced. While often mentioned as a concept in later papers and discussed in reviews, performing self-repair with real modular robots of any architecture, or simulations reflecting the capabilities of real robots, only appear in one more place again as a major topic of any work until the SYMBRION project.

### 2.4.4 Self-Repair in an ATRON Lattice

Christensen's paper [82] considers three scenarios: one relating to reversing mistaken actions, and irrelevant to the discussion of self-repair; one relating to how modular failures affect structure building, of some relevance to debris shoring [190] scenarios but again not related to self-repair; and one where faulty robots are replaced. The latter two scenarios are conducted only in simulation. Structures for the modules to build are defined using attraction points in space, when an attraction point is surrounded by modules it is inhibited so other modules do not try to approach it. When modules are removed from the structure the attraction points which they were close to are no longer inhibited and groups of modules move to replace them, due to the limited movement abilities of the ATRON modules motions use “meta-modules" consisting of groups of several modules working together to move. No consideration is given to how removal of failed modules would occur, the type of damage modelled in this study is as if modules have been pulled away rather than experienced internal failures and found themselves in need of replacement. As simulation papers go Christensen’s work does well for making consideration of the way in which modules perform their motions, but the perils of the reality gap would likely raise their head were one to try defining attraction points in the real world. Such attraction points would appear to require robots to make use of a global co-ordinate system to navigate, and specify overall structural boundaries rather than the explicit lists of connections one would expect when using smaller numbers of modules. Christensen’s work gives another example of lattice self-repair, the first attempt at self-repair using modules with a mobile or hybrid architecture took place as part of SYMBRION.

### 2.4.5 Self-repair with SYMBRION’s Mobile Modules

As part of the SYMBRION project Murray [51] developed a number of interesting techniques in his thesis research. While the thesis covered a wide range of topics those discussed in his Chapter 6 are of most relevance here.

Murray discusses work with SYMBRION [36] robots considering ways to split up, remove failed units from a group and then reform. Murray developed a strategy where a multi-robot structure will break into substructures when a module fails, the failed module was then able to move away under its own power or be moved away by other, still working, modules acting in supporting roles. The substructures then compare scores called “repair potentials", the structure with the highest repair potential remains together while the others break up into individual units which then dock with the largest structure to re-form the structure. These methods were generally very restrictive in terms of how they operated, for repair of each structure an exact order was specified which all modules had to follow, one of the key reasons for this was the SYMBRION scout modules’ inability to move sideways. The method was unable to handle structures containing loops.

A series of experiments showed that the “repair potential"-based strategy was slower than a total breakup into individual units when overall structures were very small, this was due to the time spent by substructures debating which had the best repair-potential. Perhaps the results would be improved if these techniques were applied to a system with more processing power and better communications which was able to make these decisions more quickly. With larger initial structures “repair potential"-based repair was much quicker than total breakup strategies, the time for repair rose linearly with the number of robots, the “repair potential"/substructure-based strategy had a very shallow positive gradient, the total breakup strategy had a steep gradient. Fig.(2.11) illustrates this behaviour. A correlation was also

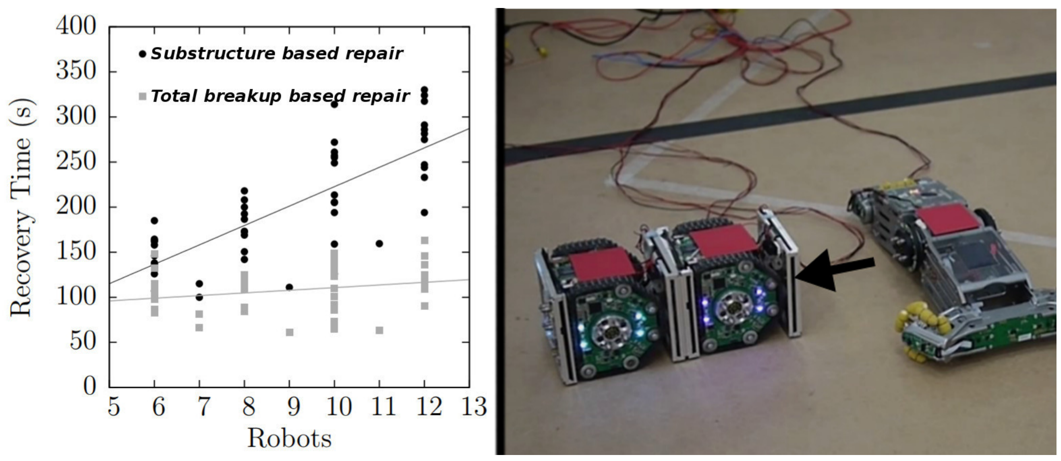


**Fig. 2.11:** A graph demonstrating how for larger numbers of robots the time taken to repair by substructure and repair-potential based self-repair and total breakup self-repair rise linearly. Note the far greater gradient for total breakup self-repair. A section of a hardware structure (right) undergoing a self-repair procedure is also shown. Images from L.Murray's thesis [51].

found between larger "repair potentials" of the best substructure and quicker times for repair.

Most of Murray's experiments were limited to simulation in the Robot3D simulator, those performed with SYMBRION hardware were very limited in numbers of robots and complexity of the structures being repaired.

Murray notes potential for improvement if multi-module groups could dock as connected units, not feasible with the SYMBRION scout hardware. He considers that by overcoming the difficulties of "precisely coordinating the movement of a structure on a 2D-plane" improved self-repair strategies would be possible.

### 2.4.6 Mergeable Nervous Systems

This paper [26] follows on and fills many of the gaps left by Murray's work, including showing a form of self-repair which can dock groups of robots with other groups of robots. The idea is inspired by the way in which biological nervous systems are morphology dependent. Using this inspiration, Mathews and colleagues develop a system which can self-heal and perform group locomotion in response to a local stimulus [26].

The overall morphology is stored as a string describing which robots should be connected to which. One robot acts as a brain unit, stimuli are received by robots and fed along towards the brain unit, at each robot they pass through they are combined together so that no individual robot, including the brain, ever has to deal with inputs from more than the number of robots which can connect to a single module. The amount of information flow through a module is proportional to the number of immediate child robots, that is to say those further from the brain, that each robot has, not to the total number of robots. Similarly actuator commands from the brain unit are fed to other units, each of which acts locally in its own reference frame in the appropriate way to produce its part of the group motion ordered by the brain. This can be seen in Fig.(2.12). Messages are addressed to child robots based on the angle at which the child is docked to its parent, messages from child robots are identified by which angle they are coming from. Robots near to the brain place a time delay on their responses to commanded actions, hence synchronising themselves with the time taken for more distant robots to receive the brain's signals.

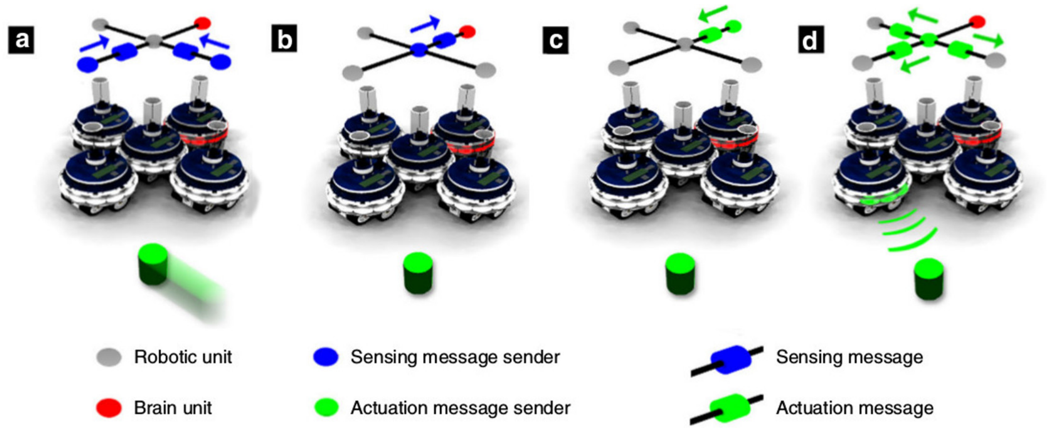


**Fig. 2.12:** A diagram showing command flows in a simple 5 robot mergeable nervous system. In a) the green light is detected by two robots and the central unit is informed, in b) that central unit takes the information flow from the two sensing units and combines it into a single message to the brain. In c) the brain tells the central unit, the immediate child of the brain, how to respond, in d) the central unit then tells it's immediate children the actuator responses necessary. Image from the paper [26].

Furthermore, the system provides the ability for self-repair. Robots produce messages according to a "heartbeat" protocol, parent robots send heartbeats to their children and the children acknowledge beats back. If these messages cease, a robot will be recognised as faulty by its children and/or parents. The substructures around it can disconnect [3], then reform either into a close approximation of the original, or into the original again if spare modules are supplied. As they recombine, all robots in the subformation which will not contain the brain of the complete structure have their internal representation of the structure updated to account for the new layout, no time for self-discovery is needed. The brain unit is then updated by its children as to the state of the newly formed structure. This self-repair can work if the brain robot fails, with new robots nominating themselves as temporary brains for their subformations. This also works for failures of non-brain robots. The work was limited to 2D by, amongst other things, the choice of platform, Swarmanoid Footbots [14], an upgrade to the s-bot design used in [60]. It was also limited to not considering platforms with flexible joints connecting them to one-another. As it has been claimed to offer "sensorimotor co-ordination equivalent to that observed in monolithic robots" it is very plausible to imagine elements of this study being used as the basis for future modular robotic control, including self-repair

[3]Gendered docking is used so the faulty robot will typically remain locked to another which must also be sacrificed when removing the faulty unit.

strategies.

This review of self-repair methods has found an overall lack of work focused on practical methods, with even less prior work considering self-repair with the kind of individually mobile modules likely to be used in a near-future system. The existing self-repair methods which require a collective task to cease before a repair occurs can, however, provide a basis from which to develop further self-repair strategies.

## 2.5 Why does Self-Repair During Continuous Motion Matter?

So far we have seen that existing work in modular robotics has not focused on attempting to perform assembly or repair while modules are engaged in other tasks. Authors talk of stopping a task to perform self-repair as if it were just “the done thing" [166] and do not appear to have considered why concurrent performance of self-repair and other tasks could matter. Here some justifications for the self-repair during continuous motion (Dynamic Self-repair) concept are presented.

A compelling reason to self-repair while maintaining motion is that it may offer a speed advantage as the group of robots can continue moving as it happens. The expectation of such a speed advantage comes from the concept of parallelisation, if the repair and the main task can proceed in parallel then the total time taken will be less than the sum of the time required for each. This allows for the time taken for a mission to be completed to be reduced as compared to self-repair strategies which require the group to stop and repair. The importance of speed might not immediately come to mind when considering modular robot applications, but there are infact many situations which can be time critical. It should be noted that these circumstances are all also of relevance to the concept of using modular robots in tasks where they are monitoring and fixing infrastructure or large machines, the self-repair of the robots during these tasks should not be confused with the overall mission, often a “monitor or maintain the infrastructure" mission, that the robots are undertaking as a group.

- For use in a disaster zone where unstable debris could collapse and crush a robot at any moment it is important that even while self-repairing it retains the ability to rapidly move out of the way of falling objects, lest the operators find themselves having to rescue their robots before they can resume searching for survivors.

- For use inside a nuclear reactor operators will want to minimise the time their robots spend inside to minimise the heat and radiation exposure of components. The less time spent in the hot radioactive environment during each monitoring mission, the longer a lifespan the robots will have before they succumb to radiation damage [19] to electronics or to effects such as neutron embrittlement affecting mechanical elements. This scenario is depicted in Fig.(2.13).

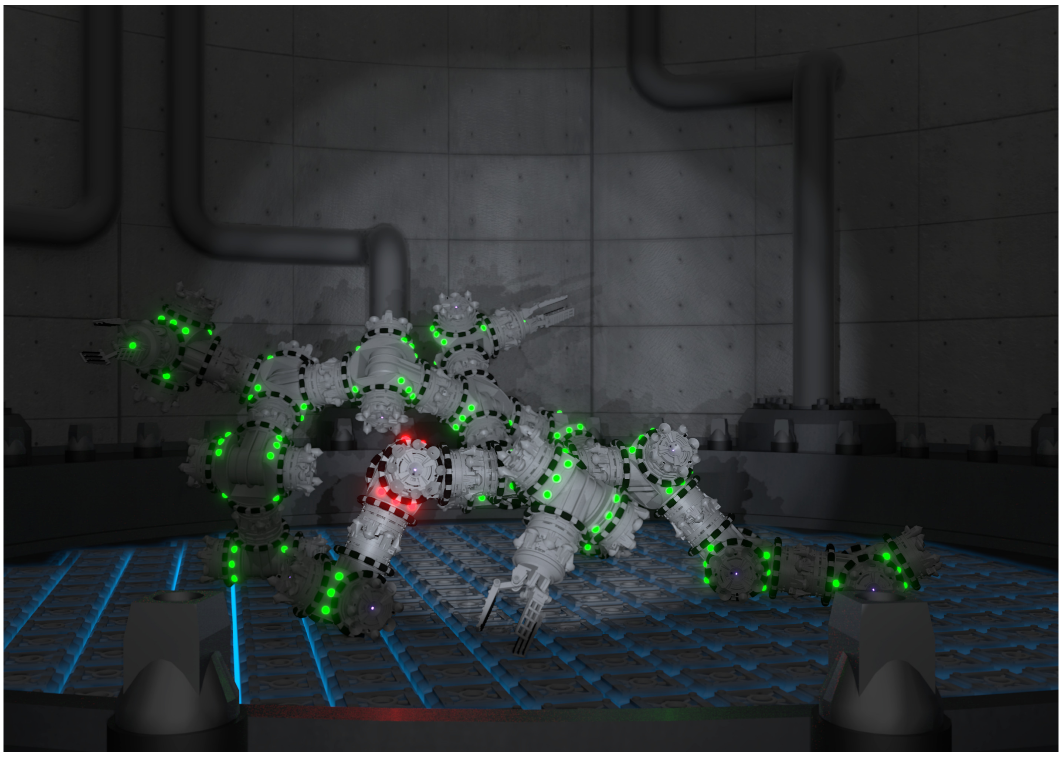


**Fig. 2.13:** An artist's impression of a modular robot organism operating within a nuclear reactor. A failed module within the organism is illuminated in red. Without a dynamic self-repair strategy, replacement of this module could incur a considerable cost in time and leave the organism as a whole unable to perform any group tasks while performing self-repair.

- Robots working on critical infrastructure may be required to perform repairs to the infrastructure while it is sustaining damage in real time, for example a robot experiences a failure of a module and must self-repair while attempting as a group to perform repairs on a fraying cable of a suspension bridge during a hurricane. Although this is not a locomotion task in the way that many other situations described here are, this too requires the robot to maintain some elements of its group action while repairing itself. Failure to act quickly in these kinds of situations could lead to extensive and irreparable damage to infrastructure should a robot be slowed by its own self-repair procedures. This same scenario could apply to robotic modules acting to shore-up debris around a disaster survivor, it would be undesirable for this modular shoring structure to loose integrity because one module in it requires replacement.

- Battlefield use is another example where speed is necessary. It is certain that any robots which prove themselves useful fixing civil infrastructure will also be used to maintain military equipment, if these robots must stop to self-repair themselves they become a sitting duck and an easy target. Robots which could keep moving while self-repairing would have a better chance of survival.

- Modular robots could be of use to clean the hulls of ships while the ships are underway.

For the time for which robots are clinging to the hull underwater either the ship will experience increased drag and fuel use [20] or will be forced to slow its velocity through the water to prevent robots being swept off. Under such circumstances robots needing to self-repair while performing their hull cleaning routines must do so quickly or cause either increased pollutant and greenhouse gas emissions from the ships funnel or delayed arrival of cargoes to port. This example also indicates how dynamic self-repair may be concerned where robots are having to resist environmental forces, in this case drag from the ocean rushing past, robots may be required to self-assemble or self-repair while performing tasks in which lone modules separated from the group would be unable to withstand forces and be swept away.

- In the monitoring and repair of critical infrastructure time is money, time when a robot must stop moving to self-repair is time when it cannot be doing useful monitoring and repairs to the infrastructure. Faster times for self-repair operations will reduce costs and enable more inspections to be carried out, both combining to give better warning of impending failure of infrastructure and more opportunities to repair it.
- It is also possible that dynamic self-repair with ground-based modular robots could provide insight into airborne situations where stopping to self-repair may not be possible. Fixed-wing drones cannot stop forward motion without losing lift and although rotorcraft may be able to hover on the spot they are unable to immediately reduce their velocity from cruise speed to a stop so reconfiguration during motion could be necessary as well as desirable.

### 2.5.1 Revisiting the Hypothesis

Given the possible usefulness of self-repair during continuous motion a hypothesis can be proposed, one which formed the basis of our 2018 paper[31], which will let us decide whether self-repair on the move is as worthwhile an idea as it initially sounds.

**Hypothesis 1.1:** *Modular robots using self-repair strategies which can operate while the group maintains collective motion will be able to complete tasks faster, and more effectively, than if using a self-repair strategy which requires the system to halt collective tasks so as to perform self-repair.*

Focusing specifically on locomotion as the task to complete while self-repair occurs, a series of experiments can be considered to address this hypothesis. The experimental setup will involve a group of robots initially assembled in a morphology for group locomotion, they will begin to cross a distance between a start and finish line but at a random time one of the units would fail. Failure in these experiments would be assumed to be a total death, the module would cease all actuator function, all sensing and all communication. It would be this lack of communication, potentially detected with some form of “heartbeat protocol" as used in [26] and in [47], which alerts others to the module’s failure. At this point there are three courses of action the surviving robots can take:

- **Fault masking**, attempt to continue the current group locomotion despite the failed unit. There may be steps taken to adapt locomotion slightly to account for the damage, such as in Christensen and Schultz's work [106], but overall it would be expected, in driving based scenarios, that the damaged module would provide extra friction against the floor. For 3D structures this attempt to ignore faults and rely purely on other modules for redundancy may be inappropriate. If a joint along an arm fails, robots may not be able to continue certain tasks without replacing it. Due to limitations in the accuracy with which robotic simulators handle frictional effects, this strategy is not tested for comparison to self-repair strategies in this thesis.

- **Static self-repair**, based on the work of Murray's thesis [51] and Mathew's mergeable nervous systems [26] the robots would stop, break up into separate formations, eject the failed robot and then reassemble. Reassembly would be into the original form. It is assumed that spare robots would be available to replace the failure, in real-world situations this would mean the group would usually carry some spare robots with it that were not essential for group motion, much as were used in [49] and [47] when considering self-repair for the first time.

- **Self-repair during continuous motion**, the robots would self-repair while continuing group locomotion. In 2D this would be similar to Murray's work, but with a velocity transform applied, in 3D structures it would typically require robots to be able to climb across one another to get to the necessary locations to remove and replace failed modules. The 3D version, which is outside the scope of this thesis due to time constraints, would go some way to addressing what may happen when robots are operating in an environment where robots alone move slower than as a group or are unable to move at all. Assumptions about the presence of spare modules are the same as in the ordinary self-repair situation.

Concept videos with conceptual depictions of these strategies can be found at [211] and in the supplementary material (Self-repair_concepts.mp4). These videos show an example of the three repair strategies operating in a 2 dimensional situation, and depict ways in which the rest of the robot group could respond to an individual failing. Fig.(2.14) shows a series of still images from the classical “static" and continuous motion “dynamic" examples.

Experiments testing the Dynamic Self-repair hypothesis are conducted with groups of modular robots which are homogeneous from a mechanical perspective and all run the same software controller. The robots communicate both globally and locally, with strategies which are distributed but make use of hierarchies at some points. These hierarchies are assigned using similar “seed" principles to those SYMBRION used when a module encountered an obstacle

[74], parts of the work consider dynamic methods of reordering these hierarchies based on circumstances.

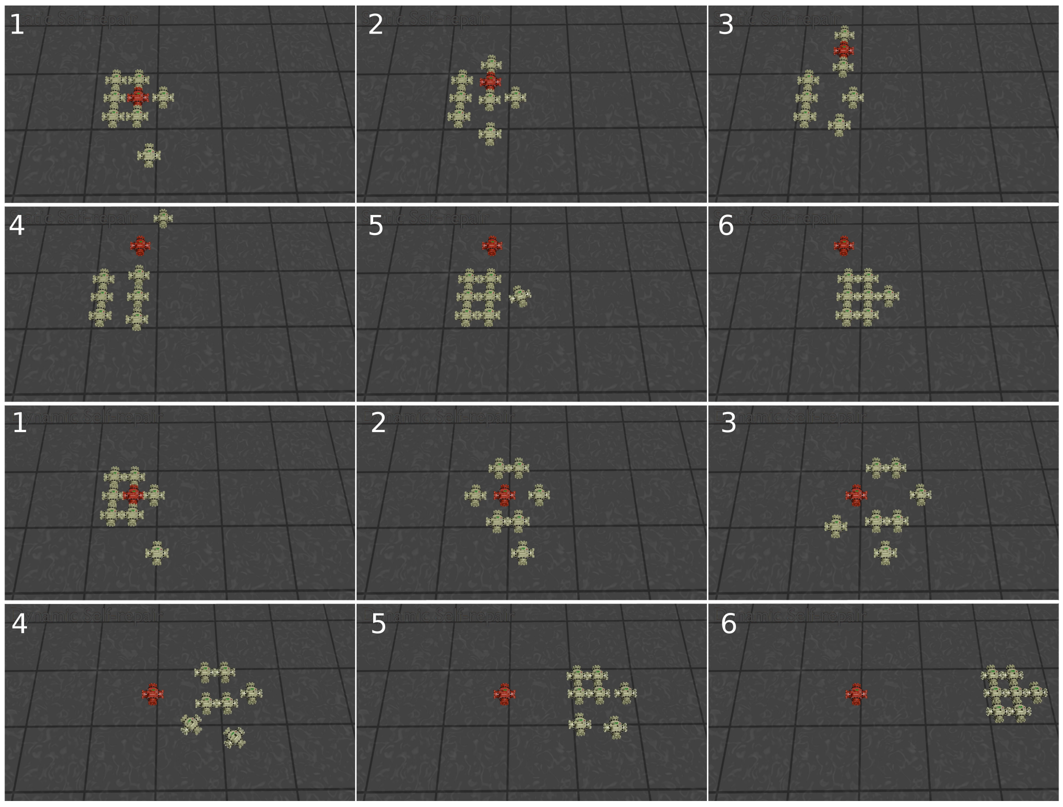


**Fig. 2.14:** A series of still images showing how classical (upper pair of rows) and dynamic (lower pair of rows) self-repair could be expected to respond to a failed module, marked in red. Notice how the structure continues in motion in the dynamic self-repair scenario.

### 2.5.2 Hardware for Dynamic Self-repair

Given the lack of practical focus in so much of the existing self-repair work, it is considered to be of paramount importance to test the hypothesis with accurate simulated modular robots and real hardware modular robots. Simulations with “magic cubes" cannot suffice to provide proof for, or against, the hypothesis. The concept of embodiment [206] must be introduced here, the overall abilities of a system are strongly influenced by how it is able to interact with its environment, not solely by its internal controller software. Low-level sensory and actuation processes can have major effects on behaviours influenced by the detailed physics of a situation regardless of how higher-level software may be acting. Simulated environments rarely have the detail necessary to fully reflect these embodied interactions. Beyond the purely simulated we must also recognise that robots unrepresentative of those we expect to find in use as near-future real-world systems, for example, without docking ports or with unsuitable drive mechanisms, would also be unable to properly test the hypothesis, embodied effects would not match between experiments and applications if an thoroughly inappropriate body for the final use case is considered. Hardware must therefore be developed to provide a combination of capabilities which previous platforms have not been able to provide. Testing the hypothesis therefore requires both production of hardware robots with appropriate capabilities, and the production of an appropriate simulation environment in which to perform other forms of experiment. This chapter's review of the modular robotics field has identified what is lacking in current designs, and discussed how existing hardware has performed existing forms of self-repair. In the next chapter we will use these conclusions to create a specification for the robotic platform, necessary to test the Dynamic Self-repair hypothesis.

## 2.6 Summary

This chapter has considered the state-of-the-art in modular robotics and found work on self-repair to be lacking, especially in terms of repairing while other tasks are in progress for which no attempts have been made. Existing modular robotic platforms have been discussed and the lack of one with the full set of capabilities, at least representative of, those we would expect, and others would recommend [30] [190], for a field-ready systems has been noted. The next chapter examines the hardware requirements further and introduces the design of the Omni-Pi-tent platform.

# Chapter 3

# Hardware Concept and Design

This chapter describes the initial hardware designs for a robot capable of forming part of a physically connected group and performing self-repair functions which can operate while the robot group maintains collective motion [31], as are required for Dynamic Self-repair, see Sec.(2.5). Having the ability to maintain motion while self-repair occurs should offer significant advantages for robots in time-critical situations, for example, minimising radiation dose on a robot operating in a reactor; for robots which must be able to react quickly to dangers such as attempting to self-repair while performing search and rescue under rubble which may further collapse at any moment; and for robots in locations where mobility is reduced such as trying to self-repair while crossing obstacles or resisting environmental forces [31]. Throughout this work it has been intended that the Dynamic Self-repair strategies developed be constrained to using only those systems and sensors which could be plausibly implemented in a future modular robot for infrastructure monitoring or urban search and rescue, hence, introducing the concept and outline of the platform earlier rather than later allows us to recognise early the constraints which this will place on to the Dynamic Self-repair algorithms to be developed. These constraints mean externally supplied sensor information or computing systems external to the robots cannot be used. Hence, the sensing, actuation and computation capabilities the robotic platform should have are introduced in this chapter. The Omni-Pi-tent modular robot platform has been developed to meet the following goals, it uniquely combines:

- Genderless docking
- Omnidirectional motion
- A 2-Degree of Freedom (DoF) hinge
- A full suite of the sensors needed for autonomous docking
- Both local and global communication systems, and

- A central computer (in this case a Raspberry Pi Zero W) running a Linux OS.

The rest of this chapter is structured as follows: Sec.(3.1) explains the unique contributions which hardware development provides to robotics projects, whilst Sec.(3.2) considers the requirements of a platform capable of Dynamic Self-repair, Sec.(3.3) explains the conceptual implementation of these requirements within our design, Sec.(3.4) and Sec.(3.5) provide a brief overview of the electronic architecture and practical production of the modules. Once these facts are introduced, later chapters cover further details of the design's implementation.

## 3.1 Embodiment, and Learning Lessons from Hardware

Within many robotics projects, the Dynamic Self-repair project included, there are, in effect, three levels of abstraction. There is the highest most abstract level, the theoretical underpinning for the work undertaken, at this level robots are represented by logical or mathematical objects and their abilities and limitations are deduced via chains of reasoning. The middle level is that of software, here the decision making processes of the robot are reflected in something approaching their full complexity, but the world with which they interact and the means by which they do so remains an idealised representation. Finally there is the lowest level, the real physical system, in which abstract depictions are no longer present and both the software and hardware are present in their full forms.

Whilst hardware development brings results in its own right, it is indeed a legitimate question to ask why it should be occurring within a project where the key question relates to the feasibility and effectiveness of robotic behaviours. To answer this the concept of embodiment should be considered.

Embodiment is often described as "intelligence requires a body" [206], expressed more fully it indicates that physical constraints play a major role in the way that an intelligent system interacts with the world around it, and points to the concept of morphological computation, whereby the physics of actuator and sensor systems can result in carefully controlled actions without requiring, or while only minimally requiring, input from a decision making controller. Whilst simulations can reflect some aspects of embodiment, the effects which embodiment exploits are often deep within the detailed dynamics of systems. Even advanced simulators struggle to accurately reproduce these details [207], a form of reality gap [24]. Embodiment effects allow a robotic system to complete a task without requiring explicit planning and control of all aspects, some proportion of what could otherwise be a highly computationally intensive task is outsourced to the physical dynamics of the system. In the case of modular robotic behaviours with the Omni-Pi-tent platform this includes the last millimetres of misalignment during docking being corrected for by the geometry of docking spikes and pits, see Sec.(3.3.1), and the use of communication methods with intensities which drop below the noise threshold with range to convey relative positional information, see Sec.(3.3.4).

Another key limitation of simulations is the virtually boundless opportunities they offer. All too easily physical parameters can be set values which enable results to be gained most easily, rather than to values reflecting physical reality. All too often scenarios can drift away from physical practicality, without any obvious way for an experimenter to realise this is occurring. Hardware development therefore provides a method to anchor simulations to reality, uncertain parameters to be set within simulations become fixed to the values of observable measurements and the physical constraints inherent in the real system are replicated within the simulation. The constraints which hardware provides in this way go beyond those of mere physical possibility and also provide constraints of practicality. Even in a highly physically accurate simulator it is still a simple task to produce mechanisms which, while not violating the laws of physics, would prove utterly impractical to fabricate. A simulation model, however detailed, is rarely, if ever, designed with questions in mind such as “how can wires be run through this joint?", “how many milliseconds will a burst communication last, and can I still use sensors during that time?" or “where should the battery be placed, not only to evenly distribute mass but also to allow access to charge it?", each of these questions, however, has implications for robotic design as profound as those derived from questions at the theoretical and simulation levels of abstraction. Developing hardware and simulations in parallel avoids the risk of simulating the impossible, simulations allow options to be explored, whilst each test of hardware subsystems keeps constraints on performance ever present, both strands of work tend towards one another until they meet. Within the Dynamic Self-repair project it would have been possible from the start to simply define an arbitrary robot equipped with every device imaginable, and with unlimited sensor resolution and actuator capabilities, but this would not be guaranteed to give results applicable to robots which could be fabricated in the foreseeable future. Working with hardware ensures that not only are the results found applicable to real-world systems, but also that they can be expressed and communicated in a manner interpretable by hardware researchers and industry, with the constraints of real world practically built in from the start.

## 3.2 Design Requirements

With the rationale behind hardware development explained this thesis can now consider the functionality required for an individual module able to form part of a structure which will enable the testing of self-repair during continuous motion, deducing the required set of capabilities for Dynamic Self-repair constitutes one of the first steps towards developing it. These requirements are broadly similar to those in [30] for an infrastructure monitoring robot. A key difference being that whereas the authors of [30] envision operational uses of ruggedised industrial modular robots in harsh (dusty, wet, corrosive, hot ... ) locations, the

Omni-Pi-tent design has been built to operate in relatively benign environmental conditions such as laboratories, offices and other indoor spaces in which experiments and demonstrations have been run. Key design requirements include:

- Active genderless docking: this is necessary to let robots free themselves even when the robot they are docked to fails. Mechanical links tend to be stronger and stiffer than magnetic docking, making the construction of complex structures more reliable [18]. They consume power only when changing between locked and unlocked states, whereas electromagnetic docking uses power at all times while the connection is held. The HyMod [42] platform and the RoGenSid [73] port on ModRED provide good examples of such mechanisms.

- Omnidirectional locomotion: necessary for group locomotion. For docking to a moving structure this is especially important, as doing so requires a robot to maintain a compass orientation independently of the driving direction, omnidirectional drive is also important for enabling groups of docked robots to drive together without running into situations where docked robots would encounter major resistive forces when attempting to move their companions perpendicular to their wheels. To date, the only omnidirectional modular robots have been Trimobot [62] and some SYMBRION [36] modules. Robots like the two-wheeled SMORES [35] or the tracked SYMBRION Scout [36] would be unsuitable for the situations robots are envisioned to encounter during Dynamic Self-repair experiments.

- Modules should contain within them all the sensors needed for autonomous docking. Many modular platforms, such as SMORES [35], have had few sensors and relied on specialised sensor modules for these purposes. Such modules can limit reconfiguration options and act as single points of failure.

- Joint actuators in each module: for a 3 dimensional system this is necessary to allow structures to fold up from the floor into the 3rd dimension. SMORES (4 DoF) [35], HyMod (3 DoF) [42], SYMBRION [36] (2 DoF) and Trimobot (1 DoF) [62] all have forms of these within their modules. It is considered in this work that being able to lift a chain of two modules is the minimum necessary to form interesting structures. For robots of mass $M$ with distance between module centres $r$, the torque requirement for lifting such a chain of two modules is given by Eq.(3.1).

$$\tau = gMr\cos(\theta) + 2gMr\cos(\theta) \tag{3.1}$$

- Modules should each contain equipment for global communication across the swarm, and local line-of-sight communication to neighbours they are docked or docking to.

- Common robotic hardware should exist in the real-world and in simulation. The high-level interfaces by which a user can program a robot's behaviour should be as similar as possible between simulation and real world hardware.

With these fundamental requirements in mind, the next section provides details of how they were to be met in the initial Omni-Pi-tent design. Given the necessary complexity of the design this section relies heavily on images and diagrams.

## 3.3 Design Features

The Omni-Pi-tent modular robot platform has been designed and constructed to enable the higher-level goals of the Dynamic Self-repair project to proceed. A thorough literature search found no modular robot in existence with all of the necessary attributes. Fig.(3.1) introduces the design. Omni-Pi-tent's key features are further explained in the following subsections.

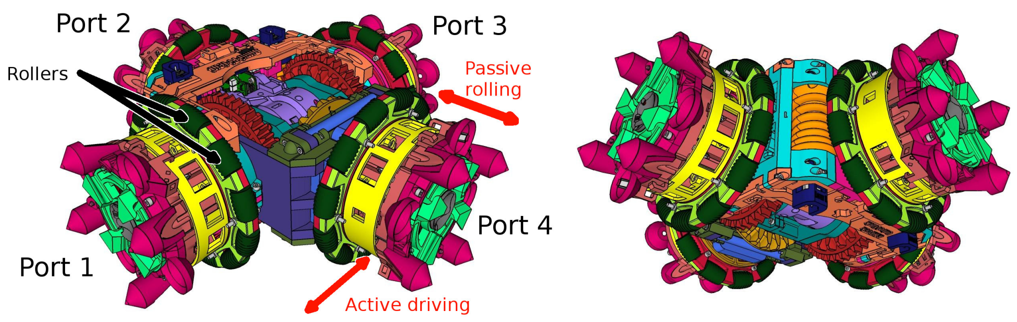


**Fig. 3.1:** An early concept for the Omni-Pi-tent modular robot platform, note the rollers (dark green) around the edge of the omniwheels allowing each wheel to actively drive while also passively sliding in a perpendicular direction. Four omniwheels were used instead of 3 as this enabled the robot to be designed with 4 ports to fit a cubic lattice structure. The ports are numbered clockwise with the hinged port at the front being Port 4. The module overall had a 3.1 kg mass, somewhat increased in later design revisions, and a lattice spacing of 29.2 cm.

### 3.3.1 Docking Ports

When initially designing the docking ports a wealth of existing designs were available for inspiration [103] [90] [92] [73] [95] [109] [111] [112] [113]. Magnetic docking mechanisms were ruled out due to the difficulties in connecting and separating at the desired places in a multi-module structure if no mechanical elements are involved. Electromagnetic systems were ruled out due to their constant demand for power, and switchable or electropermanent magnets, such as those on of the EP-face upgrade to SMORES [103], were ruled out due to the interference that the close-range presence of magnets powerful enough to hold robots together against gravity when performing 3D reconfigurations could cause to compass-based navigation. This left only mechanical connectors in the pool of options. One of the key requirements was that the connector must be genderless or bigendered, necessary to ensure any port on any module could connect with any other and avoid some of the limitations which ATRON [18] encountered. Amongst the genderless and bigendered systems those with single-sided disconnection capacity, typically only found in genderless systems, stood out as particularly relevant. This capability being necessary when dealing with fault tolerance and failed modules, unless one wishes for any failed module to have a strong risk of taking other connected units out of action too, should they be docked such that they are unable to escape its grasp. A more limited selection of docking port designs were therefore considered, notably the HiGen [90] connector from HyMod [42], the RoGenSid system [73] from ModRED [50], and the SINGO [92] connector from SuperBot [94] and the GHEFT mechanism [95] from the STORM platform [97]. These single-sided disconnect systems are more typically genderless than hermaphroditic, as the latter type usually requires both ports to co-operate to release, better for some forms of safety critical mechanism where disconnection without both sides agreement could be disastrous, such as spacecraft airlocks, but not suitable where the risk of either side being unable to disconnect is more concerning. Whilst the ability to retract, allowing rotation of modules within a lattice, which HiGen offers is attractive, a RoGenSid based design was chosen for simplicity and ease of enlarging elements of the design for improved strength when dealing with large loads and 3D printed plastic, the design used on Omni-Pi-tent is shown in see Fig.(3.2).

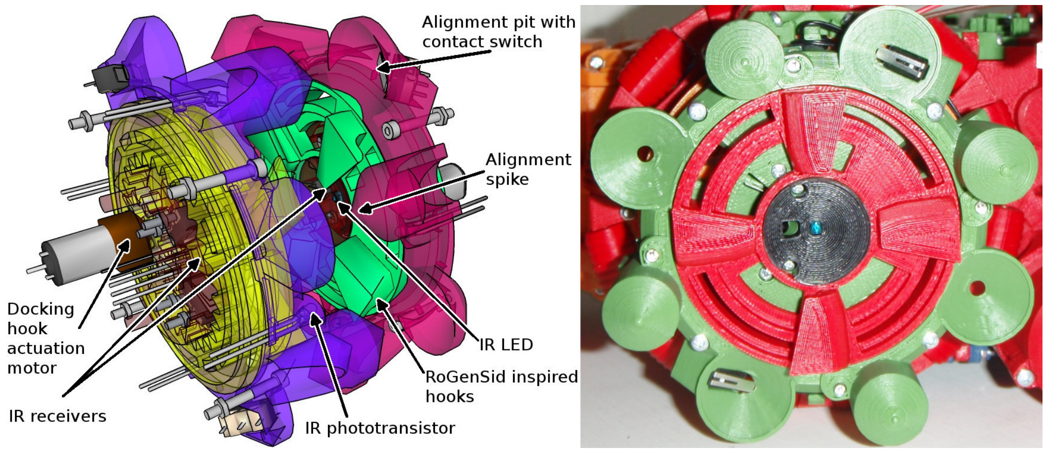


**Fig. 3.2:** Omni-Pi-tent has a genderless mechanical docking system actuated by an electric motor. The hooks are based on the previously tested RoGenSid [73] docking mechanism and allow single-sided disconnection by either robot in a docked pair. The spikes around the rotating centre of the docking port help guide the robots together with the proper alignment as they dock, enabling docking without the need for perfect alignment before docking begins. An IR LED and 38KHz receiver are centrally mounted where they will remain in line-of-sight between docked modules, no electrical connection is made between modules as that would complicate docking by requiring extremely precise connection of contacts. Modular robots in future use may forego using inter-module electrical connections for similar reasons, amplified by the ways in which a harsh environment may oxidise exposed connectors.

Each port on Omni-Pi-tent rotates back and forth between a locked and unlocked position, when two ports approach one usually stays locked, the other is unlocked, once correctly positioned the unlocked port locks itself. Both ports remain firmly connected until either decides to unlock to escape, see Fig.(3.3). Rotational alignment about the port-to-port axis is preserved when connected via an array of spikes and pits around the ports, these are of large sizes as this allows for larger misalignments in the ports to be easily corrected as the ports push together in the final stages of approach, for example compared to SYMBRION's need to align to within $< 3$mm [98] laterally, Omni-Pi-tent modules need only be within 13mm for spike and pit to mate. The docking ports also provide mountings for docking and proximity sensors, as well as line-of-sight communication systems. The use of 4 ports in a square arrangement was chosen as this gives some of the most user friendly options in terms of the shapes which can be formed by docking modules together, and of how 3D structures can fold and unfold. When viewed from above ports are numbered clockwise 1 to 4, port 1 being the port clockwise from the hinged port, port 2 being opposite to the hinge, port 3 180° away from port 1, and port 4 the port which can be raised when the hinge is used.

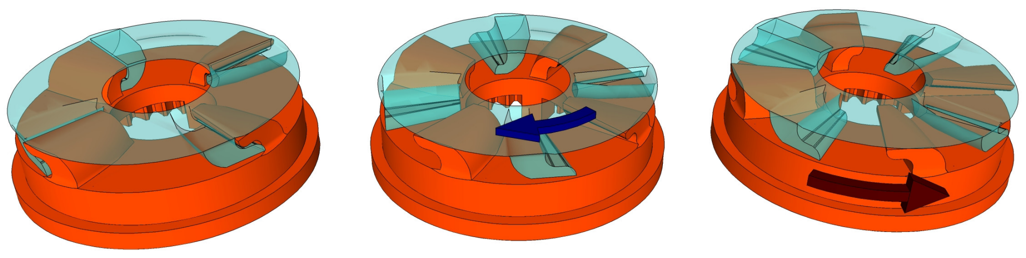


**Fig. 3.3:** Diagram of an Omni-Pi-tent module's docking hooks rotating back and forth between locked and unlocked positions, noted the way in which unlocking either set of hooks can break a docked connection. From a docked situation (left) undocking can be performed either by rotating back the upper hook set (centre) or by rotating the lower hook set (right).

### 3.3.2 Omniwheels

Omniwheels allow the Omni-Pi-tent modules to move in any direction whilst maintaining any orientation, avoiding the steering difficulties SYMBRION, for example, suffered where long distance retreats where necessary to correct lateral misalignments when docking. Rollers around the edge of these wheels turn freely at 90° to the axis of the main wheel, allowing each wheel to passively slide in a direction perpendicular to that in which it can provide a driving force. Four wheels reflect the 4 fold symmetry of the docking port arrangement, although 3 omniwheels in a triangular arrangement could produce the same versatility of motion. This use of 4 omniwheels, one per port, allows the wheels to be sized so they can run when the module is upside down, giving it some level of vertical symmetry. Examples of how these wheels can operate together are shown in Fig.(3.4).

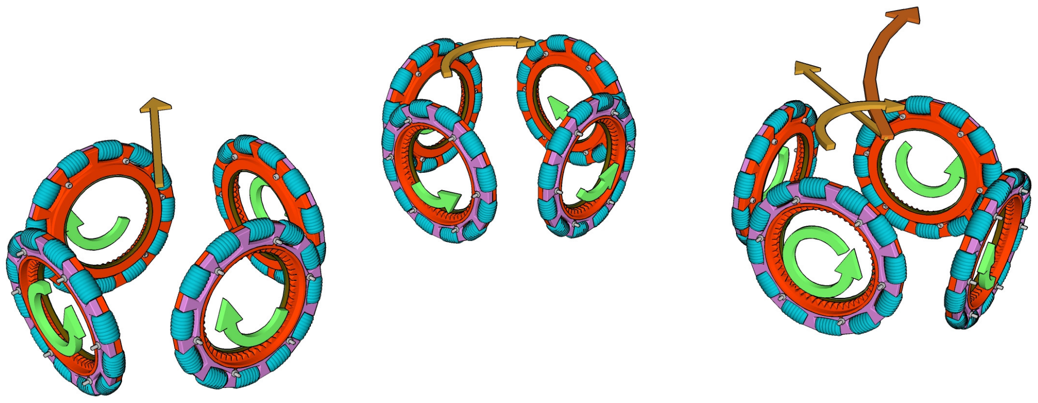


**Fig. 3.4:** Four Omniwheels provide modules with excellent manoeuvrability. By being able to actively drive in one direction and passively roll in a direction perpendicular to that, each wheel can run so motions can be combined to produce motion in any direction (left), rotation about the module's centre (centre), or both translation and rotation at once. In this example they are added to give driving motions in an arc about a circle centred outside the module (right). A particularly important use case for these omniwheels is that of allowing docked structures of modules to move as gracefully as lone robots, whatever relative orientations the docked modules may have. For comparison, modules using two-wheeled or tracked differential drives, such as SYMBRION or SMORES, can only move as a group for modules arranged in structures where all wheels are parallel, or structures which must first partially fold up off the ground before driving.

### 3.3.3 Docking Guidance

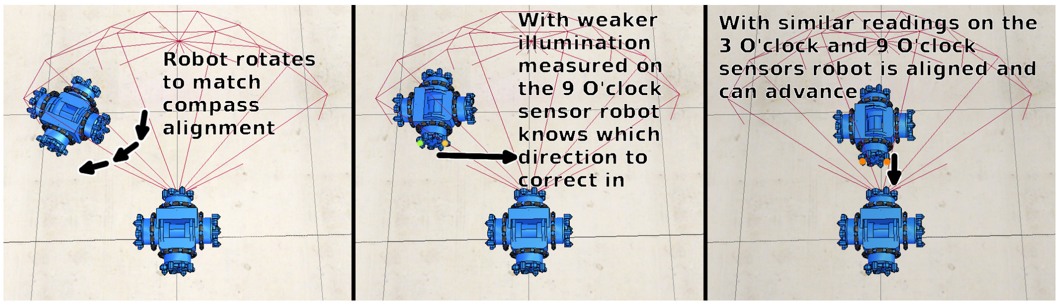

Robot rotates to match compass alignment
With weaker illumination measured on the 9 O'clock sensor robot knows which direction to correct in
With similar readings on the 3 O'clock and 9 O'clock sensors robot is aligned and can advance

**Fig. 3.5:** An example of the docking guidance system in operation, once compass alignment is handled, for whatever pair of recruiting and recruited ports are to be involved the approaching robot can compare readings on sensors around its docking port face. The sensor at a 9 O'clock position (green) is towards the edge of the cone and getting weaker readings than the 3 O'clock sensor (yellow). To equalise these readings the approaching robot must laterally align towards the centre-line of the recruiting robot's illuminated cone.

The multitude of ways to get robots to travel to defined positions and orientations so as to achieve docking to other robots or static infrastructure can form a project all of their own. For the purposes of Omni-Pi-tent, the key requirements for the method of docking were that it must be able to work without explicit global positional information, it must not require the presence of devices other than the robots themselves, it must not require intensive computation, and it must be conceptually simple. The requirement to avoid use of global positional information rules out use of GPS, or equivalent, satellite systems. This type of sensing is also not practical to the necessary resolution when using reasonably-priced receiver chips indoors. The same requirement ruled out any docking method requiring overhead tracking cameras, which would have also restricted Omni-Pi-tent's operation to use only in areas with overhead infrastructure present. The computational capabilities of the Pi Zero W made image recognition methods requiring cameras and computer vision algorithms an impractical solution, solutions based on the methods used by Sambot II [100], M-TRAN [102], or the SMORES camera tower sensor module [101] were therefore not available. This left open options based on combining geometry with analogue sensors.

A cone of light cast by an emitter can provide a simple geometry which allows for robots to align using local sensing. To remain within this cone of light a robot approaching a recruiting docking port must, as it gets closer, become more closely aligned.

To this end, docking ports are fitted with a central Infrared LED and a ring of light sensors. A recruiting robot shines this LED and approaching robots, within the 68cm detectable range of the cone, drive towards it by moving in such a direction as to ensure that the sensors on opposite sides of their approaching port both stay illuminated. The original plan during hardware development was that measurement of the simple presence or lack of light on a particular sensor would be enough to perform this guidance. However it was noted that the type of LED used [121] cast an axially symmetric pattern of illumination which reduced in intensity greatly from 50° to 65° away from the centre-line. This variation in illumination with angle could therefore be used to let an approaching port not merely align with the docking port at the moment it reached the apex of the cone, but rather balance illumination intensity on the sensors either side of the port so as to match centre-line to the recruiting port from as far away as possible. Simulation work, see Sec(4.8), found that docking controllers based on balancing the intensities performed better especially for docking to moving robots, hence those were the ones implemented in hardware.

However, from IR intensity levels alone it is possible for readings to give ambiguous information about the robots' relative positions. On the 2D plane of the floor robot alignments have 3 degrees of freedom, the translational distance between ports, the translational sideways misalignment and also rotational angular misalignment. From IR intensity readings alone the effects of rotational and translational misalignments would be hard to tell apart. Whilst some robotic docking systems have overcome such complexities [114] [115], this does tend to force them to use more precisely pre-planned approach manoeuvres. For Omni-Pi-tent it was considered a simpler prospect, and crucially one which would enable a wider range of possible docking scenarios, to use an extra sensor's information to constrain these degrees of freedom further. Using a compass to get a global directional heading was not considered a serious difficulty when deciding on the docking method, so this was included despite the complexities it would later cause in hardware, see Sec.(5.7). Fig.(3.5) shows this principle of rotational and translational alignment. This compass, specifically a 3 axis magnetometer and accelerometer chip in the Omni-Pi-tent design [78], also allows for some level of global navigation without positional information, it is useful for allowing robots in motion to compare relative directions of travel and providing a consistent heading in which a robot can wander even when changing the orientation of the robot throughout, as omniwheels allow the modules to do. Finally, use of a compass also makes those scenarios in which robots are expected to drive in a specific direction to reach a goal more practical, but again without involving explicit global positions. During docking compass information is shared between robots using Port-to-Port communications.

Although 2 light sensors (phototransistors specifically) could be enough to perform this action on a 2D plane, 4 light sensors form a ring around each port as when 3D reconfiguration is considered it is useful to be able to align on both the horizontal and vertical axes. The 4 sensors, due to the nature of the spike and pit geometry around the docking port, are not directly above or to the side of the centre-line of the port, but have instead been shifted round by 8.3°. This provides the advantage of allowing the upper and lower sensors to also be offset to the sides of the centre-line and therefore provide extra horizontal alignment data than the left and right sensors alone could provide. Fig.(3.6A) displays these locations on a docking port as compared to the position of the central LED.

Of course while the requirement for a set of “light sensors" and “IR LEDs" are simple to state, developing these systems to perform reliably on real robots proved a rather more involved undertaking. Near-Infrared LEDs and phototransistors [122] were chosen so as to use a wavelength region where signals would not be drowned out by visible light. However, Near-Infrared wavelengths are still significantly polluted by electric lighting and sunlight. A phototransistor in common-collector mode gives an output which is strongly affected by these background levels, hence inter-module signals could easily be lost. A beacon system with a feasibly powerful LED attempting to use such a phototransistor’s output directly would only manage a maximum range of several cm, even then only in darkened conditions. Yet as sunlight gives a roughly constant level of illumination, and electric lighting varies at twice mains frequency, a modulated infrared signal and high pass filter can be used to filter out those relatively slowly varying background levels. A system consisting of a high pass filtering amplifier, a tight bandpass amplifier and a software homodyne detector allow for the phototransistor output to be read by ADCs within the robot and for background interference to be eliminated. A 5KHz frequency for this IR beacon system was used to give a balance between being well above the frequencies at which interference is present and being below the maximum frequency which the phototransistors used could respond to. The system constructed allows for guidance from as far away as 68 cm, in any commonly encountered lighting conditions including direct sunlight. Omni-Pi-tent’s system has some similarities to that used on SYMBRION [124], however appears to be more reliable, longer ranged, and is able to operate with multiple recruiting ports and robots simultaneously active in the same room. Details of the electronics and software used within this guidance system are given in Sec.(5.2.2).

The 5KHz LED and guidance phototransistors also serve double duty for proximity-sensing, the difference being that in guidance mode a passive reading of ambient IR light levels is taken, whereas in proximity sensing mode the port's LED is active and the phototransistors detect reflected light levels. Using a system based on the strength of measured reflections is strongly affected by the size and material of objects encountered, for coloured plastic obstacles on the same order of size as an Omni-Pi-tent docking port ranges for useful detections are below 15 cm. The shorter range for proximity detection than docking guidance being because the IR illumination flux at the phototransistors drops off with $1/R^4$ for reflection based systems but only $1/R^2$ for beacon detection.

### 3.3.4 Port-to-Port Communications

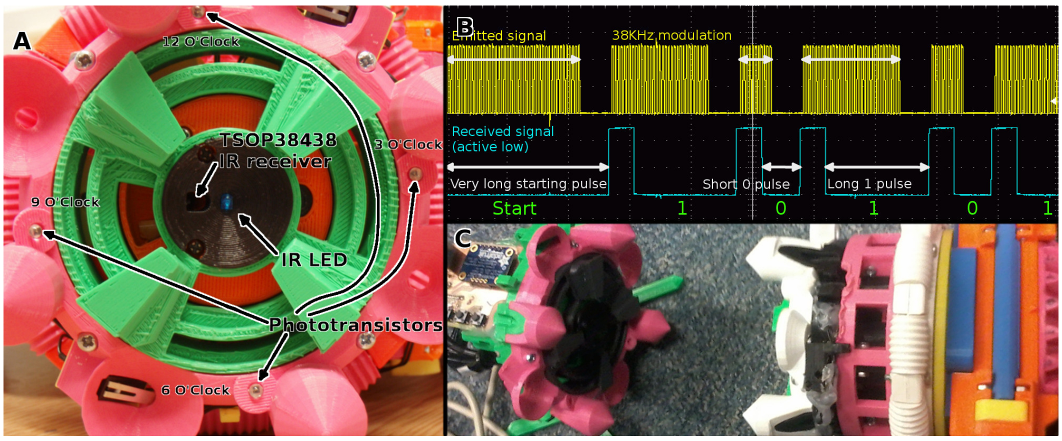


**Fig. 3.6:** Local line-of-sight communication between modules uses IR transmissions with a 38KHz modulated carrier. A single message takes up to 144ms to transmit, as a pulse width encoding method is used, common for infrared communications, messages can be shorter depending on the content. A sample section of the waveform is shown in **3.6B**. There are three possible power levels for transmission allowing a robot to target messages to others within 10 cm, 40 cm or 140 cm. The 10 cm mode is especially useful for Mergeable Nervous Systems [26] inspired communication across a multi-robot structure. The physical positions of the IR communication and guidance components on a docking port face are shown in **3.6A**, with **3.6C** showing a test of the short-ranged communication mode in progress between a test port and the prototype module.

Port to port communications are handled with TSOP38438 [123] 38KHz modulated IR receiver modules and IR LEDs. These provide line-of-sight information transfer both between modules which are connected, and for modules which are passing close to one another or engaged in docking operations. Having line-of-sight and local communication gives the modules a vital capability of being able to recognise which other modules are close to them and pass messages which are related to physical alignments and proximity. These are the types of messages for which the necessary ID numbers of the appropriate receiving module may not be known, hence global communication is impractical for such situations. The port-to-port messaging systems send IR messages as packets with a fixed size of 17 bytes, two of those bytes being reserved for the purposes of a CRC checksum to guard against message corruption. The 17 byte size was chosen after testing in simulation to find which data, and how many bits of precision for continuous values within this data, needed to be exchanged between modules to perform docking. These messages are sent on repeat at a defined interval, this interval can be changed from the Pi to be longer for low-priority messages. A random offset delay is added or subtracted from every such interval to ensure that the timing of any messages which may collide and corrupt one-another cannot remain “in phase" for long. The message sent from a particular port can be changed at any time, or stopped entirely. The CRC checksum guards against corrupted messages being misinterpreted but cannot ensure that every message gets through successfully, and although they have a relatively slow data rate, this port to port messaging is a vital capability for the modules, especially for establishing the identity of nearby modules so other communication methods can be utilised between them. Fig.(3.6) provides further details on this system. A Wi-Fi capability for global communications across the whole group of robots is provided by the Pi Zero W contained within each module, this is used in situations where local communication is not appropriate, such as broadcasting instructions to all robots present.

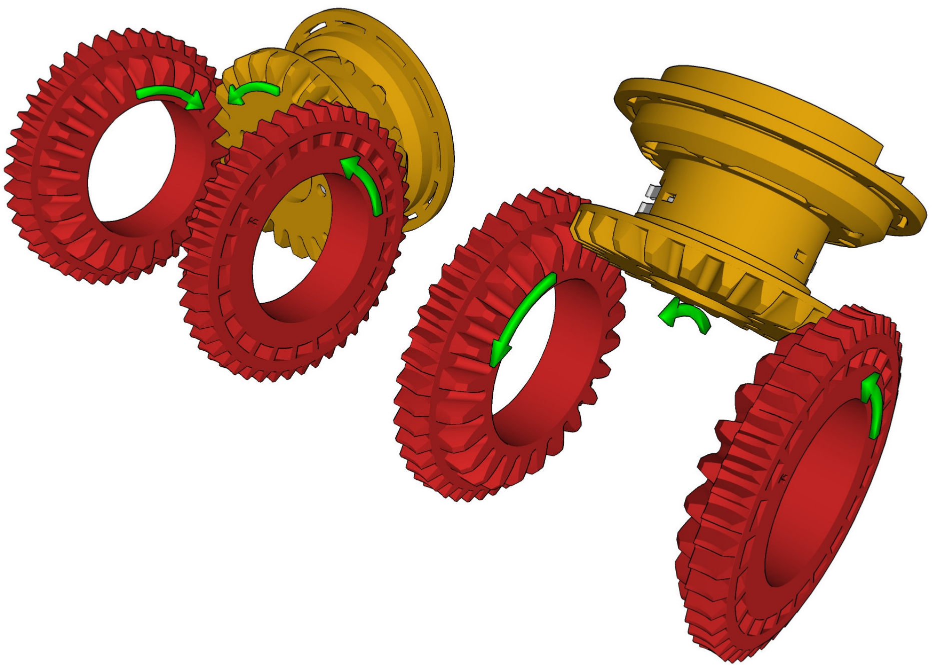


**Fig. 3.7:** Three bevel gears acting together enable a 2DoF rotation, averaged positions of the two driving bevels give the pitch of the output (right), their difference relates to the output roll (left). This 2DoF mechanism provides the kind of dexterity which is greatly desired for multi-module structures [33].

### 3.3.5 Hinge System

The hinge provides the ability for modules to make 3D structures and provides assistance when crossing obstacles and terrain. The hinge was also conceived to enable modules to form structures which can exploit multiple degrees of freedom, such as acting as arms or effectors. 2 degrees of freedom are provided, see Fig(3.7), both a $\pm$ 90° pitch and a $\pm$ 90° roll, both degrees of freedom rotate about the geometric centre of the module. Whilst an idealised hinge in a modular robot would perhaps allow coincident pitch and yaw, as demonstrated in [169], this would limit the maximum reachable angles in such a fashion that the hinge would only be of use in chain-type motion scenarios and could not reach the 90° angles necessary for lattice reconfigurations. Eq.(3.2) describes the way in which the roll and pitch of Port 4 are controlled by the positions of the two driven bevels, Bevel 1 being closer to Port 3, and Bevel 2 on the Port 1 side of the robot, with $\theta_{B1}$ representing Bevel 1's angle away from its initial position at a roll of 0°and pitch 0 °and $\theta_{B2}$ representing Bevel 2's angle.

Due to the torques necessary it is not possible to have all docking ports able to provide rotational degrees of freedom, so a SMORES or HyMod like 4 or 3 DoF system was not possible. Omni-Pi-tent is able to use both degrees of freedom within a single docked connection. So arm like structures can be formed more easily than with SYMBRION,[189] which used a pitch DoF for the forward port and a roll DoF for the side ports. Having pitch and roll DoFs on the same port also enables a chain of two modules to form a structure which can “walk" across a surface of other modules' docking ports to traverse the structure, as SMORES [22] and HyMod [21] have been demonstrated to do. The torques on modules doing this are less than those involved in the two-module-chain lifting requirement outlined in Sec.(3.2).

$$\begin{aligned} P_{itch} &= (\theta_{B1} + \theta_{B2})/2 \\ R_{oll} &= (\theta_{B1} - \theta_{B2})/2 \end{aligned} \tag{3.2}$$

The hinge system has very high torque requirements which made it a significant challenge to produce using low powered DC motors and 3D printed plastic. The initial intention was to be able to lift a chain of two horizontally connected modules, which seemed feasible at the time given the overly-optimistic early predictions of module weight and motor performance, this goal later had to be reduced to lifting a single module, although it has some excess torque capacity beyond that goal meaning that two modules stacked vertically atop one-another could potentially also be lifted providing frictional losses were low. The initial design used worm gears to reduce the speed from the motor shafts and increase the torque, later designs tried a variety of involute spur, cycloidal and planetary gear systems before settling on a multi-stage involute spur design. A SMORES like [35] arrangement of bevel gears combines the motions of multiple motors to produce separately controllable pitch and roll motions. Hinge position

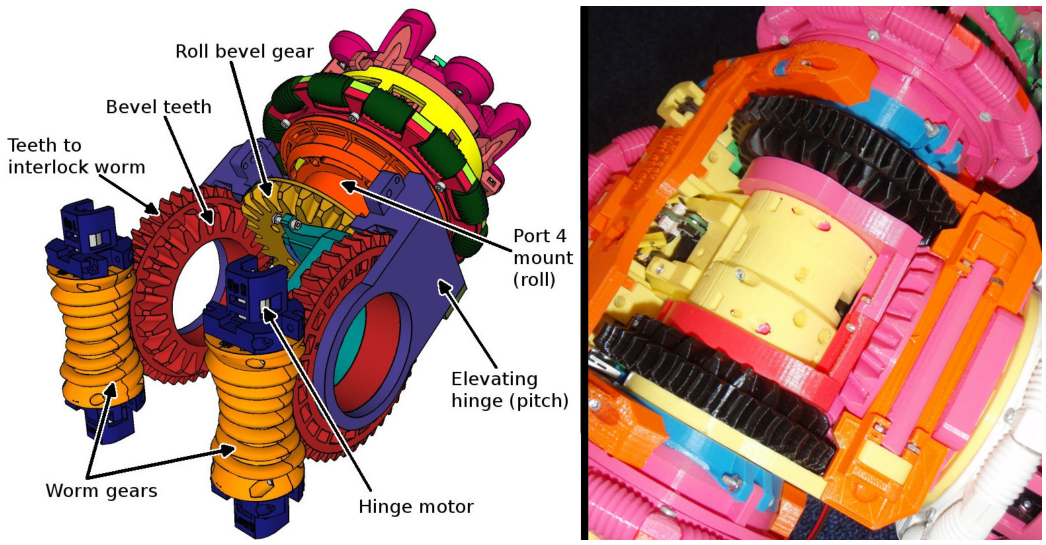


**Fig. 3.8:** The initial design of an Omni-Pi-tent module's hinge, note the worm gear reducers intended to provide increased torque for the mechanism.

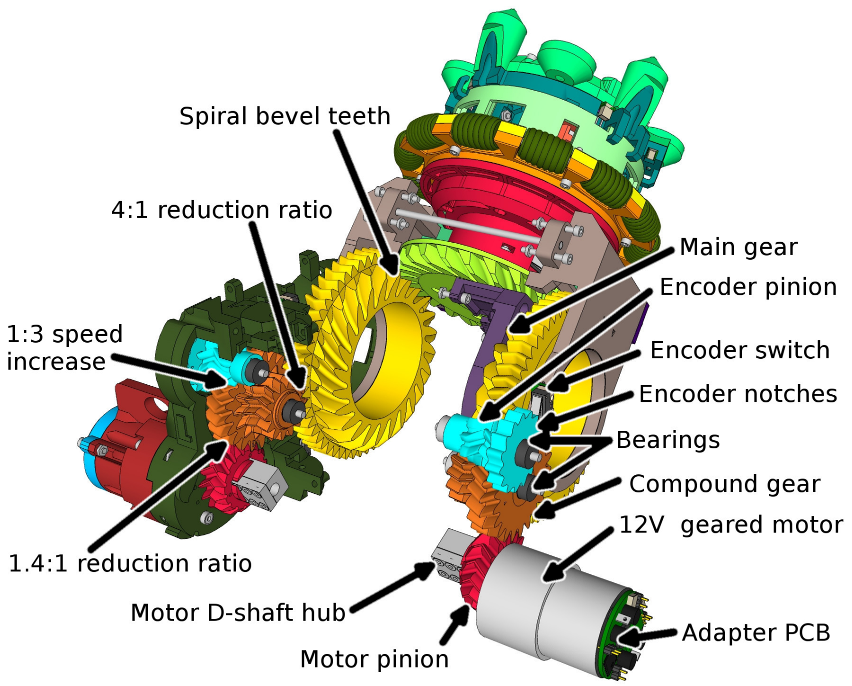


**Fig. 3.9:** The final design of an Omni-Pi-tent module's hinge, note the large new motors and multi-stage involute gear train. The bevel gear teeth have been converted to a spiral form for smoother transfer of load and to spread the tooth-on-tooth pressures over a wider surface area. A speed reduction with a ratio of 1.4 is performed between the motor pinions and the compound gears, a further four-fold reduction happens before the bevel gears. The compound gear also powers an encoder pinion, the encoder pinion is turned at 3 times the rate of the compound gear allowing the encoder notches to give a higher angular resolution on the output.

is tracked via incremental encoding and an ATMEGA328P microcontroller, referred to as the Hinge ATMEGA, which also controls the hinge motors. Fig.(3.8) displays the key features of the hinge design. The later hinge design with the involute gearing and an alternative type of DC motor is shown in Fig.(3.9).

## 3.4 System Architecture

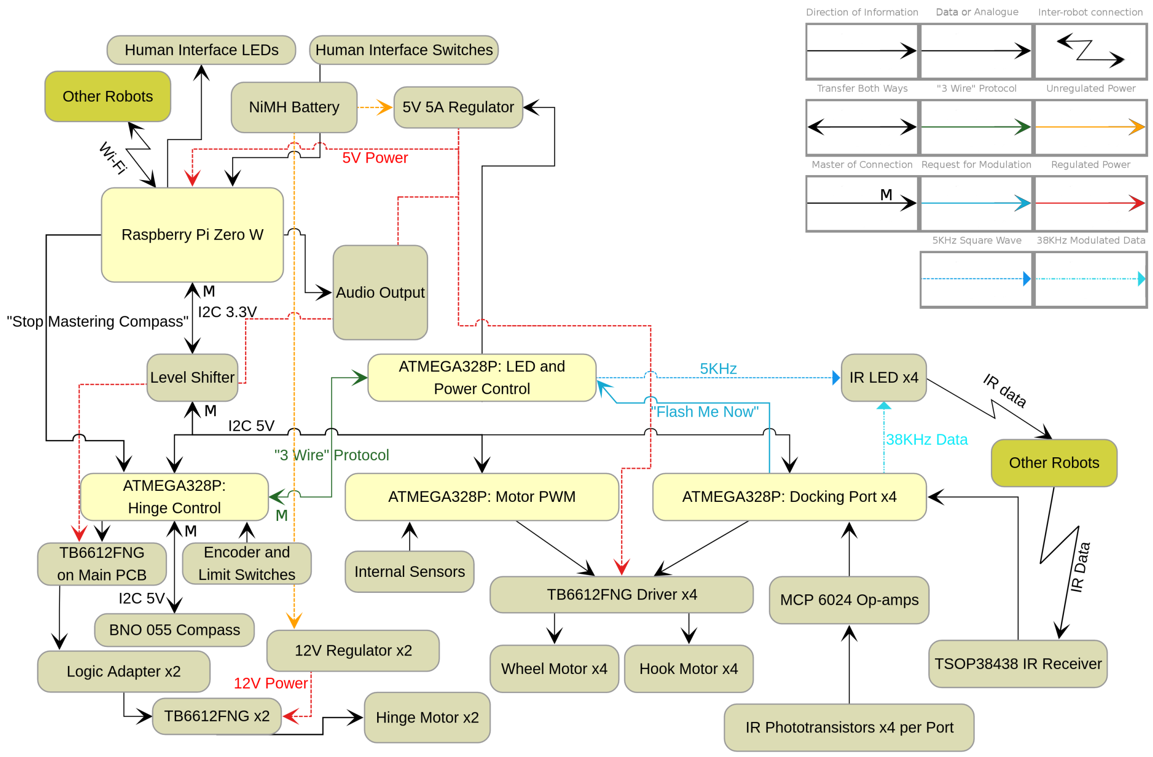


**Fig. 3.10:** A block diagram of the electronic components inside an Omni-Pi-tent module. The Pi with a Raspbian operating system runs the program which the robot is currently executing, it communicates over I2C with ATMEGA328P microcontrollers which handle real-time sensing, actuation and IR communication. During operation, a user's controller code runs on the Raspberry Pi Zero W. Sec.(3.4) explains the diagram in further detail.

The Omni-Pi-tent design uses a central Raspberry Pi Zero W single board computer for high level decision making, plus an array of ATMEGA328P AVR microcontrollers for real-time embedded functionality, Fig.(3.10) illustrates the complete architecture described in this section. The tasks required to operate the module are separated between the microcontrollers, each port being run independently, with further microcontrollers responsible for wheel control and hinge control. Each microcontroller receives commands, of up to 32 bytes length, from the Pi over an I2C bus, the bus operating a rate of around 20 downward commands per microcontroller per second. Each microcontroller also responds to requests from the Pi, also over the I2C bus, for their most recent sensor data information which they relay upward to it.

Each docking port's sensors and actuators are read and controlled by a microcontroller sited on a PCB close to that port, this microcontroller being referred to as the Port X ATMEGA (X being the port number from 1 to 4). The Port X ATMEGA controls Port X's hooks as well using its inbuilt ADC to read analogue waveforms in real-time from the docking guidance phototransistors to give unitless strength readings of the illumination levels. This microcontroller decides whether the phototransistors are to operate in passive guidance or active proximity modes, and also enables rapid switching between modes so that I2C data being relayed to the Pi can provide both recent guidance and proximity levels. Port-to-port 38KHz transmission and reception is controlled from each port microcontroller, this being a task for which accurate real-time operation is vital.

The hinge is controlled by a further microcontroller, the Hinge ATMEGA. This microcontroller provides direction and PWM signals for the hinge motors, and reads an array of microswitches which act as limit sensors and encoder channels. Again real-time operation is vital here to keep track of position encoding pulses and ensure that the hinge stops when required, avoiding the risk of over-travel and the serious damage this could do to the hardware given the torques involved. As a separate task, this microcontroller also acts as an intermediary between the compass chip and the Pi, aided by a “Stop Mastering Compass" negotiation line. On the first prototype module this microcontroller directly controls a pair of TB6612FNG H-bridges which run the 5V DC motors powering that module's hinge. In later design upgrades, made after production of the central PCBs, the outputs of the 5V H-bridges become logic inputs to adapter PCBs which operate the upgraded hinge motors from a 12V power source.

Wheel motors are controlled from another microcontroller, the Motors ATMEGA. This chip has a much simpler task than the Port or Hinge ATMEGAs, just controlling the directions of the 4 wheel motors and varying their speed via PWM. With this undemanding, but again real-time and hence unsuitable for the Pi, task, the Motors ATMEGA also serves to monitor internal temperature and voltage sensors which ensure safe operation of the modules.

The final microcontroller in the system is the LEDs ATMEGA, this produces, on demand from a “Flash me Now" binary signal from the relevant ports, modulated signals required for the guidance and proximity system. It was chosen to use a microcontroller here, in favour of a simpler square wave source, as it also acts to control the module's power and battery system. It is not on the I2C bus and only communicates with the Pi over a number of parallel wires for binary status signals. The tasks separated on to this module ensure that the module can

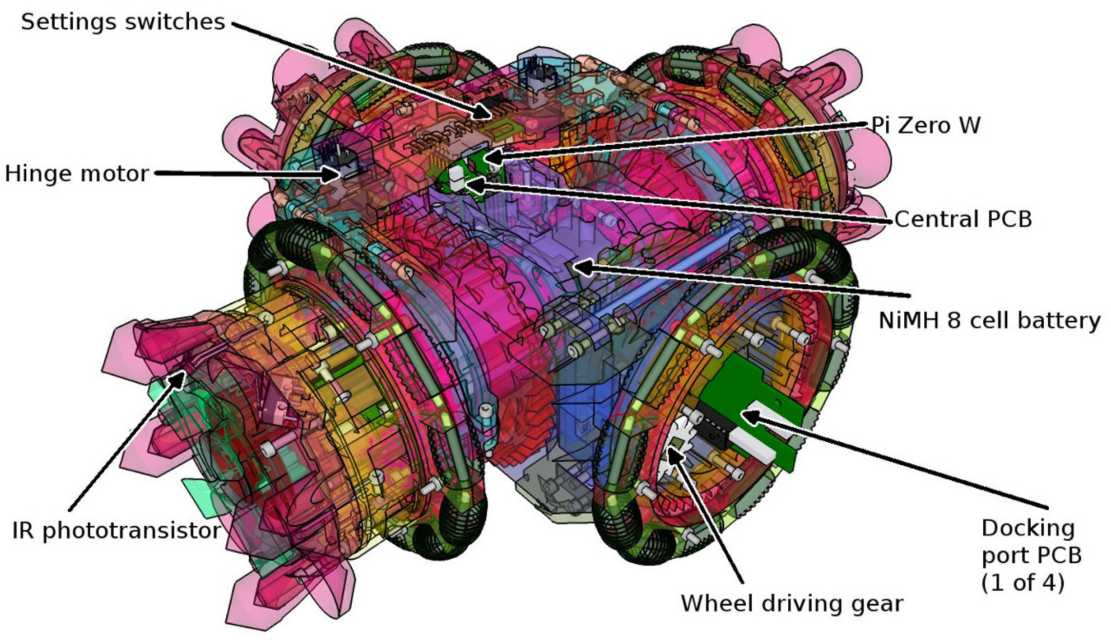


**Fig. 3.11:** A diagram displaying the locations of the various PCBs and other components within the first Omni-Pi-tent module.

power up and down as necessary even if everything else were to encounter software crashes. The fact that this microcontroller is never reset during robot operation also makes it the ideal place to hold persistent hinge position data across restart events, this data is handled through a custom 3 wire protocol, based on principles similar to SPI, further described in Sec.(5.12).

All microcontrollers, as well as the Pi, are attached to an array of PCBs within each module.

Using elements of the BCM2835 library [1], a C library was written for use on the Pi which abstracts away the details of I2C communication with the real-time ATMEGA chips, actuators and sensors. Timing and expected data formats are automatically handled freeing the user to concentrate on high-level multi-robot behaviours and multi-robot geometry. C programs written with this library can either be compiled and preloaded onto the Pi's MicroSD card to begin execution after the Omni-Pi-tent module is powered on, or can be transferred over ssh to the Pi. The 3D printed body provides easy access to the MicroSD slot of the Pi to enable simple changing of cards. Status and debugging messages are generated by a speech synthesizer and are then output through a speaker. When in use, the low-level AVR code on the ATMEGA chips does not need changing once written to the relevant microcontrollers, hence ease of physical access to these, once the robot was constructed, was not a design priority.

## 3.5 Constructing the Omni-Pi-tent Prototype

The Omni-Pi-tent prototype was designed to contain 7 circuit boards, illustrated in Fig.(3.11), which are restricted to two layers for ease of fabrication, positioned in a 3D printed mechanical body. The first prototype's printing was all performed with a Robox FDM printer with a 0.2mm layer height. The motors and other electronic components are low cost off-the-shelf types. An NiMH battery fitted within the axis of the 2 DoF hinge supplies power to a central circuit board which hosts the Pi Zero W and hinge control microcontroller, this board connects to docking port PCBs which handle infrared strength measurement and IR communication. All components were held together with machine screws and nuts for ease of access to replace and upgrade 3D printed components. Around 200 screws are used per module, assembly was somewhat simplified by using screws all of the same diameter (M2.5) except where other sizes were required to mount purchased components such as microswitches. In most locations hexagonal sinks in the printed parts provide a convenient mount for the nuts. The first completed robot is shown in Fig.(3.12), Fig.(3.13) and Fig.(3.14).

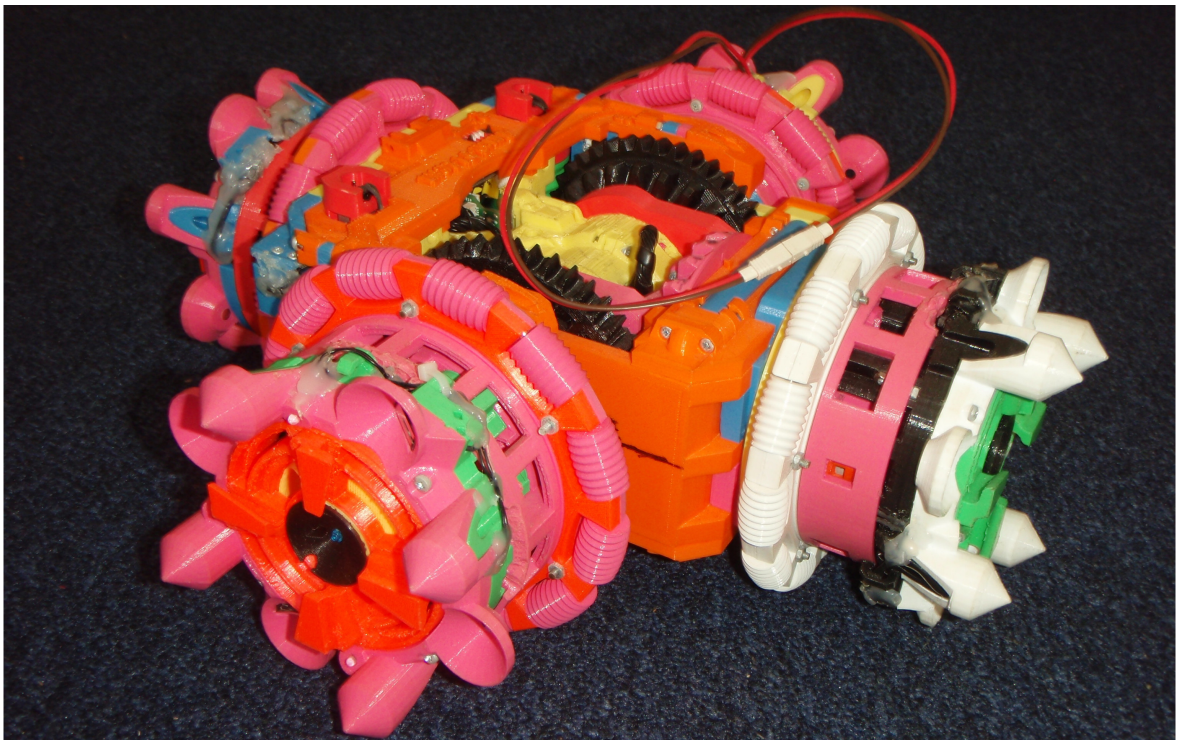


**Fig. 3.12:** The first prototype module, Port 4 has the white docking spikes and omniwheel, Port 1 is to the lower left in this image.

Later modifications to the robot design are detailed in Chapter 5, these included modifications made to improve the hinge mechanism (Sec.(5.16)), refine actuator performance (Sec.(5.4.1)), and simplify the construction of modules(Sec.(5.15)). The improved design is shown in Fig.(3.15), with Fig.(3.16) showing the locations of some of the extra added features.

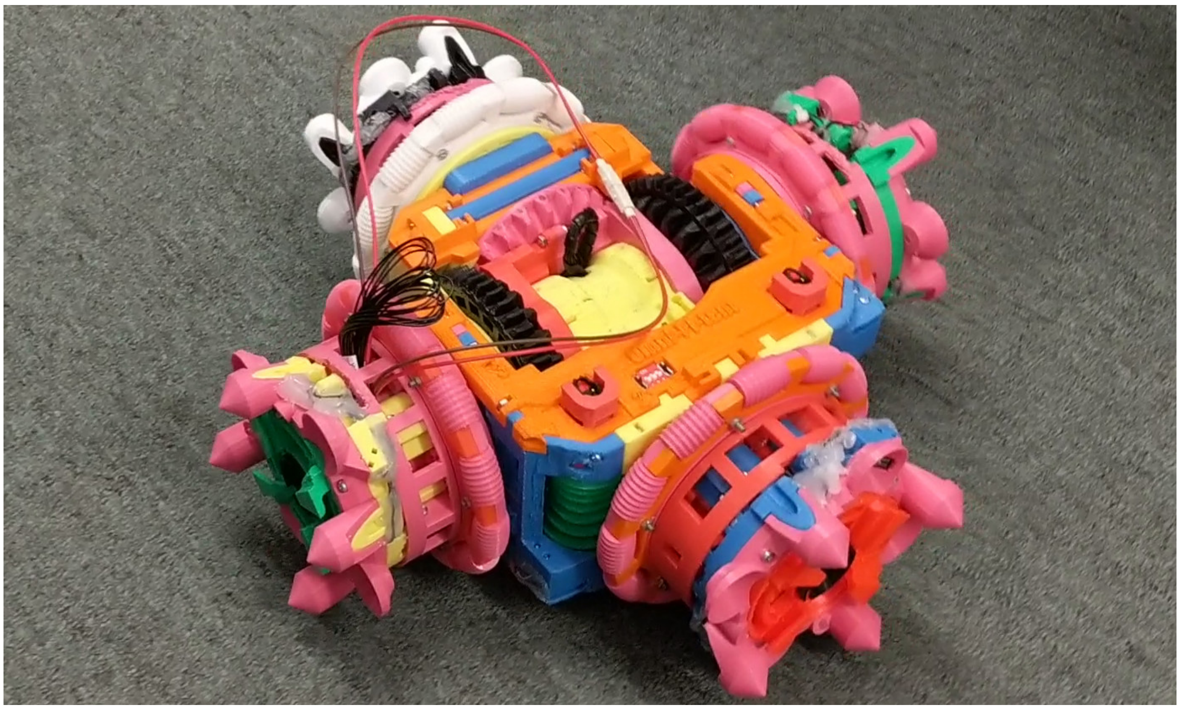


**Fig. 3.13:** The first prototype module, viewed from above and behind. Colours of the component parts reflect which material was in the printer at the time an element was designed or revised, this has a side-effect of making the separate elements of the design and joins between them easily identifiable.

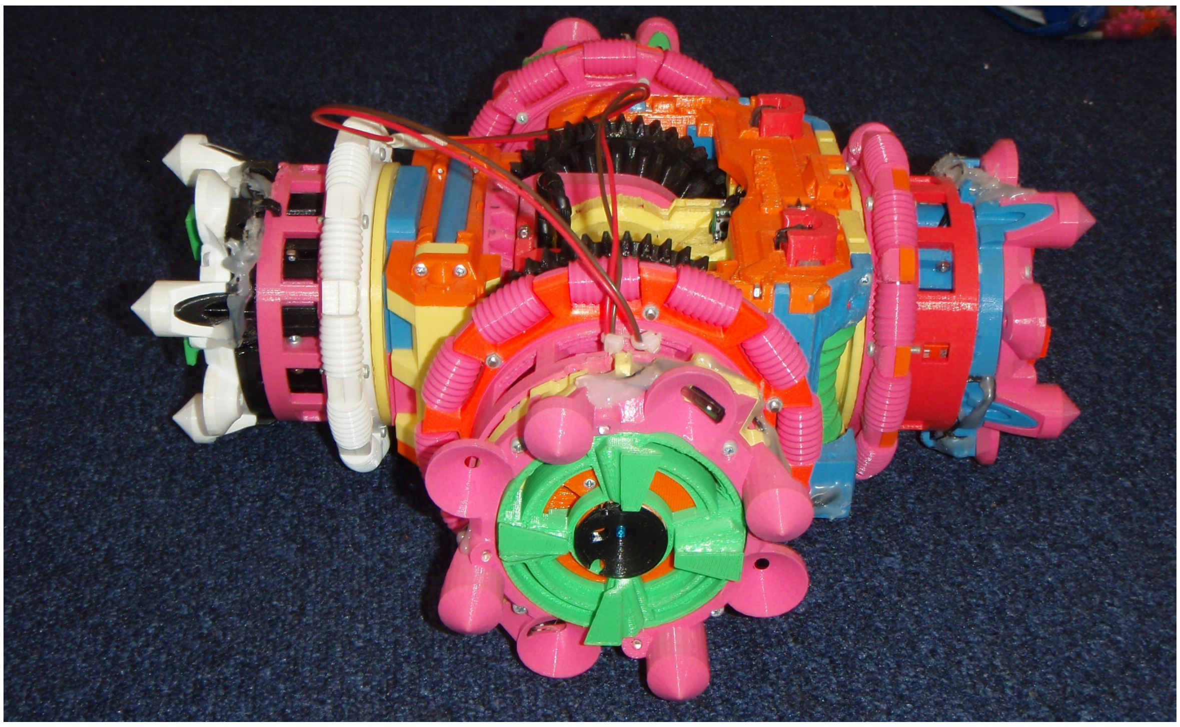


**Fig. 3.14:** The first prototype viewed from the Port 3 direction.

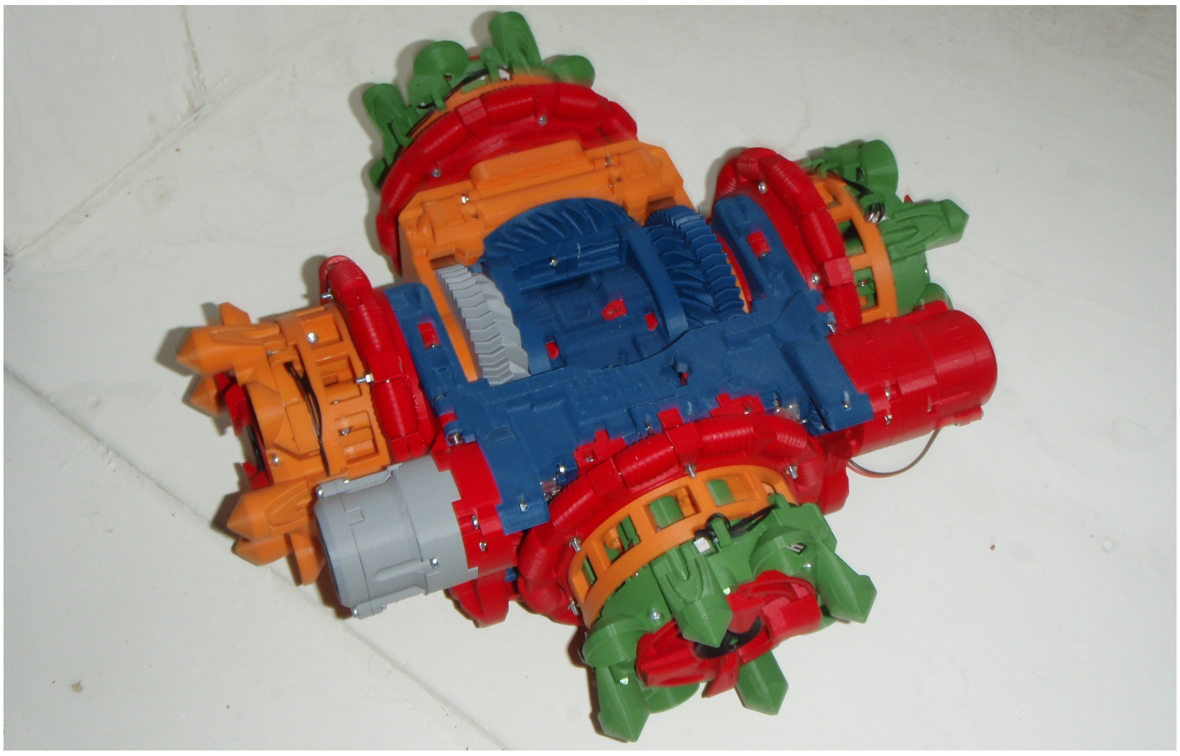


**Fig. 3.15:** An Omni-Pi-tent module with the design upgrades applied, shown from the underside to best display the regions where modifications were made.

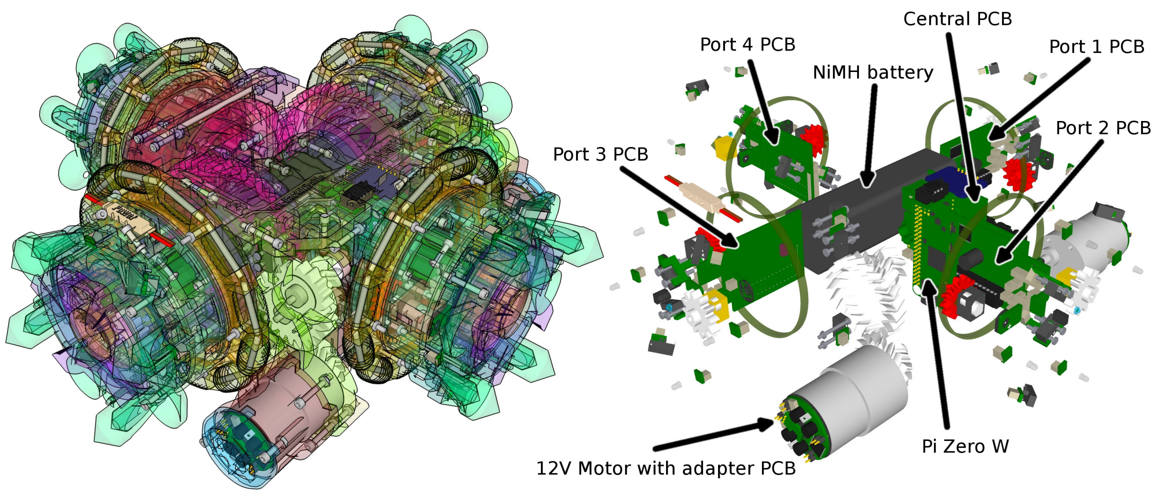


**Fig. 3.16:** A diagram displaying the locations of the various PCBs and other components within an upgraded Omni-Pi-tent module, note the new adapter boards on the backs of the 12V hinge motors.

## 3.6 Summary

This chapter has described the concept of the physical Omni-Pi-tent modular robot platform and provided a high-level overview of the design. The need for key features such as: genderless docking, joint actuators, a full suite of docking sensors and an omnidirectional drive, has been explained, and the implementation of these features described. Concepts for docking and multi-module activities have been introduced. Further information on the Omni-Pi-tent platform is also available at `https://www.york.ac.uk/robot-lab/omr/` with some project updates posted at `http://omni-pi-tent.blogspot.com/`.

The next chapter introduces the simulation environment used in this thesis's experiments and details work undertaken to reduce the reality gap between simulation and hardware. The findings of those simulations then informed some of the practicalities of hardware development, which are discussed as part of Chapter 5.

# Chapter 4

# Simulations, Models and the Reality Gap

This chapter presents a description of the simulation environment, produced within V-REP [28], used for the work throughout this thesis. This simulation environment acts as a companion to the Omni-Pi-tent hardware platform, introduced in Chapter 3. The purposes of simulation in robotics are introduced as is the V-REP simulator. The modelled representation of Omni-Pi-tent is described in detail and the rationale behind decisions taken is explained, features of the model are listed and approximations made are discussed. The chapter also covers the designs of simulated experimental environments, and details experiments performed to reduce the “reality gap"[24] by producing refined simulations based on the real performance of Omni-Pi-tent sensors and actuators.

## 4.1 Robotics Simulation

Running simulations enables concepts for programs, algorithms and controllers to be rapidly tested and adjusted at a rate which would be infeasible with real hardware. More physically accurate simulators can also help to verify the plausibility of robot hardware enabling ideas to be tested, and infeasible ones to be eliminated before effort is wasted on constructing them as real robots. Other useful capabilities enabled by simulators are for real robots to be linked to a simulated copy of themselves so they can trial actions in simulation before risking them with the real robots, also known as in-the-loop simulation [72].

In swarm robotics and modular robotics there is a practice of using simulators to run a large number of simulations with randomly varied initial starting conditions to get averages, distributions and other statistical measures as to the performance of a robot group and its controllers, often allowing much larger groups of robots to be considered than are available with real hardware. Many popular robot platforms have either custom simulators optimised for their hardware or easily available plugins for common multi-platform robot simulators [72].

By having a simulated model of the platform available, it was possible to perform a significant amount of programming and controller testing work in the period before hardware development was complete, and even after multiple modules of real hardware were produced, the simulation proved useful for gathering large amounts of data for statistical use while the hardware experiments provided real-world proof of the concepts and helped keep the simulations anchored to reality. Due to the user friendliness of its interface, the thorough documentation and ease of implementing customised robot platforms within it, V-REP [28] was chosen as the simulation environment to be used for this work.

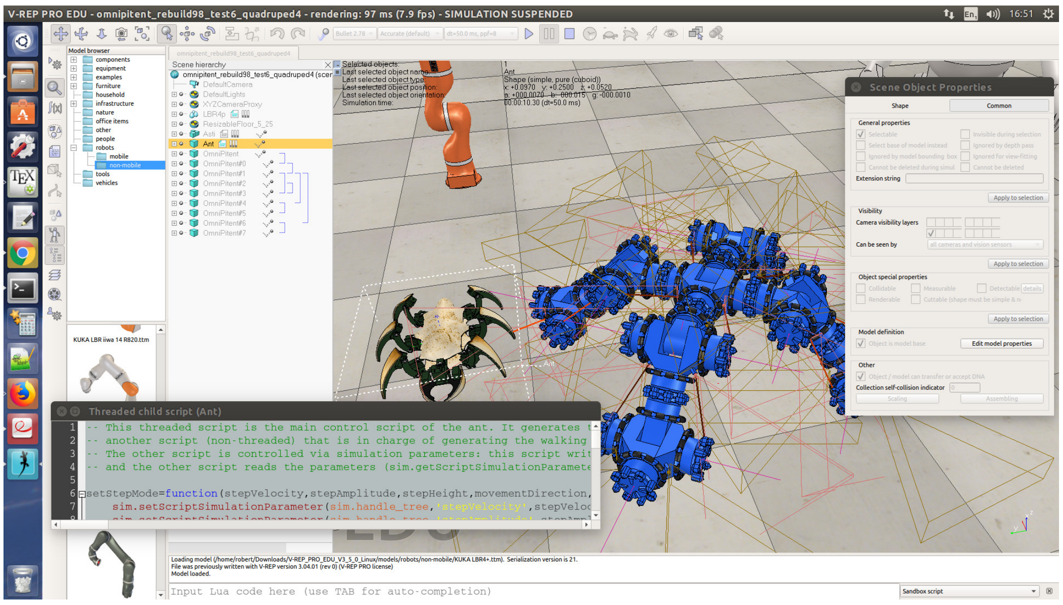
We begin with a discussion of the reality gap before describing the simulation software in use, then move to discussing the simulated model of the hardware. A later section then describes some of the most relevant commands available for use in Lua[23] controllers within the simulator.

