

Fault Tolerant Morphogenesis in Self-reconfigurable Modular Robotic Systems

Lachlan James Murray

Submitted for the degree of Doctor of Philosophy

THE UNIVERSITY OF YORK
DEPARTMENT OF ELECTRONICS

September 2013

Abstract

The number of people affected by natural disasters and the financial costs of recovery are increasing. To improve the survival rate, reduce the financial costs and reduce the risks posed to rescue workers, several authors have proposed using robotic modules to aid in the search and rescue and cleanup operations that follow natural disasters. Due to the potentially high-levels of adaptivity that they may provide, there is one class of system in particular which is often highlighted: self-reconfigurable modular robots.

Self-reconfigurable modular robots are robotic modules which can physically connect with one another and dynamically alter their structural configuration through a process known as morphogenesis. For self-reconfigurable modular robotic systems to be effective in tasks like search and rescue they will need to be capable of surviving autonomously for long periods of time, whilst demonstrating high levels of reliability and fault tolerance.

In this thesis, a novel approach is used to study the reliability of an existing morphogenesis algorithm, using techniques from the field of reliability engineering. The techniques are found to be effective in identifying problems with the algorithm and the findings are used to help develop new strategies for detecting faults and recovering from failures. Results from simulated and real robot experiments show that systems employing these strategies are able to recover significantly quicker and survive longer than systems which do not. The design of a new platform extension and algorithms for controlling the collective locomotion of robots equipped with the extension are also presented. Through the novel use of ‘virtual sensors’, the same locomotion strategy is used to demonstrate implicit forms of self-assembly and self-reconfiguration.

It is concluded that the long-term autonomy of self-reconfigurable modular robotic systems can be improved through the study and development of new and existing approaches to fault tolerant morphogenesis.

Contents

1	Introduction	1
1.1	Autonomous Robotics	4
1.2	Fault Tolerance	5
1.3	Symbiotic Evolutionary Robot Organisms	6
1.4	Thesis Outline	7
1.4.1	Contribution	10
2	Fault Tolerant Autonomous Robotics	11
2.1	Reliability Analysis	11
2.2	Fault Detection	14
2.3	Failure Recovery	16
2.4	Summary	18
3	Self-reconfigurable Modular Robotics	20
3.1	Symbicator	21
3.1.1	Platform	21
3.1.2	Simulators	26
3.1.3	100 Robots 100 Days	28
3.2	Chain	29
3.3	Lattice	32
3.4	Hybrid	36
3.5	Mobile	38
3.6	Summary	40
4	Reliability Analysis and Morphogenesis	43
4.1	Morphogenesis Controller	43
4.1.1	Structural Representation	44
4.1.2	Controller	45
4.2	Failure Mode and Effect Analysis	49
4.2.1	Failure Modes	50
4.2.2	Effects	50
4.2.3	Analysis	55
4.3	Fault Tree Analysis	56
4.3.1	Construction	57

4.3.2	Analysis	60
4.4	Discussion	62
4.5	Summary and Future Work	64
5	Energy Foraging and Anomaly Detection	66
5.1	The ‘modified’ Dendritic Cell Algorithm	67
5.1.1	Overview	67
5.1.2	Improvements	69
5.2	Energy Foraging Controller	70
5.2.1	Robots	70
5.2.2	Sensor Data	72
5.2.3	Controller	73
5.2.4	Comparisons	76
5.3	Multi-objective Optimisation	78
5.3.1	Support Vector Machines	78
5.3.2	Experimental Setup	79
5.3.3	Results	83
5.3.4	Analysis	89
5.4	Long-term Survival	91
5.4.1	Experimental Setup	92
5.4.2	Results	95
5.4.3	Analysis	100
5.5	Summary and Future Work	102
6	Self-repairing Robotic Structures	104
6.1	Strategy	105
6.1.1	Assumptions	106
6.1.2	Controller	108
6.1.3	Comparisons	125
6.2	Symbicator Implementation	126
6.2.1	DetermineShape Adaptation	126
6.2.2	CalculateScore Implementation	129
6.2.3	Broadcasting Period	133
6.2.4	Communication Redundancy	133
6.2.5	Limitations	134
6.3	Stage Experiments	136
6.3.1	Experimental Setup	136
6.3.2	Results	138
6.3.3	Analysis	140
6.4	Robot3D Experiments	141
6.4.1	Experimental Setup	141
6.4.2	Results	146
6.4.3	Analysis	151