## 4.2 The Reality Gap

Robotics simulators are usually formed from a physics engine software package to handle the dynamics of moving parts [127], plus extra components which run the simulated robot controllers and process actions such as communication which the physics engine usually does not handle. Inaccuracies in physics engines involving difficult-to-model effects, rounding errors and floating point maths, or simply the difference between the step based computation in simulators as compared to the non-zero time of execution for a real robot's controller, can all produce inconsistencies between the simulations and the real world, referred to as "the reality gap". Although particularly noted in evolutionary robotics [128] [129] [130], where evolved controllers find exploiting flaws in the physics engine a much more attractive form of optimisation than generating real-world applicable behaviours, the reality gap can be found in any robotic discipline [134] [132] [131] where simulation is used to develop robotic controllers before later being transferred to real robots. Simulators, especially given that to run fast enough to be of any use they can never model all of the effects that a real-world interaction will involve, are unlikely to ever be quite good enough to eliminate reality gap effects entirely. Some researchers [131] have proposed that standardised tests could be used to improve simulation accuracy in this way, but what those tests are will inevitably vary significantly with the physical design and intended tasks of the robot being simulated. Others [128] [133] have developed sophisticated toolchains so that controllers can be developed in parallel across both simulations and real-world robots, focusing on ensuring controllers make use of the subset of features within simulations which do accurately represent real-world effects. Given that the Dynamic Self-repair project of this thesis entailed both the design of hardware and of controller software, running as parallel strands, reduction of the reality gap was considered most practical to achieve for this project by manually modifying the simulation environment based on test results from real-world hardware subsystems as they became available.

## 4.3 V-REP

V-REP (Virtual Robot Experimentation Platform) [28] is a robot simulator with a vast range of capabilities, most of which can be further extended through plugins and other user-created features. Version 3.5 was the latest iteration of the program available when the work in the Dynamic Self-repair project was begun, but simulation scene files (.ttt) and model files (.ttm) are generally forward and backward compatible between V-REP versions. V-REP is operating system cross-compatible and can be run on Linux, Windows and Mac. Fig.(4.1) shows the user interface in a scenario where several early iteration models of Omni-Pi-tent units have formed a quadruped while several other robot models, supplied in V-REP as examples, perform various actions around them.


V-REP PRO EDU - omnipitent_rebuild98_test6_quadruped4 - rendering: 97 ms (7.9 fps) - SIMULATION SUSPENDED

Threaded child script (Ant)

```
-- This threaded script is the main control script of the ant. It generates t
-- another script (non-threaded) that is in charge of generating the walking
-- The other script is controlled via simulation parameters: this script writ
-- and the other script reads the parameters (sim.getScriptSimulationParamete

setStepMode=function(stepVelocity,stepAmplitude,stepHeight,movementDirection,
    sim.setScriptSimulationParameter(sim.handle_tree,'stepVelocity',stepVeloc
```

```
Loading model (/home/robert/Downloads/V-REP_PRO_EDU_V3_5_0_Linux/models/robots/non-mobile/KUKA LBR4+.ttm).  Serialization version is 21.
File was previously written with V-REP version 3.04.01 (rev 0) (V-REP PRO license)
Model loaded.
```

**Fig. 4.1:** The V-REP graphical user interface provides many of the tools for constructing and then running simulations, the properties of objects in the scene and simulation parameters can be altered in pop-up windows and programmed scripts can be written to be executed by objects.

Computationally, V-REP uses a main script for the scene which calls child scripts, each of which is generally associated with a robot, a robot part or an environmental item in the scene, as such it can be run in a distributed way to easily simulate multi-robot systems [64]. When objects are copied, the new object names are suffixed by `#n` where `n` is an integer from 0 upwards, any child scripts running on the object will automatically act on objects with the relevant suffix unless explicitly coded to do otherwise.

Each simulation step is made up from several smaller steps, some for the reading of sensors and others for actuation and motion [65], actuation is performed before sensing such that the sensors will give readings appropriate to the positions they find themselves in after actuation is complete. Non-threaded child scripts run once in each sub-step of a simulation, threaded child scripts are activated at a specific time then allowed to continue in the background while the simulation runs, this latter type of script was not needed in the simulation of Omni-Pi-tent modules. Any non-threaded child script can contain sections of controller code which will be run only during certain types of sub-step, for example a certain piece of code may only run when the simulation is first started, another may only run during actuation or sensing steps and some may only run as cleanup routines when a simulation is stopped.

As well as the internal interface one may use: plugins, remote APIs or server based methods to control child scripts. This enables the use of any programming language for code as well as the default Lua [23]. This is a useful feature, being able to have a common programming language between simulation and reality is highly desirable in robotics research fields, as indicated by Ivaldi's survey [61]. The computational power of the machine on which V-REP is being run provides the only known limit on how many child scripts can be used.

Several physics engines are available with Bullet 2.78 being the default, these enable realistic dynamical situations to be simulated and allow collisions to be modelled. Because of the computational loads associated with processing complex geometry, a well-designed model for use in V-REP simulations will typically have two sets of geometric data, a simplified low-polygon-count version for the physics engine to handle and a high-polygon-count detailed version to be shown visually and detected by vision and proximity sensors, Fig.(4.2) displays this comparison for several robot models, Fig.(4.3) shows the model of the Omni-Pi-tent module in more detail.

By default, there are three sensor types built in. Proximity sensors can monitor a region for any visible item which enters the volume and can report either the presence or lack of objects or give numerical readings for the minimum distance to a point on any intruding objects. Vision sensors can act as cameras, their views can be fed into basic inbuilt image analysis algorithms, for example edge detection, or they can be reduced to $1{\times}1$ pixels and used to measure colours or light levels. Force sensors can measure the forces between parts and also allow for items to be joined together. Other forms of sensor can be made by combining the properties of these or by reading other values out from the simulation within scripts.

The main disadvantage of V-REP is that, by containing so many advanced features, especially the physics engine(s), it is unable to run at the extreme speeds achievable with simulators which ignore physics. Increases in the speed of V-REP simulations are limited by the performance of a single core of the CPU, as V-REP does not use multiple threads. Using a single core of a node (1 core of a 2.0GHz Intel Xeon 6138) on the university's Viking cluster

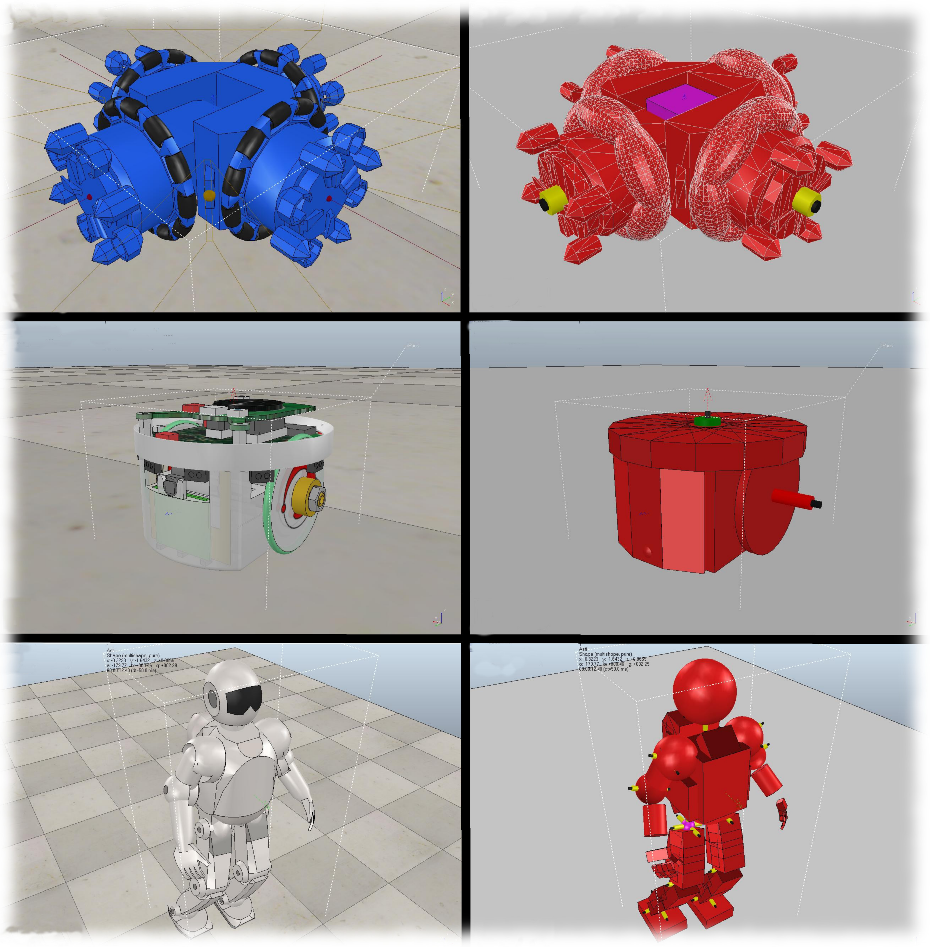


**Fig. 4.2:** In the left-hand column: visual models for V-REP, note the realistic shapes and complex curves, including concave surfaces. In the right-hand column: underlying models for the physics engine to handle, note the simplified shapes and care taken to ensure that all individual objects are convex. From the top the robots depicted are: an Omni-Pi-tent module(R.H.Peck), an e-puck (EPFL [85]), and Asti (L.Randall [86]).

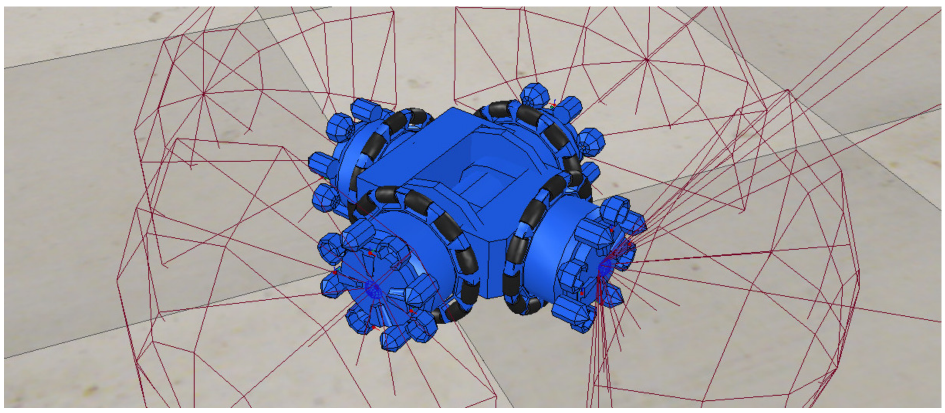


**Fig. 4.3:** An Omni-Pi-tent module as modelled for the V-REP simulation

[84] 30 modules could be simulated at a speed where 12 minutes of simulation time took 12 hours and 20 minutes of computing time to complete. The speed of simulation is largely

independent of the complexity of the controllers running on the simulated robots, instead being dominated by the effects of the processor-hungry physics engine and the number and detail level of any proximity or vision sensors in use. Tests of V-REP's performance against less physically accurate simulators show speed differences of over 200 in extreme scenarios [66] [67] with complex environments containing very large numbers of robots. The complexity of Omni-Pi-tent modules compared to the simpler robots used in [66] and [67] would likely make that measured speed difference even larger.

## 4.4 Simulating the Omni-Pi-tent Actuators

The first attempt to simulate the hardware of an Omni-Pi-tent unit in V-REP used relatively high-polygon-count meshes for all components including the physics engine's components as well as their visual equivalents. Whilst this was able to run it was highly jerky in its motions, could not drive along the correct vectors due to unusual drag forces in the simulation and was computationally slow and inefficient, being unable to run anywhere near to real-time speed even with only one robot in simulation. These physics issues were fixed in a subsequent version of the simulation model, before any further development was done.

### 4.4.1 Mass and Moments of Inertia in V-REP

The Bullet physics engine, and infact any other physics engine, contains inaccuracies. Of particular note in V-REP is the difficulty involved in dynamically simulating kinematic chains when the objects along them have low masses, or low moments of inertia, or have mass ratios of 10 or greater between subsequent “links" in such chains [70]. Initial attempts were made to simulate the hardware using accurate predicted masses, moments of inertia (as derived from geometry and some density approximations) and torques (derived from motor performance datasheets and ratios of planned 3D printed gears) however these suffered from difficulties where meshes would begin to randomly float off from the axes of joints they were attached to. These simulations also suffered issues where robots docked in a chain and trying to lift those infront of them via the main hinge would be unable to lift their companions by more than a fraction of a degree, even when ample torque was provided by the motorised joint. This was because the physics engine was unable to simulate the transmission of forces along the chain of low-mass, low-moment-of-inertia modules. Due to these aspects of the physics engine, arising from matters such as the extent to which the smallest value in a decimal can be handled before being rounded, the mass of the Omni-Pi-tent simulation model had to be set far in excess of the real value, the moments of inertia had to be significantly increased and maximum torques available from the motorised joints had to be increased accordingly. With this done the objects behaved in a physically sensible way. Although the simulated models no longer used accurate masses and torques they were still appropriate for use in simulating

Omni-Pi-tent group behaviour. In the simulations used throughout this thesis, robots interact primarily with other robots and with some immovable or otherwise simplified environmental obstacles, as simulated model robots do not interact with simulated models of specific and accurately formed real-world objects (for example: tools, doors, animals, people, other types of robots...) these mass and torque inaccuracies do not have major effects on simulation results because everything in the simulation scenes has masses and torques inflated by the same factor.

### 4.4.2 Omni-wheels

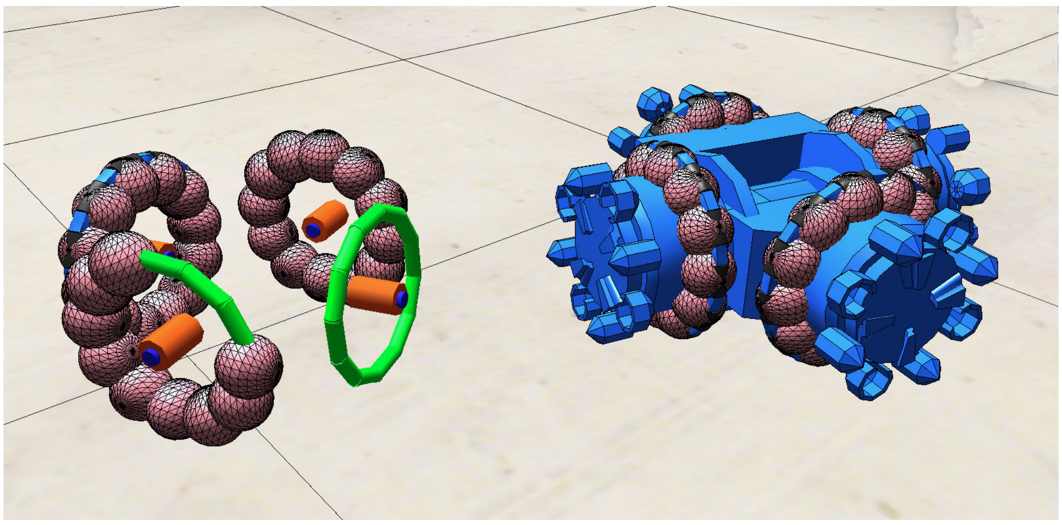


**Fig. 4.4:** The dynamic simulation of the omni-wheels on an Omni-Pi-tent unit. In orange and dark blue the motorised joints powering the wheels, the non-respondable mesh object acting to connect the motorised joint to the free joints is not shown but is centred on the orange joints. The free joints are shown in green. In pink the rollers are shown. Some of the pink rollers, have been hidden to make the joints at their centres more visible. On the right the overlaps between the wheels and the visual model of Omni-Pi-tent are shown.

A crucial feature of the Omni-Pi-tent platform are the omni-wheels which enable it to drive along any specified vector direction, turn on the spot or move in ways which combine these forms of motion in varying proportions. On the physical hardware each omni-wheel has 12 rollers around the rim supported on 3 mm diameter metal axles. V-REP has many ways to produce omni-wheel simulations and the one which is most widely recommended online [69] is made as follows. First a parent revolute joint is created and motorised to provide drive motion, in the hierarchy below this joint is a dynamic but non-respondable object, this non-respondable object then has a joint, this joint being free to rotate and aligned parallel to the floor but $90^\circ$ away from the motorised joint, as a child, lastly the free joint has a respondable dynamic sphere as a child. A child script is attached to the motorised joint which acts to regularly reset the orientation of the free joint so gimbal lock does not occur. Unfortunately such a method is not appropriate for the Omni-Pi-tent platform. With the

Omni-Pi-tent platform, multiple robots must be able to come close together to dock, when docked to each others' port the robots are close enough together that, if a sphere was placed at the centre of each omni-wheel, it would intersect with the spheres centred on the wheels of other robots. Because the spheres must be both dynamic and respondable, this would result in highly problematic collisions of a kind which do not occur in the real-world. There is no practical method to use dynamic masks to solve such an issue when multiple robots interact as only 8 mask channels are available and each robot already has 4 wheels.

Therefore, the omni-wheels on Omni-Pi-tent have instead been modelled in a much more detailed fashion. A central motorised joint turns a wheel (dynamic, non-respondable) with 12 free joints around its rim. Each free joint then has a roller upon it, on the real hardware the rollers are textured "cylinders" with their waists widened so as to match the overall curve of the rim of circle centred about the omni-wheel's centre. To save on the computation requirements the rollers in simulation are instead modelled as simple spheres. As the roller "cylinders" are all children within a model of an Omni-Pi-tent module further computational efficiencies can be made by using local dynamic masks to let them intersect with one another without collision while still generating collision responses with environmental objects or parts of *other* Omni-Pi-tent modules. This allowance for intersection means 12 spheres can produce a circular rim sufficiently accurate to drive "forwards" while allowing "sideways" rolling. This arrangement is shown in Fig.(4.4). It was found necessary to give large simulated masses and moments of inertia to these components, otherwise unphysical effects such as wheels refusing to turn and rollers floating out of place occured due to the rounding of small decimal points in the physics engine, as explained in the previous section.

In the multi-robot simulations the controller code for modules controls the wheels via an interface function which rotates the wheels at a fraction of their maximum speed, rather than simply at a specified angular speed, this maximum speed was found from hardware tests to be 68 ° per second. Later design refinements of the hardware raised this maximum value of the wheel speed to 90 °/s. This helps keep the simulations anchored to reality and avoids any situations where an unrealistically high wheel speed could be commanded.

### 4.4.3 Docking Ports

The docking ports were initially modelled using a crude version of the actual hook, spike and pit geometry, described in Sec.(3.3.1).When this geometry, especially the pits for the spikes to enter, was broken down from "random meshes" into groupings of "convex meshes" a realistic docking action could be simulated by rotating the hooks of one port to $45°$ then approaching the target port, and finally returning the hooks' position to $0°$. Fig.(4.5) shows how this appeared, from a simulated dynamics perspective, once docked. However, simulating this form of docking was very computationally intensive as forces acting on small and complex shapes had to be calculated in every simulation step. These also meant that, when forces were transmitted across larger structures of robots, for example during group turning or lifting actions, V-REP had to model it as a chain of objects, each free to move but connected by the interlocking geometries of the docking ports. Such simulation of chain of items docked in this way gave many unrealistic results such as a failure of any items to move even when colossal forces were applied. Hence, a dummy based docking method was chosen instead.

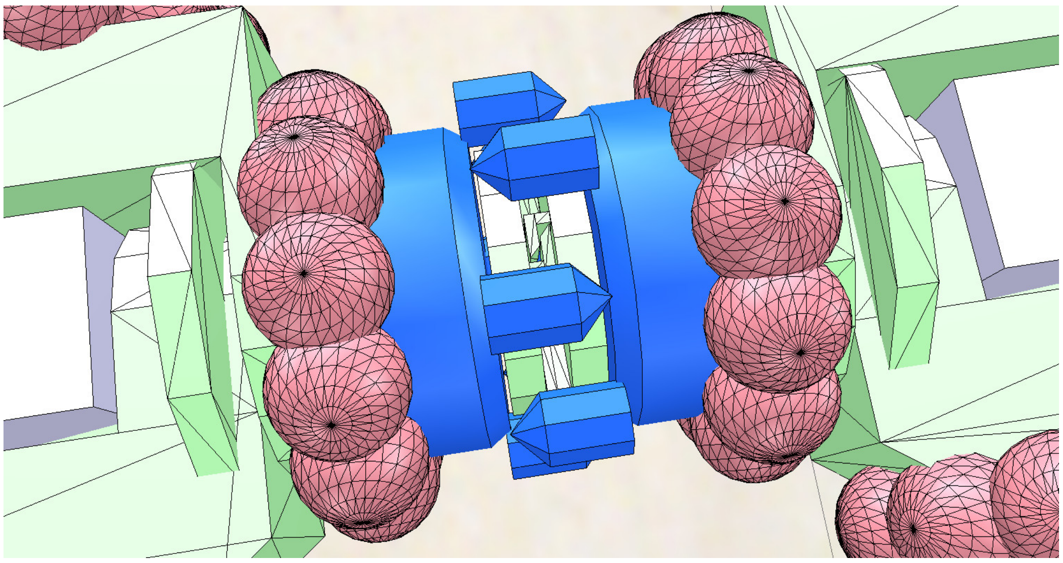


**Fig. 4.5:** Two Omni-Pi-tent docking ports connected together using dynamic simulation of the meshes. In pink at the side the dynamic structure of some of the omni-wheels can be seen, in blue the spike and hook structures, each composed of multiple convex meshes assembled together to make the complex shape. In green are the docking hook meshes, which can be rotated using a joint.

### 4.4.4 Dummy Docking

In V-REP separate models can be joined together by a rigid link when dummies are fitted to each object and linked with a dynamics overlap constraint. When this is done the physics engine stops considering the forces of the two objects upon one another and performs later calculations treating the parent objects of the dummies as if they are a single solid object. Dummies can be linked and unlinked over the course of a simulation so docking and undocking can be handled by function calls in the child scripts running on each robot. A dummy has been fitted to each docking port on the module model, when the module wishes to dock to others `sim.setLinkDummy(MyDummy,HisDummy)` is called and the dummies, where `MyDummy` and `HisDummy` are references to the dummy objects involved, link together. It is vital that, before this function is called the two dummies are colocated, or very close, in space as when linking happens both dummies will jump from their current locations to match each other, if this jump occurs over a significant distance then unrealistically large acceleration jerks will occur. The dummies must also be coaligned using `Vector=sim.getObjectOrientation(HisDummy,-1)` to find the target dummy's orientation and then aligning the approaching dummy to the target one is done by making use of the function `sim.setObjectOrientation(MyDummy,-1,Vector)`. Because dummies coalign when linking, if they are misaligned beforehand their parent objects will experience a rapid angular motion if those functions are not used to ensure coalignment.

Dummy docking is computationally efficient to simulate and while the programmed actions involved in dummy docking require the use of global information unlikely to be available in the real world, such information is only used by the actuator details of the program, not by the logical and behaviour elements. In the real world, of course, all the dummy interactions are simply replaced by commands to turn the docking hooks to the open and locked positions, and the sensing of contact or lack thereof on the detection switches within the upper and lower spike accepting pits, so no such global orientation and location information of this level of detail is required. In the finished version of the simulation model the docking spikes and hooks are shown, but for visualisation and mass and moments of inertia only, they do not experience respondable collision interactions either locally (with other objects in the same module model) or globally (with objects outside of the model).

### 4.4.5 Hinge

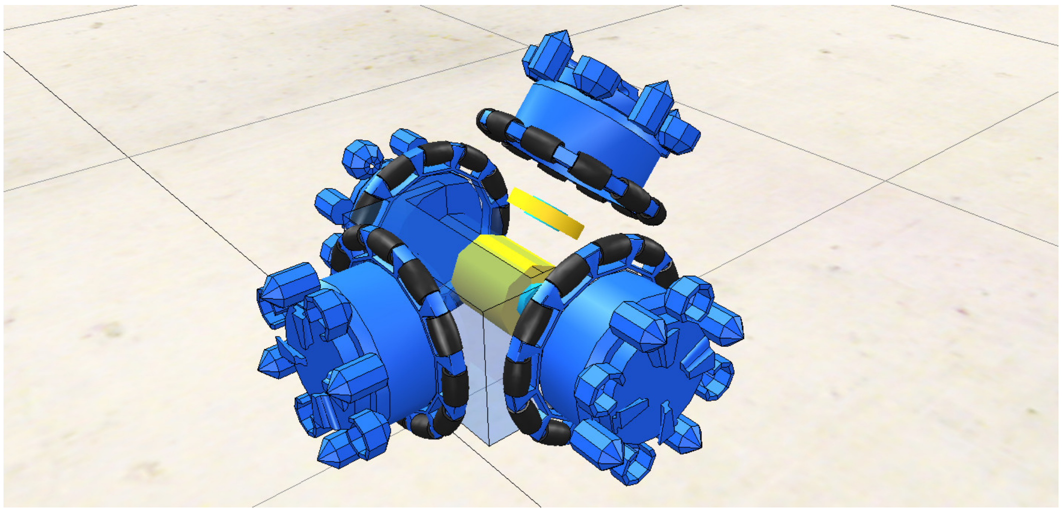


**Fig. 4.6:** The hinge joints in an early simulation model of the Omni-Pi-tent platform. The hinge joints are shown in yellow with light blue centres. The larger yellow cylinder mainly obscured by the body of the module rotates to provide tilt while the shorter cylinder is the roll joint.

The hinge on an Omni-Pi-tent module enables the module to perform 3D interactions in space as well as 2D interactions on the ground plane. In the real world, a pair of 12 V DC motors enable lifting and turning of the forward (Port 4) docking port depending on whether they are run in parallel or in opposition, similar to the mechanism on SMORES [35]. In simulation, the exact detail of the 3D printed bevel gear systems is not modelled, rather the system is simplified to two independent joints. Firstly one which controls the tilt motion of the hinge, and then as its child another joint which controls the roll motion of the forward port, see Fig.(4.6).

The joints are each restricted to a $\pm 90°$ range about their default position, matching the design of the real hardware. The torques provided by the joints have had to be made extremely large due to the inaccurate masses of the module models in V-REP. However the maximum rotational speeds had been informed by the predicted abilities of the hardware so while the simulated hinge joints, when supplied with an angular position to move to, may take an unrealistically short time to accelerate from standstill to the desired angular speed they should still take a similar time to complete their motion because the accelerations at the start and end of hinge motions take, in hardware as well as simulation, much less time than the slow travel at constant speed from angle to angle. On the hardware robots PWM values can be used to run the hinge at reduced speeds, although given the slow maximum speed this is rarely likely to be done. Within the V-REP simulation this functionality is replicated by modifying the maximum speed at which the joint, in position control mode, is able to move. Values for this maximum speed are supplied using custom datablocks which are written to by the child script representing the robot controller and read from within the joint object's callback script.

## 4.5 Simulating Omni-Pi-tent's Sensors and Communications

As well as representing Omni-Pi-tent's actuators, the simulation model approximates both the sensing and communication capabilities of the hardware. These are:

- Docking contact detection switches
- Orientation sensing via a compass and accelerometer
- Hinge position sensing, both limit detection and incremental encoding
- Short range proximity sensing on each docking port face
- Global Wi-Fi connectivity to other modules throughout the swarm
- Analogue infrared beacon sensors around each port
- 38KHz infrared port-to-port communications

Other proprioceptive sensors for internal monitoring of battery voltages and temperatures are there simply for protection of the hardware and to enable emergency shutdowns if faults occur.

### 4.5.1 Simulating Contact Switches

Far and above the simplest way to simulate this functionality is to check distances between docking dummy objects, any distance above 1 cm between dummies is treated as the switches being unpressed.

### 4.5.2 Simulating the Compass

Compass and accelerometer functionality can be replicated with function calls [71] in the simulated robot's controller child script to read global orientation angles but must be used with caution as these sensing parameters typically have a more limited accuracy in the real world. Sec.(5.7) gives more details of the effect of limited real-world compass accuracy on hardware performance.

### 4.5.3 Simulating Hinge Sensors

Hinge positioning on the Omni-Pi-tent hardware is sensed and controlled from a slave microcontroller, the Hinge ATMEGA, not from code running on the robot's Pi itself. Therefore the details of how encoding switches are used for hinge position sensing are not required within the high-level controller code in which a robot runs the type of collective behaviour algorithms which are the focus of the Dynamic Self-repair concept. Just as in the real robots, hinge positions are, as far as the high-level code is concerned, simply presented from a function call.

### 4.5.4 Simulating Proximity Sensors

V-REP natively contains proximity sensor functionality which allows distance calculation to the nearest visible object to have entered the region which a sensor object is viewing. This is somewhat different from the way in which the real robot's proximity sensors behave. On the Omni-Pi-tent hardware a modulated IR LED shines from the centre of each docking port, and the strength of the reflected IR signal is used to detect objects. This strength is affected not only by distance but by object size, affecting how much area it reflects from, and surface properties such as colour. Black paint for example absorbs most of the incident IR radiation, white paper reflects strongly. One thing to note is that, while the real proximity system is not affected by reflections off the floor, these simply form part of the constant level of background noise, the V-REP models of sensors, when arranged and sized to match those of the hardware, would give anomalous floor detections regularly. Therefore, in simulation, the floor object had to be made undetectable to the proximity sensors. The real system therefore reads an analogue value related to various properties of the objects nearby. In V-REP modifications were not made for this, the controller code reads the distances as presented, rather the decision was made for this sensor to calibrate the hardware against the simulation.

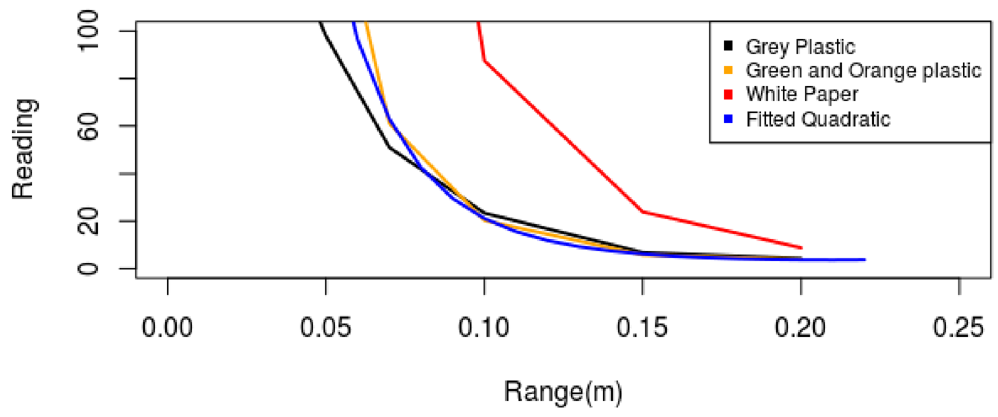


**Fig. 4.7:** The reading (unitless) given by an Omni-Pi-tent module's port proximity sensors, plotted against range. The test was repeated for a number of differently sized objects with different material finishes, the curves for plastic items were used to generate the fitted curve. Plastic objects of similar size to robot docking ports are, depending on colour, detected at ranges of 12 to 15cm.

### Calibrating Proximity sensors

An experiment was performed using a docking port, and associated circuitry, with objects of varying sizes and surface finishes, especially objects of the same kinds of plastic as the robotic swarm is printed with. These were positioned at different distances and strength readings taken on all 4 phototransistors of the port. The data for each object at each range was averaged and plotted with R, see Fig.(4.7). A quadratic, raised within an exponential, was used as a model for the fitting, on account of being easily solvable. With this done, the quadratic formula could be used to calculate the range for any given proximity reading, this code can then run on the real robot to convert unitless strength readings into a range equivalent to that provided to simulated controllers.

## 4.5.5 Simulating Wi-Fi

Global Wi-Fi communication can be simulated in a number of ways, the simplest and most relevant to the Omni-Pi-tent use case being with V-REP's signals feature[68]. Signals in V-REP are shared pieces of data which can be read and written from any object in the scene. It was assumed whilst developing this part of the simulation that in the later hardware experiments with Omni-Pi-tent the Wi-Fi would provide a reliable and almost instantaneous global method of communicating between all robots, that loss of messages was not likely and that all robots would remain in range of the router hub, hence the simulation model simply provides a global method of communication rather than simulate Wi-Fi physical layers and protocols precisely. Each robot model in V-REP is associated with a named signal including its hardware ID number within the name string, this signal object in V-REP acts to hold a queue of incoming Wi-Fi messages. When sending a Wi-Fi message to another robot, addressed by hardware ID number, the sending robot accesses the relevant named signal, and takes a copy of the existing queue. The message to be sent is added to the last place in this queue, if adding the message to the queue would over-run the buffer size of the queue, set arbitrarily at 10 messages for now, then the oldest message is deleted before the new one is added. The queue therefore never exceeds the buffer size. In the next simulation step the receiving robot will take a copy of the now updated global signal object, and attempt to read from the queue, taking the oldest messages first. Each message read is removed from the queue, reducing its length. The queue with read messages removed is again shared as a global signal under the ID number related naming convention. Most of the Wi-Fi messaging used in Omni-Pi-tent's group behaviours is addressed to the whole group, so often Wi-Fi messages are broadcast to all receiving signals by repeating this writing procedure for the whole range of ID numbers of robots present in the scene, this broadcasting method often helps avoid the difficulties which can be involved in maintaining updated lists of which hardware ID number corresponds to which temporary ID number, the latter indicating a robot's role in a structure or group. Although wasteful of Wi-Fi data bandwidth a robot needing to address another over Wi-Fi by temporary ID can send a message with the temporary ID included in the contents, which receiving robots can read and ignore unless the included temporary ID matches their own. When developing the Wi-Fi system for the real robots, specifically the C code to process network data, this architecture of named queues was copied to ensure consistency with simulation.

### 4.5.6 Simulating Docking Guidance Beacons

One of the most significant efforts performed in aid of improving the V-REP simulation of Omni-Pi-tent was that focused on the docking guidance system, and on reducing the reality gap related to modelling the behaviour of the analogue IR method used to provide guidance during port-to-port docking. While testing and results are discussed here, readers may wish to consult Sec.(3.3.3) for a reminder of the rationale behind this sensor system. Later on, Sec.(5.2) explains the circuitry used to implement this sensor, but such details are not required to appreciate the nature of the reality gap tightening exercise described here.

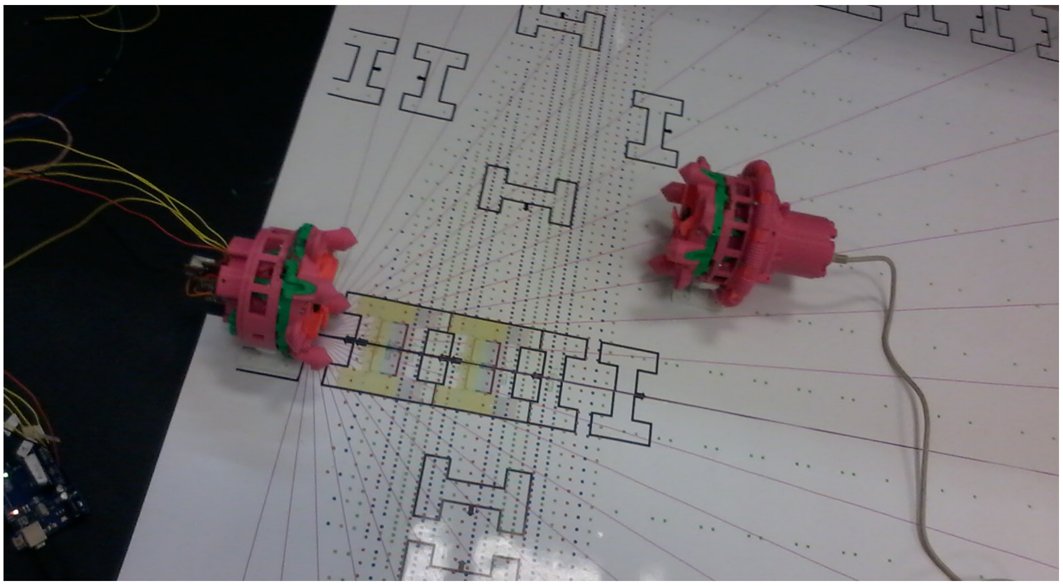


**Fig. 4.8:** A photograph showing the arrangement of the emitter (left) and receiver (right) docking ports used to gather range and readings data. The yellow wires on the left carry data from the amplifier circuit to the analogue inputs of an Arduino which contains the same type of ATMEGA328P chip as used inside the docking port.

### Refining Guidance Beacon Simulations

This work was conducted at a point when two docking ports for the first prototype had been constructed. One acted as the recruited port and was placed at a central position on a measured grid. The analogue values derived from the 4 phototransistors around the rim were logged from this port. The other port, acting as recruiter and blinking an IR LED at 5 KHz, was placed at a variety of locations with $x$ and $y$ co-ordinates differing relative to the recruited port, Fig.(4.8). We define $x$ as being towards and away from the port and $y$ as side to side. Of course during actual docking procedures the emitter would be a stationary or moving seed and the receiver would change position relative to it, however the wires trailing from the receiving port in this experiment made it easier to move the emitter instead, the situation is symmetrical either way so long as “left" and “right" are carefully defined when comparing signals from different phototransistors. At each position of the emitter the receiver was allowed to run for enough time to collect 50 to 150 datapoints from each of the 4 phototransistors. The full data gathered at each position was saved to a separate text file. Positions for the emitter to move between were all spaced apart by centimetre distances on the $x$ and $y$ axes, as appropriate to the marked grid. Errors of more than $0.5$mm in position could not have feasibly occured. When moving between positions, the relative angles of the ports were maintained at all times as if both ports were on robots which had already matched compass orientation before trying to dock. The $x = 0$, $y = 0$ point was set as being when both ports were docked together, other positions were recorded as how many centimetres in $x$ and $y$ away from this situation the emitter port was. In total, data was taken at 218 different positions, spaced $1$cm apart at close

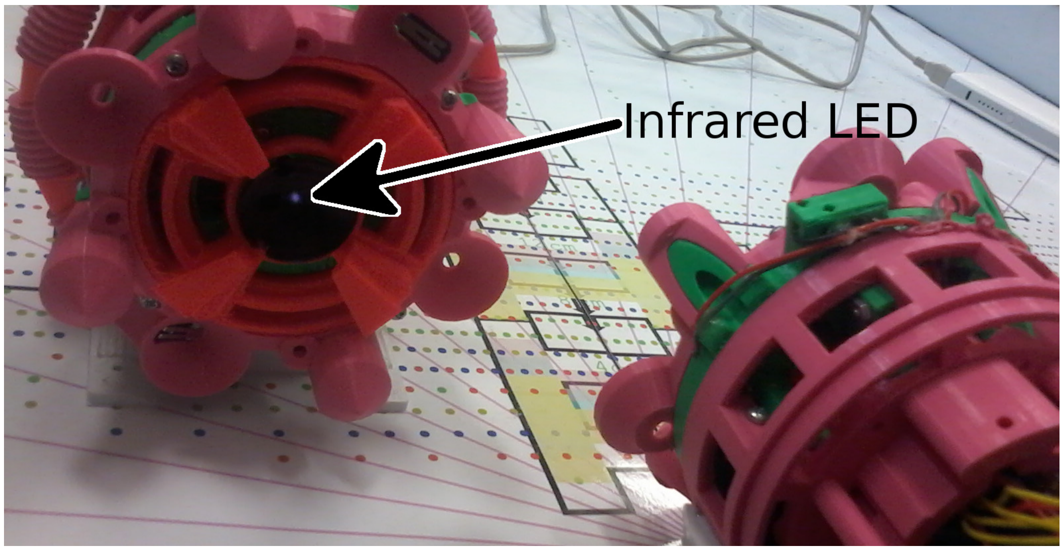


**Fig. 4.9:** A close up of the docking ports experiment when the ports were at a small relative distance on $x$ and $y$, the emitter is shown on the left with the IR LED just noticeable in the camera image. Note the dotted grid visible below with 1cm spaced markings in the regions of low $x$.

ranges, see Fig.(4.9), and many centimetres apart at longer ranges. It was considered too time consuming to take data at a 1cm resolution in the outer regions of the illuminated cone, as measured values here changed slowly across space. The 1cm spacing was used close-in where values had been expected to, and were then found by experiment to, change rapidly with distance.

Data was analysed using R [25]; the plot3D, akima, pracma and spatial libraries proved especially useful. For each of the 218 files, an average and standard deviation for each phototransistor's readings was taken. An average signal strength across all phototransistors was also taken for each data file. Then for each of the data files taken at $y = 0$ positions a coefficient by which each phototransistor's average should be multiplied to get the overall average across all 4 phototransistors at that position was found. The coefficients found for Phototransistor 1 at each $y = 0$ position were mean averaged to get a coefficient for Phototransistor 1, this was repeated for the other three phototransistors. With the coefficients found all data, including for $y \neq 0$, was multiplied by the relevant coefficients. An appropriately averaged standard deviation was also found. This data, collected on an unevenly spaced grid with tight spacing at low $x$ and longer spacing at high $x$, was put through R's `interp` function to generate a matrix of signal strengths, as predicted by linear interpolation, at positions evenly spaced in $x$ and $y$. The results of the interpolation are shown in Fig.(4.10) and Fig.(4.11).The interpolation was necessary to be able to produce visibly readable 3D graphs.

We can observe how, while the averaged strengths between all phototransistors are symmetrical about the $x$ axis, the signal strengths on the individual phototransistors are not. This is to be expected as when off-axis the phototransistors on the port, especially Phototransistor 1 and 3,

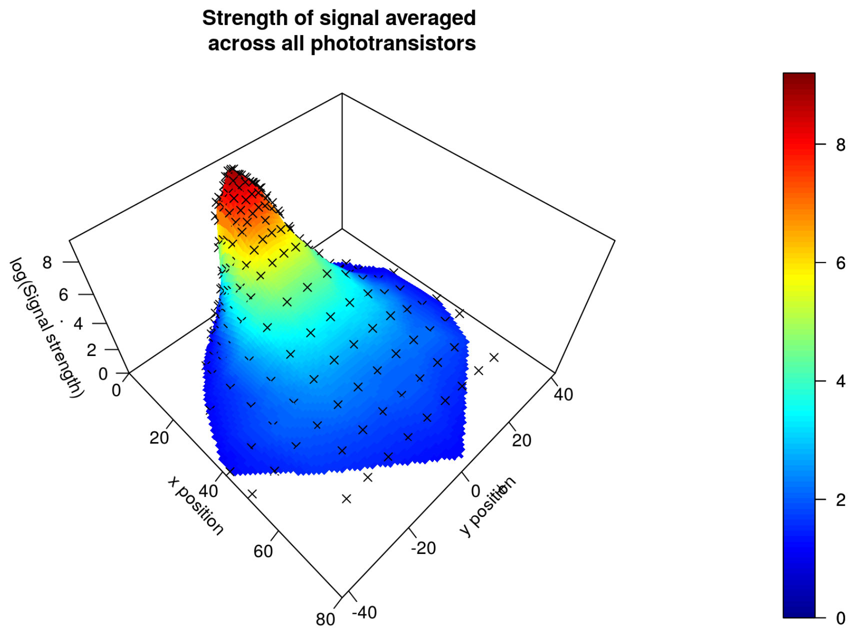


**Fig. 4.10:** A graph showing the interpolated function of the averaged, across all 4 phototransistors, signal strength (arbitrary units) across $x$-$y$ space. Black crosses show the actual data at the appropriate signal strength $z$ values and $x$-$y$ measurement locations. The graph uses a logarithmic scale to bring all the data into a conveniently viewable range. Note that this graph is best viewed in the full colour pdf file and will not be readable on colour printed paper.

will each be at different distances from the emitting LED and some will be illuminated by the stronger light at the centre of the LED's cone while others will be in the dimmer 5KHz light levels to the side. Fig.(4.12) and Fig.(4.13) make this point apparent.

Model fitting was empirical and without physical basis, the purpose of the models was to act as more easily expressable versions of lookup tables to go into functions in the refined V-REP simulation so overfitting is infact desirable rather than concerning in this case.

A $z = f(x, y)$ model was fitted using linear regression functionality and a 5th order polynomial model. Even at 5th order this struggled to match the steepness on some parts of the already logarithmic data. However by recognising that $z$ values in the data are symmetrical about the $x$ axis we can see that we only need to fit the equation for the region $-42 < x < 0$ and can mirror our model once finished making sure to use the negative absolute value of $y$ when making calculations with our model. We can also limit the area to fit over further by recognising that beyond $y = -30$ or $x = 60$ there is no useful signal, it is barely above background levels. We also clip away some of the region $x < 5$ to save having to model the point of the peak itself and instead just focus on its important sloping side. We re-extend our region to model across space on the $y > 0$ side of the $x$ axis, so the linear regression equation

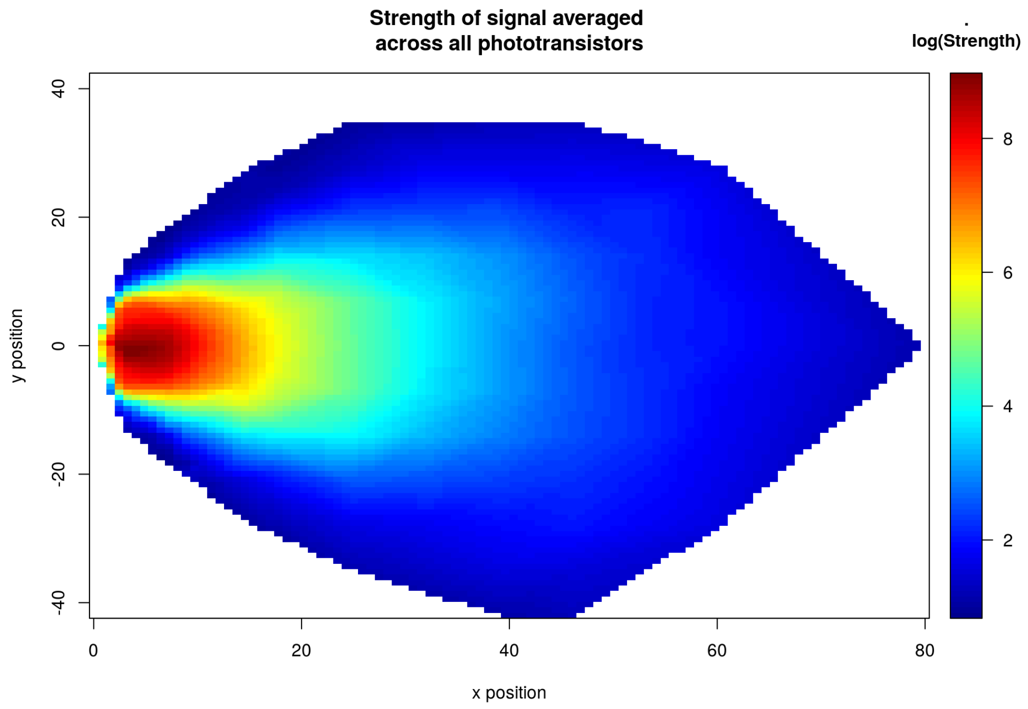


**Fig. 4.11:** An alternative depiction of Fig.(4.10), using a heatmap like plot rather than a 3D effect. Positions of actual data are not marked in this graph.

can get a better idea of the fact that at any given $x$ values the maximum signal strength always occurs at $y = 0$. The model produced, see Fig.(4.14), by linear regression enabled a 21 term equation,Eq.(4.1), to be written out, not hugely convenient to write by hand but much easier to insert into a V-REP simulation than a large lookup table representing each point on the interpolated grid. Before using Eq.(4.1) the negative absolute value of $y$ must be taken.

$$\begin{aligned}z = exp(7.763874 + 2.296599 \times 10^{-1}x \\ - 2.966970 \times 10^{-2}x^2 + 9.507012 \times 10^{-4}x^3 \\ - 1.331212 \times 10^{-5}x^4 + 7.049903 \times 10^{-8}x^5 \\ + 1.343266 \times 10^{-1}y - 1.651029 \times 10^{-2}yx \\ + 7.387918 \times 10^{-4}yx^2 - 1.496500 \times 10^{-5}yx^3 \\ + 1.128575 \times 10^{-7}yx^4 - 5.219740 \times 10^{-2}y^2 \\ + 2.591042 \times 10^{-3}y^2x - 4.200419 \times 10^{-5}y^2x^2 \\ + 2.097531 \times 10^{-7}y^2x^3 - 7.801737 \times 10^{-4}y^3 \\ + 4.647594 \times 10^{-5}y^3x - 6.018875 \times 10^{-7}y^3x^2 \\ + 1.684589 \times 10^{-5}y^4 - 2.431474 \times 10^{-7}y^4x \\ + 1.224325 \times 10^{-7}y^5)\end{aligned} \tag{4.1}$$

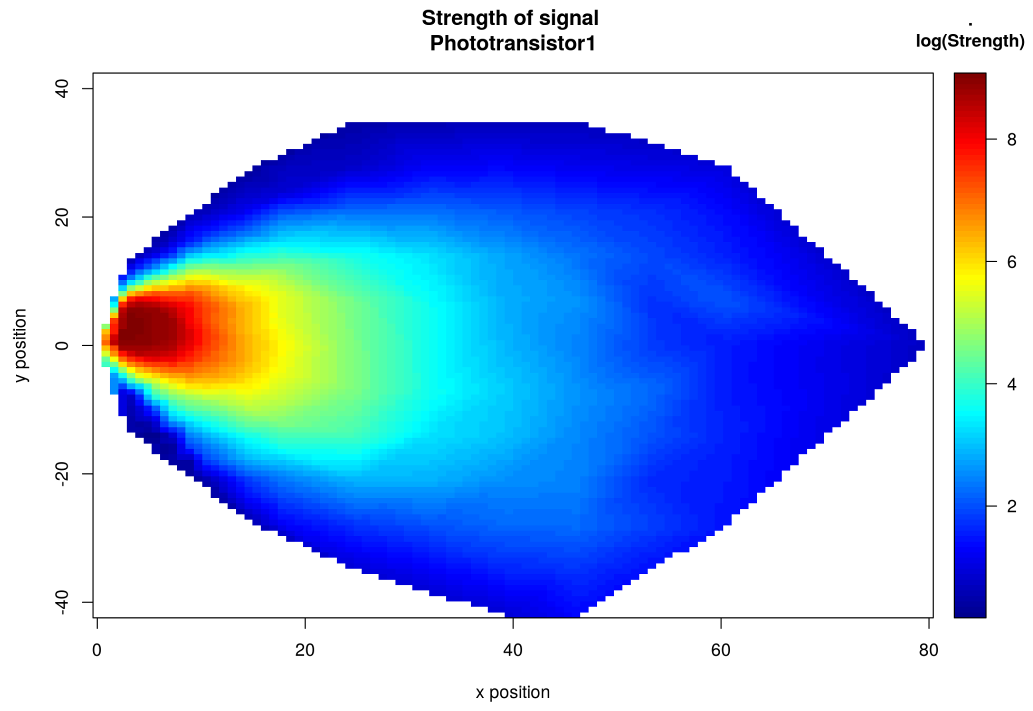


**Fig. 4.12:** A heatmap plot of readings from Phototransistor 1, note the lobe of strong signals occurring to one side.

We can compare the model to the data by looking at the percentage difference between the model's predicted value and the data's interpolated value at a variety of $x$,$y$ points. At the sizes of the images in this thesis it is not practical to distinguish by eye the model and data both overplotted on a single 3D graph, hence Fig.(4.15) shows a plot of the percentage difference between model and data across the regions of space which interest us, those within the cone. Except at a few points on the extremities of the cone, and many regions outside it which will not matter in the V-REP simulation, most predictions by the model are within $\pm 50\%$ of the data values at the same locations. This was considered as an adequate enough fit for use in the simulations which followed.

The testing confirmed what had already been expected from the data sheets for the LEDs and phototransistors, that their brightness/sensitivity dropped off with angle. While neither the approaching nor the recruiting port were rotated relative to each other it should be apparent that the angles at which the LED was illuminating the phototransistor, and the angle at which the phototransistor was being illuminated, both varied when the ports were moved apart on the $\pm y$ axis. Although this acts somewhat to reduce the size of the cone, narrowing it's width especially at longer ranges, it also provides a beneficial effect which can be exploited to improve docking above what would have been possible only with information indicating if a given phototransistor was inside or outside the illuminated cone.

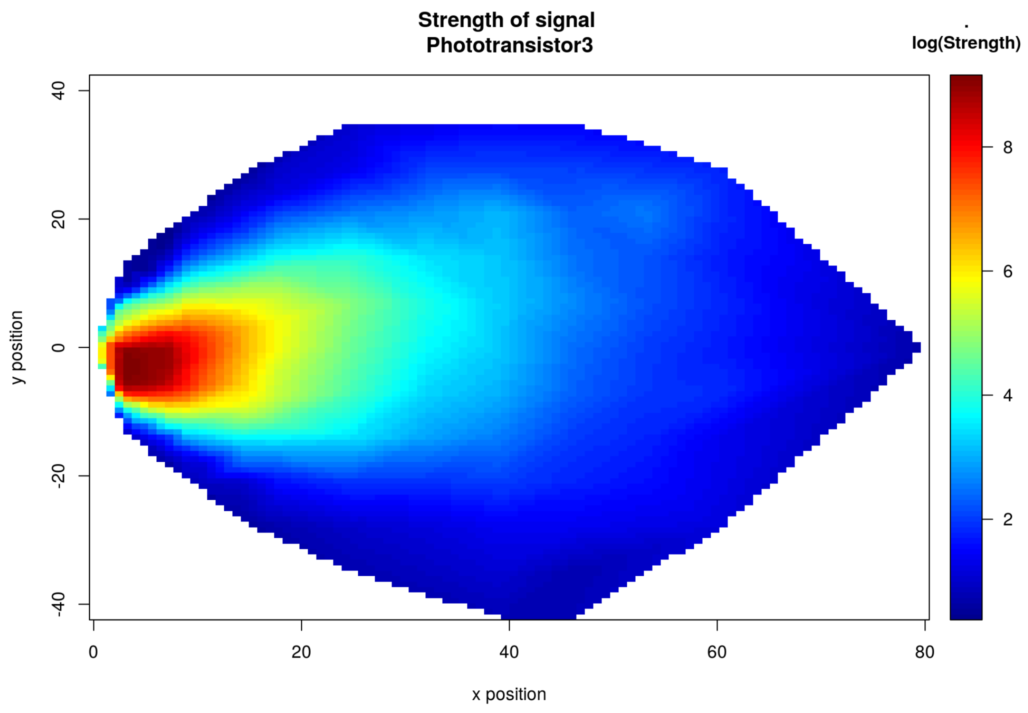


**Fig. 4.13:** A heatmap plot of readings from Phototransistor 3, a lobe of strong signals occurs on the opposite side to the lobe seen in Phototransistor 1's plot.

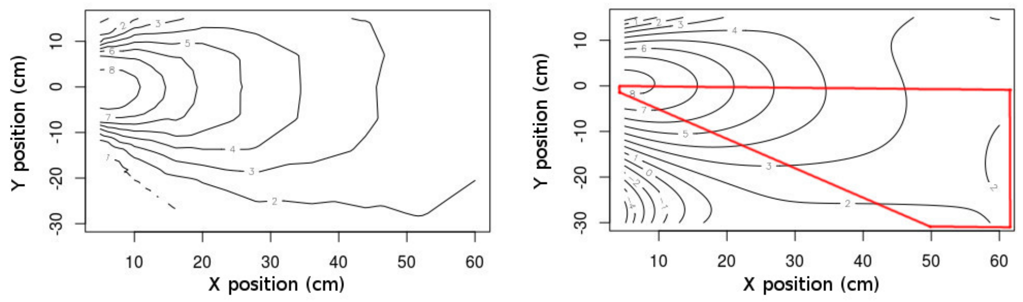


**Fig. 4.14:** A comparison of experimental data against, on the left, against the 5th order polynomial model, on the right, fitted to the data in the red bordered region representing the limited angle of the cone. The regions of the model at $y > 0$ are not used as values always have an absolute value taken and turned negative before being entered into the polynomial. These regions were present when fitting the model as they helped to ensure that the model did not create contours which deflected towards low $x$ at $y = 0$, including the $0 < y < 15$ region in the graph helped get the contours fitting the data better by allowing the linear regression algorithms to exploit some aspects of the data's symmetry.

### Implementing IR Guidance in V-REP

Guidance beacons were modelled in V-REP by adapting the code previously used by a "proximity sensor object", here representing the shining IR LED, to write custom datablocks to

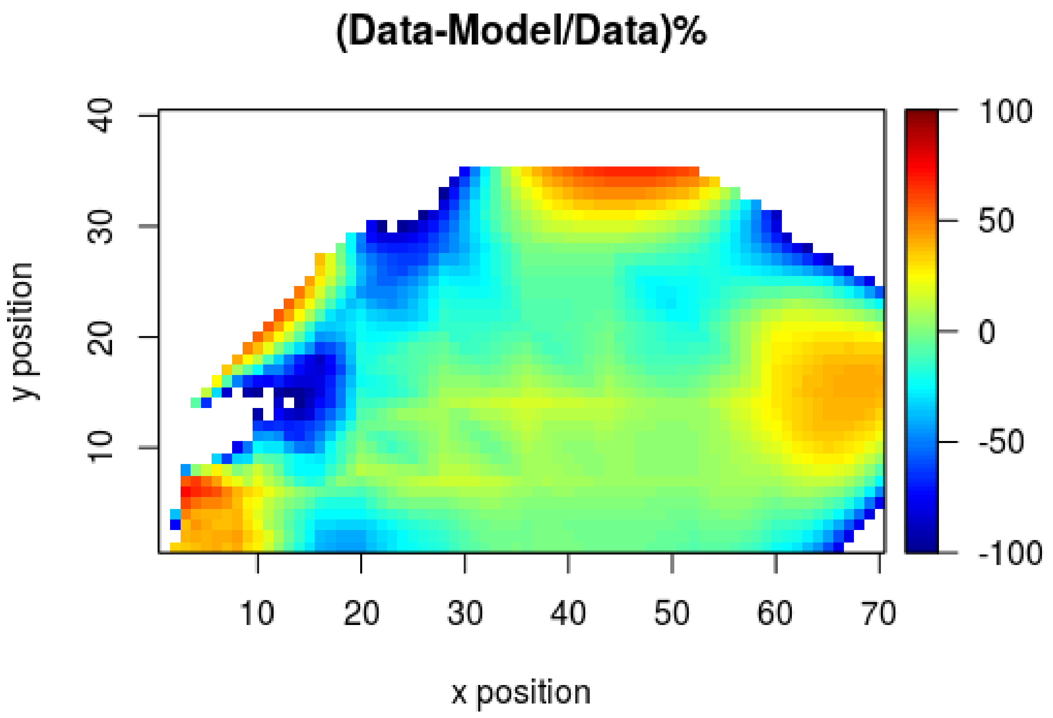


**Fig. 4.15:** A plot displaying, over the regions in which the model will need to be evaluated in simulation, proportional differences between the model and data. These are proportional differences in actual values not in logarithmed values. Greens, light blues and pale yellows are the best regions. White areas indicate where no data was taken and also show regions where the proportional difference between model and data are beyond the $\pm 100\%$ range which the colour scale has been fitted to. Careful use of a few **if** loops in the V-REP simulation can be used to ensure that outside the illuminated region a background level is always used rather than trying to evaluate the polynomial, **if** loops can also be used to handle the few regions along the cone's edges where the differences between model and data are most extreme.

"dummies" which represent the phototransistors arranged around each docking port. In each sensing step of the simulation all phototransistor dummies are first given a zero value as a reading. Then for each dummy in the "field of view" of the proximity sensor object representing an LED the dummy's custom datablock is given a value derived from the polynomial model with inputs of the relative positions of the (phototransistor) dummy to the (IR LED) proximity sensor object. The geometry of the proximity sensor object's cone means that although the polynomial model diverges sharply from the real data outside the conical region, the model is still suitable for use as only dummy objects within the limited volume of space which the proximity sensor object can scan, the same limited volume in which the model is accurate, have datablock values written to them. The simulation code responsible for performing the model's calculations runs, during each simulation step, once for each separate instance of a dummy within an operating proximity sensor cone, a number which rises rapidly when large numbers of simulated robots are within range of each other. Running is improved by the fact that, like in the real robot hardware, each guidance LED can be turned off when it is not being used to guide a robot in to the relevant docking port. The Lua scripts are written such that only LEDs which are turned on during any given step need to have their full child scripts called.

If a proximity sensor object tries to write to a dummy which has already been written to in this simulation step then the reading on the dummy and the reading being added are combined together in such a way as to give a strength based on multiplying the stronger of the two readings by $0.5$. This attempts to capture the fact that the $5$KHz signals could interfere constructively, strengthening is considered by picking the stronger of the two values, or destructively, weakening is considered by multiplying the value by $0.5$. A more detailed treatment based on randomising the type of interference to account for the randomly differing phases of $5$KHz LEDs was considered to be unwise to add, as it could greatly increase the time taken to execute a section of code which gets called very regularly and hence drag the whole simulation to a crawl. In the actuation step of the simulation the datablock value is read by the robot to which the phototransistor dummy belongs. The value on the phototransistor is set back to zero in the next sensing step of the simulation and again polynomially derived values are written to it.

This model for the IR guidance system is entirely contained within the lower hierarchy objects making up the model of the Omni-Pi-tent platform, it can operate, although more slowly, for as many modules as are in a scene and does not require robot controller code to be operated any differently than when using the hardware IR system on the real robots.

### 4.5.7 Simulating Infrared Communications

The line-of-sight TSOP38438 $38$KHz infrared communication system has been modelled in V-REP in a similar fashion to the IR guidance system. Again a “proximity sensor object" acts as the emitter and “dummies" act as the receivers. Controller code running in the child script of an Omni-Pi-tent module model can specify, in a similar fashion to the functions used to control the IR communications of the real hardware, a message to send, and one of three discrete power levels for the sending LED to operate at. The simulation of this communication system has been refined to includes considerations for different ranges depending on the power level and, although implementing a truly line of sight simulation is rather challenging and extremely computationally hungry, simplified restrictions have been placed on the relative angles between a TSOP and a $38$KHz LED at which communication can take place. The TSOP sensor in the real robot is off-centre in the docking port and could not be mounted in a way which gave as wide a field-of-view as the IR LED can cover, this has been included in the simulation model by rotating the centre-lines of the cones simulating the TSOP sensitivity so they are not co-aligned with the cones of $5$KHz guidance light. The simulation does not account for those regions where a TSOP reads an LED's $38$KHz signal which is strong enough to be detectable but weak enough that much data is lost and CRC verification, as described in Sec.(3.3.4), fails, the simulation does not account for interference which such a weak signal may cause to stronger signals also being received by the TSOP. If a TSOP, represented by a special type of dummy in V-REP, receives multiple messages during a single simulation step then only one will be kept and the others discarded. Which is kept is determined by the internal details of how V-REP orders the running of different object's child scripts within it, however this acts as a plausible approximation for the way that in the real hardware the docking port ATMEGA will discard earlier TSOP messages which have arrived since the Pi last requested data from it and will only pass on the latest IR message received to the Pi the next time the Pi makes a request. Overall less data about the TSOP receivers was gathered from hardware tests than was gathered in hardware tests of the 5KHz guidance system, largely because at any point the only data point is a binary "can a message be received or not" rather than a continuous range of intensity readings. The TSOP simulation is not as refined as the guidance one but still provides far more useful results than cruder simulation methods could.

## 4.6 Simulation Sensitivity

Whilst many of the simulation parameters were refined according to measurements from hardware subsystem testing, some limitations remained, especially in the details of simulating actions involving multiple subsystems which could not be experimentally verified until a later stage of hardware development. This section provides a discussion of the concept of sensitivity analysis and its application to parallelising hardware and simulation development. This section also highlights some differences discovered between the simulation model and the hardware, not mentioned elsewhere in the thesis. It considers what effects inaccuracies in these aspects of simulation could have on overall behaviours. It should be noted that these may also allow some qualitative estimations to be postulated regarding the effects that using alternative hardware modules with different values for key performance parameters may have on a multi-robot organism's overall group abilities.

Sensitivity analysis allows the effects of various inputs on a system to be probed by observing how outputs differ when inputs are varied. Such an analysis can be used to, for example, judge how important the sensor range of a robotic module design is for the time taken to self-assemble, and in doing so allow a designer to decide whether ploughing their efforts in to sensor range improvement is worthwhile, or whether other parameters may have greater effects on the performance figures they seek to optimise. The technique can also be used to judge which parameters of a model require the most precise measurement for the model to yield accurate results. In the context of parallelising hardware and simulation development sensitivity analysis can be of particular use to estimate which aspects of the simulation require the most thorough models of subsystem to be used, and which can get away with much cruder approximations. Although time constraints did not permit formal sensitivity analyses to be run in the Dynamic Self-repair project, a qualitative discussion of a few cases can provide interesting insights.

Whilst physics engine details forced the adoption of inaccurate masses and moments of inertia in the simulation models, these should have little effect on overall results as they would affect different self-repair strategies in similar ways and be cancelled out by torque modifications which the physics engine required. Inaccuracies in friction models within the simulation could on the other hand have much greater effects, particularly on the simulation of failed robots being dragged away by other modules, this could introduce important differences between strategies which rely more or less heavily on the robotic modules' ability to manoeuvre their neighbours. Frictional coefficients between a wheel with fine details and a partially compressible carpeted floor are parameters which are both difficult to estimate from theory and challenging to test experimentally until full-sized full-weight modules have been constructed. Inaccuracies in simulating proximity sensors would primarily only affect how close robots would come to obstacles before changing course, whereas inaccuracies in simulating the docking detection sensors, the contact switches, could have a major effect on modules' understanding of whether they are docked on any given port at any given time.

## 4.7 Using the Omni-Pi-tent Simulation Model

All of the main functions of the robot model can be controlled with various function calls in a robot's Lua child script, it was chosen to use internal Lua child scripts for the simulated robot controllers as this would reduce V-REP's dependency on the external API and any other elements which may have proved trickier to clusterise for later bulk simulation runs. The child objects within an Omni-Pi-tent module model have been carefully named to avoid having any numerical characters at the end of their names, this ensures that `sim.getObjectHandle` will give an identifier for the correct object when used.

For the wheel joints the command `sim.setJointTargetVelocity` is typically used, noting that the speed is by default in radians/second. The hinge and hook joints can be controlled via use of `sim.setJointTargetPosition` which allows their target position, in radians, to be set. `sim.setLinkDummy` and `sim.setLinkDummy` can be used to dock by setting links between dummies, subject to the alignment conditions explained earlier. `sim.readProximitySensor` is also often needed, it outputs both a resolution value indicating the presence of a detection and a range to the nearest point within its field of view.

While other parameters for all these objects *can* be set from scripts, for example the speed of the hinge and hook joints, they are not generally expected to need to be, rather most parameters have already been set to realistic values within sub-menus of the GUI and some would not be expected to be variable on real-world robots.

While the code for the simulation and hardware are not in the same language, making direct reuse of simulation code on hardware impossible, within the simulation environment Lua functions were written which provide closely comparable interfaces to the actuation, sensing and communication subsystems to those available in the C API used by the Omni-Pi-tent hardware. This ensures an easy process when translating controllers from Lua for simulations to C for hardware.

## 4.8 Simulating Docking

V-REP [28] was used to verify the simple principle underlying the method for docking of robots, both to static and moving seed modules.

The earliest simulations in V-REP verified that simply recognising whether a robot was within the illuminated cone or not, using just a binary in or out on each phototransistor rather than the calibrated model of analogue strengths, and matching the compass orientation with the recruiting robot was all that was necessary for the very simplest docking operations. Those simulations demonstrated the usefulness of an omnidirectional drive in docking scenarios and showed the importance of having a method for robots to “time out" from docking if they failed to achieve it within a specified time of entering the cone. Notable difficulties encountered, to be fixed in more refined later simulations, were:

- The difficulty of distinguishing: when a robot found that suddenly, the phototransistors on its docking port were no longer illuminated, whether it had drifted out of the curved outer regions of the cone or had overshot in the inner regions of the cone, see Fig.(4.16).
- Situations where multiple recruited robots attempt to dock to the same recruiting port at once, these are further considered, alongside mitigations using a handshake protocol over IR communication links, and discussed in Sec.(6.2.6).

Early simulations initially struggled with docking to a moving robot, as it was far too easy for an approaching robot to become lost outside the illuminated cone as the recruiting robot passed. When a seed robot was recruiting on a port perpendicular to its driving direction there was also a difficulty during the last few centimetres of docking where the phototransistors of the approaching port will no longer be within the cone. For a stationary seed the approaching robot can simply drive along the direction of its recruited port to cover these last few centimetres before the spikes make contact, but with a moving seed a robot attempting to move solely in the direction of its recruited port would misalign, often severely enough to prevent spike-pit contact, due to the change in the seed robot’s position during the time taken for this final approach, see Fig.(4.17). This was corrected by having all motions of the approaching robot defined relative to the reference frame of the recruiting robot, hence, while in the reference

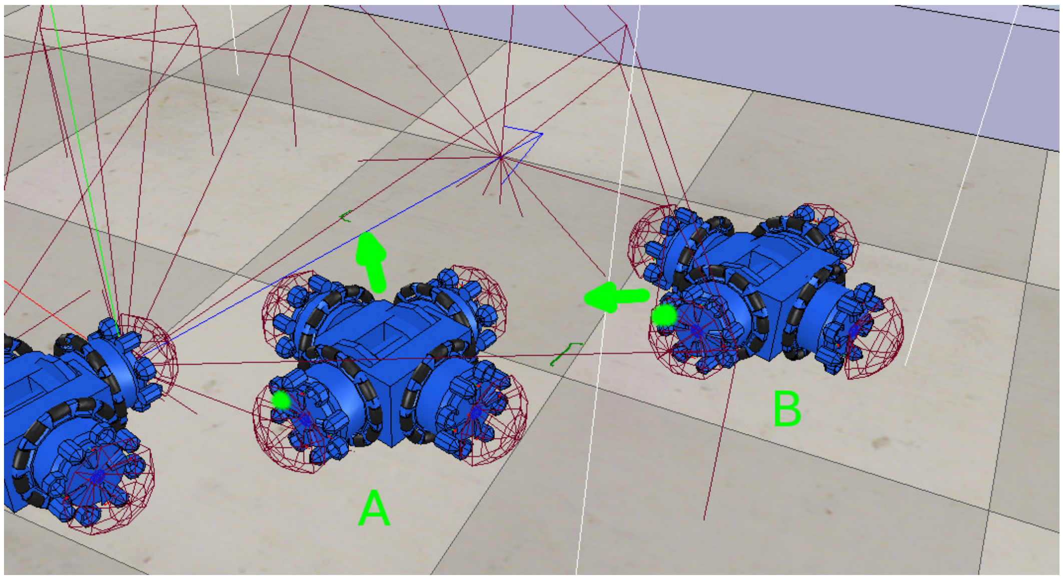


**Fig. 4.16:** A robot at either or the marked positions detects that phototransistor 1, 2 and 4 are outside the cone, but using only binary “in or out" information cannot tell which position it is in. In the image green dots show those phototransistor which are illuminated. A robot in position A will need to drive in a direction some 45° between Port 1 and Port 4 so as to re-enter the cone fully, whereas driving in such a direction from position B will cause the robot to lose the cone completely, a robot at B would instead need to drive at 45° between Port 3 and Port 4. These correct directions are shown by arrows.

frame of the seed the approaching robot appears to move straight in an observer in the ground’s reference frame sees the approaching robot moving at an angle, see Fig.(4.18).

Another interesting finding was that robots docking towards ports, especially in the V structure shape, would often find themselves colliding with already completed parts of the structure if they move in along the edge of the cone, this is because the half-angle of the cone is greater than 45° hence wide enough that parts of other robots near to a recruiting robot may protrude into the cone. A solution to this problem emerged in the refined simulation, by allowing robots to guide themselves inward without being so reliant on the very outer edges of the cone.

In the later simulations, once the guidance beacon system had been refined to give an accurate reading based on hardware experiments, docking was made much more reliable by comparing the strengths of IR signals on each phototransistor around the rim of the approaching docking port. Imbalances in signal strengths can be used to infer the direction to steer to return towards the centre-line of the recruiting cone, and when strengths are equal between the phototransistors it is known that the approaching robot is on that centre-line. Extreme inequality between the signal strengths, such as the phototransistors on one side giving a zero reading while those on the other give a strong reading, trigger a partial retreat to realign, as they typically occur if the recruited robot is close to the recruiting port but so severely misaligned that the matching spikes and pits would make contact in such a way as to prevent correct docking. This use of analogue levels also helped identify confusion situations

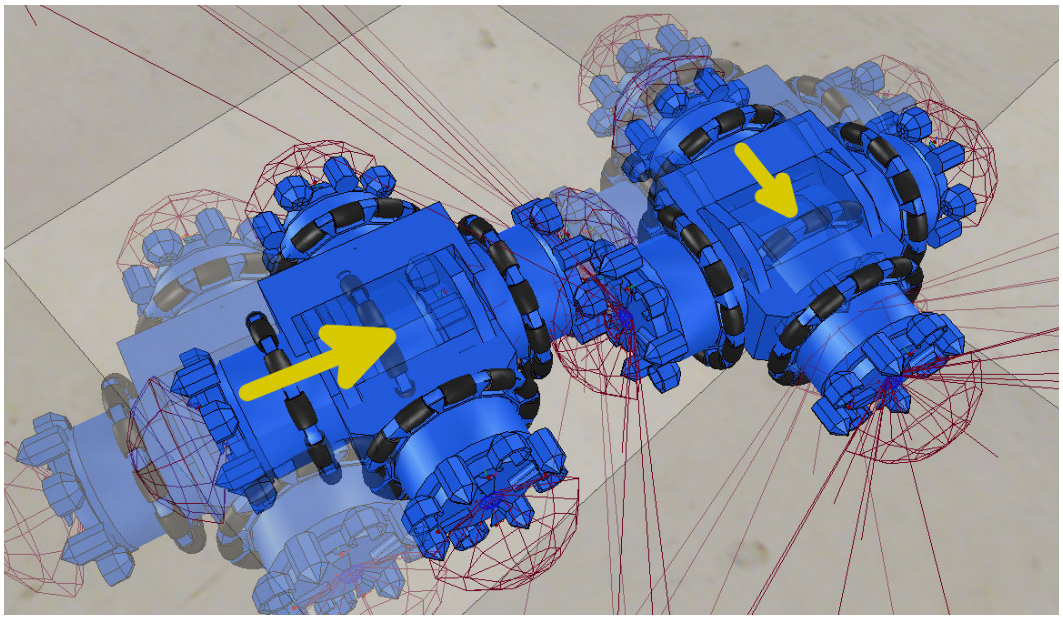


**Fig. 4.17:** The approaching robot (left) drives straight inwards (the approacher's Port 3 direction) but during that time the recruiting robot has moved sideways (the recruiter's Port 4 direction) far enough to prevent alignment of the spikes and pits. Previous positions are shown in faded colours, current positions at the moment of misalignment in solid colours. The misalignment has been exaggerated for greater visibility in this image. In reality, the approaching robot is much closer than the length of the yellow arrow before it becomes too close to use cone-derived position information.

such as illustrated in Fig.(4.16), these could now be distinguished as the readings on the phototransistor with the highest readings would be much lower in the far outer reaches of the cone that in an overshooting scenario on the sloping flanks. The findings from these simulations allowed for optimising the controller running on the hardware robots to ensure more reliable docking.

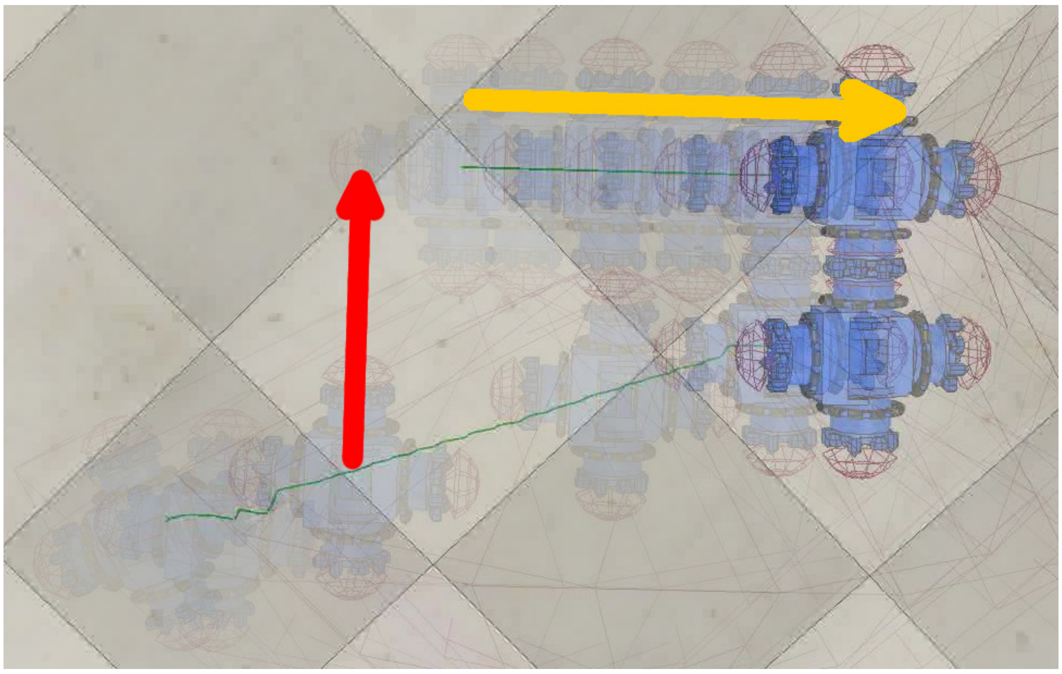


**Fig. 4.18:** To fix the misalignment in the last few centimetres the approaching robot makes all motions relative to the, moving, reference frame of the recruiter. In the recruiter's reference frame the final docking motions are straight inwards (red arrow) whereas in the reference frame of a ground observer the final motions are angled (green ground track). The yellow arrow shows the recruiter's motion. For a non-moving seed the mathematics used in the controller for this reference frame based docking simplifies to exactly that used in stationary seed docking.

## 4.9 Clusterising V-REP

To gather large amounts of data for statistical analysis it was necessary to run V-REP on the university's Viking cluster [84]. Multiple considerations had to be accounted for to enable the V-REP software to function correctly in a clusterised environment, whilst ensuring that the simulations themselves were not behaving differently from on the desktop PC where they had been developed and where group behaviour algorithms were refined and subjected to capability test scenarios before statistical runs were made on the cluster.

Viking, like most clusters, uses a headless Linux operating system, whilst a GUI is available for some programs during preparation of jobs there is no graphical front-end on the nodes where jobs run. V-REP is heavily dependent on a GUI to which it outputs some data, and which the simulation engine also seems to require for processing some aspects of sensing and physics. xvfb (X Virtual Frame Buffer) software was used to create a virtual graphical environment which V-REP could interact with as if it were a GUI, a server number for xvfb was specified which it was expected would not conflict with any other jobs running on the cluster.

V-REP, again due to being developed primarily as a desktop application, does not easily run multiple instances simultaneously. Some computing resources are shared between multiple open simulations which cause difficulties for others to perform functionality such as stopping and starting simulations, and writing out to data files. Notably the Remote API port number is shared and can cause conflicts between simulations, if multiple users each wish to submit V-REP jobs to a cluster they should each set a unique value for this port number in their configuration files, the most practical solution to averting these problems is to have each V-REP instance running on a separate node of the cluster, 100 jobs on 100 nodes in parallel were used for many of the submitted simulation scenarios. V-REP only makes use of a single CPU core, so only one core on each node was required, this meant V-REP could not be substantially accelerated by providing extra cores or RAM, so the simulations each ran with a relatively low computing resource demand, but across many nodes and for time periods similar to those required to run a single simulation on a desktop PC.

Jobs submitted to a cluster have a maximum allowed time for which they can run, therefore the V-REP simulations must have a time based as well as result based ending condition included. In simulations conducted for the Dynamic Self-repair project all V-REP simulations were set to end after desired conditions were met, such as the formation of a structure and its motion to a position in the dynamic self-assembly simulations, but also to time out after an amount of simulated time, see Sec.(6.3.1) and Sec.(7.6). Simulated time, for a simulation featuring a given number of robots, is linearly proportional to the time taken for the cluster to run the job, hence a value could be set here which would allow simulations the best chance to complete, but to ensure they still wrote out to csv result files, providing information on why the robots had not yet completed the task before the cluster's grid engine cut-off the job entirely. Such simulations are referred to as timed-out and could not be used in statistical analyses, but could still be used to estimate proportions of a certain strategy able to complete within time, and to provide results useful for debugging.

Debugging was also aided by making video recordings of the clusterised runs, the technique for this was not developed in time for the Self-Assembly experiments of Chapter 6, but proved invaluable in diagnosing reasons that some repairs did not complete in the experiments in Chapter 7. While V-REP internally contains a video recording module this could not be made to activate by script when run non-interactively on the cluster. Hence recording was instead done by taking a series of automated screenshots of the xvfb buffer, then combining them with ffmpeg after the simulation had finished so as to produce an mp4 file. These mp4 files are automatically named and placed in folders such that it is simple to recognise which csv datafile each of them corresponds to.

With these capabilities developed it was possible to perform large batch runs of simulations on the cluster, as used in Chapters 6 and 7.

## 4.10 Summary

This chapter has presented an introduction to V-REP and to simulating the Omni-Pi-tent platform within it, the challenges of the reality gap, and some mitigations taken, have been described.

While the V-REP simulation model of the Omni-Pi-tent hardware, as discussed, is not accurate to the real robot hardware in every exquisite detail, it enables a user to write controllers for complex behaviours and then have them executed by simulated robots in a virtual space. By providing calibrated models of sensors and actuators within the simulation, which are interfaced with via closely comparable functions, the simulation environment enabled development of multi-robot algorithms while the hardware was still in development, and provides an environment from which code can be relatively simply translated for use on hardware after any obvious bugs have been ironed out in simulation.

The next chapter continues our exploration of the infrastructure underlying the Dynamic Self-repair project by discussing the complexities involved in progressing the hardware from concepts and subsystem prototypes to a fully functional robotic platform.

Chapter 5

# Physical Implementation of the Omni-Pi-tent Platform

Development of the Omni-Pi-tent module proceeded in a number of stages, an initial prototype was constructed, in time for TAROS 2019, to test many of the initial design ideas. A second prototype was then built to optimise the electronic architecture and overcome a number of flaws in the initial design which had hindered the first prototype's production and fully functional operation. The second prototype was used as a basis for the final version and production of further robots. During the construction and testing of this “final" version the need for a number of design improvements was identified. Hence the second prototype and production robots were upgraded via the replacement of several subsystems, mostly mechanical, to produce the design used in experiments.

This chapter explains the key points from across those design iterations, handling each subsystem and design consideration separately but with considerations of how it affected, and was affected by, other aspects of the design. A subsection is dedicated to each subsystem, which is described in terms of its contribution to robot capabilities, the chapter concludes by discussing demonstrations and tests undertaken with the fully integrated hardware before switching the focus back to multi-module capabilities.

Fig.(3.10) acts as a useful reference throughout this chapter and is reproduced here as Fig.(5.1).

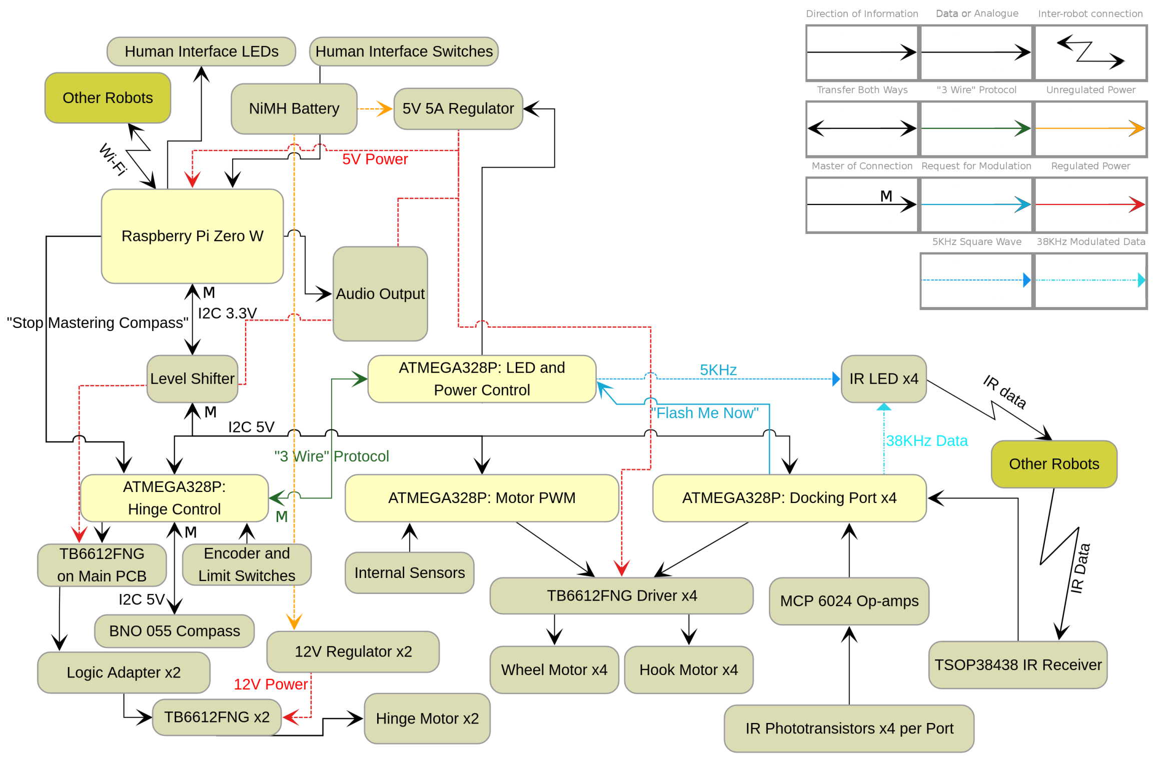


**Fig. 5.1:** A block diagram of the electronic components inside an Omni-Pi-tent robot. Reproduced from Sec.(3.4) as a quickly accessible reference for this chapter.

## 5.1 Docking Port Design

Omni-Pi-tent's docking ports provide genderless connections between robots with the ability for single-sided disconnection included. The ports also contain most of an Omni-Pi-tent robot's sensors.

The docking hook mechanism is based on Hossain's RoGenSid design [73], for which he kindly provided original CAD files for the hook structure which were then adapted to the requirements introduced by the Omni-Pi-tent design. The hooks on Omni-Pi-tent were modified from [73] to increase the area of contact between connected hooks and hence reduce the stresses upon them, see Fig.(5.2). The hooks on Omni-Pi-tent are powered through a small DC motor operating between a pair of microswitches connected to the motor's H-bridge and used as end-stops, see Fig.(5.3). While the microcontroller operating these hooks does have a limit on the time for which it sends a signal commanding the hooks to open or close, it is the microswitch system which cuts power to the appropriate channel of the motor's H-bridge to stop it at either end of its rotation. The hooks rotate back-and-forth by around 45 degrees, in the locked position they can connect with another pair in the locked position, two pairs of hooks in the unlocked position provide some measure of pushing to force recently disconnected ports apart, see Fig.(5.2)and Fig.(5.4). The actuation takes 0.3 seconds, making Omni-Pi-tent's system one of the faster modular robot connector designs in use [42]. This speed increase over RoGenSid

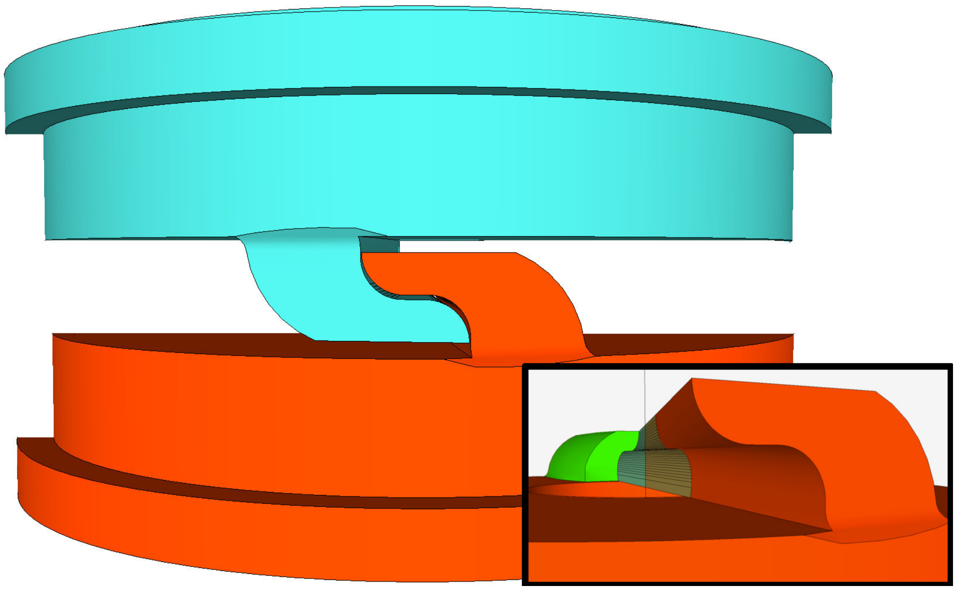


**Fig. 5.2:** CAD image of the profile of a connected pair of docking hooks. The profile steepens towards the centre so that if extended to the centre of rotation of the hooks it would become a vertical line, see the inset image where such hypothetical extensions of geometry towards the centre (pale yellow and turquoise) are shown extended from red and green hooks. This shape ensures that the hooks make contact throughout their width.

is partly due to using an internal spur gear mechanism rather than a high reduction worm gear, the shape of the hooks ensures that only minimal force is exerted on them by inter-module forces and hence a non-backdrivable worm is not needed. This internal spur gear also allows the motor powering the hooks to be located off axis, and the central region around the axis be kept free for static components to be located in. A clip of the docking hooks actuating is available at [213] and in the supplementary material (Docking_port_actuation_clips.mp4), Fig.(5.5) shows a time series from the actuation of another docking port.

The very nature of the hook mechanism makes rotation about the axis normal to the port face's plane a distinct possibility, hence four spikes and four corresponding pits act to prevent connected ports from rotating relative to each other and coming apart. The spikes and pits are conically shaped so as to aid in the final alignment of ports as they are pushed together during docking. As well as ensuring alignment, these spikes and pits serve to strengthen the docked connection against sideways loads, a pair of ports proved easily able to handle up to 59N in tensional and shear directions. Larger loads were not tested as, given the intended lifting capabilities of the robots' hinge mechanism, it was not expected that the robots would need to hold $> 6$kg against gravity in any part of this body of work. The surface of the spike cone is angled 45 degrees away from the port-to-port axis, this makes them steep enough that forces experienced when spikes and pits push together will be able to produce a significant sideways component, while also being shallow enough in angle to ensure that there is little risk of a spike connecting with the outer edge of a pit and being pushed further out of alignment if modules dock whilst one of them is approaching from a large sideways

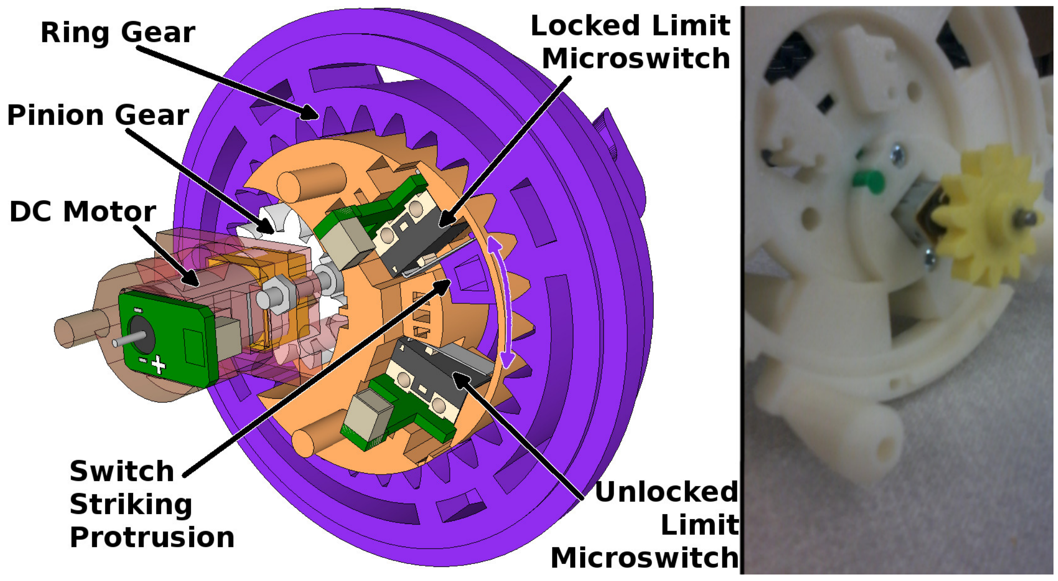


**Fig. 5.3:** The layout of the hook actuation mechanism, seen from behind the port face. Note the two microswitches at either end of an approximately 45° arc which are struck by the protrusion behind the hooks to stop the motor at the position limits. The photograph (right) shows the pinion gear as seen from the front as it is slotted in to place.

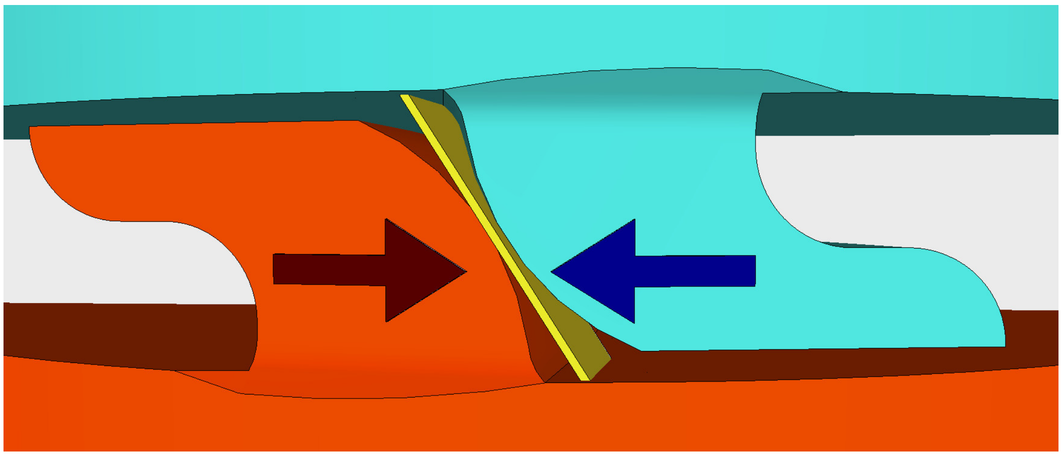


**Fig. 5.4:** The yellow plane shows the angle of surface contact between the rear faces of hooks as they unlock, the normal reaction force therefore pushes the docking ports away from each other as the hooks unlock.

displacement away from perfect alignment. This 45 ° angle on the spikes and pits also ensures a docked connection can be made from a robot coming in and turning as it does so, this is particularly useful when considering possibilities involving 3D reconfiguration of structures. Given that, unlike HyMod's [42], Omni-Pi-tent's docking ports do not retract, this shape is vital to ensure that ports can still correctly join if a pair of modules, acting as a meta-module [82], are “walking" across the surface of a large structure, see Fig.(5.6).

During testing of two robots for proof-of-principle of hardware docking during motion, see Sec.(6.5), it was observed that there was significant flexibility in the docked connections between two robots and a risk of the connection coming apart under vibration. The hook

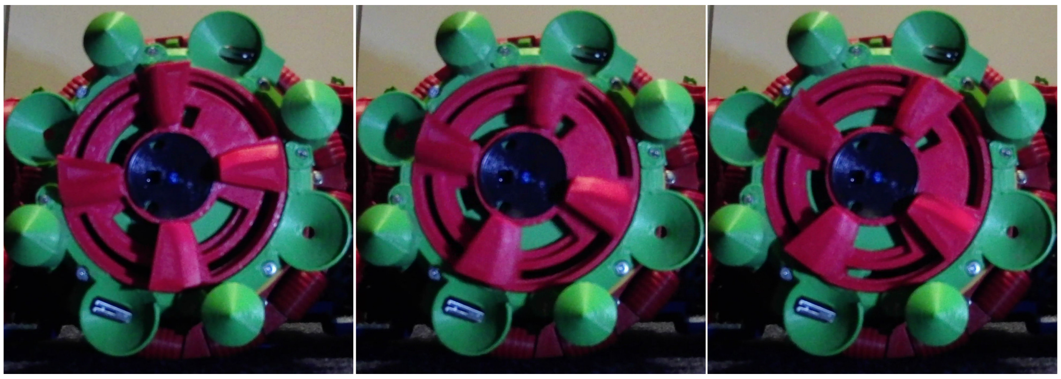


**Fig. 5.5:** Time series photographs displaying the docking hooks in motion, going from unlocked (left) to locked (right)

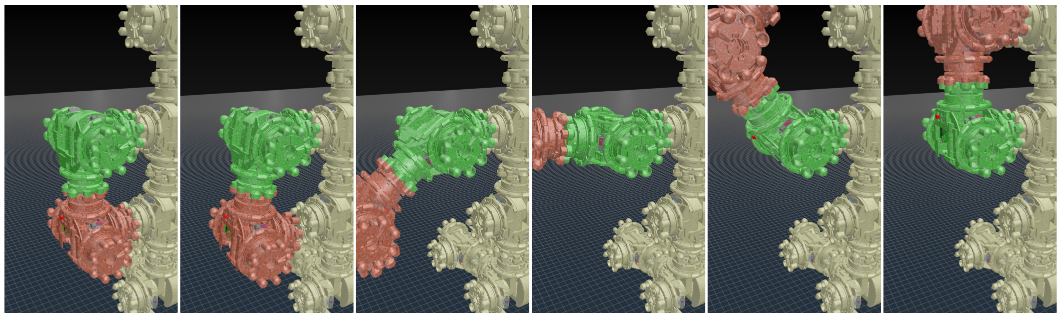


**Fig. 5.6:** Stills from a pre-development video of the Omni-Pi-tent concept displaying how a pair of modules are intended to be able to walk across a surface of other modules. In this example the pair connect with a series of ports so as to climb up the side of a tower. The concept of crossing surfaces in this fashion placed design constraints on the docking port geometry in regard to ensuring it could accept modules approaching while turning about a point centred at one lattice spacing out on two axes.

profile was therefore subject to some further modifications, and the tolerances around it modified, to ensure a more rigid link. It was also found useful to include forcible locking and unlocking commands for the hook state of the port. Whilst ordinarily the port ATMEGA keeps track of whether the hooks are locked or unlocked based on the previous command given to the hook motor H-bridge, there is no explicit sensing of whether the hooks are presently at this location, so if they are pushed in to an alternative position the port ATMEGA will not realise this. The commands to forcibly lock or unlock over-ride the port ATMEGA's "knowledge" of where the hook ought to presently be and ensure it sends the H-bridge command regardless, ensuring it is possible to get the hook back in to a known state.

Two of the alignment pits contain microswitches which are held closed when robots are docked together, these provide contact detection to aid in docking and let robots recognise when those around them disconnect.

The ports provide mounting for a ring of 4 phototransistors located around the rim, used as IR sensors for the docking guidance system. The IR LED which emits the guidance cone is located centrally for ease of navigation during docking. Beside the IR LED sits a TSOP38438 [123]

38KHz modulated IR receiver, located in the centre so as to maintain, both during docking approach and when connected, line of sight with the IR LED of a corresponding port Fig.(5.7).

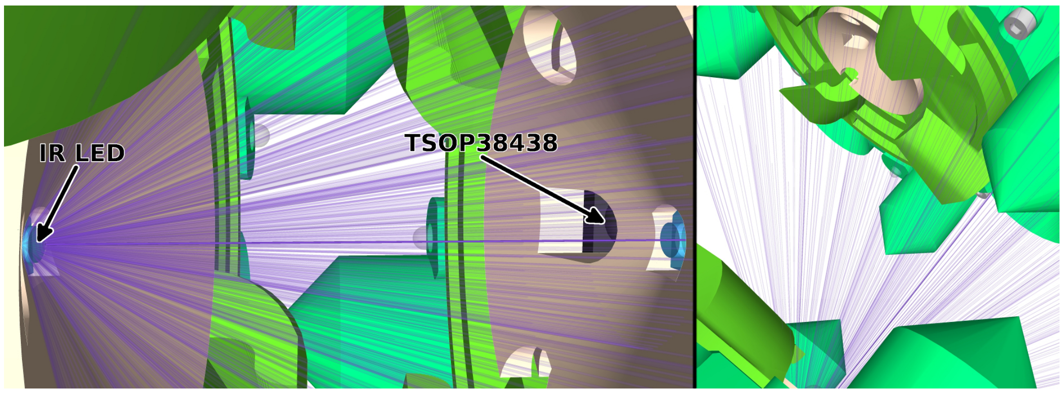


**Fig. 5.7:** A diagram showing the illuminated cone of a docking port IR LED, notice how line of sight between the IR receiver and LED are maintained both between ports while docking and once close together. Note also how each TSOP38438 receiver is shielded, by plastic parts, from being exposed to messages sent from the LED on its own port.

## 5.2 Docking Port Circuits

While aspects of the infrared docking system have been discussed in Sec.(4.5.6) there are significant details of the docking port subsystems still to be considered. Each docking port includes circuits for hook motor control, contact detection, infrared 38KHz communications, proximity sensing and docking guidance.

### 5.2.1 Infrared communications

Infrared communication between separated and connected ports is performed using an LED [121] as the transmitter and a Vishay TSOP38438 [123], a receiving device of a type commonly used for TV remote controls, as the receiver. The TSOP38438 detects bursts of 38KHz infrared and gives a digital output during periods when they are present. Using a modulation detecting digitising receiver of this kind simplifies the design of other circuitry required in the IR communications system by avoiding the need to provide amplification and other functions using separate components, it does however give limitations such as not getting an intensity reading from which the range to the emitting LED could be calculated. The TSOP38438 is designed to filter out strong and periodic noise away from the 38KHz modulation frequency, however it is still vulnerable to false readings from the 5KHz docking guidance signals, although not so severely as to make it un-usable. The TSOP38438 is connected to an interrupt pin on the port ATMEGA microcontroller which is used to detect messages as they arrive.

Messages are sent as a repeating broadcast until the sending port is instructed by the robot's high level controller code, running on the Raspberry Pi, to cease transmission or to begin sending a different message. It is not guaranteed that any specific sending of the message will be received, rather messages are repeated, increasing the probability that copies of the message will be received correctly. Messages are carried using a pulse length encoding scheme, after an initial starting sequence of timings to let a receiving ATMEGA know a message is on the way, 1s are encoded as longer periods (33 square waves) modulated at 38KHz, 0s with shorter periods (11 waves). As the infrared data stream can provide only a single “channel" of digital output a separate clock signal cannot be sent, hence these messages are self-clocking. Each message has a fixed size of 17 bytes, for which the last two are reserved for a CRC checksum to verify that received messages are correct and ensure corrupted messages are discarded without causing erroneous actions. The Port ATMEGA confirms the CRC checksum matches the received message without involving the Pi, hence corrupted messages can be discarded immediately without wasting time relaying them to the robot’s main processor. During the time when a message is being sent or received the Port ATMEGAs sending and receiving it cease other tasks and ensure 5KHz emission does not emanate from either port. There is no MAC protocol used except in that a port will not start emitting when it is receiving, and ports wait a pre-defined interval between resending of messages. This interval can be adjusted to be longer for messages of lower priority, as decided by the robot’s high-level controller code. The interval is also modified by the addition of a randomised percentage of the defined value, ensuring that when multiple ports are operating they cannot remain “in phase" with one-another such that one’s sending repeatedly starts at the same time as another’s.

By placing the emitting LED in series with an array of three different valued resistors it is possible to vary the power of the message sent so as to use short, medium and long ranged modes, although in the presence of 5KHz IR illumination the automatic gain control of the TSOP38438 receivers sometimes adjusts such that detecting the shorter ranged modes can be difficult. Fig.(5.8) shows this arrangement.

### 5.2.2 Analogue Infrared

The proximity sensing and guidance functionalities are handled by a separate system from the IR communications, a different modulation frequency, 5KHz, is used to reduce conflict between the systems. Where the 38KHz communications system provides a digital ON or OFF value regarding the presence of illumination, the 5KHz system gives an analogue level. Similarly to the 38KHz system, there is an independent copy of this system within each docking port, again being controlled from the port ATMEGA microcontrollers.

When a port wishes to recruit other robots to connect to it a 5KHz signal is emitted from the IR LED, the same LED as used at other times for 38KHz communications. This 5KHz signal is generated by the LEDs ATMEGA, on the central circuit board, and can be started and stopped, for each port, on demand via a wire from the Port ATMEGA to the LEDs ATMEGA. Sourcing

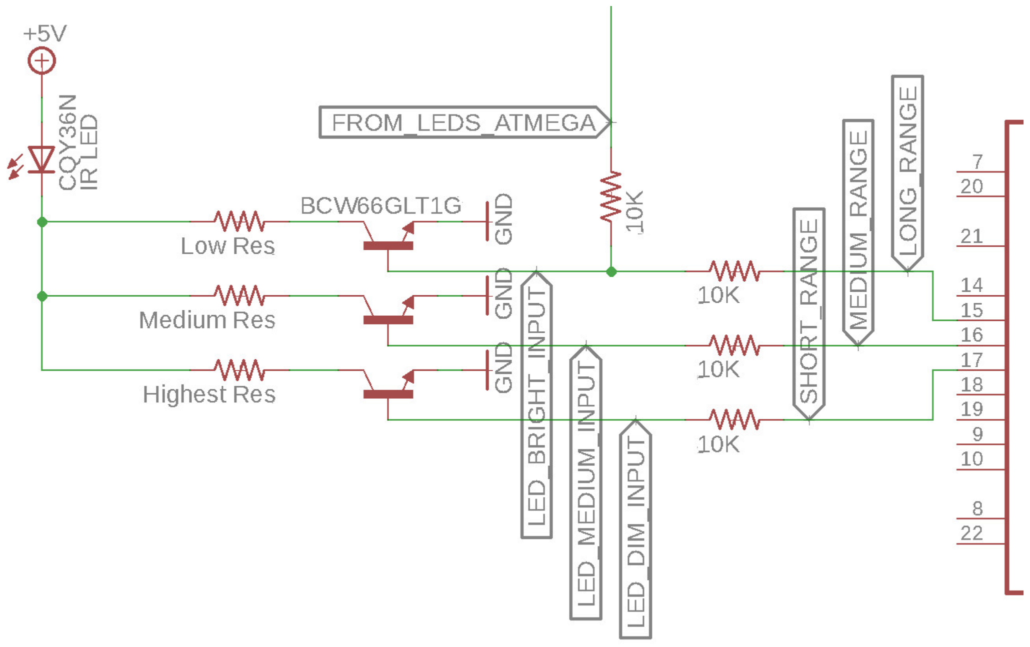


**Fig. 5.8:** Circuit diagram showing the resistor arrangement used to allow port-to-port IR LEDs to be operated at different power levels. Note how the LED can emit with different strengths depending on which microcontroller pin is used to control it. Note also the extra input used to allow the LED to be flashed at 5KHz by the LEDs ATMEGA.

the 5KHz signal from a device separate to the Port ATMEGA frees the Port ATMEGA to handle other tasks. This 5KHz signal is always emitted in the highest power mode.

This 5KHz IR system uses phototransistors (L-93DP3C [122] ) to measure the strength of IR illumination which, after filtering, amplifying and other pre-processing provides sensor data which the Omni-Pi-tent robot can make use of. Fig.(5.9) illustrates the analogue receiver circuit.

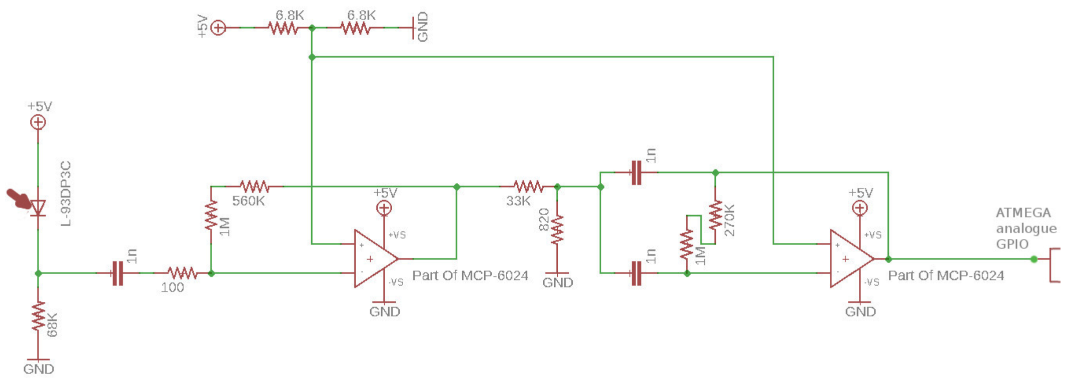


**Fig. 5.9:** Circuit diagram showing the analogue receiver circuit for the 5KHz system. This circuit was arrived at after much experimentation with a wide variety of different designs, most of which had comparatively poor performance.

Four copies of the analogue infrared detection circuit are included on each Port PCB, one copy for each of the four phototransistors arranged around the rim of the docking port. These feed to four of the analogue input capable pins on the Port ATMEGA.

Each phototransistor is used in common collector mode with a resistor to ground being used to take the output low when no light is present. Early testing found that 68K$\Omega$ was the optimum value for this resistor so as to ensure a discernible output in the presence of infrared light whilst also avoiding saturation. Testing also established that the L-93DP3C phototransistors had rise and fall times slow enough that they could not correctly show square wave signals of 10KHz and above, it was for this reason, as well as avoiding harmonics due to 38KHz signals, that a frequency of 5KHz was selected as the highest frequency easily usable for the analogue infrared system.

The next stage of the circuit in Fig.(5.9), from the 1nF capacitor through the first op-amp, serves as a highpass amplifier to remove DC effects, such as daylight, and relatively slowly varying periodic signals, such as from mains powered lighting in the room. The cut-off frequency of this amplifier, $f_{cut} = 1/(2\pi C(R_1 + R_2))$ with $R_1$ the 68K resistor below the phototransistor and $R_2$ the 100$\Omega$ resistor on the input, was set at around 2.3KHz to eliminate in particular any 50Hz or 100Hz noise. The ratio of the 100$\Omega$ $R_2$ to the 1.56M$\Omega$ resistance between the inverting input and the op-amp's output gives a substantial amplification of $H_0 = 15600$. Clipping does occur in this op-amp however as the square wave structure of the signal is still preserved this distortion does not affect the operation of the circuit. Whereas in many example op-amp circuits a dual rail power supply is used, the robot can only supply voltages in the GND to +5V range to the port PCBs, hence a 2.5V virtual ground is attached to the non-inverting input.

The output of the highpass amplifier feeds to a bandpass amplifier tuned to give a tight peak, bandwidth $B$ =250Hz, at $f_0$ =5KHz. This amplifier provides rather less amplification, $H_0$ =20, than the earlier stage. Both the highpass and bandpass amplifiers are designed to use only 1nF capacitors, simplifying assembly of the circuit boards. From these performance specifications Eq.(5.1) allowed the resistor values to be calculated, with $R_4$ being the 33k$\Omega$ on the input to this amplifier, $R_5$ being the 820$\Omega$ pulldown and $R_6$ the 1.27M$\Omega$ between the op-amp's inverting input and output. The same 2.5V virtual ground is used on the non-inverting input.

$$\begin{aligned} Q &= f_0/B \\ R_6 &= Q/(\pi f_0 C) \\ R_5 &= R_6/(4Q^2 - 2H_0) \\ R_4 &= R_6/(2H_0) \end{aligned} \tag{5.1}$$

The filtered and amplified output is fed to an analogue input pin on the Port ATMEGA. This microcontroller reads its analogue input pins so as to determine if 5KHz signals are present, and if so their strength. On the Port ATMEGA software is run which puts the ADC in “free running mode" and each time that the ADC completes a reading code is run which performs a homodyne measurement of the input signal strength by multiplying each incoming ADC value by a pair of 5KHz square waves in quadrature. The result of these multiplications are put through a software low-pass filter and stored. As the ADC readings on an ATMEGA328P can be prone to noise for the first few readings after the ADC has been switched between different input pins it is not possible to read each pin after another on each cycle. Hence each pin is read repeatedly for 300 microseconds before a “power" reading is taken as the sum of squares of the two low-pass-filtered quadrature waves, then the ADC is switched to the next input pin so as to read from the next phototransistor around the port’s rim. To ensure effective operation for sampling a 5KHz signal the ADC is set to a slightly lower accuracy mode able to take subsequent readings at 76.9KHz. The “power" readings correlate to the strength of the incoming IR signal, however do not do so linearly, the various stages of electrical amplification, plus the details of the software homodyne detector, were not designed so as to ensure a linear mapping. As the infrared system is used to measure relative strengths, not calculate the exact flux of incident IR light, it was considered sufficient to simply calibrate the, unitless, output digital values against the distance to the emitting LED.

The performance of the infrared guidance system has been discussed in great detail in Sec.(4.5.6), however for reference it should be repeated that the IR guidance beacon can be reliably detected at ranges of up to 68 cm.

To operate in proximity detection mode rather than for guidance this system is used to detect reflected light from the LED on its own port, rather than ambient light levels. The relative positions of the port’s IR LED and phototransistors ensure that there is no direct line of sight between them, and the 3D printed plastic parts immediately around the IR LED are coloured black to minimise any chance of them reflecting IR light on to other parts of the port from which they might subsequently reflect such light in to the phototransistors. Hence all detected IR radiation is reflected from objects in the environment, not from direct illumination between the LED and phototransistors of the same robot, or spurious reflections off the port’s spikes or hooks. When using the analogue infrared systems for proximity sensing the size of objects in view strongly affects the range at which they can be detected, as does the colour and surface finish of them, however for detecting other robots with strong hued 3D printed plastic parts ranges of 15 cm are around the furthest practical. Given the relatively short range of this IR proximity system, problems with spurious bright reflections from shiny objects in the environment were not found. When in proximity mode the Port ATMEGA commands the LEDs ATMEGA to supply a 5KHz signal to the LED, the LEDs ATMEGA responds within a quarter of a 5KHz cycle and the relevant port’s IR LED emits a 5KHz signal. To avoid causing confusion with docking guidance cones the 5KHz is transmitted for only a 1.3ms burst per 117ms period when used for proximity detection. A port which is not recruiting will usually, unless ordered otherwise by the Raspberry Pi, gather these proximity readings at about 9Hz and spend times between them gathering ambient IR readings from which to detect recruiting cones. A port which is recruiting will calculate proximity rather than ambient IR readings most of the time, however a mode can be activated within the Port ATMEGA's software which briefly turns off the guidance cone emission on a small fraction of program cycles and hence allows occasional ambient readings to be recorded.

## 5.3 Docking Port PCB

Each docking port has its individual circuits contained on a PCB located behind it, all ports use the same design of PCB for this, simplifying design and production of robots. This PCB connects to the central PCB via a 14 way connector wire, and also to the sensors and actuators of each port. As there are 4 of these PCBs in the robot each of them also provides the H-bridge for a wheel motor. Fig(5.10) shows photographs of this board design. An ATMEGA328P microcontroller is fitted on each port PCB to run the real-time operations of that port, all port sensor data is collected here, before being relayed to the Pi. Except for the wheel motor all actuation capabilities related to this PCB are also controlled from this ATMEGA.

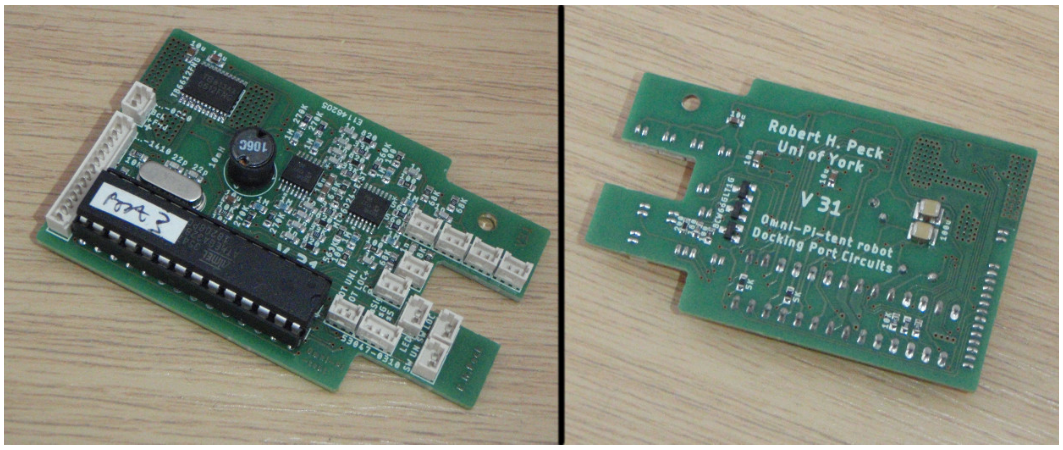


**Fig. 5.10:** A Port PCB from an Omni-Pi-tent robot.

A Pi filter, see Fig.(5.11), is used to prevent voltage ripples caused by 5KHz LED emission from propagating in to the power rails of the amplifier circuits and generating spurious proximity readings.

In the completed Omni-Pi-tent robot any ripple effects are further reduced by having the 5KHz modulated wave for the IR LEDs supplied by one of the ATMEGA chips on the central PCB, the 5KHz waves for each port's LED have phases shifted $90°$ to reduce any possibility of 5KHz ripples in the power supply.

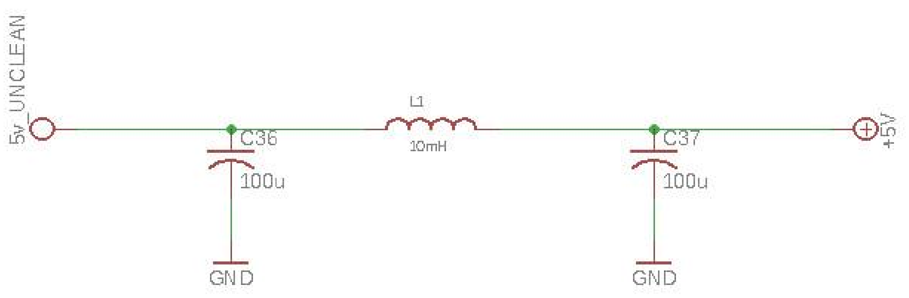


**Fig. 5.11:** The Pi filter as included on the port PCB, voltage is supplied from one side of the inductor (left), the output on the other side (right) has any voltage ripples present in the incoming supply heavily damped out.

## 5.4 Omni-Wheels

Omni-wheels are used to give Omni-Pi-tent modules the capability to maintain orientations independently of the driving direction. Although 3 of these would be enough for the same capabilities, and would present less risk of some wheels spinning clear of the ground when on uneven surfaces, the Omni-Pi-tent design uses 4 as this is more practical to implement in a way which matches the 4 fold symmetry of the docking ports, with one wheel per port. The wheels are deliberately large, similar to the non-omnidirectional wheels on HyMod [42], such that running the robot upside-down is possible.

Each wheel has 12 rollers around the edge, running on 3 mm steel axles which are clamped between two 3D printed wheel pieces, Fig.(5.12) shows this arrangement. The rollers have a profile designed to conform to an overall circle centred on the wheel, with some ridges distorting it, see Fig.(5.13). Smoothing of the ridge profile was used to prevent stalls which could propagate large and potentially damaging forces on to the wheel motor reduction gearboxes. Most omni-wheels have two rows of rollers to ensure that a roller is always touching the ground at any wheel angle, however for weight and space saving reasons the omni-wheels on Omni-Pi-tent use a single row of 12 rollers, the wheel therefore does encounter 12 bumps during each rotation, but performance is not affected.

Power is supplied to the Omni-Wheels via an internal gear arrangement from a 5V DC motor, see Fig.(5.13). This internal gearing allows the wheel to rotate around the axis of a docking port, but for static objects to be fed through it's centre. The docking ports therefore do not rotate with the wheels, as those on HyMod [42] do, and hence there is no need for port angles to be correctly rotated before docking, they always stay in the desired orientation regardless of wheel position. This internal gear also acts to slow the speed of the wheels, compared to the motor shaft, and increase the torque, by a factor of ≈4. The ratio between the tooth counts of the motor pinion and ring gear is a non-integer value, ensuring wear is spread more evenly over this almost constantly moving part of the robots. The motor is mounted on the

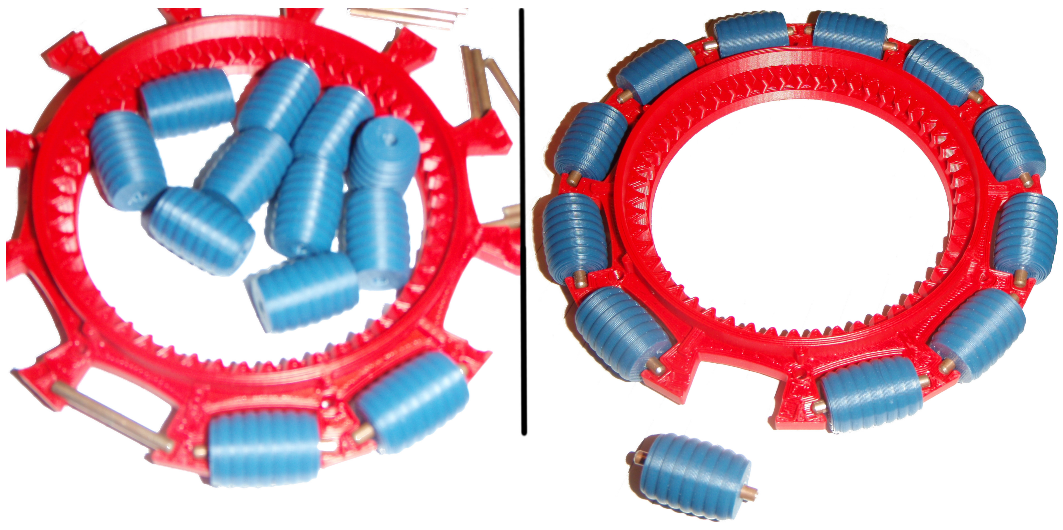


**Fig. 5.12:** Photograph of a wheel under construction, note the metal axles for the rollers and the depressions in the lower half of the wheel to hold their ends.

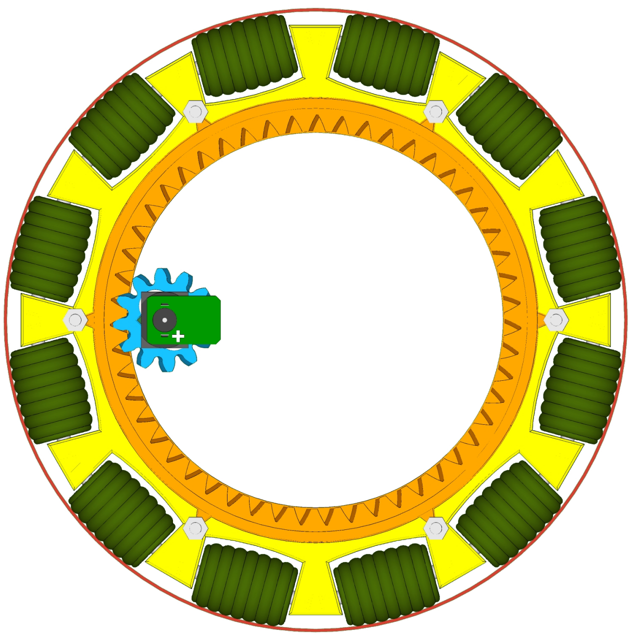


**Fig. 5.13:** The profile of the rollers on an omniwheel as compared against a circle (red) centred on the wheel's axis. Note also how the wheel is powered with an internal gear mechanism, ensuring a gap through the middle is available for wires and other items to feed through when connecting from the port faces to the core of the robot.

outward, from the robot's centre, side of the wheels, as space on the inward side of the wheels is needed for hinge mechanisms.

Encoders for these wheel motor were omitted, despite their possible usefulness, as their spatial requirements around the DC motor backshaft would have forced the port faces to be mounted further outward from the robot's centre to make room for them.

### 5.4.1 Improving the Wheels

Some problems with the wheel design were discovered when printer tolerances changed, see Sec.(5.15). This was particularly apparent during testing of the first production robot, where wheel motors were close to stall at all times when driving. Possibilities of weight increases between the second prototype and first production robot were ruled out as the cause of performance loss as the second robot continued to operate well when a weight was placed atop it. It was observed that the wheel tolerances, while tight for wheels produced on the Robox printers, were loose when Stratasys printed parts were used. The wheel motor's output spur gear's teeth were often misaligned to the position of the wheel's internal gear's teeth. Furthermore, during rotation in one direction, the wheel motor was effectively having to push the wheel down and lift the whole robot up because of this tolerance issue. This was further complicated by the fact that straight spur gears, as initially used on the wheels, experience a specific instance at which a new tooth makes contact all along the face.

The solution was to place a narrow laser cut ring of acrylic between the inner edge of the wheel and the outer rim of the mount, this substantially reduced friction while maintaining a tight tolerance with little opportunity for misalignment. The spur and ring gears were modified to herringbone, double helical, forms such that sections of more teeth were in contact at any given moment, smoothing the motion. Fig.(5.14) displays a comparison of the two wheel mounting and gearing arrangements.

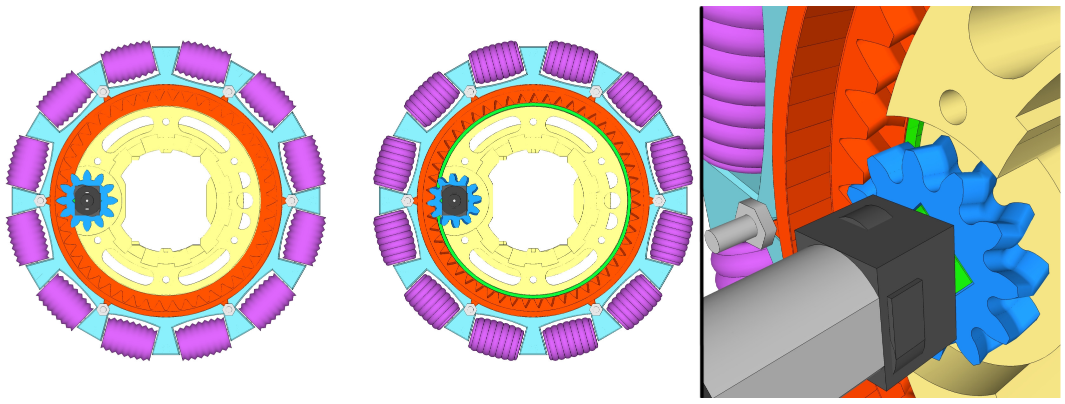


**Fig. 5.14:** Improvements made to the omniwheel design included the addition of a laser cut acrylic ring (green) to reduce friction and a herringbone pattern on the gear teeth, the later design (middle) is contrasted to the previous arrangement(left). With a close-up of the herringbone teeth also shown (right).

### 5.4.2 H-Bridges

For simplicity of PCB design, and to allow wider traces able to handle higher currents, the entirety of an Omni-Pi-tent robot, excepting some later modifications, is powered from a single 5V rail. The Raspberry Pi Zero W operates from this voltage, as do the ATMEGA328P microcontrollers and the logic for other analogue and digital ICs. The motors used are rated at 6V, however 5V was used to match with the power supplied to other components.

Motor control is achieved with TB6612FNG H-bridges containing MOSFETs with low voltage drops. Use of a low voltage drop H-bridge was vital to achieve useful performance from motors while still only needing a single 5V supply.

## 5.5 Initial Hinge Concept

Omni-Pi-tent robots use a 2 DoF hinge similar to that within SMORES robots [35], unlike SMORES worm gears were initially used to theoretically increase the torque of the motor by a 45:1 ratio and ensure that the hinge could not be backdriven and hence maintained position under load without needing motor power to be applied. The SMORES-like bevel gear arrangement, as well as providing 2 DoF in a convenient way has the advantage that the torques of all the hinge motors can be combined for both pitch and roll motions, the different motions resulting from whether the motors are run together or in opposition. During motion Port 4 remains on the surface of a spherical region, easily forming cubic lattice structures. Port 4 can pitch within a range of $\pm 90°$, although the gears could conceivably enable roll across an infinite range the need to avoid tangling of the wires between the Central PCB and Port 4 PCB cause the roll to be restricted to $\pm 90°$. Ports 1, 2 and 3 do not have rotational DoFs due to the space required for the 3D printed gearing necessary to produce torques useful for a robot of this size and mass.

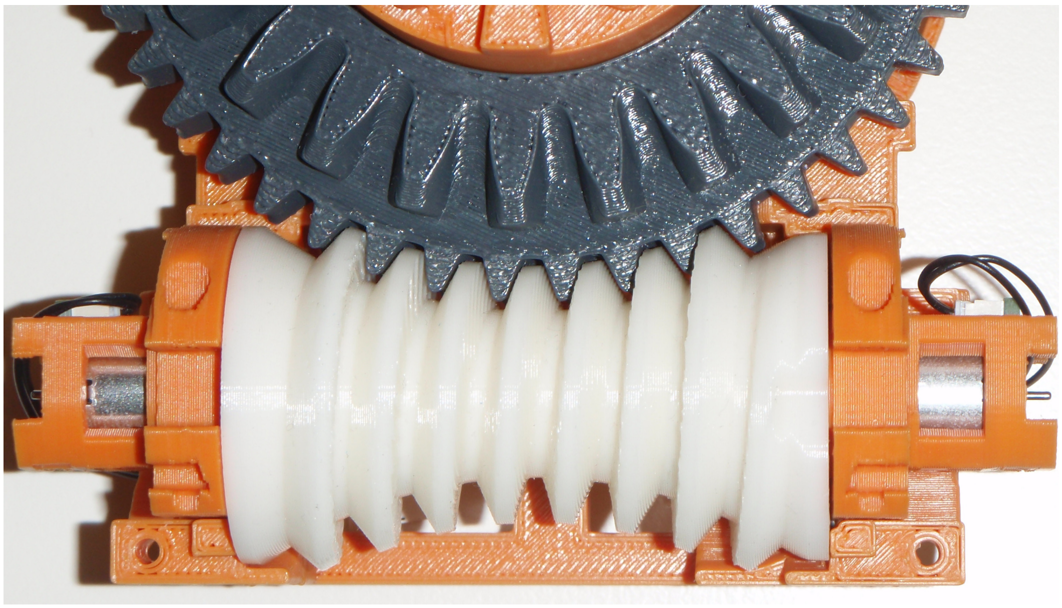


**Fig. 5.15:** Photograph of the 3D printed worm gear and worm wheel, note the globoid shape of the worm ensuring multiple worm teeth and gear teeth can be in contact at any time.

The bevel gears, especially their teeth, were as large as possible to provide maximum strength in 3D printed plastic. A double enveloping, or globoid, worm gear was used as this ensured multiple gear teeth would interlock the worm at once and stresses would be better spread. Fig.(5.15) shows this design.

To further improve torque each worm gear was powered by a pair of two 5V DC motors, one above and one below. This pair of motors both connected to the worm and did not use any kind of differential gearing to adjust for non-ideal differences in their speeds, due to variability in the motor's manufacturing, when supplied with the same voltage. Rather they were both directly connected to the worm gear and it was established by testing that even if the motor speeds were slightly mismatched there was not a significant reduction in torque, at the worm gear, below twice that of a single DC motor. The final output of the hinge, via the bevel gears acted, and in later designs still does, as a differential gear so any mismatch in motor speeds between the left and right worms was compensated for.

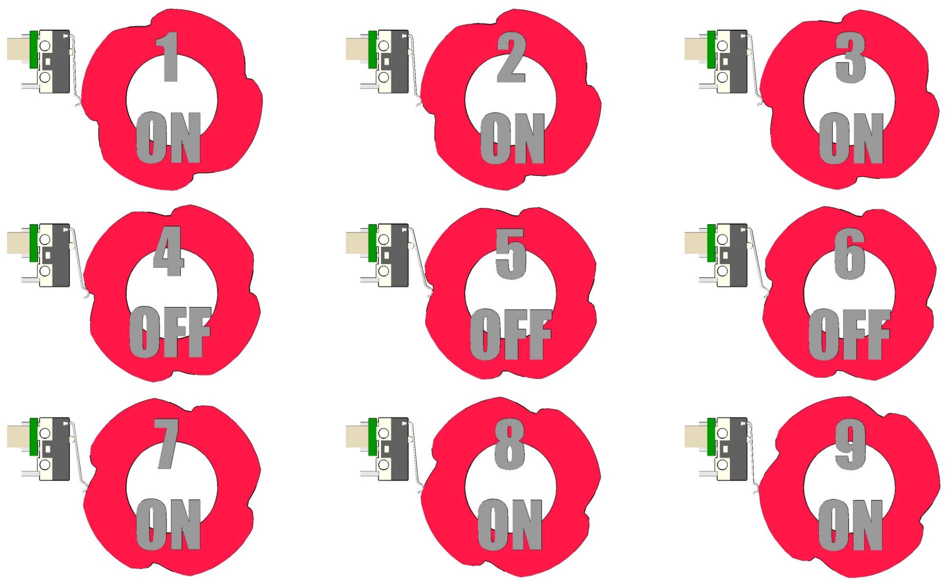


**Fig. 5.16:** CAD diagram of the notch shape cut in to the worm gears to tap against the position monitoring microswitch, note how the notch is shaped to ensure that the lever is pressed in to the switch body as the worm turns in either direction and how shallow angles have been used to avoid pushing the microswitch lever "upward".

Position monitoring was originally planned to be with commercially purchased rotary encoders, but these proved impractical as all models available had to be mounted on-axis. On-axis mounting was impossible if the encoders were connected directly to the bevel gears, as these rotate about a hollow axis in which the battery compartment is located. It was also impossible to mount such encoders on the worm gears as their axes were occupied, above and below, by the pair of motors. Hence microswitches were used for incremental encoding. A microswitch was mounted beside each worm gear and shaped notches at the end of each worm cause a tap and release of the switch during each rotation of the worm, see Fig.(5.16). This notch was specially shaped for compatibility with the DM1-02P-30-3 [120] "simulated roller" subminiature microswitches used throughout the design, with the necessary angles of entry and exit slopes to push the switch inwards without catching mapped in to the cylindrical frame of the worm gear to form the shape. As the hinge was non-backdrivable it was considered safe to use a single channel of incremental encoding for each worm gear.

### 5.5.1 Initial Hinge Tests

Calculations based on the motor data-sheet's stall torque figure and the ratio of the worm gearing predicted Omni-Pi-tent's hinge to have a stall torque of 93.6 Nm, well above the necessary torque for lifting a two-module chain. The 2 DoF hinge on the first prototype was tested with minimal loads, testing larger weights was not feasible with the equipment and printed parts available at the time. With the load of Port 4 itself, the ability of the hinge to rotate the robot's fourth docking port through both pitch and roll degrees of freedom, both for independent and simultaneous motions, was verified Fig.(5.17). The hinge was indeed, as designed, not backdrivable. The hinge was found to draw peaks of 240 mA of current, with an average close to 220 mA, 55 mA in each of the four motors.

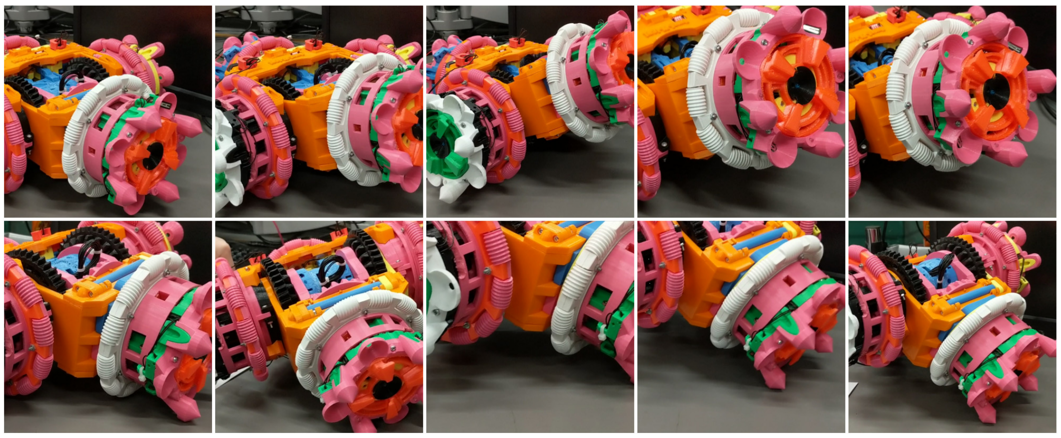


**Fig. 5.17:** Testing of the hinge system with a partly assembled robot and temporary external driver circuitry. The upper series of images shows the hinge elevating Port 4's pitch to 45° then rolling Port 4 clockwise. The lower series shows the robot raising itself up by pitching the hinge downwards, even while lifting the weight of the robot, currents did not rise above a 240 mA peak. A video of the test's highlights is available at [212] and in the supplementary material (Early_hinge_test.mp4).

A later test with the first production robot, with the second prototype available to be lifted, attempted to have the second prototype robot lift a single robot from infront of to above it, Fig.(5.18) shows the attempt. With the robots weighing 3.15 kg at this point in the design iteration, this corresponded to a torque of 902 Ncm needed to lift a robot at a distance of one lattice spacing (29.2 cm). The test showed it to have woefully poor performance, with all 4 DC motors driving the hinge stalling immediately. The load was removed from the hinge and the hinge allowed to climb to higher angles before reattaching the load and resuming testing. It was found only to produce enough torque to power the hinge, with a single module load,

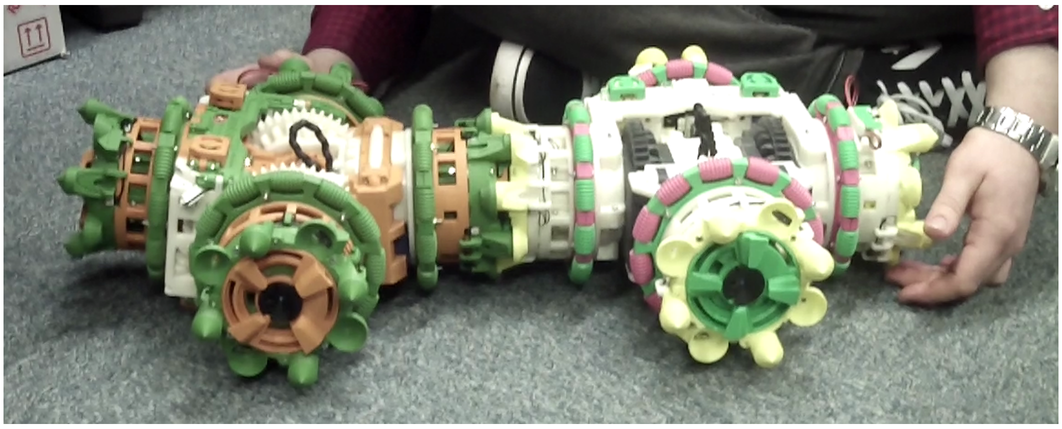


**Fig. 5.18:** A test in which a green and orange robot (left) attempts to lift a white and yellow robot (right). During the attempt the robot performing hinge actuation was held down to counterbalance it and ensure it would lift the other robot rather than rotate itself while leaving its own Port 4 static.

once it had risen above 82 °. From this it can be calculated that the worm gear-based hinge had a maximum torque of 125 Ncm ($902cos(82°)$). It was concluded both that the worm gear mechanism was inefficient, and that the datasheets for the type of small motor used in the early hinge design were misleading. Testing a single DC motor of that type with a barrel-winch type jig mounted directly on the shaft and a weight hung from the winch found them to stall at much lower torques than indicated. The 3D printed worm gears, whilst as strong as intended and as non-backdrivable as required, were found to be spectacularly inefficient at high torques because of how the terraced layers of the 3D printed worm interacted with the gear teeth to cause brief periods of very high friction. Sec.(5.16) describes the alterations made in later iterations of the design to overcome the limits of these worm gears.

## 5.6 Power and Battery

Excepting the revised hinge motors, see Sec.(5.17), the entire robot is powered from a regulated 5V rail. This voltage was chosen due to general compatibility of most of the IC chips within the robot, compatibility with easily available USB power for subsystem testing and being close to the rated voltage of the micrometal DC motors fitted. Using regulated power ensures consistent performance throughout the draining of a charged battery, avoiding the problems some robots have where performance, especially of motors, changes throughout the discharge profile. Regulation is achieved with a step-down switching regulator able to supply up to 5A at 5V [77]. This regulator comes as a standalone board, space is saved on the Omni-Pi-tent robot's central PCB by mounting it via header pins, see Fig.(5.19). A higher voltage battery pack, higher than 5V plus the regulator's dropout voltage, is required to power this. Switching down through a regulator, rather than up, also has the advantage of, for any given current in the 5V load, reducing the current which the battery pack must supply.

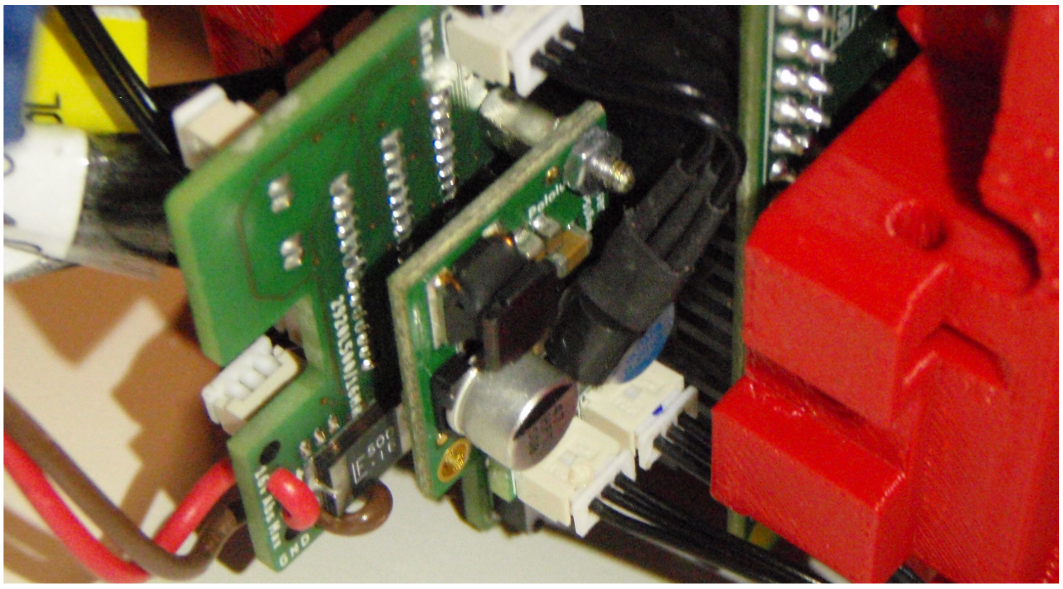


**Fig. 5.19:** In-situ photograph of the regulator board and polyfuse, lower left, involved in supplying power to the central PCB. Note also the temperature sensor hanging from above to monitor the regulator, and the two 3 wire breakout connectors below to supply power to the hinge motor PCBs.

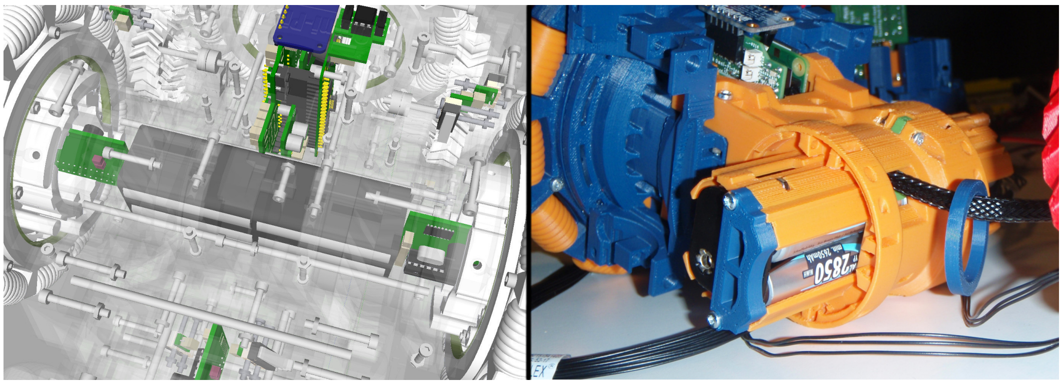


**Fig. 5.20:** The battery location within an Omni-Pi-tent robot, an image from the CAD design (left) is presented alongside a photograph from the assembly of a production robot (right).The battery is marked in dark grey in the CAD image.

Nickel Metal Hydride (NiMH) cells were chosen for the pack due to the simplifications which their inherent safety over Lithium ion batteries brings to the design. NiMH cells allow the pack to be both discharged and recharged without the need for complex protective circuitry, and can be, unlike lithium cells, safely arranged in series combinations without the need for cell matching. Eight NiMH cells are used in series to provide a theoretical 9.6V throughout most of the discharge profile. Given the simplicity of combining NiMH cells these are able to be physically arranged in a $2 \times 2 \times 2$ formation which slots in to the robot's core within the axis of the hinge gears, see Fig.(5.20). The NiMH cells have a discharge curve which reduces slowly until a sharp voltage drop when they are close to empty. Fig.(5.21) shows the discharge profile in terms of the per-cell voltage of the 8 cell pack. A series of tests with battery packs, discharged both by actual use in a robot and in breadboard test rigs, found an optimum point to cut the batteries off at 1050mV per cell. This cut-off value also has the advantage that if the cells are unequally drained, it will cut off before any one cell could be subject to damaging reverse charging from the others. The approximately 2 hour battery life achieved by Omni-Pi-tent when using this 1050mV per cell cut-off level is impressive in comparison to other similar modular robots [21].

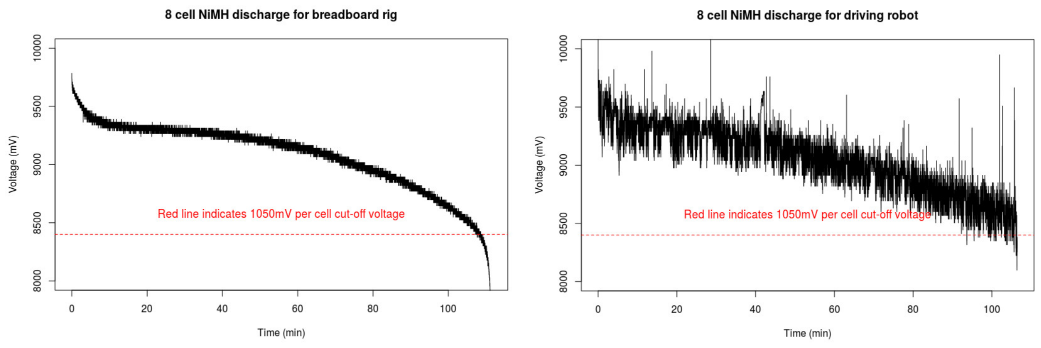


**Fig. 5.21:** The total voltage of an 8 cell NiMH battery pack. On the left, the cleaner curve is derived from an experiment discharging the battery through a test rig involving a 5V regulator powering high current resistors calibrated to draw a 0.9A current similar to the average when driving of a full robot, this discharge continued until the battery voltage was too low to power the regulator. On the right, the noisier curve shows the discharge within a robot performing random wandering on a carpeted floor, this discharge continued until the under-voltage cut-off occured. Notice how the value, 1050mV per cell, for the under-voltage cut-off is set to give robot running times close to 2 hours, while shutting down before any risk of cumulative long term health damage to the battery or regulator. Lower voltage cut-off values would barely increase running time.

A 5A holding polyfuse was placed in series with the battery and regulator, to protect against over-current. An MCP9701A temperature sensor overhangs the regulator to provide extra monitoring and emergency shutdown capability against over-current conditions, it is monitored by an ADC input on the Motors ATMEGA chip. Another ADC pin on the Motors ATMEGA reads the voltage from a potential divider, with high value resistors to minimise current draw, which gives a known fraction of the battery terminal voltage allowing the robot to measure its own battery state. Powering the robot on is handled by use of non-locking push button switches, using either the upper or lower switch as either may be obstructed when the hinge is at a raised or lowered position, to briefly supply a raised voltage from the battery to an otherwise pulled-down regulator enable pin. Once the LEDs ATMEGA is powered it uses an output pin to keep holding this enable pin high after the on-switch is released. Shutting down occurs either when the Motors ATMEGA detects a temperature fault or under-voltage condition and signals the LEDs ATMEGA, or the LEDs ATMEGA itself detects an off-switch, again positioned as a pair, being pressed. Software on the Pi can also supply a command to initiate shutdown. The LEDs ATMEGA then sends the enable pin low, powering down the regulator.

Recharging is achieved with a common commercially available 3 to 10 cell 800mA multicell NiMH charger, fitted with an adapter wire to convert from the charger's jack to a lead compatible with the battery pack, recharging takes under 2 hours. The battery lead is routed out to a connector near Port 3's wheel where it can be easily swapped between powering the robot or connection to the charger.

## 5.7 Compass and Accelerometer

As outlined in Sec.(3.3.3) while the robots do not require global GPS-style positioning they do need to know their orientations relative to a global frame, atleast for robots within a few module-lengths separation from one-another. Omni-Pi-tent robots use a BNO 055 sensor [78] which had been intended to combine a magnetometer with accelerometer and gyroscope MEMs devices so as to provide absolute orientation sensing with partial protections against the indoor limitations of a magnetometer compass alone. In the normal operation mode the BNO 055 fuses magnetometer, accelerometer and gyroscope data internally to provide simply an $x$,$y$ and $z$ angle for the sensor relative to North. It also allows direct reading of the accelerometer allowing robots to establish which way up they are. Remember that an Omni-Pi-tent robot's wheels are sized such that it can drive upside down.

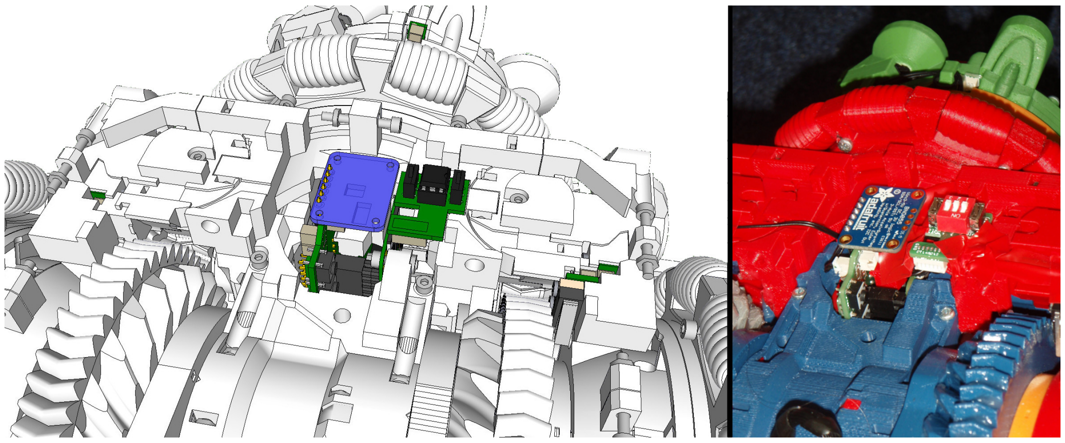


**Fig. 5.22:** The compass location within an Omni-Pi-tent robot, an image from the CAD design is presented (left) alongside a photograph from the assembly of a production robot (right). The compass board is marked in blue in the CAD image.

Sources online warned of complications [79] in interfacing a Raspberry Pi directly to the BNO 055 over I2C due to the non-real-time nature of a Raspberry Pi and due to known difficulties with Raspberry Pi I2C clock stretching [80], therefore the BNO 055 is arranged as an I2C slave to the Hinge ATMEGA. The BNO 055 is fitted to the same I2C bus as all other I2C slaves within the robot, the Hinge ATMEGA using a number of negotiation lines to the Pi to promote itself to master of the BNO 055 only while the Pi is not acting as I2C master to the Hinge ATMEGA or any other device on the robot's I2C bus.

The compass is located above the central PCB of the robot, see Fig.(5.22). This keeps it well away from metal strengthening bars within an Omni-Pi-tent robot, far from the high current PCB traces near the voltage regulator and well away from the hook and wheel DC motors, all of which could provide sources of magnetic interference.

The BNO 055 was not perfect in avoiding magnetic interference, even in magnetically clean environments there was still a need for the compass to be calibrated regularly during robot operation if any small magnetic anomalies caused a loss of accuracy. The BNO 055 sensor reported a confidence value which varies between 0, uncalibrated, up to 3 for high accuracy. When the value was low calibration, or recalibration, was required and the robot could not trust headings supplied by the compass. In early experiments with the robot this calibration was done manually by lifting the robots and rotating them to enable the BNO 055 to enter a self-calibration routine whenever the robot reported, over audio, a low confidence value. It was attempted to remove this laborious manual step by allowing the robot to restore compass calibration for itself using its own actuators. Self-calibration of the compass was first attempted by having the robot spin on the spot, in either direction for a randomised time period, until the confidence value was again high enough for useful navigation. Unfortunately the BNO 055 was not able to calibrate purely from rotation in a single plane as caused by wheel rotation. An alternative method to calibrate the compass was therefore sought, as described in the following section.

### 5.7.1 Magnetic Offset Calibration

The BNO 055 was put into a magnetometer only mode with the self-calibration deactivated, and $B_X$, $B_Y$, $B_Z$ component readings of magnetic field strength were recorded while the robot was rotated by hand about all axes. The magnetic field components read being a combination of the effects of magnetic distortions due to items within the robot, and the effects of external global magnetic fields. The manual calibration method used allows those magnetic distortions which do not vary with the orientation of the robot in space, that is to say those which stay fixed to its reference frame, to be subtracted out so the external field can be read. Rotation about all axes produced a table of $x$-$y$-$z$ readings from which a point cloud, see Fig.(5.23), could be plotted. This initial cloud was both non-spherical, a consequence of soft iron type distortions from metal parts inside the robot, and off-centre, from hard iron effects such as magnets in the robot's motors. A C implementation of an algorithm was found [87] which converted the distorted ellipsoid to a sphere using a mapping given by Eq.(5.2).

$$\begin{aligned}
B_{Xdebiased} &= B_{Xin} - X_{offset} \\
B_{Ydebiased} &= B_{Yin} - Y_{offset} \\
B_{Zdebiased} &= B_{Zin} - Z_{offset} \\
B_{Xout} &= C_{11}B_{Xdebiased} + C_{12}B_{Ydebiased} + C_{13}B_{Zdebiased} \\
B_{Yout} &= C_{21}B_{Xdebiased} + C_{22}B_{Ydebiased} + C_{23}B_{Zdebiased} \\
B_{Zout} &= C_{31}B_{Xdebiased} + C_{32}B_{Ydebiased} + C_{33}B_{Zdebiased}
\end{aligned} \tag{5.2}$$

The C implementation finds offset values for the hard-iron distortion on each axis, and translates $B_X$, $B_Y$, $B_Z$ datapoints to remove this hard-iron effect. The longest axis of the ellipsoid is found and rotated to match the $x$-$y$-$z$ co-ordinate frame. The ellipsoid is stretched and compressed about each axis to make it spherical, and is then rotated back to the original orientation and scaled to the known strength of the earth's magnetic field at this latitude. This processing provides each $B_X$, $B_Y$, $B_Z$ location on the off-centred ellipsoid with a mapping to a point on a centred sphere. Li's Least Squares Ellipsoid Specific Fitting [89] method was used to calculate, as a matrix, the $C_{nm}$ constants for the soft iron distortions. Fig.(5.23) illustrates the original point cloud in comparison to the modified one.

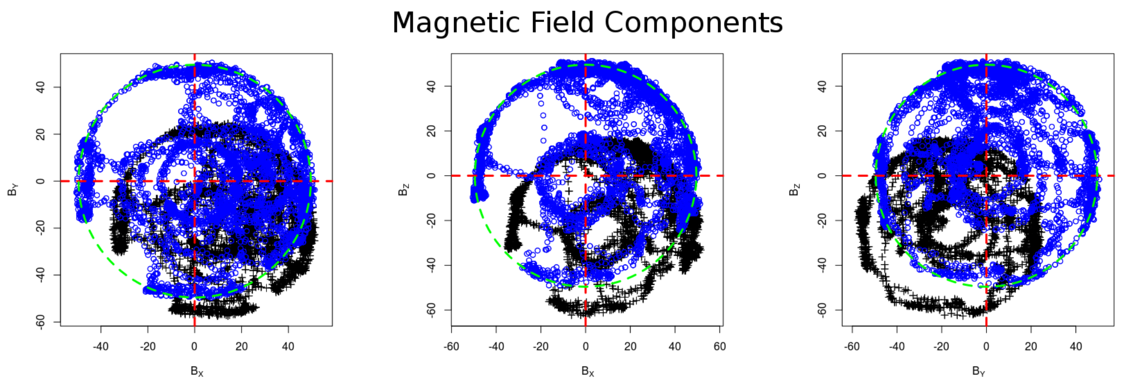


**Fig. 5.23:** Three graphs show, as viewed from each axis, a spread of $B_X$, $B_Y$, $B_Z$ magnetometry datapoints measured on the other axes, taken while manually rotating the robot across all angles. The initial measurements (black) are compared with their values (blue) after processing with Eq.(5.2). The centre position is marked (red) as is a circle around it (green) against which processed points are compared.

The equations of Eq.(5.2) were then used on the robots to convert from raw magnetic field readings to measurements of the global magnetic field from which a heading could be derived. This heading was derived via a tilt compensation algorithm which combined accelerometer readings with magnetometer readings to account for the fact that in the latitude regions where this work was done, 53.96°N, the earth's magnetic field has a large, 68.4° below horizontal, downward component. The tilt compensation procedure used for Omni-Pi-tent's compass was derived from [88], as an alternative to long-winded and bug-prone methods based on manually attempting trigonometry in multiple reference frames. A cross product of the downward vertical gravity vector and the partly downward magnetic field vector give a vector to the West on the horizontal plane, the vector of North on the horizontal plane is then found from the vector cross product of the downward gravity vector and the West vector. Scalar dot products of the North and West vectors with the magnetometer chip's co-ordinate frame give the heading of the robot on the horizontal plane.

This pure magnetometry method meant a single initial calibration for each robot could be performed just once, rather than an automatic calibration constantly running and risking unpredictable readings when in low confidence states. As reprogramming the hinge ATMEGA chips, on which the BNO 055 reading code runs, requires the laborious process of disassembling the entire robot to reach these chips, the hinge ATMEGA code was modified simply to feed the raw magnetic field readings to the Raspberry Pi on which a function in the controller of the robot could apply that robot's individual matrix solution values to the calibration equations. It should be noted that the calibration process is needed independently for each BNO 055 chip and each robot. Due to variability in factory production, and sensitivity to effects such as stresses on the chip from its solder connections to the breakout board, each BNO 055 board requires a different set of calibration parameters even if it is operating in an identical magnetic offset environment. The calibration process was therefore performed for each robot immediately after assembly and the calibration parameters saved on the Pi for later use. This method of compass calibration, compared to the self-calibration built in to the BNO 055 chip, is more sensitive to severe variations across space in external magnetic fields, so forces the robot to be operated in environments where the field is relatively constant across the room. However, as the requirement to frequently hand calibrate the robots while in operation is removed, this new method of magnetic offset-based calibration makes the robots significantly more independent and easier to use in practice. As all motors in the robot are brushed DC motors, with fixed magnetic stators, the fields from these do not vary during operation hence do not disrupt the calibration.

## 5.8 Human Interface devices

Each robot is fitted with a limited selection of human interface equipment. Ignoring the On and Off switches there are 5 switches, 3 locking, 2 non-locking, with states which can be read by code running on the Raspberry Pi. These are located so they can be reached by hand while the robot is moving, but are unlikely to be interfered with by environmental obstacles or other robots. 3 pairs of visible light LEDs, positioned to ensure some can always be seen regardless of hinge position, in red, green and white, are included in the design. One of these LEDs is reserved for indicating battery state and wired to the LEDs ATMEGA, the duty cycle of blinking becoming longer as the battery discharges. The other two can be controlled from programs running on the Pi. One of them is typically blinked to indicate successful I2C transactions, the other may be used to display a binary status at any given moment during an experiment. The main output for debugging, as well as general use, is a 0.3W 8Ω SMD mounted microspeaker, powered via an LM386 audio amplifier, see Fig.(5.24). SMD mounting avoided any need for the fragile solder-tab wire joints which many very small speakers have. The espeak voice synthesising software [81] is used on the Pi to report messages and variable values from running programs in a satisfying received pronunciation accent.

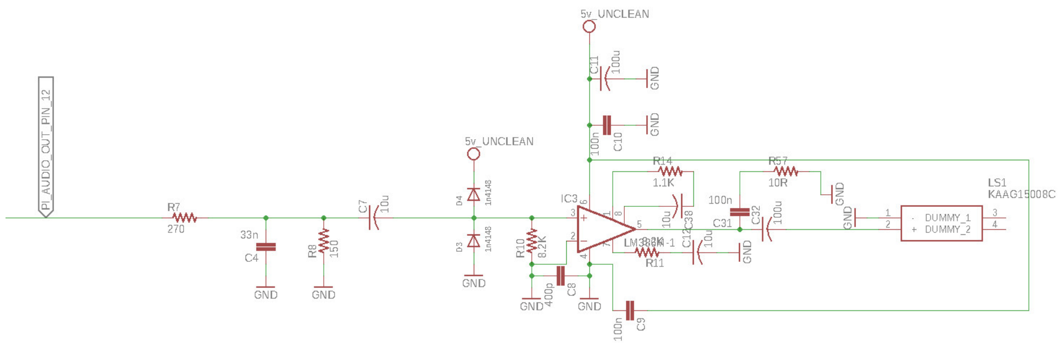


**Fig. 5.24:** The circuit constructed on the central PCB to allow the Pi Zero W to output an audio signal. A PWM signal is supplied by the Pi's pin 12, and low pass filtered before amplification in an LM386.

## 5.9 Central PCB

The Central PCB acts as mounting for most components of the circuits described in Sections.(5.6), (5.7) and (5.8). This central PCB is extremely mechanically constrained, having to fit within limits defined particularly by the arc of travel of the 2DoF hinge and the bevel gears used within it, see Fig.(5.25). It is also constrained by having to mount the Raspberry Pi Zero W central computer of the Omni-Pi-tent robot without colliding with the Port 2 PCB which slides in from the rear. Much of the space is taken up by headers for the 1.25mm pitch 14 way cables which connect it to the Port PCBs, other large occupiers are the carriers for the 28 pin DIP format LEDs, Motors and Hinge ATMEGAs, and the 40 pin header for the Pi. To make sufficient room on this board for everything, whilst keeping to two layer PCB designs, a daughter board is fitted to it, this daughter board primarily handles the logic level translations between the 3.3V Pi and the 5V ATMEGAs, with many of the signals between them going from the Pi and along the header pin to the daughter board without connecting to the main board, then returning to the main board along other pins once level shifted. Fig.(5.26) shows one of these Central PCBs with all components soldered to it and some of the connector wires fitted.

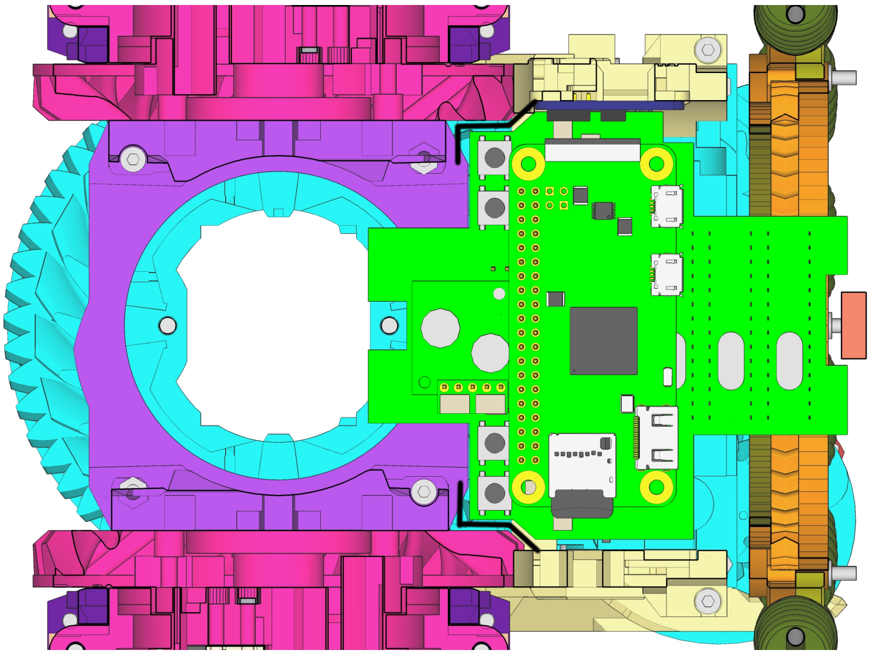


**Fig. 5.25:** Diagram of the mechanical constraints on the Central PCB's shape. Note how the board has been designed to fit within the black highlighted lines which indicate the regions in to which the front bevel (pink) and hinge centre mountings (purple) can extend when at their angular limits. The square cuts in to the forward and rear edges of the Central PCB are, respectively, to accommodate for metal supporting rods running through the hinge centre and avoid contacting the backshaft of the Port 2 hinge motor.

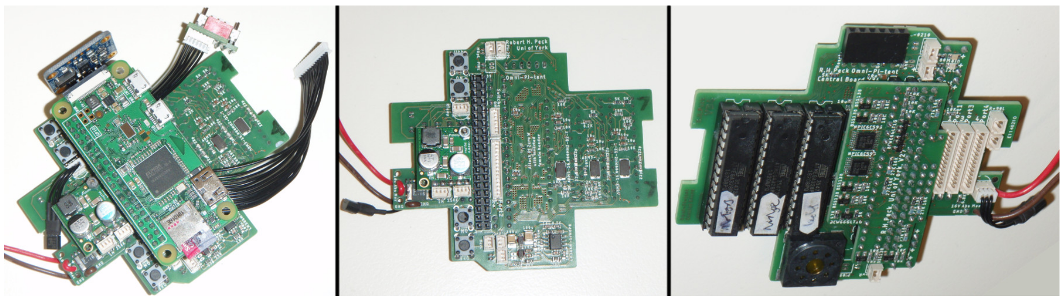


**Fig. 5.26:** Photographs of the Central PCB of an Omni-Pi-tent robot.

## 5.10 Embedded Software Overview

Embedded software runs on each ATMEGA chip and on the Raspberry Pi Zero W. Real-time tasks required to run the robot are separated out among the different chips with each running software required to control the specific elements of hardware it is connected to. All of the software was written in C/C++, the software on the ATMEGA chips was developed using a mixture of Arduino libraries and AVR commands, the software on the Pi used C and the BCM2835 library [1].

The simplest tasks are assigned to the LEDs ATMEGA, this microcontroller acts to control the robot's power regulation and battery cut-off functionality, whilst also generating a 5KHz square wave for each port LED. This ATMEGA never gets reset, it is not on the I2C bus either. Wires from each of the port ATMEGAs are taken low or high by those port microcontrollers, telling the LEDs ATMEGA which ports require a 5KHz wave to be supplied. Later, see Sec.(5.12), the fact that this microcontroller is always on during operation was harnessed to allow it to act as a non-volatile data storage device for other parts of the robot, and more complex inputs were used to achieve this, however they still do not modify the basic functionality of this microcontroller's software.

The Motors ATMEGA serves to provide PWM and direction signals for the four omniwheels. Each omniwheel has one GPIO pin dedicated to supplying a PWM signal with which to control speed, and a further two GPIO pins to control the direction of the wheel. Because PWM control of DC motors, for any given required level of torque, typically has a minimum achievable speed for the motor, this microcontroller allows wheels to be put in to a "stuttering" mode for slower motions. The stuttering mode switches the wheel motor between operating at a specified low PWM value or being stopped, it acts much like PWM but over much slower timescales, typically on the order of seconds. The Motors ATMEGA can generate these stuttering signals for each motor, and can also, as commanded over I2C vary the phase of the stuttering between different motors and vary the stuttering period up to 2.53s. The minimum PWM speed below which stuttering is used can also be set, over I2C with options to set different threshold PWM levels for each wheel. This Motors ATMEGA microcontroller also acts to monitor the internal temperature readings and battery state and pass them over I2C to the Raspberry Pi. This microcontroller is connected by a trace to one of the input pins on the LEDs ATMEGA, upon receipt of a special I2C command sent from the Pi it can use this connection to the LEDs ATMEGA to initiate a controlled shutdown of the robot.

The Hinge ATMEGA controls both the hinge system and the compass, it provides PWM and direction signals to the left and right sides of the hinge mechanism and reads from position encoding switches as well as over-travel safety limit switches. When the controller code on the Raspberry Pi wishes to move the hinge it communicates the desired new pitch and roll positions to the Hinge ATMEGA over I2C, the Hinge ATMEGA calculates the angles to which each of the left and right bevel gears must be moved so as to produce the required pitch and roll. If a pitch and roll combination is specified which would require either the left or right bevel to be positioned at an angle which does not correspond to an integer number of taps on the position encoding switch then this microcontroller calculates the closest alternative positions which it can reach with an exact number of encoding taps and informs the Pi what this achievable pitch and roll combination would be. The Hinge ATMEGA then starts the left and right halves of the hinge moving in appropriate directions so as to travel to the desired position, this decision is made subject to rules preventing the hinge from travelling beyond what it calculates as its pitch and roll limits and is further protected from causing damaging over-travel by software routines which stop the hinge motion entirely if it trips an over-travel limit switch and is not moving in such a direction as to be returning to normal position ranges from the over-travelled extreme. As the hinge moves this microcontroller counts the number of taps on the left and right encoder microswitches and brings the hinge to a stop once it is at the right position. The position encoding switches undergo debouncing within the Hinge ATMEGA's software so as to ensure that electrical noise, or mechanical vibrations, cannot cause false tap counts and positional drift. This ATMEGA was chosen to control the compass as it is the one with sufficient GPIO pins free to control a number of negotiation lines which make this possible. As the BNO 055 compass chip is an I2C device this Hinge ATMEGA must switch between being an I2C slave to the Pi and an I2C master to the compass it is necessary for connections outside the I2C bus to be provided to negotiate which device is master at any given time. An input from the Pi ensures that the Hinge ATMEGA does not attempt to control the compass during periods when the Pi must act as master of the I2C bus, an output to the Pi allows the Hinge ATMEGA to warn the Pi when it is controlling the compass and the Pi should delay any planned I2C transactions.

Finally four Port ATMEGAs are included, each handling real-time operations of a separate port. These serve to handle IR port-to-port communications, see Sec.(5.2.1), and IR guidance and proximity readings, see Sec.(5.2.2). These ATMEGAs also control the docking hooks and read from switches in the docking port's spike accepting pits which can be used to identify whether another port is successfully docked to it. These microcontrollers handle the repetition of emitted infrared messages, the randomised alteration of timing intervals between them, and how to balance these timings against the need to dedicate periods to ambient (guidance) and reflective (proximity) analogue sensing. Commands including the desired hook state and the contents of IR messages to be sent are supplied over I2C to these ATMEGAs which can respond over I2C to report sensor readings and incoming IR messages.

## 5.11 I2C and Internal Messaging

The I2C protocol is used for communication between the internal elements of the robot as it allows the Pi to send data down to the ATMEGA chips, but also to send requests for them to make a response with a message for the Pi. Many difficulties were encountered when implementing this, and hence many fail-safes had to be constructed around it to ensure I2C failures could be overcome easily.

The first difficulty which I2C had to overcome was the voltage level difference between the Raspberry Pi and the rest of Omni-Pi-tent's systems. The Pi uses 3.3V logic for its GPIO pins and risks serious damage if any, except the power supply, are exposed to 5V. Ordinarily an I2C line could work in these circumstances with the Pi as master and the 5V ATMEGAs as slaves, the fact the lines are pulled up by a resistor, then taken low when transferring data means the Pi would not suffer 5V exposures. However because the Hinge ATMEGA must sometimes become master of the bus so as to control the compass chip the I2C system in an Omni-Pi-tent robot can put the Pi at risk. Therefore BSS138 MOSFETs were used as 3.3V to 5V level shifters for bidirectional signalling on the I2C lines.

The next complication involved is related to the use of interrupts on the ATMEGA slaves. These microcontrollers, particularly the port ones, have some time critical routines which cannot run at the same time as the I2C bus. The I2C reception and transmitting functions on the ATMEGA chips involve interrupts and can cause conflicts with interrupts required for purposes such as detecting an incoming IR message, or disrupt the bit-banging used to provide a 38KHz output waveform when emitting IR communications. This problem is solved by using negotiation lines, one for each Port ATMEGA and one for the Hinge ATMEGA. Neither the Motors ATMEGA nor the LEDs ATMEGA require these negotiation lines as they are, respectively, running code which does not contain time critical interrupts or not on the I2C bus at all. These negotiation lines operate in a single direction, conveying information from the ATMEGA to the Pi, hence potential dividers act as level shifters. These lines are taken high when the ATMEGA can accept I2C transactions from the Pi and kept low otherwise. Delays were used between the times when the negotiation line changes and the times when the ATMEGA becomes able or not able to accept I2C messages, these ensure an I2C transaction started at the moment when the ATMEGA drops the line will complete correctly before the ATMEGA becomes unable to receive.

For the Hinge ATMEGA some slightly more detailed negotiation is required, the Hinge ATMEGA can delay the Pi's communications to it by keeping its output negotiation line low, but the Pi can also use a “stop mastering compass" line to warn this ATMEGA when it must demote itself from I2C master of the compass to become an I2C slave. This is required not only when the Pi is communicating with the Hinge ATMEGA but also when the Pi is communicating to any other ATMEGA slave, as the Hinge ATMEGA could otherwise try to communicate with the compass during this time and disrupt the Pi's messages. While the I2C protocol is supposed to support multi-master use it seems that neither the default implementation used on the Pi or the ATMEGA328P support those master-collision-prevention features of the I2C specification. The “stop mastering compass" line ensures that the Pi can make the Hinge ATMEGA fall silent for long enough for the Pi to complete communications with all other I2C devices.

CRC checksums are used to provide further resilience to corruptions of communications. Mainly they serve to filter out communications which are corrupted entirely and consist of all 1s or all 0s, but they also serve to protect against the rarer possibilities of corruption of a few bits or bytes. A 2 byte checksum is used and occupies the final 2 bytes of the message format, both for downward commands and upward sensor data, which each chip uses. The receiving device, Pi or ATMEGA, for any message can compare the value of the checksum bytes at the end of an incoming message to the value of checksum it calculates for the rest of the bytes of the message, if both match the message is considered uncorrupted. Specifically with the use of the an $n$ bit CRC algorithm any single burst of interference under $n$ bits in length will always be detected and recognised as a corrupt message and a fraction $1 - 2^{-n}$ of longer error bursts will also be detected, for a 2 byte (16 bit) CRC that expression evaluates to $> 0.99998$.

There are circumstances under which an I2C bus can hang and one or multiple devices upon it can cease to communicate, to deal with this there are multiple mechanisms to reset it. On the initial robot design only a hard reset of crashed slave devices was possible, done by briefly toggling the reset pins of those ATMEGAs to a low state. After a reset an ATMEGA chip with an Arduino bootloader on it takes several seconds to restart. Hence, within the robot all of the chips are programmed using an ICSP method on pins 1 (reset), 17 (Arduino D11), 18 (Arduino D12) and 19 (Arduino D13), rather than using the bootloader and pins 2 (Arduino D0) and 3 (Arduino D1). Using directly programmed chips instead ensures a much faster reset, modifying the ATMEGA328P's “lfuse" setting to $0 \times DF$ rather than the default $0 \times FF$, further hastens the time for a chip to resume function after a reset. The reset pins are usually pulled high to the 5V rail using 5KΩ resistors, NPIC6C595 shift registers with open drain outputs allow the Pi Zero W to trigger resets on each microcontroller slave, except on the LEDs ATMEGA for that would result in the whole robot being powered down, while sacrificing relatively few GPIO pins on the Pi for this role. Level shifters are again used to adapt from the Pi's 3.3V pins to control the 5V shift registers. Of particular note is that the output enable line of the shift registers uses a logic inverting circuit instead of a MOSFET, this ensures that accidental resets of microcontrollers cannot occur whilst the robot is powering up and before the Pi has booted and gained control of its GPIO pins.

From the second prototype onwards two extra reset methods for ATMEGA chips with a hung I2C bus were added. The first makes use of some details of the I2C specification which have been properly implemented on both Raspberry Pi Zeros and ATMEGA328Ps, in particular that any slave device should release a pulled data line if the clock line is toggled for more than as byte worth of data. This “nineclocks" reset provides a first line of defence against I2C bugs without resorting to anything as drastic as resetting one of the ATMEGA slaves. For when the “nineclocks" method failed a soft reset functionality was added for a form of what has been referred to in the Omni-Pi-tent code as a “soft reset". This required one GPIO pin on each ATMEGA slave to be configured as an interrupt input usually pulled high to 5V. Extra outputs of the NPIC6C595 shift registers control these “soft reset" pins. When a falling interrupt is received on them the ATMEGA which has been ordered to “soft reset" runs a routine which resets all the I2C bus elements of the ATMEGA328P but without disrupting the running program or memory the way that a full reset would. Full resets can therefore be kept as a last resort for when both “nineclocks" and “soft reset" functionality cannot release an I2C hang.

On the Raspberry Pi a section of code is run during every loop of the controller program, see Sec.(5.13), which handles I2C communication to each slave device. This code starts by waiting

```
static ResetStatus[6]={0,0,0,0,0,0};
SlavesDoneArray[6]={0,0,0,0,0,0};
gpio_write(StopMasteringCompassPin, HIGH);
delay(4ms);
while(SlavesDoneArray[0]+SlavesDoneArray[1]+SlavesDoneArray[2]+SlavesDoneArray[3]+SlavesDoneArray[4]+SlavesDoneArray[5] < 6){
    SlaveID=0;
    ExitSlaveChecker=0;
    SlaveDontI2CMeChecker[6]={0,0,0,0,0,1};//sixth slave, the motors, is always free
    for(CounterS=0; CounterS<6; CounterS++){
        if(gpio_read(SlaveDontI2CMePins[CounterS])==HIGH){
            SlaveDontI2CMeChecker[CounterS]=1;
        }else{
            SlaveDontI2CMeChecker[CounterS]=0;
        }
    }
    LoopedRoundAll=0;
    while(SlaveID<6 && ExitSlaveChecker==0){
        if(LoopedRoundAll==0){
            if(SlaveDontI2CMeChecker[SlaveID]==1 && SlavesDoneArray[SlaveID]==0){//if slave is free to be talked to we exit and talk to it, if we've not talked to it already
                ExitSlaveChecker=1;
                SlavesDoneArray[SlaveID]=1;//prevents us returning to this slave
            }else{//continue and test next slave
                SlaveID=SlaveID+1;
            }
        }else{
            if(SlavesDoneArray[SlaveID]==0){//exit and talk to it, if we've not talked to it already
                ExitSlaveChecker=1;
                SlavesDoneArray[SlaveID]=1;
            }else{//continue and test next slave
                SlaveID=SlaveID+1;
            }
            if(SlaveID>5){
                ExitSlaveChecker=1;
            }
        }
        if(SlaveID > 5){//we've checked the fifth slave, none were free, and not all are done, then we pick the lowest numbered Slave not yet done
            SlaveID=0;
            LoopedRoundAll=1;
        }
    }
    PerformI2CTransaction(I2CAddressOf(SlaveID));
    if(ChecksumMatches(SlaveID response)){
        ResetStatus[SlaveID]=0;
        StoreI2CResponseMessage(SlaveID);
    }else{
        if(ResetStatus[SlaveID]==0){
            NineClocksReset(SlaveID);
        }else if(ResetStatus[SlaveID]<5){
            SoftReset(SlaveID);
        }else if(ResetStatus[SlaveID]>=5{
            HardReset(SlaveID);
        }
        ResetStatus[SlaveID]=ResetStatus[SlaveID]+1;
    }
}
gpio_write(StopMasteringCompassPin, LOW);
```

**Fig. 5.27:** Pseudocode of the software on the Pi handling I2C transaction order, see main text for details.

for any ongoing communication between the Hinge ATMEGA and compass to end, as signalled by the dropping of the Hinge ATMEGA's negotiation line, the Pi then takes the “stop mastering compass" line high. The Pi then goes about passing an I2C command down to, and receiving a response message from, each ATMEGA on the I2C bus. Originally this was done in a fixed order. Firstly the Port 1 ATMEGA's negotiation line would be checked, the Pi would wait for the negotiation line to go high, if it was not high already, and then perform the I2C transaction. This would repeat for the other Ports, the Hinge and the Motors ATMEGAs. When using this method it was not possible to complete an up and down transaction with each ATMEGA fast enough to meet the required, so as to maintain compatibility with simulations already developed at that stage, 20Hz frequency for controller loops. A new method was therefore developed which dynamically orders the slave ATMEGAs for transactions by communicating first with those which happen to have their negotiation lines already high just after the Pi has raised the “stop mastering compass" line. An array stores records of which ATMEGA slaves have been communicated with already and after a successful transaction with one of them only the remaining slave devices are listed as being required to be communicated with. Times spent waiting for each ATMEGA to enter the right parts of its embedded program so as to be able to receive I2C transactions are greatly reduced, making the 20Hz controller refresh rate feasible. Fig.(5.27) shows pseudocode explaining this principle.

If a “NACK" (No Acknowledge) is received from an attempt to write an I2C command to a particular ATMEGA, or from an attempt to read a response message from one, or if a corrupt checksum is found in the repines message, then the ATMEGA in question is given some form of reset. A “nineclocks" when a failure first occurs, a “soft reset" on subsequent loops of the controller, and a full hard reset 5 loops later if that ATMEGA continues to suffer I2C problems. Persistent failures of I2C transactions with all ATMEGA devices can also trigger a soft or hard reset of all ATMEGAs.

Excepting for circumstances where an I2C or negotiation line wire is damaged during assembly of a robot the need for any resets is very rare during normal operation.

## 5.12 Incremental Encoding, or How I Learned to Stop Worrying and Love the EEPROM

As discussed earlier, the hinge, both in the initial and upgraded designs, uses microswitches tapped by notches on moving parts of the mechanism for position tracking. These switches are wired so that an input pin is pulled low when they are pressed and high when they are released. Switches often, at the moment of changing state between conducting and insulating, suffer a series of bounces which, when rising and falling edges are added up can appear as several press and release events. This would prove seriously problematic for a position tracking system which requires an accurate count of the number of times the switch is pressed, hence software debouncing is used to ensure a clean measurement of a single edge as a switch changes states. The history of each switch is stored as a series of bits within a byte value, with a new reading of the input pin and bit shift each time the debouncing routine is called. The history byte undergoes a binary AND with a masking byte and is checked for patterns indicating whether the switch was consistently high and is now consistently low, in which case a push event is counted and the history adjusted to a 0B00000000, or whether it was consistently low and is now consistently high, in which case a release event is counted and the history byte adjusted to 0B11111111. Running the debounce routine at 100Hz gives a good balance between reliably and quickly detecting when a notch pushes or releases a switch and ignoring any bouncing involved in the process.

Counting the pushes and releases serves as a form of incremental encoding. Due to both geometry and to the availability of GPIO pins on the Hinge ATMEGA, there can only be one switch for each of the left and right sides of the hinge. Therefore quadrature encoding is not possible and it is necessary to know the direction of the motors on either half of the hinge so as to determine whether a push or release event should be added or subtracted from the position count. This requires low backslash within elements of the hinge system between the motor and the point at which encoding taps are read, otherwise under some circumstances the tap notched section of the mechanism could be moving oppositely to the motor for a brief period and false counts may cause position drift. Backlash within the motor's own gearbox could also be problematic here if large enough. In the initial worm geared design of the hinge the wedge shape made between the worm itself and the worm wheel ensured that the system was non-backdrivable, this meant that the effects of altering loads on the hinge's output could not easily pass back in to the worm's rotation and periods when the worm's motion direction was not matched to the motor's motion direction were unlikely to occur. With later upgrades to the hinge design these risks became more significant and hence very rigid construction was required within the hinge mechanism and motor adapter.

With this encoding mechanism the hinge can be set to positions separated by $2^{\circ}$ increments. The accuracy of this positioning is to within a margin determined only by the amount of backlash present in the bevel stage of the hinge's gearing due to the clearances between the rolling mount of Port 4 and the “barrel" structure within which it turns. So long as position drift in the encoding software does not occur, these $2^{\circ}$ increment positions are entirely repeatable to within the range of backlash.