6.5	Real Robot Experiments	153
6.5.1	Experimental Setup	153
6.5.2	Results	157
6.5.3	Analysis	160
6.6	Summary and Future Work	161
7	Self-assembling and Self-reconfiguring Robotic Structures	164
7.1	Comparisons	165
7.1.1	Platforms	165
7.1.2	Control Strategies	166
7.1.3	Summary	167
7.2	Modular e-puck Extension	168
7.3	Algorithm Description	170
7.3.1	Static synchronisation	172
7.3.2	Sensor-aware alignment	173
7.3.3	Obstacle Avoidance	175
7.3.4	Forward Bias	176
7.4	Experimental Setup	177
7.5	Collective Locomotion	178
7.5.1	Stationary Alignment Results	179
7.5.2	Collective Locomotion Results	179
7.5.3	Scalability	185
7.6	Self-assembly	186
7.6.1	Experimental Setup	187
7.6.2	Analysis	188
7.7	Self-reconfiguration	191
7.7.1	Self-disassembly	191
7.7.2	Self-reconfiguration	193
7.8	Summary and Future Work	195
8	Conclusions	198
8.1	Summary and Contribution	198
8.2	Concluding Remarks	202
8.3	Future Work	202
	Bibliography	205

List of Tables

3.1	The various infrared components on-board the Symbricator robots . . .	24
3.2	The main properties of the ‘chain’ platforms reviewed in this chapter .	30
3.3	The main properties of the ‘lattice’ platforms reviewed in this chapter .	33
3.4	The main properties of the ‘hybrid’ platforms reviewed in this chapter .	36
3.5	The main properties of the ‘mobile’ platforms reviewed in this chapter .	38
4.1	Symbols for representing different robots and docking sides	45
4.2	Hazards investigated by [193] and those analysed in this study	50
4.3	A summary of the FMEA study	55
5.1	The five main parameters of the original mDCA	69
5.2	The names, descriptions and formulae of the mDCA features	71
5.3	The 11 parameters of the new version of the mDCA	72
5.4	The three different types of fault simulated in this chapter	73
5.5	The four different thresholds of the energy sharing strategy	74
5.6	The NSGA-II parameters used during the optimisation experiments . .	80
5.7	The parameters optimised in mDCA and SVM experiments	80
5.8	The ranges over which the evolvable parameters are restricted	81
5.9	Measures for assessing the performance of anomaly detection	82
5.10	The features from the final generation of the mDCA-II experiment . . .	85
5.11	The features from the final generation of the SVM experiment	87
5.12	A comparison of AUC values from the mDCA-I/II and SVM systems .	90
5.13	The mean run-time of the mDCA and SVM experiments	91
5.14	Hypotheses for comparing the long-term survival of different systems .	94
5.15	The p-values obtained by comparing the amount of stored energy . . .	96
5.16	The p-values obtained by comparing the number of surviving robots . .	97
6.1	The different types of message sent between robots during self-repair .	109
6.2	A summary of the results from the Robot3D experiments	146
6.3	The number of successful and unsuccessful runs	161
7.1	The list of parts required to construct a single e-puck extension	170
7.2	The two parameter sets used in the experiments	178
7.3	A summary of the results from the collective locomotion experiments .	181
7.4	A summary of results from the self-assembly experiments	188

List of Figures

1.1	An example scenario for the Symbricator robots	7
2.1	The taxonomy of robot failures introduced by [20]	12
3.1	Images of the three Symbricator robots and a passive module	22
3.2	The docking elements and infrared components of a Scout robot	23
3.3	The ‘T’ shaped Active Wheel module, docked with a Scout robot	26
3.4	Screenshots from the Robot3D simulator	27
3.5	Screenshots from the Stage simulator	28
3.6	A graphical representation of the 100 Robots 100 Days challenge [88]	29
3.7	Examples of chain-based self-reconfigurable modular robotic systems	30
3.8	Examples of lattice-based self-reconfigurable modular robotic systems	33
3.9	Examples of hybrid self-reconfigurable modular robotic systems	36
3.10	Examples of mobile self-reconfigurable modular robotic systems	39
4.1	A multi-robot structure and its corresponding graph representation	44
4.2	The partial construction of a string-based structural representation	46
4.3	A finite state machine for the morphogenesis controller of [108]	46
4.4	A fault tree for examining a stall in the formation of a structure	58
4.5	A continuation of the fault tree from figure 4.4	59
4.6	A minimal version of figures 4.4 and 4.5	61
5.1	Screenshots of robot models within the Stage simulator	72
5.2	A finite state machine for the energy foraging controller	74
5.3	The progress and TPR/FPR trade-off during the mDCA-I experiment	84
5.4	The progress and TPR/FPR trade-off during the mDCA-II experiment	86
5.5	The features used during each generation of the mDCA-II experiment	86
5.6	The progress and TPR/FPR trade-off during the SVM experiment	88
5.7	The features used during each generation of the SVM experiment	88
5.8	Comparisons of the objective function scores and the TPR/FPR	90
5.9	A histogram of features from the mDCA-II and SVM experiments	91
5.10	The environment used during the long-term survival experiments	93
5.11	The total stored energy of 50 robots operating over a ten hour period	95
5.12	The amount of energy present at the end of each experiment	96
5.13	The total number of operational robots over a ten hour period	97

5.14	The number of robots present at the end of each experiment	98
5.15	Boxplots showing how the FPR affects the amount of stored energy . .	99
5.16	Boxplots showing how the FPR affects the number of surviving robots .	99
6.1	The four stages of the self-repair strategy	105
6.2	An extended version of the finite state machine from chapter 4	109
6.3	An example structural configuration alongside its graph representation	110
6.4	An example of modules entering the Supporting state	111
6.5	A finite state machine for the Repairing behaviour	112
6.6	The messages sent between modules in the DetermineShape state . . .	115
6.7	The construction of a sub-structure string	116
6.8	How one robot may calculate the <i>Heading</i> of its neighbour	118
6.9	How the <i>StructureID</i> may be determined in two different scenarios . .	119
6.10	How a consensus is reached between three separate sub-structures . . .	121
6.11	An example of modules executing the Reshaping behaviour	124
6.12	The behaviour of modules in the old and new DetermineShape states .	127
6.13	A target shape and the scores assigned to three potential sub-structures	131
6.14	How a graph representation can be altered to move the <i>Seed</i> location .	132
6.15	How a string representation can be altered to move the <i>Seed</i> location .	132
6.16	Two partially assembled structures within the Stage simulator	137
6.17	The self-repair process during the assembly of structure <i>B</i>	139
6.18	Boxplots showing the time taken to assemble structures <i>A</i> and <i>B</i> . . .	140
6.19	Screenshots of the Robot3D simulator	142
6.20	The seven hand-designed structures used in this section	143
6.21	Some of the randomly generated structures used in this section	144
6.22	One complete and one partially complete 12-module structure	147
6.23	The number of structures during the assembly of 7A, 10A and 12A . .	149
6.24	Assembly time of the baseline, self-repair and naive recovery systems .	150
6.25	Recovery time of the self-repair and naive recovery systems	150
6.26	Recovery time plotted against structure size	151
6.27	Recovery time plotted against repair potential	152
6.28	A diagram outlining scenario A	154
6.29	A diagram outlining scenario B	154
6.30	A diagram outlining scenario C	156
6.31	A diagram outlining scenario D	156
6.32	The average distance separating three robots during scenario A	157
6.33	Composite images from a video of scenario B	159
6.34	Still images from a video of scenario C	159
6.35	Still images from a video of scenario D	160
7.2	A high-level overview of the collective locomotion controller	171
7.3	The positioning of the infrared sensors on board an e-puck robot	171
7.4	An intensity heatmap of messages sent between two modules	172

7.5	The strategy for correcting the misalignment of two robots	174
7.6	Top down views of the arena used for the experiments	177
7.7	The various different configurations used in the experiments	178
7.8	The mean polarisation during the stationary alignment experiments . .	180
7.9	The centre of mass of each structure during collective locomotion . . .	182
7.10	The average speed and neighbour-count of different structures	184
7.11	The relationship between speed and coverage in linear structures	185
7.12	The pairwise distance between three robots over time	186
7.13	The number of structures over time, with and without virtual sensors .	189
7.14	The number of structures and the structure size over time	192
7.15	The number of structures during the self-reconfiguration experiment . .	194
7.16	Still images from a video of environment driven self-reconfiguration . .	194