Early testing of the hinge mechanism in the first robot prototype discovered a concerning problem with this solution for position monitoring. As the Hinge ATMEGA was responsible for tracking the position via totals of counted push and release events it would not know the hinge position at the moment it was turned on, worse still it would also loose track of hinge position during the rare occasions when a hard reset, see Sec.(5.11), of the Hinge ATMEGA was required. The question became how to ensure that hinge position information could survive across a reset of this microcontroller. ATMEGA328P chips have an internal EEPROM which provides non-volatile memory, however write cycles to this EEPROM are limited. As the EEPROM would have to be rewritten every time that the hinge moved between tap notches this could quickly wear out the EEPROM. An alternative place to store data in a fashion which could survive across resets of the Hinge ATMEGA presented itself within the one microcontroller in the Omni-Pi-tent robot which is never reset and only powers off when the rest of the robot does so, the LEDs ATMEGA. A custom 3 wire protocol, based on SPI principles, is used between the two ATMEGAs, with the Hinge ATMEGA serving as master and generating the clock signal. At any time when the tap counts for either side of the hinge change the Hinge ATMEGA begins generating a clock signal on one of the 3 wires, as the clock line rises the LEDs ATMEGA checks whether the second wire (Hinge Out LEDS IN) is high or low and records this as one bit of a byte, as the clock line falls the LEDs ATMEGA sets the third wire (LEDS Out Hinge In) low or high for each bit of a byte value. Messages passed from the Hinge ATMEGA to the LEDs ATMEGA carry a 3 byte representation of the tap counts, plus a 1 byte checksum. So long as the checksum matches the message the LEDs ATMEGA stores the incoming 3 bytes in a variable. Fig(5.28) shows this protocol in action. During each operation of the 3 wire protocol the LEDs ATMEGA will report back the variable values as stored during the previous operation. At most times the Hinge ATMEGA ignored the incoming messages on the third (LOHI) wire, however after a hard reset the Hinge ATMEGA sends deliberately corrupted values over the protocol. The LEDs ATMEGA recognises not to store these corrupted values, and responds in the normal way informing the Hinge ATMEGA of the current contents of its variables, which the Hinge ATMEGA provided before the hard reset event. As well as sending these 3 wire protocol messages to the LEDs ATMEGA whenever the Hinge ATMEGA counts a tap on either encoder microswitch, they are also sent by the Hinge ATMEGA immediately after any soft reset is performed on it. As any ATMEGA chip suffering I2C crashes within the robot will always be given a soft reset before being subjected to a hard reset this further ensures that the LEDs ATMEGA has accurate data in its variables to restore to the Hinge ATMEGA after the hard reset. Stopping the hinge motors briefly after any soft reset is also necessary here to ensure that the hinge position cannot change between the time at which values were transferred to the LEDs ATMEGA and the time at which a hard reset may occur. For further reliability the 3 wire protocol has time limits built in to it so that if the Hinge ATMEGA were to be reset while sending a message then future messages will be considered the start of new messages, not added in to the bytes of a partially received message. Within the LEDs ATMEGA the 3 wire protocol is handled using interrupts on the rise and fall of the clock line, these are fast enough not to affect any 5KHz square waves it may be producing at the time.

The versatility of this method of storing hinge position data is further extended by making use of an ATMEGA chip's EEPROM functionality. Although the limited write cycles could cause problems if they were used to store, on the Hinge ATMEGA, every position change, there is scope for the hinge position to be stored across power cycles of the robot by using the LEDs ATMEGA's EEPROM. As it is never subject to hard resets during robot operation such writes to the LEDs ATMEGA's EEPROM are only required when the robot is being powered off, this is a much rarer event than the changing of the hinge position and so does not risk eating in to life cycles as quickly as EEPROM writes upon every position change would. Write frequency to this EEPROM is even further reduced by only writing new data to the EEPROM if the hinge position stored in the LEDs ATMEGA's variables moments before shutdown is different from the EEPROM readings which the LEDs ATMEGA makes as the robot is powered on. Using this method, despite using an incremental rather than absolute encoding method, hinge position data can be stored safely in the LEDs ATMEGA to avoid position drift caused by hard resets of the Hinge ATMEGA, and by using the LEDs ATMEGA's EEPROM we need not rule out the

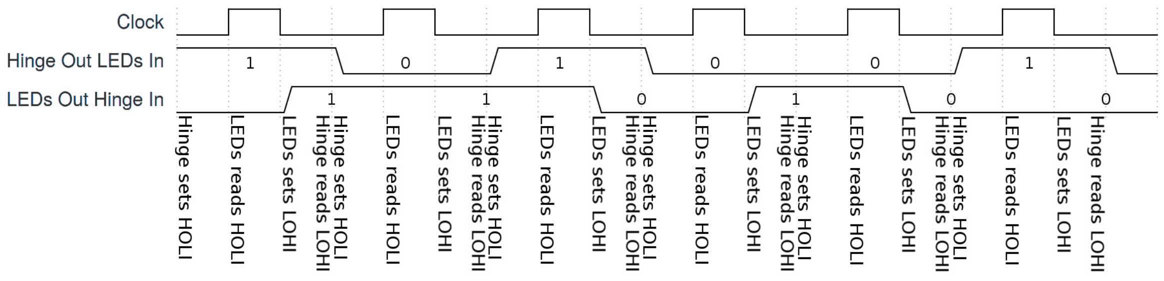


**Fig. 5.28:** Diagram showing the 3 wire protocol used to communicate hinge position information between the Hinge ATMEGA and the stable storage on the LEDs ATMEGA. In this example, as part of a longer message, 101001 is transferred from the Hinge to the LEDs and 110100 is passed from the LEDs to the Hinge. The Hinge ATMEGA sets a value on the Hinge Out LEDs In (HOLI) line then toggles the clock line up, this triggers the interrupt on the LEDs ATMEGA to read HOLI. As the Hinge ATMEGA lowers the clock an interrupt is triggered on the LEDs ATMEGA to set LEDs Out Hinge In (LOHI) which is read one time period later by the Hinge ATMEGA. The HOLI and LOHI lines may each be in either the low or high states before or after a complete message has been passed.

chance to preserve a nucleus of this data across power cycles of the whole robot too.

For use in testing the hinge control subsystem, and as yet an extra backup should any position occur despite the measures described previously, it is possible for the Pi to send hinge position calibration data over I2C, allowing the zero positions of either side of the hinge to be adjusted as necessary.

### 5.12.1 Fixing Hinge Drift During Rapidly Reversing Motions

Once the improvements to the hinge had been made a further situation which could lead to hinge drift was discovered, this did not become apparent in the earlier worm gear based hinge mechanism design as motions were much slower. With the involute gears of the later hinge design, see Sec.(5.16), it was found that if the hinge were repeatedly commanded to move in mutually opposite directions, for example a command to roll right, followed by one to roll left before the first roll had completed, some taps on the encoding microswitches may be improperly registered, or counted as being in the incorrect direction. This could lead to serious position drift, and worsen over time with the calculated position getting further from the true position each time the hinge changed direction before reaching its target position.

The first attempt to fix this had the hinge pause, on one side or both as necessary, when a position command was received which required one, or both, sides of the hinge to change direction. For example if the left bevel gear of the hinge was rising and, before it had reached its commanded position, the Hinge ATMEGA received a new command requiring the left bevel gear to be sent to a lower angle than its present location, it would stop, pause for a fraction of a second, then begin lowering. This however did not fully protect against the observed form of position drift. The problem appeared to be more a consequence of the reversal of hinge direction whilst it was between accurately known encoder tap positions, than due to fast timings associated with the direction reversal. The finalised version of the embedded software on the Hinge ATMEGA therefore uses a more thorough method to handle direction reversals. The Hinge ATMEGA now checks each new incoming position command to see if it would require a change of direction by either side of the hinge, if so then rather than changing direction immediately that side of the hinge is allowed to run onwards in its initial direction until it reaches an accurately known position, available at 2° increments. Only once an accurate 2° incremented position has been reached does the hinge stop and begin moving in the new, reversed, direction. When a reversal is commanded the hinge may therefore continue to travel, if running at full speed, for up to 0.8 seconds in the original direction, it will however never go more than 2° beyond the position at which it was when the new position command was given, the software also ensures that the accurately known position at which the hinge reverses will never be such as to result in travel beyond the limit of where the hinge was previously instructed to go to.

## 5.13 C API

The embedded software on the Pi is controlled via a h file library developed to convert between the low level details and the high level functions comparable to those used in the V-REP simulations. This library is based around operations performed on a large C struct which contains details of the robot's hardware state. The struct *RobotHardwareState* contains within it arrays to represent the I2C commands to be sent down to the various ATMEGA microcontrollers, arrays to represent the sensor data received over I2C back from them, and also arrays keeping a record of the previous commands given.

Functions are provided to access the specific bytes within the struct which control specific hardware features, the functions take a pointer to the *RobotHardwareState* struct as well as inputs and outputs in terms of the actuator command to give or the sensor value to return.

The functions behave much like the timestep model on which the V-REP simulation is run, the overall structure of a program to be run on an Omni-Pi-tent robot consist of two functions, repeatedly called alternately. Firstly *RunControllerCode()* in which behaviour algorithms for the robot can query sensor data from the previous timestep, make decisions and give actuator commands, secondly *UpdateHardwareStatus()* in which the latest commands in the struct are sent to the hardware and new sensor data is transferred in to the struct. The struct contains an array for each slave device, which code in *RunControllerCode()* can use functions to write to, which will give downward commands to the ATMEGA chips next time *UpdateHardwareStatus()* runs. A series of arrays in the struct provide sensor data from the ATMEGAs each time *UpdateHardwareStatus()* runs, these can then be read via the functions when *RunControllerCode()* next executes. A third series of arrays takes a *ReadCopy* of each of the downward commands, this copying takes place towards the end of *UpdateHardwareStatus()* but only for the arrays for which the I2C transactions were successful, the *ReadCopy* of an array does not update if the I2C transaction fails, hence it always holds a copy of the last successful message supplied to any given ATMEGA slave, and hence accurately reflects what that slave is currently attempting to do. *UpdateHardwareStatus()* also supplies the necessary delays to ensure that *RunControllerCode()* repeats at around 20Hz, to match the rate of controller update in V-REP simulations.

This structure for the overall programs means that functions in the h file library come in 3 types. *SetX* functions can set a command to send to the relevant slave ATMEGAs, for example a hook state of 0 to unlock, 1 to lock, 2 to forcibly lock and 3 to forcibly unlock can be specified for a specific port, the *SetHookState()* function converts this in to the compressed message format for I2C and places it within the correct I2C command to be supplied to the relevant port's ATMEGA next time *UpdateHardwareStatus()* is run. *ReadX* functions are provided which can query the previous successful command sent for any given aspect of actuation, this uses the *ReadCopy* of the arrays, for example *ReadHookState()* can return for a given port, from a stored copy of a historic I2C message, whether it was in the previous timestep, or the previous successful command if the most recent I2C transaction failed, commanded with a 0,1,2 or 3. Other *ReadX* functions read sensor data supplied by the slave ATMEGAs during the previous *UpdateHardwareStatus()* call, for example *ReadPtransAmbient()* returns an array of analogue values from the phototransistors about the rim of a specified port.

A number of other functions are also included to handle electronics connected directly to the Pi, this includes the reset controlling NPIC6C595 shift registers, which are automatically controlled from within UpdateHardwareStatus() when I2C transactions experience repeated failures. Also connected to the Pi are visual LEDs and switches for the Human interface systems, see Sec.(5.8), the switches are debounced using a method similar to that used for switch debouncing on the Hinge ATMEGA, but running somewhat slower due to the 20Hz, or often slightly less, frequency of the *UpdateHardwareStatus()* function. Finally, a function to trigger spoken print statements is included, this launches a pthread which runs a terminal command for espeak, the pthread system allows the main controller to continue executing at 20Hz, but also prevents multiple audio print statements trying to speak on top of one-another. Upon starting any audio pthread creates a flag to warn that it is speaking, this flag is only erased after speech has finished as that pthread is terminating. Other pthreads started up check for the flag and remain silent if it is present, this means some spoken print statements can be lost if they follow shortly after others, but those heard are atleast not garbled. The h file library created also includes features to elegantly shut down the Pi either when it is powered off by switch, and to handle automatic shutdowns if the battery voltage gets too low or one of the internal temperatures too high.

## 5.14 Wi-Fi

A second h file library was produced to provide Wi-Fi capabilities for when multiple Omni-Pi-tent robots are present in the same environment. In terms of how it interfaces with the rest of the controller code running on a robot the functionality which this Wi-Fi library provides was designed to replicate that which the global communications simulation within the V-REP model provides.

For the most part the Wi-Fi library for Omni-Pi-tent controllers makes use of the Linux (Raspbian) networking stack, details of wireless transmission and reception hardware, as well as IEEE 802.11 protocols are all abstracted away lower in the stack than the code developed here. Ordinary commercial Wi-Fi router hardware serves to manage the network, this being the sole item which is not an Omni-Pi-tent robot which is required when operating a group of Omni-Pi-tent robots. This router does not need a connection to the wider internet, so there is no dependence on external servers or other remote infrastructure. Theoretically a Pi Zero W in a robot could act as a hotspot, or mesh Wi-FI networking could be used. However either of these would have taken an unnecessarily large amount of time to implement when compared to an easily available router device.

Like the audio output functions the Wi-Fi code makes use of C's pthread capabilities to run the Wi-Fi code in parallel with the main controller of the robot. When controller code involving Wi-FI functionality starts up it begins by creating a pthread for Wi-Fi message reception. This thread opens a listening socket on a high numbered, non-reserved, network port for IPv4 TCP packet reception. An infinite *while()* loop is started within this pthread which waits for incoming network connections, then accepts an incoming connection, reads the bytes it supplies and closes the connection before returning to the listening state for later incoming messages. The bytes read from a connection are processed, checked for validity using another CRC checksum, and placed in to a queue struct. This queue, like the queue in the V-REP simulation, holds up to 10 messages, discarding the earliest ones if overfilled. The queue is shared between the receiving pthread and the main thread of the controller code, with access managed by a mutex. In the controller code the function *ReadFromQueue()* is used to check for new messages and output them for processing if they have arrived. Each call to *ReadFromQueue()* removes the earliest message in the queue. It should be noted here that while in the simulation it is safe to, upon each loop of the controller code, call *ReadFromQueue()* repeatedly until it is emptied, on the hardware any such repeated call should also be accompanied by a conditional which ceases calling *ReadFromQueue()* after some predefined maximum number, 10 works well in practice, of iterations. This is because in simulation the Wi-Fi message sending happens entirely within the simulation's timestep structure and all reading of the queues in another part of the timestep, however on the real robot it could be possible for extra Wi-Fi messages to join the queue at the same rate as

*ReadFromQueue()* is emptying it, in which case the controller code is at risk of situations where it could infinitely hang on a single iteration of the *RunControllerCode()* loop if such a maximum-function-call-count exit condition is not used.

Sending a Wi-Fi message is accomplished by a function call which serves as a convenient wrapper around Linux's netcat command. A message is provided in whatever form the bytes of its variables or struct may take, with the numerical ID numbers of the sending robot and the desired receiving robot also included. Robots are given static IP addresses which correspond to their ID number so netcat finds it simple to convert between the receiving robot's ID and the appropriate local IP address for it. Checksums for ensuring the validity of outgoing messages, and filtering out the possibility of irrelevant network traffic sharing the same port number as inter-module communications, are applied at this point. Because a Linux terminal command, in the form of netcat, is invoked in the Wi-Fi sending function all data in the variables or structs constituting a message is temporarily converted from raw bytes in to a text string representation of the hexadecimal values of those bytes. This avoids unpredictable behaviour which could result if raw bytes were fed to the Linux terminal, for example any occurrence of a zero in the message would act to cause the message's premature termination in those circumstances. At the receiving end this conversion to a hexadecimal filled string is undone and byte values of variables returned as part of the processing before checksum verification is attempted. The Wi-Fi sending command is fast and the netcat terminal command issued is non-blocking, hence there is no need for a pthread system to be used for message sending.

## 5.15 Redesign for Assembly

The first Omni-Pi-tent prototype, while functional, was neither pleasant to assemble nor maintainable for upgrades. Therefore the second prototype, beyond modifying some electrical elements to improve reliability, contained major changes to the wiring architecture. One of the most painstaking, and damage prone, elements of the first design had been where wires were directly soldered onto the terminals of those components, such as the IR LEDs and the motors, which were required for mechanical reasons to be positioned away from the PCBs. In photographs of the first prototype white amorphous masses of hot glue are visible, see Fig.(5.29), around some of the most vulnerable component-to-wire solder joints, this glue was applied in a post-construction attempt to protect these wires from being tugged or bent. In the redesign all such soldered wiring was replaced by small shim PCBs adapting from each offboard component to a 2 or 3 way pre-made cable harness. See Fig.(5.29) and Fig.(5.30) for a comparison of the old and new designs.

The shim PCBs were soldered to components, in some locations the soldering of this PCB had to be done after locating the components within 3D printed housings, with joints which were not subject to the bending stresses exerted upon wires. The cable harnesses could then be

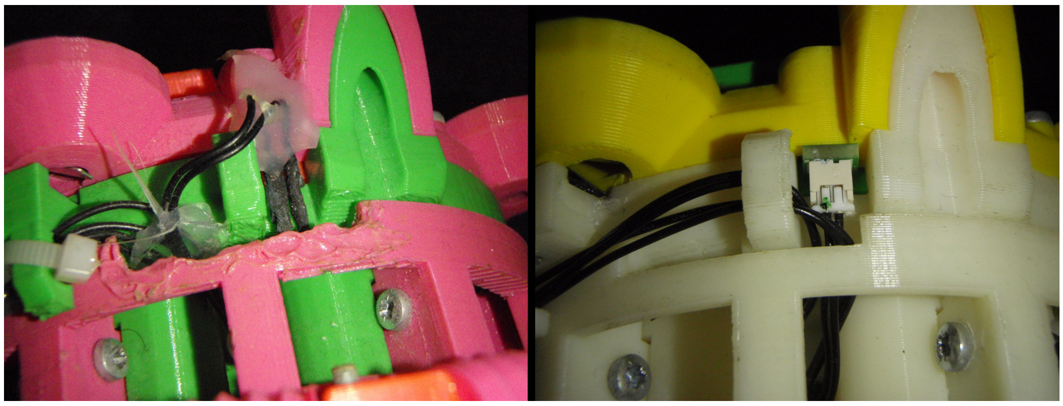


**Fig. 5.29:** A comparison of phototransistor connections on the prototype (left) and the later robots (right).

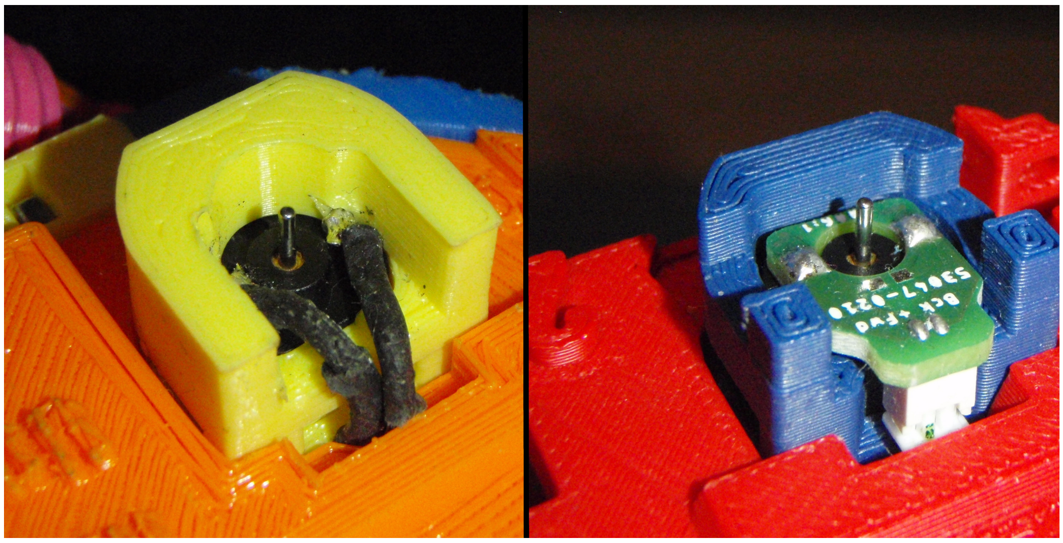


**Fig. 5.30:** A comparison of motor terminal connections on the prototype (left) and the later robots (right). Shown here is the rear end of a 5V DC motor, note the vulnerable heatshrunk wires on the first prototype, compared to the board and connector on the later robots.

connected and removed as necessary to facilitate easier assembly of parts than was possible when soldered wires were used. The location of some screws was altered to enable parts to be fastened together in more convenient orders, however the complexity involved in fitting Omni-Pi-tent's mechanisms in to volumes constrained by the need to avoid obstructing the 2 DoF hinge limited the extent to which overall assembly could be simplified.

## 5.16 Improving the Hinge

Unfortunately the discovery of the limited hinge torque available, see Sec.(5.5.1), was not made until after production of the robots had begun, mainly because the most relevant test required the presence of multiple robots and other methods of verifying the torque capabilities had escaped the authors attention at that time. Improvements to the hinge therefore had to be made in such a way as to enable retrofitting to the already produced robots. Multiple concepts were considered, initially focusing on high reduction ratio methods to increase the torque supplied by 5V “micrometal" gearmotors of the type previously used to power the worm gears. To avoid the “terrace" problems which the worms gears suffered all the new designs kept all rotation within a single plane so that gear teeth would be smooth when printed with the gears’ axes aligned to a 3D printer’s z-axis. Multi-stage involute and cycloidal gears were tried first, with the design requirements greatly complicated by the limited space, a slightly enlarged region around the location where the worm gears were previously housed, in which they had to operate. A number of 45:1 and 60:1 systems were designed, each operated well at low torques but most designs stalled at 294 Ncm (per side) when tested with a weighted lever arm, see Fig(5.31). This would have barely been enough for both sides of the hinge system to lift even a 2kg weight at the 29.2cm distance between docked modules, let alone a >3kg Omni-Pi-tent module or a chain of two such modules.

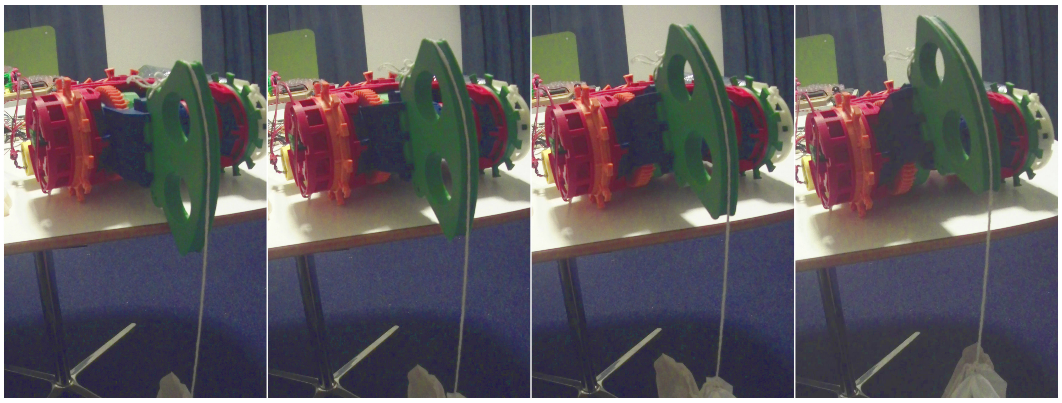


**Fig. 5.31:** Time series photograph of the testing jig used to evaluate torque capabilities of different hinge motor and reduction gear train options. The core of a robot is partially assembled, with the lever arm jig (green) substituted in place of port 4 and the front bevel gear so that the notches on the arm mate directly with the robot’s hinge’s right-hand bevel gear. A string hung across the arm is attached to a weighted bag, the curved arc of the arm’s end ensures the torque requirement stays the same throughout a large angle of lifting. Testing with this lever arm jig allowed the torque for one half of the hinge to be calculated, the torque capabilities of a completed robot are double those calculated by this method. Testing with only one side of the hinge allowed quicker iteration of hinge designs without needing to print left and right copies for each iteration. Motor currents were monitored throughout the tests to ensure the motor was not too close to its stall condition.

The next tests used even more extreme reduction ratios, combining elements of the earlier 45:1 and 60:1 systems to produce reductions of 150:1. The most effective of such designs stalled at just below 588 Ncm of torque (per side). Although both sides of the hinge mechanism giving this amount of torque would be slightly greater than the requirement for lifting one module, the fact that the motors were close to stall meant this design was not reliable, especially not if any further frictional losses were incurred within the bevel gears of the full hinge mechanism than on the testing jig. This design was also unsuitable due to being extremely slow to actuate.

Configurations involving planetary gear stages between involute and cycloidal stages were also tried, with successful, slow, operation but similarly insufficient torque. By this point the hinge motion was painfully slow and it became clear than further reduction ratios simply gave diminishing returns due to friction and other inefficiencies. It was at that point judged that there was no feasible method to source the necessary torques for 3D reconfigurations from the 5V micrometal motors.

Matching the mechanical power $P = \tau\omega$ to the motor's electrical power $P = IV$ made it clear that a more powerful motor would be required. Whilst it had been initially thought that by accepting a slow angular velocity a low power motor, or rather 4 of them together, could serve the purpose, the inefficiencies of 3D printed high reduction ratio gearing prevented this route being further followed. The motor current could not be increased without risking scenarios where the hinge motor's current demands could brown-out the robot's 5V power regulator and all the logic and control circuitry too. That left the only option for increasing the power delivered to the motors being one of increasing the motor voltage, a type of 12V motor with a 900:1 gear ratio within its casing and a low stall current of 0.5A, safe for the TB6612FNG drivers to handle, was found. This required some adaption of the robot's circuits, but could be handled entirely as a retrofit with no need to modify the Central PCB, see Sec(5.17). With the use of higher power motors a hinge design with a small reduction ratio could be used.

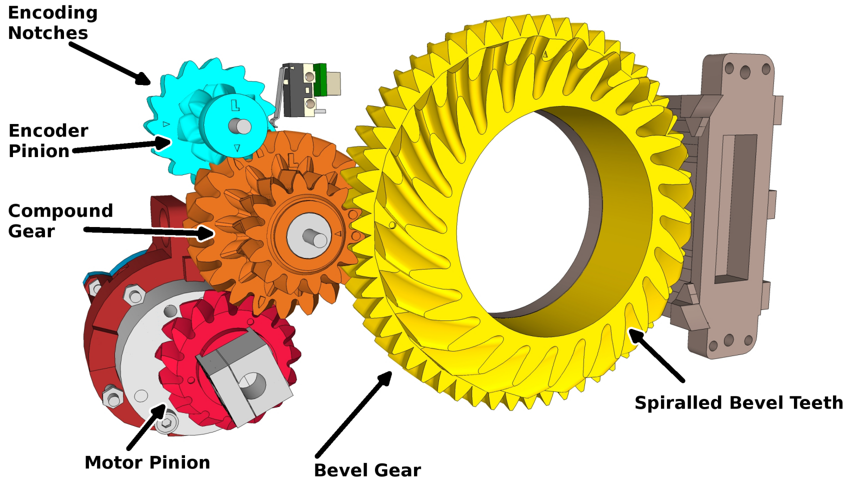


**Fig. 5.32:** The upgraded hinge design within the production Omni-Pi-tent robots.Instead of a worm gear a multi-stage involute gear train is used. A speed reduction with a ratio of 1.4 is performed between the motor pinion and the compound gear, a further four-fold reduction happens before the bevel gear. The compound gear also powers an encoder pinion, the encoder pinion is turned at 3 times the rate of the compound gear allowing the encoder notches to give a higher angular resolution on the output. 15 tap notches could be squeezed around the encoder pinion's rim giving an output resolution at the bevel gear of 2 degrees.

As shown in Fig.(5.32) and Fig.(5.33) the improved hinge design still maintains the bevel system previously used for 2DoF motion, however the high reduction worms are replaced with involute gears. Having a stall torque of 700Ncm (per motor) and readily operating at a third of this these motors need only a small reduction ratio to provide the torques necessary for lifting other robots. Although two stages of reduction are used the first stage, increasing torque by a factor of 1.4 under ideal conditions, is only present in practice so that the motor can be positioned far enough back and down in the robot to avoid colliding with the Port 1 and Port 3 wheels. The main reduction is a 4:1 ratio between an intermediate compound gear and the output bevel. As the reduction ratio is this low it is no longer appropriate to have the tap notches for position encoding running from the same shaft as the motor. Given the maximum number of tap notches which could be fitted onto a rotating element attached to this shaft and remain small enough not to collide with other parts of the robot, the angular increments to which the bevel gears could be positioned would be much larger and the hinge resolution poorer. The tap notches are therefore placed on an encoder pinion which is driven in from the intermediate compound gear with a 3-fold increase in angular speed, 15 tap notches, plus a 3:1 ratio between the encoder pinion and compound gear and a 4:1 ratio between the compound and the bevel, preserves the 2° increment position resolution. Herringbone patterns were applied to all gear teeth ensuring multiple teeth interlocking at any time, a relatively steep helical angle was used, particularly to ensure smooth motion of the encoder pinion gear. Fitting this arrangement of gears into the limited space available was a significant challenge, especially with the need for convenient integer ratios between encoder pinion and bevel so as to ensure that floating point maths, with possibilities for rounding errors, did not need to risk being used on the Hinge ATMEGA when converting between encoding tap counts, processed internally, and positions in degrees, supplied over I2C.

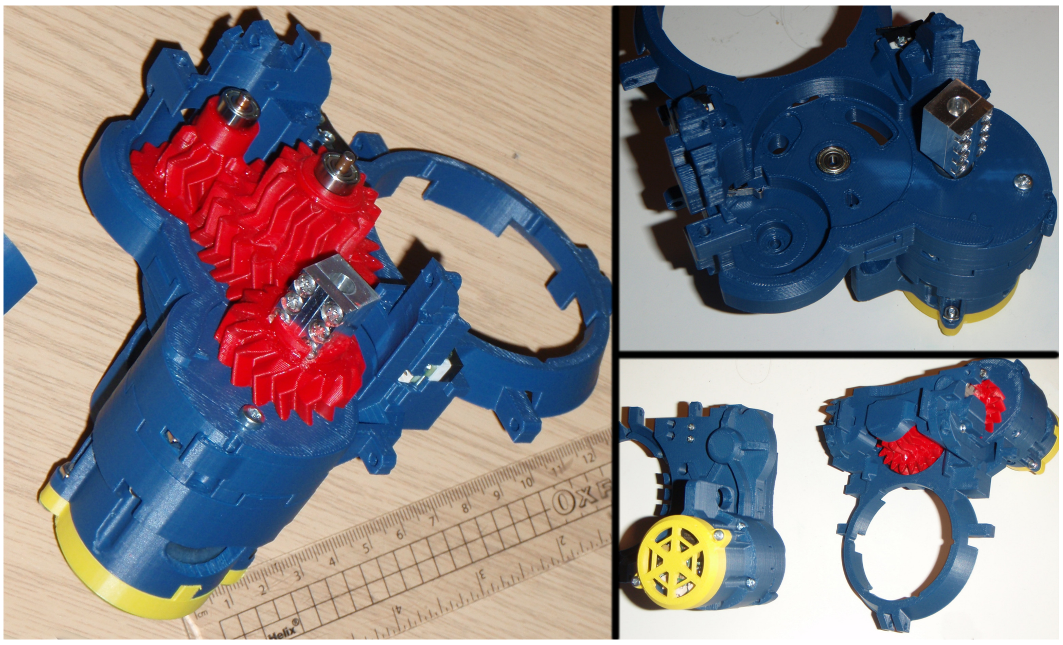


**Fig. 5.33:** Photograph of the new hinge design, notice the rectangular metal block of the motor shaft adapter and the bearings on the gear axles.

Replacing the older worm powered bevel gears with the involute powered type also presented an opportunity to spiral the teeth on the bevel gears and increase the contact area between them. On the earlier design the full forces of pitching and rolling could at times be transmitted entirely through just two bevel teeth, one on either side, while not breaking the teeth this had led to some erosion of sharper features on their edges. By adding spiralling the area of teeth in contact is increased, the tooth cross-section is enlarged and loads are shared between sections of the greater number of teeth which remain in contact at any given time.

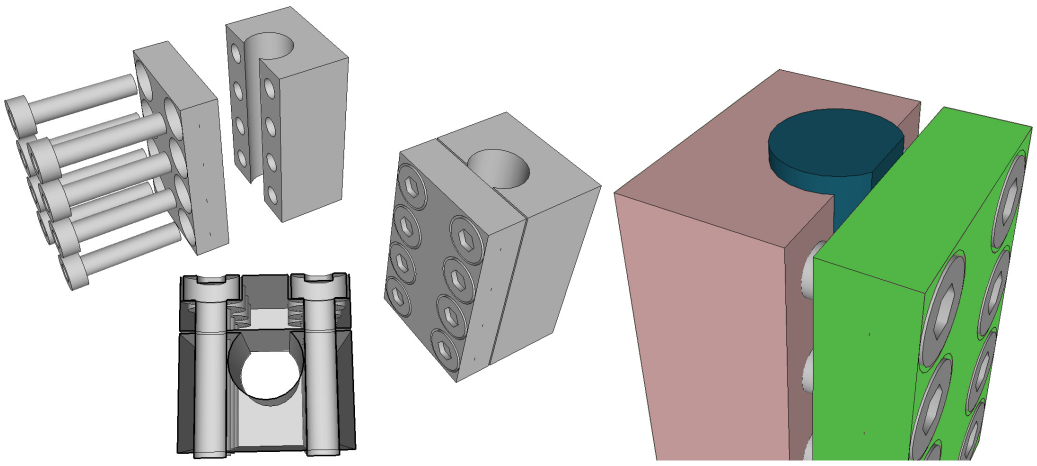


**Fig. 5.34:** CAD diagram of the adapter hub fitted to the shaft of the 12V hinge motors. A diagram of the actual hub is shown to the left, on the right the D shaft is shown (blue) fitted between an upper half (green) and a lower (pink) with the gap between the two halves visually exaggerated. This gap ensures that the upper half is held against the shaft's flat by the tension in the screws and that no space exists between the flat of the shaft and the underside of the upper half.

The forces which acted to dull the edges on the bevel teeth proved far more destructive elsewhere in the gear train. With forces increasing at reduced radii, for a given torque, the greatest forces where experienced around the 6mm diameter shaft of the 12V motor. Using laser cut plastic pieces to adapt between the motor's D shaft and a larger square to mount in to the 3D printed pinion gear, as done for the hook and wheel motors and the earlier worm gear based hinge mechanism, was not a viable option as they would shatter even when the hinge's output was providing relatively low torques. The next option tried used pre-cut purchased steel D to hex adapters, however these proved not to be viable as they could rotate up to 15 degrees back-and-forth independently of the motor shaft, this would have reduced the rigidity of the hinge mechanism severely enough to cause constant problems with counting the encoding taps, as the encoding pinion would regularly be moving in the opposite direction to the motor and counted taps would therefore often be incorrectly added or subtracted. A two channel quadrature encoder system for either side of the hinge could have helped here, however this would have required more GPIO pins to read it than were available on the Hinge ATMEGA and therefore would not have been possible without substantial changes to both the

Central PCB, already manufactured and soldered at that time, and the embedded software running upon it. For a lower backlash solution aluminium grub screw based hubs were tried for adapting from the hinge motor shaft to the pinion gear, these however suffered failures where the grub screw would either work itself loose, or, if tightened sufficiently to prevent that, literally tear the screw threading out of the hole in which the grub screws were mounted. A steel version of this grub screw type of hub was produced and tested, and while it did not experience the tearing of threads which the aluminium part suffered it still worked loose over time when exposed to the forces which the flat face of the 12V motor's D shaft inflicted upon it, even when thread locking liquids were applied. The final iteration of the hub design used two custom machined metal halves with a partial 6mm cylinder cut in to the “lower" half. The “upper" half provided a flat surface which could be tightened against the flat of a motor D shaft, placed in the partial cylinder, and tightened with an array of 8 separate screws across which forces are shared. Fig.(5.34) shows this design. By matching the D of the motor shaft precisely this type of adapter keeps the hinge mechanism rigid whilst tolerating the forces, estimated around 2.7KN, involved. The printed pinion gear is manually forced on to the outer square of this motor hub, giving a tight friction fit. As the outer square of the hub is at a much greater radius than the D shaft the forces exerted on the plastic parts are much less than those affecting the inner edge of the hub.

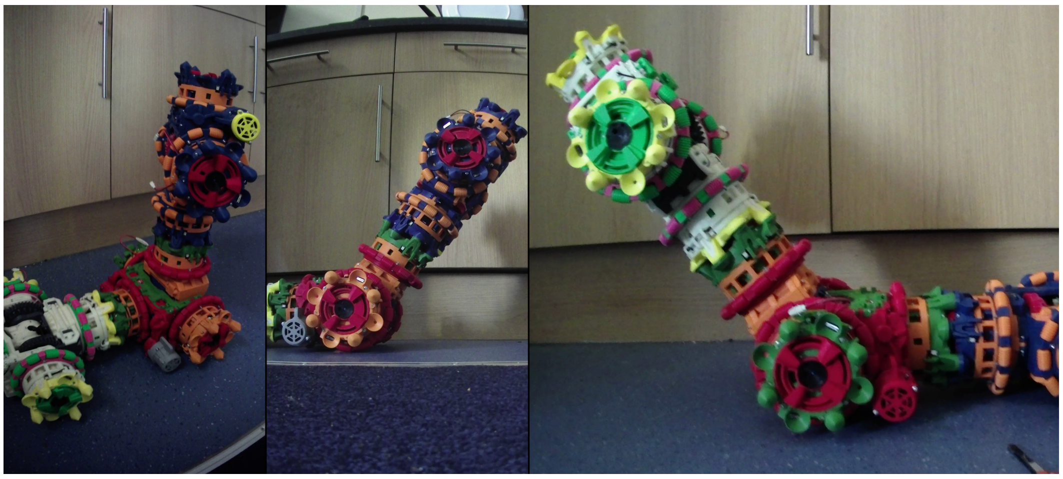


**Fig. 5.35:** The new hinge design in use, pitching and rolling other robots. The robot on the ground behind the actuating robot serves as a counterbalance for the lifted module.

Fig.(5.35) and Fig.(5.36) show the new hinge in operation to lift and roll other robots. Torques of over 1045N cm (overall) are easily supplied and lifting is able to take place at 2.5°/s even at this level of load. Testing with the weighted lever arm found that torques of up to 1030 Ncm (per side) (2060 Ncm overall) could be supplied, and that above this level failures of the 3D printed teeth between the compound gear and the bevel occured rather than the motors stalling.

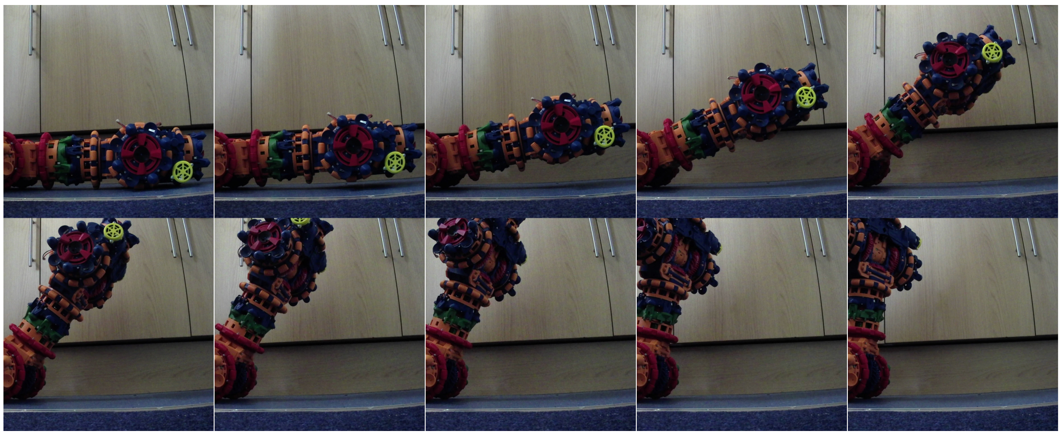


**Fig. 5.36:** Time series photographs showing the new hinge elevating to pitch and then roll another robot. A video is available at [214] and in the supplementary material (Hinge_actuation_tests.mp4)

## 5.17 Powering the New Hinge

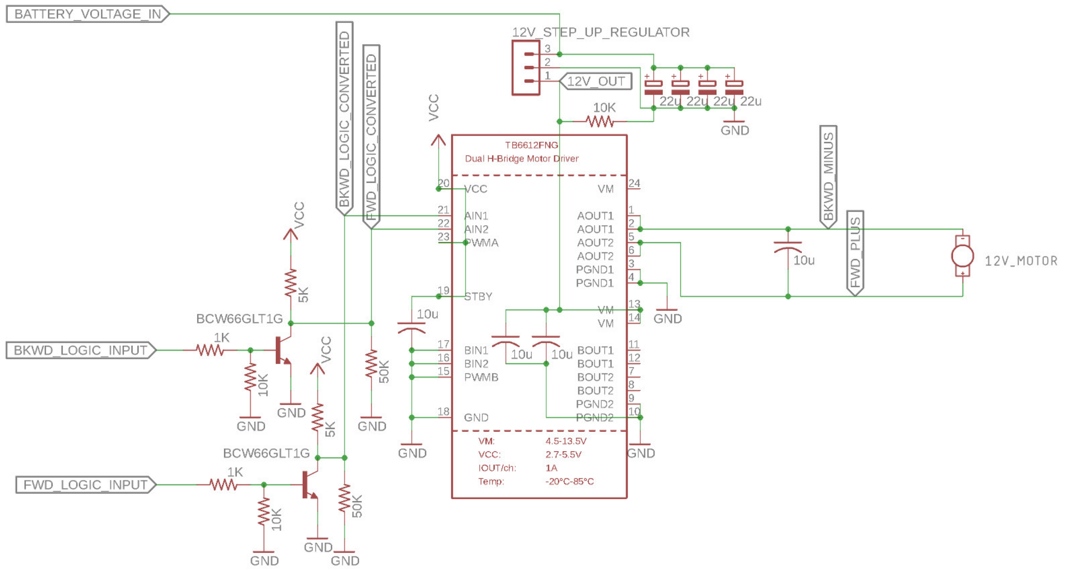


**Fig. 5.37:** Diagram of the circuits used to adapt from the 5V motor control outputs on the Central PCB to the H-bridge controlling the 12V motors. Note how the direction inputs coming from the central PCB are also flipped as they are put through the inverters.

The original hinge system had used a set of 4 5V DC motors to supply torque to the worm gear reducers, however testing had found a pair of high gear ratio 12V motors to be most effective. With the Central PCBs of all robots already constructed it was necessary to produce adapter boards to convert from the accessible pins available on the central boards to a form the 12V motors could handle, Fig.(5.37) shows the schematics for these adapters with Fig.(5.38) showing the adapter PCBs in situ. The new 12V motors are controlled from H-bridges included on the adapter PCBs, these H-bridges require: a supply of the motor's voltage, a low current 5V supply for internal logic and signal inputs to control direction and PWM. Breakout connections

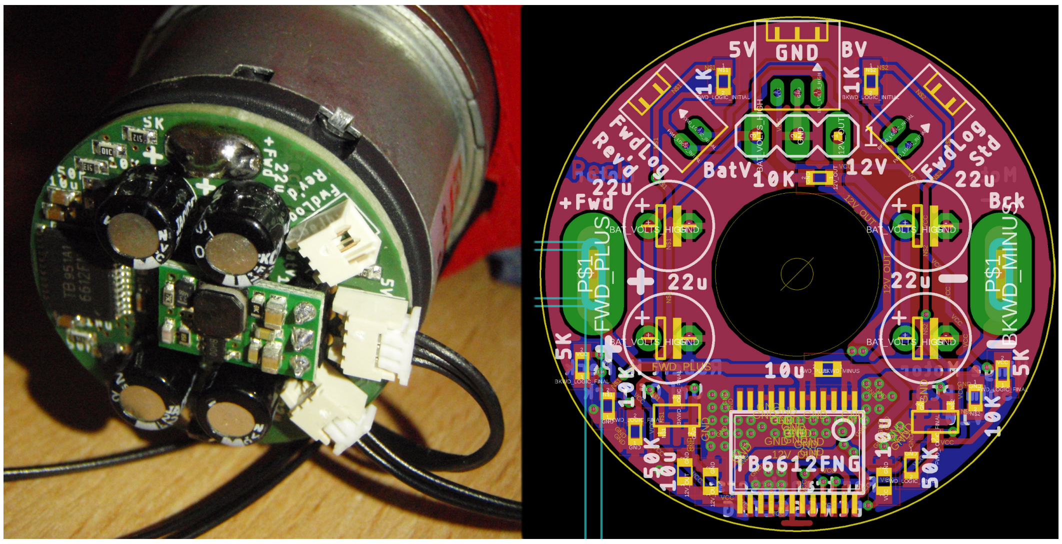


**Fig. 5.38:** Photograph and layout diagram of the adapter boards for the 12V motors, notice how the adapter board fits on to the back of the motor and keeps within the motor's radius.

were fitted to the header of the 5V 5A regulator on the Central PCB, supplying ground, regulated 5V and unregulated battery voltage to both motors' adapters, on each adapter board the battery voltage is stepped up through a regulator [126] to provide the 12V motor voltage. For the H-bridge's logic inputs the only available outputs from the Central PCB were the existing 5V motor connectors, as they are already modulated according to PWM signals within Central PCB's H-bridges there is no need to separately supply PWM to the adapter PCBs. In effect the H-bridges of the Central PCB are reduced to performing the role of logical AND gates combining the Hinge ATMEGA's directional and PWM outputs, the results of which are passed to the adapter boards.

The TB6612FNG H-bridges used on the adapter boards can stop the motors in two different ways, a high impedance mode in which the motor coasts to a stop, and a short-brake mode in which both outputs are grounded and the motor stops sharply, the latter usually being activated by taking the H-bridge's PWM input pin LOW. As the 12V motors, and the gear mechanisms they connect to, can have significant inertia when in motion use of the short-brake mode was required to prevent positional inaccuracies which coasting could cause, particularly when the hinge was lowering or otherwise acting with rather than against gravity. Testing found severe coasting when lowering even though the motors could not be backdriven from a standstill when a high impedance existed between the terminals. However the adapter PCBs only have two signal inputs, one for each direction with PWM modulation already included in the waveforms, hence the PWM input pins of the H-bridges must be tied to 5V for the motors to operate at all. Short-braking can also be achieved by taking both directional inputs of the H-bridge HIGH, but the output pins available from the H-bridges of the central PCB cannot both be simultaneously HIGH. Transistor based inverters are therefore included on the adapter PCBs for use between the outputs of the Central PCB's H-bridges and the inputs of the adapter

PCB's H-bridges. These inverters hold both directional inputs of the adapter PCB's H-bridge HIGH except when one of the signals from the Central PCB is HIGH, by this method the 12V motors can be driven in either direction or short-braked.

As the data-sheet for the 12V regulators [126] warns of a risk of high voltage spikes generated on the inputs when they are powered ON and OFF, the adapter PCBs also contain an array of electrolytic capacitors to suppress these and remove any risk of their damaging other components.

## 5.18 Performance Figures

With all the design details explained this section summarises the capabilities of a single robot. These measures of module performance are also considered in comparison to similar statistics for previous modular platforms. Table.(5.1) summarises the key statistics for the Omni-Pi-tent platform.

| Measure | Value | Notes |
|---|---|---|
| **Architecture** | Mobile and Hybrid | Similar to SYMBRION (Scout) |
| **Mass (kg)** | 3.65 | Heavier than most modular platforms |
| **Lattice Spacing (cm)** | 29.2 | Centre-to-centre when docked |
| **Module Size (cm)** | 32.9 * 32.9 * 13.2 | Larger than most modular platforms |
| **Maximum Wheel Speed (°/s)** | 90 | Since upgrade of wheel design |
| **Maximum Drive Speed (cm/s)** | 14.6 | Driving on a 45° diagonal |
| **Docking Method** | Mechanical, Genderless | Derived from ModRED |
| **Docking Actuation Time (s)** | 0.3 | Similar to HyMod |
| **Hinge Torque (N*cm)** | >1045 | Subsystem tests suggest as much as 2060 Ncm |
| **Hinge Speed (°/s)** | 2.5 | Lifting a one module load |
| **Component Cost (£ per module)** | 630 | Not including printing and PCB fabrication |
| **Battery Lifetime (minutes)** | 110 | Driving with all sensors active |
| **Number of Docking Ports** | 4 | All Active |
| **Degrees of Freedom (Driving)** | 3 | Omnidirectional drive |
| **Degrees of Freedom (In structure)** | 2 | Pitch and Roll of Port 4 |
| **Sensors** | IR docking guidance,<br>IR reflection proximity,<br>Magnetometer and Accelerometer,<br>Docking contact switches | 1 emitter with 4 receivers per port<br>Secondary use of IR guidance hardware<br>BNO 055<br>2 per port |
| **Proximity Sensing Range (cm)** | 12 to 15 | For detection of a plastic docking port |
| **Docking Guidance Range (cm)** | 68 | Maximum extent of reliable detection |
| **IR (38KHz) Communications Range (cm)** | 140 | Without 5KHz in use at the time |
| **Actuators** | 2 * Hinge Motors,<br>4 * Wheel Motors,<br>4 * Docking Hook Motors | SparkFun standard size, 12V, 0.5A stall<br>Pololu micrometal, 5V 1.25A stall<br>Unbranded micrometal, 5V 0.75A stall |
| **Communications** | IR line-of sight, Wi-Fi | Local and global use respectively |
| **CPU** | Raspberry Pi Zero W (BCM2835) | Real-time assistance from 7 * ATMEGA328P |
| **Liu et. al.'s Autonomy Level "Cobweb"[45]** | 7.13 | Result assumes multi-module use |
| **Tan et. al.'s Autonomy Index "Cobweb"[125]** | 3.33 to 6.29 | Range due to unclear scores on some axes |

**Table 5.1:** Table giving key performance parameters of the Omni-Pi-tent modular robotic platform.

The Omni-Pi-tent platform is larger and heavier than most other modular platforms (many are sized to fit within $10 \times 10 \times 10$ or $15 \times 15 \times 15$ cm [43]), this presented extra challenges in terms of hinge torque requirements but provided space for the extensive array of actuators necessary for omnidirectional drive and actuation of all docking ports as well as allowing for a relatively large and long lifetime battery to be included. Fig(5.39) shows an Omni-Pi-tent prototype beside a HyMod module for comparison. The speed of wheel motions is greater than for many modular platforms, SMORES for example moves so slowly that videos tend to be presented at 6× speed to make motion visible, this enables experiments in docking to be conducted more rapidly and a wider range of scenarios tried. The sensory capabilities, while poor compared to many non-modular robots, are quite impressive when compared against other modular robots of similar mechanical versatility, many of which require externally supplied information to guide docking operations. Omni-Pi-tent's onboard computational power is greater than that available in similar robot designs, many platforms have only a single microcontroller whereas Omni-Pi-tent includes a full Linux computer, allowing a wide scope for future work in scenarios requiring significant onboard intelligence. Omni-Pi-tent scores highly in Autonomy Level rankings [45], a score of 7.13 places it slightly above SMORES due to the extra driving degree of freedom offered by Omni-Pi-tent's omniwheel drive over SMORES's differential drive. Under another classification scheme [125] Omni-Pi-tent scores between 3.33 and 6.29, this range of possible scores being due to the highly subjective description of how Reconfiguration Planning and Reconfiguration Decision Making differ from Reconfiguration Execution. It should be noted that the score for Omni-Pi-tent quoted within [125] is incorrect as the authors misclassified Omni-Pi-tent's fully autonomous docking interfaces as well as its fully autonomous execution of reconfiguration.

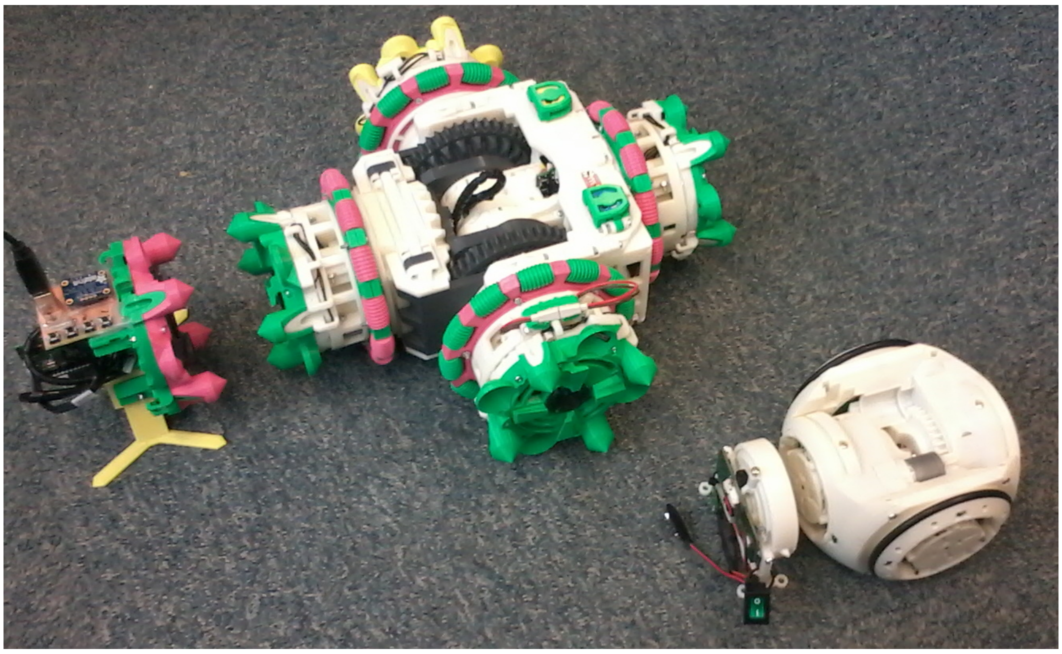


**Fig. 5.39:** An Omni-Pi-tent module beside a HyMod module.

Another notable measure of performance is the maximum range for successful docking operations, this was defined as the maximum range, within the angle of the IR cone emitted from a docking port, from which the robot successfully docked for a large proportion of random starting locations and orientations. Tests were performed from a variety of starting positions and found docking to succeed for 22/27 attempts where the robot was positioned

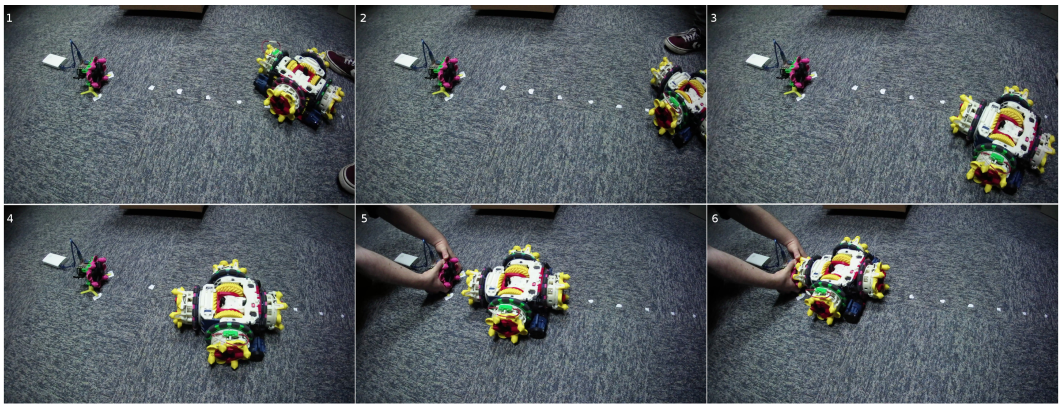


**Fig. 5.40:** An Omni-Pi-tent module performs an autonomous docking test. A test port, with compass equipment fitted, is seen (left) with white floor markers at 20cm, 30cm, 40cm and so on. The robot is placed, misaligned, at approximately 60cm. It receives IR 38KHz messages from the test port telling it the compass angle to align to, then turns to bring the requested port, port 4 in this example, to face the recruiting port. The robot then drives inward, correcting compass and lateral alignment as it approaches. The port is held during the final moments of docking to simulate the mass of a full robot in its place while the approaching robot slides its conical spikes in to the pits.

with its closest port face at 60cm from a recruiting port within the central regions of the port's illuminated 5KHz cone. Of the 5 attempts which failed in two of them the robot did not receive strong enough 5KHz IR signals when initially placed in the cone and hence wandered away rather than docking. In the other three failed attempts the robot struggled in the final moments of alignment to match the compass angle whilst inserting its spikes in to the test ports pits and pressing upon dock detection microswitches, in these scenarios the robot continued to make adjustments to lateral and angular position until it timed out and retreated. For attempts starting at 70cm or further there were occasional detections of 5KHz illumination by the approaching robot however they were not reliable enough to guide the robot in the correct direction to enter the stronger regions of the cone and dock. Fig.(5.40) shows an example of a successful docking attempt. This range appears to be rather larger than the ranges from which many modular robots can reliably begin docking operations, although less impressive when considered as a ratio to the large size of an Omni-Pi-tent module.

## 5.19 Demonstrations

Throughout the development of Omni-Pi-tent a number of demonstrations were given, partly to display the design to the research community, partly to provide hard deadlines for certain aspects of system development, and partly to test whether the hardware could be reliable enough to operate in unfamiliar locations.

These included demonstrations of docking ports and guidance circuits at the TAROS 2018 Conference [208], demonstrations of the first Omni-Pi-tent prototype at TAROS 2019 [209], and demonstrations of the second prototype at a 2019 Workshop on Self-Configuring Modular Robotics for Earth and Space as well as during a 2020 YorRobots [210] exhibition day. A video from the TAROS 2019 demonstration is available at [215] and in the supplementary material (Hardware_prototype_conference_demo.mp4).

During demonstrations a number of complications were discovered which informed later hardware and software modifications, for example the 2019 TAROS demonstration proved the effectiveness of the docking system in unfamiliar environments but found limitations on the range at which guidance was effective due to geometric errors in the phototransistor mounting which were later corrected. The YorRobots exhibition took place in a very unfriendly magnetic environment which aided in identifying the flaws in the BNO 055 compass navigation method and began the search for self-calibration methods which the robotic modules could execute, later resulting in the distortion compensation method described in Sec.(5.7.1).

## 5.20 Summary

This chapter has provided the details of the physical Omni-Pi-tent platform hardware and software. The complexities underlying each subsystem have been detailed and the design decisions and some important iterations explained, Fig.(5.41) displays some of these side-by-side. Key performance statistics have also been given. The focus can now return to how multiple modules make use of the capabilities which these subsystems offer.

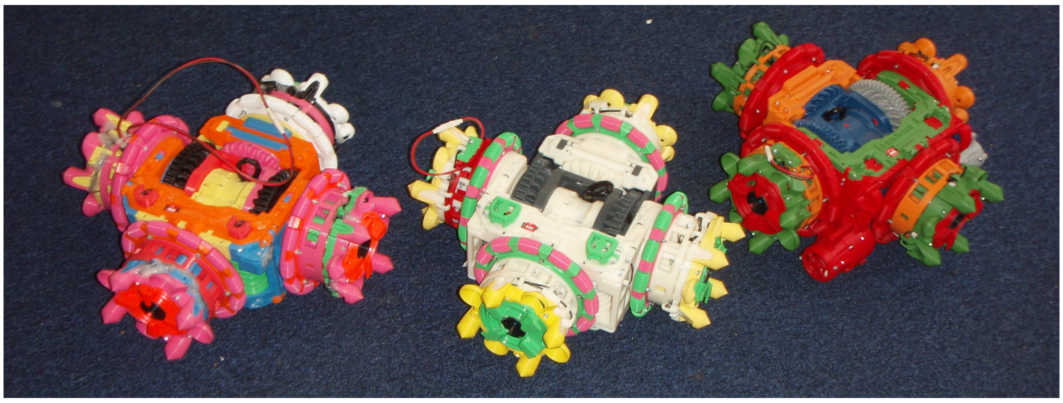


**Fig. 5.41:** The first prototype (left), the second prototype (centre) before upgrading with the new hinge components, and the first production module (right) after upgrading with new hinge and wheel parts. The changes between the versions improved subsystem effectiveness but did not not change the intended functionalities, hence all versions have a broadly similar outward appearance.

Some of the practical design lessons learnt, which underpin design decisions in this chapter, are summarised in Appendix A as a resource which may be of use to future researchers in the modular robotics field, especially those from interdisciplinary backgrounds new to the entire robotics field.

The next chapter describes group capabilities developed within simulation and of general use in a wide variety of tasks for Omni-Pi-tent robots, real or simulated. This begins by describing the self-assembly processes used to form structures from the modules, structures which can then engage in self-repair. The physical Omni-Pi-tent robots described in this chapter return towards the ends of Chapter 6 and Chapter 7 in which their abilities for self-assembly and self-repair are tested.

# Chapter 6

# Self-assembly

This chapter focuses on the development of strategies for self-assembly with modular robots. The process of self-assembly, also referred to in the modular robotics field as morphogenesis [27], allows a collection of independent modules to join together, without needing to be directed by external systems, to form a connected organism which can handle tasks beyond the capabilities of a single module. Without some form of self-assembly an organism of self-reconfigurable robotic modules could not form in the first place, let alone be able to complete group tasks or to self-repair if required. Self-assembly strategies can provide a stepping stone from which to develop self-repair, they can also act as a worst-case scenario for a self-repair strategy, much like in [166]. Full breakup of an organism followed by a period of self-assembly on a module-by-module basis can provide a comparison against more optimised strategies for self-repair [51].

One of the key novel aspects of the Dynamic Self-repair project is the ability of the self-repair methods to function while the robots are in continuous motion, hence self-assembly strategies developed must also be able to proceed while a group of robots is in motion. This chapter uses strongly physically inspired, high fidelity V-REP [28] simulations based around the Omni-Pi-tent hardware design, see Chapter 4, to develop multi-module capabilities for the Omni-Pi-tent platform. The multi-module capabilities are used as a basis for self-assembly, the performance abilities of two new self-assembly strategies, both with the ability to self-assemble while in motion, are compared to a classical self-assembly strategy based on the work of Liu and Winfield [27] [29]. This comparison is performed during a scenario in which robots must both assemble and complete a locomotion task.

This chapter is structured as follows: Sec.(6.1) presents a series of hypotheses which this chapter will test. In Sec.(6.2) three self-assembly strategies, the two novel ones plus an implementation of [27] adapted for Omni-Pi-tent's hardware, are discussed in detail and the implementations explained both from a general viewpoint and with reference to their interaction with the hardware capabilities of the Omni-Pi-tent modules used in this work.

Sec.(6.3) explains the experimental scenarios and setup. Sec.(6.4) provides experimental results along with their statistical analysis. This section also includes a discussion of the results. Sec.(6.5) provides a hardware demonstration of key elements of these new strategies to verify their feasibility and discusses reality gap effects found in the transition between simulation and the real world. Sec.(6.6) continues this hardware demonstration with a more complete test involving a 4-module structure with robots of the finalised hardware design. Sec.(6.7) concludes the chapter and points toward the subsequent self-repair work of Chapter 7.

This chapter's key contributions include:

- Implementation of an existing [27] strategy adapted for the new Omni-Pi-tent hardware platform.
- Development of two new self-assembly strategies capable of forming structures while modular robots are in motion,
- One of these strategies is able to simultaneously recruit robots in multiple concentric “layers".
- Comparison in simulation of these new strategies against [27] in a scenario where a motion task is not required and only self-assembly time is considered.
- Comparison in simulation of these new strategies to the one outlined in [27] in scenarios where both the time to assemble and the time to complete a locomotion task are measured.
- Proving the feasibility of docking to a moving recruiter and Mergeable Nervous Systems control on Omni-Pi-tent robotic hardware.

## 6.1 Hypotheses

As yet none of the self-assembly methods described in the literature ([27] [5] [187] [195] [193] [166] [60] [192] [26] [52], See Sec.(2.3)) have attempted to take place while other tasks, for example group locomotion, are undertaken. They have all thus-far assumed that the structure being formed simply stays still throughout the formation process. The self-assembly processes thus far developed also neglect to include any of the kinds of safeguards that would be expected in a system for practical use such as methods to ensure that the structure is not severely disrupted should incorrect docking somehow occur and then permanently prevent others attempting to dock to the recruiting port, such safeguards and timeouts have been proposed as future work [51] but do not seem to have been implemented. There have been no attempts to combine the features of easily interpreted structural descriptions, such as Liu and Winfield's work [27], with the group capabilities, once assembled, of the Mergeable Nervous

Systems project [26], whilst also adding in capabilities for self-assembly while in motion and timeouts from various states to ensure errors can be overcome. As well as the potential to improve real world reliability by combining these types of work there is also the possibility to use their combination to increase the speed of assembly.

As a means to test the effectiveness of self-assembly strategies able to operate during continuous motion, the following null hypotheses are proposed:

**Null Hypothesis 6.1:** *Being able to assemble while in motion will not, when executed as a step in a locomotion task, provide performance advantages over self-assembly methods which require the structure to remain static.*

**Null Hypothesis 6.2:** *The capabilities required by a robot which can self-assemble while in motion will not be enough for it to self-assemble in ways which can allow recruited robots to begin further recruitment before they themselves have docked.*

**Null Hypothesis 6.3:** *Self-assembly strategies able to recruit multiple "layers" of robots, as described in $H_0$6.2 , will not outperform in terms of speed those in which robots can only be recruited by robots which are already docked.*

**Null Hypothesis 6.4:** *Such methods of self-assembly during motion will not cause a significant performance difference compared to more classical static self-assembly when used in a task where locomotion is not counted as a success metric.*

The investigation of these hypotheses starts by explaining the algorithms underpinning self-assembly with Omni-Pi-tent modules.

## 6.2 Self-Assembly with Omni-Pi-tent

The Omni-Pi-tent modular robot platform has a number of notable capabilities and features, these open up new possibilities when implementing self-assembly, even while sticking closely to prior strategies developed on previous hardware. A finite state machine for this implementation of the [27] self-assembly controller is outlined in Fig.(6.1) and explained in the next subsections.

### 6.2.1 Wandering

Robots by default start in a "Wandering" state, acting as individual units performing obstacle avoidance and random wandering within the environment. Wandering robots have a Temporary ID value set to 0, this value can later be changed to reflect a module's role in a structure. Seed modules transition out of the wandering state to enter the "In Organism" state (transition not shown in the FSM) upon making a decision, typically based on environmental information,

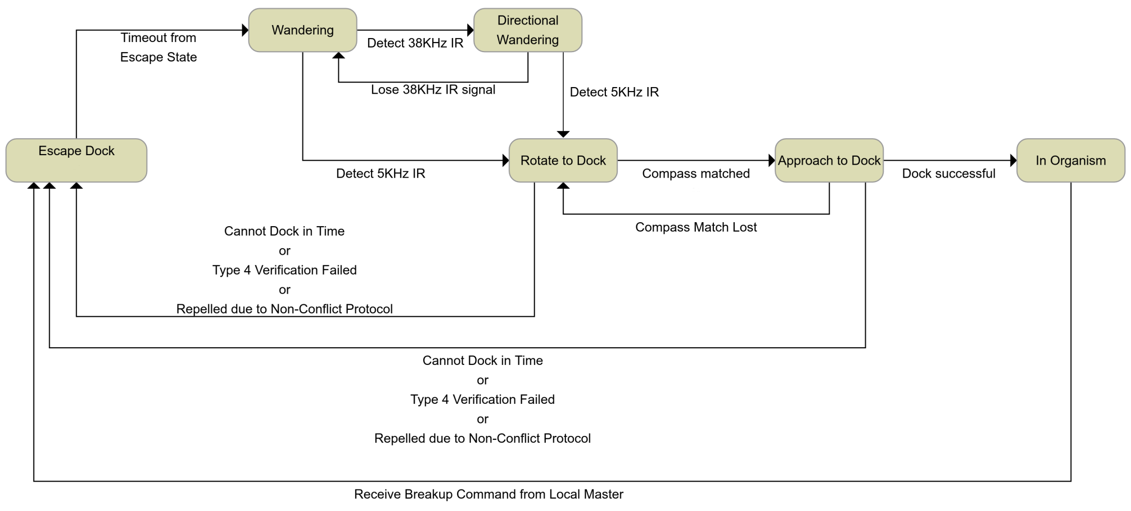

Wandering
Directional Wandering
Escape Dock
Rotate to Dock
Approach to Dock
In Organism
Timeout from Escape State
Detect 38KHz IR
Lose 38KHz IR signal
Detect 5KHz IR
Detect 5KHz IR
Compass matched
Dock successful
Compass Match Lost
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Receive Breakup Command from Local Master

**Fig. 6.1:** A finite state machine for the self-assembly controller. Robots start by default in the wandering state. Type 4 verification is the process by which robots check for messages sent at low power by the recruiting port, and ensure that the information in these matches to the approaching robot's understanding of the docking operation as gathered from earlier longer ranged messages. See body text for full details.

to form a larger organism to complete a task which lone robots cannot. Modules becoming seeds take on a Temporary ID value of 1 and share, over global Wi-Fi, a Recruitment List quadruplet data structure defining the structure to be formed. However, except in the case of a seed, modules will only exit this “Wandering" state if they detect IR signals. Detecting 38KHz line of sight infrared messages causes robots to enter a "Directional Wandering" state which continues a wandering action but biased in the approximate direction from which the signal was received. These messaging signals sent at the highest power level available have the longest range of any IR signals used on the platform. The detection of slightly shorter ranged 5KHz analogue signal levels then triggers a transition of a robot from a “Wandering" or “Directional Wandering" state to begin the docking procedure by entering the "Rotate to Dock" state.

### 6.2.2 The Docking Procedure

On Omni-Pi-tent just two sensor systems are necessary to allow docking, a global angle reading supplied by a “compass" for which an adafruit BNO 055 board proves remarkably resilient to the magnetic interference found in most environments, and a system of infrared LEDs and phototransistors. For docking to occur a recruiting robot must supply data to an approaching robot via local communications, such as the 38KHz IR, or, in some cases, via global communications, such as Wi-Fi. This data includes:

- A message type identifier

- Temporary and permanent ID numbers for the recruiting robot and in some cases the approaching robot

- The global compass angle of the recruiting robot

- Details of any motion the recruiting robot is currently performing

- Details of which docking ports are to be used by the recruiting and approaching robots

Except when certain values for message type identifiers are used, see Sec.(6.2.6) about non-conflict docking features, the message does not contain an ID for the approaching robot but rather any robot within the area illuminated by the $38$KHz message will respond to such messages. A robot receiving the message matches its global orientation to that of the recruiting robot, in many cases with offsets in multiples of $90^\circ$ to account for the choice of ports specified by the recruiting robot. This compass matching behaviour is shown by the “Rotate to Dock" state in Fig.(6.1).

Once correctly rotated, the approaching robot enters the "Approach to Dock" state and drives in the direction of its recruited port towards the recruiting port, returning, if it drifts out by more than a few degrees, to the “Rotate to Dock" state to correct its compass rotation either by stopping and turning or by subtly changing the speeds on each wheel so as to “add" together a turning motion and the driving vector. As it drives in the “Approach to Dock" state it may find itself drifting away from the centre-line of the illuminated $5$KHz cone of light cast by an LED on the recruiting port, this is detected by comparing the analogue IR strengths on phototransistors around the recruited port’s rim and the direction of driving is adjusted so as to bring the analogue values closer together. Successful docking is identified on the physical robots by contact on a pair of microswitches inside two of the spike accepting pits of the docking ports, this need for proximity and angle alignment is reflected in the simulation.

### 6.2.3 The “In Organism" State

Once connected to the port which recruited it, a robot can act as a sensor and actuator slave to the robot it has connected to, referred to as its local master. This local master can relay the docked robot’s sensor readings towards the global master of the organism, the seed robot, or to relay actuator commands from the global master down to slave robots. Sec.(6.2.7) describes this messaging in greater detail. When a robot is within the “In Organism" state it can also recruit further robots to it, to which it will then become their local master. Such recruitment can be most easily described as a modification upon Liu and Winfield’s work [27].

### 6.2.4 Defining Structures for Omni-Pi-tent

By omitting use of the ordering array and making other alterations to Liu and Winfield's Single Entry Recruitment[27] strategy, I found that it is possible to produce Multiple Entry Recruitment using the same, easily human readable and writable, array type of data structure as Liu and Winfield used in Single Entry Recruitment.

Liu and Winfield's Array Format

$\{\{X_1,Y_1\},\{X_2,Y_2\},...,\{X_n,Y_n\}\}$

The Altered Format

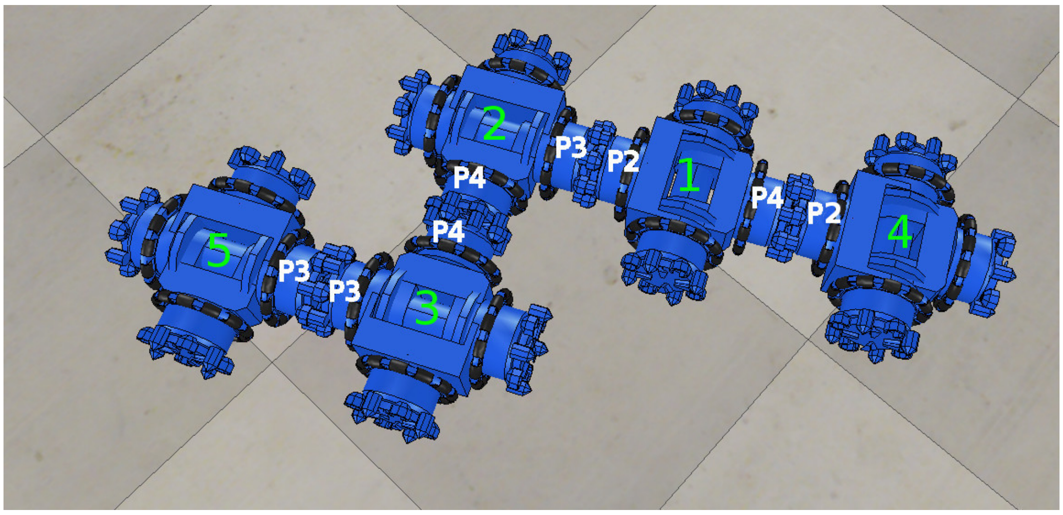
$\{\{A_1,B_1,C_1,D_1\},\{A_2,B_2,C_2,D_2\},...,\{A_n,B_n,C_n,D_n\}\}$

**Fig. 6.2:** A comparison of the arrays used by Liu and Winfield to define shapes for Single Entry Recruitment, and the newly used array types for Multiple Entry Recruitment. See main body text for explanation.

In Liu and Winfield's arrays, see Fig.(6.2), each pair of values defines the temporary ID number of a robot which is to perform a recruiting action, X, and the port it is to recruit on, Y. An array of these pairs has as many pairs, $n$, as there are docked connections within the structure the array describes. Liu and Winfield also needed a recruitment order list. Reading both the pairs array and the order list is necessary to understand the structure's shape, if somehow robots accidentally docked in an unintended order, then the shape formed from then onwards would no longer match the plan. Recording the order of incoming robots is vital to making Liu and Winfield's Single Entry Recruitment strategy assign the correct temporary IDs to new robots and hence vital to getting those new robots to then start the correct further recruitments.

In the altered format, see Fig.(6.2), each quadruplet of the Recruitment List contains: A, the temporary ID number of the robot which is to do recruiting; B, which port it is to recruit on; C, which port it wants an approaching robot to dock with; and D, what temporary ID the new robot should take once docked within the organism. C is used in the new system because Omni-Pi-tent is able to use any port to actively dock, hence any pair of ports can be recruit and recruiter. $n$ quadruplets are needed, one for each docked connection in the finished structure. Recruitment will occur on whichever ports it is required, at whatever times those ports are available to recruit robots to them. All the necessary information for describing the structure is contained in this single data structure, there is no requirement to refer to an additional list when interpreting the meaning. This data structure is shared across the global Wi-Fi communication system with updated versions transferred to all robots each time a module modifies its own local copy. Temporary ID numbers, used by both Liu and Winfield's strategy and the improved strategy described in this chapter, are only assigned to robots within organisms, free wandering robots do not have such a value specified until post-docking. Fig.(6.3) shows an example of how this new form of array translates to a structure.

The quadruplet data structure as formatted here is specific to the Omni-Pi-tent platform. This method of representing the organisms to be assembled automatically respects the nature



{{1,2,3,2},{2,4,4,3},{3,3,3,5},{1,4,2,4}}

**Fig. 6.3:** An example of how a quadruplet array format converts to a robot structure, note that the various quadruplets in the array can be presented in any order. Robot Temporary ID numbers are shown in green, port numbers involved in docking are shown in white. Reading from the first quadruplet: the seed robot, with Temporary ID 1, is to recruit using its second port and call for port 3 of a robot to attach to it. This recruited robot is to identify itself with Temporary ID 2 once docked. The second quadruplet indicates that robot 2 is to recruit using port 4, and the recruit, which is to take a Temporary ID of 3, is to dock using its port 4. The third quadruplet indicates robot 3 should recruit robot 5 and which ports are to be used. The fourth quadruplet describes how robot 1 is also to recruit, on its fourth port, a robot to attach using its own fourth port and using a temporary ID of 4. These quadruplets could be supplied in any order to define this same structure.

of port-to-port connections and lets physical geometry be easily derived from connection information. Although a physically impossible structure in which, for example, a module tried to dock together a pair of its own ports, or multiple modules attempted to dock to a single port, could be written out, such unphysical structures will not be generated by any of the algorithms used to process and modify quadruplet data structures when working with them. It would also not be difficult, in a future implementation, to insert software routines to quickly check for such impossibilities in quadruplet structures. The quadruplet data structure and the algorithms used to handle it serve to, in effect, enforce upon the software the geometric possibilities available to the robotic hardware. Although presently designed for the Omni-Pi-tent platform it would also be feasible to adapt the quadruplet structure to other modular robotic platforms, for example allowing values from 1 to 6 rather than 1 to 4 in the second and third numbers of each quadruplet, to describe cubic modules with docking ports on all faces. Whilst robots with alternative geometries may require some modifications to the algorithms used when deriving lengths and angles from the quadruplet formatted data, the algorithms for finding a robot's global master, tracing the number of layers to it, or other forms of processing will still apply. Adapting the use of quadruplet data structures to alternative modular robotic platforms requires simply swapping the constraints on the structures to reflect the abilities of different hardware platforms, rather than assigning constraints to a highly abstracted mathematical object which may lack an initial conceptual picture to work from.

The size of this quadruplet data structure increases linearly with the number of robots involved in a structure, this is not a prohibitive requirement when considering macro-scale modular robots operating in groups of tens or even hundreds. As the middle two characters within each quadruplet are each limited to the range of 1 to 4, corresponding to the numbering of the docking ports on a module, a quadruplet array of this form can be produced with just two bytes to represent each docked connection for structures of up to 64 modules, and in just 3 bytes per connection for structures of up to 256 modules. Only modules which have docked, or in the case of Multi-layered recruitment, discussed in Sec.(6.2.9), modules which have been recruited, need providing with an up-to-date copy of this data structure.

### 6.2.5 Multiple Entry Recruitment for Omni-Pi-tent

During each loop of any “In Organism" robot’s controller code the robot checks how many quadruplets are in the global quadruplets array and checks what temporary ID it has, if there are no quadruplets in the array the self-assembly is complete. If there are quadruplets in the array then if the robot has a non-zero Temporary ID it searches through the array for quadruplets where the A value, see Fig.(6.2), matches its Temporary ID. For any such quadruplets which it finds it begins recruiting on the ports specified by the B value in the quadruplet. A robot may find that there are multiple quadruplets with its Temporary ID as the A value, in which case it will recruit on all the ports specified by the B values in those quadruplets. When recruiting from each port the robot broadcasts from that port’s LED the local communication 38KHz IR TSOP message detailing data such as compass orientations, speeds and port number requirements, the port number requirements having been read directly from the quadruplets. These messages also include the Temporary ID number, from D in the quadruplet, to be taken by a recruited robot once it docks. The robot also uses the same LED for 5KHz flashing. In simulation both the 5KHz cone and 38KHz message are represented as being constantly emitted. On the physical robots sending both the 5KHz signal and the 38KHz message is not possible at the same time, however as the message is repeated with an inter-message period at-least several times longer than the message length there is sufficient time inbetween messages for 5KHz flashing to occur and for an approaching robot to observe both the 5KHz analogue and 38KHz digital signals.

When a previously wandering robot docks to a port that robot will take on the Temporary ID numbers which it had been informed of via the IR messages. Come the next loop of their controllers robots which have just docked can consult the global quadruplets array to see whether they should start recruiting.

Robots which are recruiting check their port microswitches, if they have been pressed then a recruit has docked to them, hence they edit their copy of the quadruplet array to delete any quadruplet which identifies this specific docking connection. They then share the edited version with the rest of the organism-forming and wandering robots via global Wi-Fi. Over time the quadruplets array gets emptied as docking connections are made. When no quadruplets are left the self-assembly is complete. A second globally accessibly copy of the recruitment list, known as the Structure Recruitment List, is also maintained, this copy does not get edited when connections are formed and instead contains at all times a list of all desired connections to be made. This second list can be used as a guide when repairing a structure. In this way a complete structure can be formed, at any point in time all the robots in the structure are able to recruit on all of the ports which require a robot to be attached.

For the most part, a specific module involved in self-assembly only requires knowledge of those quadruplets in the data structure which include its own ID number. Therefore, whilst all modules receive updated copies of the recruitment list over Wi-Fi when it is updated, delays in these updates reaching modules will not typically matter as those parts of the recruitment list most crucial to a certain module will be those parts which it updates for itself when immediate neighbours are docked or disconnected.

### 6.2.6 Docking Failures and Escapes

The implementation of self-assembly for Omni-Pi-tent also includes non-conflict docking features, which ensure that if multiple robots attempt to dock to a recruiting port then the recruiter accepts one of them by sending a 38KHz message containing its hardware ID, this message also acts as a signal to other robots, those with a different hardware ID, to retreat from the area so as not to interfere with docking. Such retreating robots enter the "Escape Dock" state as seen in Fig.(6.1).

These features are further extended by the fact that, through the use of a number of different resistors available to dim the ports' IR LEDs, ports can send messages at different power levels which are detectable at different ranges. Robots which have already negotiated the non-conflict docking protocol and are in the final stages of docking check to see whether they can receive short range messages from the port they are docking to. These short range messages contain a message type identifier, Type 4 in our system, highlighting them as having been sent at low power. Hence an approaching robot is able to tell, despite the 38KHz receiver (TSOP38438) not giving an analogue level output from which range could be inferred from signal strength, whether the message it has read was sent at low or high power. This information allows it to establish, roughly, what sort of range regime it is currently located in. If an approaching robot makes contact with a port, but is unable to receive a short range message to verify that it is at the correct port, then it retreats without actuating the docking hooks. This method ensures that in the unlikely, but not impossible, event that a robot could while docking attempt to follow a 5KHz beacon coming from a different location than the 38KHz messages it has read, such a robot will not form a docked connection and will instead transition to the “Escape Dock" state. This docking verification prevents the possibility of structures being corrupted by robots being attracted to the wrong port, and then beginning recruitments of their own which would be inappropriate to the location they are in due to the mismatch between the temporary ID which corresponds to their believed position and their actual position.

All of these verification, non-conflict retreat and other behaviours are governed by variables which ensure that such behaviours time-out in a preset time, such that robots can then return to “Wandering" operation. Timeouts are also used for docking attempts, a robot which is trying to dock but consistently failing for long periods of time will transition to the "Escape Dock" state, giving up, driving away and letting other robots attempt a docking at the recruiting port. The “Escape Dock" state can also be used to breakup a structure if the seed robot commands this to be done, while a robot which already has attached slaves is in the “Escape Dock" state it passes messages to them ensuring they also enter this “Escape Dock" state to break free from it.

With this new and improved implementation of Liu and Winfield’s strategy, henceforth referred to as the LW+ strategy, explained the development of a strategy able to self-assemble while in motion will be considered.

### 6.2.7 Mergeable Nervous Systems for Omni-Pi-tent

The Mergeable Nervous Systems concept [26] provides a method for modular robots to exercise accurate control over the motion of an assembled organism. By combining elements of it with LW+ it is possible to develop a strategy which can self-assemble while moving.

The Mergeable Nervous Systems concept requires that sensor data can flow from peripheral modules which detect surrounding stimuli, upward through robot to robot links as far as the seed [1]. Methods for this could either suffer from being non-scalable, where every body robot tries to provide full sensor data to the brain and communication becomes rapidly unreliable as the number of robots in the organism rises, or could tend to loose large amounts of data via the compression required for communicating in a scalable way. For the new implementation used in this work, the sensor data which needs transporting is proximity information. A method was chosen which was scalable while still preserving useful features of the sensor data along the way. Each peripheral robot in the structure checks its proximity sensors for readings and scales these to a 0 to 254 range, for each sensor the robot then uses knowledge about its location in the structure, knowledge derived from a non-editable copy of the recruitment list, to find what the angular position of that sensor is in terms of a clock face centred on the global master robot. The robot creates a 12 byte array with each place corresponding to one of the hour positions on the clock face centred on the highest level master of the structure, see

[1]A robot in this position is also referred to as the global master and was described in [26] as the brain robot

Figs.(6.4),(6.5) and (6.6) for the methods used by robots to calculate their locations relative to the master using only Recruitment List quadruplet data. Sensor data is entered into any places in that array, which represent angular regions of the clock face, in which the robot has sensory information. Bytes representing angular regions in which the robot has no sensors observing are given a value of 255. Being less than the 15 byte maximum message length which the Omni-Pi-tent IR hardware was designed to handle this message is passed up to the robot's immediate local master. This immediate master carries out a similar procedure but as well as using its own sensor data to fill in the 12 byte array it has messages from its peripheral slave robots sending 12 byte arrays to it. Where a 255 exists in either the immediate master's sensor data or in a message from a slave it will be overwritten by any measured sensor value from sensors or another slave which also corresponds to the same clock direction. Where an immediate master has known data in the same clock direction, the same position in the array, from multiple sources (multiple slaves or a slave and its own sensors) the closest value is placed in the array. This array is then passed over another IR port to port link to the next master up the hierarchy. This data flow continues until the seed receives a 12 byte array containing information about the closest obstacle in each hour hand direction, the values having been combined across all the organism's robots, see Fig.(6.7). Through careful use of the Recruitment List data structures robots are also able to check whether any of their ports are docked or otherwise located such that parts of the same organism are within sensor range, data from these sensors can be discarded in the robot and not relayed up the hierarchy so as not to swamp the brain robot with false detections where parts of the organism can see other modules. Ports which are actively involved in recruiting and have already negotiated the non-conflict docking protocol to confirm that a recruit is approaching also dump their data so as to avoid the whole structure trying to steer away from approaching recruits.

Once the brain has processed sensor data it decides on the correct actuator response for the organism to make. The brain sets the relevant outputs from its actuators, usually the omniwheels but this can also include the 2DoF joint actuators in each module. Robots also send infrared messages of another type outward to any slave robots immediately connected to them. Slave robots receiving such messages read their copies of the recruitment list to find which of their ports is connected to which port on the master, then calculate the transform between the brain's reference frame and their own. The linear and angular velocities of the brain are calculated using its wheel speed information and then used to identify an instantaneous centre of rotation about which the slave robot should move so as to follow the brain's motion. By calculating distances from each of its wheels to this instantaneous centre of rotation, and the angles between each wheel and a line linking this wheel to the instantaneous centre of rotation, the necessary wheel velocities for the slave can be found. Once its own wheel velocities are calculated and set the slave robot passes on its wheel velocities via port to port IR message to any further slaves subservient to it. They can perform similar calculations which ensure that their motion also matches that of the master. By these means the whole

```
function FindHighestLevelMaster(InputTempID)--outputs TempID of highest level master, outputs 0 if no master found
  InputRobotsMaster=InputTempID
  if (InputTempID==0)then --if this gets called on a robot which isn't docked to its master
    InputRobotsMaster=0
  else --we will check which docked connections exist
    RobotPresentArray={0,0,... ,0}--says whether a robot with this (index) TempID is in the structure yet
    local indexB=1
    while(indexB<= table.getn(StructureRecruitmentList))do --check over the SRL
      local StructureInstruction=StructureRecruitmentList[indexB]
      local index2=1
      local NotFoundInRecList=1
      while(index2<= table.getn(RecruitmentList))do--RL has quads removed when a dock is formed, check to ensure that the TempID is NOT present as fourth element of a quad, hence it IS docked
        local SInstruction=RecruitmentList[index2]
        if(SInstruction[4]==StructureInstruction[4])then
          NotFoundInRecList=0
        end
        index2=index2+1
      end

      if(NotFoundInRecList==1)then
        RobotPresentArray[StructureInstruction[4]]=1--robot referred to in quad IS present
      else
        RobotPresentArray[StructureInstruction[4]]=0 --robot ISN'T present
      end
      indexB=indexB+1
    end
    --check for a docked robot mentioned only in place 1 of a quad, this is the master
    indexB=1
    while(indexB<= table.getn(StructureRecruitmentList))do --check over the SRL
      local StructureInstruction=StructureRecruitmentList[indexB]
      if(RobotPresentArray[StructureInstruction[1]]==0)then--check that this isn't referring in place 1 of the quad to a robot already found present from checking fourth places
        local index2=1
        local NotFoundInRecList=1
        while(index2<= table.getn(RecruitmentList))do
          local SInstruction=RecruitmentList[index2]
          if(SInstruction[1]==StructureInstruction[1] and SInstruction[2]==StructureInstruction[2] and SInstruction[3]==StructureInstruction[3] and SInstruction[4]==StructureInstruction[4])then
            NotFoundInRecList=0 --if we can find a matching quad in the editable RL
          end
          index2=index2+1
        end

        if(NotFoundInRecList==1)then
          if(RobotPresentArray[StructureInstruction[4]]==1 and CheckStructRecListForThisTempIDInPlaceFour(StructureInstruction[1])==0)then --in these circumstances we only count this robot as present if the quad can't be found in the RL and if the TempID the quad says is to be recruited is already present and if nowhere in the SRL is the value in first place of this quad also present in the fourth place of another quad
            RobotPresentArray[StructureInstruction[1]]=1--robot which the quad refers to AS THE RECRUITER is present
          end
        end
      end
      indexB=indexB+1
    end
    --now start tracing through the structure for our highest connected master
    MasterTempID=0
    CurrentTempID=InputTempID
    HighestConnectedMasterReached =0 --used to indicate when we've reached the highest connected master if it isn't in the main structure yet
    while(HighestConnectedMasterReached ~=1)do
      --search fourth place of quads within SRL looking for CurrentTempID
      local indexE=1
      local QuadOfInterest=1
      local QuadWithMasterFound=0
      while(indexE<= table.getn(StructureRecruitmentList))do
        local StructureInstruction2=StructureRecruitmentList[indexE]
        if (StructureInstruction2[4]==CurrentTempID)then
          QuadOfInterest=indexE
          QuadWithMasterFound=1
        end
        indexE=indexE+1
      end
      --see how we have noted down which quad of the list the had CurrentTempID as fourth element of it's quad, for branching non-looping structures there will only be exactly one such element, now check the editable copy of the recruitment list to see whether the connected link yet exists
      if(RobotPresentArray[CurrentTempID]==1 and QuadWithMasterFound==1)then --if the link described by the quad has already been docked
        local StructureInstruction=StructureRecruitmentList[QuadOfInterest]
        InputRobotsMaster=StructureInstruction[1]--set MasterTempID to the first element of the quad
        MasterTempID=StructureInstruction[1]--set CurrentTempID to the first element of the quad
        CurrentTempID=StructureInstruction[1]--this while loop will end when we reach the top level master
      else
        HighestConnectedMasterReached =1 --we are at the highest master
      end

    end
  end
  return InputRobotsMaster
end
```

**Fig. 6.4:** Pseudocode to find the highest connected master of a structure by using Recruitment List data.

```
function FindCompassRelToMaster(AngleInputTempID)--outputs relative angle between robot and master
  InputRobotsAngleMinusMasters=0
  IDofHighestLevelMaster=FindHighestLevelMaster(AngleInputTempID)--we check which robot is our highest level connected master
  if (AngleInputTempID==IDofHighestLevelMaster or AngleInputTempID==0)then --if this gets called on the master or a robot which isn't docked
    InputRobotsAngleMinusMasters=0
  else--we check which docked connections have formed so far, robot's which are present are those already docked to their masters
    RobotPresentArray={0,0,... ,0}
    PopulateRobotPresentArray()--procedure identical to in FindHighestLevelMaster()
    RobotPresentArray[IDofHighestLevelMaster]=1--modify RobotPresentArray to include the master, which won't be referenced in the 4th place of any quad
    --now to trace out relative compass orientations
    AngleMasterTempID=0
    AngleCurrentTempID=AngleInputTempID
    HighestConnectedMasterReached =0 --used to indicate when we've reached the highest connected master if it isn't in the main structure yet
    while(AngleMasterTempID ~= IDofHighestLevelMaster and HighestConnectedMasterReached ~=1)do
      --search fourth place of quads within SRL looking for CurrentTempID
      local Angleindex=1
      local AngleQuadOfInterest=1
      while(Angleindex<= table.getn(StructureRecruitmentList))do
        local AngleStructureInstruction2=StructureRecruitmentList[Angleindex]
        if (AngleStructureInstruction2[4]==AngleCurrentTempID)then
          AngleQuadOfInterest=Angleindex
        end
        Angleindex=Angleindex+1
      end
      --see how we have noted down which quad of the list the had CurrentTempID as fourth element of it's quad, for branching non-looping structures there will only be exactly one such element
      --we now check the editable copy of the RL to see whether the connected link yet exists
      if(RobotPresentArray[AngleCurrentTempID]==1)then --if the link described by the quad has already been docked, we could also describe this as "if the level of master we are currently checking has in turn made the docking to its immediate master"
        --from that quad note down the second and third elements
        local AngleStructureInstruction=StructureRecruitmentList[AngleQuadOfInterest]
        local AngleRelAnglesDockLinks={{180,90,0,−90},{−90,180,90,0},{0,−90,180,90},{90,0,−90,180}}--RelAnglesDockLinks[secondElement][thirdElement] gives angle of bodyward robot minus angle of its master
        local AngleRelAnglesOuter=AngleRelAnglesDockLinks[AngleStructureInstruction[2]]
        local AngleRelAngle=AngleRelAnglesOuter[AngleStructureInstruction[3]]
        --use this relative angle to add or subtract from our running count, InputRobotsAngleMinusMasters
        InputRobotsAngleMinusMasters=InputRobotsAngleMinusMasters+AngleRelAngle
        AngleMasterTempID=AngleStructureInstruction[1]--set MasterTempID to the first element of the quad
        AngleCurrentTempID=AngleStructureInstruction[1]--set CurrentTempID to the first element of the quad
        --this while loop will end when we reach the top level master
      else
        HighestConnectedMasterReached =1 --have reached highest master
        --InputRobotsAngleMinusMasters already holds the appropriate relative angle between us and our highest master of our local substructure
      end
    end
  end
  InputRobotsAngleMinusMasters=AngleClamp(InputRobotsAngleMinusMasters)
  return InputRobotsAngleMinusMasters--0,90,180 or −90
end
```

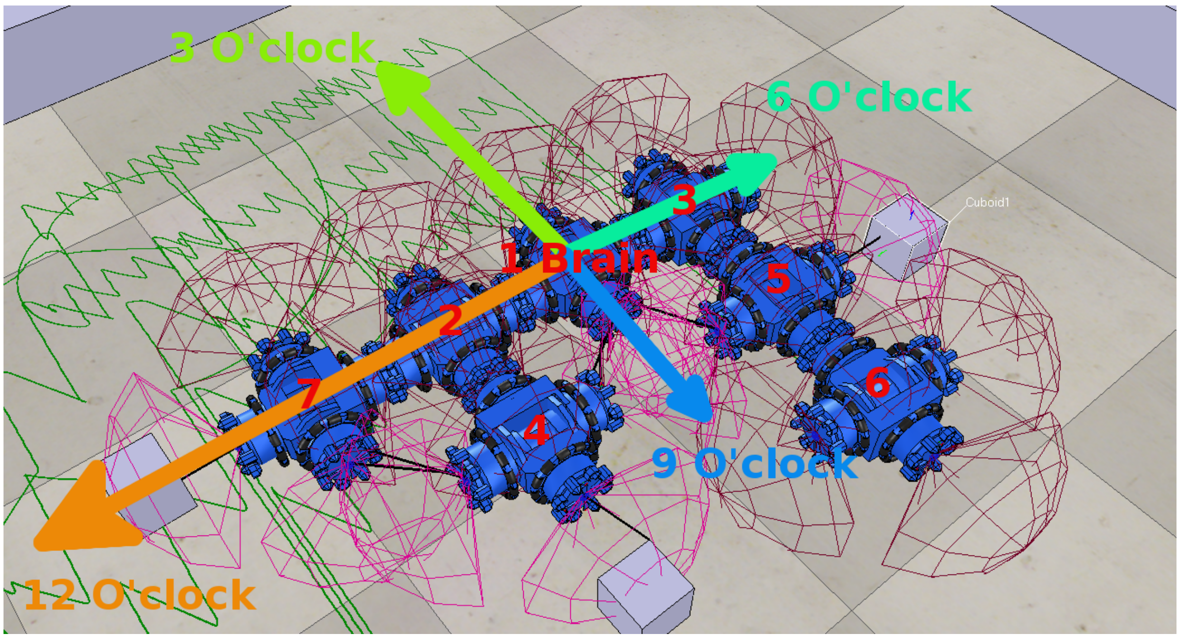
**Fig. 6.5:** Pseudocode to find the relative compass orientation of a robot relative to the highest connected master in a structure by using Recruitment List data.

organism can rotate about a fixed centre of the master's command, see Fig.(6.8), or drive linearly or perform any combination of both actions. As the wheels have a maximum possible speed it should be noted that for structures extending sufficiently far from the instantaneous centre of rotation the extremity robots will, when the whole structure attempts to rotate at high speeds, be at maximum wheel speed but below the desired speed so in cases of this kind the rotation may no longer match the desired speed, or in an asymmetric structure the central point of rotation, and increased drag may be introduced, such limitations are inevitable in real world systems.