List of Accompanying Material

5.1	mDCA anomaly detection	anomaly_detection.mp4
5.2	Long-term survival	long_term_survival.mp4
6.1	Self-repair during assembly (Stage)	stage_self-repair_1.mp4
6.2	Self-repair following assembly (Stage)	stage_self-repair_2.mp4
6.3	Self-repair following assembly (Robot3D)	robot3d_self-repair.mp4
6.4	Repair scenario A (Symbricator)	symbricator_repair_A.mp4
6.5	Repair scenario B (Symbricator)	symbricator_repair_B.avi
6.6	Repair scenario C (Symbricator)	symbricator_repair_C.avi
6.7	Repair scenario D (Symbricator)	symbricator_repair_D.mp4
7.1	Stationary alignment	stationary_alignment.mp4
7.2	Collective locomotion	collective_motion.mp4
7.3	Five module self-assembly	five_module_assembly.mp4
7.4	Environment driven self-reconfiguration	reconfiguration.mp4

Acknowledgements

I would like to thank my supervisors Jon and Andy for their guidance, my family for their support and Nick and Alice for their patience and company.

Declaration

With the exception of the original morphogenesis controller and the ‘modified’ Dendritic Cell Algorithm (mDCA), all of the research presented within this thesis is the author’s own. Some parts of this research have previously been published elsewhere:

L. Murray, W. Liu, A. Winfield, J. Timmis, and A. Tyrrell. Analysing the reliability of a self-reconfigurable modular robotic system. In *Proceedings of the 6th International Conference on Bio-Inspired Models of Network, Information, and Computing Systems, BIONETICS 2011*, December 2011

L. Murray, J. Timmis, and A. Tyrrell. Self-reconfigurable modular e-pucks. In *Proceedings of ANTS 2012, 8th international conference on Swarm intelligence*, volume 7461 of *LNCS*, pages 133–144. Springer, 2012

L. Murray, J. Timmis, and A. Tyrrell. Modular self-assembling and self-reconfiguring e-pucks. *Swarm Intelligence*, 7(2-3):83–113, 2013

CHAPTER 1

Introduction

On 11 March 2011, at 14:46 (JST), an earthquake of magnitude 9.0 M_W occurred 70 km off the coast of Japan. The resulting tsunami struck the Fukushima Daiichi Nuclear Power Facility and led to the largest nuclear accident since the Chernobyl disaster in 1986. Prior to impact, three of the plant's six reactors were out of service and three were in a cooling phase, having been automatically shut down after the initial earthquake. When the tsunami hit, the plant's main power supply was cut and the back-up generators were damaged, disrupting the cooling process. Overheating, and the build up of hydrogen gas, led to explosions which damaged the reactors and surrounding buildings, resulting in the release of radioactive material [35]. In the subsequent efforts to contain the situation, members of the response team underwent significant personal risk, exposing themselves to high levels of heat and radiation [188].

The Fukushima disaster is just one of several large-scale natural disasters that have occurred within the last century. According to statistics from [102], the number of people affected by natural disasters, and the financial cost of recovery, is increasing. In 2010-12, there were 700 natural disasters recorded worldwide, affecting more than 450 million people. In the 1990s, the estimated cost of damages from natural disasters was \$20 billion per year, in the period between 2000-10 this rose to around \$100 billion.

To improve the survival rate and reduce the financial costs of natural disasters, several authors have proposed using *self-reconfigurable modular robotic systems* [207] as part of the search and rescue (SAR) and clean-up operations. Self-reconfigurable modular robotic systems are autonomous kinematic machines, composed of several relatively simple connected modules. By altering the connections between neighbouring modules, such systems may dynamically reconfigure in order to meet the demands of their current task or environment. Following a natural disaster, it is envisaged that the deployment of a self-reconfigurable modular robotic system could help to improve the chances of survival for those directly affected, whilst at the same time reducing the risks posed to rescue workers.

In the aftermath of the Fukushima disaster, emergency response workers were required to work long hours, in highly stressful circumstances. To minimise exposure to radiation, they wore double-layer protective overalls, but with ambient temperatures in

excess of 28°C, this further increased the physical strain on the workers, particularly during heavy labour tasks such as removing rubble [188]. Whilst robots were utilised at Fukushima [84, 138], these were large teleoperated machines, well equipped to operate in urban terrain, but less able to autonomously adapt to dynamic environments.

Although there are currently no field deployable self-reconfigurable robotic systems in existence, it is possible to imagine how, in the future, such systems may be used to aid in the search and rescue (SAR) or clean-up operations that follow natural disasters. For example, after deployment, modules could first form several small, highly agile robot-structures. The mobility and small size of such structures would allow them to easily navigate between gaps in the rubble of a collapsed building and autonomously search for survivors or monitor environmental conditions. Upon locating a survivor, or some other area of interest, the robots could then notify rescue workers or recruit other modules and adapt their configuration to form a structure that is more suitable for the new scenario. For example, the robots may form a larger structure that is capable of protecting survivors by stabilising the surrounding area, or a stronger configuration which is able to aid the rescue workers by excavating rubble.

For self-reconfigurable modular robotic systems to be successful in tasks such as SAR or autonomous clean-up, they will need to be capable of operating for long periods of time, without any form of human interaction. In doing so, they will need to demonstrate both high degrees of adaptivity to changes in their environment, and high levels of reliability with regards to the presence of failures and faulty individuals. It is the study and development of such behaviours that form the main focus of this thesis.

To help motivate research into the problem of long-term autonomy, and as a step towards the development of field deployable self-reconfigurable modular robotic systems, members of the SYMBRION (Symbiotic Evolutionary Robot Organisms) [176] and REPLICATOR (Robotic Evolutionary Self-Programming and Self-Assembling Organisms) [151] projects have proposed the ‘100 Robots 100 Days’ grand challenge [88]. The challenge, which is introduced in greater detail in section 3.1.3, involves placing a large group of robots in an enclosed arena for an extended period of time, and monitoring whether or not they are able to survive without any form of human interaction. To make the challenge more difficult, the location and availability of energy resources within the environment should change dynamically. To survive, the robots must adapt their behaviour accordingly.

Over the last five years, the SYMBRION and REPLICATOR projects have developed three new forms of self-reconfigurable modular robot (introduced in detail in section 3.1.1). The 100 Robots 100 Days challenge will serve as a test-bed for the robots. The success of the challenge will rely on the development of control strategies that allow the robots to harvest, share and store energy, in a manner that most benefits the collective. Such strategies will themselves rely, fundamentally, on the ability of the robots to dynamically alter their structural configuration. For example, in order to reach an energy source which is located beyond a large obstacle, the robots may be required to reconfigure themselves into a structural arrangement that allows them to climb over the obstacle. The process through which modular robots perform such

reconfiguration is often referred to as *morphogenesis* [88, 108, 136], a term borrowed from developmental biology in which it is used to describe how shape and structure emerge in biological organisms.

Over long periods of operation, within a large group of robots, it is inevitable that some modules will suffer failures. For the collective as a whole to survive, it must be possible for the system to continue operating even if one or more modules is faulty. That is to say, the system must exhibit *fault tolerance*. The ability to self-reconfigure provides one method of ensuring fault tolerance, allowing failed modules to be removed from a robotic structure and replaced by functional ones. However, for the approach to be successful, the process of morphogenesis itself must also be capable of tolerating the presence of faults.

This thesis is focused on improving the long-term survival of self-reconfigurable modular robotic systems through the study and development of distributed approaches to fault tolerant morphogenesis. Its main contributions include the presentation of an analytical study on the reliability of an existing morphogenesis controller, the continued development and optimisation of an existing algorithm for detecting faults in robotic modules, and the development of new strategies for sharing energy and efficiently repairing and reconfiguring robotic structures.

This research is supported by the SYMBRION project, consequently, the majority of the algorithms and strategies discussed in this thesis were originally developed for the SYMBRION robots. The SYMBRION robots are highly complex machines, and are not currently available outside of the SYMBRION and REPLICATOR projects. This follows a general trend in the field of self-reconfigurable modular robotics, in which very few modular robots are available commercially, or have their designs released publicly (two exceptions being [159] and [165], which are both available to buy online). In contrast, within the closely related field of *swarm robotics* [160], several robotic platforms are available to buy, or have been released as open hardware projects [44, 49, 81, 85, 122, 146, 155]. To address this issue, the final contribution of this thesis is to introduce a low-cost open source extension which transforms what is traditionally a swarm robotic platform into a self-reconfigurable modular robotic system. The platform is used to demonstrate simple forms of collective locomotion and morphogenesis, behaviours which are critical to the long-term survival of a self-reconfigurable modular robotic system.