If a problem should occur during self-assembly this implementation of Mergeable Nervous Systems also contains the ability to send specialised port to port messages between robots signalling for them to undock from the structure, in this way it will, when self-repair is implemented, be possible to break up part or all of a structure. These messages cause transitions into the “Escape Dock" state.

```
function FindStepsToMaster(InputTempID)--outputs number of lattice vectors to master as seen from a body robot's reference frame
  Port1WardStepsCount=0
  Port4WardStepsCount=0
  IDofHighestLevelMaster=FindHighestLevelMaster(InputTempID)--check which robot is our highest level master
  if (InputTempID==IDofHighestLevelMaster or InputTempID==0)then --if this gets called on the master or a robot which isn't docked to any master
    Port1WardStepsCount=0
    Port4WardStepsCount=0
  else--we check which docked connections have formed so far, robot's which are present are those already docked to their masters
    RobotPresentArray={0,0,... ,0}
    PopulateRobotPresentArray()--procedure identical to in FindHighestLevelMaster()
    RobotPresentArray[IDofHighestLevelMaster]=1--modify RobotPresentArray to include the master, which won't be referenced in the 4th place of any quad
    MasterTempID=0
    StepsHighestConnectedMasterReached =0 --used to indicate when we've reached the highest connected master if it isn't in the main structure yet
    CurrentTempID=InputTempID
    while(MasterTempID ~= IDofHighestLevelMaster and StepsHighestConnectedMasterReached ~=1)do
      --search fourth place of quads within StructureRecruitmentList looking for CurrentTempID
      local indexA=1
      local QuadOfInterest=1
      while(indexA<= table.getn(StructureRecruitmentList))do
        local StructureInstruction=StructureRecruitmentList[indexA]
        if (StructureInstruction[4]==CurrentTempID) then
          QuadOfInterest=indexA
        end
        indexA=indexA+1
      end
      --see how we have noted down which quad of the list the had CurrentTempID as fourth element of it's quad, for branching non-looping structures there will only be exactly one such element
      if(RobotPresentArray[CurrentTempID]==1)then --if the link described by the quad has already been docked, we could also describe this as "if the level of master we are currently checking has in turn made the docking to its immediate master"
        local StructureInstruction=StructureRecruitmentList[QuadOfInterest]
        --we now look at element 3 of the quad to see which port on the current robot is used to connect to it's local master
        local PortToLocalMaster=StructureInstruction[3]
        --we use FindCompassRelToMaster to work out what direction this port points in relative to the structure master's ref frame
        local CurrentRobotAngleRelToMaster=FindCompassRelToMaster(CurrentTempID)
        local AnglesArrayIndex=1
        if(CurrentRobotAngleRelToMaster==0)then
          AnglesArrayIndex=1
        elseif(CurrentRobotAngleRelToMaster==90)then
          AnglesArrayIndex=2
        elseif(CurrentRobotAngleRelToMaster==180)then
          AnglesArrayIndex=3
        else --if(CurrentRobotAngleRelToMaster==-90)then
          AnglesArrayIndex=4
        end
        local Port1StepCountingArrayOuter={{-1,0,1,0},{0,1,0,-1},{1,0,-1,0},{0,-1,0,1}}--directions of slave robot away from global master if it connects using PortNumberInnerIndex to it's local master and is angularly separated from the global master by OuterIndex where (outer index1 is 0 deg, outer index2 is 90, outer index 3 is 180, outer index4 is -90)
        local Port4StepCountingArrayOuter={{0,1,0,-1},{1,0,-1,0},{0,-1,0,1},{-1,0,1,0}}
        local Port1StepCountingArrayInner=Port1StepCountingArrayOuter[AnglesArrayIndex]
        local Port4StepCountingArrayInner=Port4StepCountingArrayOuter[AnglesArrayIndex]
        Port1WardStepsCount=Port1WardStepsCount+Port1StepCountingArrayInner[PortToLocalMaster]
        Port4WardStepsCount=Port4WardStepsCount+Port4StepCountingArrayInner[PortToLocalMaster]
        MasterTempID=StructureInstruction[1]--set MasterTempID to the first element of the quad
        CurrentTempID=StructureInstruction[1]--set CurrentTempID to the first element of the quad
      else
        StepsHighestConnectedMasterReached =1 --have reached highest master in substructure
        --InputRobotsAngleMinusMasters already holds the appropriate relative angle between us and our highest master of our local substructure
      end
      --this while loop will end when we reach the top level master
    end
  end
  return Port1WardStepsCount,Port4WardStepsCount
end
```

**Fig. 6.6:** Pseudocode to find the relative location of robot relative to the highest connected master in a structure by using Recruitment List data.



Robot 4's {255, 255, 255, 255, 255, 255, 255, 255, 255, 069, 255, 255}
Robot 7's {255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 121}
Robot 2's (as seen) {000, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255}
Robot 2's (including data from 4 and 7) {000, 255, 255, 255, 255, 255, 255, 255, 255, 069, 255, 121}
Robot 6's {255, 255, 255, 255, 255, 255, 255, 000, 000, 255, 255, 255}
Robot 5's (including data from 6) {255, 255, 255, 255, 255, 255, 084, 000, 000, 255, 255, 255}
Robot 3's (including data from 5) {255, 255, 255, 255, 000, 000, 084, 000, 000, 255, 255, 255}
Robot 1's (including data from 3 and 2) {000, 255, 000, 255, 000, 000, 084, 000, 000, 069, 255, 121}

**Fig. 6.7:** A visualisation of clock hand data flows in an F shaped morphology. Temporary ID numbers are marked in red on the robots, the robot with Temporary ID=1 being the brain. The distance measurements data structures seen by each robot are displayed, as are the merged versions of clocks which they relay brainward. Note that the data structures start at 1 O'clock direction and run to 12 O'clock. 255 implies no known data in this direction, 0 implies no objects in range, numbers from 1 to 254 show objects at decreasing ranges. Note how, to avoid false positive detections, Robots 4 and 7 ignore the detections of each other in their own data structures, they use the recruitment list to check for other robots in close proximity which could cause false detections. Note that the clock face is centred on the master robot, so while Robot 6, for example, has three active proximity sensors, their locations are such that they are all counting as looking in the 8 O'clock and 9 O'clock directions.

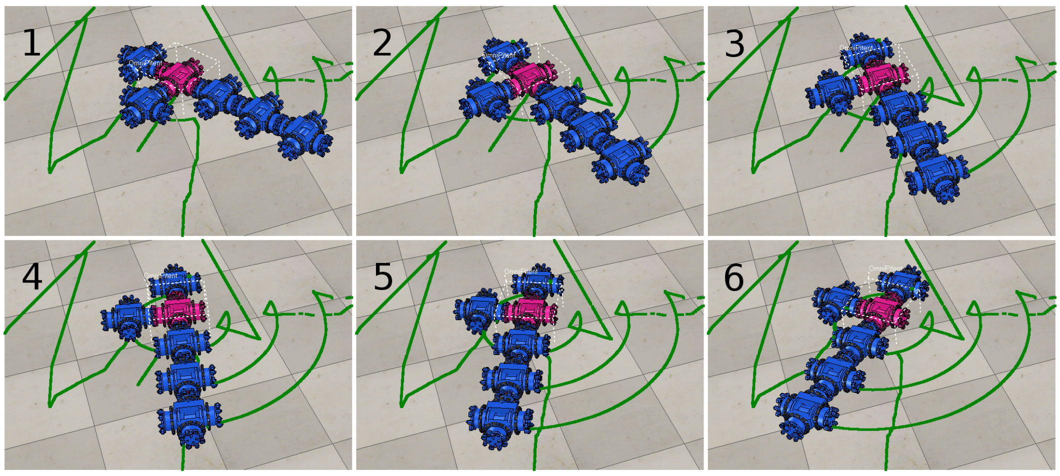


**Fig. 6.8:** The rotation of a structure about a centre of rotation under the seed robot. The seed is marked in pink with the paths traced out by modules, during initial self-assembly and during rotation shown in green.

### 6.2.8 Docking while in Motion

Unlike previous modular robot platforms Omni-Pi-tent is intended to be able to dock to moving robots. The ability of an omniwheeled robot to maintain orientation separately from direction of travel means this potentially difficult act becomes conceptually simple. A moving recruiting robot, either the brain robot or any recruiting robot already in an organism, broadcasts its current wheel speeds as part of its 38KHz IR recruitment message. Robots responding to the recruitment message use the wheel speed and compass information contained in the 38KHz message to match motion so as to be stationary within the recruiting robot's reference frame. The motions of the approaching robot are added to this matched motion, allowing docking to proceed similarly to if the recruiting port was stationary, see Fig.(6.9). This provides an opportunity to parallelise the self-assembly and motion tasks, which should enable faster overall task completion.

With the combination of Multiple Entry Recruitment, Mergeable Nervous Systems and the other new features added to the implementation a self-assembly strategy results which, combined with the unique features of the Omni-Pi-tent hardware would appear to be one of the most capable and versatile self-assembly strategies yet devised. This strategy which combines features from Liu and Winfield's work with Mergeable Nervous Systems features is referred to in this thesis as LW+MNS.

### 6.2.9 The Multi-Layered Recruitment concept

In the LW+ and LW+MNS strategies robots already connected to the structure recruit others to attach to them, recruitment proceeds, in effect, as a "concentric" shell moving further and further away from the initial seed as new robots are added. A new strategy, which is

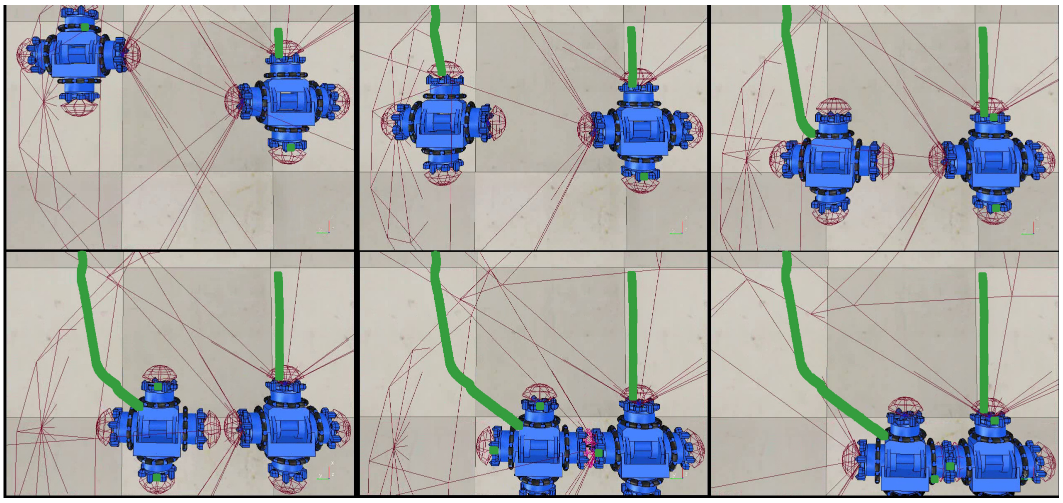


**Fig. 6.9:** Docking of a robot to a moving seed, note how, while in the global reference frame, the recruited robot follows an angled path, from the reference frame of the recruiter it is simply approaching while keeping aligned to the cone's centre.

termed Multi-Layered Recruitment (MLR), has been developed which allows robots in the act of approaching a structure to begin their recruiting before they have themselves docked. Rather than happening on a "ring" around the structure's edges recruitment will now occur simultaneously across several "layers" of concentric steps away from the seed. This is achieved by having robots gain a Temporary ID value as soon as they first make the transition from "Rotate to Dock" to "Approach to Dock", where in other strategies this was not assigned until the transition into the "In Organism" state. Robots can therefore begin recruitments to them before having completed the docking approach to their local master. This gaining of a temporary ID whilst entering the "Approach to Dock" state means than in MLR robots can be recruiting both when in the "Approach to Dock" and "In Organism" states. Using Mergeable Nervous Systems methods described previously any robot which gains recruits to it before it has completed its own docking to a master can co-ordinate the motions of its slaves. Should a recruited robot dock to its immediate master before having any immediate slaves attached to it then the fact it has already been recruiting for some time should have at least helped get other robots in position to dock to it sooner than had recruitment only begun after it docked. The breakup protocols built into this Mergeable Nervous Systems implementation also ensure that if a robot with parts of a body assembled around it but which has yet to dock to its own master times out from a docking attempt it can release all of its slaves again to give them opportunities to be individually recruited elsewhere. See Fig.(6.10) for further illustration.

It is proposed that this approach will be faster than regular LW+ or LW+MNS methods where a robot must dock before beginning 5KHz emissions. There will be practical limits, due to a finite top speed for Omni-Pi-tent robots, on how many "layers" of this docking can go on at once. Robot 2 might well be able to recruit for robot 3 as soon as it has entered robot 1's cone, however although robot 3 might try to recruit a fourth robot, robot 3 may itself have to move

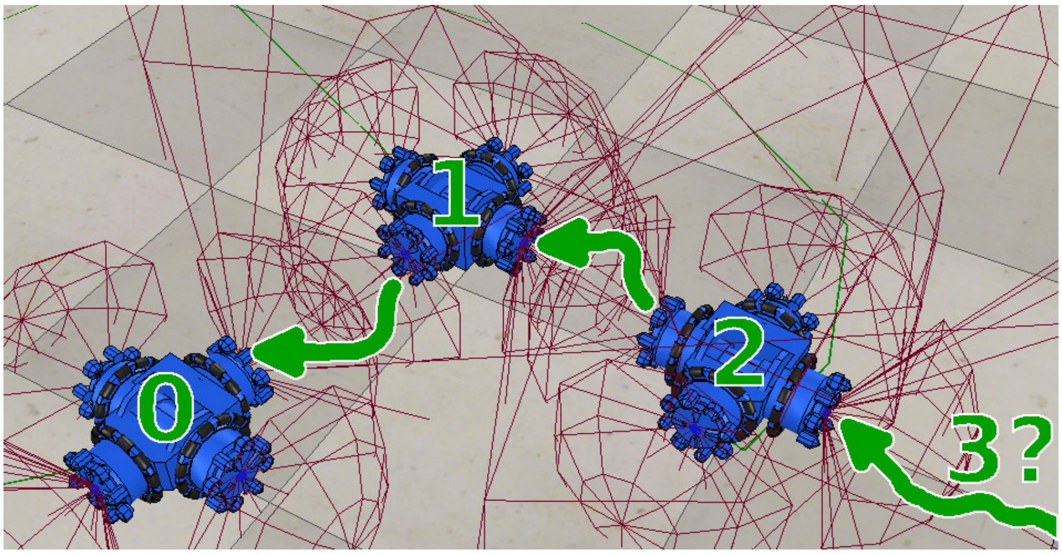


**Fig. 6.10:** The Multi-Layer Recruitment concept. Consider robot 0 to be part of a structure, as robot 1 wanders into robot 0's 5KHz recruitment cone it can, once compass alignment and IR handshaking are complete, immediately begin recruiting for robot 2 to join it using itself as the reference frame to which robot 2 will try to match. Robot 2, once rotated, can begin recruiting for robot 3 should such an extra robot be within IR guidance range. If robot 2 docks with robot 1 before robot 1 docks with robot 0 then robot 1 uses MNS strategies to control the two robot organism of which it is the brain until it has docked to robot 0.

fast enough to dock to 2 that robot 4 cannot keep up. Robot 4 might therefore not be able to dock to 3 until 3 has finished docking to 2. Whether such a limit on the “layers" would occur at robot 3, 4 ,5 or further “out" is unknown but would depend on the minimum speed difference by which a robot further out must travel faster than a robot closer in so as to dock to it, the smaller this minimum the more “layers" could potentially be handled.

From a logical perspective Multi-layered recruitment cuts down the number of actions that robots must perform sequentially as compared to the LW+ and LW+MNS strategies by parallelising a robot's recruiting of further slaves to itself to operate concurrently while that robot performs the alignment and connection phase of docking to its own master. The time taken for aligning and approaching can vary substantially depending on relative robot locations when this process begins, by ensuring that further recruitment can begin before alignment has finished the upper bound of how long structure formation may take is decreased.

### 6.2.10 Looped Structures

It should be noted here that forming structures containing loops is not attempted in this work due to the physical impossibility of a robot docking onto multiple ports arranged so as to form a concave space it must enter into, see Fig.(6.11). Except with retractable docking ports, such as on [42], such loop formation is impossible using classical and previous strategies such as [27] [29]. For example, SYMBRION could not form such looped structures at all due to hardware limitations such as only having passive docking ports available on the sides [51]. However, because it is able to perform recruitment to robots before they have joined to the group, a future extension to the multi-layered strategy loop formation could be handled by inserting a rule to form multiple non-loop containing structures first, ensuring that the multiple robots in each structure which will need to connect to multiple robots in other structures do so only in non-concave ways. Then, only once these structures are completed, are they brought together under control of their brain robots to form any loops, avoiding the need at any point for single robots to engage in concave docking. Unfortunately, time constraints did not permit for this loop formation concept to be developed further in the course of the Dynamic Self-repair project.

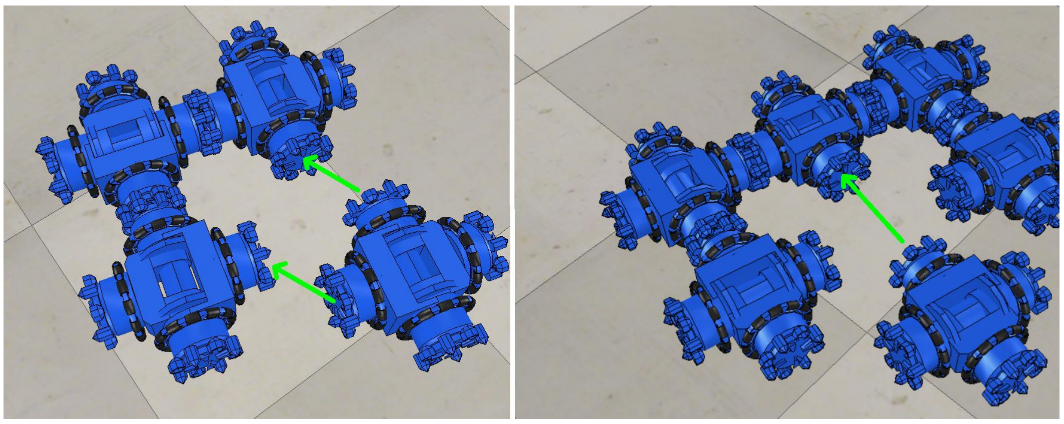


**Fig. 6.11:** Two types of concave docking scenarios, the left hand scenario almost impossible except perhaps in some cases of a very fine chance alignment allowing Ports 3 and 2 of the approaching robot to slip into their respective places at $45°$, the right hand scenario physically impossible because docking port alignment spikes are fixed and unretractable. Avoiding these scenarios will typically require producing non-loop-containing structures first then bringing them together in a way which does not involve concave docking, likely by dividing any loop along straight line cuts.

## 6.3 Experimental Setup

With these three strategies, two of them able to self-assemble while moving, described, the experimental scenario in which they are compared is now considered.

To demonstrate the effectiveness of self-assembly during motion it was necessary to devise a scenario in which strategies are compared not simply on the time to form an organism's structure but in which the completion of a concurrent locomotion task also has an important effect.

Consider a scenario where a swarm of modular robots, perhaps exploring a planetary surface, monitoring hard to reach infrastructure or penetrating the rubble beneath a collapsed building, have entered a space. At the far end of this space is some item of interest, the handling of which requires the independent modules to dock together into some defined morphology. A robot near the entrance end of this space detects the item of interest at the far end and makes itself the seed robot, recruiting others to form the multi-robot structure, such self-promotion to seed robot due to circumstances has strong precedent from earlier research [146] [74] [2]. For the purpose of these experiments the robot which was to become the seed was instructed, immediately as the simulation began, of the need to begin forming a structure and given the quadruplet data structure for the structure to be formed.

When using the strategy referred to as LW+ the robots cannot form a structure while moving so must assemble at the entrance end of the space before driving towards the item of interest at the opposite end. The LW+MNS and MLR strategies both have the ability to recruit and dock robots to them while in motion, hence strategies allow the structures to form as the robots drive along this “corridor", see Fig.(6.12). When driving the overall direction of the motion is straight, however during the run the robotic organism will regularly steer to either side, or temporarily reverse, so as to manoeuvre around wandering robots, which appear as obstacles to be avoided. For these experiments if robots using the LW+MNS or MLR strategies reach the end before forming the full structure they return back and forth through the space until the organism is complete at which point it heads for the item of interest again. In all scenarios the mission time is counted from the start of recruitment until the structure is both formed and is positioned at the far end where the item of interest was detected, in this way both the speed of assembly and the speed of locomotion are accounted for.

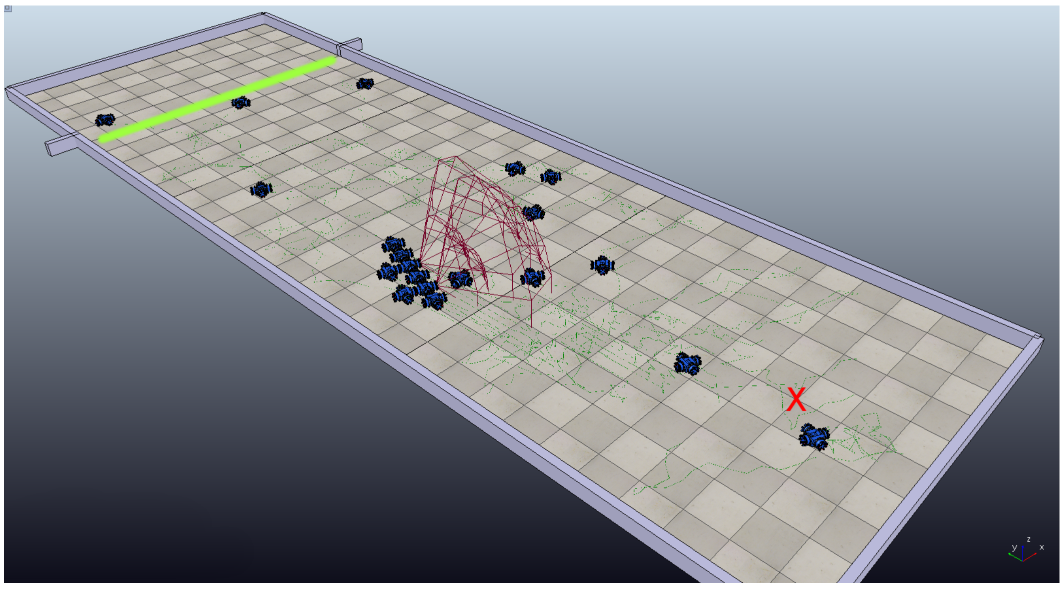


**Fig. 6.12:** A screenshot from a simulation using the MLR strategy, note the scattering of robots throughout the space as the forming organism drives towards the finishing line, marked in green, at the upper left end. The location where the seed initial starts is towards the right hand end of the arena, marked with a cross. The larger red wireframe shapes represent IR communication ranges on the highest power setting, the smaller ones show docking guidance signal ranges. Proximity sensors are not visually shown here.

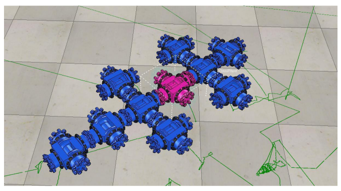
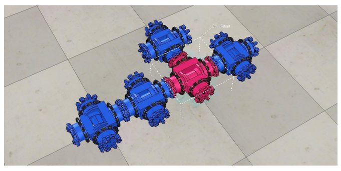
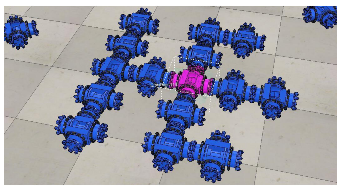
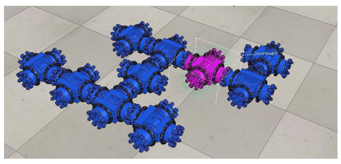
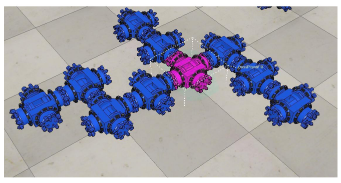
| Name | Image | Widths (m) | Lengths (m) | Robots within Structure | Robots in Scene | Number of Layers |
|---|---|---|---|---|---|---|
| S1 |  | 3,5 | 10 | 10 | 20 | 3 |
| S2 |  | 3,5 | 10 | 5 | 20 | 2 |
| S3 |  | 10 | 10 | 15 | 30 | 4 |
| S4 |  | 3,5,10 | 10 | 10 | 20,30 | 5 |
| S5 |  | 3,5 | 10,20 | 10 | 20 | 4 |

**Table 6.1:** Table of structures formed, the seed robot in each structure being highlighted in pink. Comma separated numbers in columns indicate multiple values were tested.

### 6.3.1 Structures and Spaces

Using V-REP for simulations, tests were performed using five different structures. Tests were performed on structures with 5 robots, 10 robots and 15 robots. The number of robots present was 20 in most cases, however runs with the 15 robot structure were performed with 30 robots present in the environment. Some scenarios were tested with changes to the width and length of the space. See Table.(6.1) for a list of these structures and notes as to which spatial parameters and robot populations each was tested with.

In each simulation scenario 40 runs of each of the three strategies were attempted with randomised starting positions and compass orientations for robots, except the seed, scattered throughout the space. The seed always began at a starting location, see Fig.(6.12), but had a randomised orientation. The randomised positions and orientations used a rectangular uniform distribution between the limits of the arena area for position and 0 to 360 degrees for orientation, with randomness sourced from the Linux kernel. Each scenario was given a maximum simulated time after which the simulation would be ended if it had not succeeded in meeting both the formation and end reaching goals by then. If many runs within a scenario timed out then it was re-run with a longer maximum allowable time specified. Maximum times ranged from 15 minutes to an hour. In terms of CPU time, running these simulations took between 6 and 30 hours per run. In theory any run which timed out would be able to complete eventually, but the practicalities of cluster usage meant the unknown time required for completion could not be allocated. It was possible to run the 40 attempted runs of each scenario in parallel using the University of York's HPC Viking cluster [84]. As not all of the 40 runs completed within the cluster's time limit in all scenarios, some scenarios were analysed using the lower numbers of replicates which had managed to finish and write out to data files.

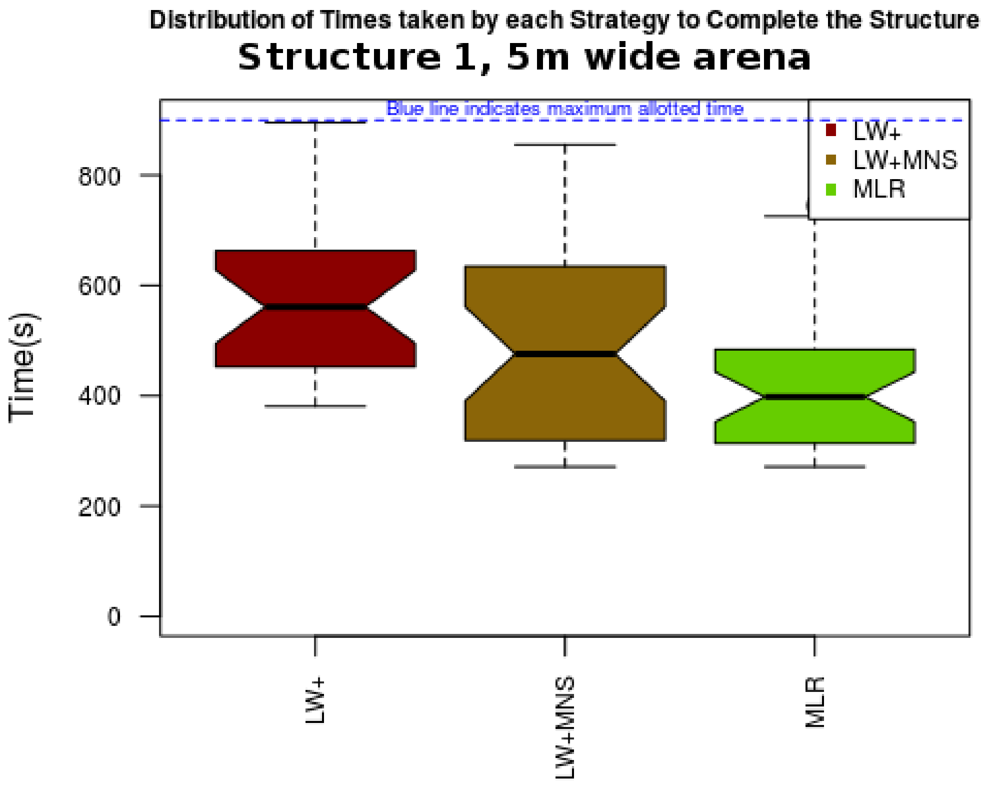


**Fig. 6.13:** Structure 1, run in a corridor of 5 metres width, a null hypothesis that the different strategies would have the same performance was used. Amongst those simulations which did not time out LW+MNS and MLR have statistically significantly superior performance to LW+, p-values of 0.062 and 5.6e-3 Compared to the results of the Structure 1 simulation in the 3m wide space all strategies have seen an improvement in performance, likely due to the unrecruited robots during later stages of assembly being able to more easily pass around the structure to reach whichever parts of it are still recruiting. The LW+, LW+MNS and MLR strategies had, respectively, time-out rates of 29%, 13% and 3%.

## 6.4 Results

Completion times for the runs were recorded, as were the times at which finish line crossings and dockings involved in the self-assembly procedure occured. Some time-outs occurred in most of the scenarios, where time-outs did occur the organisms had usually recruited most of the desired robots with only one or two remaining to be recruited, however attempts to recruit these final modules had typically been ongoing for some time, suggesting that time-outs usually occurred because the reduced number of free robots still wandering in the arena were in positions such that they rarely crossed within range of those ports which were still recruiting. R [25] was used to analyse the completion time distributions among only those runs which did not time-out, with Mann-Whitney tests used to find p-values and Vargha-Delaney A-tests used for effect size in cases where p-values were sufficiently small to imply a statistical difference between strategies. In some scenarios, the timed-out runs meant that some distributions had smaller sample sizes, however the Mann-Whitney and Vargha-Delaney tests are both immune to such effects and resulting p-values and effect sizes have been adjusted by R's functions to take account of reduced statistical power. The p-value and A-tests results across all experimental scenarios are summarised in Tables (6.2) and (6.3) while several scenarios of particular interest are discussed with the graphs in Figs.(6.13), (6.14), (6.15), (6.16) and (6.17) .

| Scenario | LW+ vs LW+MNS | LW+ vs MLR | LW+MNS vs MLR |
|---|---:|---:|---:|
| S1 R20 W3 | 5.3e-5 | 5.5e-6 | 0.56 |
| S1 R20 W5 | 0.062 | 5.6e-3 | 0.27 |
| S2 R20 W3 | 4.2e-12 | 1.2e-13 | 5.6e-3 |
| S2 R20 W5 | 8.2e-9 | 2.4e-11 | 6.4e-3 |
| S3 R30 W10 | 3.6e-10 | 2.0e-9 | 0.15 |
| S4 R20 W3 | 0.056 | 0.041 | 0.43 |
| S4 R20 W5 | 3.4e-3 | 4.0e-3 | 0.89 |
| S4 R30 W10 | 5.4e-3 | 0.26 | 0.19 |
| S4 R20 W5 Seed Changed | Inconclusive | Inconclusive | Inconclusive |
| S5 R20 W3 | 1.2e-5 | 7.5e-6 | 0.21 |
| S5 R20 W5 | 2.3e-3 | 1.3e-3 | 0.99 |
| S5 R20 W5 L20 | 1.7e-12 | 1.0e-13 | 0.95 |
| S5 R20 W5 Seed Changed | 6.4e-3 | 0.013 | 0.90 |

**Table 6.2:** P values using Mann-Whitney tests for the null hypothesis that the strategies listed as versus one another give the same performance. Inconclusive indicates that too few of the runs for a scenario succeeded to allow useful data for strategy comparison to be gathered. S indicates the structure used, R the number of robots present in the environment and W the corridor width. The L20 scenario used a 20 metre long space, all others used a 10 metre long space. Results below the 5% threshold are marked in green.

A point of interest to note from these results is the existence of time-outs, especially those that occur in the dynamic self-assembly strategies despite these typically being quicker. It is

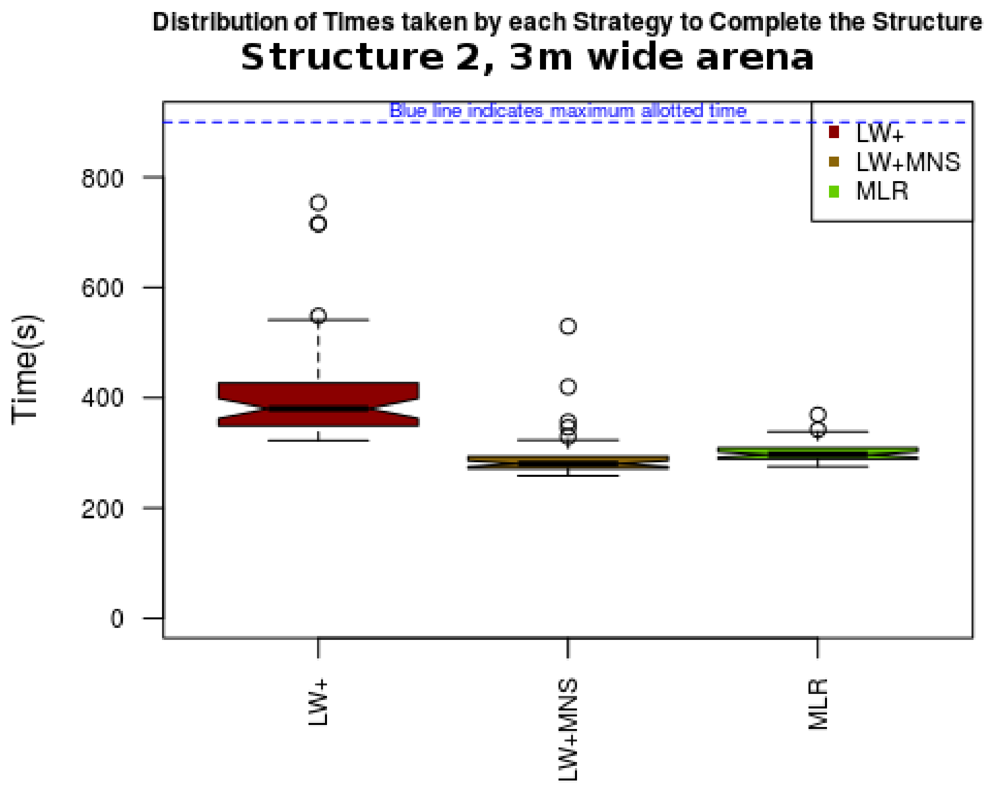


**Fig. 6.14:** Structure 2, run in a corridor of 3 metres width. No strategies had any time-outs when completing this small 5 robot structure. For this structure the performance reduces as the corridor becomes wider, assumed to be due to a wider initial spread of robots increasing the time typically necessary for them to wander into recruitment cones. As the structure is small the effects seen for Structure 1 where recruitment was accelerated in a wider corridor due to easier opportunities for wandering robots to pass around the structure are not seen, this smaller structure means that even in the narrower space scenarios there is little difficulty for other robots to pass it. p-values of 4.2e-12 and 1.2e-13 for LW+MNS and MLR, respectively, against LW+, show that the “in motion" strategies stay superior for smaller structures.

considered that due to the robots which cross the finish line before they have completed their morphology needing to wander some distance back down towards the starting line that this turning and wandering may create a second small population of results taking longer than those which form the structure during the first trip along the “corridor" space.

As indicated from these results where a low p-value shows a strong statistical difference between strategy performance the dynamic self-assembly strategies produce A-test measures indicating their superiority, in many of these scenarios there is a particularly strong effect size showing that the dynamic self-assembly methods perform better in over 90% of randomised runs. The LW+MNS and MLR scenarios rarely see a statistically significant performance difference, where they do the A-test, indicates that MLR is the inferior of the two.

The null hypotheses $H_0$6.1, $H_0$6.2 and $H_0$6.3 can now be revisited.

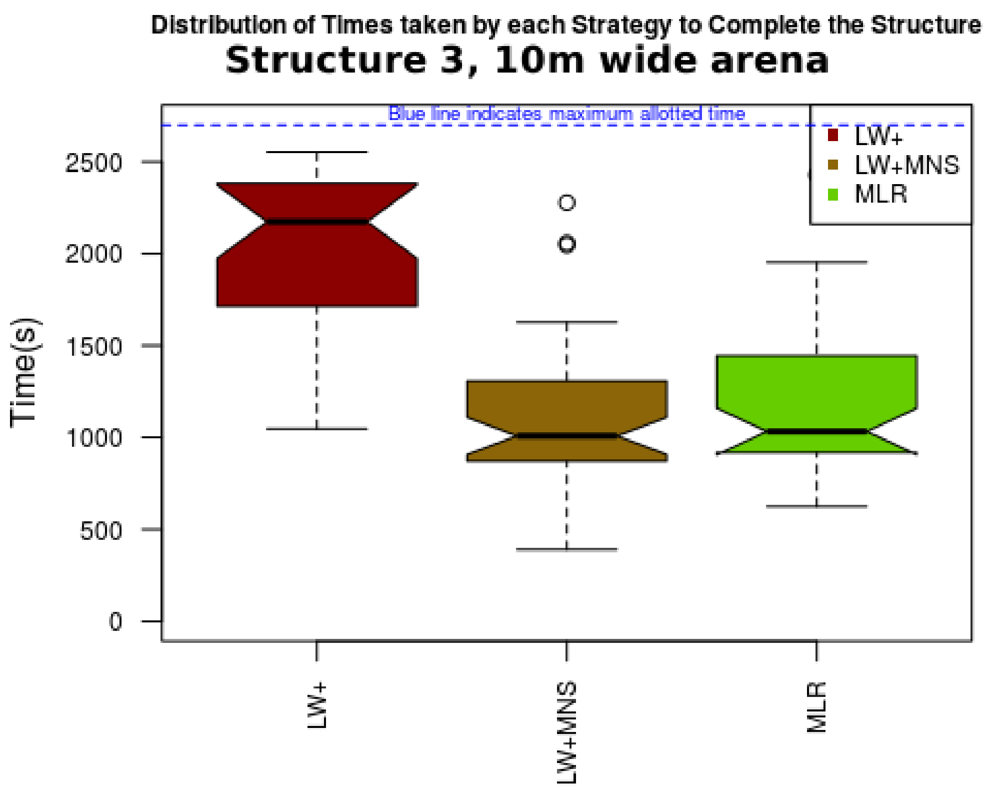


**Fig. 6.15:** Structure 3, run in a corridor of 10 metres width and using a population of 30 robots. Earlier runs performed with all strategies for structure 3 in a 3 or 5 metre corridor mostly timed out, hence the cut-off time was increased to 45 minutes and the scenario re-run with a wider corridor and increased robot population. With these adjustments to the scenario, to ensure that a high proportion of simulations managed to complete within the time limit, a strong advantage can once again be seen for the LW+MNS and MLR strategies over LW+. Consideration across S1, S2 and S3 structures shows that the superiority of dynamic self-assembly strategies appears to be maintained regardless of structure size. p-values of 3.6e-10 and 2.0e-9 for LW+MNS and MLR, respectively, against LW+ show that "in motion" strategies give better performance in large structures and with higher densities of robots in the arena.

- Significantly low p-values (<0.05) from the comparison, for most scenarios, of the LW+ distribution to the LW+MNS or MLR distributions allow $H_0$6.1 to be rejected. Effect sizes for the comparison of these strategies all favour the self-assembly strategies which can assemble during motion, for many of the scenarios these effect sizes are extremely strong indeed.

- $H_0$6.2 has also been disproven, multi-layered recruitment has operated successfully in these experiments and has done so requiring only those capabilities necessary for simpler types of self-assembly and for Mergeable Nervous Systems-style multi-robot co-ordinated motion.

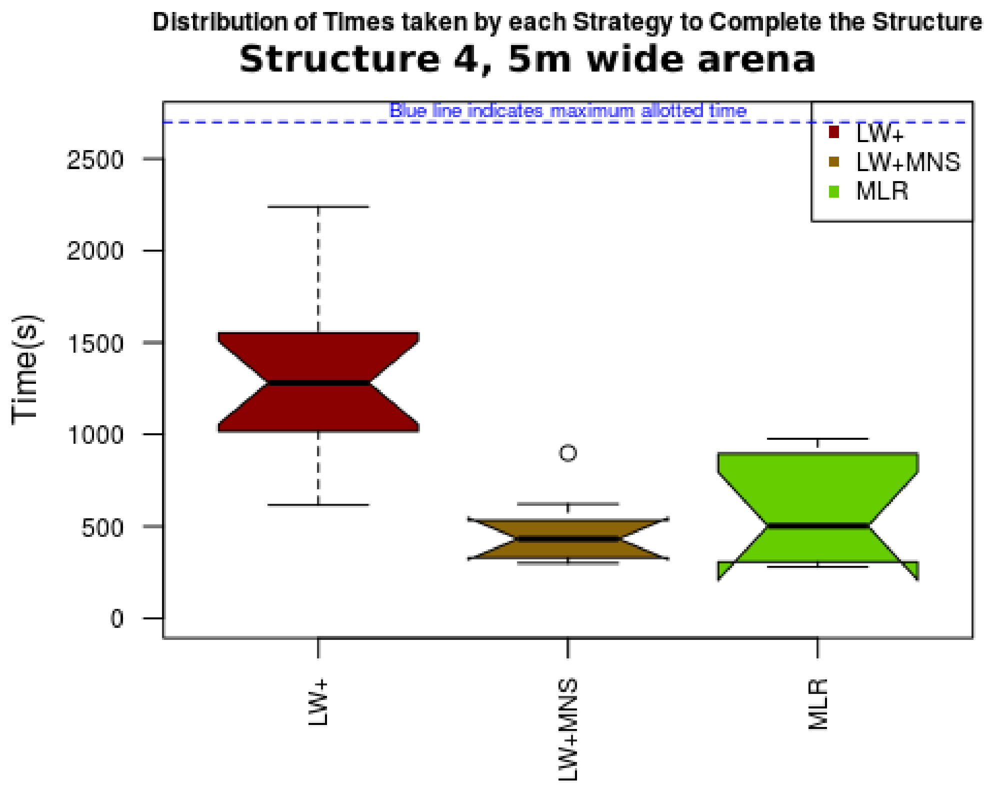


**Fig. 6.16:** Structure 4 had difficulties completing within 15 minutes in the "narrow" corridor. It was re-run using a 10 metre corridor, 30 robot population and 45 minute cut-off. Running again with the seed repositioned elsewhere within the structure did not improve the performance for completion times for this structure. Amongst those which did complete the dynamic strategies manage better, however a large number of runs of even these strategies still experienced time-outs. Where time-outs occured the structures were within a few robots of completion but had been at this "almost finished" stage for some time, implying a difficulty for the reduced number of remaining robots to enter the recruitment cones of other robots. Although robot orientations in terms of port directions differ from those in the Structure 3 this structure can otherwise be considered a sub-set of that shape, making this failure all the more unexpected given the successes seen for Structure 3. Runs of the Structure 4 with a seed robot position at another location within a structure of the same shape gave worse performance than the standard S4 structure, to the point that these runs had too many time-outs for useful data to be collected and neither p-values nor A-tests could be reasonably calculated. At a later time, when the improved cluster managing script developed for self-repair experiments had been developed, a series of S4 self-assembly scenarios were run with video recording enabled. These found that the overlapping recruitment cones which existed at some points during the formation of the structure often had difficulty with the initial recruitment of modules, before the handshaking procedure, and therefore assembly often took longer than the self-assembly experiments had been assigned sufficient computing time for.

- In most scenarios $H_0$6.3 could not be rejected, while MLR was able to operate it did not provide a statistically significant performance difference except for the very small structure, S2.

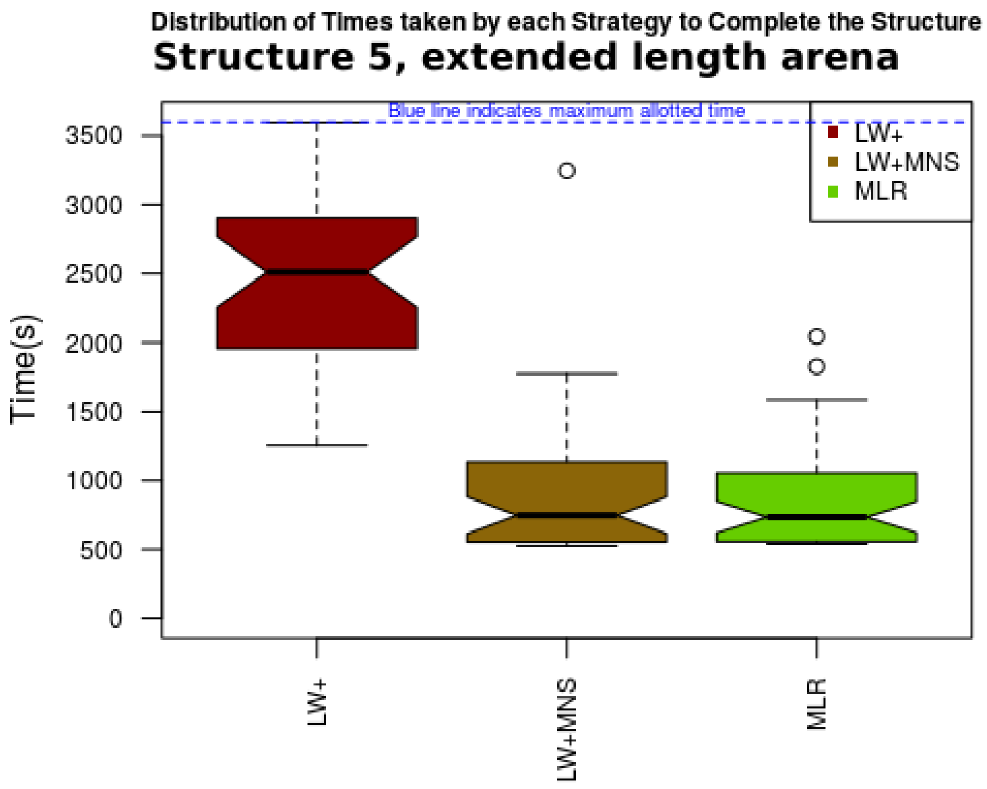


**Fig. 6.17:** Structure 5 had good performance in the 3m and 5m wide 10m long arenas for both of the dynamic self-assembly strategies, their superiority over the LW+ strategy being more pronounced than for the Structure 1 scenarios. The extension of the corridor length to 20m in this scenario clearly gives more benefit to the strategies which can assemble as they move, with A-test measures rising to 0.96 and 0.98 for the LW+MNS and MLR strategies, respectively, over LW+. An aspect to this success may be that as the space increases in length the density of wandering robots decreases and therefore it becomes more difficult for a robot staying stationary with the LW+ strategy to have anything better than a very slow rate of recruits wandering close enough to it.

One key factor responsible for this lack of difference, hence the non-rejection of $H_0$6.3, may be the existence of a maximum possible wheel speed for robots. A robot undergoing recruitment, therefore in the "Rotate to Dock" or "Approach to Dock" phase of the finite state machine, must move so as to match the motions of the recruiter's reference frame and travel towards the recruiting port. It would appear sensible to use, once speed matching to the recruiter is achieved, whatever reserves of possible motor speed are left to perform this approach. Robots therefore attempt this approach at somewhere at, or above 70% of their maximum wheel speed. Any robot recruited to a moving module will have only up to 30% of wheel speed with which to approach towards it. MLR therefore often does not in practice result in multi-module structures forming around a robot as it is still in its "Approach to Dock" state. The only way that such formation would become more common would be if robots used lower speeds for

| Scenario | LW+ vs LW+MNS | LW+ vs MLR | LW+MNS vs MLR |
|---|---|---|---|
| S1 R20 W3 | 0.81 | 0.85 | 0.45 |
| S1 R20 W5 | 0.64 | 0.76 | 0.58 |
| S2 R20 W3 | 0.94 | 0.96 | 0.27 |
| S2 R20 W5 | 0.89 | 0.93 | 0.27 |
| S3 R30 W10 | 0.94 | 0.92 | 0.41 |
| S4 R20 W3 | 0.73 | 0.74 | 0.61 |
| S4 R20 W5 | 0.97 | 0.94 | 0.48 |
| S4 R30 W10 | 0.75 | 0.63 | 0.35 |
| S4 R20 W5 Seed Changed | Inconclusive | Inconclusive | Inconclusive |
| S5 R20 W3 | 0.91 | 0.91 | 0.41 |
| S5 R20 W5 | 0.79 | 0.82 | 0.50 |
| S5 R20 W5 L20 | 0.96 | 0.98 | 0.50 |
| S5 R20 W5 Seed Changed | 0.72 | 0.70 | 0.51 |

**Table 6.3:** Vargha-Delaney A-test scores comparing the various strategies. In each column the strategy named on the left of the *vs* is compared to the strategy named on the right, values above 0.5 indicate slower performance by the strategy on the left, scores below 0.5 indicate the right hand strategy was worse. Results for scenarios in which statistical significance was below the 5% threshold are marked in green.

their approach to recruiting ports. It appears unlikely that slowing the speeds, as a percentage of the maximum wheel speed, used for the approach would hasten assembly overall and it would make timing-out, entering the “Escape Dock" state, more common. Optimising the wheel speeds used during docking to see whether it is more beneficial to overall self-assembly time to use a slow approach, with more scope for multiple layers of recruitment, or a fast approach, despite the reduction of opportunities for multi-layered recruitment that this would cause, could be an interesting topic for further research. It is perhaps therefore best considered that while MLR provides an interesting idea, and a useful stepping-stone from which further capabilities could be built, it is not in itself able to, typically, provide significantly better performance, when self-assembling while in motion, than the LW+MNS strategy.

### 6.4.1 Verifying Comparable Performance in Other Scenarios

To test $H_0$6.4 experiments were also performed where the importance of motion across a space was not considered, rather robots self-assembling using each strategy were confined within a square arena and completion occured as soon as the structure had formed without need for a finish line crossing. If for pairs of the three strategies the distributions of completion times from those strategies are similar enough to conclude that they have equal performance, then the $H_0$6.4 null hypothesis cannot be rejected. For these experiments the LW+ strategy assembled about a static seed within the arena while the LW+MNS and MLR strategies self-assembled about a seed which wandered back and forth, with the assembling organism retreating away from obstacles such as walls and those robots in the environment which were not in the process of docking to it.

- No statistically significant difference in strategy performance was found between LW+ and MLR for this scenario with a p-value of 0.32 suggesting both distributions were similar. LW+MNS statistically proved somewhat slower in this scenario with a p-value of 7.5e-5 when comparing it to LW+ and an A-test result that LW+ would be slower than LW+MNS in 24% of cases. This shows that as well as being a faster strategy in scenarios where both the time to assemble and ability to move matter the MLR strategy manages equal performance to classical static seed strategies in more classical scenarios, for the MLR strategy $H_0$6.4 cannot be rejected. However LW+MNS does not match classical Liu and Winfield-like performance in classical scenarios, and the null hypothesis can be rejected to conclude that performance here is negatively affected.

The performance difference between MLR and LW+MNS in these scenarios where motion is not a measured success criteria is an interesting contrast to the other scenarios in which MLR and LW+MNS were not significantly different. It may be the case that as the structure moved back and forth within the arena in these scenarios MLR had more opportunity to attract robots in to regions where they would be more likely to enter 5KHz recruitment cones than LW+MNS did, even if the speed considerations involved mean that separate substructures were not often formed. Fig.(6.18) shows an example of some of the distributions arising from this experimental scenario.

### 6.4.2 The Effect of Absolute Speed

The experiments conducted here have used simulated Omni-Pi-tent modules where the maximum wheel speed possible was derived from hardware tests, however it is possible to generalise the results to any maximum speed. Whilst the maximum speed of any given modular robotic platform may be different, when the same platform is considered for all strategies the relative speeds of strategies, in terms of the ratio between their completion times, should remain constant. As all speeds used by robots within the simulations are set as fractions of a maximum physically possible speed, the differences between strategies should scale inversely with the absolute speed but maintain relative proportions. However in some speed regimes this logic may break down if significant periods must be spent accelerating and decelerating. The Omni-Pi-tent modules used in simulation and hardware here had sufficiently low speeds and large enough wheel forces that the time spent accelerating between required velocities were negligible, however for modules at very high speeds, or perhaps using low thrust propulsion in an orbital space scenario, this would not be the case. The most extreme example of this may occur in very high velocity, heavily acceleration dependent, scenarios where relatively few computation steps are possible in the time taken to dock. In such scenarios corrective manoeuvring to dock to a moving target may prove difficult enough to prevent

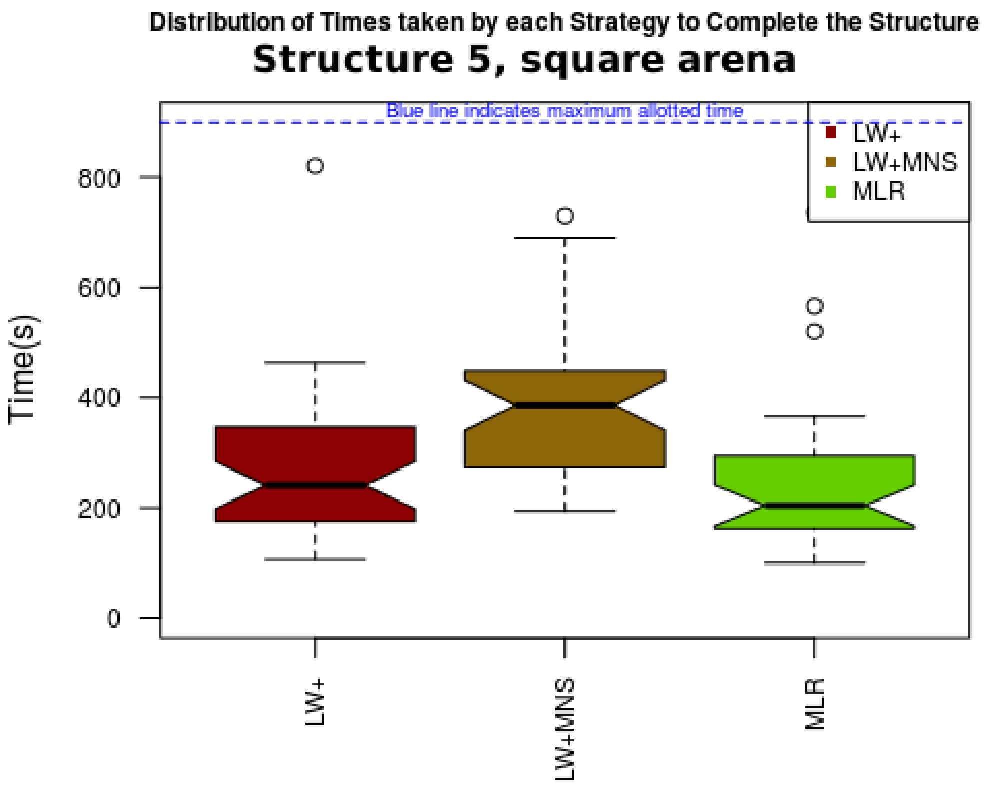


**Fig. 6.18:** Structure 5 run within a square arena where robots were in constant motion back and forth between the edges if they used LW+MNS or MLR strategies. 20 robots were present within an arena of 5m $\times$ 5m. MLR and LW+ have similar performance, a p-value of 0.32 failing to disprove the null hypothesis that performance of these strategies is equal, showing MLR is superior in scenarios where organism motion is required and no-worse than LW+ in classical scenarios. LW+MNS did not manage so well in this scenario, a p-value of 7.5e-5 disproves the null hypothesis and the A-test result shows LW+MNS's inferiority.

LW+MNS and MLR like strategies entirely, or cause them to require so many re-attempts at each docking action that LW+ like strategies may prove faster overall.

## 6.5 Hardware Verification of Principles, Preparing for Self-assembly

This section illustrates through simple experiments the feasibility of the MLR dynamic self-assembly strategy using two Omni-Pi-tent robots. An initial proof of concept was performed with only two robots, once the concept was verified, and more robots had been manufactured, further experiments were then performed with more modules involved, see Sec(6.6). These initial experiments used the two modules to demonstrate the feasibility of docking to a moving robot and of Mergeable Nervous Systems control of the two module organism.

The MLR Lua controller used in V-REP was translated to C for use on the hardware, no changes to the underlying algorithm were required. A robot acting as seed was set moving on a northward or southward trajectory, and a second module was placed to wander, at some point moving into the cones of IR light where the seed was recruiting. The seed on each run chose a random combination of which port to recruit from, and which port of the other robot to recruit for. Once connected they moved as a co-ordinated group until a manual command was given to break the structure up and another experiment started. Various aspects of these experiments are commented on below, see Fig.(6.19) for images.

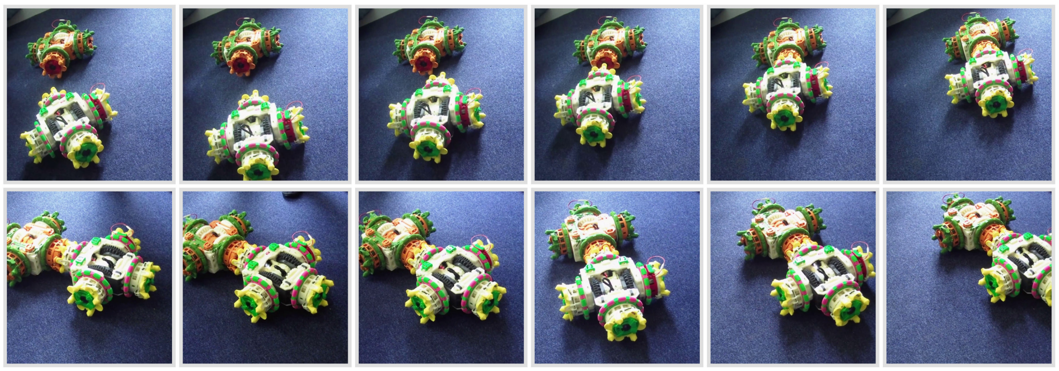


**Fig. 6.19:** A series of stills from docking and Mergeable Nervous Systems demonstrations with the hardware. The upper row shows docking on the move between two modules, seed robot in the background. The lower row shows a two-module organism rotating before driving off to the right. Video footage is included at [216] and in the supplementary material (Initial_MLR_hardware_tests.mp4).

- The infrared messaging proved less reliable at long range than had been expected in earlier tests of 38KHz IR systems acting alone, so part of the IR handshaking protocol during docking had to be carried out over Wi-Fi instead, this step in the handshaking procedure did not call for strict line-of-sight communication so did not lose relevance when transmitted over Wi-Fi between specific modules. All IR messaging which required line-of-sight continued to use the 38KHz IR hardware.
- The docked group moved as expected under Mergeable Nervous Systems control by the master using data gathered from both its own and the slave's sensors. In the video, the group can be seen retreating from obstacles positioned around it. There was some lag in the group's motions, as expected given the need to transfer data over the IR links. Situations arose, when prototyping the controllers and working around the limits of the IR communications system, where one module would attempt to move while the other tried to remain stationary. Simulations with a deliberately slowed controller, running on only a fraction of simulation steps, have shown that, while jerky, group behaviours and docking still appear to function correctly in the presence of IR communications lag. This may affect Multi-layered recruitment to a greater extent than the more classical strategies as it makes more use of separate structures under Mergeable Nervous Systems

control. In later experiments some of these port to port communications were handled by module to module Wi-Fi messages, with correct addressing within these messages the data flow still followed the Mergeable Nervous Systems routing through the structure, but each port to port IR link was replaced by a faster Wi-Fi data transfer. Such a use of Wi-Fi for robot to robot communications seems similar to how [26] performed MNS on their hardware. The reduced lag of higher bandwidth communication here ensures less jerky motion from the group.

- Real world non-uniformity in magnetic fields meant that, in the distance between two connected robots, the field often turned by up to $10°$, so robots coming together would often be $10°$ off compass alignment. Random noise of $\pm 10°$, varying every 0.6 seconds, was added to the compass reading during final approach for docking ensuring that at some point the two modules would align their hooks to within the angle accuracy needed to dock successfully. This magnetic field non-uniformity did not affect robots once docked within an organism, as the MLR implementation of the Mergeable Nervous Systems protocols was carefully designed to avoid explicit reference to local compass readings and handle all data in the reference frame of the master robot with inter-robot angles calculated by knowledge of port to port links and not by compass readings.

- Breaking up of the structure was tested by triggering a breakup mode in the master via pressing a switch. In self-assembly scenarios breakup would only be expected for an MLR substructure in which the local master of the structure had timed out of a docking attempt to another group, however testing this directly was not possible with only two robots. In self-repair scenarios this method could be used to fully disassemble organisms as one method for replacing failed modules.

- The docking hooks gave a somewhat more flexible connection than had previously been expected between modules. This did not, however, prevent the docking actions or the co-ordinated group motion. Code modification on the ATMEGA328P microcontrollers responsible for controlling the hooks ensured that hooks would re-lock immediately when experiencing any twisting forces and hence maintain closer alignment throughout the period for which a module pair are docked. Modification of the docking hook geometry to give tighter tolerances when connected was performed before further multi-module experiments were run.

## 6.6 Hardware Self-assembly Demonstrations

The initial verification of self-assembly using hardware was conducted both with the second prototype and first production module before hook, wheel and hinge improvements were made, see Sec.(5.1), (5.4.1) and (5.16) for details. This accounts for many of the limitations discussed above, especially in terms of driving performance and the reliability of docked connections. Later tests were performed with hardware which had been upgraded to the finalised design and with a larger swarm of 4 robots available.

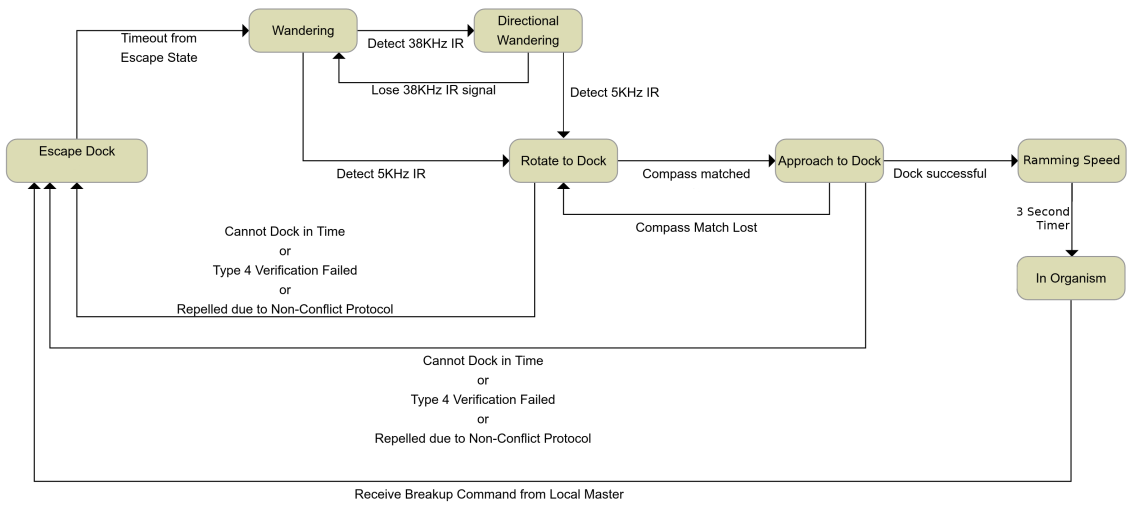
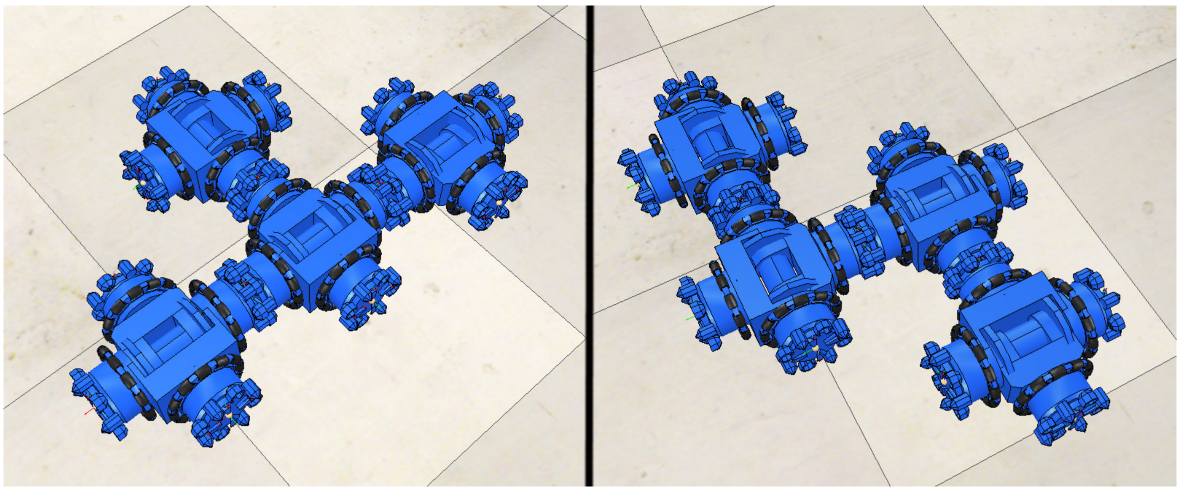

Wandering
Directional Wandering
Timeout from Escape State
Detect 38KHz IR
Lose 38KHz IR signal
Detect 5KHz IR
Escape Dock
Rotate to Dock
Approach to Dock
Ramming Speed
Detect 5KHz IR
Compass matched
Dock successful
Compass Match Lost
3 Second Timer
In Organism
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Receive Breakup Command from Local Master

**Fig. 6.20:** The modified finite state machine used on the hardware. Notice the "Ramming Speed" state added to improve docking reliability. One of the key reasons for this extra state is the 0.3 second time required by the hardware to actuate the docking hooks, which has been represented by an instantaneous action in simulation.



T structure
{{1,1,**1**,2},{1,3,**1**,3},{1,4,**1**,4}}
S structure
{{1,2,2,2},{1,3,4,3},{3,3,1,4}}

**Fig. 6.21:** The T (left) and S (right) structures formed using the hardware. For a wider range of test examples the T shape was also redefined in some of the runs such that Port 2, 3 or 4 was used by each of the docking robots instead of Port 1, this is marked in bold.

Each robot ran a C implementation of the Multi-Layered Recruitment controller, derived from translating the original Lua controller used in the simulations. Due to the findings from earlier work with the hardware, see Sec.(6.5), it was known that the IR 38KHz communications could prove unreliable, and features were added to this controller which broadcasts copies of some of the IR message types over Wi-Fi. The C implementation used also added an extra state to the Finite State Machine, this state, referred to as “Ramming Speed", was placed between “Approach to Dock" and “In Organism", see Fig.(6.20). This state was entered as soon as both docking switches on the appropriate port of a recruited robot were pressed indicating reaching the docking position, and lasts for 3 seconds during which the recruited robot drives at full speed against the port to maintain good contact while its hooks lock, after which the robot transitions to the “In Organism" state and acts equivalently to in simulation. This extra state compensated for the “reality gap" effects of a short, but non-zero, docking actuation time in the real world as compared to instant, single-timestep, docking in the simulations. Showing how even a small difference, the 0.3 second actuation time is much shorter than the time spent rotating and aligning to dock, between simulation and hardware can have major effects.

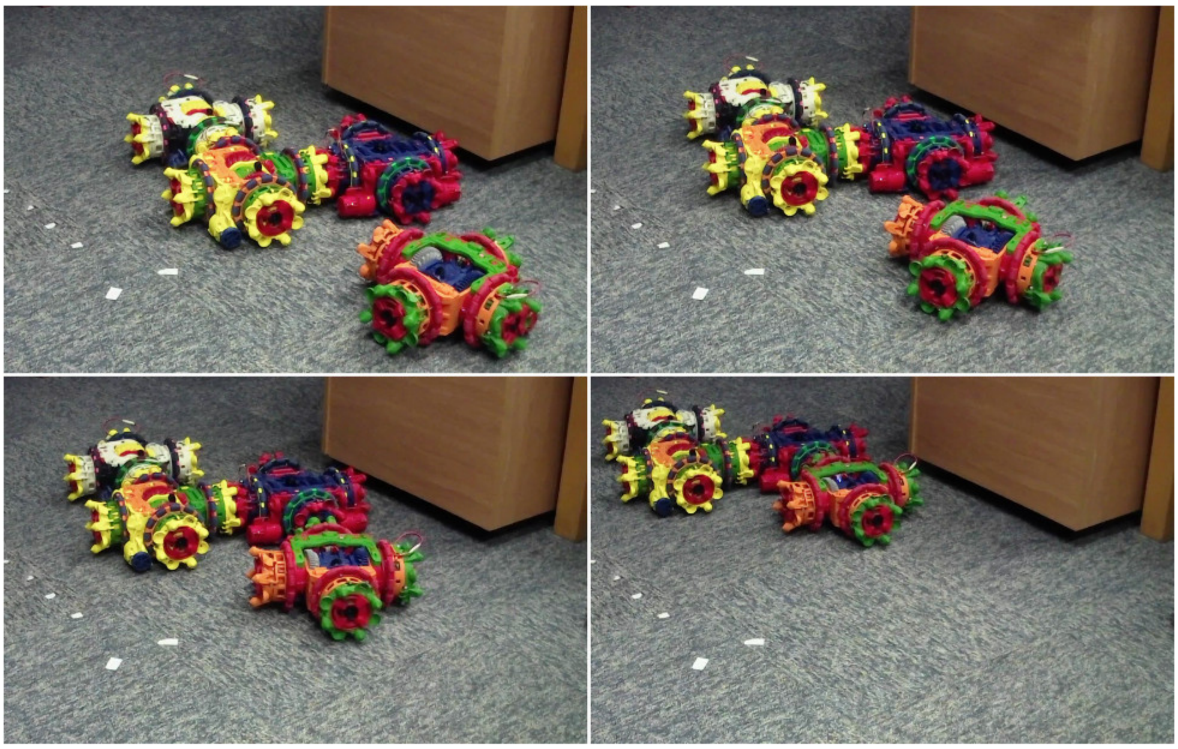


**Fig. 6.22:** A robot is placed at a random orientation within IR communication range of a docking port and matches compass alignment and position to dock.

A robot within the swarm was manually instructed to become seed for one of two different structures, a T shape or an S shape, and began moving along a northward direction calculated from its compass readings [2]. Fig.(6.21) shows the shapes and Structure Recruitment Lists for these shapes. Due to the limited number of robots available it was not feasible for random wandering to bring robots within recruitment range regularly, hence other modules were

[2]When the robotic group reached the edge of the laboratory space and began obstacle avoiding a manual command was supplied to change motion to an eastward or southward direction after the avoidance was complete.

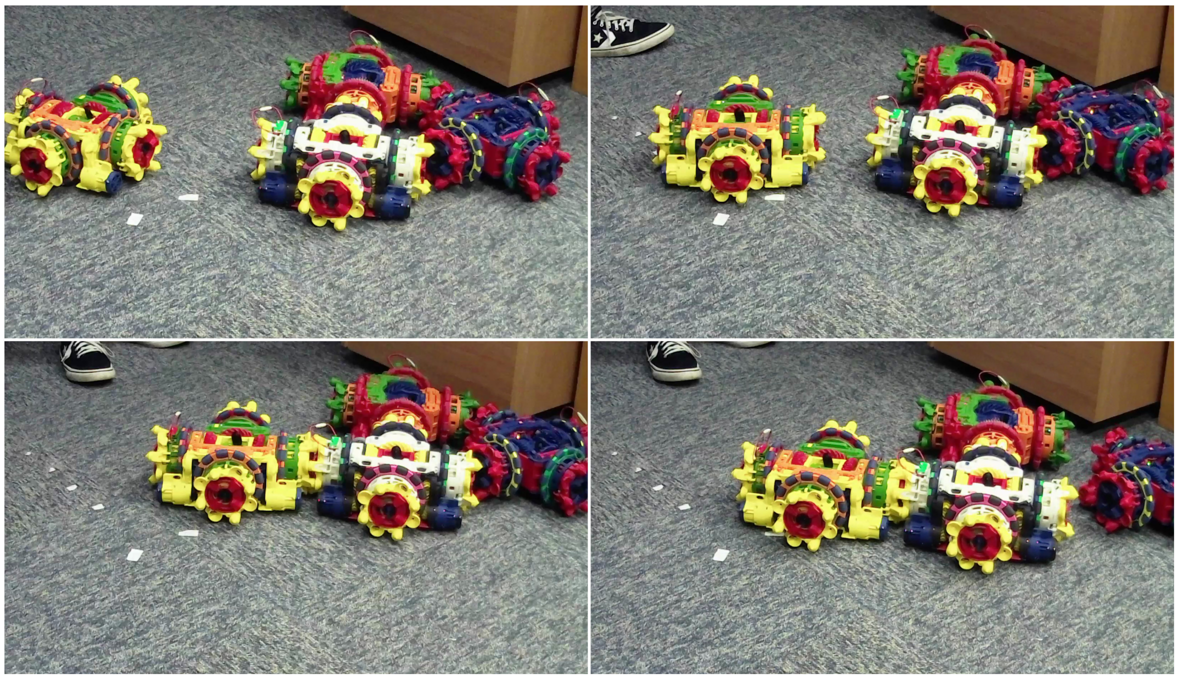


**Fig. 6.23:** The yellow robot (right) navigates to dock with port 3 of the white robot, meanwhile a blue robot (left) in the “wandering" state avoids the forming structure.

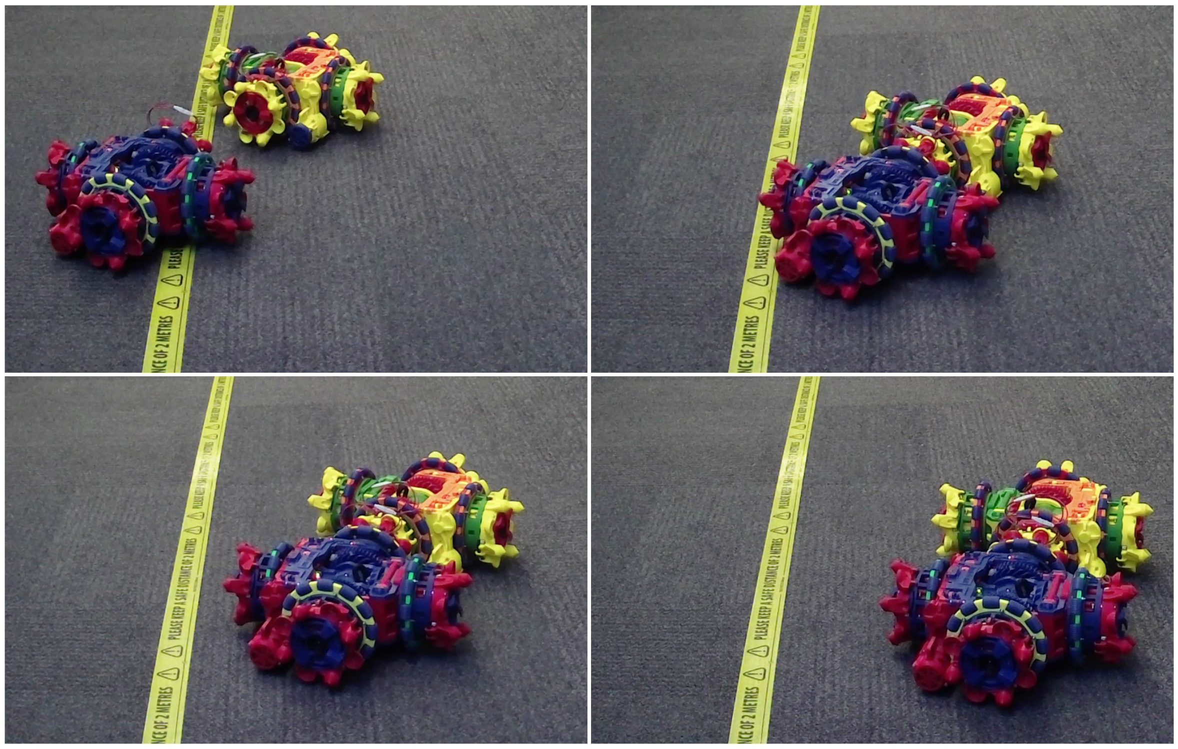


**Fig. 6.24:** In this demonstration a robot (upper) approaches the seed robot to form one arm of the T structure, note how, in this example, the structure forms in a different order than in Fig.(6.23), with the robot which takes a Temporary ID of 3 arriving first in this example.

placed in regions where they could receive its recruitment communications over IR and allowed to dock autonomously from there. Fig.(6.22) shows such a robot placed near the

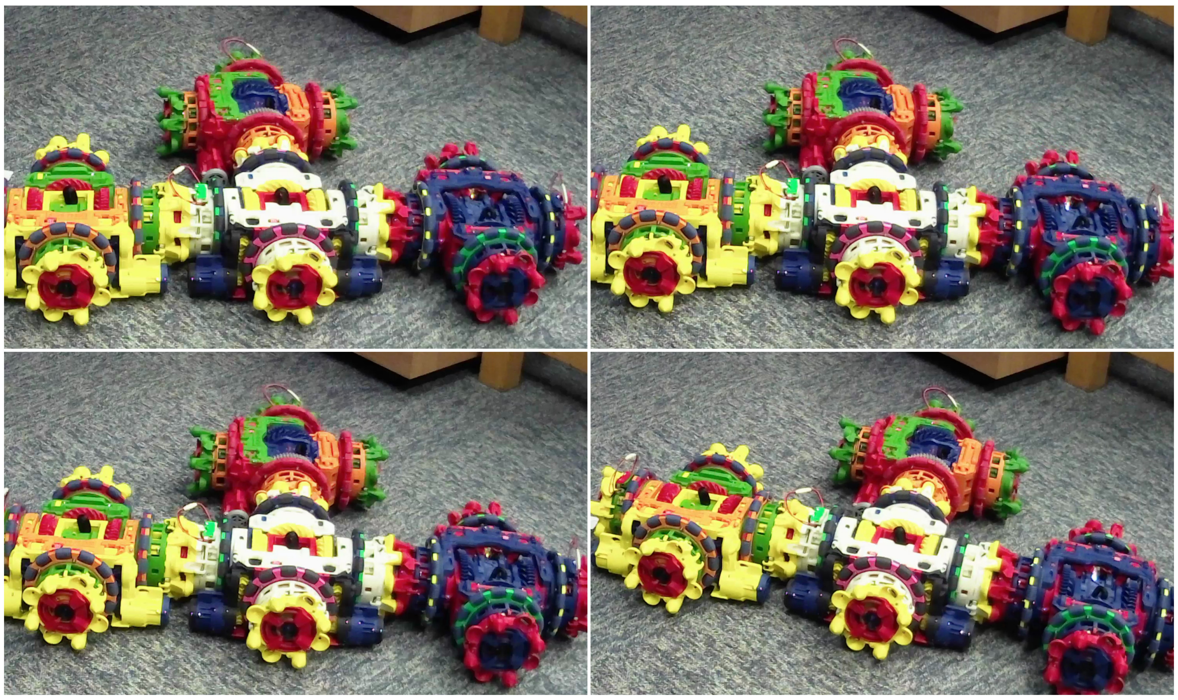


**Fig. 6.25:** The T structure rotates under MNS control.

moving seed. Robots were allowed to dock to form the structures and navigated themselves for both positioning and compass alignment, some of the placed robots were put at 90°or 180°away from the orientations required to dock and often accomplished compass alignment to within that necessary for docking. Figs.(6.23) and (6.24) show examples of these docking operations. Further robots were placed in recruiting regions, and in some cases randomly wandered into them while the experimenter's back was turned, and docked until the structure was formed. The multi-robot structures were then allowed to move around and avoid obstacles to provide proof of the Mergeable Nervous Systems capabilities, see Fig.(6.25). Videos of some of these recruitment events and of coordinated group motion are provided at[217] and in the supplementary material (MLR_hardware_demo.mp4).

A notable problem encountered was establishing whether a robot was docked on any given port. Whilst a simple check to perform in simulation (`sim.getLinkDummy(MyPortXDummy)`)the physical hardware's tolerances are such that when two robots are docked there is sufficient play within the mechanism to allow for the contact switches to break contact even whilst a dock exists. When determining whether a port is docked there are two considerations of importance, the first is simple, knowing whether a robot's own docking hooks on a particular port are locked or unlocked, in the latter state there definitely cannot be a dock present on that port. However if it is known that a port's hooks are locked a robot still requires further information to know whether the other robot involved in the dock is actually present. It had been intended to use the states of the contact switches in the spike accepting pits to determine whether the other robot was attached, however the relative looseness of even the improved docking hooks meant this could not be permanently guaranteed. Methods based on low pass filtering the docking switch states to remove brief fluctuations struggled to tell the repeated impacts on each switch involved in pre-docking manoeuvres from a successful dock, and also incorrectly assumed a dock no longer existed when robots were driving in such a direction as to relieve the pressure on the switches for significant time periods. A Wi-Fi aided method was therefore developed by which a robot which was entering the “Ramming Speed" state would signal this over Wi-Fi to the robot it was docking with, their hardware IDs for Wi-Fi addressing purposes having been shared through the non-conflict docking handshaking procedures, and let it know that a dock now existed between them. Any robot undocking would send another form of Wi-Fi message to inform that it was doing so. This Wi-Fi aided method still exhibits some of the fault tolerant properties crucial for later work on self-repair, a dock can be judged no longer to exist as soon as a functioning robot releases its own hooks therefore Wi-Fi communication to or from a failed robot is not required. But reliance on communications to signal the making and breaking of docked connections allowed for some scenarios where a robot wrongly believed itself to be docked, and was believed by its immediate master to have docked, despite having become disconnected in the 0.3 seconds between beginning actuating the docking hooks and having them reach the locked position due to the master robot moving at that instant in ways which “Ramming Speed" was not able to correct for. Almost all activities with modular robots would appear to require reliable ways for a robot to check at any given moment whether any given one of its ports is docked, so it may not be possible to develop controllers which can achieve desirable levels of reliability without accurate “dock presence sensing". Future modular robotic designs may perhaps benefit from being fitted with non-contact sensors to verify whether they are docked via a short range proximity sensor or potentially some form of hall effect or metal detecting method with a small magnetised or metal target on each port.

Difficulties were also encountered in some runs due to the replacement of line-of-sight IR communications in some roles with universally available Wi-Fi messages. This made the non-conflict docking procedures more difficult, as only the initial recruitment messages were now sent over the line-of-sight IR system. With Type 4 short range verification messaging not able to work reliably this too had to be replaced with Wi-Fi, therefore in a small number of cases robots were able to dock to ports other than the one which they thought they were approaching, unable to rely on the readings of the contact switches robots were not able to easily detect such incorrect dockings. Further Wi-Fi handshaking was implemented to partially mitigate this, but some incorrect connections could not be prevented as the Wi-Fi signalling is not position dependent in the way that the IR was designed to be. These problems show the benefit which line-of-sight communication can bring if it is reliable, particularly that approximate positional information can be inferred and that receipt of certain message types guarantees relative positions. The problems also highlight the perils involved in trying to work around line-of-sight to use global communications instead. It is worth repeating that the difficulties caused by a lack of line-of-sight communication here show that line-of-sight communication is not inferior to global communication in all modular robotics applications, despite the popularity of broadcast methods [93] [110]. Future modular robotic designs would benefit from making development of multi-ranged, high data-rate high-reliability, line-of-sight communication systems a major priority.

The Mergeable Nervous Systems [26] inspired co-ordinated group control methods worked extremely well, giving reliable control of the multi-robot organism. Tests were performed by placing an IR reflecting obstacle in to various positions around a completed, or partially completed, organism and watching its reactions. As intended obstacles placed beside any docking port, whether on the seed robot, on a robot connected to the seed, or on a robot two layers away from the seed, were detected and the structure retreated away from them according to the clock direction in which they were observed. Fig.(6.26) shows an example. Ports facing towards other parts of the structure were, as intended, not sensitive to obstacles. An example of such a pair of ports are Port 4 on the robot with a Temporary ID of 4 and Port 2 on the robot with a Temporary ID of 2, in the T structure, see Fig.(6.27). This ignorance of readings from such ports is particularly useful with the real hardware as where docking ports face towards one-another very strong false positive detections can occur where one port perceives the other's emitted 5KHz signal as if it were a reflected return of its own signal, as well as risks of the robot detecting other parts of itself from IR reflections. Rotation of the robotic group was demonstrated by placing metal objects close to the master robot's compass and watching the group rotate as the master realigned to the altered magnetic field, then return to the initial orientation when the metal was removed.

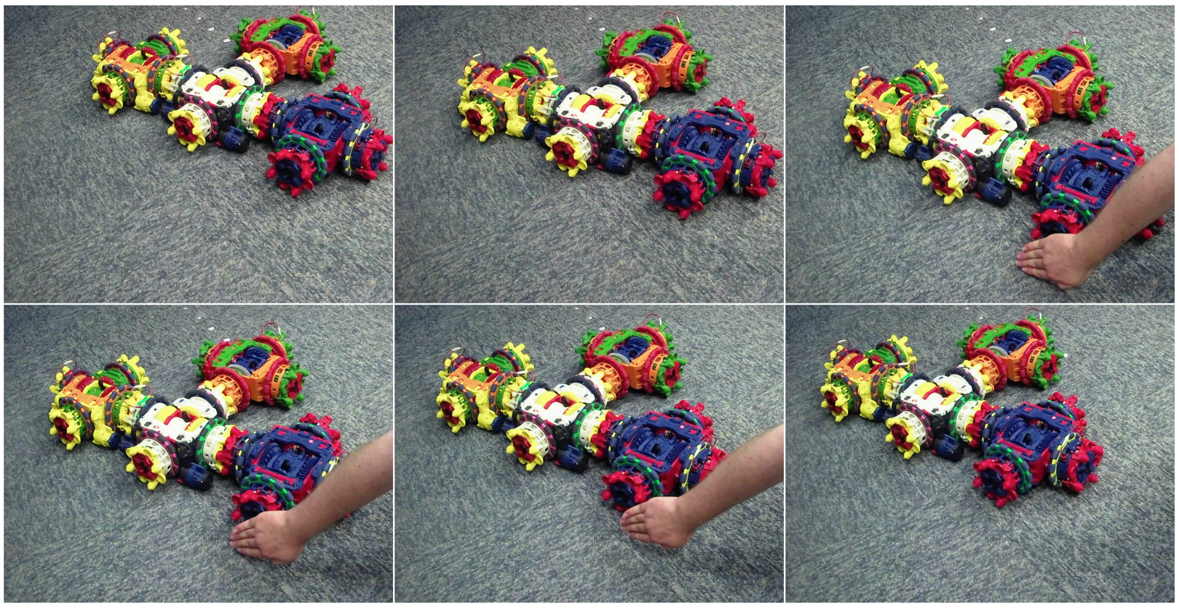


**Fig. 6.26:** An infrared reflecting obstacle is placed before Port 4 on Robot 2 of the T structure, the obstacle is seen, information about it is packed in to a 12 byte array, this is forwarded to the master robot, which then moves the structure to retreat from it.

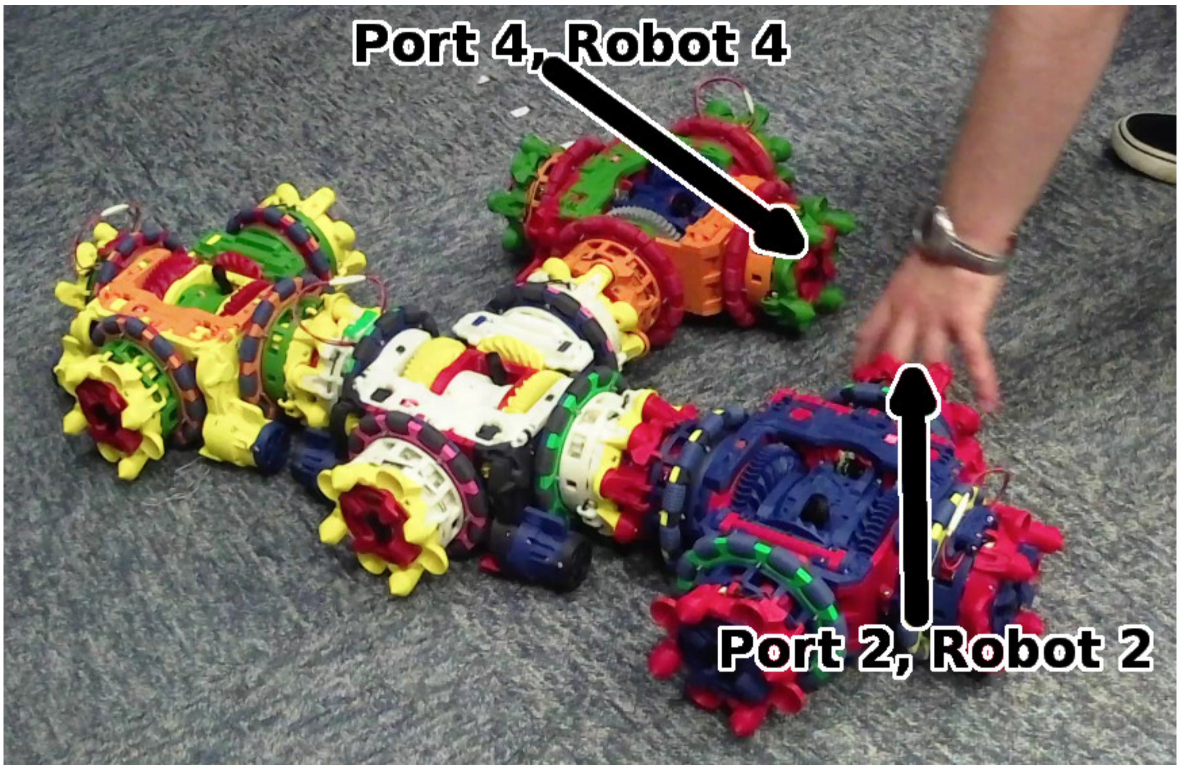


**Fig. 6.27:** The multi-robot organism ignores proximity detections on ports which face inward upon one another.

Among the runs performed out of 26 dockings attempted between wandering modules and moving recruiters 20 of these actions succeeded, those which failed were due to: 3 occurrences of the approaching robot timing out of the docking attempt after taking too long, one occurrence of a serious compass error and problems on the “stop mastering compass" line occurring on the wandering robot during the final stages of docking and requiring several seconds for the I2C bus to reset itself, see Sec.(5.11), one occurrence of a robot becoming separated from the docking port during the “Ramming Speed" state and both it and its immediate master being unable to detect this loss of contact, and one incident in which a robot docked to the wrong port and then behaved incorrectly once under MNS control.

Tests were also performed to demonstrate MLR’s use of co-ordinated docking between robotic groups. For the S shaped structure a robot entering the position equivalent to a Temporary ID of 3 was allowed to complete the handshaking procedure and then temporarily manually restrained. An extra robot was placed close by this robot’s Port 3 and allowed to dock to it. The robot with a Temporary ID of 3 was then released, to act as master of a two robot group and attempt docking to the seed robot. Fig.(6.28) shows an example. Across three attempts one of these succeeded in docking the co-ordinated group to the seed robot without human assistance, one attempt required human handling to improve the compass alignment in the final stages of docking and one timed out. Overall it was difficult to create the circumstances for this event and the robot with a Temporary ID of 3 usually completed a docking to the seed

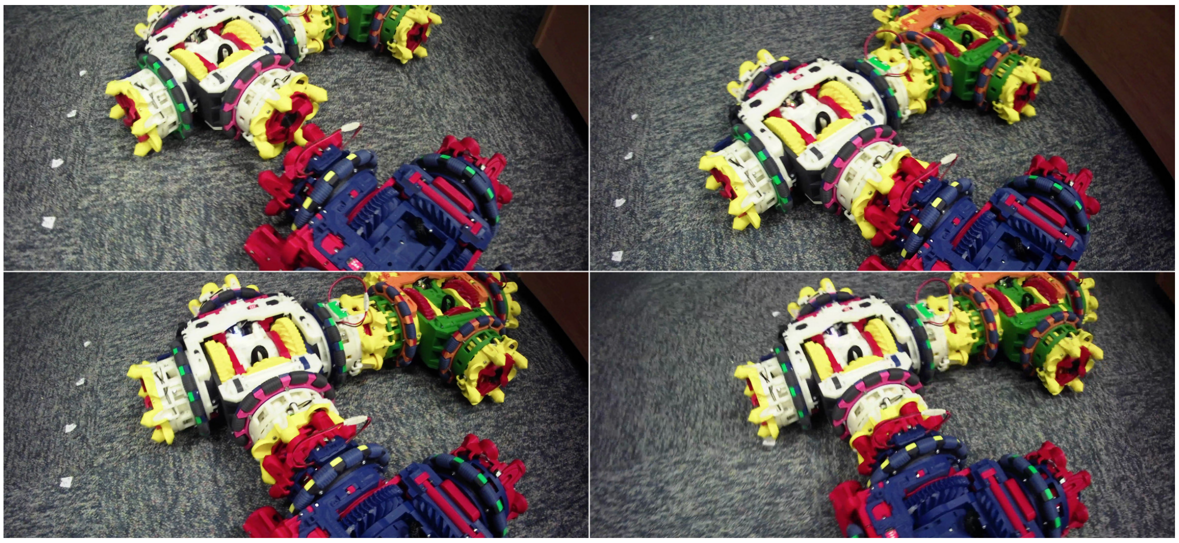


**Fig. 6.28:** The seed robot (blue) recruits for a robot to take a Temporary ID of 3 (the white robot), the robots were positioned and held so that a further robot (green) could dock to this recruit before the recruit docked with the seed. This image sequence shows the use of co-ordinated MNS motion to enable a docking between the two robot group and the seed.

before its subslave could be introduced to the area. The close proximity of a second robot to the robot with the Temporary ID of 3 appears to have typically caused enough magnetic anomalies to affect the compass calibration so as to make it difficult for the robot to get within a small enough alignment error for the randomised rotation motions to prove helpful. This proves the principle of MLR to be feasible with hardware, but also highlights how the MLR strategy's key novel feature proves particularly vulnerable to the increased difficulty for docking actions which the reality gap creates. It should ofcourse be noted that even in most of the simulations robots did not typically form such substructures before docking, given the speed considerations involved, hence these difficulties with hardware do not make the MLR self-assembly strategy pointless, having robots begin further layers of IR recruitment before docking to their immediate masters still has uses.

## 6.7 Summary

Recruitment strategies for self-assembly with modular robots during motion have been proposed and developed. It has been explained how the unique features of the Omni-Pi-tent platform make such strategies feasible and discussion of implementation details has been presented. These new strategies were compared against a “classical" self-assembly to a static seed, inspired by Liu and Winfield's work [27]. It was found that, in all scenarios for which conclusive data could be gathered, one or both dynamic self-assembly strategies had statistically significantly better performance. Among the scenarios Vargha-Delaney A-tests always favoured the dynamic self-assembly strategies over the classically inspired static method. The performance difference between the two dynamic strategies varied by scenario and was often not statistically significant, among the scenarios with statistically significant differences, and among all the scenarios in general, the MLR strategy proved somewhat inferior. Tests were also performed to test for scenarios where the seed simply moved around within an arena and the ability to cover distance was not considered as a metric of success, the formation times using MLR were similar to when a classical static seed strategy was used, although LW+MNS did not match the classical strategies here. This suggests that due to its close performance to LW+MNS in “corridor" scenarios and comparable performance to LW+ in classical scenarios the MLR strategy is most worthy of future development towards a Dynamic Self-repair algorithm. Hardware tests verified the real-world validity of the principles of docking and Mergeable Nervous Systems group control via which the MLR strategy operates. No prior self-assembly work has managed to operate at all with co-ordinated group motion occurring while assembly progresses. Docking between moving modules and groups provides an extremely useful capability to modular robots and aids their ability to self-assemble.

With self-assembly methods developed it is possible to form structures which can engage in self-repair. With multi-module capabilities proven one can be confident in the ability of these assembled structures to move, and in the ability of modules to handle the manoeuvres required for self-repair operations. In Chapter 7 the self-assembly controllers are modified to include self-repair functionality which is then tested.

# Chapter 7

# Self-repair

This chapter outlines the self-repair methods developed for Dynamic Self-repair and examines the features included within them before comparing the new Dynamic Self-repair strategy described in this thesis to older strategies implemented on the Omni-Pi-tent platform. Self-repair, the removal of failed module(s) and their replacement with other working modules, consists of the detection of failed module(s), their removal and their replacement. Very few previous studies [57] [55] [39] [56] [47] [48] [166] have considered modular robotic self-repair at all, fewer still have considered it in the full practical context [51] [26] with physical, and simulations closely representing physical, robots and considerations of removal of failed units as well as the attachment of replacements. This chapter builds on the work of [51] to implement self-repair for Omni-Pi-tent, introducing new features both those inspired by [26] and those required to enable self-repair to operate during continuous motion. This chapter serves both to test Dynamic Self-repair, and to show that some of the findings of [51] still hold as true for Dynamic Self-repair with the new Omni-Pi-tent platform as they did for his static self-repair scenarios with SYMBRION modules.

This chapter is structured as follows: it begins by introducing the assumptions made about the forms of faults to be tackled in Sec.(7.1), and the null hypotheses to be tested, Sec.(7.2). The concept for the self-repair method is then explained, Sec.(7.3), and the underlying controller and algorithms presented, Sec.(7.4). In Sec.(7.5) the new self-repair algorithm is then contrasted to earlier methods, before Sec.(7.6) introduces the experimental scenarios used to test the self-repair strategies against one-another. Sec.(7.7) and Sec.(7.8) consider the results of these experiments. Sec.(7.9) then modifies the Dynamic self-repair algorithm to account for failures of the seed robot within a structure and dynamically switch the location of a structure's global master robot during repair. Sec.(7.10) and Sec.(7.11) analyse the results of experiments to test this Dynamic Master Switching capability. To probe the capabilities of the optimised Dynamic Self-repair strategy Sec.(7.12) considers multiple module failures. Finally Sec.(7.13) attempts a hardware replication of the Dynamic Self-repair strategy developed in simulation, and encounters reality gap complications. Sec.(7.14) summarises the chapter.

This chapter's key contributions include:

- Development of self-repair strategies for modular robots which make use of mobile groups of connected robots to both remove failed units and perform replacement. Previous work has either not used truly modular robots [26], relied only on robots individually being recruited [51] to repair the structure or has only used mobile groups of robots for re-assembly [166] but not for removal of failed modules.

- Development of self-repair strategies able to operate whilst a group of robots is in motion. Such strategies have never been previously demonstrated for modular robots.

- Implementation of these self-repair strategies for the new Omni-Pi-tent platform, this includes the development methods of processing quadruplet array Recruitment Lists data structures to assign tasks during self-repair.

- Comparison in simulation of the effect of the two novel elements of the strategies on times for task completion.

- Exploring the further work required to use these strategies on real Omni-Pi-tent robotic hardware.

## 7.1 Assumptions

Faults can take many forms, and there are large bodies of work on how faults within robots can be detected and diagnosed, for example [151] [51] [163]. Methods exist by which partial failures can be detected, either endogenously (within the failed robot) or exogenously (from the sensor data of others around it). For the purposes of this work only total failures will be considered, a module is considered as either sufficiently fault free to perform its duties correctly, or as suffering from faults severe enough to make it useless. Modules with faults therefore are treated as if they were to suffer a complete power failure and become unable to sense, actuate or communicate. Such failures are to be detected by other modules, still in functioning condition. This assumption is plausible for real-world modular robots if an inbuilt endogenous diagnostic system were fitted which would allow modules to detect internal faults and shutdown if any occur. For more minor partial failures there may be good cause to reconfigure to change robot positions and move robots with sensor failures to actuation-only roles in a given morphology, doing vice-versa for robots with actuator faults. Such decisions on when to reconfigure could perhaps be developed from [108]. This work only focuses on the removal and replacement of fully failed modules as it is expected that reconfigurations to swap across partly damaged modules would be extensions on to the same strategies as used to replace fully failed modules.

When the removal of failed robots is performed it was initially assumed, pending further investigation with hardware, that two modules would be necessary to provide enough wheel torque to manoeuvre a failed module. It had already been found in practice that a single robot alone was not able to successfully reposition another module with uncooperative wheels. Any geometric arrangement of two, or more, operating modules around a failed module was assumed to be adequate, whether these modules are all connected as part of a multi-robot structure on a single port on the failed module or connected to several of the failed module's ports.

Looped structures, due to the inability of the present implementation of MLR derived self-assembly to form them, are not considered.

Only 2D structures are considered. Whilst Omni-Pi-tent has some 3D capabilities it was considered out of scope in this project to fully develop them and devise methods of transferring modules across 3D structures.

Similarly to earlier work [51] it was also assumed that robots will not suffer complex control system failures and “byzantine faults" which could cause them to conduct undesired active behaviours, such as broadcasting IR or Wi-Fi transmissions of erroneous data with valid checksums, broadcasts with invalid checksums would of course be ignored anyway.

The scenarios tested here will be such that there are always sufficient spare robots present to replace failed modules, and will initially consider single module failures only, however Sec.(7.12) uses multi-module failure scenarios to challenge the newly developed Dynamic Self-repair strategy and probe for limitations.

Initially only failures of robots which are not the overall master of a structure are considered, but Sec.(7.9) then considers such cases. This decision was made due to the difficulties which some of the competitor strategies to Dynamic Self-repair, presented in Sec.(7.5), would suffer during a failure of the global master, hence the capability for global master replacement was developed later.

## 7.2 Hypotheses

This chapter considers several hypotheses, these hypotheses relate to the performance of different methods of self-repair and how they compare against each other. The hypotheses both test for the effect of being able to repair whilst in motion on self-repair performance, but also act to test whether self-repair continues to out-compete what have been referred to as “naive" [51] strategies when this continuous motion ability is added. In doing so the experiments to be performed will not only help answer the question raised by the Dynamic Self Repair Hypothesis, H1.1, but also allow the reproducibility of some of Murray's conclusions from [51] to be tested with the new Omni-Pi-tent platform and these new implementations of self-repair strategies.

The null hypotheses to be tested are therefore outlined as follows:

**Null Hypothesis 7.1:** *Being able to repair while in motion has no effect on the chance of whether self-repair, and a locomotion task, will complete within a specified time limit.*

H7.1 arises from the question of reliability, and seeks to compare strategies to see which are most often successful.

**Null Hypothesis 7.2:** *Being able to repair while in motion has no effect on the time taken for locomotion tasks during which a repair is required.*

**Null Hypothesis 7.3:** *The act of repairing, independent of the time to complete other tasks, takes equally long for repairs undertaken while in motion as it does for repairs during which the structure remains static.*

**Null Hypothesis 7.4:** *There is no difference in the time taken for a group task between repair strategies requiring a significant breakup of the structure and repair through ordinary self-assembly methods, as versus repair strategies which keep multiple substructures of robots connected throughout.*

**Null Hypothesis 7.5:** *The act of repairing, independent of the time to complete other tasks, takes equally long for repairs using breakup and re-assembly type strategies as it does for repairs using self-repair strategies in which substructures are preserved.*

H7.2, H7.3, H7.4 and H7.5 form, in effect, a $2 \times 2$ grid. H7.2 and H7.4 compare, respectively, the effect on task completion times of dynamic versus static and substructure versus breakup strategies. H7.3 and H7.5 compare the effects of, respectively, dynamic versus static and substructure versus breakup, on the time taken for the repair operations themselves. From our expectation that parallelising tasks is the key advantage of dynamic strategies over static ones we would expect H7.2 to be rejected, but may find effects to reject H7.3. H7.3 and H7.5 will provide a close comparison to the work of [51] and see whether those conclusions stay true when continuous motion strategies are involved.

**Null Hypothesis 7.6:** *Relative effectiveness, in regards to task completion time, of different self-repair strategies does not vary with the size of the structure below the failed module in the multi-robot organism's hierarchy.*

**Null Hypothesis 7.7:** *The size of the structure below the failed module in the multi-robot organism's hierarchy does not alter the relative times, between different strategies, for the self-repair procedures themselves to complete.*

H7.6, H7.7 test the scalability of different self-repair strategies, with H7.6 focused on task completion and H7.7 on the repair operations themselves.

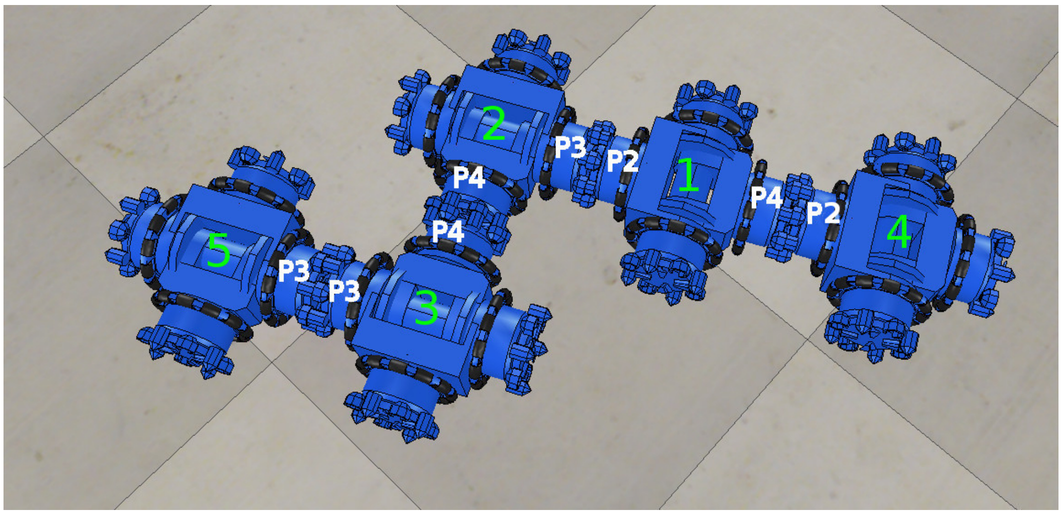
**Null Hypothesis 7.8:** *There is no benefit to be had in optimising which connected robot or group of robots attached to a failed module is responsible for which aspect of the repair procedure.*

H7.8 asks whether strategies can vary in performance depending on which robot in a structure fails, and whether a single strategy can be developed which is able to adapt its method of repair according to which robot has failed.

**Null Hypothesis 7.9:** *Successful dynamic self-repair is impossible to achieve using hardware robots.*

H7.9 tests whether self-repair in hardware proves more complex than in simulation, and investigates how embodiment affects the actions involved in self-repair procedures.

## 7.3 Self-repair Concept



{{1,2,3,2},{2,4,4,3},{3,3,3,5},{1,4,2,4}}
{{A,B,C,D},...}

**Fig. 7.1:** A reminder of the quadruplet data structures used to provide recruitment lists from which to generate an organism's morphology. Each quadruplet represents one docked connection within the structure, A represents the Temporary ID number of the robot to perform recruitment, B the port it is to recruit with, C the port of an approaching robot which it is to recruit for, and D the Temporary ID to be assigned to the robot which connects according to these details. Reproduced from Fig.(6.3).

The concept underlying self-repair is based around detecting a failed module and effecting its removal, then replacing it and reforming around the replacement. As structures are defined by a quadruplet Recruitment List, see Sec.(6.2.4) and Fig.(7.1), then any given failed module will either have modules lower in the hierarchy than it (local slaves) or higher in the hierarchy than it (a local master), or very often both. It should be clearly noted here that the direction of the hierarchy runs downwards for each docked connection outward from the seed robot. If we consider Robot 2 in Fig.(7.1) then it has a single slave, Robot 3 on Port 4, and a single master, Robot 1 on Port 3. Robot 3 has a slave of its own, Robot 5, which can also be considered as a subslave of Robot 2. Given the form of the recruitment list data structures an Omni-Pi-tent module may have up to 4 local slaves and up to 1 local master. A module without any local slaves will be a module on the periphery of a structure, repair here is a matter of detaching and recruiting for a replacement, much like normal self-assembly. A module without a local master will be the master of the structure it is in, recovering from these failures was developed later and is covered in Sec.(7.9). Specifically it was handled by reprocessing the recruitment list to ensure that the failed module gets treated as if it did have both slaves and a master. The failure case on which the bulk of the work is therefore focused is on that of a failed module with both a master and slave(s) attached.

The process of repair can therefore be split into those activities which must be performed by the immediate local slaves of the failed module, and those which must be performed by the immediate local master. The slave side of the repair process is responsible for removing the failed module, and the master side for the recruitment of a replacement, then the slave side continues in the repair procedure until it has re-docked to the replacement. To some extent this parallelises aspects of the repair procedure, enabling re-recruitment to occur without needing to wait for the disposal of the failed module, this should in theory provide a speed advantage over forms of self-repair which would require removal to be entirely complete before replacement could begin. This type of parallelisation also allows the re-use of many aspects of the self-assembly procedure during the recruitment both of a replacement module to the master of the failed module, and of the slave-side substructures of the failed module to the replacement module. Fig.(7.2) shows the key stages of the self-repair procedure within a static scenario.

During the moment at which a self-repair operation begins the modules neighbouring a failed module can each take one of the following roles:

- **Master to the Failed Module (MFM)**, The robot which is specified in the quadruplet list as the failed module's immediate master handles the re-recruitment side of the self-repair procedure. This consists of performing a modified version of the standard self-assembly recruitment procedure so as to attract a replacement module, once the replacement is docked the MFM performs some updates to globally shared recruitment list data after which it returns to ordinary operation as a member of the structure. This module, for a brief period immediately after undocking itself from the failed module, also acts to steer the main structure, those parts of the hierarchy above it, away from the failed module so as to provide room for recruitment to occur. This is done by the MFM

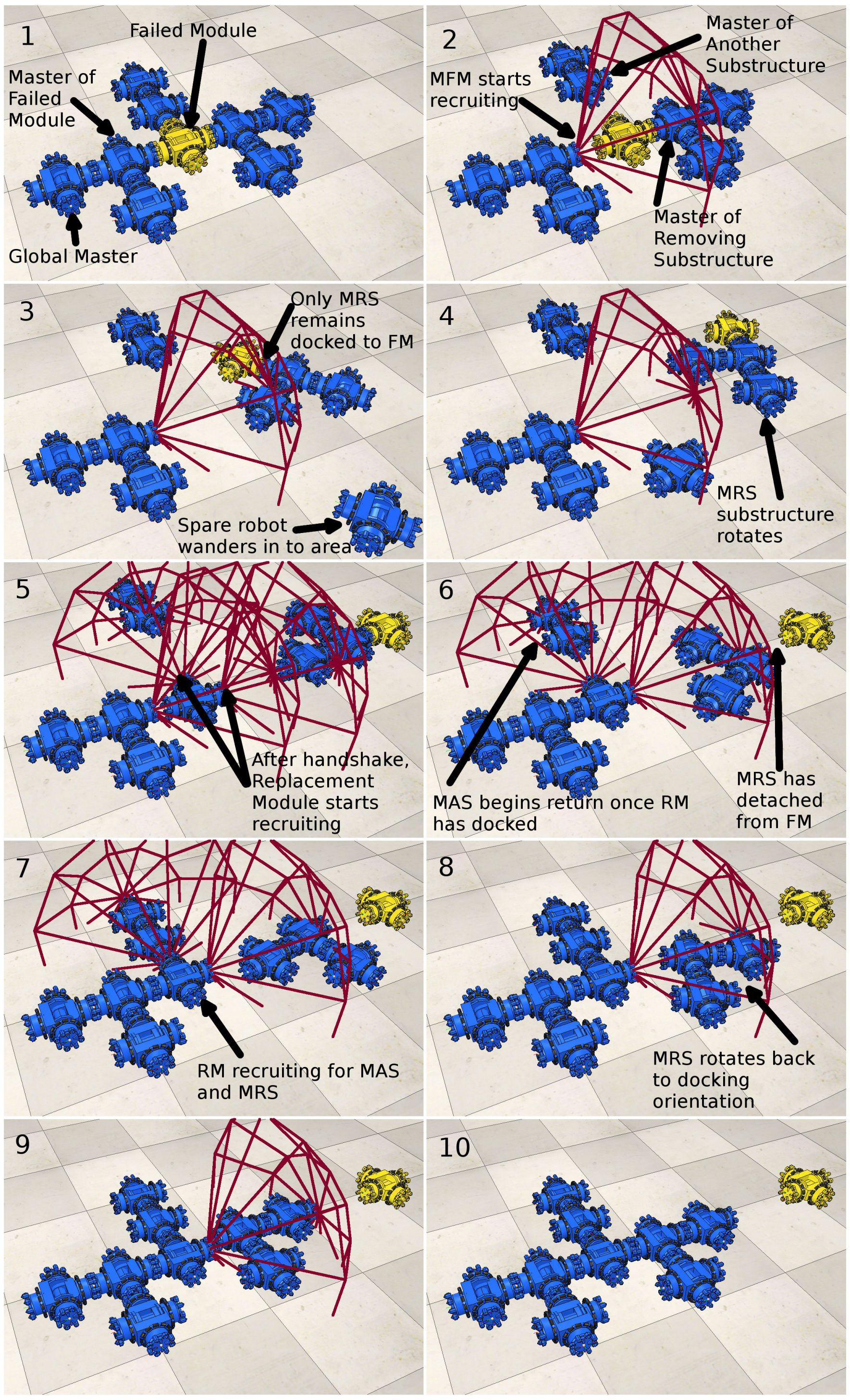

1
Failed Module
Master of Failed Module
Global Master
2
Master of Another Substructure
MFM starts recruiting
Master of Removing Substructure
3
Only MRS remains docked to FM
Spare robot wanders in to area
4
MRS substructure rotates
5
After handshake, Replacement Module starts recruiting
6
MAS begins return once RM has docked
MRS has detached from FM
7
RM recruiting for MAS and MRS
8
MRS rotates back to docking orientation
9
10

**Fig. 7.2:** A series of images present the concept of self-repair underpinning the controller developed in this work. The Failed Module is highlighted and the actions of those around it labelled. Note how only the MFM, MRS, MAS and RM are involved in self-repair, the global master and other parts of the structure simply continue to serve as immediate masters or slaves to the robots around them. For purposes of visual clarity, the self-repair process in these images uses a static rather than dynamic strategy, a dynamic strategy will appear similar when viewed from a reference frame moving at the correct velocity. Also for visual reasons, retreat distances shown in this diagram are not to scale.

module “hallucinating" the existence of a very close obstacle on its proximity sensors on the docking port which previously attached to the failed robot. Such a falsified sensor reading can be transferred via the existing MNS methods to the organism’s global master, encouraging the whole organism to take evasive action and back away from where the failed modules is located. This retreat ensures space is available around the MFM module so that wandering robots can gain access and be recruited.

- **Master of the Removing Substructure (MRS)**, Among the robots connected to a failed module which served as its slaves one module will be promoted to act as the MRS. The MRS will control the substructure responsible for dragging away the failed module, will perform the safe disposal of the failed module, and will then guide itself back to the replacement module such that it, and its attached substructure, can reconnect to the main structure. Unless a failed robot was a peripheral robot, hence had no slaves connected to it, then one, and no more than one, of its slave robots will always take on this role. Methods, see Sec.(7.4.4), exist to ensure that this can still be done even if none of its slaves had subslaves attached.

- **Master of Another Substructure (MAS)**, Robots which were slaves to the failed module and have slaves of their own also form substructures. The actions performed by these substructures are simpler than that substructure command by the MRS module. They act only to retreat from the immediate vicinity of the failed module, so as not to interfere with recruitment of the replacement module, and then return to re-dock once the replacement is in place. The timing of this return is handled by observing changes in the shared Recruitment List quadruplet array to check whether the RM is yet in place. Again like for the MRS, the MAS is in control of this substructure and its slave robots simply follow its commands and provide it with sensor data no differently than if they were performing their roles within a complete organism. The use of these extra substructures provides the advantage of not having to break up the structures attached to other slaves of the failed module, this is expected to accelerate recovery of the structure as compared to strategies which involve breaking up a structure, or can, such as [51], only preserve one substructure.

- **Lone Module (LM)**, Modules which served as slaves to the failed module and which have no slaves of their own detach from the structure and retreat. As they lack any slaves of their own they do not have substructures attached, they are therefore physically indistinguishable from any other free-wandering, as yet unrecruited, robot. Such LM modules therefore enter the “Escape Dock" state, developed to aid in self-assembly procedures in Sec.(6.2), revert their Temporary ID values to zero and return to the free “Wandering" state. These modules may later be re-recruited, or others may be in their place.

The above roles are assigned by consideration of the Recruitment List quadruplet data structure which describes the docked connections within an organism. Each robot assigns itself the appropriate role without requiring any communication except for being in possession of a reasonably up-to-date copy of this Recruitment List. During self-repair a module in a further special role also enters the picture:

- **Replacement Module (RM)**, The RM involved in a self-repair procedure is, at the start of the procedure, just one of many free wandering modules. However once it completes the IR handshaking procedure involved in the transition from "Rotate to Dock" to "Approach to Dock" it behaves somewhat differently from an ordinary recruited module. For any connections which the Recruitment List requires it to recruit for, instead of sending an ordinary general recruitment message via its line-of-sight IR emitters it sends a specialised "Type 10" message. These Type 10 messages are read only by robots with the Temporary ID's of the MRS and MAS modules. Any other robots detecting these Type 10 messages treat them as repulsion instructions and vacate the area around the re-recruitment process. Once the RM has all connections on the Recruitment List, for which it is master, filled the self-repair procedure is complete.

This self-repair procedure, as presently described, has great similarities to [51] although significant advantages are provided by the Omni-Pi-tent hardware on which it has been implemented. Among other considerations, the use of genderless active docking on all ports and the use of an omnidirectional drive mean a much wider array of structures can be handled, and connections and breaks can occur as determined solely by the requirements of the hierarchy of the organism without needing to account for limitations of differential drive robots which could only form active connections with some of their ports. Dynamic Self-repair is achieved by applying velocity transforms to this repair procedure, much as Sec.(6.2.8) does for self-assembly. Self-repair however is a procedure requiring more complex actions to be performed, and longer term relative positions to be maintained. The methods used to keep these relative alignments are discussed in Sec.(7.4.7).

The use of mobile substructures within this self-repair procedure is a novel development, made possible by Omni-Pi-tent's omnidirectional drive and MNS [26] co-ordinated control, previous self-repair work [51] had noted how the ability for multiple robots to dock together as a precisely co-ordinated group would be important to enable more versatile self-repair procedures.

It should be noted that the self-repair method developed here relies almost exclusively on the robots neighbouring the failed robot performing special functions. Robots elsewhere in an organism, including those several steps further up or down the hierarchy are, unless one chooses to add features to the self-repair controllers such that they will be explicitly informed, oblivious to the self-repair operation taking place amongst them. Robots not immediately neighbouring the failed module can continue to serve the organism by acting as slaves and/or masters within the hierarchy and can continue to play their usual roles in actuation and sensing.

## 7.4 The Self-repair Controller

In the Dynamic Self-repair project a controller has been developed from the earlier self-assembly controller of Chapter 6. Developing a self-repair controller in this way has precedent [51] but in previous work was done as an add-on after the self-assembly controller had been designed. The Dynamic Self-repair project has benefited from having developed a self-assembly controller with the requirements of self-repair already in mind, the self-assembly controller was therefore well equipped to provide functionality which could be harnessed within self-repair.

This controller uses Recruitment Lists as they presently are when a failure occurs, not just the Structure Recruitment List which represents the plan for a completed organism. The self-repair methods therefore ought to operate as effectively for failures occurring during the assembly of an organism as it does for failures within a completed structure. As the Recruitment List is shared during self-assembly, and updated copies shared over Wi-Fi each time a robot docks or undocks, there is no requirement for robots to communicate descriptions of the structure to each other during self-repair, they already have this information and can make all necessary decisions using data derived from it. It should also be noted that organisms experiencing self-repair are rebuilt to form the original shape, and that modules, unless Lone Modules, maintain their Temporary ID values and return to the same position in the structure after the repair as they had before it. Possibilities for other forms of repair are speculated on in Sec.(8.3.5).

While the self-repair procedure relies on both the IR and Wi-Fi communications it does not depend on them operating instantly or at the same time. The failure of a large proportion of IR messages is expected and therefore IR messages are continually repeated over port-to-port links for the duration of their relevance. Wi-Fi failures are not considered in this work, they could have more serious effects as Wi-Fi messages are often used to perform single updates to global information. Modules also do not need perfectly synchronised clocks, it is expected that different modules around a failed module will detect the failure at slightly different times and the strategy is tolerant to this potential for modules around the failed one to react to it in any order and with time between them.

Fig.(7.3) shows the finite state machine for MLR self-assembly modified to include extra states required to enable self-repair. The new states added include those which a module enters upon detecting a neighbouring module has failed, a series of states involved in the repair procedure, and an extra state within the docking routines which replacement robots enter upon docking. It should be noted that if self-assembly were ongoing at the same time as self-repair a module in any of the new states, except those related to Wi-Fi controlled retreat actions, can continue to recruit robots to it on any ports where it is not involved in self-repair.

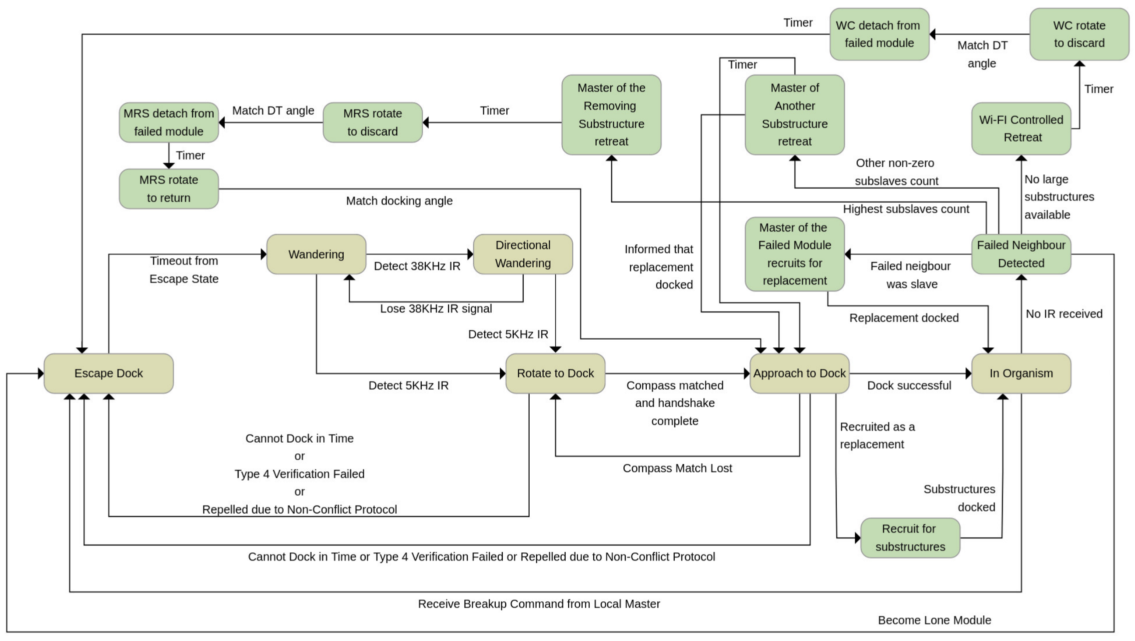

Timer
WC detach from failed module
Match DT angle
WC rotate to discard
Timer
Timer
Master of Another Substructure retreat
Wi-FI Controlled Retreat
MRS detach from failed module
Match DT angle
MRS rotate to discard
Timer
Master of the Removing Substructure retreat
Timer
MRS rotate to return
Match docking angle
Other non-zero subslaves count
Highest subslaves count
No large substructures available
Timeout from Escape State
Wandering
Detect 38KHz IR
Directional Wandering
Informed that replacement docked
Master of the Failed Module recruits for replacement
Failed neigbour was slave
Failed Neighbour Detected
Lose 38KHz IR signal
Replacement docked
No IR received
Detect 5KHz IR
Escape Dock
Detect 5KHz IR
Rotate to Dock
Compass matched and handshake complete
Approach to Dock
Dock successful
In Organism
Recruited as a replacement
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Compass Match Lost
Substructures docked
Recruit for substructures
Cannot Dock in Time or Type 4 Verification Failed or Repelled due to Non-Conflict Protocol
Receive Breakup Command from Local Master
Become Lone Module

**Fig. 7.3:** A simplified diagram of the FSM underlying the self-repair controller, newly added states are highlighted in green. See main text for explanation.

The discussion continues by considering the transition in the state machine which causes robots to enter the “Failed Neighbour Detected" state.

## 7.4.1 Detecting Failed Modules

Within a structure all modules use IR port-to-port links to transfer data necessary for co-ordinated MNS [26] activity. As failed modules, in this work’s scenarios, shut themselves down upon endogenous detection of an internal fault, detecting a failed module is therefore a matter of the modules around it observing the lack of outward IR messages from its ports. It would not be wise to rely on faulty modules to self-report their fault to neighbours as that would depend upon them being in sufficiently operational condition to transmit warning messages, something which would not happen in the case of faults in IR LED failures, Wi-Fi failures, power failures or prolonged failures of the Pi, the I2C bus and other central internal circuits. While previous work has [26] sometimes used an explicit “heartbeat protocol" message for these purposes, on Omni-Pi-tent the port-to-port infrared messages continually carrying sensor data upward to the global master and actuator commands downward towards slaves can serve this purpose while also serving their primary data carrying function. All modules within an organism continually monitor the ports on which they are docked for IR messages, it is expected that some IR messages will be corrupted by noise in transit therefore not all such messages get through, however it is statistically very unlikely that a long series of consecutive messages would all be corrupted. Robots are therefore programmed to consider a period of 6 seconds without any valid IR messages on a port which is docked to be a sign of the failure of the robot neighbouring them on that port. When this happens they enter the “Failed Neighbour Detected" state and must make decisions, based on Recruitment List derived information, on how to react.

### 7.4.2 Processing Recruitment Lists

It has been assumed, and infact found in practice with hardware whilst preparing the work of Sec.(6.5), that a single module attempting to move an uncooperative module would not be able to do so alone. Therefore modules connected to the failed module must act collectively to remove it, except in some circumstances, see Sec.(7.4.4). This collective action requires a group of modules connected to the failed module to become the Removing Substructure. Robots entering the “Failed Neighbour Detected" state must therefore reach decisions on which role to play in the self-repair process. These decisions must be reached in such a way as to ensure that there will be one, and only one, substructure attached to the failed module serving as the Removing Substructure. The decisions must also be reached without relying on any communication being passed through the failed module, and while Wi-Fi could be used without requiring communication through the failed module, making these decisions with no communication at all between the neighbours of the failed module is desirable. By making these role allocation decisions without communication there is no time required for robots to identify the Wi-Fi identities of the other neighbours, these are not the same as the Temporary ID numbers, of the failed module and no time taken up by back-and-forth negotiation. Each robot runs the pseudocode presented in Fig.(7.4) to decide the role it is to take. As it works from the Recruitment List data structure, of which all robots should have an identical copy, the neighbouring robots will be able to take roles without conflicting or leaving the role of Master of the Removing Substructure unfilled.

By selecting the largest substructure to serve as the Removal Substructure it is ensured that in any scenario where the failed module has a slave attached which in turn has subslaves the system will be able to use this substructure of $\geq 2$ modules as the Removal Substructure. If no large enough substructure is available then a Wi-Fi controlled retreat procedure is used, see Sec.(7.4.4). Retreat directions are set so that the Removal Substructure pulls a failed module directly outward and away from the failed module’s port to its master, note that unlike the modules in all the other roles around a failed module the Master of the Removal Substructure remains docked, hence dragging the failed module with it. Other retreating modules, be they Lone Modules or Masters of Another Substructure, retreat directly away from the failed module and use the single-sided disconnection capability which Omni-Pi-tent’s genderless docking provides to escape the clutches of their failed master. The Master to the Failed Module

```
if(NoIRMessagesSeenInLast6SecOnPort[PortNum]==TRUE and DockingPort[PortNum]==DOCKED and TimeNow-PortDockedAtTime[PortNum]>6)then
  if(PortNum==PortToOurMaster)then--a master of ours has failed
    SubslaveCountArray={0,0,0,0} --example derived from Lua code, Lua arrays start at 1
    DeadMasterSlavesAttachedArray={0,0,0,0}
    DeadMastersSlaveTempIDs={0,0,0,0}
    index=1
    while(index <= 4)do--check if the indexth port on our dead master had a slave docked to it
      if(index ~= DeadMastersPortToMaster and RobotHasPortDocked(DeadMastersTempID,index)==TRUE)then --by only being true if the thing on this port is a slave we ensure we don't miscount the dead master's own master's structure as a substructure
        DeadMasterSlavesAttachedArray[index]=1--if so find what that robot's TempID was
        DeadMastersSlaveTempIDs[index]=FindTempIDofSlave(DeadMastersTempID, index)--count how many slaves it had, put this number in the indexth place of SubslaveCountArray
        SubslaveCountArray[index]=CountSubslaves(DeadMastersSlaveTempIDs[index])
      end
      index=index+1
    end
    MySubslaveCount=CountSubslaves(MyTempID)
    TempIDsOfSubstructsEqualToOurSize={0,0,0,0}--this will include our TempID and the TempID of the masters of any substructure the same size as the one we master, any spare places will contain zeroes
    index=1
    while(index <= 4)do
      if(SubslaveCountArray[index]==MySubslaveCount)then
        TempIDsOfSubstructsEqualToOurSize[index]=DeadMastersSlaveTempIDs[index];
      end
      index=index+1;
    end
    --the master of the biggest substruct will handle the dragging away
    if(MySubslaveCount>=max(SubslaveCountArray) and MySubslaveCount>=1)then
      if(OccurrencesOf(MySubslaveCount)In(SubslaveCountArray) > 1 and (MyTempID>=max(TempIDsOfSubstructsEqualToOurSize))then
        UndockOnPort(PortNum)
        RetreatDirection=FindRetreatDirection(PortToOurMaster)--retreat directly away from failed module
        BecomeMasterOfAnotherSubstructure()
      else--only become MRS if either there are no other substructures as big as ours, or there are others but we have the highest TempID among the TempIDs of the substructure's masters
        DeadMastersPortToMaster=FindPortToMaster(DeadMastersTempID)
        RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)--retreat so as to pull failed module away from its master
        BecomeMasterOfRemovalSubstructure()
      end
    elseif(MySubslaveCount>=1)then--for robots with subslaves attached, but which aren't the biggest substructures
      UndockOnPort(PortNum)
      RetreatDirection=FindRetreatDirection(PortToOurMaster)
      BecomeMasterOfAnotherSubstructure()
    elseif(sum(DeadMasterSlavesAttachedArray)>=2 and NooneElseHasAtleastOneSubslave()==TRUE)then--if there are no multi-robot substructures attached to the dead master, but there are two or more single robots
      --we must use something unique about the docked single robots to decide which will be in command of the retreat and dump, as against which will just follow along
      RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)
      if(MyTempID>=max(DeadMastersSlaveTempIDs))then--TempID is unique, so we use the highest of them as ``boss" of the retreat
        BeginWifiControlledRetreatAsMaster()
      else
        BeginWifiControlledRetreatAsSlave()
      end
    else--if we don't meet any of those considerations
      RetreatDirection=FindRetreatDirection(PortToOurMaster)
      BecomeLoneModule()
    end

  elseif(PortNum ~= PortToOurMaster)then--a slave of ours has failed
    UndockOnPort(PortNum)
    BecomeMasterToTheFailedModule()
  end
end
```

**Fig. 7.4:** Pseudocode of the algorithm for allocating self-repair roles to the neighbouring robots of a failed module, see main text for details. Pseudocode for the CountSubslaves() function is also shown, see Fig.(7.5).

also undocks when it runs this procedure. As all substructures connected to the failed module are able to remain complete as either Removal Substructures or other Substructures there is no requirement to compare “Repair Potentials" [51] to choose which are to remain and which are to break up. When the decision-making process outlined here is complete, modules transition to the “Master of Another Substructure retreat", “Master of The Removal Substructure retreat", “Wi-Fi Controlled Retreat", “Master of the Failed Module recruits for replacement" or “Escape Dock" state.

```
function CountSubslaves(MasterInQuestion)--counts slaves docked (actually docked, not just according to SRL) as slaves to a robot, and counts all their slaves in turn, and so on
  local output=0
  local RobotIsSubslaveArray={}--says whether a robot with this (index) TempID is in the structure yet
  local RobotMightBeSubslaveArray={}
  for ii=1, 64 do
    RobotIsSubslaveArray[ii] = 0
  end
  for ii=1, 64 do
    RobotMightBeSubslaveArray[ii] = 0 --just tells us if a robot is a docked one, hence could be a subslave before further checking is done
  end
  local indexB=1
  while(indexB<= table.getn(StructureRecruitmentList))do --check over the structure recruitment list
    local StructureInstruction=StructureRecruitmentList[indexB]
   --we check whether the robot with this id is actually docked yet
    local index2=1
    local NotFoundInRecList=1
    while(index2<= table.getn(RecruitmentList))do--we cycle through the recruitment list, the version which has quadruplets removed when a dock is formed, checking to ensure that the TempID of interest is NOT present as fourth element of a quadruplet, hence ensuring it IS docked
      local SInstruction=RecruitmentList[index2]
      if(SInstruction[4]==StructureInstruction[4])then
        NotFoundInRecList=0 --if we can find a quadruplet with the TempID in question as it's fourth element then we know that a robot of that TempID hasn't docked to the structure yet
      end
      index2=index2+1
    end

    if(NotFoundInRecList==1)then
      RobotMightBeSubslaveArray[StructureInstruction[4]]=1--we note that the robot which the quadruplet refers to the recruiting of IS present
    else
      RobotIsSubslaveArray[StructureInstruction[4]]=0 --we note down that the robot ISN'T present
      RobotMightBeSubslaveArray[StructureInstruction[4]]=0
    end

    indexB=indexB+1
  end
  RobotMightBeSubslaveArray[FindHighestLevelMaster(MasterInQuestion)]=1
  local indexH=1
 --we now iterate over each TempID number
  while(indexH <= 64 and MasterInQuestion ~= 0)do
    if(RobotMightBeSubslaveArray[indexH]==1)then --if a robot is docked in place
      --if it is a subslave of MasterInQuestion then we put a 1 in this place of the array
      --print("checking temp ID", indexH)
      if(CheckIfSlaveIsSubslaveOfMaster(indexH,MasterInQuestion)==1)then --traces using a similar process to FindHighestLevelMaster() but terminates upon reaching MasterInQuestion
        RobotIsSubslaveArray[indexH]=1
        --print("is subslave")
      else
        --print("not subslave")
      end
    end
    indexH=indexH+1
  end
  indexH=1
  while(indexH <= 64)do --we now sum up the contents of the arrays
    output=output+RobotIsSubslaveArray[indexH]
    indexH=indexH+1
  end
  return output
end
```

**Fig. 7.5:** Pseudocode of the function for counting subslaves attached below, in the hierarchy, a specific port on a failed module. This provides another example of some of the useful capabilities which the quadruplet data structure enables. Although less efficient, this method was chosen in preference to recursion based methods due to the risk of overflowing the stack which recursion could cause if run on a microcontroller at some future time. See also Fig.(6.4) for the FindHighestLevelMaster() function.

### 7.4.3 Removing the Failed Module

Entering the "Master of The Removing Substructure retreat" state causes a robot to follow a series of procedures relating to removing the failed module. This begins with a 20 second period of retreat. During this period the MRS module acts as master of an organism consisting of its attached substructure, via the same MNS principles described in Sec.(6.2.7) sensor data is forwarded to it and actuator commands sent out to its slaves. The MRS module keeps the substructure aligned to the compass direction it had at the moment of detecting the failure and drives outward away from the direction of the failed module's port to its master. This choice of direction is due to the inability of Omni-Pi-tent docking ports to slide sideways past one-another. At a later point in the retreat, once given time (7 seconds) to get clear of the main organism from which the substructure is retreating, the MRS robot also begins considering the angle to which it will turn before detaching from the failed module. Wi-Fi messages sent from the MFM, and later the RM, provide information on the motion direction of the main structure, the modules above the failed module in the hierarchy. These are used to inform calculations which ensure that if the retreat direction is taking the MRS's substructure away in a direction forward along the main structure's motion then during the final stages of retreat the substructure will gain a sideways components, relative to the main substructure's motion, and rotate to an angle so that once detached the failed module will not be littering the main substructure's path.

After the period of retreat in the “Master of removing Substructure retreat" state the MRS module transitions to the “MRS rotate to discard" state. This state continues for as long as necessary for the substructure to rotate such that it is facing in the correct direction, as calculated from the main structure's motions, so as to detach the failed module without impeding the main structure's progress. This angle at which to dump the failed module, and the compass direction along which to perform the later stages of retreat away from the main substructure are calculated from the long term averaged motion of the master side structure, as read from the MFM or RM's Wi-Fi wheel speed messages. The next state entered is “MRS detach from failed module", the genderless docking hooks on the relevant port of the MRS module are released and the MRS module spends a timed period driving directly away from this port direction.

A failed module once dumped, effectively becomes simply an obstacle in the environment, to be detected and navigated around by other robots using their proximity sensors. The same considerations would apply if a module were to fail while in a free “Wandering" state.

The MRS module uses the “MRS rotate to return" state to match rotation direction to that which, from compass data supplied in Wi-Fi messages from the MFM or RM, will be needed for it to reconnect to the replacement module. The Master of The Removal Substructure robot enters the “Approach to Dock state" once roughly aligned to the required direction and begins searching for the appropriate IR cone of recruitment signals. Once this occurs it can return to dock to the replacement module and complete the repair of the structure. Routines within the controller code ensure that if the RM has not yet docked then the MRS will remain in the “Approach to Dock" state and keep some distance from the MFM so as to enable “wandering" robots to wander in to place.

However there is also an alternative procedure available for situations when a failed robot has no single substructure attached below it in the hierarchy which would be capable of removing it.

### 7.4.4 Mergeable Nervous Systems over Wi-Fi

If a failed module has slaves attached but none of these slaves have subslaves the question arises as to which of the connected modules will be responsible for its disposal. This is further complicated by our understanding that a single module alone will not be able to effectively drag the failed module away, each module afterall has the same weight and the same frictional properties on its wheels, a powered down module's wheels are technically back-drivable but the forces involved are large and in excess of the torques which powered wheels can provide. A failed module with two or more slaves attached has enough slaves present to drag it away, but as these slaves are each connected to the failed module, and not connected to each other, they cannot rely on conventional MNS methods to transfer sensor and actuator data between them. The use of Wi-Fi as a channel to carry what would usually be infrared port-to-port commands, as developed in Sec.(6.5) to mitigate low data rates in the IR hardware of the Omni-Pi-tent module design, can be harnessed here.

Around the failed module the immediate slaves decide, based on comparing their Temporary ID numbers to the others nearby as specified in the Recruitment List, which will act as MNS master and which will be slaves to it. Unlike port-to-port IR communications Wi-Fi communication also requires the permanent ID numbers of modules by which they will be addressed, modules therefore send broadcast Wi-Fi messages to all Wi-Fi addresses which act as requests for robots with the correct Temporary ID to respond with their Permanent ID. With this complete MNS control can proceed ordinarily using features included within the “In organism" state of the controller for those robots which will act as MNS slaves, except that theses messages are supplied over Wi-Fi rather than IR links and the instantaneous centre of rotation calculation is modified to use to relative positions of modules separated by a failed module located inbetween them.

For the master of a Wi-Fi controlled retreat substructure similar behaviours and timings occur as used by an MRS module, again with a timed transition in to a rotation state and then the detachment of the failed module.

Once the failed module is discarded the Wi-Fi controlled retreat structure breaks up in to Lone Modules, each of which is in the “Wandering" state. This is done by the master of the Wi-Fi controlled retreat substructure sending targeted Wi-Fi messages to each robot which is also connected to the failed module. As the Wi-Fi controlled retreat structure does not involve any connections between multiple operational robots there would be no point in attempting to recover it in the way that multi-robot substructures are re-recruited.

Before returning to consider how re-recruitment of substructures, when they are available to act as Removal or Other substructures, is performed the Master to the Failed Module is considered as well as and how the replacement for the failed module is recruited.

### 7.4.5 Recruiting a Replacement

Whilst the removal was taking place the Master of the Failed Module has also played an important role. When it detached from the failed module it began recruiting for a replacement module by entering the “Master of the Failed Module recruits for replacement" state. In this state initial recruitment is identical to that used during self-assembly, however once the recruited module, which will become the RM, has completed the handshake protocol an alternative form of recruitment, known as Type 9 recruitment, is used in the targeted phase where the recruited module is in the “Approach to Dock" state. This alternative form of recruitment is used subject to the condition that the globally non-edited Structure Recruitment List indicates that the robot to be recruited in this location is required to have subslaves, meaning that this alternative form of recruitment is only used if the failed module is required by the Structure Recruitment List to have slaves, not if the failed module was a peripheral module.

Receipt of this specialised message type triggers the recruited robot to begin another new type of recruitment, Type 10, on its docking ports. Type 10 recruitment messages are emitted by the Replacement Module only on ports which both require multi-module structures to connect to them, as per the Structure Recruitment List and for which multi-module structures with the correct Temporary ID values are already in existence, as shown by the Recruitment List. This ensures that Type 10 messages are only used for recruitment where a completed substructure exists which can be recruited for. Where only single robots are required or the substructures required never completed formation before the failure of the failed module occurred, ordinary recruitment methods are used instead to recruit lone robots from the “Wandering" state. Type 10 messages act as recruitment messages when read by robots with the appropriate Temporary ID values as expected for MAS and MRS modules, whilst providing repulsion messages to free wandering robots to ensure that such robots cannot obstruct the repair procedure.

The special roles performed by the MFM module cease to operate as soon as it is no longer recruiting on any ports, the RM module will have taken up these roles, such as sharing Wi-Fi information to guide the MRS substructure back in to place, when it completed the handshake procedure.

### 7.4.6 Re-forming the Structure

Re-forming the structure can only take place once other aspects of self-repair are completed. The failed module must have been removed from the immediate vicinity of the main structure and the replacement module must have been recruited.

Upon observing the change in the Recruitment List which occurs when the RM is docked any MAS modules mastering other substructures can begin an immediate return. The MRS module usually takes longer to perform the retreat, rotate, detach and rotate again steps involved in disposing of the failed module than the time required for the replacement module to dock, so cannot make use of such a timing shortcut.

Returning substructures, whether MRS or MAS headed, use the “Approach to Dock" state in the finite state machine, similar to that used by post-handshake modules. This state is however modified to allow modules which are in command of substructures to spend potentially long periods engaged in wandering like behaviours outside of IR recruitment cones, and also modified to ensure that if timing out of a docking operation these modules acting as substructure masters will **not** break up their attached modules in the way that can occur during MLR recruitment.

The controller offers scope for MAS and MRS substructures to break up after long timeouts and for RM modules recruiting via Type 10 to switch to ordinary recruitment of single modules at that point, making the system more resilient by not relying on substructure based repair. However for the purposes of the experiments in this chapter that feature was disabled to ensure that self-repair strategies could be accurately compared without dynamic or static strategies resorting to ordinary recruitment behaviour if re-recruitment of substructures was delayed.

Once redocking of the MAS and MRS modules is complete the internal states in which all modules in the structure will be are the same as those they occupied before the self-repair operation, except ofcourse that a different hardware module with a different Permanent ID value will have taken the place of the failed module.

### 7.4.7 Guiding the Substructures

The key feature of Dynamic Self-repair is the ability to operate while group motion is maintained. Many of the features so far discussed are very useful to dynamic self-repair, and newly implemented in the field of modular robotic self-repair in general, but it is the ability to operate in motion which crucially distinguishes Dynamic Self-repair from some of the other strategies tested in the following experiments.

Upon detaching from the failed module, the Master to the Failed Module (MFM) begins transmitting a combination of wheel speed and compass data over Wi-Fi broadcast. These messages are read by the MAS and MRS modules and used to set reference frames from which their retreat and return motions are structured. Robots acting as MAS and MRS modules record the sum of the wheel speeds received over Wi-Fi and record the sums of their own motions to perform odometry calculations and calculate which driving direction will be necessary to return them to a region in which they will find the recruitment IR cone of the relevant docking port on the RM module.

Even in the perfect environment of simulation this odometry data gives relatively poor readings with returning modules often overshooting or undershooting the point at which they cross the trajectory of the main structure, hence missing the recruitment cones. Therefore the infrared line of sight messaging is also used to guide MRS and MAS substructures back. The Replacement Module within the main structure commands all undocked ports on the structure to, for a fraction of a second every few seconds, emit messages at full power. These messages identify which module, by Temporary ID, within the substructure is emitting the message. Any robot in an MAS or MRS controlled substructure may receive such messages and forward them on to the MRS of MAS module of its substructure. Using a combination of compass readings, for the emitting and receiving robots, and of Structure Recruitment List derived knowledge of distances between robots with a given Temporary ID and the location of the RM within the main structure, an MRS or MAS robot can switch to a behaviour in which it attempts to enter the recruiting cone by “orbiting" around the main substructure in whichever direction, given knowledge of the Structure Recruitment List, allows it to travel a short distance to reach the recruiting cone.

Dynamic self-repair is therefore able to reliably return detached substructures to the correct location to perform docking procedures despite the motion of the main structure during these events.

## 7.5 Naive Breakup Vs Static Self-Repair Vs Dynamic Self-Repair

In the initial set of experiments 4 forms of self-repair were compared. Dynamic Self-repair (DSR), Static Self-repair (SSR), naive Dynamic Self-repair (nDSR) and naive Static Self-repair (nSSR). Dynamic Self-repair has been discussed in detail above. Static Self-repair provides a close analogue to the strategies developed by Murray [51], acting like DSR but pausing the group motion at the moment of failure and remaining static until the structure is repaired. The key difference between these two self-repair strategies is that Dynamic Self-repair can parallelise the self-repair and locomotion tasks whereas Static Self-repair must perform them sequentially, this should enable faster overall task completion times. The two “naive" strategies are based on Murray’s strategies referred to as “naive", in these strategies he fully broke up the structures then formed them again. In my experiments the advantages of the Omni-Pi-tent platform make a complete breakup utterly superfluous, so with my platform a complete breakup is performed of everything *below* the failed module in the Recruitment List hierarchy, while leaving other sections of the structure unchanged. When the breakup happens all slaves of the failed module undock from it and become independent robots, they send a command down in to any sub-structures for which they are local masters which causes those sub-structures to recursively break up into free robots in the same fashion. On the master side naive self-repair involves only an undocking from the failed module and a brief retreat away from it so as to make space for replacements to be recruited, this re-recruitment is done exactly like ordinary self-assembly. The naive strategies, in effect, scarcely modify the self-assembly controller at all, except for the addition of failure detection features comparable to those required in SSR and DSR. These naive strategies have similarities to [166], except that in Yim's work the modules were separated by an external force whereas here the modules separate themselves before reattaching. nSSR and nDSR differ in that in nDSR Omni-Pi-tent's ability to dock during motion, as displayed in the MLR experiments of Chapter 6, means that the main structure can keep driving whilst this recruitment of a replacement and of replacement subslaves for it takes place, in nSSR the structure's overall motion stops after the brief retreat and remains static until all robots have been recruited in to place to complete the repair. The reliance of the nSSR and SSR strategies on communicating to the global master that self-repair is underway, so it can cease group motion, mean in practice these methods could be considered as somewhat less decentralised than DSR and nDSR. Overall then, comparing nSSR vs SSR and nDSR vs DSR looks at the effect of preserving substructures while repairing as compared to purely re-recruiting, this tests null hypothesis $H_0$7.4. Comparing SSR vs DSR and nSSR vs nDSR looks at the effect of maintaining group motion whilst repairing, testing $H_0$7.2.

Example videos of all four strategies are available at [218],[219],[220] and [221] and in the supplementary material (Example_DSR_scenario.mp4, Example_nDSR_scenario.mp4, Example_SSR_scenario.mp4, Example_nSSR_scenario.mp4).

## 7.6 Experimental Scenarios

In the experiments performed to compare DSR, nDSR, SSR and nSSR self-repair strategies a structure was initially formed at a starting point by an MLR based self-assembly strategy, although in the self-repair experiments it was necessary that the seed (global master) for the structure held stationary until the structure was completed. This initial self-assembly was necessary only because of the nature of the V-REP simulation and the difficulties V-REP would cause if one wished to initiate a simulation with modules already in a docked state, the time taken here did not count towards the measurements used in the analysis. Once that self-assembly was complete the structure began a drive towards the opposite end of a long arena, and the timer clock began ticking. 10 seconds after starting motion, the exact time at which this event occurred was not expected to make any difference to the effect it would have on the structure's progress so 10 was used as an arbitrary value, a failure was injected in to a selected module within the structure. At that moment the failed module's port-to-port communication ceased. The task for the other modules in the structure was to identify this failure and perform the self-repair procedure, with static strategies stopping group motion to perform the repair and dynamic strategies continuing along the arena. Unlike in some of the work in [51] the removal of the failed module is performed without any cooperation on the failed module's behalf, and the failed module remains failed, and acts as an obstacle in the arena, after disposal. The recruited replacement is always an operational robot recruited from a free wandering state. The finishing time for each run was recorded as the moment when a completely repaired structure crossed a finishing line at the far end of the arena, as in the self-assembly experiments of Chapter 6 if a structure crossed the line before completing this repair it would turn back and travel back and forth until the repair was complete and it would drive to the finishing line again. To investigate $H_0$7.3 and $H_0$7.5, times were also recorded which noted when the repair procedure began and ended, regardless of other aspects of the locomotion task. In all scenarios 30 robots were present in the scene at the start, an arena of 7 m width was used with the finish line placed 10 m away from the starting point, and 20 minutes of simulated time were given as a maximum time after which a run was considered timed-out.

To ensure timely completion of the initial self-assembly robots were initially positioned so they were evenly scattered, not over the whole arena, but over an area towards the starting end. This was necessary to avoid experimental runs failing due to the initial assembly of the structure never completing, as could happen if it relied on robots randomly wandering across a large fraction of the arena before coming in to recruitment range. This scattering within a limited area is still considered plausible within real scenarios, as robots would often be expected to enter a region together by the same route and be close to one-another at the time when a structure begins to form.

This scenario was run for 7 different structures, see Table.(7.1), some derived from structures used in [51] and others adapted from Sec.(6.3.1), and for each structure runs were performed involving the failure of a variety of different modules, as specified by Temporary ID number, within it. 100 runs of each of the nSSR, SSR, nDSR and DSR strategies were performed for each combination of structure and failed robot. On the Viking HPC cluster [84] each set of 100 parallel runs, the largest number feasible due to the number of available nodes and the requirement, see Sec.(4.9), of only 1 V-REP instance per node, took around 24 hours to complete. Fig.(7.6) shows a still image from one of these runs.

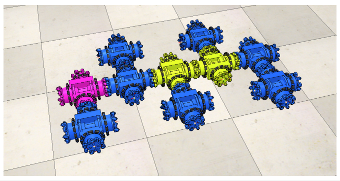
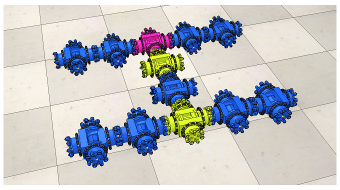
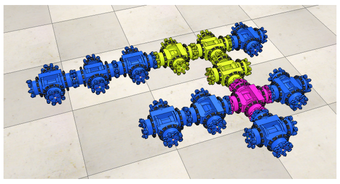
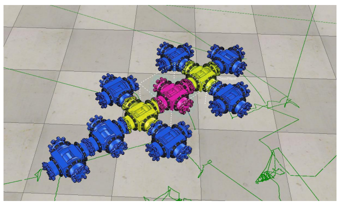
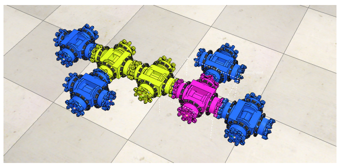
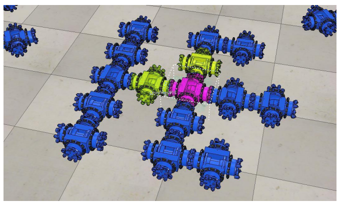
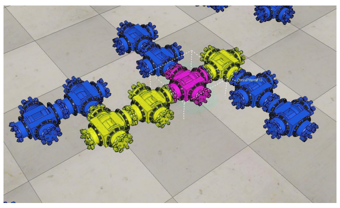
| Name | Image | Temporary IDs of fault injected modules | Recruitment List |
|---|---|---|---|
| 10B |  | 5,7 | { {1,4,2,2},{1,3,3,3}, {3,4,2,4},{3,1,1,5}, {5,4,2,6},{5,3,3,7}, {7,4,2,8},{7,1,1,9}, {9,4,2,10} } |
| 12A |  | 2,4 | { {1,4,2,2},{1,1,4,6}, {1,3,4,5},{5,2,4,8}, {6,2,4,7},{2,4,4,3}, {3,2,4,4},{4,1,4,11}, {4,3,4,10},{11,2,4,12}, {10,2,4,9} } |
| Rand |  | 6,7,9 | { {1,2,2,2},{1,3,2,3}, {1,4,2,4},{4,4,3,5}, {1,1,2,6},{6,4,2,7}, {7,1,4,8},{7,3,2,9}, {9,4,3,10},{10,1,1,11}, {11,3,3,12} } |
| S1 |  | 2,5 | { {1,1,3,5},{1,3,1,2}, {2,4,4,9},{2,2,4,10}, {2,3,2,3},{3,4,4,4}, {5,4,2,8},{5,1,2,6}, {5,2,2,7} } |
| S2 |  | 4,5 | { {1,1,1,4},{1,2,2,2}, {1,3,2,3},{4,3,2,5}, {5,3,2,6},{5,4,4,7} } |
| S3 |  | 2,10 | { {1,1,4,3},{1,2,4,10}, {1,3,4,2},{1,4,4,4}, {4,2,2,5},{2,2,4,7}, {3,2,4,6},{7,3,3,8}, {6,1,1,9},{10,2,4,11}, {11,1,2,12},{12,4,4,14}, {11,3,2,13},{13,4,4,15} } |
| S5 |  | 4,7,8 | { {1,1,4,4},{1,3,4,7}, {1,4,1,2},{2,3,3,3}, {4,3,4,5},{5,2,4,6}, {7,2,4,8},{8,3,3,9}, {9,4,1,10} } |

**Table 7.1:** Structures used in the self-repair experiments, each structure is named with a recruitment list and an image displayed. The seed robot (pink) and those which were injected with faults (yellow) are highlighted. 10B, 12A and Rand are sourced from [51], the rest are sourced from Chapter 6 with S2 enlarged and modified to make the repair task more interesting.

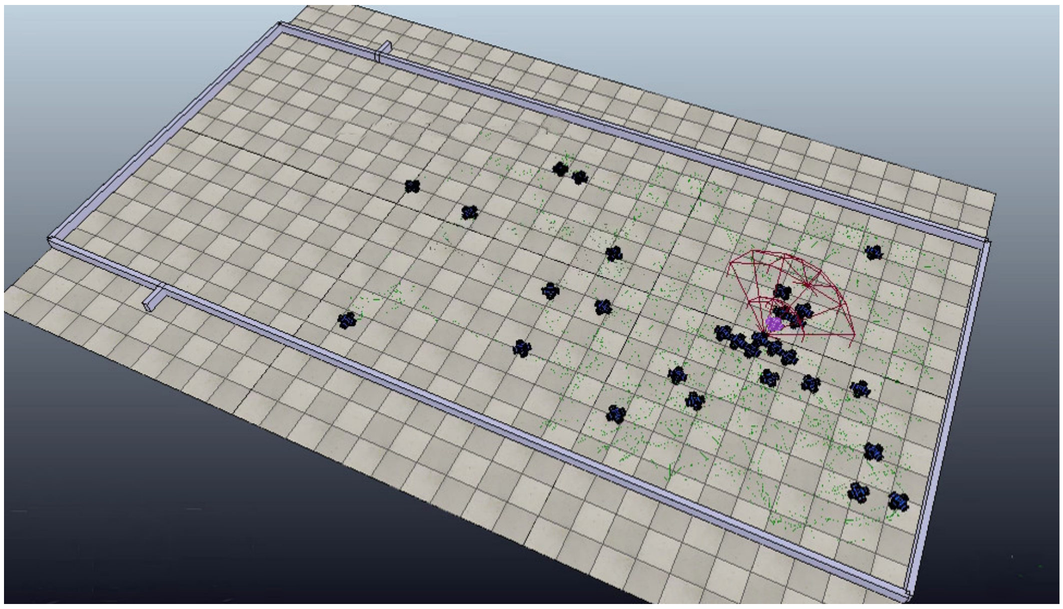


**Fig. 7.6:** Screenshot from a self-repair experiment in which Structure 5 experiences a failure of robot 7, highlighted, and performs a DSR strategy repair. The run from which this still was taken completed the initial assembly, the self-repair and the locomotion task in 5 minutes 43 seconds.

## 7.7 Results

Similarly to in the self-assembly experiments a proportion of the cluster runs timed out without completing, these runs could not be subjected to further analysis however the proportion which completed provides an interesting metric for comparing the reliability of the four strategies. Whereas the cluster scripts developed at the time at which self-assembly experiments were run made it difficult to fairly compare rates of time-outs, as many of the timed out runs were not recorded due to the way in which the earlier cluster scripts saved results, this data was readily available for self-repair runs with an improved cluster script in operation. These proportions of completion are therefore shown in Table (7.2).

R [25] was used for analysis of results from those runs which did not time-out, Mann-Whitney tests were again used to calculate p-values for these non-parametric distributions with Vargha-Delaney A-tests used for effect size calculation. Tables (7.3) and (7.4) summarise the p-value and A-test scores in the self-repair runs while several scenarios of particular interest are discussed with the graphs in Figs.(7.7), (7.8), (7.9), (7.10) , (7.11), (7.12), (7.13), (7.14) and (7.15).

Tables (7.5) and (7.6) summarise p-value and A-test scores when comparing only the time taken for the repair itself to complete rather than the total time for the repair and the locomotion task to finish, Figs.(7.16),(7.17) and (7.18) show boxplots for the repair time distributions in some of the scenarios.

The size of a structure below the failed module can be considered either as an absolute count

| Structure | Temp. ID of Failed Robot | nSSR | nDSR | SSR | DSR |
|---|---|---|---|---|---|
| 10B | 5 | 3% | 95% | 45% | 72% |
| 10B | 7 | 15% | 99% | 58% | 72% |
| 12A | 2 | 15% | 79% | 81% | 62% |
| 12A | 4 | 78% | 99% | 56% | 61% |
| Rand | 6 | 15% | 77% | 69% | 62% |
| Rand | 7 | 40% | 71% | 61% | 62% |
| Rand | 9 | 12% | 75% | 62% | 60% |
| S1 | 2 | 53% | 97% | 63% | 82% |
| S1 | 5 | 45% | 97% | 99% | 98% |
| S2 | 4 | 61% | 98% | 98% | 91% |
| S2 | 5 | 67% | 100% | 99% | 100% |
| S3 | 2 | 3% | 81% | 32% | 35% |
| S3 | 10 | 5% | 69% | 60% | 49% |
| S5 | 4 | 90% | 99% | 91% | 53% |
| S5 | 7 | 18% | 98% | 74% | 56% |
| S5 | 8 | 68% | 98% | 99% | 74% |

**Table 7.2:** Proportions of runs completed without timing out for the self-repair strategies under comparison. Yellow cells mark situations where, for a given structure and failed robot, a particular strategy achieved $>$ 60% success, green cells indicate completion rates above 80%. Averages across all structures and failures tested give: nSSR 37%, nDSR 90%, SSR 72% and DSR 68%. This indicates that both the motion involved in Dynamic strategies and the ability to self-repair both provide reliability advantages over static self-assembly based nSSR. SSR's higher proportion of completed runs within the time limit suggests that DSR's speed advantage is traded against a reduced reliability.

of subslaves or as a measure of how large specific substructures below it are. Figs.(7.19) and (7.20) show task completion, and repair completion, times plotted against the number of robots below the failed robot in the hierarchy of a structure. Lines of best fit indicate how each strategy's time requirements scale with the amount of robots below the failure site. Figs.(7.21) and (7.22) explore the same trend but in terms of the largest single substructure below the failed module, rather than the total number of subslaves.

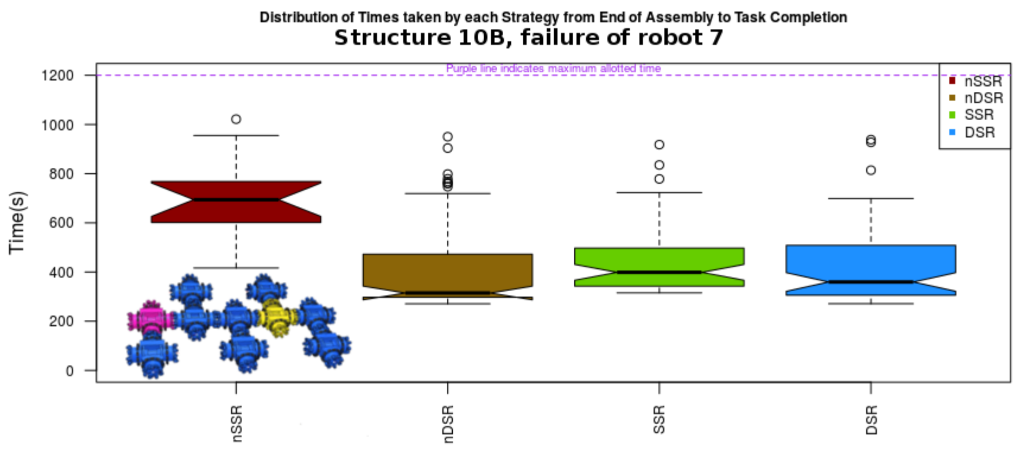


**Fig. 7.7:** A boxplot showing the times taken by different strategies to repair and complete the locomotion task after robot 7 in structure 10B experienced a failure. Robot 7 in this structure has a single slave attached to port 4 and a substructure of two slaves attached to port 1. nSSR proved slow to complete the task and the repair, with nDSR, SSR and DSR giving relatively similar results. nDSR proved fastest here, perhaps because re-recruiting for self-assembly did not take long for a total of only 3 subslaves. DSR and nDSR's advantage over nSSR may be explained by nSSR's disadvantage of staying in one place and waiting for recruits to come to the structure as compared to recruiting while moving through a group of wandering robots. SSR's advantage over nSSR reflects how use of self-repair reduces the need for recruitments of wandering robots, giving good performance despite being static. A potential reason that DSR does not enjoy a significant further advantage over nDSR and SSR, despite combining their advantages over nSSR, may be due to the risk of the returning MRS substructure getting lost in this strategy. The structure is shown for reference with the master robot marked in pink and the failed robot in yellow.

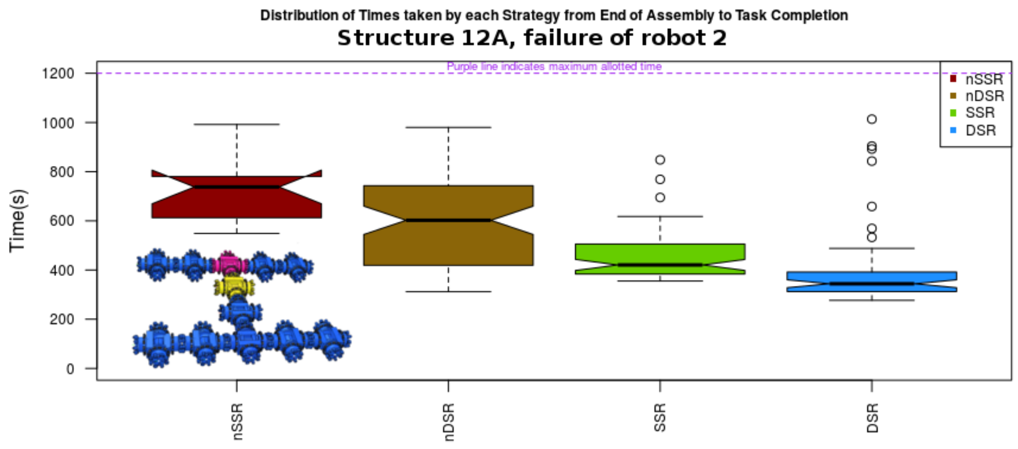


**Fig. 7.8:** Failures of robot 2 in structure 12A. Robot 2 in this structure has a single large substructure of 6 modules attached below port 4. In these circumstances the self-repair focused strategies allow for faster task completion than the breakup and reassemble methods, likely due to the significant amounts of extra time required for nSSR and nDSR strategies to do enough recruitment to rebuild the 7 module substructure. DSR out-competes SSR, but has quite a few outliers where substructures became lost and struggled to find the Replacement Module's recruitment cone.

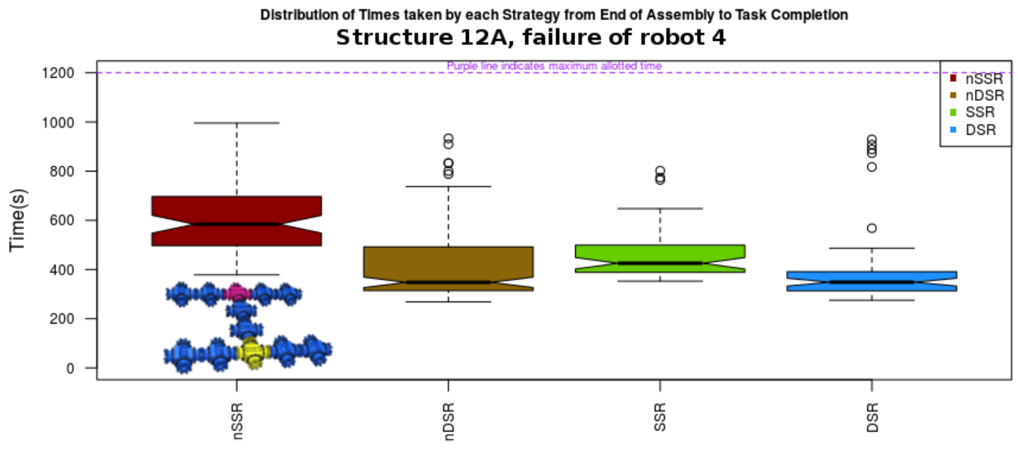


**Fig. 7.9:** A boxplot showing the times taken by different strategies to repair and complete the locomotion task after robot 4 in structure 12A experienced a failure. Having two substructures of 2 robots each attached to it the performance difference between DSR, SSR and nDSR when robot 4 fails is not as severe as for the failure of robot 2.

| Structure | ID of Failed Robot | nSSR vs nDSR | nSSR vs SSR | nSSR vs DSR | nDSR vs SSR | nDSR vs DSR | SSR vs DSR |
|---|---|---|---|---|---|---|---|
| 10B | 5 | 0.11 | 0.029 | 0.043 | 0.68 | 0.16 | 0.07 |
| 10B | 7 | 2.7e-6 | 3.6e-6 | 1.5e-6 | 5.0e-5 | 0.24 | 0.049 |
| 12A | 2 | 0.015 | 2.9e-8 | 1.8e-7 | 2.0e-5 | 9.6e-11 | 1.3e-8 |
| 12A | 4 | 1.0e-17 | 1.5e-9 | 5.2e-15 | 2.3e-6 | 0.52 | 2.8e-8 |
| Rand | 6 | 0.03 | 8.3e-8 | 1.1e-8 | 3.0e-6 | 6.0e-15 | 4.9e-15 |
| Rand | 7 | 0.41 | 2.1e-5 | 6.2e-5 | 0.19 | 0.11 | 0.014 |
| Rand | 9 | 3.5e-4 | 5.3e-6 | 2.5e-6 | 0.13 | 6.6e-5 | 9.1e-5 |
| S1 | 2 | 7.4e-15 | 9.1e-10 | 1.2e-15 | 3.6e-3 | 0.016 | 4.4e-8 |
| S1 | 5 | 2.2e-16 | 2.2e-8 | 2.0e-16 | 1.5e-16 | 0.26 | 6.9e-17 |
| S2 | 4 | 3.2e-15 | 4.0e-19 | 2.1e-26 | 3.1e-11 | 2.4e-5 | 9.4e-27 |
| S2 | 5 | 5.3e-23 | 7.4e-15 | 4.5e-23 | 1.4e-19 | 0.57 | 2.3e-19 |
| S3 | 2 | 0.04 | 8.7e-3 | 0.03 | 0.19 | 3.3e-3 | 2.9e-3 |
| S3 | 10 | 0.28 | 9.3e-4 | 9.2e-4 | 6.0e-5 | 9.7e-11 | 6.4e-10 |
| S5 | 4 | 4.2e-8 | 8.1e-14 | 1.4e-11 | 7.5e-4 | 0.64 | 3.6e-6 |
| S5 | 7 | 5.1e-8 | 1.9e-9 | 3.3e-8 | 0.086 | 0.33 | 1.1e-4 |
| S5 | 8 | 4.4e-11 | 2.0e-17 | 9.7e-17 | 0.038 | 0.27 | 1.8e-9 |

**Table 7.3:** p-values using Mann-Whitney tests for the null hypotheses, $H_0$7.2 and $H_0$7.4, that the strategies being compared give the same times for completion of the locomotion task during which self-repair occurs. Green cells mark those below the 5% threshold. Note that nDSR and DSR often give statistically similar performance, whereas pronounced differences between other pairs of strategies occur in more scenarios.

| Structure | ID of Failed Robot | nSSR vs nDSR | nSSR vs SSR | nSSR vs DSR | nDSR vs SSR | nDSR vs DSR | SSR vs DSR |
|---|---|---|---|---|---|---|---|
| 10B | 5 | 0.77 | 0.88 | 0.85 | 0.52 | 0.56 | 0.60 |
| 10B | 7 | 0.88 | 0.89 | 0.90 | 0.31 | 0.45 | 0.60 |
| 12A | 2 | 0.69 | 0.96 | 0.94 | 0.70 | 0.82 | 0.78 |
| 12A | 4 | 0.84 | 0.81 | 0.89 | 0.29 | 0.52 | 0.80 |
| Rand | 6 | 0.67 | 0.94 | 0.98 | 0.72 | 0.89 | 0.90 |
| Rand | 7 | 0.55 | 0.75 | 0.74 | 0.57 | 0.58 | 0.62 |
| Rand | 9 | 0.82 | 0.92 | 0.91 | 0.57 | 0.67 | 0.68 |
| S1 | 2 | 0.89 | 0.83 | 0.91 | 0.36 | 0.60 | 0.77 |
| S1 | 5 | 0.93 | 0.79 | 0.93 | 0.16 | 0.45 | 0.84 |
| S2 | 4 | 0.88 | 0.92 | 0.99 | 0.23 | 0.67 | 0.92 |
| S2 | 5 | 0.95 | 0.86 | 0.95 | 0.13 | 0.52 | 0.87 |
| S3 | 2 | 0.86 | 0.97 | 0.90 | 0.58 | 0.67 | 0.71 |
| S3 | 10 | 0.65 | 0.95 | 0.96 | 0.71 | 0.85 | 0.85 |
| S5 | 4 | 0.73 | 0.82 | 0.84 | 0.36 | 0.47 | 0.73 |
| S5 | 7 | 0.91 | 0.96 | 0.94 | 0.42 | 0.55 | 0.70 |
| S5 | 8 | 0.80 | 0.89 | 0.90 | 0.41 | 0.55 | 0.77 |

**Table 7.4:** A-test results showing effect sizes of the differences between distributions of total task completion times. For each column of Strategy A vs Strategy B results show the chance that A will take longer than B. Green cells mark those for which the p-value indicated a statistically significant difference in strategy performance. In all statistically significant examples nSSR is inferior, slower, than all other strategies. DSR and nDSR are often faster than SSR, with some exceptions where SSR outpaces nDSR. nDSR and DSR usually have similar performance, but where they are statistically significantly different DSR proves faster.

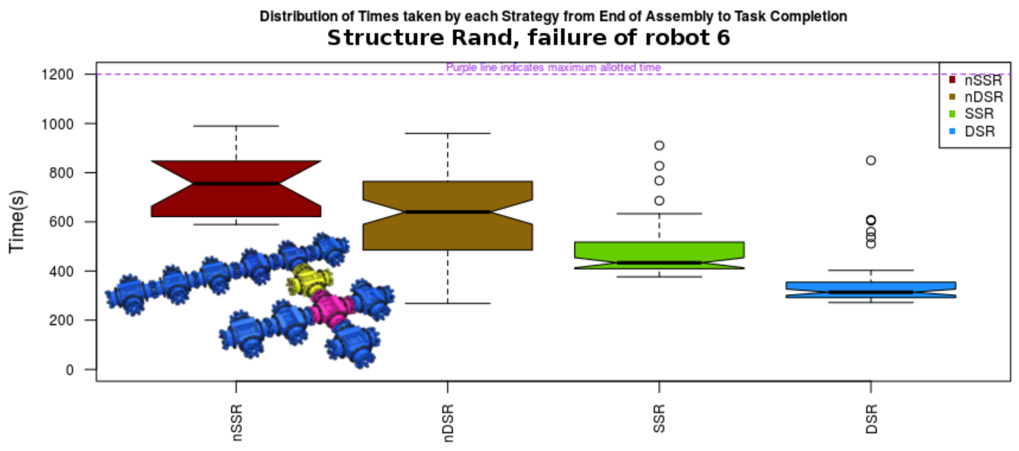


**Fig. 7.10:** Robot 6 in structure Rand experiences a failure. Robot 6 has a single large substructure of 6 robots attached. Due to this large substructure size nSSR and nDSR performed slowly, the large size of the rest of the Rand structure also meaning less robots were available in the arena to recruit for. SSR's performance was better but DSR enjoyed a clear lead. Although faster the self-repair strategies were less reliable than nDSR, leading to greater numbers of timed out runs, again highlighting the problem of substructures getting lost. This also shows how the loss of substructures tends to cause a proportion of runs to fail to complete within the time limit, but does not diminish performance in those cases where such navigational confusion does not occur.

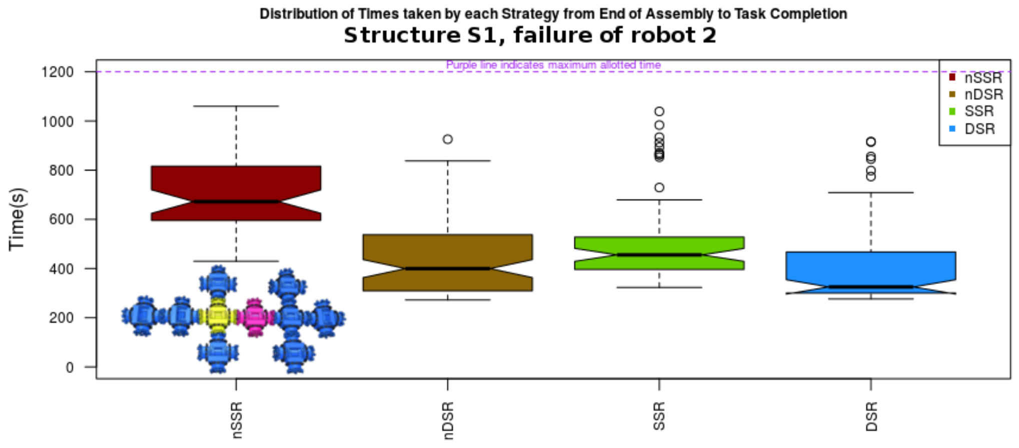


**Fig. 7.11:** Time distributions for the locomotion task after robot 2 in structure S1 experienced a failure. In this example a 2 module substructure acts as the MRS headed substructure responsible for removing the failed module, DSR enjoys a statistically significant advantage over nDSR, but the A-test score shows that DSR is only faster 60% of the time. This close competition is likely due to the fact that nDSR need only recruit a 2 module substructure to perform repair here, not much faster a procedure than removal and return by the MRS substructure. The superiority of nDSR and DSR over SSR show the advantage which motion brings.

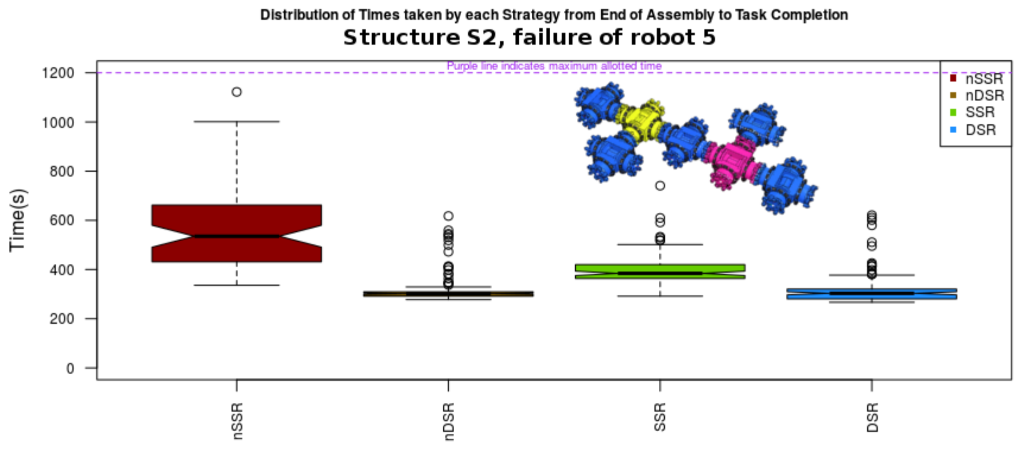


**Fig. 7.12:** Failures of robot 5 in structure S2. In this scenario Wi-Fi controlled retreat is used as the failed module has no multi-module substructures attached. DSR and nDSR therefore experience very similar performance as, in practice, they are doing almost identical things in this scenario. The only difference being that the Wi-Fi controlled retreat ensures that the failed module is placed further from the main structure than can be achieved by the main structure's retreat via "hallucination", see Sec.(7.3),under an nDSR strategy. This removal of the failed module before its slaves return to a "Wandering" state also explains why SSR outperforms nSSR, in nSSR the failed module remains close to the location of its former master and makes recruiting the Replacement Module more difficult.

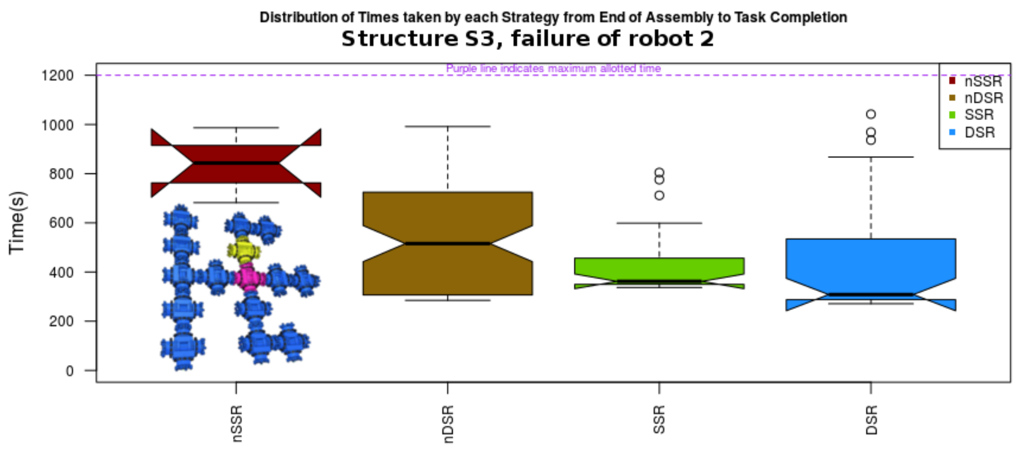


**Fig. 7.13:** Failures of robot 2 in structure S3. Robot 2 has only a single small, 2 module, substructure below it in the hierarchy, but the overall structure of S3 is large. This large structure reduces the number of wandering robots available in the environment to recruit for, and presents navigational challenges for the MRS headed substructure to overcome during the return towards docking. nDSR proved substantially more reliable, 81% success within the time-out limit, than other strategies here, it was however often slow. When other strategies did succeed in completing in time, DSR proved fastest, however suffered a long tail of slower results due to the complexities of navigating the MRS headed substructure back in to place. SSR, whilst slower on average and from the A-test, did not suffer this long tail to the distribution, indicating an easier navigational situation when the main substructure was not moving.

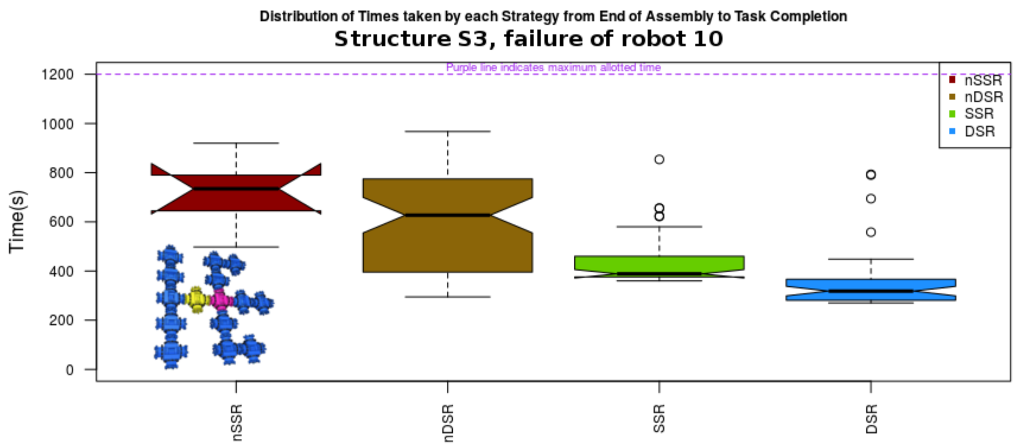


**Fig. 7.14:** A boxplot showing the time distributions for different strategies to repair and complete the locomotion task after robot 10 in structure S3 experienced a failure. Robot 10 has a single large substructure of 5 robots below it in the structure's hierarchy. Here self-repair strategies outperformed breakup and recruitment, with DSR's performance now strongly superior to SSR's. The position at which the Replacement Module is recruited in this structure is more exposed to the outside world than the location of robot 2, potentially making this scenario an easier navigational challenge for the DSR strategy than the failure of robot 2 in this structure had been.

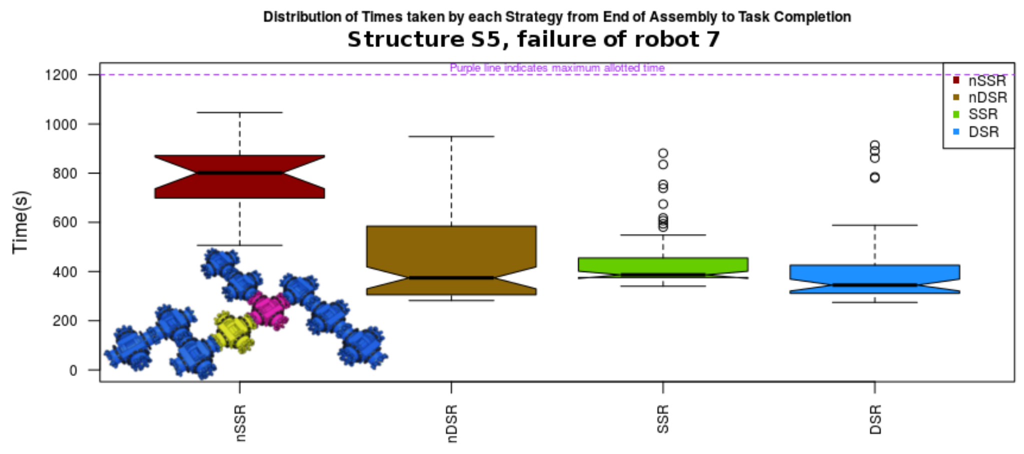


**Fig. 7.15:** A boxplot showing the times taken by different strategies to repair and complete the locomotion task after robot 7 in structure S5 experienced a failure. Robot 7 has a single substructure docked to it, consisting of 3 modules. This 3 module structure is responsible for a long tail in the times taken for nDSR to perform re-recruitment, although overall performance differences between nDSR and DSR are statistically insignificant here.

| Structure | ID of Failed Robot | nSSR vs nDSR | nSSR vs SSR | nSSR vs DSR | nDSR vs SSR | nDSR vs DSR | SSR vs DSR |
|---|---|---|---|---|---|---|---|
| 10B | 5 | 0.27 | 0.097 | 0.11 | 3.3e-3 | 4.8e-3 | 0.60 |
| 10B | 7 | 4.6e-3 | 7.3e-3 | 9.3e-3 | 6.5e-3 | 0.80 | 0.99 |
| 12A | 2 | 0.41 | 2.1e-7 | 5.5e-4 | 1.2e-19 | 1.3e-9 | 7.2e-6 |
| 12A | 4 | 0.31 | 2.1e-9 | 0.51 | 5.6e-11 | 0.21 | 4.3e-10 |
| Rand | 6 | 0.99 | 2.0e-7 | 4.3e-6 | 9.9e-20 | 4.5e-14 | 1.7e-5 |
| Rand | 7 | 1.6e-3 | 0.054 | 0.16 | 1.4e-6 | 0.041 | 4.3e-4 |
| Rand | 9 | 0.38 | 9.2e-6 | 2.4e-3 | 6.7e-16 | 1.9e-8 | 4.6e-10 |
| S1 | 2 | 2.7e-5 | 1.6e-7 | 4.8e-8 | 9.1e-3 | 2.8e-3 | 0.39 |
| S1 | 5 | 9.2e-7 | 1.0e-8 | 1.2e-5 | 0.074 | 0.32 | 5.9e-3 |
| S2 | 4 | 0.039 | 4.1e-17 | 4.9e-7 | 5.3e-22 | 4.4e-6 | 7.2e-17 |
| S2 | 5 | 7.1e-4 | 3.27e-8 | 1.1e-4 | 1.5e-3 | 0.54 | 0.011 |
| S3 | 2 | 0.10 | 8.7e-3 | 0.10 | 2.0e-10 | 3.0e-3 | 1.7e-4 |
| S3 | 10 | 0.63 | 6.5e-4 | 3.8e-3 | 1.4e-18 | 5.5e-12 | 4.8e-7 |
| S5 | 4 | 6.8e-4 | 7.4e-8 | 0.16 | 7.9e-15 | 0.13 | 2.4e-8 |
| S5 | 7 | 3.6e-4 | 2.9e-9 | 1.1e-5 | 2.2e-17 | 1.4e-5 | 3.8e-7 |
| S5 | 8 | 0.52 | 8.9e-17 | 7.4e-3 | 3.5e-19 | 9.6e-3 | 9.9e-8 |

**Table 7.5:** p-values using Mann-Whitney tests for the null hypotheses, $H_0 7.3$ and $H_0 7.5$, that the strategies being compared give the same times for the self-repair operation itself to complete, regardless of the time required for the locomotion task to finish. Green cells mark those below the 5% threshold.

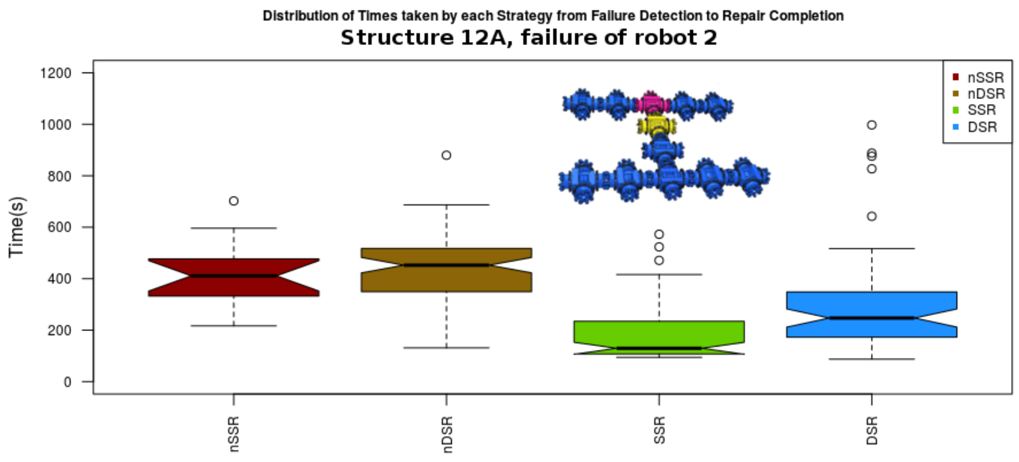


**Fig. 7.16:** Boxplot showing the distribution of times taken for the repair operation, from detection of the failed module, robot 2, until the rebuilding of the structure, 12A, has completed. For the large substructure connected to robot 2 in structure 12A we see the repair times of the self-repair, rather than naive, strategies tending to be faster. Static Self-repair proves the fastest to repair, with DSR taking longer due to the complexity of navigating substructures back in to place when the main structure is in motion. The speed advantage enjoyed by SSR for repair times does not equate to an overall task time advantage due to DSR's ability to parallelise locomotion task completion and repair.

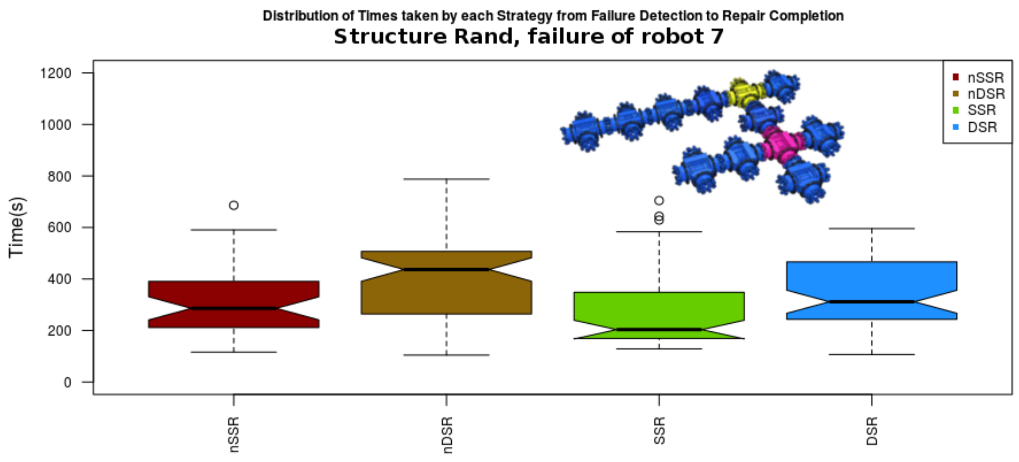


**Fig. 7.17:** Repair times for the failure of robot 7 in the Rand structure. SSR's advantage over the naive strategies is likely due to the rarity with which free wandering robots pass in to relevant regions to be recruited in the naive strategies. The SSR strategy reduces these complications by only needing to make one recruitment of a wandering robot. When comparing the effect of dynamic repair versus static we see, for this structure and failure scenario, that in both naive and repair based strategies that dynamic strategies repair more slowly. As this occurs for naive as well as self-repair based strategies it seems plausible that, for this structure, being in motion may also slow the docking operations involved in nDSR as well as causing the navigational complexities which delay DSR.

| Structure | ID of Failed Robot | nSSR vs nDSR | nSSR vs SSR | nSSR vs DSR | nDSR vs SSR | nDSR vs DSR | SSR vs DSR |
|---|---|---|---|---|---|---|---|
| 10B | 5 | 0.69 | 0.79 | 0.77 | 0.65 | 0.63 | 0.47 |
| 10B | 7 | 0.73 | 0.73 | 0.71 | 0.52 | 0.51 | 0.49 |
| 12A | 2 | 0.43 | 0.93 | 0.79 | 0.92 | 0.80 | 0.28 |
| 12A | 4 | 0.54 | 0.81 | 0.47 | 0.79 | 0.45 | 0.16 |
| Rand | 6 | 0.50 | 0.93 | 0.88 | 0.94 | 0.87 | 0.28 |
| Rand | 7 | 0.32 | 0.61 | 0.42 | 0.74 | 0.60 | 0.31 |
| Rand | 9 | 0.58 | 0.91 | 0.77 | 0.90 | 0.74 | 0.22 |
| S1 | 2 | 0.71 | 0.78 | 0.78 | 0.62 | 0.63 | 0.54 |
| S1 | 5 | 0.76 | 0.80 | 0.73 | 0.57 | 0.46 | 0.39 |
| S2 | 4 | 0.60 | 0.90 | 0.73 | 0.90 | 0.68 | 0.16 |
| S2 | 5 | 0.65 | 0.75 | 0.67 | 0.63 | 0.52 | 0.39 |
| S3 | 2 | 0.78 | 0.97 | 0.79 | 0.89 | 0.67 | 0.23 |
| S3 | 10 | 0.43 | 0.96 | 0.90 | 0.95 | 0.87 | 0.21 |
| S5 | 4 | 0.36 | 0.73 | 0.43 | 0.83 | 0.57 | 0.22 |
| S5 | 7 | 0.77 | 0.95 | 0.85 | 0.88 | 0.71 | 0.24 |
| S5 | 8 | 0.53 | 0.88 | 0.63 | 0.87 | 0.62 | 0.26 |

**Table 7.6:** A-tests comparing the times taken for the self-repair operation. Amongst the naive strategies nDSR proves faster in most of the significantly different scenarios, although 2 scenarios of statistical significance favour nSSR. However when comparing the self-repairing methods SSR proves faster to repair than DSR. Self-repair based strategies prove superior to breakup based strategies throughout.

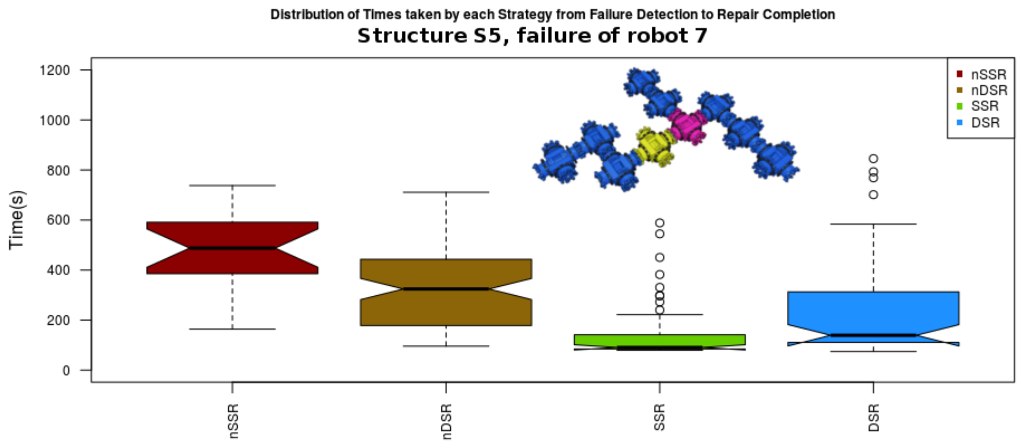


**Fig. 7.18:** Repair times for the failure of robot 7 in the S5 structure. SSR proves extremely effective here with all other strategies lagging behind, naive breakup based methods prove slow. Among naive strategies dynamic features prove useful to accelerate the repair process. Again DSR's time to complete the repair operation is increased above SSR's due to the time taken for the substructures to navigate back to dock with the moving main structure.

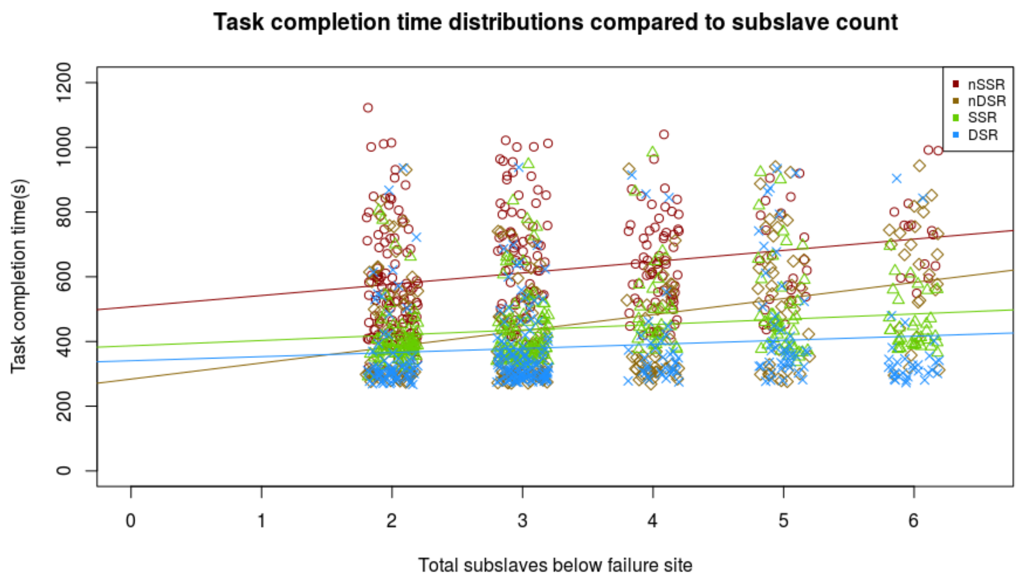


**Fig. 7.19:** Datapoints from all structure and failure scenarios, excepting those which timed out before completion, showing how the performance of each of the four strategies varies with the number of subslaves docked below the failed module in the structure's hierarchy. The naive breakup based strategies exhibit stronger Pearson correlation coefficients than the self-repairing strategies, for nSSR, nDSR, SSR and DSR respectively these are 0.25, 0.34, 0.19 and 0.11. For graphical clarity a random sample of datapoints have been hidden and left-right jitter has been applied, lines of best fit are still derived from full unjittered datasets.

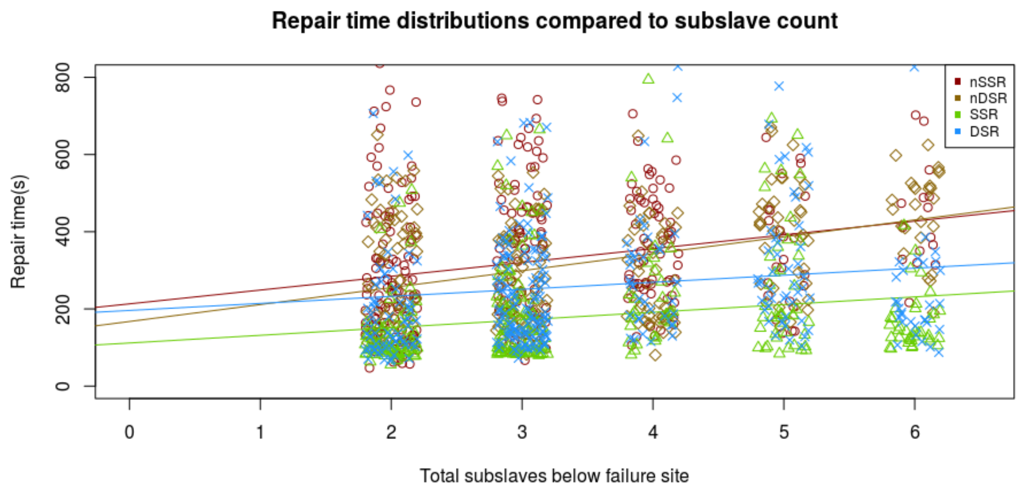


**Fig. 7.20:** Any correlations found between times to complete the repair operation and the total number of modules below a failure site vary according to the strategy used. Pearson correlation coefficients are, by commonly suggested classifications [194], “weak" for nSSR (0.25),nDSR(0.36) and SSR(0.21) and “very weak" for DSR(0.14).

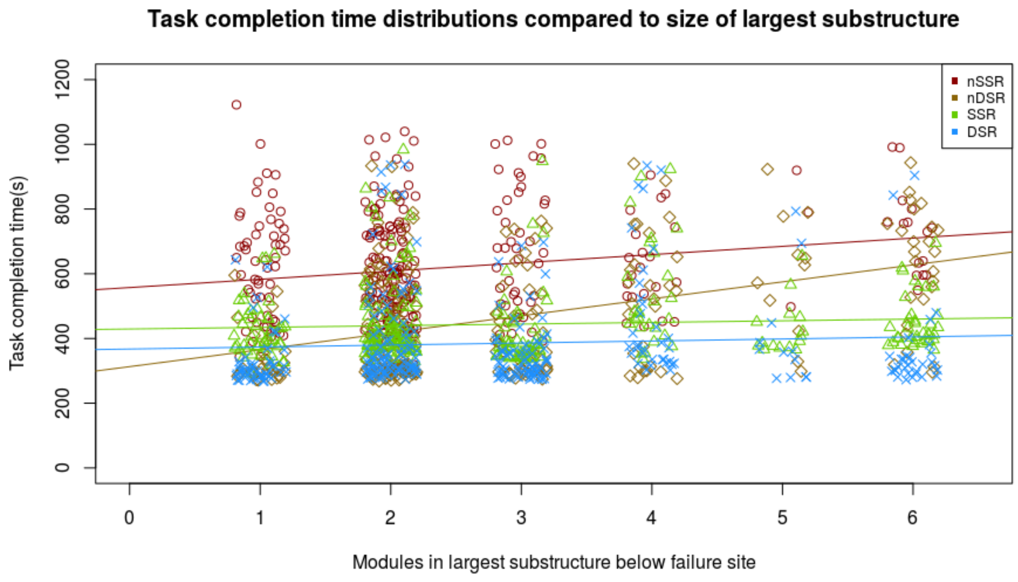


**Fig. 7.21:** Comparing completion times for the locomotion task, during which the self-repair operation occurred, to the size of the largest single substructure below the failed module shows stronger correlations, 0.18 and 0.4 for nSSR and nDSR respectively, for naive strategies than for self-repair based strategies. The self-repair strategies plausibly appear uncorrelated to the size of the largest substructure involved in the self-repair operation, suggesting them to be highly scalable.

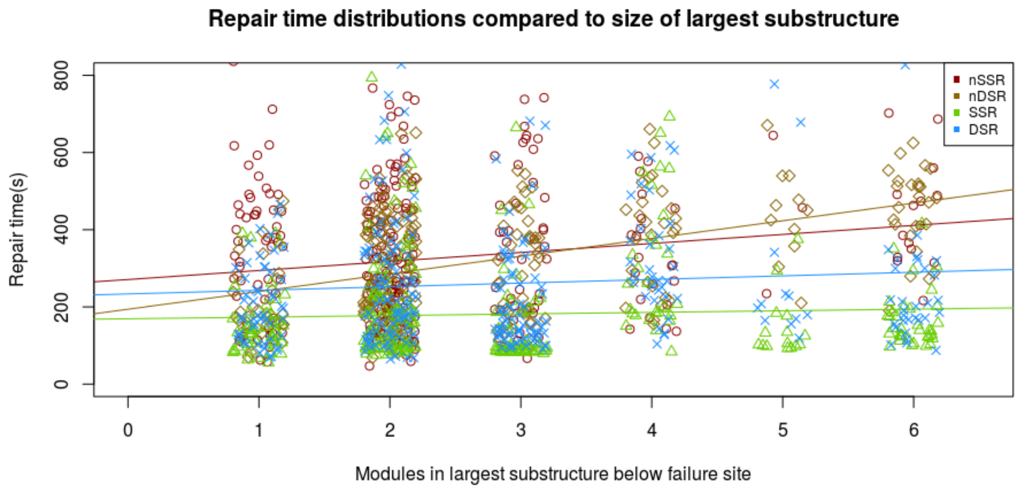


**Fig. 7.22:** Comparing the repair times to the size of the largest single substructure below the failure site separates out self-repairing from naive strategies more strongly. Pearson coefficients for SSR (0.051) and DSR (0.082) are weaker than the 0.17 seen for nSSR or the “moderate" correlation seen for nDSR. The size of the largest substructure may give more noticeable differences between strategies than the total subslave count due to the way in which a large substructure is likely to require multiple “layers" of recruitment to form it whereas total subslave count can reflect modules in different branches of the structure which can all be replaced in parallel rather than at different sequential, despite MLR’s best efforts, “layers" in the hierarchy.

## 7.8 Discussion

The results of the self-repair experiments can now be considered against some of this chapter's null hypotheses:

- Comparison of the proportion of runs which completed without timing out gives some indication of the reliability of a particular self-repair strategy, although statistical analysis in a binary situation of this kind can be difficult, the fact that the nSSR strategy gave low proportions of non-timed out runs across all scenarios suggest it to be unreliable. nDSR's high proportion of successful runs across all scenarios indicates reliability. The two substructure-based self-repair strategies displayed levels of reliability which varied across scenarios. $H_0$7.1 can therefore be plausibly rejected, although it would appear that in reliability terms nDSR is the optimum strategy. The cause behind these reliability differences appears to be that nSSR requires the re-recruitment of a comparatively large number of modules, and must wait for all of them to enter in to its recruitment cone, for the time-out periods used in these experiments the chance of enough modules wandering in to the right region in time can be quite low. nDSR's reliability seems related to its simplicity, it recruits for any wandering modules it can, and does so while moving to increase the chance of encountering them. nDSR does not depend on more complex processes such as requiring substructures containing specific modules to navigate between particular locations. While an argument can also be made about nDSR's effectiveness being related to the large pool of potential recruitable wandering robots which it generates, this performance benefit is not observed in nSSR which also generates a large pool of wandering modules to recruit from. nDSR has many concepts in common with the methods used in [166] these reliability advantages may explain why a strategy of that kind was considered appropriate for recovery of modular robots after destructive impact events. The reliability of self-repair based strategies, both dynamic and static, is inbetween those nSSR and nDSR, with the failures often related to the substructures involved becoming lost. As the Replacement Module is recruiting specifically for these substructures the repair cannot complete if they do not wander in to the IR recruitment cone(s). This highlights that there may a trade-off between speed and reliability, with strategies which manage the greatest speed, such as DSR, having reduced reliability compared to somewhat slower strategies like nDSR. The main lesson here appears to be that for industrially useful self-repair, with a near 100% completion rate within some timeframe, more work must be done on extending the range of guidance and finding solutions to aid in navigation over distance. Subject to the condition of not relying on explicit GPS style location data this is an interesting problem for sensor development and search pattern driving algorithms.
- The ability to repair while in motion, both for nDSR and DSR, gave statistically significant

performance differences, with p-values below the 5% threshold, in most scenarios. A-test results largely above 0.8 in statistically significant scenarios show nDSR outperformed nSSR, and A-test results above 0.7 for a vast majority of scenarios also show DSR's speed advantage over SSR. $H_0$7.2 can be confidently rejected, showing dynamic strategies regularly outperform their equivalent static strategies. Group motion tasks have been shown to be able to proceed faster overall if self-repair can take place while the motion proceeds. Performing both the repair and motion tasks in parallel rather than serial can be considered as one of the causes behind this effect. This provides a good indication of the assistance that Dynamic Self-repair methods can provide for scenarios where overall task time is the key measurement, such as reducing accumulated radiation doses or minimising the time which modules must spend causing extra drag on a ship's hull, see Sec.(2.5).

- The advantage provided by strategies able to operate in motion does little to accelerate the repair procedure itself, despite the speed advantages which it provides for the overall task being attempted. Whilst DSR and nDSR mostly outperformed nSSR the SSR strategy proved most effective in terms of time for the repair itself to complete. $H_0$7.3 can be rejected, but it appears that, in terms of the repair time, having the ability to repair during continuous motion is useful for breakup based strategies but disadvantageous for strategies performing substructure-based repair. A likely cause for the advantages of dynamic strategies over nSSR may be that remaining in motion strongly increases the probability of a wandering robot entering the recruitment cone of the Master to the Failed Module. Differences, measured in seconds, between different strategies times to perform the repair operation itself tend to be considerably smaller than the differences in task completion times, showing that the differences in repair times are not the only cause for the differences in task completion times. One of the considerations behind the Dynamic Self-repair concept was the ability which Dynamic Self-repair could provide to minimise the risks of robots being further harmed during a repair procedure, for example by falling debris. In the light of this finding we must make our judgement on this matter as a weighing of whether preserving the ability to move is more important for avoiding damage, which would still favour DSR which also completes tasks fastest, or whether making the repair procedure itself as fast as possible is the critical element, perhaps if a robot was in use for a masonry shoring [190] task, in which case SSR would be more appropriate.

- Naive breakup based strategies were outperformed in terms of task completion times by substructure-based self-repair in some of the scenarios. Across static strategies SSR performed faster than nSSR by a statistically significant amount for all combinations of structure and failed module, A-test scores for the effect size were all high with many above 0.90. For static strategies, therefore, there is very strong cause for rejection of

$H_0$7.4. This would appear to agree with Murray's findings [51] in which his self-repair strategies operating on the SYMBRION platform found a similar advantage for strategies able to repair without a complete breakup. When comparing between dynamic strategies the situation is less clear with statistically significant differences only occurring in 7 of the 16 scenarios. In those scenarios where statistically significant differences were found all favoured the substructure-based DSR over the breakup-based nDSR, with A-test scores ranging from 0.6 to 0.89. The scenarios in which there was a statistically significant difference between nDSR and DSR do not seem to be marked by any common factor, such as size of the overall structure or of substructures, or the count of subslaves below the failed module. DSR would in these regards seem to be definitely *not inferior* to nDSR, but potentially superior only in limited circumstances and therefore perhaps not strongly enough to entirely reject $H_0$7.4 for dynamic strategies. It is possible here that the extremely high reliability of nDSR makes it difficult for DSR, a less reliable strategy, to outperform it, improving sensors or search patterns may change this. DSR may also display less advantage over nDSR than SSR does over nSSR due to the increased probability of substructures involved in the repair getting lost whilst attempting to relocate the main structure, a scenario less likely to happen when the main structure remains static.

- When comparing breakup and re-assembly strategies against substructure self-repair strategies in terms of the time taken for the repair itself, rather than the overall task completion time, differences are still strongly visible. A statistically significant difference is found between naive breakup and substructure repairing methods in a vast majority of scenarios. $H_0$7.5 can be conclusively rejected. A-test scores for scenarios where a significant difference is found show an effect size favouring substructure strategies in all cases, 76% of scenarios have A-test scores above 0.70 with half of scenarios being above 0.80. This matches [51]'s findings and shows the same findings to be applicable to dynamic self-repair strategies as well as static ones. The differences are more pronounced between static strategies than between dynamic ones.

- When considering task completion times this chapter's experiments have found, for substructure based self-repair strategies, no strong evidence of correlations between either the total number of subslaves below a failed module in a hierarchy, or the number of modules in the largest single substructure formed during self-repair by the failed module's slaves. Both static and dynamic self-repair methods allow for task completion times which do not appear to scale up with structure size by either the subslave count or largest substructure measure. Task completion times do however rise, with Pearson correlation coefficients in the range of 0.18 to 0.40 for breakup based strategies. nDSR has the strongest correlation coefficient, being strongest when compared against the size of the largest substructure rather than the total subslave count. This suggests that naive breakup based strategies are less scalable than substructure preserving self-repair, that the size of the largest substructure has a stronger effect than the total subslave count also indicates that an increased number of “layers" on which the re-assembly must take place may be of importance here. With these further layers requiring, despite the Multi-layered Recruitment methods used, extra time due to the need of wandering modules to perform rotation and handshaking procedures before they can themselves begin recruiting. $H_0$7.6 can be rejected, breakup based strategies lose performance, relative to substructure preserving ones, as the number of modules and size of substructures below the repair site rises. The relative performance of DSR against SSR stays constant with structure sizing, showing that the beneficial effect of being able to self-repair during motion remains roughly constant with structure size. This finding indicates that while nDSR has done remarkably well in many of the tests this advantage could swiftly be lost at even moderately larger structures, at which time the difficulties in recruiting extra robots, especially in an environment with few free wandering robots present, could easily outweigh the difficulties of navigating substructures back to dock, even for moving structures.

- Times required for self-repair procedures scale, to some extent, with the size of structures, more strongly when measures by counting total subslaves rather than the size of the largest individual substructure. Graphs produced show a positive correlation between repair time and subslave count for all strategies, these correlation lines rise faster with subslave count for naive strategies than repair-based ones. It is, however, entirely possible that the weaker correlations among these are simply statistical artefacts where minimal correlation actually exists. When considering the size of the largest substructure below the failure site correlations are found for the naive strategies but not for either of the self-repair strategies showing self-repair to be more scalable then breakup and repair strategies. $H_0$7.7 can be rejected with a recognition that the substructure based strategies maintain relatively constant performance with structure size, below the failure site, whereas naive breakup strategies rise, somewhat, in time requirements. This agrees with Murray’s findings [51] on the scaling of repair times with structure size for breakup strategies but relatively constant performance for substructure strategies, it also shows that this generalises to continuous motion repair strategies as well as static ones. The slower performance, for the repair although not task times, for DSR as compared to SSR remains fairly constant with structure size.

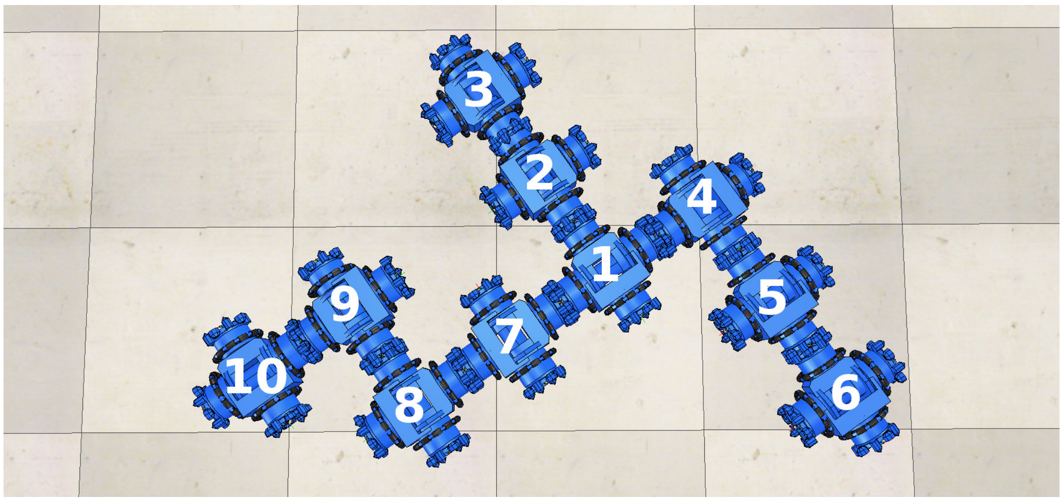
At this point the general advantages of the basic DSR Dynamic Self-repair strategy should be clear, although some limitations have also been found. Other points worthy of note are: the importance of removing failed modules and depositing them a substantial distance away, even in the absence of other differences between the operation of different strategies, see Fig.(7.12); the complexities involved in navigating substructures back to dock, more pronounced for MRS substructures which drive away and turn than for MAS substructures which simply make a linear retreat; and a recognition of the fact that for each strategy beyond a certain time taken for the repair process very few outliers exist which still complete within the time limit. The next step, therefore, is to see whether DSR can be optimised by the addition of extra features, as well as modifying DSR's algorithms so as to enable it to handle failures of a structure's highest level master module. This chapter now extends the Dynamic Self-repair strategy and tests it in a wider range of scenarios, before considering operating both the previously discussed DSR strategy and the extended strategy on hardware robots, see Sec.(7.13), and then summarising the findings of both the earlier and later parts of this chapter in Sec.(7.14).

## 7.9 Dynamic Master Switching

So far the failures considered have ignored failures of the global master unit, the Dynamic Self-repair strategy is now adapted so that this can be handled, as well as making some optimisations to the strategy's performance based on the results of earlier experiments. The new features act not only to make the Dynamic Self-repair strategy resilient to failures of any module, hence enabling maximum exploitation of the modular nature of modular robots, but also can be considered as a form of neuroplasticity. If the global master of a structure fails, effectively the loss of its brain, other parts of the organism can promote themselves to this role and replicate the lost master's functionality. The need to be able to handle failures of the master robot has been recognised in previous work [51] [26] and implemented, however the quadruplet data structures used in this work provide an interesting new basis from which to develop methods of replacing the master module.

We start by considering, again, the recruitment list quadruplet array data structures used to define morphologies. In these lists each docked connection is specified in terms of which robot will be master within that connection, which will be the slave, and which port each will use. Each robot processes these lists in such a way as to recruit for those robots which are to be its slaves, and each robot also uses the lists to recognise which of its ports forms its connection towards its immediate master, to which it should relay Mergeable Nervous Systems sensory information and receive Mergeable Nervous Systems actuator commands. However whilst this quadruplet array acts as a form of directed graph, it also contains all the information necessary to plot out alternative hierarchies of mastery able to represent the same structure. It is therefore possible to remap the location of the master within the organism, dynamically switching it between different robots while preserving connections and Temporary ID numbers.

Remapping occurs immediately after a failure is detected in a neighbouring module, and as described with the earlier self-repair process, is handled independently by each module neighbouring the failed module, with the expectation that all will, as they have the same understanding of the structure and the failed module's position within it, reach common conclusions on how to proceed with repair. The process, see Fig.(7.24) for pseudocode, takes a copy of the current Structure Recruitment List, which represents the structural plan regardless of the current physical state, and the Recruitment List, modified in real-time to reflect docking and undocking events and stores them in other variables. The Structure Recruitment List is then remapped, see Fig.(7.26) for the remapping algorithm, to be centred, temporarily, on the failed robot. The Recruitment List is updated by remapping as necessary to match the new Structure Recruitment List. With this temporary remapping in place all other modules in the organism can be treated as slaves to the failed module, and sub-slave counting methods can compare the size of the structures connected to each of the failed module's ports. Decisions can then be made on whether to transfer global mastery away from the current global master, and in to part of the largest structure attached to the failed module, or revert to the copies taken of the unedited quadruplet arrays. A key point to note about the algorithms presented here is that, when used correctly, they cannot convert a physically feasible structure in to an impossible one during the process of remapping it. Such a transfer of global mastery is ofcourse essential if the global master is the failed module.



{{1,1,4,4},{1,3,4,7},{1,4,1,2},{2,3,3,3},{4,3,4,5},{5,2,4,6},{7,2,4,8},{8,3,3,9},{9,4,1,10}}
{**{4,4,1,1}**,{1,3,4,7},{1,4,1,2},{2,3,3,3},{4,3,4,5},{5,2,4,6},{7,2,4,8},{8,3,3,9},{9,4,1,10}}
{{1,1,4,4},**{7,4,3,1}**,{1,4,1,2},{2,3,3,3},{4,3,4,5},{5,2,4,6},**{8,4,2,7}**,{8,3,3,9},{9,4,1,10}}

**Fig. 7.23:** Three Structure Recruitment Lists for the same structure but with the global master located at either Robot 1, Robot 4 or Robot 8, notice which quadruplet brackets have been modified and how the elements within them have reversed in order. The structure in this example is Structure 5 from the experiments, the first mapping shows the structure as defined ordinarily, the second mapping may occur in the recovery after Robot 5 suffered a failure and the third as a step after a failure of Robot 9. Note that the order in which quadruplets are presented does not matter for this or any other uses of the data structure.

After the remapping process is complete modified Structure Recruitment Lists and Recruitment Lists are shared over Wi-Fi in a specialised “atomic" operation. This is necessary so that other modules within the organism, aside from those directly connected to the failed module which calculated the remapping for themselves, have the correct information necessary to supply Mergeable Nervous Systems sensor data in the reference frame of their new master. An “atomic" operation is used to prevent chaos which could ensue if modules received updates to the Structure Recruitment List and Recruitment List at different times and spent the intervening time with an updated copy of one but an obsolete copy of the other. As the remapping simply changes the order of terms inside specific quadruplets within the Structure Recruitment List and Recruitment List arrays, see Fig.(7.23) for an example, no robot will have experienced any disruptive change such as a modification of Temporary ID value, and no undocking or redocking is required for the master switched structure to resume MNS activity.

As well as being able to recover from a failure of the global master this Dynamic Master Switching also provides an opportunity to allocate roles to the substructures around the failed module while taking in to account the lessons of the earlier self-repair experiments. Unlike with the earlier DSR strategy, if the largest group of modules connected to a failed module is a slave of the failed module the largest substructure can be placed in-charge of the re-recruitment part of self-repair and can contain the MFM module. Mergeable Nervous Systems over Wi-Fi methods can be prioritised where possible to allow the disposal of failed modules to be handled by single modules which can be replaced individually, decreasing the need for large substructures to be guided back to dock after retreating and turning to dispose of a failed module. Where possible, if sufficient smaller substructures exist, larger substructures can be prioritised for MAS roles rather than MRS, meaning they will typically spend less time wandering away from the main structure and should be easier to guide back to dock once a Replacement Module is in place. However if there are not sufficient individual modules connected to a failed module for a Wi-Fi controlled retreat, and no smaller substructures available to act as MRS either, then the Dynamic Self-repair with Dynamic Master Switching strategy will allocate any available substructure to play the vital role of removing the failed module.

Further optimisations also added to the Dynamic Master Switching strategy included timeouts after which it would switch back to an nDSR strategy, with the length of timeout being scaled, depending on the subslave count below the failure site, to match the time after which less than 5% of successful DSR repair operations would complete. An extra addition was a number of modifications to the “orbit" direction calculations used in long range Infrared guidance of substructures back in to range to dock, these modifications prioritised an MAS or MRS robot’s sensory readings above those of its connected slaves and subslaves as a means to ensure that guidance data was less likely to prove misleading in situations where slaves at the extremities of a returning substructure may overshoot an area and receive guidance signals recommending a reversal of direction even though the substructure master had not yet entered the line-of-sight IR communications range.

```
if(NoIRMessagesSeenInLast6SecOnPort[PortNum]==TRUE and DockingPort[PortNum]==DOCKED and TimeNow-PortDockedAtTime[PortNum]>6)then
  RemappingIsNeeded=0
  if(CheckStructRecListForThisTempIDInPlaceFour(FindHighestLevelMaster(MyTempID))==0)then --this only applies to robots in the substructure containing the global master, we can't allow master switching in other substructures in the case of multiple failures
    GlobalMasterTempID=FindHighestLevelMaster(MyTempID) --gather this info now, before remapping
    FailedNeighbourTempID=PortXDockedTempID[PortNum]
    FailedNeighboursPortToItsMasterOld=FindPortToMaster(FailedNeighbourTempID
    WasFailedNeighbourNotGlobalMaster=CheckStructRecListForThisTempIDInPlaceFour(FailedNeighbourTempID)
    OldDMSStructureRecruitmentList=NewCloneCopyOfArray(StructureRecruitmentList)--make copies of the two lists before doing anything else
    OldDMSRecruitmentList=NewCloneCopyOfArray(RecruitmentList) --old lists kept in reserve incase we wish to undo remapping
    RemapSRL(FailedNeighbourTempID)--SRL centred on new robot
    RemapRL()--RL remapped to match SRL
    --now we have our RecList remapped we can make scans of subslave counts from each port of the failed module
    SubslaveCountArray={0,0,0,0}
    DeadMastersSlaveTempIDs={0,0,0,0}
    index=1
    while(index <= 4)do
      if(RobotHasPortDocked(FailedNeighbourTempID,index)==1)then
        DeadMastersSlaveTempIDs[index]=FindTempIDofSlave(FailedNeighbourTempID, index)
        SubslaveCountArray[index]=CountSubslaves(DeadMastersSlaveTempIDs[index])--count how many slaves it had
      end
      index=index+1
    end
    PortOfHighestSubSlaCount=FindPortWithMostSubslaves()--if the failed module has multiple ports with an equal highest subslave count on them then we end up select the higher port numbered subslave group as the largest one
    --if the failed neighbour was not originally global master and the subslave counts say that the port of the failed slave which used to be masterward has the largest substruct on it
    if(WasFailedNeighbourNotGlobalMaster~=0 and FailedNeighboursPortToItsMasterOld==PortOfHighestSubSlaCount)then
      StructureRecruitmentList= NewCloneCopyOfArray(OldDMSStructureRecruitmentList)--we return to the old mapping
      RecruitmentList= NewCloneCopyOfArray(OldDMSRecruitmentList)
    else --either the failed neighbour was global master, or one of its ports on what used to be its slave sides has the biggest substruct attached
      RemappingIsNeeded=1
      --note down the temp ID of the robot heading the biggest substruct attached to the failed neighbour
      BigSubstructMasterID=FindTempIDofSlave(FailedNeighbourTempID,PortOfHighestSubSlaCount)
      RemapSRL(BigSubstructMasterID)--now make this robot global master
      RemapRL()
      UpdateVariablesDerivedFromRLorSRL()
      if(CheckStructRecListForThisTempIDInPlaceFour(MyTempID)==0)then
        BecomeGlobalMaster()
      else
        CeaseGlobalMasteryIfPreviouslyMaster()
      end
    end
  end
  --slave side decision making below here works so as to prioritise MNS over Wi-Fi where possible, and otherwise when possible make the largest substructs in to MAS not MRS
  if(BrainRobot==0 and PortNum==PortToOurMaster)then
    SubslaveCountArray={0,0,0,0}
    DeadMasterSlavesAttachedArray={0,0,0,0}
    DeadMastersSlaveTempIDs={0,0,0,0}
    index=1
    while(index <= 4)do
      if(index ~= DeadMastersPortToMaster and RobotHasPortDocked(DeadMastersTempID,index)==1)then --by only being true if the thing on this port is a slave we ensure we don't miscount the dead master's own master's structure as a substructure
        DeadMasterSlavesAttachedArray[index]=1--if so find what that robot's TempID was
        DeadMastersSlaveTempIDs[index]=FindTempIDofSlave(DeadMastersTempID, index)--count how many slaves it had, put this number in the indexth place of SubslaveCountArray
        SubslaveCountArray[index]=CountSubslaves(DeadMastersSlaveTempIDs[index])
      end
      index=index+1
    end
    MySubslaveCount=CountSubslaves(MyTempID)
    TempIDsOfSubstructsEqualToOurSize={0,0,0,0}
    DeadMastersSingleSlaveTempIDs={0,0,0,0}
    WiFiMNSPossible=0
    SizableSubstructuresCount=0
    LowestSubslaveCount=1000 --higher than any realistic value
    index=1
    while(index <= 4)do
      if(DeadMasterSlavesAttachedArray[index]==1 and SubslaveCountArray[index]==MySubslaveCount)then
        TempIDsOfSubstructsEqualToOurSize[index]=DeadMastersSlaveTempIDs[index]
      end
      if(DeadMasterSlavesAttachedArray[index]==1 and SubslaveCountArray[index]==0)then
        DeadMastersSingleSlaveTempIDs[index]=DeadMastersSlaveTempIDs[index]
      end
      if(DeadMasterSlavesAttachedArray[index]==1 and SubslaveCountArray[index]==0)then
        WiFiMNSPossible=WiFiMNSPossible+1
      end
      if(DeadMasterSlavesAttachedArray[index]==1 and SubslaveCountArray[index]>=1)then
        SizableSubstructuresCount=SizableSubstructuresCount+1
      end
      if(DeadMasterSlavesAttachedArray[index]==1 and SubslaveCountArray[index]<LowestSubslaveCount)then
        LowestSubslaveCount=SubslaveCountArray[index]
      end
      index=index+1
    end
```

**Fig. 7.24:** Pseudocode (1/2) of the algorithm for allocating self-repair roles through use of the Dynamic Master Switching method, see main text for details. See Fig.(7.25) for continuation.

```
    if(WiFiMNSPossible>=2)then--if Wi-Fi over MNS is possible, that is if there are 2 or more substructs of only 1 robot each (with 0 subslaves)
      --we must use something unique about the docked single robots to decide which will be in command of the retreat and dump, as against which will just follow along
      RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)
      if(MySubslaveCount==0 and MyTempID>=max(DeadMastersSingleSlaveTempIDs))then--TempID is unique, so we use the highest of them (for values of them only considering those slaves which are alone, and not masters of large substructs) as "boss" of the retreat, we use >= not <= because if a zero got in to DeadMastersSlaveTempIDs[] due to an undocked port on the dead master then that 0 would mess up a <= check by falsely winning over all the real slaves
        BeginWifiControlledRetreatAsMaster()
      elseif(MySubslaveCount==0)then--my TempID is not the highest one among attached slaves with no subslaves
        BeginWifiControlledRetreatAsSlave()
      else --if we have a subslave count of 1 or more we will be an MAS (non main substructure) instead
        UndockOnPort(PortNum)
        RetreatDirection=FindRetreatDirection(PortToOurMaster)--retreat directly away from failed module
        BecomeMasterOfAnotherSubstructure()
      end
    else--if MNS cannot be done, too many large substructures or not enough small ones
      if(MySubslaveCount>=1)then
        if(SizableSubstructuresCount>=2)then--if there are >1 substructs with >=1 subslaves each
          if(MySubslaveCount==LowestSubslaveCount and OccurrencesOf(LowestSubslaveCount)in(SubslaveCountArray)==1)then--if we are the smallest substruct, no others of the same size
            RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)
            BecomeMasterOfRemovalSubstructure()
          elseif(MySubslaveCount==LowestSubslaveCount)then--if we are equal smallest with 1 or more of the other substructs
            if(MyTempID>=max(TempIDsOfSubstructsEqualToOurSize))then--if our TempID is largest amongst them
              RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)
              BecomeMasterOfRemovalSubstructure()
            else
              UndockOnPort(PortNum)
              RetreatDirection=FindRetreatDirection(PortToOurMaster)--retreat directly away from failed module
              BecomeMasterOfAnotherSubstructure()
            end
          else--if other substructs exist which are >1 subslave and smaller than us
            UndockOnPort(PortNum)
            RetreatDirection=FindRetreatDirection(PortToOurMaster)--retreat directly away from failed module
            BecomeMasterOfAnotherSubstructure()
          end
        else --we are the only substruct
          RetreatDirection=FindRetreatDirection(DeadMastersPortToMaster)
          BecomeMasterOfRemovalSubstructure()
        end
      else
        RetreatDirection=FindRetreatDirection(PortToOurMaster)
        BecomeLoneModule()
      end

    end
  elseif(PortNum~=PortToOurMaster)then
    UndockOnPort(PortNum)
    BecomeMasterToTheFailedModule()
  end
  if(RemappingIsNeeded==1)then
    ShareOverWifiAtomicMessages(StructureRecruitmentList+RecruitmentList)
  end
end
```

**Fig. 7.25:** Pseudocode (2/2) of the algorithm for allocating self-repair roles through use of the Dynamic Master Switching method, see main text for details.

```
function RemapSRL(TempIDOfNewMaster)--modifies the SRL to set a new robot to be master, re-orders directions of docking accordingly without changing any Temp ID values
  if(CheckStructRecListForThisTempIDInPlaceFour(TempIDOfNewMaster)==0)then --the input temp ID is already global master, there is nothing to do
    --SRL need not be remapped
  else
    local CopySRL=NewCloneCopyOfArray(StructureRecruitmentList)--we make a copy of the SRL, we'll keep one copy normal, edit the other, work from the normal one, then replace the normal with edited right at the end, this way we do all the workings with a consistent SRL
    --for each TempID, starting with TempIDOfNewMaster, we search the SRL for quads in which it is specified as slave. For any quad where a robot is specified as slave we flip that quad unless the quad has an already processed robot as the master
    local RobotNotYetProcessed={1,1,... ,1}--create an array for holding processed robots
    --create a queue of robots still to be dealt with
    local RobotsQueue=RemapQueueSetUp()--starts off empty
    local IDtoCheck=TempIDOfNewMaster
    local validity=0
    local error2=0 --goes to 1 if serious error occurs
        local NewMasterDoneYet=0
    --start iterating
    while((NewMasterDoneYet==0 or RemapQueueGetLength(RobotsQueue)~=0) and error2==0)do
            if(NewMasterDoneYet==0)then--on first run
        IDtoCheck=TempIDOfNewMaster
        validity=1
        NewMasterDoneYet=1
      else
        validity,IDtoCheck=RemapQueueReadFrom(RobotsQueue)--first item taken from queue
      end

      if(validity==1)then --else the queue is empty
        local indexE=1
        while(indexE<= table.getn(StructureRecruitmentList))do
          local StructureInstruction=StructureRecruitmentList[indexE]
          if(StructureInstruction[4]==IDtoCheck and RobotNotYetProcessed[StructureInstruction[1]]==1)then
            CopySRL[indexE]={StructureInstruction[4],StructureInstruction[3],StructureInstruction[2],StructureInstruction[1]}
            RobotNotYetProcessed[IDtoCheck]=0
            error2=RemapQueueAddTo(RobotsQueue,StructureInstruction[1])--ID of master as specified by old SRL added to last place in the queue
          end
          indexE=indexE+1
        end
      end
    end
  StructureRecruitmentList=NewCloneCopyOfArray(CopySRL)
  end
end
```

**Fig. 7.26:** The algorithm used to remap the Structure Recruitment List quadruplet array so as to place a chosen module in the global master position, see main text for details. See also Fig.(7.27) for pseudocode describing the updating of the editable Recruitment List.

```
function RemapRL()--takes existing RL and swaps round terms as and where necessary to make them consistent with the latest SRL, that is to say the SRL currently in use
  --we take each term in the RL and compare against terms in the SRL, altering term order in the RL where necessary
  local indexD=1
  while(indexD<= table.getn(RecruitmentList))do
    if(indexD<1)then --just incase funny stuff happens due to the table.remove
      indexD=1
    end
    local Instruction=RecruitmentList[indexD]
    -- Instruction represents an RL term, now we cycle through the SRL to find it
    local indexE=1
    while(indexE<= table.getn(StructureRecruitmentList))do
      local StructureInstruction=StructureRecruitmentList[indexE]
      if(StructureInstruction[1]==Instruction[1] and StructureInstruction[2]==Instruction[2] and StructureInstruction[3]==Instruction[3] and StructureInstruction[4]==Instruction[4])then --if SRL and RL contain an identical element
        --leave it alone, don't edit
      elseif(StructureInstruction[1]==Instruction[4] and StructureInstruction[2]==Instruction[3] and StructureInstruction[3]==Instruction[2] and StructureInstruction[4]==Instruction[1])then --if there is an SRL giving the reversed equivalent of what is in the RL
        --reverse the RL element to become like the SRL one
        local RevdInstruction={Instruction[4],Instruction[3],Instruction[2],Instruction[1]}
        RecruitmentList[indexD]=RevdInstruction
        --as we've inserted exactly where we removed from we need not worry about fiddling the indexD counter
      else
        --remove any RL element which neither matches nor mirrors an SRL element
        table.remove(RecruitmentList,indexD)
        indexD=indexD-1 --ensures that after the quad is removed from the RL we come back around using the same indexD value and don't skip over any quads
      end
      indexE=indexE+1
    end
    indexD=indexD+1
  end
  --the RL is now remapped
end
```

**Fig. 7.27:** The algorithm used to remap the Recruitment List quadruplet array so as to match a newly remapped Structure Recruitment List.

## 7.10 Dynamic Master Switching Results

100 randomised runs of Dynamic Self-repair with Dynamic Master Switching were performed for each scenario used in the earlier self-repair experiments, as well as for scenarios in which the global master of each structure experienced a failure. The arena maintained the same geometry as before (7 m width, 10m from starting point to finishing line, 30 robots present), the locomotion task setup was the same and identical time logging code was used.

Tables (7.7), (7.8) and (7.9) give, respectively, the proportions of completed runs for DSR with DMS in each scenario and, for those scenarios which other strategies could handle, the statistical differences and effect sizes found when comparing DSR with DMS to other strategies. Figs.(7.28),(7.29),(7.30) and (7.31) show boxplots comparing the performance of DSR with DMS to other strategies in particularly interesting scenarios. As failures of the global master could not be handled by other self-repair or naive strategies the Dynamic Master Switching runs in which the global master failed are summarised separately in Fig.(7.32).

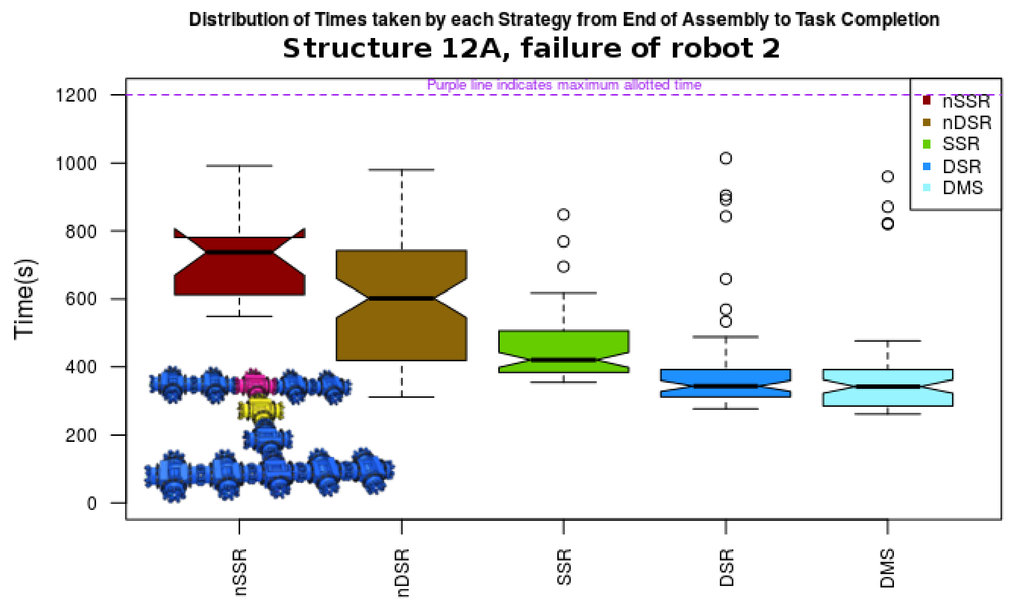


**Fig. 7.28:** A boxplot showing the task completion times when Robot 2 fails in structure 12A, Dynamic Master Switching allows the larger part of the structure to remain and has the row of 5 robots, containing the original master, act as the Removal substructure. This alteration has negligible effect on task completion times, the challenge here still remains that of navigating a large substructure back in to place.

A video of an example DMS self-repair scenario is available at [222] and in the supplementary material (Example_DMS_scenario.mp4).

| Structure | Temp. ID of Failed Robot | DMS |
|---|---|---|
| 10B | 5 | 81% |
| 10B | 7 | 87% |
| 12A | 2 | 72% |
| 12A | 4 | 54% |
| Rand | 6 | 52% |
| Rand | 7 | 51% |
| Rand | 9 | 64% |
| S1 | 2 | 94% |
| S1 | 5 | 97% |
| S2 | 4 | 82% |
| S2 | 5 | 99% |
| S3 | 2 | 68% |
| S3 | 10 | 69% |
| S5 | 4 | 78% |
| S5 | 7 | 81% |
| S5 | 8 | 85% |
| 10B | 1 | 91% |
| 12A | 1 | 53% |
| Rand | 1 | 79% |
| S1 | 1 | 79% |
| S2 | 1 | 100% |
| S3 | 1 | 26% |
| S5 | 1 | 52% |

**Table 7.7:** The proportion of non-timed-out runs for the Dynamic Master Switching enabled Dynamic Self-repair strategy across the different scenarios used. Averaged across all scenarios a 74% completion rate is found, this rises to 76% if averaged only across those scenarios which were also used for testing the other strategies. As well as a higher averaged completion percentage than the standard DSR strategy, DSR with DMS manages a completion rate above 80% in 43% of scenarios to standard DSR's 25%.

| Structure | ID of Failed Robot | nSSR vs DMS | nDSR vs DMS | SSR vs DMS | DSR vs DMS |
|---|---|---|---|---|---|
| 10B | 5 | 0.074 | 0.46 | 0.10 | 7.8e-3 |
| 10B | 7 | 6.9e-5 | 4.0e-4 | 1 | 0.039 |
| 12A | 2 | 3.7e-8 | 2.6e-13 | 9.1e-11 | 0.13 |
| 12A | 4 | 6.3e-11 | 0.45 | 1.8e-5 | 0.32 |
| Rand | 6 | 4.6e-6 | 1.8e-8 | 5.4e-6 | 0.91 |
| Rand | 7 | 0.010 | 0.13 | 0.095 | 0.89 |
| Rand | 9 | 5.7e-6 | 4.9e-4 | 9.2e-6 | 0.88 |
| S1 | 2 | 9.4e-15 | 0.091 | 1.3e-6 | 0.47 |
| S1 | 5 | 2.2e-18 | 0.56 | 7.0e-21 | 0.65 |
| S2 | 4 | 1.1e-17 | 1.6e-7 | 2.2e-7 | 5.4e-18 |
| S2 | 5 | 5.2e-23 | 4.2e-9 | 4.2e-23 | 5.5e-5 |
| S3 | 2 | 8.9e-3 | 0.53 | 0.23 | 5.6e-3 |
| S3 | 10 | 1.3e-3 | 1.7e-12 | 6.1e-12 | 0.66 |
| S5 | 4 | 1.4e-20 | 1.7e-4 | 3.9e-19 | 3.3e-6 |
| S5 | 7 | 5.2e-9 | 2.3e-4 | 2.8e-12 | 4.3e-3 |
| S5 | 8 | 1.3e-19 | 0.54 | 8.7e-16 | 0.71 |

**Table 7.8:** p-values using Mann-Whitney tests for a null hypotheses that DSR with DMS's performance would be the same as that of other strategies hence giving similar task completion times. DSR with DMS proves statistically significantly different to nSSR in almost all scenarios, whilst differing from DSR and nDSR in fewer scenarios.

| Structure | ID of Failed Robot | nSSR vs DMS | nDSR vs DMS | SSR vs DMS | DSR vs DMS |
|---|---|---|---|---|---|
| 10B | 5 | 0.80 | 0.47 | 0.41 | 0.37 |
| 10B | 7 | 0.82 | 0.34 | 0.50 | 0.40 |
| 12A | 2 | 0.95 | 0.84 | 0.81 | 0.57 |
| 12A | 4 | 0.83 | 0.47 | 0.74 | 0.45 |
| Rand | 6 | 0.89 | 0.79 | 0.74 | 0.49 |
| Rand | 7 | 0.70 | 0.58 | 0.59 | 0.51 |
| Rand | 9 | 0.91 | 0.67 | 0.73 | 0.49 |
| S1 | 2 | 0.88 | 0.57 | 0.72 | 0.48 |
| S1 | 5 | 0.95 | 0.47 | 0.89 | 0.52 |
| S2 | 4 | 0.92 | 0.27 | 0.72 | 0.13 |
| S2 | 5 | 0.95 | 0.74 | 0.91 | 0.67 |
| S3 | 2 | 0.95 | 0.52 | 0.42 | 0.33 |
| S3 | 10 | 0.93 | 0.85 | 0.85 | 0.52 |
| S5 | 4 | 0.92 | 0.66 | 0.89 | 0.74 |
| S5 | 7 | 0.94 | 0.66 | 0.83 | 0.64 |
| S5 | 8 | 0.92 | 0.52 | 0.84 | 0.48 |

**Table 7.9:** A-test results comparing the task completion times from the Dynamic Self-repair with Dynamic Master Switching strategy to those of previously tested strategies. Amongst the scenarios with significant statistical differences DSR with DMS proves faster than nSSR in all cases and faster than SSR in all cases. It is faster than nDSR for a majority of scenarios and slower than standard DSR for almost all scenarios.

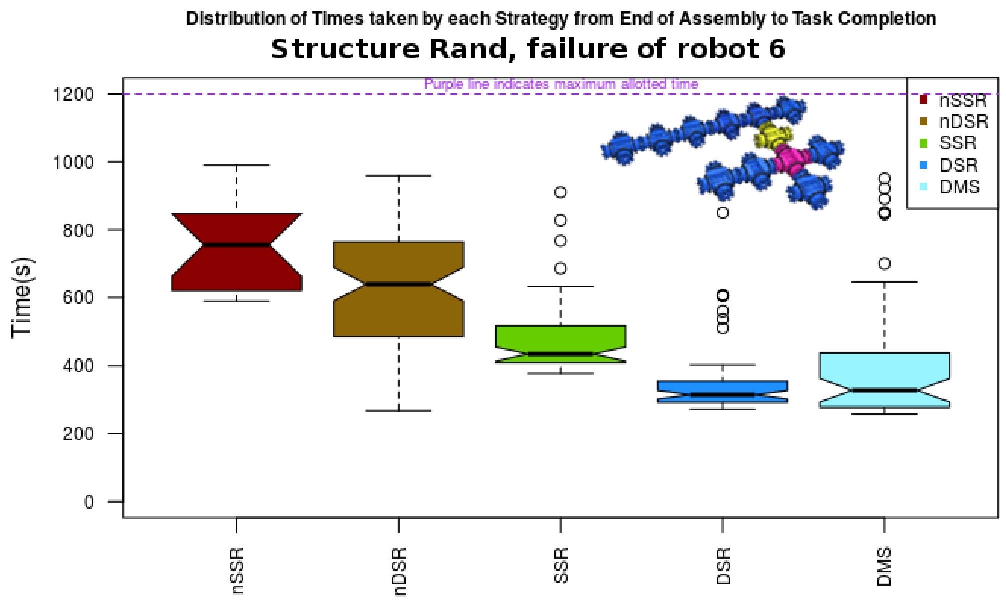


**Fig. 7.29:** When robot 6 fails in structure Rand Dynamic Master Switching promotes robot 7 to become the new brain, 7 ordinarily being the immediate slave of 6. This remapping of the structure means a 5 robot substructure takes on the role of Removal substructure. Whilst some visual difference exists between the distributions for standard DSR and DMS enabled, again the problem is that of navigating a large substructure back in to place. Whether this is a 5 robot cross shaped substructure, as is the case in DMS enabled DSR, or a 6 robot linear substructure, as is the case for standard DSR, makes little difference to task completion times.

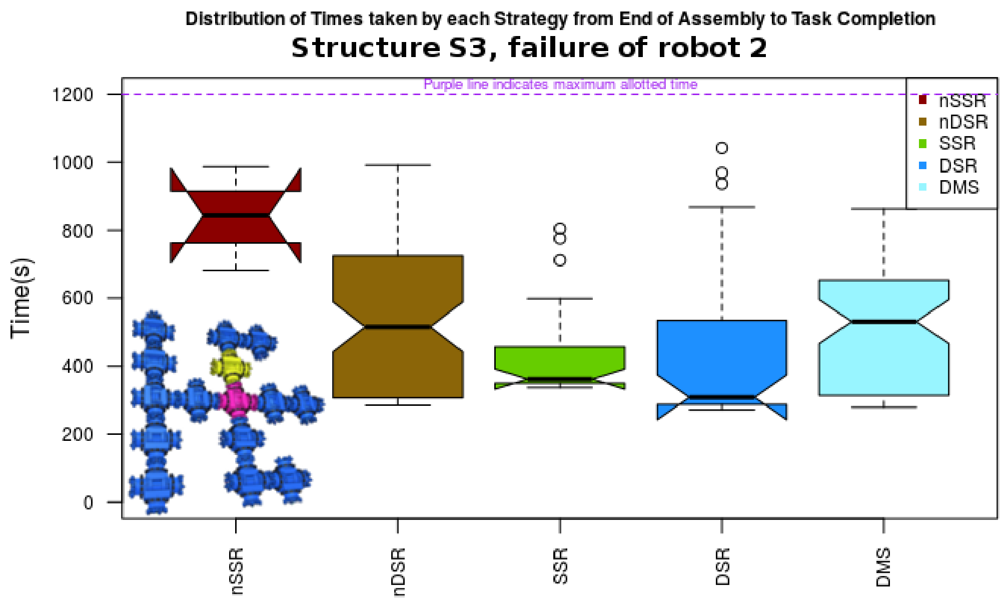


**Fig. 7.30:** The failure of robot 2 within structure S3 proved difficult for DSR due to the challenges of navigating a 2 module substructure back in to place while the main structure moved. The Dynamic Master Switching strategy's features include time limits dependent on substructure size, after which nDSR methods can be resorted to. This allowed for a higher proportion of runs to be completed without timing out, however caused the DSR with DMS strategy to be slower than those standard DSR runs which did succeed.

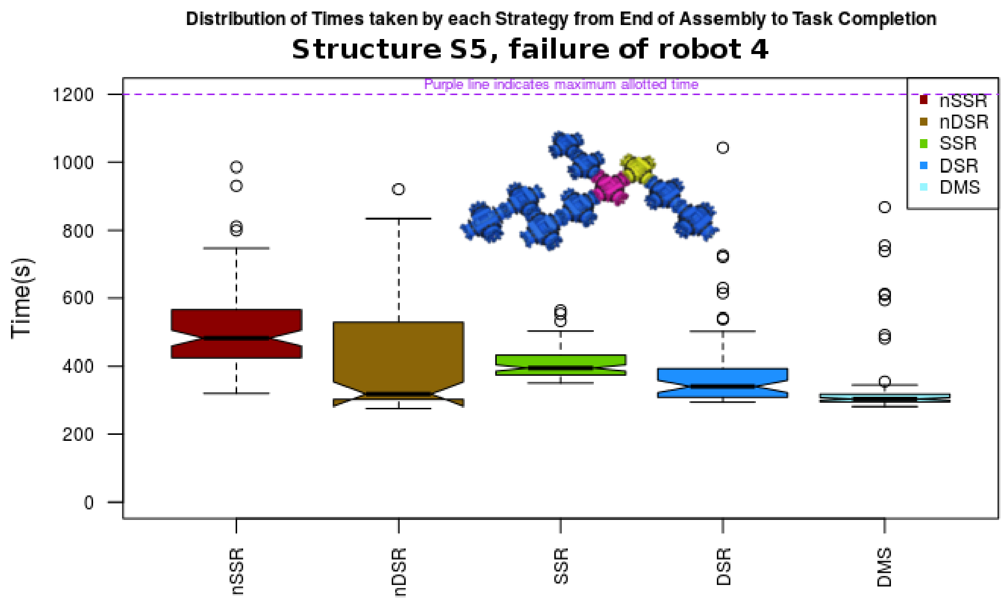


**Fig. 7.31:** The failure of robot 4 in structure S5 is one of the rare scenarios in which Dynamic Master Switching enabled Dynamic Self-repair had a significant speed advantage over standard DSR. This is another scenario in which the breakup of a 2 module substructure and the use of nDSR recruitment as an alternative can occur after a time limit, the shape of structure S5 appears to complicate the return of substructures hence allowing the nDSR features in DMS enabled DSR to make the difference here.

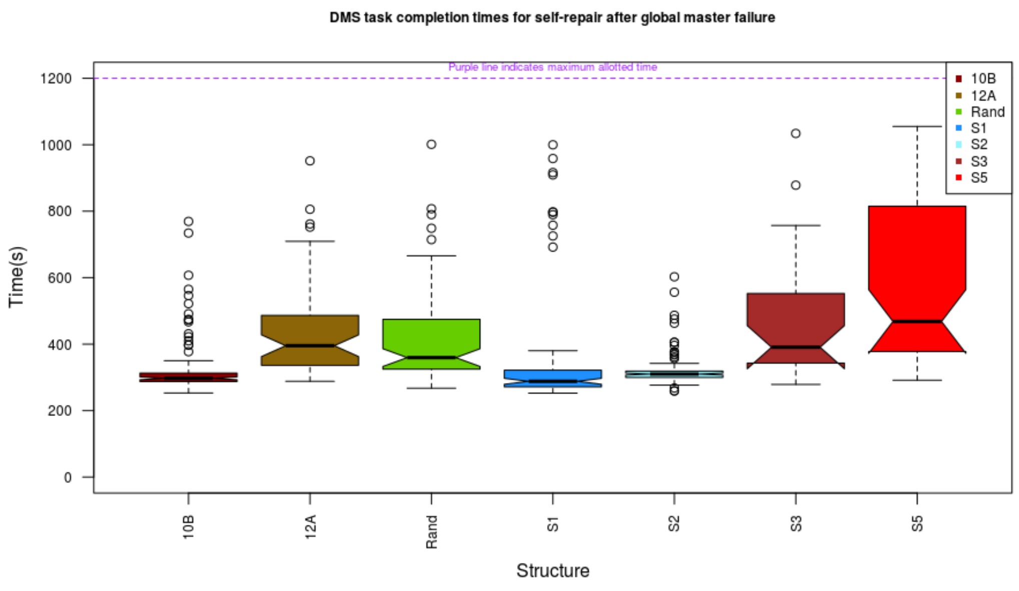


**Fig. 7.32:** A boxplot showing the distributions of completion times for the DMS strategy across scenarios in which the global master robot, the seed for the original structure, failed. Those which complete quickly are those for which MNS over Wi-Fi methods are used to remove the failed master or only a small substructure is assigned to the Removal role. For structure S1 the simplicity of the substructure which is responsible for disposing of the failed module and then returning may allow this substructure to avoid getting caught up around other parts of the main structure and therefore allow quick recovery without a long navigation sequence. Reasonably quick recovery and task completion is also achieved for the structures in which a large substructure is promoted to the MFM and another large substructure acts as the Removal substructure. S3 and S5 have particularly slow long tails to their distributions due to the complications which can occur when many substructures are in motion.

## 7.11 Dynamic Master Switching Discussion

The results of the Dynamic Master Switching experiments can now be considered in the light of the relevant null hypothesis, $H_0$7.8.

- Dynamic Master Switching typically gives similar performance to ordinary Dynamic Self-repair, and amongst those scenarios where a statistical difference exists standard DSR is generally quicker. A speed benefit cannot therefore be found for the Dynamic Master Switching strategy. However the capabilities which this strategy provides are not without other positive effects. Dynamic Master Switching, especially by being able to resort back to the slower but more dependable nDSR re-recruitment strategy, allows greater proportions of runs to complete before the simulation time-out threshold. Dynamic Master Switching also proves vital for being able to repair at all when it is the global master robot of a structure which suffers a failure. For these two reasons $H_0$7.8 can be rejected. Furthermore the size and form of the structures used in these experiments may not have tested all the capabilities which Dynamic Master Switching enabled Dynamic Self-repair can offer, DMS's preferences for assigning larger substructures to MFM roles and MAS roles, before taking MRS roles could prove a great advantage in larger and more heavily branched structures, especially perhaps once loop containing structures are involved.

## 7.12 Multiple Failures

To further test the Dynamic Master Switching strategy a small number of scenarios were run where multiple robots in the structure failed, earlier self-repair work, among those studies which considered failures within specific modules and the need to remove those modules [51] [26], has considered only the failure of single robots. All forms of modular robotic self-repair will have a boundary condition on the number of modules which, should they require replacement, will be too great to recover from. For Dynamic Self-repair with Dynamic Master Switching, in theory, that boundary condition would be that there are enough wandering modules in the environment to replace all modules which have failed, and that at least one module within the initial organism does not fail, and hence can become the new master. The Dynamic Self-repair strategy was however designed with the intention of being able to handle multiple failures at once, subject to certain conditions.

Whilst Dynamic Master Switching can be used for the first failure to be detected in many situations it cannot necessarily be used on the second failure detected, if both failures infact occur simultaneously then it will be a matter of chance which is detected first or second. Because the global master of a structure is remapped, usually to be placed in to the MFM module involved in a repair operation, with two failures there could be a risk of multiple modules attempting to become global master and chaos would ensue. To protect against this possibility DSR with DMS was designed so that only robots which were in the main substructure, that is to say robots which are in the same structure as the global master, were able to engage in master switching. Robots within other substructures can still engage in self-repair but were not intended to be able to remap structural hierarchies, for such second failures to be detected in an organism only modules which are slaves to the failed module within the hierarchy can be assigned to be Masters of the Removal (MRS) Substructure or Masters of Another (MAS) Substructure and only the immediate master of the failed robot, as according to the hierarchy, can take the Master to the Failed Module (MFM) role.

A series of tests injected failures in to pairs of modules within a variety of different structures, some tests considered failures of modules in widely separated parts of the structure, others considered failures within the same branch in a structure's hierarchy such that substructures which were acting as MRS or MAS substructures to a failure occurring "above" them also contained modules acting as MFM to failures taking place "below" them. Results from these tests are summarised in a qualitative sense, as this is a test of capabilities and limitations there is no comparison of performance against other strategies, particularly as they would not be able to handle all of the scenarios considered. 100 replicates of scenario were performed for the failures of specific pairs of robots in structures S3, 12A and Rand, see Fig.(7.33). For the two robots which failed one was injected with a failure 5 seconds after the other had been. Table.(7.10) summarises the key statistics from these tests.

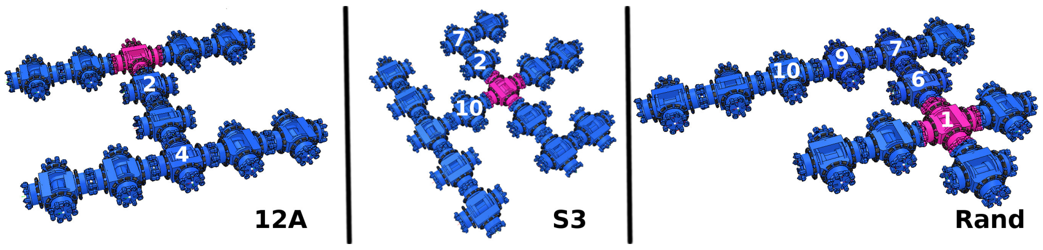


**Fig. 7.33:** Diagrams showing the robots used in multiple failure tests. On the 12A structure 2 and 4 were used as a failing pair, the second number in the pair indicating which failed 5 seconds after the other. For S3 failures were injected for 10 and 7, and 2 and 10. On the Rand structure: 6 and 9, 9 and 6, 6 and 10, 7 and 1, and 6 and 7 were all injected with failures.

### 7.12.1 Repairs in Separate Branches

For tests of the former kind with failures of widely separated modules within a structure two scenarios were tested: the failure of robots 10 then 7 in the S3 structure, and the failure of robots 2 then 10 in the S3 structure. Although the proportions of repairs completed within the simulation time-out limit were relatively low, repair proceeded in the usual fashion. Timed out runs among these two scenarios were due to the difficulties of navigating substructures back in to place and the relative scarcity of free wandering robots left in the environment delaying the

| Structure | Temp. IDs of Failed Robots | Proportion Completed within Time Limit | Median Time (s) | Mean Time (s) | Standard Deviation (s) | Minimum Time (s) | Maximum Time (s) |
|---|---|---|---|---|---|---|---|
| S3 | 10 and 7 | 25% | 419 | 516 | 218 | 278 | 910 |
| S3 | 2 and 10 | 40% | 568 | 577 | 216 | 282 | 998 |
| 12A | 2 and 4 | 34% | 583 | 620 | 218 | 315 | 1060 |
| Rand | 6 and 10 | 29% | 383 | 448 | 180 | 277 | 931 |
| Rand | 7 and 1 | 0% | NA | NA | NA | NA | NA |
| Rand | 6 and 9 | 35% | 467 | 520 | 207 | 269 | 996 |
| Rand | 9 and 6 | 35% | 383 | 471 | 203 | 262 | 979 |
| Rand | 6 and 7 | 0% | NA | NA | NA | NA | NA |

**Table 7.10:** Key statistics from testing the effect of multiple failures in structures. Times given indicate the task completion time, the time to both repair and reach the finish line, and are calculated solely from those runs which completed within the time-out limit.

recruitment of Replacement Modules necessary for the substructures to dock again. Overall the repair strategy appears to be working ideally here, the only problem being the increased navigational difficulties involved when so many moving substructures are present, which reduces the proportions completed within the simulation time-out limit as compared to single failures of robots in this structure. Among these scenarios those runs which complete have median, mean, minimum and maximum times broadly similar to those of found in DMS experiments for the self-repair of a failure of robot 2 in S3 (531, 507, 278 and 862 seconds respectively). Amongst single robot failure experiments it was previously found that task completion times by the DMS strategy were slower for a failure of robot 2 than of robot 10, in these double failure experiments it therefore appears that the slower operation required has the main influence on the overall time. For the failure of 10 and 7 the replacement of 7 and its slave via nDSR methods is slowed due to the difficulty of recruiting robots in to this corner area, which is comparable to the difficulties of navigating a substructure back to this location when replacing robot 2.

### 7.12.2 Repairs of Hierarchically Positioned Robots

In tests where the failed robots were both within the same branch of the structure, hence the failed robots were above and below each other in the hierarchy, the scenarios used were: 2 and 4 in structure 12A, 6 and 10 in Rand, 7 and 1 in Rand, 6 and 9 in Rand, 9 and 6 in Rand, and 6 and 7 in Rand.

For the failure of 2 and 4 in 12A global mastery was initially shifted in to robot 3, inbetween robots 2 and 4, and then 4 failed. Unexpectedly 3 then retained global mastery after 4's failure, it had been expected that further remapping would occur and transfer mastery to one of 4's slaves as at the time of 4's failure all of these slaves were in the same substructure as the global master. As this did not occur 3 remained master and began recruiting replacements for robots 2 and 4, acting in the MFM role for both of its neighbours. The large substructures formed by the self-repair operations disposed of the failed modules correctly whilst robot 3 recruited free modules to act as replacements, these replacements then recruited the substructures back in to place, repairing the structure.

For the failure of 6 and 10 in Rand mastery was transferred after 6's failure in to the largest substructure, making 7 the master. When 10 also failed its slave robot acted to drag it away whilst 7's connected substructure dealt with recruiting replacements. Times taken here are similar to those in the single robot failure experiment for robot 6 in Rand (mean 327 s, median 412 s, minimum 257 s, maximum 951 s), with those runs which fail often being caused by the difficulty of recruiting replacement robots when two large substructures which have disposed of failed modules are orbiting close to the recruiting structure.

The failure of 7 and 1 in Rand gave mysterious results with all runs failing to complete in time, further analysis found that while the first failure was handled correctly and robots began to take appropriate roles to handle the failure of 7, the failure of the global master then caused the MFM robot responsible for recruitment of a replacement robot to enter the "Wandering" state. Dynamic Master Switching did not seem to have activated correctly this second time, potentially an error with a similar cause to the unexpected master position in the 12A double failure example, implying an error in the way in which robots determine whether they are in the global master containing substructure for purposes of deciding whether they can engage in Dynamic Master Switching. Unlike in the 12A scenario where the error was overcome, the error here meant that once the previously assigned MFM robot entered the "Wandering" state no robots were recruiting, hence they could not rebuild the structure, repair would never have completed even if the time-out on the simulation had been increased.

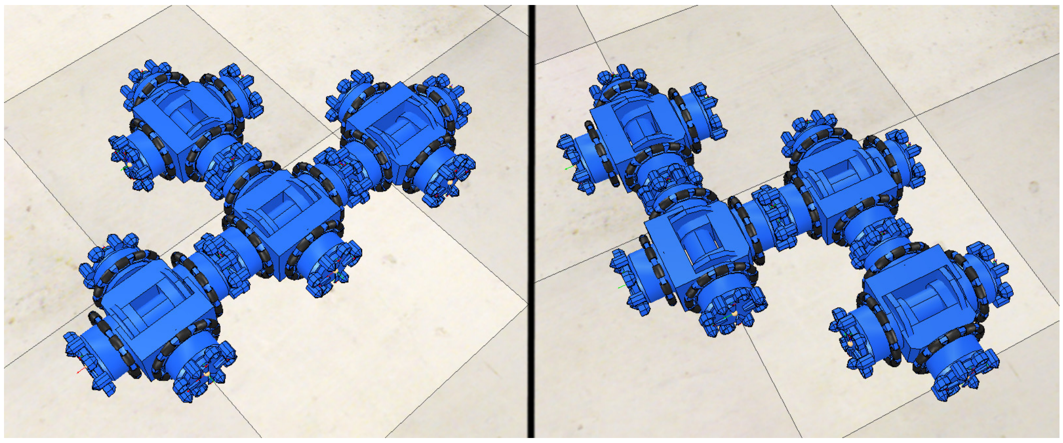
A pair of these scenarios tested the failure of two robots within the same branch of a structure in different orders, using modules 9 and 6 in the Rand structure. When 6 failed first repair proceeded similarly to in the scenario where 6 and 10 failed. When 9 failed first mastery also transferred initially to 7, expected as 7 was the immediately connected to robot to 9 in the largest structure connected to 9, mastery then failed to transfer in to 1 when 6 failed, despite 1 now being in the largest connected structure. In practice this meant that for either order of the failures of 6 and 9 the same repair procedure occured, however this was not the expected behaviour. Due to this unexpected master switching behaviour it was not possible to observe how self-repair would handle scenarios where one substructure was recruiting for another substructure which was itself recruiting another to repair it.

One type of scenario which was expected to prove particularly tricky was where the two failed modules are connected to each other, this is due to the way in which the recruited replacement module uses information about whether its slaves will have subslaves to decide whether to recruit for wandering robots or to recruit for specific substructures. It was expected that DSR with DMS could handle this form of failure only by resorting to an nDSR type strategy. This was tested with a single example, the failures of robots 6 and 7 in the Rand structure. The runs of this scenario did not complete for reasons related to those which prevented the Rand structure with failures of 7 and 1 scenario from repairing. Where 6 then 7 were injected with failures it appears that the mastery was initially transferred in to 7, but that the initial breaking of connections around 6 did not occur. This appears to be because by the time that 6's neighbours had recognised 6's failure, due to a lack of communications from it, 7 had already failed but not yet been detected as having done so. Two substructures resulted from this event, both acting in what appeared to be MAS roles. With no robot in an MFM role recruiting for replacements no repair occured and the substructures eventually broke themselves up in an attempt to resort to nDSR, which unbeknown to them could not succeed due to the lack of any recruiting robots. The amount of analysis possible in the limited time available suggested that undesired remappings like this occur due to an error when robots decide whether they can or cannot perform dynamic master switching, line 3 in Fig.(7.24), potentially due to dead robots not being correctly accounted for when tracing mastery through a structure. The failure of the DSR with DMS strategy to repair in this scenario is particularly unfortunate given that in real-world situations where mechanical damage is responsible for the failure of multiple modules, for example falling debris, it is highly likely that damage would occur to modules which are next to one-another. Self-repair in these scenarios may provide an interesting and relevant challenge for future work.

To conclude this section, the DSR with DMS strategy has demonstrated some level of ability to recover from multiple failures, however in the case of multiple failures within the same branch of a structure's hierarchy it does not choose the most sensible positions to transfer the mastery to. This can lead to a total failure of the repair strategy if robots which are connected together fail or if the master robot experiences a failure after another failure has been detected. An example video is available at [223] and in the supplementary material (Example_multi-failure_scenario.mp4) which shows how mastery switches in to robot 3 in the structure 12A scenario but then does not further relocate after the second failure despite the failure having occured within the same structure as the global master. It is possible that these problems may be addressed by more carefully considering the way in which recruitment lists can be traversed to remap them when some of the modules within them are failed.

## 7.13 Attempting a Hardware Demonstration of Principles

The Dynamic Self-repair controller was translated from a V-REP Lua script to C for use on the physical modules, these controllers were then modified to take account of the lessons learnt from implementing self-assembly on the hardware in regards to docking detection and IR communication, see Sec.(6.5) and (6.6). The version of the controller translated included the Dynamic Master Switching capabilities as demonstrating the ability of hardware to perform this task was considered more relevant than attempting a maximised performance run using the original Dynamic Self-repair method. Due to the number of modules available, multiple failure scenarios were not planned to be replicated with the hardware.



T structure
{{1,1,1,2},{1,3,1,3},{1,4,1,4}}
S structure
{{1,2,2,2},{2,3,4,3},{3,3,1,4}}

**Fig. 7.34:** The T (left) and S (right) structures used in the demonstrations. These are the same structures as from Fig.(6.21) in Chapter 6, however the S structure has been modified so the global master is at one end. Hence a failure of the robot immediately neighbouring the global master allows for a 2 module substructure.

Modules were manually placed to allow a quick initial self-assembly, then once they were moving a switch was pressed on one module to trigger a failure mode. In this failure mode the module's wheels stopped, all infrared and Wi-Fi communications ceased, and proximity sensor pulsing was disabled. Connected modules were required to detect the failure, then remap their structure and take on appropriate roles for self-repairing. Once the failed module had been removed it was manually released from its failure mode and the software reinitialised such that it acted as a free wandering robot. This method of reinitialisation required a reversible induced failure mode, to enable the failed module to be reused later as a wandering module. The limited number of modules available meant that providing a spare module to become the Replacement Module, in the way that self-repair would in a practical context, would too greatly limit the structure sizes possible. Modifying the C code to insert this reversible failure feature proved a time consuming and challenging task. The module was then allowed to dock again either as the Replacement Module or as a recruit to replace other parts of the broken up structure when Mergeable Nervous Systems over Wi-Fi was in use.

With only 4 hardware modules available scenarios involving large substructures could not be created, a T structure, see Fig(7.34), was therefore used and the failure induced in the global

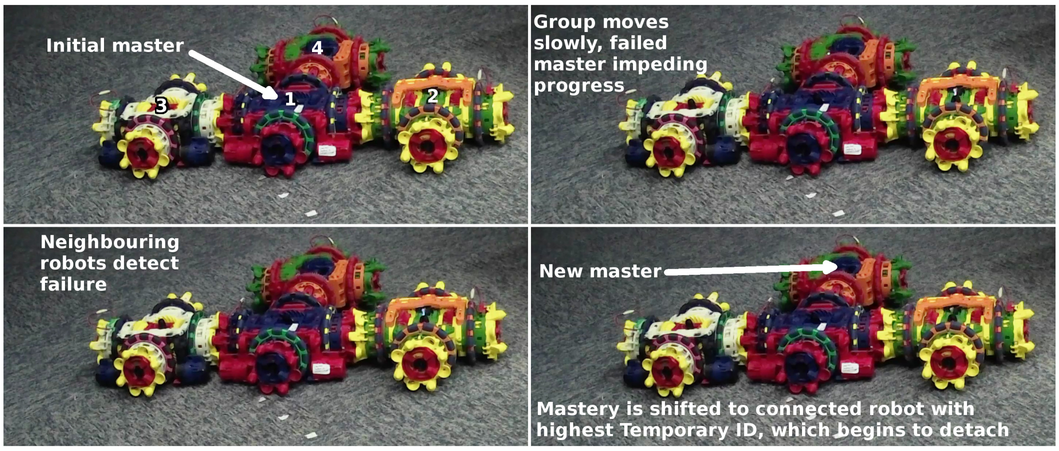

Initial master
4
1
2
3
Group moves slowly, failed master impeding progress
Neighbouring robots detect failure
New master
Mastery is shifted to connected robot with highest Temporary ID, which begins to detach

**Fig. 7.35:** Dynamic Master Switching seen in hardware, video at [224] and in the supplementary material (DMS_testing_with_hardware.mp4) provides an example where the robots' announcements as the structure's mastery is shifted are partially audible. The modules then take roles as the Master to the Failed Module and as the MNS over Wi-Fi removal substructure.

master. After Dynamic Master Switching processes had occurred, see Fig.(7.35), Mergeable Nervous Systems over Wi-Fi was automatically selected by the robots as the method for removing the failed module. Unfortunately the assumption that two modules would be able to manoeuvre a connected failed module proved incorrect, limited torque from the robots' wheel motors, and the limited amount of grip on the wheels, meant that two modules attempting to move an uncooperative module would stall. With this assumption invalidated, self-repair could not proceed in the same way as it had been simulated. The collapse of this assumption also suggests a greater sensitivity analysis should have been performed during the production of simulation assets, and upon finding the problem at that time a greater focus could have been given to the improvement of wheel torques during hardware development. Features in the C code were developed to allow a failed module to still listen to MNS actuation commands over Wi-Fi from the commanding robot of an MNS over Wi-Fi retreat or from the MRS module, depending on the form of self-repair in progress, however these communications were not always acted upon correctly by the failed robot, likely due to other effects created by the failure state induced.

Whilst the section of the robotic organism which had become the master side to the failed module was therefore able to escape and begin re-recruitment, the failed module could not be dragged away effectively. Fig.(7.36) shows the attempt in progress, the failed module followed the Wi-Fi commands at some points but at others the whole retreating structure was stalled when it did not. After a period of stalling the modules acting in MNS over Wi-Fi roles detached and became free wandering again, fortunately the use of MNS over Wi-Fi disposal methods does not require robots to successfully complete a rotation before undocking from the failed unit.

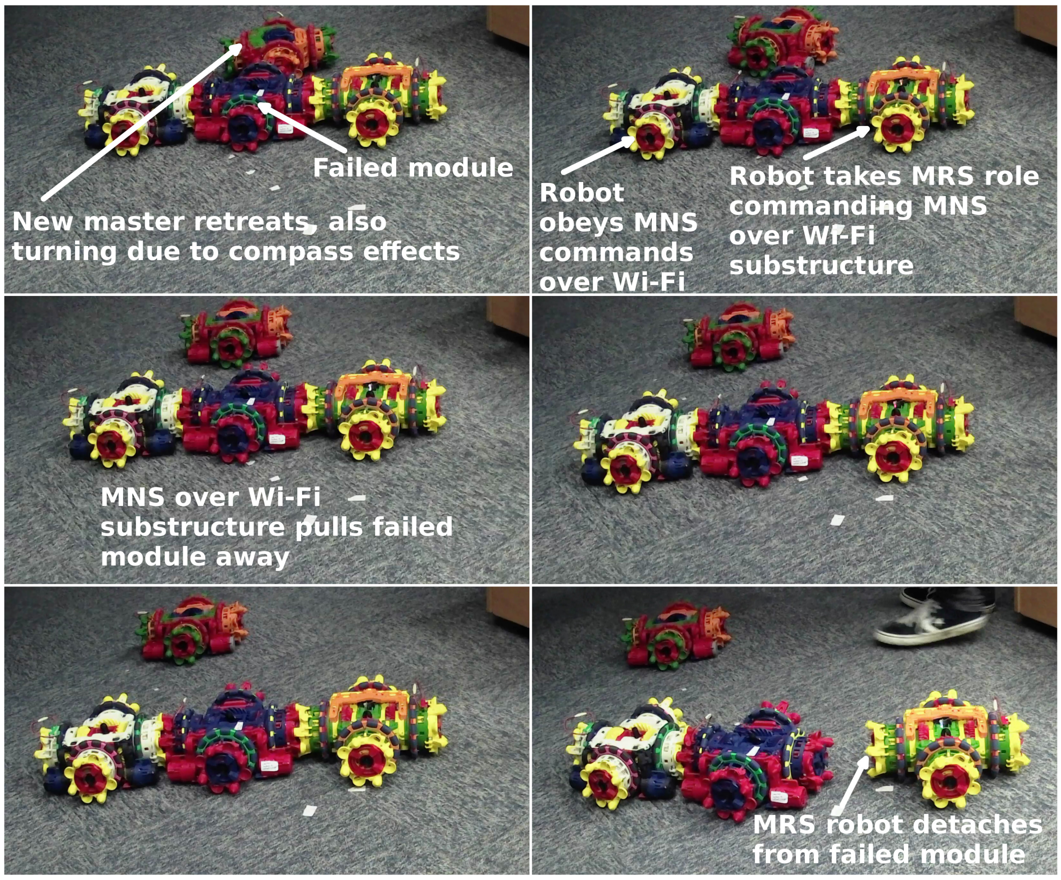

Failed module
New master retreats, also turning due to compass effects
Robot obeys MNS commands over Wi-Fi
Robot takes MRS role commanding MNS over Wi-Fi substructure
MNS over Wi-Fi substructure pulls failed module away
MRS robot detaches from failed module

**Fig. 7.36:** Neighbouring robots attempt to manoeuvre a failed module, whilst the new master retreats via the "hallucination" method.

This demonstration proved the ability of robotic modules to detect a failed neighbour, and showed Dynamic Master Switching working over Wi-Fi. Although coding errors made it difficult to return a module to correct operation after a failed state was induced, a single incident was also observed in which modules which had attempted an MNS over Wi-Fi retreat were re-recruited in to partially rebuild the structure, see Fig.(7.37) and video at [224] as well as in the supplementary material (DMS_testing_with_hardware.mp4).

Another demonstration was attempted where Dynamic Master Switching features were deliberately disabled to allow something close to ordinary Dynamic Self-repair to occur, this was intended to cause a situation where the docking of a multi-module substructure to the newly recruited Replacement Module could be observed. A modified S, see Fig(7.34), structure was used with the fault injected in to module 2 such that a two module MRS substructure was formed. This demonstration however suffered from a number of reality gap issues. The difficulties already noted in Sec.(6.6) in Chapter 6 not only made it difficult for a robot to reliably determine whether it was docked, but also meant robots struggled to recognise undocking events, Recruitment Lists shared between them were therefore often inaccurate descriptions of the structure's state. Compared to self-assembly, the requirements which self-repair placed upon the reliability of docking detection equipment, in Omni-Pi-tent's case

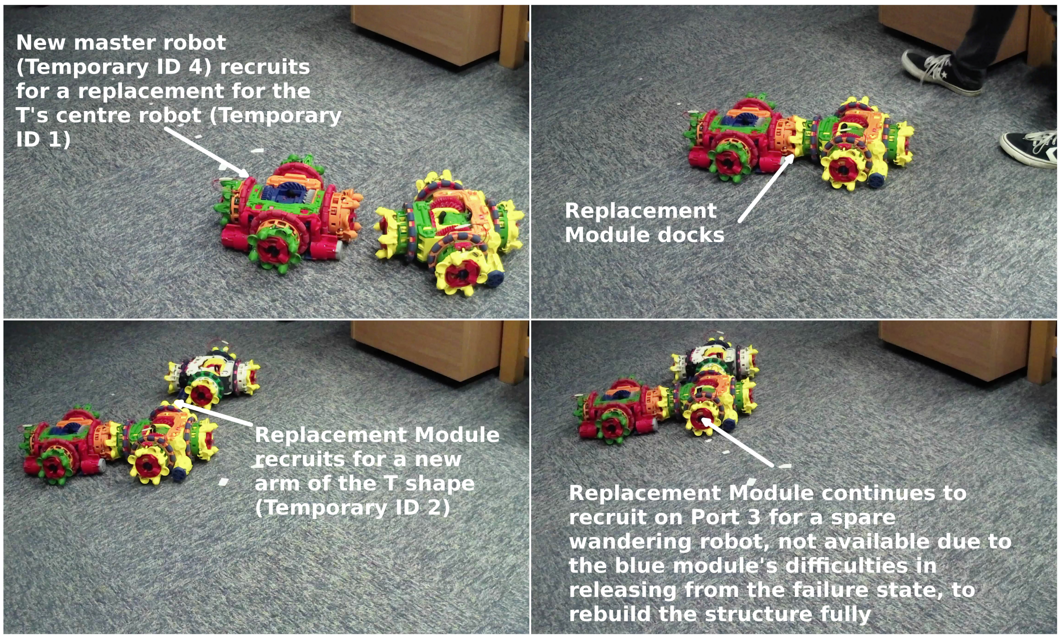

New master robot (Temporary ID 4) recruits for a replacement for the T's centre robot (Temporary ID 1)
Replacement Module docks
Replacement Module recruits for a new arm of the T shape (Temporary ID 2)
Replacement Module continues to recruit on Port 3 for a spare wandering robot, not available due to the blue module's difficulties in releasing from the failure state, to rebuild the structure fully

**Fig. 7.37:** A series of images shows the rebuilding of part of the T structure according to the Dynamically Master Switched Structure Recruitment List.

the microswitches in the spike accepting pits, are far more exacting. Whilst in self-assembly the difficulties could be overcome with software work-arounds, self-repair's more frequent and repetitive docking and undocking events overwhelmed the ability of software work-arounds to accurately keep track of the port states. This provides an example of how self-repair proves to be a much more complex task than self-assembly, not only at the theoretical level and in the design of robotic controller software, but also at the practical level in ways which do not become apparent until real hardware is involved. As many communications which were planned during the development of simulated strategies as line-of-sight were forced to use Wi-Fi due to the limitations of 38KHz IR communications, problems also occurred in regards to addressing communications. Over port-to-port infrared communications addressing was un-necessary, robots received and acted upon these by virtue of being in the right region at the right time, for Mergeable Nervous Systems messaging this was simply a matter of having ports close enough for the shortest ranged low power messages to transfer and including a Temporary ID value so the receiving robot could ensure the messages were meant for it. The Omni-Pi-tent platform's Wi-Fi message addressing however operates using hardware ID numbers, separate to the Temporary ID numbers used for Recruitment Lists. Various methods were attempted to provide correct communications, a lack of which was recognised by the robots as a sign of module failure, however each attempt was either unsuccessful, or lead to pointer errors, corruption of other variables in the C stack and side-effects including the inability for robots to dock to form the structure initially. Again, software work-arounds for hardware limitations which had enabled successful self-assembly with hardware modules proved inadequate for the challenges of self-repair. The task of self-repair highlights the problems caused by communications between fully operational modules being less reliable than intended. It highlights also the vital nature of the implicit information, carried separately from the message contents themselves, available when local line-of-sight messaging is used, as evidenced by the confusion which resulted from non-line-of-sight global messaging methods being used to replace them. The method used to disable the Dynamic Master Switching features may also have caused errors affecting some of the variables used to track docking port states. Video at [224] and in the supplementary material (DMS_testing_with_hardware.mp4) shows an example of one of the ways in which a Dynamic Self-repair attempt failed.

With limited time available for these experiments [1] it was not possible to test all aspects of importance for achieving Dynamic self-repair with hardware modules. However by providing proof that it is possible for neighbouring hardware modules to detect failed modules, and given that the Mergeable Nervous Systems and Multi-Layered Recruitment hardware demonstrations of Sec.(6.6) show that the collaborative driving necessary for removing failed modules and returning substructures to dock is possible, many of the main physical tasks involved in Dynamic Self-repair have been accomplished. One noted task which, due to the complications discussed preventing a Dynamic Self-repair scenario being produced, was neither shown here nor as a side-effect of Sec.(6.6)'s work was the use of wheel speed odometry and guidance via line-of-sight infrared communication to aid the navigation of substructures during their return. The practical difficulties which may be encountered when implementing it, beyond the difficulty of performing such navigation already apparent in simulations, remain an open topic for investigation.

## 7.14 Summary

In this chapter self-repair strategies for self-assembly with modular robots during motion have been considered. Building on the work of [51], and using techniques derived from [26] for the implementations, a number of strategies were developed and compared in experiments. Key among them was the newly developed Dynamic Self-repair strategy which both serves to enable modular robotic structures to remain in motion while repair takes place and to preserve all multi-robot substructures which had been connected to the failed robot before the failure occurred. By comparing the speeds of task completion when the Dynamic Self-repair strategy was employed to the speeds of task completion when self-repair operations required group motion to cease, and to methods of fault recovery which involved full breakups of substructures and reconstruction of structures via standard self-assembly methods, Dynamic Self-repair was found to be superior. Dynamic Self-repair's speed advantage during tasks was

[1]Due to the dependency chains involved in developing and refining hardware, embedded software and group behaviour strategies, Dynamic Self-repair with hardware could not be begun until the final month of the project's timeframe. Despite debugging until the final day some errors remained unsolved, a further few weeks may have been sufficient to overcome them. However, some of the issues still appear to stem from hardware limitations, the solutions for which are not prohibitively complex but would require a long process of new module production.

found to be a consequence both of its ability to avoid lengthy re-recruitment processes and of its ability to allow self-repair to operate concurrently with the group task. The sub-conclusion involved in this determination, that self-repair strategies are more effective than breakup and reassembly, provides a replication of some of [51]'s conclusions with our new platform and strategies. An initial probing also used some of the data from these experiments to consider how the relative effectiveness of self-repair strategies scales with the number of robots connected to the failed module.

To summarise the results of the earlier part of this chapter the null hypotheses tested are listed below with the key findings beside each:

**Null Hypothesis 7.1:** *Being able to repair while in motion has no effect on the chance of whether self-repair, and a locomotion task, will complete within a specified time limit.* **- Rejected:** Tasks which are interrupted by a requirement to self-repair have a greater chance of completing within a time limit when conducted with dynamic strategies than with equivalent static strategies.

**Null Hypothesis 7.2:** *Being able to repair while in motion has no effect on the time taken for locomotion tasks during which a repair is required.* **- Rejected:** Dynamic Self-repair strategies allow for faster completion of tasks, particularly due to the way in which such strategies parallelise the repair operation alongside task completion.

**Null Hypothesis 7.3:** *The act of repairing, independent of the time to complete other tasks, takes equally long for repairs undertaken while in motion as it does for repairs during which the structure remains static.* **- Rejected:** Dynamic Self-repair strategies prove slower to perform the repair itself, despite improved overall task performance.

**Null Hypothesis 7.4:** *There is no difference in the time taken for a group task between repair strategies requiring a significant breakup of the structure and repair through ordinary self-assembly methods, as versus repair strategies which keep multiple substructures of robots connected throughout.* **- Rejected:** Among static strategies with breakup strategies being slower, but **Not Rejected** among dynamic strategies where in many scenarios DSR displays no advantage over the breakup based nDSR.

**Null Hypothesis 7.5:** *The act of repairing, independent of the time to complete other tasks, takes equally long for repairs using breakup and re-assembly type strategies as it does for repairs using self-repair strategies in which substructures are preserved.* **- Rejected:** Repair operations take longer for breakup based strategies than substructure preserving ones, with significant differences found between dynamic strategies as well as between static ones although differences are less.

**Null Hypothesis 7.6:** *Relative effectiveness, in regards to task completion time, of different self-repair strategies does not vary with the size of the structure below the failed module in the* *multi-robot organism's hierarchy.* **- Rejected:** As structures grow larger, breakup strategies lose performance while substructure based strategies remain relatively consistent.

**Null Hypothesis 7.7:** *The size of the structure below the failed module in the multi-robot organism's hierarchy does not alter the relative times, between different strategies, for the self-repair procedures themselves to complete.***- Rejected:** For larger substructures breakup strategies lose performance while substructure based strategies remain relatively consistent, this effect is present within the times for repair operations themselves not just overall task times.

### 7.14.1 Further Conclusions

Further work in this chapter then developed a Dynamic Master Switching strategy which enabled failures of the master robot of a structure to be recovered from, and which used the lessons of earlier experiments to make an optimised choice as to which robots neighbouring the failed module would serve best in which roles. This provided an opportunity to demonstrate new capabilities in modular robotics which the quadruplet array data structures open up, and showed reliability, although not speed, advantages above the initially developed Dynamic Self-repair strategy. The Dynamic Self-repair strategy with Dynamic Master Switching was also shown to manage well with multiple failures in separated parts of a structure, but to give unexpected behaviour for multiple failed robots within each others' hierarchy, previous work had not tested any self-repair strategies in scenarios with multiple failed modules.

Hardware tests attempted to prove the real-world feasibility of this Dynamic Master Switching version of Dynamic Self-repair, however faced complications due to hardware limitations, reality gap issues and limited time. This served to highlight the far greater complexity involved in executing hardware self-repair than hardware self-assembly. While some of this increased complexity had been expected from the simulations, much of it came in the form of additional challenges not evident until attempts with hardware were performed.

These later elements within the chapter relate, as follows, to the null hypotheses:

**Null Hypothesis 7.8:** *There is no benefit to be had in optimising which connected robot or group of robots attached to a failed module is responsible for which aspect of the repair procedure.***- Rejected:** Whilst not faster than standard Dynamic Self-repair, a modified Dynamic Self-repair strategy able to dynamically switch the location of the master, and with a substructure role preference system based on the lessons from the earlier self-repair experiments, showed reliability improvements.

**Null Hypothesis 7.9:** *Successful dynamic self-repair is impossible to achieve using hardware robots.***- Not Rejected:** Whilst it was possible to achieve some aspects of Dynamic Self-repair with physical modules, a full demonstration could not be produced within the time available. Given the tasks already achieved with hardware it does not appear impossible. However this cannot be stated with enough confidence to truly reject the null hypothesis until a full replication is demonstrated, for which hardware modifications may be required. In failing to reject this hypothesis, these hardware demonstration attempts have provided an indication of how much harder a problem than self-assembly, completed successfully in Sec.(6.6), self-repair often is. This highlights the importance of reliable docking detection and local communication for modular robots.

This shows that, despite some limitations, continuous motion based self-repair strategies are both fast and effective. The simulations show that with the capabilities which an Omni-Pi-tent module has the strategies are feasible, however, due to limitations with the physical hardware, performing Dynamic Self-repair with real robots will require some amount of future research. It also suggests that self-repair could provide a good benchmarking task with which to prove the reliability of a modular robot platform's docking and communication systems.

The next chapter concludes the thesis by summarising the preceding chapters and revisiting the Dynamic Self-repair hypothesis with their results in mind, before proposing future work.

# Chapter 8

# Conclusion

This chapter reviews and highlights the important aspects of this thesis. Each chapter's contents are summarised with a focus on contributions to knowledge and capabilities, these are then considered in the light of the Dynamic Self-repair hypothesis which runs throughout this thesis. The hypothesis is revisited and the conclusions of these chapters are tested against it. This chapter, and thesis, concludes with proposals for further work following on from the Dynamic Self-repair project, and how that further work could bring modular robots out from the lab and into the wider world.

## 8.1 Chapter Summaries

This thesis can be considered as a combination of: discussions on earlier work and the conceptual framework around Dynamic Self-repair, infrastructure development, multi-module strategy development and experimentation. Chapter 2 presented much of that earlier work and set the scene for the Dynamic Self-repair concept, Chapters 3, 4 and 5 explained the infrastructure developed to underlie both hardware and simulated experiments, Chapters 6 and 7 showed the development of controllers for multi-module behaviours and then tested them, through simulated experiments, before attempting to prove their underlying principles with the hardware. These chapters, with a focus on their main findings, are summarised below:

- **Chapter 2** examined the state of the art in modular robotics and detailed the existing platform designs, with a focus towards mobile and hybrid architecture platforms. A lack of sensor capability in many designs, and the limitation this puts on their abilities for autonomous tasks, was noted. Consideration was given to deducing which properties for modular robots would be particularly valuable in commonly suggested applications (infrastructure monitoring, disaster rescue and planetary exploration) and what constraints that could put on an ideal platform design, as well as noting some of the obstacles modular robotics research must overcome before widespread adoption of the technology. Previous examples of modular robotic self-assembly were reviewed, this found methods which were both decentralised and able to form exactly defined connection-list based structures to be particularly rare despite appearing most appropriate for near-future modular robotic applications. Modular robotic self-repair was also reviewed, for which very few practically relevant attempts have been made, despite it being a “Grand Challenge" [33] for research. Limitations of the existing approaches were discussed and the benefits of a strategy which could self-repair during continuous motion (Dynamic Self-repair) were highlighted. Experimental scenarios to test concepts of Dynamic Self-repair were proposed.

- **Chapter 3** begun the discussion of the hardware which would need to be developed to enable such experiments to be run. The effects of embodiment and the other benefits that hardware development provides for robotics research were discussed, after which the hardware development goals were considered both in terms of the capabilities required for Dynamic Self-repair experiments and also in terms of how these would compare to the capabilities to be expected in, and design constraints to be expected upon, a near-future industrially relevant modular robot. The need for genderless docking, omnidirectional locomotion, capable sensors and non-reliance on external infrastructure were explained among other requirements. The Omni-Pi-tent design was outlined and its features described in this context, throughout this design the concept of being ‘not “just" a module’ was used. The electronic architecture of Omni-Pi-tent modules was introduced and brief explanations made of the techniques used to fabricate them. This design is one of relatively few modular robot platforms which is designed with sensor systems as a major consideration alongside the reconfiguration and actuation mechanisms.

- **Chapter 4** explained the simulation environment produced for use alongside the physical Omni-Pi-tent hardware platform. The use of simulation in robotics was discussed and the reality gap highlighted as a concern. For each key actuation and sensing feature on the Omni-Pi-tent platform the methods used to represent it in the V-REP [28] simulator were detailed. Particular attention was paid to the simulation of the docking guidance beacons, and how reality gap reduction was attempted with a simulation model derived from experimental data. A discussion was also included on the topic of sensitivity analysis for parallelising simulation and hardware development, and examples given in the context of the Omni-Pi-tent platform as to what effects inaccuracies in simulation could have.

- **Chapter 5** completed the description of the hardware by providing more complete technical details and some windows into the process of design and iteration. A more detailed description of each key mechanism was provided with a focus on the design and debugging necessary for all subsystems to operate correctly in the real world.

Testing and refinement of the more notable subsystems were detailed. Further details of the internal electronic architecture were provided, along with explanations of the embedded software functionality within different parts of the design. These subsystems and software were compared to their simulated equivalents as discussed in Chapter 4. Performance figures of the optimised design were summarised and some comparisons made to other platforms and against modular robot ranking methods from literature [45] [125].

- **Chapter 6** developed and tested self-assembly strategies with the new Omni-Pi-tent platform. Hypotheses, see Sec.(6.1), were presented regarding how the multi-module group capabilities required for dynamic self-repair could be applied to the task of self-assembly. An initial self-assembly strategy (LW+) was derived by adapting Liu and Winfield's [27] work with SYMBRION to the new Omni-Pi-tent platform. The self-assembly controller developed for Omni-Pi-tent was described in detail, with a new quadruplet array format for structure definitions explained. This quadruplet data structure proved particularly useful due to its highly human readable format, as well as its self-contained nature, packaging all necessary description of the structures in to an easily transferred form. The design and development of timeouts and other safety features within the new self-assembly controller, rarely found in earlier work, were also described. Further developments (LW+MNS) of the self-assembly controller were considered which introduced features derived from the Mergeable Nervous Systems project [26] to allow co-ordinated motion of Omni-Pi-tent modules via a reasonably scalable method. Docking to moving robots was also covered as a vital step towards Dynamic Self-repair as well as towards self-assembling while in motion. A "multi-layered" recruitment strategy (MLR) capable of operating on multiple concentric layers was developed to further explore the possibilities offered by parallelising tasks within modular robotic systems. Experiments then compared the LW+, LW+MNS and MLR strategies to test what performance advantages could be offered by self-assembling while in motion, the MLR and LW+MNS strategies gave similar performance to one-another and consistently outperformed the LW+ strategy to form a number of different structures within a variety of different arena geometries, this provided early indications towards the likely value of Dynamic Self-repair. The feasibility of self-assembling to a moving seed and co-ordinated group sensing and motion were demonstrated with hardware modules and the challenges involved discussed.

- **Chapter 7** extended the previous chapter's self-assembly work to produce self-repair algorithms. Building on Murray's thesis [51] self-repair methods were developed which harnessed the unique capabilities which Omni-Pi-tent could provide. The newly developed self-repair algorithms were explained in detail, as were the methods of processing quadruplet array structure definitions to control these actions. Four strategies were compared in an experimental scenario which required modules to complete a locomotion task and complete a self-repair operation to remove a failed unit. Dynamic self-repair, in which self-repair proceeded while motion continued, was compared against static self-repair, which required the group motion to halt while repair was undertaken, and a pair of “naive" strategies, both of which involved a breakup and re-assembly of the structure in parts of the structural hierarchy “below" the failed module. Hypotheses, see Sec.(7.2), were presented which investigated the effects of different features within the Dynamic Self-repair strategy. The speed advantages both of self-repairing, in contrast to naive breaking up, and of the ability to perform while in continuous motion were shown. A Dynamic Master Switching method was also developed for the quadruplet array structure descriptions which allowed the assignment of different tasks to the substructures connected to a failed module to be set in an optimised way. This Dynamic Master Switching was compared against the standard Dynamic Self-repair method and found not to offer a speed advantage in the structures tested, but was able to provide more reliable self-repair. Also, the Dynamic Master Switching method proved more versatile due to its ability to handle replacement of a failed global master within a structure. Brief consideration was also made of multiple failures and the Dynamic Master Switching strategy's abilities and limitations in handling these. Attempts were made to replicate with hardware the Dynamic Master Switching version of the Dynamic Self-repair strategy, however fell victim to time constraints which prevented the overcoming of complications in software translation. These hardware difficulties highlighted the vastly more challenging nature of self-repair as compared to self-assembly, and suggest self-repair may form a worthy benchmarking task with which to compare the reliability of modular robot platforms. They also highlighted that self-repair tasks expose not only the theoretical and conceptual increases in difficulty between self-assembly and self-repair, but also the extra complexities self-repair encounters when implemented in hardware. Also this chapter included a replication, with independently developed self-repair strategies and implementations, of some of Murray's results from [51], particularly in showing the advantages of self-repair over breakup based strategies. This appears to be the first replication in modular robotics of any specific finding in regards to self-repair.

## 8.2 Revisiting the Dynamic Self-repair Hypothesis

At the start of this thesis I introduced the Dynamic Self-repair hypothesis which stated that:
*Modular robots using self-repair strategies which can operate while the group maintains collective motion will be able to complete tasks faster, and more effectively, than if using a self-repair strategy which requires the system to halt collective tasks so as to perform self-repair.*
The contributions of the previous chapters provide strong evidence for this hypothesis being

true. It can therefore be concluded that self-repair during continuous motion is possible, this in itself is a new discovery, previous work had always simply assumed group tasks must stop for self-repair to occur. Beyond this it can be concluded that it could be undertaken with modular robotic platforms with features comparable to those to be expected on near-future real-world modular robotic systems, although perhaps somewhat more capable than those on the physical Omni-Pi-tent hardware, and without reliance on external infrastructure such as global position sensing. The work presented in this thesis shows that Dynamic Self-repair can indeed be faster, and enable more rapid completion of group tasks should a module develop a fault during operation and require replacement. As explained in Chapter 2, this speed increase makes the self-repair strategies more effective in a wide range of situations. While there are disadvantages to self-repair during continuous motion in some contexts, having the option to perform it still provides valuable capabilities for modular robotic systems. The usefulness of continuous motion strategies has also been shown to be applicable beyond self-repair via experiments focused on self-assembly.

In the act of testing this hypothesis the work of this thesis has also enabled the formulation of a set of “6 laws of dynamic modular robotics".

- **Parallelisation:** The value of using the multi-robot nature of modular systems to achieve parallelisation has been demonstrated, parallelisation can assign sections of a multi-robot organism to different parts of a task and can allow the organism as a whole to parallelise tasks relating to the organism’s interaction with the environment to run alongside tasks relating to the organism’s internal structure.
- **Human readable, physically inspired, data structures:** Parallelisation is enabled, by, amongst other things, physically inspired data structures which naturally enforce the physical constraints on how robots can connect, by also making these structures human readable they provide opportunities to simplify the adoption of modular robots in future applications.
- **Not “just" a module:** Beyond the benefits provided to modular robotic software development this thesis has provided lessons for future hardware development by strengthening the case for the concept of “Not ‘just’ a module", and the way that this concept can improve both the effectiveness of individual units, and thereby also the robotic group as a whole.
- **The embodiment principle:** This “Not ‘just’ a module" principle also allows the concept of embodiment to be brought to more prominence in the modular robotics field, noting how the practical constraints on hardware robotic design anchor simulations to reality and how tightly coupled simulations and hardware are for this field.
- **Mixed global and local communications:** A particularly interesting example of how physical considerations can be exploited by a modular robotic system is found in

the ways in which local communications can carry implicit information beyond their message contents, and how this can be harnessed by systems of mixed global and local communications.

- **The hierarchy of challenges:** The complexities of modular robotic communication provide a major component to the hierarchy of modular robotic challenges, with the increased communication and reliability requirements involved in self-repair, beyond those of self-assembly, showing self-repair to be more difficult than self-assembly not only in terms of theoretical and controller related demands, but also in terms of development required at the hardware level.

Informed and aided by these “6 laws", Dynamic self-repair, and other forms of modular robotic reconfiguration able to operate whilst in motion, can be a useful extra tool among the wide range of capabilities which modular robotics provides.

## 8.3 Future Work

There is significant scope for future work following on from that presented in this thesis, not least due to the novel Omni-Pi-tent modular robot platform now developed and the accompanying simulation assets to aid in using it. Future work can be considered in a number of different strands.

### 8.3.1 Further Self-assembly

While, within this thesis, self-assembly has been used as a stepping stone to self-repair there is much scope for further work within the assembly field too, this could include the development of strategies which can self-assemble in a 3 dimensional situation, especially in terms of how to transport robots around a multi-robot organism by using other modules to manipulate them in to position. To repeat more complex 3 dimensional operations in hardware some further hardware redevelopment to improve the 2DoF hinge mechanism and reduce module weight may be required, unless strategies can be developed which account for the effects of hinge torques limited to lifting single modules at once. Further complexities which could be considered are self-assembly scenarios in which semi-specialised robots or tool modules occupy specific positions within the structure and assembly must proceed while ensuring that such specialised modules are distributed to appropriate locations, as well as tool modules scenarios involving passive attached structures [188] could also be considered.

### 8.3.2 Loops and Concave Docking

Neither the self-assembly nor self-repair scenarios in this thesis have included structures containing loops. This is, for the most part, not a limitation of the self-assembly and self-repair algorithms but of the platform's hardware. Specifically in that, as mentioned in Sec.(6.2.10), docking in to a concave space, as required when completing a loop, is physically impossible due to the locations of the docking spikes. Further work could fully develop an algorithm to split looped structures automatically along straight lines within the shape so as to form multiple smaller structures able to form and then join together, along straight boundaries, once the substructures have formed. This method could also be used in the self-repair of loop-containing shapes when deciding how to partition the structures up into the substructure assigned to handle the removal of the failed unit, the substructure to recruit the replacement, and substructures which will simply need to temporarily retreat so as to avoid blocking the paths of the aforementioned substructures. There have been previous suggestions [51] on how assembly of subgroups before complete assembly could enable the formation of looped structures, such methods could not be achieved by algorithms processing the data structures [27] [29] developed at that time, or achieved using the robot platform [36] available then. However they can be produced by developing further algorithms around the quadruplet data structure and the Omni-Pi-tent platform. It appears plausible that modifying the quadruplet data structure so it can explicitly state that some connections are there for mechanical purposes only and are not a part of the slave and master hierarchy within organisms could be aid in this and allow loops to form while maintaining the helpful non-ambiguity which having only one master-ward connection for each module can provide.

### 8.3.3 Guidance Methods

One of the key difficulties, as determined from the simulation experiments, for dynamic self-repair is guiding the substructures which detach back to appropriate locations from which they can begin re-docking. In this work two methods were combined for this guidance, wheel odometry with equivalent information from the master substructure shared over Wi-Fi to let the returning substructure estimate relative displacements between the structures, and long ranged IR guidance in which the 38KHz communication system was used to broadcast pulses from the master substructure which returning substructures could use to estimate relative positions from knowledge of lines of sight. This combination did not prove as effective as hoped, and even may prove even less so in hardware due to the reduction in 38KHz IR communication range caused by the presence of 5KHz illumination. Future work should be able to achieve much more reliable Dynamic Self-repair, successfully rejoining in a far greater proportion of scenarios, by using longer ranged docking guidance sensor systems, extra sensor subsystems specifically for this crude but long-ranged guidance, and algorithms to enable returning substructures to plot a course which will act as a search pattern to make straying in to range of guidance systems more likely.

### 8.3.4 Adaptation for Reconfiguration

The applications for Dynamic Self-repair can be considerably expanded, not only for repair but also for general reconfiguration tasks. Reconfiguration is one of the key “selling points" of the modular robot concept, so it would be hard to believe that there would not be uses for maintaining group capabilities during any form of reconfiguration between forms, not just those involved in self-repair scenarios.

### 8.3.5 Improvements to Self-repair and Adaptation for Wider Scenarios

There are possibilities for performing self-repair where detached sections of the structure may re-assemble to replace parts of a multi-module morphology which are not the same sections which they previously acted as. This contrasts to the way in which, in the work of this thesis, retreating sub-structures would return to their original locations, and original Temporary ID values, after a replacement module had been recruited to replace a failed one. Being able to alter the location to which returning modules are recruited would require groups to compare their sub-structures against the overall structural plan of a multi-robot organism and find where, if anywhere their sub-structure reoccurs. Interesting optimisation problems might then be found in regards to selecting which part of a disassembled structure it would be most efficient for the sub-structure to replace, solutions to these may be derivable from the graph theory work involved in [4] and [5]. This type of work would be most likely to be of relevance in structures towards the larger end of those tested in this work where morphologies are big enough that containing repeating elements would be more probable.

Self-repair processes during self-assembly may also prove an interesting avenue for further research. The Dynamic Self-repair with Dynamic Master Switching strategy was developed with this in mind however there was insufficient time in which to subject the strategy to statistical testing of this expected capability. Due to the use of both Structure Recruitment Lists, describing the overall plan, and Recruitment Lists, giving information on the current state of a multi-robot structure it should be well equipped to handle failures during a self-assembly procedure. The algorithms used to assign roles to modules neighbouring a failed unit, the methods of determining which forms of recruitment to engage in, and the Mergeable Nervous Systems behaviours all make use of both such quadruplet arrays and consult the Recruitment List data, reflecting the real-time state of the structure, at many points. Repairs during assembly should therefore be able to be handled as if they were repairs performed on a structure consisting of only those modules so far docked to the structure. As well as testing the effectiveness of Dynamic Self-repair processes during self-assembly operations, there could be interesting possibilities in making further uses of Structure Recruitment Lists and Recruitment Lists working together to be able to fully exploit the capabilities of modules docked not only in their desired final configurations but also in whatever state they are in at the present moment.

The problems found when considering multiple failures also present an opportunity to modify Dynamic Self-repair and Dynamic Master Switching to better handle multiple failures. A remapping method able to compare amongst all existing structures to decide which is best suited to each of the different self-repair roles when multiple modules have failed, rather than merely make those decisions for modules within the same structure as the global master, could enable this.

Dynamic Self-repair could also be adapted to operate in scenarios beyond those of motion. Modular robots engaged in object manipulation tasks, or networked data processing could also benefit. While Omni-Pi-tent was designed deliberately to lack port-to-port electrical connectors for high-speed communication or power sharing, such capabilities would present interesting possibilities, including in scenarios where modules must at all times maintain certain topologies of connections between them so that sharing of computation capabilities and power supplies and loads can continue uninterrupted. Heterogeneity of modules could also become a consideration, with specialised tool or sensor modules added into multi-robot morphologies. Dynamic Self-repair may have to proceed subject to rules such as “the camera module must always have a clear forward line-of-sight". Possibly an ALLIANCE [108] like principle could be used to manage this.

### 8.3.6 3 Dimensional Considerations

Dynamic Self-repair could be adapted in to 3 dimensional scenarios. Many modular robots including Omni-Pi-tent have systems to enable motions in a third dimension, although, and also including Omni-Pi-tent, these are often quite limited in how many modules they have enough torque to move. This would be a challenging undertaking but extremely valuable to modular robotics in the capabilities it could bring, especially if repair and reconfiguration was handled whilst maintaining as much as possible of a 3D structure without having to “fold" it flat on the ground before breaking and making docked connections. Further work on “walking" across multi-robot surfaces, as accomplished by HyMod [21], with two-module meta-module [82] structures would likely be involved. Although an Omni-Pi-tent module has sufficient hinge torques to “walk" as a two module pair, some hardware improvements in terms of reducing module weight and increasing hinge torque would be of use to enable larger meta-modules to move around.

### 8.3.7 Further Simulation Environment Development

The extent of simulations which could be performed in this project was somewhat limited by the V-REP [28] simulation software used. V-REP was chosen on account of a user-friendly interface and the ease with which custom robots and sensors could be modelled within it, but it is slow. This slowness limited the size of datasets which could be collected and prevented a wide variety of parameter sweep simulation scenarios from being run. Comparison papers in the swarm robotics [67] and general robotics [61] fields have already noted V-REP's computational inefficiency and faster packages' user-unfriendliness. Future work in the modular robotics field could benefit from the development of a simulator as easy to use and adaptable as V-REP but with the speed advantages of other, less user-friendly [198], robotics simulation packages. Ensuring that such a simulator included physics simulation and modelling of connected and disconnected modules, which many swarm robotics focused simulators do not do, would be vital, as would truly user-friendly documentation, similar to [65] and [71] for example. Such a modular robotics simulation program would, in my opinion, do very well to emulate V-REP's accessibility to roboticists coming from an interdisciplinary rather than computer science background.

### 8.3.8 Further Hardware Development

Further hardware development could provide some interesting opportunities towards the development of a truly ideal and universal modular robot platform, which Omni-Pi-tent currently cannot be considered to be. Platform capabilities, versatility, reliability and autonomy could be expanded, especially in regards to widening the limited sensor suite, improving the infrared communications and simplifying the construction process.

While not practical due to some of the design decisions made early in the development process a number of ideas have arisen to the author of precise ways in which the Omni-Pi-tent platform could be improved had alternative decisions been made much earlier in the design process. Detailed consideration has been given to how these changes could be performed, a brief overview of them is presented here. As well as focusing on improving the hardware subsystems so that, in-situ in the robots, they behave as optimally as in the bench tests from which the simulation environment was established, many of the proposed changes focus on making the modules more modular internally, an idea which is starting to emerge in the modular robotics community [199], enabling much easier access to components for fabrication, servicing and repair. With modular robots requiring large numbers of modules, each likely to be highly complex due to the functionality necessary within each one, anything which simplifies the production and maintenance of individual modules should serve to make large scale systems more practical.

The cross shape of the module exists primarily due to the need for a region between the plane of the wheels and the rear of the docking port faces to house motors, it was designed before the port PCBs were and hence they were fitted into its mechanical constraints. An alternative design could cut the arms off the cross to give a more square overall shape for the module, and in the process greatly reduce weight, and reduce the module lattice spacing, thereby making hinge operations less demanding in terms of torque and stresses on plastic gear teeth. This change of shape would also provide an opportunity to fit further gearing or larger motors to drive the wheels and improve their torque. Future wheel systems should be thoroughly tested in physical mock-ups of scenarios involving multiple modules dragging each other, to ensure an appropriate balance between a wheel's torque, a module's weight and the force required to drag a stationary wheel.

However, it should be noted that such redesigns would likely bring motors closer to the magnetometer's location. As the magnetic offset based calibration method described in Sec.(5.7.1) has proven effective in the current robot design, it is plausible to assume that it could overcome the distortions created by closer proximity to brushed DC motors so long as they did not expose the magnetometer chip to field strengths greater than its maximum reading ( $\pm 1300\mu$T [78] ) and saturate the output such that the effects of the earth's field could not be distinguished.

The wires running from the central PCB out to port 4 through the 2DoF hinge greatly complicate design and assembly of the modules, in the current design there is no scope for altering this. The complexity of this arrangement is further increased by having to squeeze the port-4-to-central-PCB wire bundle past the top of the battery pack, while staying within the profile of the upper curve of the central core printed part. Such difficulties fitting wiring into the tight volumes involved in modular robots are not a problem unique to the Omni-Pi-tent design [21]. A method of achieving this design change has been considered and its implications for other parts of the robot also factored in. Such a design change would greatly simplify assembling the robots, a task currently made painfully laborious by the need to run wires through a spatially restricted core area very early in the build process and hold Port 4 and the central core in the correct wire protecting relative orientation while the robot's sides are attached. This change would also simplify production of the robots by removing the need to, all to frequently, during construction repeat certain steps to swap out wires severed due to being caught between parts as they are slotted together.

The 5KHz Infrared systems used for docking guidance and proximity sensing would work better if using different modulations so that they could not interfere with one-another. This could be constructed using analogue switching ICs to enable each physically separated phototransistor's signal to be amplified via a different circuit, tuned for a different modulation frequency and gain, and then further switching to feed these to different ATMEGA ADC pins as necessary. This would also reduce the board space taken up, by not needing a separate array of resistors and capacitors for each phototransistor channel's amplifier, on the port PCBs as well as shortening soldering time. The freeing of board space and switching of ADC channels would also make available enough analogue GPIO inputs to handle proximity sensing in other directions than straight outward from the ports, allowing for sensors covering blindspots around the robot, and providing proximity sensing coverage useful in 3D tasks, such as being able to check if a floor or other surface is above or below or not when a module is held aloft by others. Extra analogue proximity channels could also allow for a non-contact method of confirming the state of docking ports more reliably than with microswitches.

The 38KHz communications have proven difficult to work with once multiple modules are sharing a space, mainly because the messages are so long in duration, compared to the rate at which they need to be broadcast, that they are likely to interfere with each other. Ideally a much higher modulation for the carrier wave would be used, allowing much faster message bursts and enabling many modules to talk within a region without there being much chance of message collisions. TV remote derived IR modulation modules do not exist above 56KHz, so a custom system would be required. Given the difficulties encountered in hardware implementations of the self-assembly and self-repair strategies due to attempting to provide this functionality over global Wi-Fi, this is a particularly important subsystem and one which I am presently working on developing.

All ATMEGA328P microcontrollers would be replaced with SMD versions of the same chips, this smaller form factor would allow the use of more ATMEGA chips for more separated low-level tasks. These would be programmed with ICSP headers, this may sacrifice some GPIO pins, but the possibility of extra chips more than makes up for this. The ICSP headers could be located so as to enable reprogramming of chips with adjusted firmware without having to disassemble the robot to reach DIP chips to remove and program in an offboard jig.

It however should be noted that the work of this thesis has proved that even with the limited capabilities of the Omni-Pi-tent platform it is still feasible to perform unique feats, such as docking between moving modules, and that at the fundamental level Dynamic Self-repair can be performed without requiring further capabilities so long as the subsystems providing these capabilities operate well enough.

## 8.4 Bringing Modular Robots out of the Lab

Modular robots have the potential for incredible capabilities, machines which can repair themselves when they encounter problems could truly revolutionise how we maintain the world around us. With the growth of ever more autonomous technologies, systems which can keep themselves in working order without the need for human intervention provide very strong candidates on which to establish a new “relationship" between man and machine. Combining this resilience with the adaptability which modularity also offers gives us machines which can work to keep other machines operating too, and which can expand our horizons across the solar system.

The technologies used to achieve the work in this thesis (3D printers, cheap low-power single-board computers... ) were once science-fiction, today they are commonplace, it is hard *not* to consider that some day modular robots will be commonplace too. Even if the more outlandish concepts of programmable matter and “buckets of stuff" [33] never come to pass, there is already science-fiction being written which speculates about the kinds of things which more recognisable modular robots may make possible [201].

Previous work in modular robotics [51] has developed some forms of self-repair strategies but was often hindered by the limitations of the robots used and the lack of analogues which they provide for the expected requirements of a field ready modular robotic system. Omni-Pi-tent and the Dynamic Self-repair project have their limitations too, but in developing the hardware and software infrastructure to address the Dynamic Self-repair hypothesis, and in demonstrating what can be achieved and how new strategies can refine the performance of modular robots, I hope I have brought somewhat closer the day when such a promising new technology arrives in the wider world.

# Bibliography

[1] M.McCauley, BCM2835 library, August 2018 revision, `https://www.airspayce.com/mikem/bcm2835/`

[2] R.Groß, M.Bonani, F.Mondada, M.Dorigo, *Autonomous self-assembly in swarm-bots*, 2006, IEEE transactions on robotics, **22(6)**, p.1115-1130.

[3] N.Mathews, *Beyond self-assembly: Mergeable nervous systems, spatially targeted communication, and supervised morphogenesis for autonomous robots*, 2018, PhD thesis, Université Libre de Bruxelles, IRIDIA

[4] C.Liu, M.Yim, *Configuration Recognition with Distributed Information for Modular Robots*, 2017, IFRR International Symposium on Robotics Research, Puerto Varas, Chile

[5] C.Liu, Q.Lin, H.Kim, M.Yim, *SMORES-EP, a Modular Robot with Parallel Self-assembly*, 2021, ArXiv preprint, ArXiv:2104.00800

[6] P.Swissler, M. Rubenstein, *FireAnt3D: a 3D self-climbing robot towards non-latticed robotic self-assembly*, 2020, In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), p.3340-3347

[7] R.Groß, M.Dorigo, *Self-assembly at the macroscopic scale*, 2008, Proceedings of the IEEE 96, No.**9**: p.1490-1508.

[8] C.Schlenoff, E.Messina, *A robot ontology for urban search and rescue*, 2005, Proceedings of the 2005 ACM workshop on Research in knowledge representation for autonomous systems, p. 27-34

[9] CKBOT website, accessed 16/01/2018, `http://www.modlabupenn.org/2009/09/24/ckbot/`

[10] V.Zykov, A.Chan, H.Lipson, *Molecubes: An open-source modular robotics kit*, 2007, In IROS-2007 Self-Reconfigurable Robotics Workshop, p. 3-6

[11] V.Zykov, W.Phelps, N.Lassabe, H.Lipson, *Molecubes extended: Diversifying capabilities of open-source modular robotics*, 2008, In IROS-2008 Self-Reconfigurable Robotics Workshop, p. 22-26

[12] T.Zhang, W.Zhang, M.M.Gupta, *Resilient robots: concept, review, and future directions*, 2017, Robotics, **6(4)**, p.22

[13] S.Murata, E.Yoshida, A.Kamimura, H.Kurokawa, K.Tomita, S.Kokaji, *M-TRAN: Self-Reconfigurable Modular Robotic System*, IEEE/ASME Transactions on Mechatronics, 2002, Vol.**7**:431-441

[14] M.Dorigo, D.Floreano, L.M.Gambardella, et al. ,*Swarmanoid A Novel Concept for the Study of Heterogeneous Robotic Swarms*, IEEE Robotics and Automation Magazine, 2013, Vol.**December**:60-71

[15] J.Baca, A.Yerpes, M.Ferre, J.A.Escalera, R.Aracil, *Modelling of modular robot configurations using graph theory*, 2008, In International Workshop on Hybrid Artificial Intelligence Systems, p. 649-656, Springer, Berlin

[16] H.Kurokawa, K.Tomita, A.Kamimura, S.Kokaji, T.Hasuo, S.Murata, *Distributed self-reconfiguration of M-TRAN III modular robotic system*, 2008, The International Journal of Robotics Research, **27(3-4)**, p.373-386

[17] N.Pouliot, P.L.Richard, S.Montambault, *LineScout technology opens the way to robotic inspection and maintenance of high-voltage power lines*, 2015, IEEE Power and Energy Technology Systems Journal, **2(1)**, p.1-11

[18] E.H.Ostergaard, K.Kassow, R.Beck, H.H.Lund, *Design of the ATRON lattice-based self-reconfigurable robot*, 2006, Autonomous Robots, **21,2**, p.165-183, Springer

[19] R.French, A.Cryer, G.Kapellmann-Zafra, H.Marin-Reyes, *Evaluating the radiation tolerance of a robotic finger*. July 2018, In Proc. Towards Autonomous Robotic Systems, p. 103-111, Springer, Cham.

[20] Y.K.Demirel, D.Uzun, Y.Zhang, H.C.Fang, A.H.Day, O.Turan, *Effect of barnacle fouling on ship resistance and powering*, 2017, Biofouling, **33(10)**, p.819-834

[21] C.Parrott, *A Hybrid and Extendable Self-Reconfigurable Modular Robotic System*, 2016, PhD thesis, University of Sheffield

[22] SMORES video, timestamp 1:03, accessed 16/01/2021, `https://www.youtube.com/watch?v=y8wfs6d_r6U`

[23] Lua programming language website, accessed 22/02/2018, `https://www.lua.org/docs.html`

[24] N.Jakobi, P.Husbands, I.Harvey, *Noise and the reality gap: The use of simulation in evolutionary robotics*, 1995, Advances in Artificial Life: 3rd European Conference on Artificial Life, Springer

[25] R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, `https://www.R-project.org/, (2016)`

[26] N.Mathews, A.L.Christensen, R.O'Grady, F.Mondada, M.Dorigo, *Mergeable nervous systems for robots*, Nature communications, 2017, Vol.**8**

[27] W.Liu, A.F.T.Winfield, *Autonomous Morphogenesis in Self-assembling Robots using IR-based Sensing and Local Communications*,2010, Swarm Intelligence, Springer, p.107-118

[28] E.Rohmer, S.P.N.Singh, M.Freese, *V-REP: a Versatile and Scalable Robot Simulation Framework*, 2013, International Conference on Intelligent Robots and Systems (IROS)

[29] W.Liu, A.F.T.Winfield, *Distributed Autonomous Morphogenesis in a Self-Assembling Robotic System*, 2012, Morphogenetic Engineering: Toward Programmable Complex Systems, Ed. R.Doursat, Springer

[30] M.R.Jahanshahi, W.M.Shen, T.G.Mondal, M.Abdelbarr, S.F.Masri, U.A.Qidwai, *Reconfigurable Swarm Robots for Structural health Monitoring - A Brief Review*, 2017, International Journal of Intelligent Robotics and Applications, Vol.**1**:287-305

[31] R.H.Peck, J.Timmis, A.M.Tyrrell, *Towards Self-repair with Modular Robots during Continuous Motion*, 2018, In Proc. Towards Autonomous Robotic Systems, Bristol

[32] T.Fukuda, S.Nakagawa, *Dynamically Reconfigurable Robotic System*, 1988, Proceedings. IEEE International Conference on Robotics and Automation, Philadelphia, Vol.**3**:1581-1586

[33] M.Yim, W.M.Shen, B.Salemi, D.Rus. M.Moll. H.Lipson, E.Klavins, G.S.Chirikjian, *Modular Self-Reconfigurable Robot Systems, challenges and opportunities for the future*,2007, IEEE Robotics and Automation Magazine, Vol.**March**:39-45

[34] P.Levi, S.Kernbach(eds.), *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, 2010, Springer, doi.org/10.1007/978-3-642-11692-6

[35] J.Davey, N.Kwok, M.Yim, *Emulating Self-reconfigurable Robots - Design of the SMORES System*, 2012, IEEE/RSJ International Conference on Intelligent Robots and Systems, Vllamoura, p.4464-4469

[36] S.Kernbach, E.Meister, O.Scholz, R.Humza, J.Liedke, L.Ricotti, J.Jemai, J.Havlik, W.Liu, *Evolutionary Robotics: The Next-Generation-Platform for On-line and On-board Artificial Evolution*, 2009, IEEE Congress on Evolutionary Computation, Trondheim, p.1079-1086

[37] J.Zhao, X.Cui, Y.Zhu, S.Tang, *A New Self-Reconfigurable Modular Robotic System UBot: Multi-mode locomotion and Self-reconfiguration*, 2011, IEEE Conference on Robotics and Automation, Shanghai, p.1020-1025

[38] Y.Zhu, H.Jin, X.Zhang, J.Yin, P.Liu, J.Zhao, *A multi-sensory autonomous docking approach for a self-reconfigurable robot without mechanical guidance*, 2014, International Journal of Advanced Robotic Systems, **11(9)**, p.146

[39] Y.Zhu, D.Bie, X.Wang, Y.Zhang, H.Jin, J.Zhao, *A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata*, 2017, Journal of Parallel Distributed Computing, Vol.**102**:80-90

[40] A.Lindenmayer, *Mathematical models for cellular interactions in development I. Filaments with one-sided inputs*, 1968, Journal of Theoretical Biology, Vol.**18**, No.**3**, p. 280-299, DOI:10.1016/0022-5193(68)90079-9

[41] R.H.Peck, J.Timmis, A.M.Tyrrell, *Omni-Pi-tent: An Omnidirectional Modular Robot With Genderless Docking*, 2019, Proceedings. Towards Autonomous Robotic Systems, London,doi.org/10.1007/978-3-030-25332-5_27

[42] C.Parrott, T.J.Dodd, R.Gro$\beta$, *HyMod: A 3-DOF Hybrid Mobile and Self-Reconfigurable Modular Robot and its Extensions* , DARS, 2016, p.401-414, Springer

[43] A.Brunete, A.Ranganath, S.Segovia, J.P. de Frutos, M.Hernando, E.Gambao, *Current trends in reconfigurable modular robots design.*, 2017, International Journal of Advanced Robotic Systems, doi.org/10.1177/1729881417710457

[44] Fukushima response robotic operator's blog, accessed 02/10/2019, `https://spectrum.ieee.org/automaton/robotics/industrial-robots/fukushima -robot-operator-diaries`

[45] J.Liu, X.Zhang, G.Hao, *Survey on research and development of reconfigurable modular robots*, 2016, Advances in Mechanical Engineering, Vol.**8**:1-21

[46] A.Kawakami, A.Torii, K.Motomura, S.Hirose, *SMC Rover : Planetary Rover with transformable wheels*, 2002, SICE Annual Conference, Osaka, Vol.**1**:157-162

[47] K.Tomita, S.Murata, H.Kurokawa, E.Yoshida, S.Kokaji, *A self-assembly and self-repair method for a distributed mechanical system*, 1999, IEEE Transactions on Robotics and Automation, Vol.**15(6)**:1035-1045

[48] R.Fitch, D.Rus, M.Vona, *A basis for self-repair robots using self-reconfiguring crystal modules*, July 2000, In. Intelligent Autonomous Systems, Vol. **6**, p. 903-910, IOS Press

[49] S.Murata, E.Yoshida, H.Kurokawa, K.Tomita, S.Kokaji, *Self-repairing mechanical systems*, 2001, Autonomous Robots, **10(1)**, p.7-21

[50] J.Baca, S.G.M.Hossain, P.Dasgupta, C.A.Nelson, A.Dutta, *ModRED:Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration*,2014, Robotics and Autonomous Systems, Vol.**62**:1002-1015

[51] L.J.Murray, *Fault Tolerant Morphogenesis in Self-Reconfigurable Modular Robotic Systems*, 2013, PhD Thesis, University of York

[52] M.Rubenstein, A.Cornejo, R.Nagpal, *Programmable self-assembly in a thousand-robot swarm*, 2014, AAAS Science mag., Vol.**345**, Iss.**6198**:795-799

[53] R.Doursat, *Organically Grown Architectures: Creating Decentralized, Autonomous Systems by Embryomorphic Engineering*,2008, Understanding Complex Systems, Springer, Vol.**21**:167-199.

[54] R.Nagpal, *Programmable self-assembly: Constructing global shape using biologically-inspired local interactions and origami mathematics*, 2001, PhD thesis, Massachusetts Institute of Technology

[55] K.Stoy, R.Nagpal, *Self-repair through scale independent self-reconfiguration*, March 2004, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), Vol.**2**, p. 2062-2067, IEEE

[56] M.Rubenstein, W.M.Shen, *Scalable self-assembly and self-repair in a collective of robots*, 2009, IEEE/RSJ international conference on Intelligent robots and systems, p.1484-1489, IEEE

[57] D.J.Arbuckle, A.A.Requicha, *Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations*, 2010, Autonomous Robots, **28(2)**, p.197-211

[58] J.Werfel, *Biologically realistic primitives for engineered morphogenesis*, 2010, Proceedings. 7th international conference on Swarm intelligence ANTS 2010, p.131–142, Springer, Belgium

[59] K.Støy, *Using cellular automata and gradients to control self-reconfiguration*, 2006, Robotics and Autonomous Systems, Vol.**54**, p.135-141

[60] A.L.Christensen, R.O'Grady, M.Dorigo, *SWARMORPH-script: a language for arbitrary morphology generation in self-assembling robots*, 2008, Swarm Intelligence, Vol.**2**, p.143-165, doi.org/10.1007/s11721-008-0012-6

[61] S.Ivaldi, V.Padois, F.Nori, *Tools for dynamics simulation of robots: a survey based on user feedback*, 2014, ArXiv

[62] Y.Zhang, G.Song, S.Liu, G.Qiao, J.Zhang, H.Sun, *A Modular Self-Reconfigurable Robot with Enhanced Locomotion Performances: Design, Modelling, Simulations, and Experiments*, 2016, Journal of Intelligent and Robotic Systems, Vol.**81**:377-393

[63] S.Popesku, E.Meister, F.Schlachter, P.Levi, "*Active Wheel - An Autonomous Modular Robot*, Proceedings. 2013 International Conference on Robotics, Automation and Mechatronics (RAM), Manila

[64] V-REP features guide, accessed 27/02/2018, `http://www.coppeliarobotics.com/assets/v-repoverviewpresentation.pdf`

[65] V-REP user manual pages, accessed 27/02/2018, `http://www.coppeliarobotics.com/helpFiles/index.html`

[66] L.Pitonakova swarm robotics blog, accessed 01/03/2018, `http://lenkaspace.net/blog/show/120`

[67] L.Pitonakova, M.Giuliani, A.Pipe, A.Winfield,*Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators*, 2018, In: Giuliani M., Assaf T., Giannaccini M. (eds) Towards Autonomous Robotic Systems. TAROS 2018. LNCS, Vol. **10965**, Springer, Cham. doi.org/10.1007/978-3-319-96728-8_30

[68] V-REP signals documentation, accessed 05/03/2018, `http://www.coppeliarobotics.com/helpFiles/en/meansOfCommunication.htm`

[69] Example omni-wheel model by "kaveenliyanage" at V-REP forums, accessed 05/03/2018, `https://www.dropbox.com/s/gti6x0yjjs2emni/omni3wheel_model.ttt?v=0mcng`

[70] V-REP dynamic simulation tips, see Consideration 6 and onwards, accessed 06/03/2018, `http://www.coppeliarobotics.com/helpFiles/en/designingDynamicSimulations.htm`

[71] V-REP API function list, accessed 07/03/2018, `http://www.coppeliarobotics.com/helpFiles/en/apiFunctionListCategory.htm`

[72] L.Zlajpah, *Simulation in robotics*, Mathematics And Computers In Simulation, 2008, Vol.**79**:879-897

[73] S.G.M.Hossain, C.A.Nelson, P.Dasgupta, *RoGenSid: A Rotary Plate Genderless Single-Sided Docking Mechanism for Modular Self-Reconfigurable Robots*, 2013, In: ASME 2013 IDETC/CIE, V06BT07A011–V06BT07A011

[74] A.E.Eiben, N.Bredeche, M.Hoogendorn, J.Stradner, J.Timmis, A.M.Tyrell, A.Winfield, *The Triangle of Life: Evolving Robots in Real-time and Real-space*, 2013, European Conference on Artificial Life, Taormina, p.1056-1063

[75] G.Liang, H.Luo, M.Li, H.Qian, T.L.Lam, *FreeBOT: A Freeform Modular Self-reconfigurable Robot with Arbitrary Connection Point - Design and Implementation*, 2020, IROS 2020, p.6505-6513

[76] D.Saldana, B.Gabrich, G.Li, M.Yim, V.Kumar, *Modquad: The flying modular structure that self-assembles in midair*, May 2018, In 2018 IEEE International Conference on Robotics and Automation (ICRA), p. 691-698, IEEE

[77] Pololu D24V50F5 5V 5A regulator data-sheet, accessed 17/02/2021, `https://www.pololu.com/product/2851`

[78] Bosch SensorTec BNO055 data-sheet, June 2016 revision, `https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf`

[79] Article on interfacing Raspberry Pi to BNO 055, accessed 18/02/2021, `https://learn.adafruit.com/bno055-absolute-orientation-sensor-with-raspberry-pi-and-beaglebone-black/hardware`

[80] Article on Raspberry Pi I2C clock stretching bug, accessed 31/08/2018, `http://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html`

[81] eSpeak NG Text-to-Speech, accessed 18/02/2021, `https://github.com/espeak-ng/espeak-ng`

[82] D.J.Christensen, *Experiments on fault-tolerant self-reconfiguration and emergent self-repair*, April 2007, In 2007 IEEE Symposium on Artificial Life, p. 355-361, IEEE

[83] J.Liedke, R.Matthias, L.Winkler, H.Wörn, *The collective self-reconfigurable modular organism (CoSMO)*, July 2013, In 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, p. 1-6, IEEE

[84] Viking Cluster, high performance compute facility provided by the University of York. With thanks to the University of York High Performance Computing service, Viking and the Research Computing team, `https://wiki.york.ac.uk/display/RCS/Viking+-+University+of+York+Research+Computing+Cluster`

[85] F.Mondada, M.Bonani, X.Raemy, J.Pugh, C.Cianci, A.Klaptocz, S.Magnenat, J.-C.Zufferey, D.Floreano, A.Martinoli, *The e-puck, a Robot Designed for Education in Engineering*, 2009, Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, **1(1)**, p.59-65

[86] Model creator's webpage L.Randall “Crazy Eyes" / “Crazy Eyes NZ", accessed 05/04/2021, `https://3dwarehouse.sketchup.com/user/0374962399039393497241605/CrazyEyesNZ`

[87] Magneto v1.2 C implementation, 2013, Michel Boulanger “BerMerlin", accessed 03/03/2021, `https://sites.google.com/site/sailboatinstruments1/c-language-implementation`

[88] Pololu LSM303 Library, with tilt compensation functionality, 2013, accessed 20/03/2021, `https://github.com/pololu/lsm303-arduino`

[89] Q.Li, J.G.Griffiths, *Least squares ellipsoid specific fitting*, 2004, Geometric Modeling and Processing, Proceedings, Beijing, p.335-340, DOI: 10.1109/GMAP.2004.1290055

[90] C.Parrott, T.J.Dodd, R.Gro$\beta$, *HiGen: A high-speed genderless mechanical connection mechanism with single-sided disconnect for self-reconfigurable modular robots*, 2014, IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, p.3926-3932, doi: 10.1109/IROS.2014.6943114

[91] T.Tosun, D.Edgar, C.Liu, T.Tsabedze, M.Yim, *PaintPots: Low cost, accurate, highly customizable potentiometers for position sensing*, 2017, IEEE International Conference on Robotics and Automation (ICRA), p.1212-1218, doi: 10.1109/ICRA.2017.7989144.

[92] W.M.Shen, R.Kovac, M.Rubenstein, *SINGO: A single-end-operative and genderless connector for self-reconfiguration, self-assembly and self-healing*, 2009, Proc., 2009 IEEE International Conference on Robotics and Automation, p.4253–4258

[93] J.Seo, J.Paik, M.Yim, *Modular reconfigurable robotics*, 2019, Annual Review of Control, Robotics, and Autonomous Systems, **2**, p.63-88

[94] W.M.Shen, F.Hou, M.Rubenstein, H.Chiu, A.Kamimura, *Recent progress of SuperBot*, 2010, Modular Robots: The State of the Art, p.13–16

[95] W.Saab, P.Ben-Tzvi, *A Genderless Coupling Mechanism With Six-Degrees-of-Freedom Misalignment Capability for Modular Self-Reconfigurable Robots*, 2016, ASME Journal of Mechanisms and Robotics, Vol.**8**, doi:10.1115/1.4034014

[96] P.Moubarak, P.Ben-Tzvi, *Modular and reconfigurable mobile robotics*, 2012, Robotics and autonomous systems, **60(12)**, p.1648-1663

[97] W.Saab, P.Ben-Tzvi, *A Tristate Rigid Reversible and Non-Back-Drivable Active Docking Mechanism for Modular Robotics*, June 2014, IEEE/ASME Transactions on Mechatronics, Vol.**19**, no. **3**, p.840-851, doi:10.1109/TMECH.2013.2261531.

[98] Physical measurements of a SYMBRION scout module

[99] H.Wei, Y.Chen, J.Tan, T.Wang, *Sambot: A Self-Assembly Modular Robot System*, 2011, IEEE/ASME Transactions on Mechatronics, Vol.**16**:745-757

[100] W.Tan, H.Wei, B.Tang, *SambotII: A New Self-Assembly Modular Robot Platform Based on Sambot*,2018, MDPI Applied Sciences, Vol.**8**, No.**10**, doi.org:10.3390/app8101719

[101] T.Tosun, J.Daudelin, G.Jing, H.Kress-Gazit, M.Campbell, M.Yim, *Perception-Informed Autonomous Environment Augmentation with Modular Robots*, 2018, IEEE International Conference on Robotics and Automation (ICRA), Brisbane, p. 6818-6824, doi:10.1109/ICRA.2018.8463155

[102] S.Murata, K.Kakomura, H.Kurokawa, *Docking Experiments of a Modular Robot by Visual Feedback*, 2006, IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, p.625-630, doi:10.1109/IROS.2006.282545

[103] T. Tosun, J. Davey, C. Liu, M. Yim, *Design and characterization of the ep-face connector*, 2016, IROS, IEEE

[104] D.J.Christensen, J.C.Larsen, K.Stoy, *Fault-tolerant gait learning and morphology optimization of a polymorphic walking robot*, 2014, Evolving Systems, Vol.**5**:21-32

[105] V.Vonásek, D.Oertel, S.Neumann, H.Wörn, *Failure recovery for modular robot movements without reassembling modules*, 2015, In 10th International Workshop on Robot Motion and Control (RoMoCo), p.136-141, IEEE

[106] D.J.Christensen, U.P.Schultz, K.Stoy, *A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots*, 2013, Robotics and Autonomous Systems, Vol.**61**:1021-1035

[107] J.Neubert, H,Lipson, *Soldercubes: a self-soldering self-reconfiguring modular robot system*, 2016, Autonomous Robotics, Vol.**40**:139-158

[108] L.E.Parker, *ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation*, 1998, IEEE transactions on robotics and automation, Vol.**14**, No.**2**:220-240

[109] M.Nilsson, *Connectors for self-reconfiguring robots*, 2002, IEEE/ASME Transactions on mechatronics 7, no. **4** (2002): 473-474.

[110] V.T.Le, T.D.Ngo, *Virtual Pheromone Based Network Flow Control For Modular Robotic Systems*, 2020 Electronics, Vol.**9**, No.**(**3), p481

[111] J.Davey, J.Sastra, M.Piccoli, M.Yim, *Modlock: A manual connector for reconfigurable modular robots*, 2012, In IEEE/RSJ International Conference on Intelligent Robots and Systems, p.3217-3222, IEEE

[112] A.Lyder, R.F.M.Garcia, K.Stoy, *Genderless connection mechanism for modular robots introducing torque transmission between modules*, 2010, In Proceedings of the ICRA Workshop on Modular Robots, State of the Art, p.77-81

[113] J.Liedke, H.Wörn, *CoBoLD—A bonding mechanism for modular self-reconfigurable mobile robots*, 2011, In IEEE International Conference on Robotics and Biomimetics, p.2025-2030, IEEE

[114] K.H.Kim, H.D.Choi, S.Yoon, K W.Lee, H.S.Ryu, C.K.Woo, Y.K.Kwak, *Development of docking system for mobile robots using cheap infrared sensors*, 2005, In Proceedings of the 1st International Conference on Sensing Technology, p.287-291

[115] P.M.Vaz, R.Ferreira, V.Grossmann, M.I.Ribeiro, *Docking of a mobile platform based on infrared sensors*, 1997, In ISIE'97 Proceeding of the IEEE International Symposium on Industrial Electronics, Vol.**2**, p.735-740, IEEE

[116] S.Kernbach, E.Meister, F.Schlachter, K.Jebens, M.Szymanski, J.Liedke, D.Laneri, L.Winkler, T.Schmickl, R.Thenius, P.Corradi, *Symbiotic robot organisms: REPLICATOR and SYMBRION projects*, August 2008, In Proceedings of the 8th workshop on performance metrics for intelligent systems, p. 62-69

[117] J.W.Romanishin, K.Gilpin, D.Rus, *M-Blocks: Momentum-driven, Magnetic Modular Robots*, 2013, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo, p.4288-4295

[118] A.Sproewitrz, A.Billard, P.Dillenbourg, A.J.Ijspreet, *Roombots - Mechanical Design of Self-Reconfiguring Modular Robots for Adaptive Furniture*, 2009, IEEE International Conference on Robotics and Automation, Kobe, p.4259-4264

[119] D.Brandt, D.J.Christensen, H.H.Lund, *ATRON robots: versatility from self-reconfigurable modules*, 2007, In International Conference on Mechatronics and Automation, p.26-32, IEEE

[120] Multicomp Pro DM1-01P-30-3 microswitch data-sheet, 2020 revision, `https://www.farnell.com/datasheets/2945636.pdf`

[121] Vishay CQY36N IR LED data-sheet, May 2011 revision, `https://www.vishay.com/docs/81001/cqy36n.pdf`

[122] Kingbright L-93DP3C phototransistor data-sheet, August 2012 revision, `https://www.farnell.com/datasheets/1683591.pdf`

[123] Vishay TSOP38438 data-sheet, June 2014 revision, `https://www.vishay.com/docs/82491/tsop382.pdf`

[124] W.Liu, A.F.T.Winfield, *Implementation of an IR approach for autonomous docking in a self-configurable robotics system*, September 2009, Proc. Towards Autonomous Robotic Systems, Londonderry, p.251–258

[125] N.Tan, A.A.Hayat, M.R.Elara, K.L.Wood, *A Framework for Taxonomy and Evaluation of Self-Reconfigurable Robotic Systems*, January 2020, IEEE Access, Vol.**8**

[126] Pololu U3V12F12 12V regulator, accessed 21/07/2021, `https://www.pololu.com/product/2117`

[127] A.Roennau, F.Sutter, G.Heppner, J.Oberlaender, R.Dillmann, *Evaluation of Physics Engines for Robotic Simulations with a Special Focus on the Dynamics of Walking Robots*, November 2013, ICAR, p.1,7, 25,29, DOI: 10.1109/ICAR.2013.6766527

[128] S.Koos, J-B.Mouret, S.Doncieux, *The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics*, February 2013, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, Vol.**17**, No.**1**, DOI:10.1109/TEVC.2012.2185849

[129] S.Koos, J-B.Mouret, S.Doncieux, *Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers*, July 2010, GECCO 2010, Oregon, USA, ACM 978-1-4503-0072-8/10/07

[130] C.Hartland, N.Bredeche, *Evolutionary Robotics, Anticipation and the Reality Gap*, December 2006, IEEE ICRB, Kunming, China

[131] C.Pepper, S.Balakirsky, C.Scrapper, *Robot Simulation Physics Validation*, 2007 Workshop on Performance Metrics for Intelligent Systems, p97-104. DOI:10.1145/1660877.1660890

[132] P.Abbeel, A.Coates, M.Quigley, A.Ng, *An application of reinforcement learning to aerobatic helicopter flight*, 2007, Proc. NIPS , p.1.

[133] J. Bongard, H. Lipson, *Once more unto the breach: Co-evolving a robot and its simulator*, 2004, 9th Artificial Life conference, p.57

[134] A.Ligot, M.Birattari, *Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms*, 2020, Swarm Intell 14, 1–24 (2020), doi.org/10.1007/s11721-019-00175-w

[135] P.Flocchini, G.Prencipe, N.Santoro P.Widmayer, *Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots*, 1999, In: Algorithms and Computation. ISAAC 1999. Lecture Notes in Computer Science, vol.**1741**, Springer, doi.org/10.1007/3-540-46632-0_10

[136] M.Yim, D.Duff, K.Roufas, *PolyBot: a modular reconfigurable robot*, 2000, IEEE International Conference on Robotics and Automation ICRA2000, p.514–520

[137] D.Duff, M.Yim, K.Roufas, *Evolution of polybot: A modular reconfigurable robot*, 2001, In Proc. of the Harmonic Drive Intl. Symposium, Nagano, Japan.

[138] M.Yim, K.Roufas, D.Duff, Y.Zhang, C.Eldershaw, S.Homans, *Modular reconfigurable robots in space applications*, 2003, Autonomous Robots, **14(2)**, 225-237

[139] A.Gasparetto, L.Scalera, *From the Unimate to the Delta robot: the early decades of Industrial Robotics*, 2019, In Explorations in the History and Heritage of Machines and Mechanisms, p.284-295, Springer, Cham.

[140] J.Iqbal, A.M.Tahir, R. ul Islam, *Robotics for nuclear power plants—challenges and future perspectives*, September 2012, 2nd international conference on applied robotics for the power industry (CARPI) (p. 151-156). IEEE.

[141] P.Debenest, M.Guarnieri, S.Hirose, *PipeTron series-Robots for pipe inspection*, October 2014, In Proceedings of the 2014 3rd international conference on applied robotics for the power industry, p.1-6, IEEE

[142] T.A.Ferguson, L.Lu, *Fault Tree Analysis for an Inspection Robot in a Nuclear Power Plant*, 2017, IOP Conference Series; Materials Science and Engineering, vol.**235**, No.**1**, p.012003

[143] D.Chatziparaschis, M.G.Lagoudakis, P.Partsinevelos, *Aerial and Ground Robot Collaboration for Autonomous Mapping in Search and Rescue Missions*, 2020, Drones, **4(4)**, p.79

[144] P.Debenest, M.Guarnieri, *Expliner—From prototype towards a practical robot for inspection of high-voltage lines*, October 2010, 1st International Conference on Applied Robotics for the Power Industry, p. 1-6, IEEE

[145] L.Garattoni, M. Birattari, *Swarm robotics*, 1999, Wiley Encyclopedia of Electrical and Electronics Engineering, p.1-19

[146] P.Levi, S.Kernbach (Eds.), *Symbiotic multi-robot organisms: reliability, adaptability, evolution*, 2010, Vol.**7**, Springer Science and Business Media

[147] SYMBRION project overview and costs, accessed 24/05/2021, `https://cordis.europa.eu/project/id/216342`

[148] S.B.Stancliff, J.M.Dolan, A.Trebi-Ollennu, *Mission reliability estimation for multirobot team design*, 2006, IEEE/RSJ International Conference on Intelligent Robots and Systems, p.2206-2211

[149] A.F.T.Winfield, J.Nembrini, *Safety in numbers: fault-tolerance in robot swarms*, 2006, International Journal of Modelling, Identification and Control, **1(1)**, p.30-37

[150] A.G.Millard, J.Timmis, A.F.T.Winfield, *Towards exogenous fault detection in swarm robotic systems*, 2013, Towards Autonomous Robotic Systems, Oxford, p.429-430, Springer

[151] J. O'Keeffe, *Immune-Inspired Fault Diagnosis for Robot Swarms*, 2019, PhD thesis, University of York

[152] O. O. Oladiran, *Fault Recovery in Swarm Robotics Systems using Learning Algorithms*, 2019, PhD thesis, University of York

[153] F.Wang, S.Qi, J.Li, *An analysis of time-delay for remote piloted vehicle*, 2017, In MATEC Web of Conferences (Vol.**114**, p.04012), EDP Sciences

[154] R.D.Madder, S.VanOosterhout, A.Mulder, J.Bush, S.Martin, A.J.Rash, J.M.Tan II, J.L.Parker, A.Kalafut, Y.Li, N.Kottenstette, P.Bergman, B.Nowak, *Network latency and long-distance robotic telestenting: Exploring the potential impact of network delays on telestenting performance*, 2020, Catheterization and Cardiovascular Interventions, **95(5)**, p.914-919

[155] T.K.Tucci, B.Piranda, J.Bourgeois, *A distributed self-assembly planning algorithm for modular robots*, July 2018, International Conference on Autonomous Agents and Multiagent Systems

[156] R.C.Arkin, *Behaviour-based Robotics*, May 1998, MIT Press, ISBN 9780262011655

[157] D.Rus, Z.Butler, K.Kotay, M.Vona, *Self-Reconfiguring Robots*,2002 Communications of the ACM, Vol.**45**, No. **3**:39-45

[158] D.Crestani, K.Godary-Dejean, L.Lapierre, *Enhancing fault tolerance of autonomous mobile robots*, 2015, Robotics and Autonomous Systems, Vol,**68**, p.140-155, ISSN 0921-8890, doi.org/10.1016/j.robot.2014.12.015

[159] N.Mathews, A.L.Christensen, A.Stranieri, A. Scheidler, M.Dorigo,*Supervised morphogenesis: Exploiting morphological flexibility of self-assembling multirobot systems through cooperation with aerial robots*, 2019, Robotics and Autonomous Systems, Vol.**112**, p.154-167, ISSN 0921-8890, doi.org/10.1016/j.robot.2018.11.007

[160] J. Meyer, *The Impact of the Robot's Morphology in the Collective Transport*,2019, Proceedings. Towards Autonomous Robotic Systems, London

[161] Y.Zhang, J.Jiang, *Bibliographical review on reconfigurable fault-tolerant control systems*, 2008, Annual Reviews in Control, Vol.**32**, Iss.**2**, p.229-252, ISSN 1367-5788,doi.org/10.1016/j.arcontrol.2008.03.008

[162] J.Jiang, X.Yu, *Fault-tolerant control systems: A comparative study between active and passive approaches*, 2012, Annual Reviews in Control, Vol.**36**, Iss.**1**, p.60-72, ISSN 1367-5788, doi.org/10.1016/j.arcontrol.2012.03.005

[163] A.Christensen, R.O'Grady, M.Birattari, M.Dorigo, *Exogenous fault detection in a collective robotic task*, 2007, Advances in Artificial Life, LNCS Vol.**4648**, p.555–564, Springer

[164] J. Carlson, R. R. Murphy, *How UGVs physically fail in the field*, June 2005, IEEE Transactions on Robotics, Vol.**21**, No.**3**, p.423-437, doi: 10.1109/TRO.2004.838027.

[165] P.C.Bennett, L.D.Posey, *RHOBOT: Radiation hardened robotics*, 1997, No.**SAND-97-2405**, Sandia National Labs, Albuquerque, USA

[166] M.Yim, B.Shirmohammadi, J.Sastra, M.Park, M.Dugan, C.J.Taylor, *Towards Robotic Self-reassembly After Explosion*, 2007, IEEE/RSJ International Conference on Intelligent Robots and Systems, p.2767-2772

[167] D.T.Packard, *Dual Drive Actuators*, 1982, 16th Aerospace Mechanisms Symposium, p.13-14

[168] E.Sahin, W.M.Spears, *Swarm robotics: SAB 2004 international workshop*, July 2004, Santa Monica, USA, revised selected papers, LNCS Vol.**3342**, Springer, ISBN 9783540242963

[169] S.S.Reddy, CH.Abhimanyu, R.Godiyal, T.Zodage, T.Rane, *2DxoPod-A Modular Robot for Mimicking Locomotion in Vertebrates*, 2021, Journal of Intelligent and Robotic Systems, **101**, No.**1** (2021): 1-16

[170] M.Yim, K.Roufas, D.Duff, Y.Zhang, C.Eldershaw, s.Homans, *Modular Reconfigurable Robots in Space Applications*, 2003, Autonomous Robots, Vol.**14**, p.225-237, DOI:10.1023/A:1022287820808.

[171] M.D.Hancher, G.Hornby,*A modular robotic system with applications to space exploration*, 2006, 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06), 10.1109/SMC-IT.2006.9.

[172] L.Pedersen, D.Kortenkamp, D.Wettergreen, I.Nourbakhsh, *Space robotics technology assessment report*, December 2002, NASA Exploration Team (NEXT), Technical Report

[173] J.P.Grotzinger, J.Crisp, A.R.Vasavada, et. al., *Mars Science Laboratory Mission and Science Investigation*, 2012, Space Science Review, Vol.**170**, p.5–56, doi.org/10.1007/s11214-012-9892-2

[174] Robotic inspection article, PetroBot project, accessed 19/05/2021, `http://petrobotproject.eu/robotic-inspection/`

[175] Launch of ELSA-d in-orbit servicing mission, Astroscale, accessed 21/05/2021, `https://astroscale.com/astroscale-celebrates-successful-launch-of-elsa-d/`

[176] H.Myung, S.Lee, B.Lee, *Paired structured light for structural health monitoring robot system*, 2011, Structural Health Monitoring, **10(1)**, p.49-64

[177] M.Wilde, J.Harder, E.Stoll, *On-Orbit Servicing and Active Debris Removal: Enabling a Paradigm Shift in Spaceflight*, 2019, Frontiers in Robotics and AI, Vol.**6**, p.136, DOI:10.3389/frobt.2019.00136

[178] ESA article on deorbiting and servicing missions, accessed 21/05/2021, `http://www.esa.int/Safety_Security/Clean_Space/ESA_s_e.Deorbit_debris_removal_mission_reborn_as_servicing_vehicle`

[179] T.Fukuda, Y.Kawauchi, *Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator*, 1990, Proc. IEEE International Conference on Robotics and Automation, p.662-667

[180] A.Lyder, R.F.M.Garcia, K.Stoy, *Mechanical design of odin, an extendable heterogeneous deformable modular robot*, 2008, IEEE/RSJ International Conference on Intelligent Robots and Systems, p.883-888

[181] U.Jahn, D.Heß, M.Stampa, A.Sutorma, C.Röhrig, P.Schulz, C.Wolff, *A Taxonomy for Mobile Robots: Types, Applications, Capabilities, Implementations, Requirements, and Challenges*, 2020, Robotics, **9(4)**, p.109, doi.org/10.3390/robotics9040109

[182] L.Seminara, P.Gastaldo, S.J.Watt, K.F.Valyear, F.Zuher, F.Mastrogiovanni, *Active Haptic Perception in Robots: A Review*, 2019, Frontiers in Neurorobotics, vol.**13**, DOI:10.3389/fnbot.2019.00053

[183] Our World in Data article on urbanisation, accessed 11/05/2021, `https://ourworldindata.org/urbanization`

[184] H.Rowshandel, *The development of an autonomous robotic inspection system to detect and characterise rolling contact fatigue cracks in railway track*, 2014, PhD thesis, University of Birmingham

[185] K.Gilpin, D.Rus, *Modular Robot Systems, From Self-Assembly to Self-Disassembly*, 2010, IEEE robotics and automation magazine, p.38-55

[186] A.Castano, W.M.Shen, P.Will, *CONRO: Towards deployable robots with inter-robots metamorphic capabilities*, 2000, Autonomous Robots, **8(3)**, p.309-324

[187] M.Rubenstein, K.Payne, P.Will, W.M.Shen, *Docking among independent and autonomous CONRO self-reconfigurable robots*, 2004, IEEE International Conference on Robotics and Automation (ICRA'04), Vol.**3**, p.2877-2882

[188] L.Brodbeck, F. Iida, *Automatic Real-World Assembly of Machine-Designed Structures*, 2014, IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, p.1221-1226

[189] S.Russo, K.Harada, T.Ranzani, L.Manfredi, C.Stefanini, A.Menciassi, P.Dario, *Design of a robotic module for autonomous exploration and multimode locomotion*, 2012, IEEE/ASME Transactions on Mechatronics, **18(6)**, p.1757-1766

[190] M.Yim, D.G.Duff, D. K.Roufas, *Modular reconfigurable robots, an approach to urban search and rescue*, 2000, 1st Intl. Workshop on Human-friendly Welfare Robotics Systems, p.69-76

[191] R.R.Murphy, S.Tadokoro, D.Nardi, A.Jacoff, P.Fiorini, H.Choset, A.M.Erkmen, *Search and rescue robotics*, 2008, Springer Handbook of Robotics, Springer, p.1151–1173

[192] H.Wei, D.Li, J.Tan, T.Wang, *The distributed control and experiments of directional self-assembly for modular swarm robots*, October 2010, IEEE/RSJ International Conference on Intelligent Robots and Systems, p.4169-4174, IEEE

[193] K.Stoy, D.J.Christensen, D.Brandt, M.Bordignon, U.P.Schultz, *Exploit morphology to simplify docking of self-reconfigurable robots*, 2009, In Distributed Autonomous Robotic Systems 8, p.441-452, Springer, Berlin, Heidelberg

[194] J.D.Evans, *Straightforward statistics for the behavioral sciences*, 1996, Thomson Brooks/Cole Publishing Co.

[195] C.Bererton, P.K.Khosla, *Towards a team of robots with repair capabilities: a visual docking system*, 2001, In Experimental Robotics VII, p.333-342, Springer, Berlin, Heidelberg,

[196] M.K.Ackerman, G.S.Chirikjian, *Hex-DMR: a modular robotic test-bed for demonstrating team repair*, May 2012, In 2012 IEEE International Conference on Robotics and Automation, p. 4148-4153, IEEE

[197] R.O'Grady, C.Pinciroli, R.Groß, A.L.Christensen, F.Mondada, M.Bonani, M.Dorigo, *Swarm-bots to the rescue*, September 2009, In European Conference on Artificial Life, p.165-172, Springer, Berlin, Heidelberg

[198] C.Pinciroli, V.Trianni, R.O'Grady, et. al. , *ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems*, 2012, Swarm intelligence, **6(4)**, p.271-295

[199] M.A.Post, J.Austin, *Knowledge-based Self-reconfiguration and Self-aware Demonstration for Modular Satellite Assembly*, July 2019, 10th International Workshop on Satellite Constellations and Formation Flying

[200] L.Brodbeck, F.Iida, *An extendible reconfigurable robot based on hot melt adhesives*, 2015, Auton. Robot. **39**, p.87–100, doi.org/10.1007/s10514-015-9428-1

[201] N.Stephenson, *Seveneves*, 2015, William Morrow Publishing, ISBN: 9780062190376

[202] K.S.Van Hornweder, *A chronological survey of modular self-reconfigurable robots*, 2011, Department of Electrical Engineering and Computer Science University of Tennessee

[203] S.Chennareddy, A.Agrawal, A.Karuppiah, *Modular self-reconfigurable robotic systems: a survey on hardware architectures*, 2017, Journal of Robotics

[204] P. Alschbach, *Autonomous docking of heterogeneous robots*, 2011, Masters thesis, University of Stuttgart

[205] I.Parada, V.Sacristán, R.I.Silviera, *A new meta-module design for efficient reconfiguration of modular robots*, 2021, Autonomous Robots, **45**, p.457-472

[206] M.Hoffmann, R.Pfeifer, *The implications of embodiment for behavior and cognition: animal and robotic case studies*, 2012, ArXiv preprint, ArXiv:1202.0440

[207] R.Pfeifer, M.Lungarella, F.Iida, *Self-Organization, Embodiment and Biologically Inspired Robotics*, 2007, Science, **318.5853**, p.1088-1093

[208] TAROS 2018 conference, Bristol, 25-27/07/2018, `http://www.brl.ac.uk/taros2018.aspx`

[209] TAROS 2019 conference, London, 03-05/07/2019,`https://www.robotics.qmul.ac.uk/events/taros2019/`

[210] YorRobots exhibition day, York, 28/02/2020, `https://www.york.ac.uk/yorrobots/news-events/yorrobots-events/2020/exhibition-2020/`

[211] Dynamic Self-repair: Webpage containing videos and further description of the Dynamic Self-repair concept, `https://www.york.ac.uk/robot-lab/dsr/`

[212] Hinge test video, author's blog, `https://omni-pi-tent.blogspot.com/2019/01/hinge-testing-video.html`

[213] Docking port actuation GIF, author's blog, `https://3.bp.blogspot.com/-5nqBLe5qwLc/W0y-pUHMOAI/AAAAAAAAAIQ/0rQDtx8jpXEm9gYGb2rsCGf4aoikEAm-gCLcBGAs/s400/omni_pi_tent_port_docking.gif`

[214] Video of multi-module hinge testing, author's blog, `https://youtu.be/E4UiFdV7DYk`

[215] Video of a single robot docking demonstration, author's blog, `http://omni-pi-tent.blogspot.com/2019/07/docking-demonstration-video.html`

[216] Video of initial (Spring 2020) docking experiments, author's blog, `https://youtu.be/hDISJSnSR0Q`

[217] Video of 4 module hardware MLR, author's blog, https://youtu.be/wMm3fpArw1U

[218] Video of DSR self-repair simulation, author's blog, https://youtu.be/qs6zvUntNeE

[219] Video of nDSR self-repair simulation, author's blog, https://youtu.be/-I7TOkuWzPw

[220] Video of SSR self-repair simulation, author's blog, https://youtu.be/3OnnIyMio5o

[221] Video of nSSR self-repair simulation, author's blog, https://youtu.be/pv2toBDaw10

[222] Video of DMS self-repair simulation, author's blog, https://youtu.be/RSAyurRamn4

[223] Video of DMS self-repair simulation with multiple failures, author's blog, https://youtu.be/r_DMbdXTfkk

[224] Video of DSR with DMS test attempts using hardware modules, author's blog, https://youtu.be/US-zs4vbh7Y

Appendix A

# Maxims for the Modular Robot Manufacturer, or Reflections on Researching Reconfigurable Robots

This section provides a series of rules-of-thumb which may be useful to those seeking to further develop modular robotic platforms, or robots of other kinds for that matter. These suggestions are presented as a list, without space for much explanation to be given, but where they are appropriately used they are likely to save much time and bother for the aspiring modular robot designer.

- Simulate as you prototype, a simulator like V-REP can let you check that the sensors and actuators you plan to use really are enough for the tasks you want to do.
- Separate the processing, slave microcontrollers can handle real-time tasks while a single board computer with an OS can be the high-level master.
- An extra slave microcontroller is worth the space, power and monetary costs when it avoids having to force fewer microcontrollers to run highly complex, unreliable and unmaintainable programs to handle more simultaneous tasks than they ought.
- Test circuit elements on breadboard before designing PCBs. This need not always be a full test of everything which will be on the PCB tested on breadboard at once, but each subsystem of the PCB should be tested on breadboard independently.
- Before designing a PCB always ensure to test on breadboards whether each part you plan to include really does behave the way that the data-sheet makes you think it will.
- Never solder thin wires to boards or free hanging pins, buy cables with pre-fitted header connectors and use small shim PCBs to connect them to any offboard components.

- When 3D printing always leave a tolerance of around 0.2mm between parts,make this larger between freely moving parts, tighten it when mating between 3D prints and purchased parts.
- When 3D printing gears always use a script to make proper involutely curved teeth.
- Decoupling capacitors are vital between power pins of all IC chips, especially for microcontrollers. In low speed designs like Omni-Pi-tent a pair of 10uF ceramic multilayers in parallel tends to be suitable.
- A Pi filter (capacitor, inductor, capacitor, arranged $\pi$) is very good for removing ripples on the power rails to low current analogue circuit elements.
- Two resistors and a MOSFET make a good 3V3 to 5V level shifter.
- Test each subsystem alone before assembly, faking inputs or outputs by hand if necessary.
- Test each subsystem above and beyond the worst conditions you expect it to experience, for example if the motor can handle a normal stall check if it can cope with being backdriven against the desired direction while stalled.
- Controller code should contain only one large while(1==1) infinite loop, all other while loops within it should have a condition to kill them if they loop too many times.
- If a software task seems difficult see if it would be easier done in hardware.
- If a hardware task seems difficult see if it would be easier done in software.
- Begin integration of subsystems as early as possible.
- Rotating 3D printed parts should always ride on metal axles, bearings or both. Wherever possible avoid having one 3D printed part use another as an axle. If no other option is available use laser cut rings to reduce the friction between rotating 3D printed elements.
- Always ensure your battery and regulator systems can provide more power than you expect you'll ever need.
- Always ensure as many spare GPIO pins as possible are wired to easily reachable headers for breaking out future capabilities and additions.
- Always ensure system buses such as I2C and SPI are broken out to easily reachable connectors, as well as power and ground.
- Decide on interfaces and protocols of communication between subsystems early, try to ensure that the overall design is such that any subsystem can be fully switched out for an upgraded one without requiring changes anywhere else so long as the original interface/protocol are adhered to.

- Never ignore your imagination when it suggests a convoluted way in which something might develop a fault, test these possibilities, see if faults do occur then debug when necessary.
- Always test sensors in proximity to the other elements of the robot they will be operating near.
- Where possible avoid libraries and programs with complex installation procedures and large dependency chains. Try to contain in one folder everything needed to get a single-board-computer from the freshly installed OS to the state in which it can run your robot.
- Although it can make squeezing traces on to mechanically constrained PCBs more difficult, avoiding SMD connectors for wires and using through-hole equivalents avoids the risks of connectors being torn off when assembling robotic parts.
- When designing algorithms for group behaviours of robots, always ensure any behaviour specified has timeout conditions, atleast one of which should return the robot to the default state in which its controller program starts should nothing happen for long enough. Avoid dead ends in the FSM.
- The trick is not to get the design right in the next iteration, it is to ensure that the next iteration can be easily adapted to make the design right.
- PCB boardhouses often charge a large proportion of their cost for each new design, but relatively little extra for increased area and no extra for increased complexity. Joining together multiple boards of your own design in to a single larger board, then having this ordered as one design and carefully separating the parts and filing clean the joins when they are delivered can often be cheaper for arrays of simpler boards, such as connector adapter shims for offboard components.
- When running ribbon and other multi-wire cable assemblies through tight spaces always ensure to wrap the whole cable in something, even if only a layer of tape. Single wires which become separate from the rest in a cable can get easily trapped and crushed between moving parts, keeping wires together as a bundle makes it much more difficult for these breakages to occur.
- SMD capacitors are usually unlabelled, try to use as few different values as possible in a circuit design so as to reduce the risk of accidentally soldering them in each other's places.
- Maintain a written list of each new global variable you add to any important pieces of source code, this is especially important for non-compiled languages where you cannot simply search for float and int to find declarations.

# Appendix B

# Modular Robotic Platforms Comparison Table

| Robot | Architecture | DoF | Docking Method | Active Ports per Module | Passive Ports per Module | Independent Mobility |
|---|---|---|---|---|---|---|
| **ATRON** [18] | Lattice | 1, rotational | Gendered Mechanical | 4 | 4 | None |
| **M-Tran** [13] | Hybrid | 2, about same axis | Gendered Magnetic | 3 | 3 | Wiggling |
| **ModRED** [50] | Chain | 4, hinging of two ports, rotation and extension of core | Genderless Mechanical | 2 | 0 | Wriggling/Stretching |
| **Trimobot** [62] | Mobile chain | 1, hinging off-centre | Gendered Mechanical | 1 | 5 | 3 Omniwheels, omnidirectional |
| **SMORES** [35] | Mobile lattice | 4, rotation of 3 ports and hinging | Genderless Magnetic | 3 | 1 | 2 Wheels, forward/back/turn |
| **SYMBRION (Scout)** [189] | Mobile chain | 2, about same axis | Bigendered Mechanical | 2 | 2 | 2 Tracks, forward/back/turn |
| **HyMod** [42] | Mobile hybrid | 3, rotations of two ports and hinging about orthogonal axis | Genderless Mechanical | 4 | 0 | 2 Wheels, forward/back/turn |
| **Omni-Pi-tent** [41] | Mobile hybrid | 2, about orthogonal axes | Genderless Mechanical | 4 | 0 | 4 Omniwheels, omnidirectional |

**Table B.1:** Table providing a comparison of design features of the platforms discussed in detail in Sec.(2.1). The architecture column indicates which architecture best describes the robot, however it should be noted that hybrid and mobile robots can often form chain and lattice structures. The degrees of freedom (DoF) column indicates the number of degrees of a freedom a module has once docked, the number shown counts the number of ways in which it may hinge or, in the case of one of ModRED's degrees of freedom, stretch. The degrees of freedom count does not take account of individual driving and wheel motion where robots with a mobile architectures are concerned. The types of docking port, along with how many active (male) and/or passive (female) ports are fitted to a module are listed in the subsequent columns. The independent mobility column indicates how a single robot can move across the ground when alone, the robots which are not of mobile architectures are quite limited in this regard. The Omni-Pi-tent design developed in this thesis is also compared.