This chapter continues in sections 1.1 and 1.2 by introducing the field of autonomous robotics and the concept of fault tolerance, serving to position these two topics within the context of this thesis and to clarify the various terminology that is used throughout the remaining chapters. In section 1.3, a brief overview of the SYMBRION project is then provided. The chapter concludes in section 1.4 by presenting an outline and general hypothesis for the remainder of the thesis and expanding upon the contributions described above.

1.1 Autonomous Robotics

Whilst the research within this thesis belongs primarily to the field of self-reconfigurable modular robotics, there are aspects of other systems from the more general field of autonomous robotics, that still have relevance within this context. This section is used to briefly introduce each of the different types of ‘autonomous robotic system’ encountered within this thesis, however, before doing so, it is important to clarify what is meant by this term.

In this thesis, an autonomous robotic system is defined as a collection of one or more robotic modules, that are designed to operate within a dynamic environment. Under this definition, the robots must be capable of sensing their surroundings and reacting autonomously, with little or no human interaction. The necessity for a dynamic environment included within this definition rules out systems such as assembly-line industrial robots, which although capable of operating without human interaction, do so within a relatively static and well defined environment. It is further imposed that the system should be capable of some form of locomotion, but noted that the individual modules themselves need not be independently mobile. For example, the robots may be moved by some external force, or through the coordination of multiple connected units, as is the case for many of the self-reconfigurable modular robotic systems reviewed in chapter 3. The following four categories of autonomous robotic system are identified (ordered in terms of increasing quantity of robots and decreasing complexity):

Individual Robotic System A robotic system containing only a single robot. Individual robots are typically much more complex than those found in other systems and research involving them tends to focus on navigation or sensing tasks.

Multi-robot Team A robotic system containing several, possibly heterogeneous, physically independent modules. The individual units are typically smaller and less complex than those found in individual robotic systems. Different individuals may take on very different roles within the same team and may be controlled in a distributed or centralised manner.

Self-reconfigurable Modular Robotic System A highly adaptable robotic system, made up of several, relatively simple, physically connected modules, which are capable of dynamically altering their morphology in order to suit their task or environment [207]. The individual units may be homogeneous or heterogeneous, and can be controlled in a distributed or centralised fashion.

Swarm Robotic System A robotic system composed of several, relatively simple, physically independent modules, in which global behaviour emerges solely from the interactions of the individuals and their environment. The robots are normally much simpler than those used in the other types of robotic system but are also deployed in much larger groups. The individuals are typically homogeneous and are usually controlled in a distributed and self-organising manner, reliant entirely upon local sensing and communication [160].

The fields of swarm and self-reconfigurable modular robotics both concern the problem of coordinating the behaviour of a group of relatively simple embodied agents. The link between the two fields is exemplified by platforms such as the s-bot [121] and foot-bot [40], which are frequently described as belonging to both. The self-reconfigurable modular robotic platform developed in chapter 7, is itself based upon a swarm robotic system, further highlighting the close connection between the two fields.

Although modular robotic systems have less in common with multi-robot teams and individual robots than they do with swarm robotic systems, the fields are still relevant to one another. For example, research into providing fault tolerance to individual robots may, in some cases, be equally applicable to the individual units of a self-reconfigurable modular robotic system. Likewise, at a slightly higher level, the coordination of a small group of robots that is necessary in multi-robot teams, may be viewed as a sub-task of the type of coordination required by self-reconfigurable modular robotic systems.

In chapter 2, when reviewing alternative approaches to robotic fault tolerance, several of the most pertinent pieces of work come not from the field of self-reconfigurable modular robotics itself, but from the related areas described above.

1.2 Fault Tolerance

Fault tolerance is defined as the ability of a system to continue operating, despite the presence of one or more faulty components. Another property that is closely related to fault tolerance is ‘graceful degradation’. A system is said to exhibit graceful degradation when, following the introduction of a fault, it continues to operate at a reduced level of performance. Graceful degradation is clearly preferred to a complete system breakdown, and as such is a common component of many fault tolerant systems.

Fault tolerant systems may be loosely classified as either *implicit* or *explicit* [27]. With implicit fault tolerance, the control of the system is not altered when a fault occurs. Either through redundancy or as an emergent property of the system’s controller, the failure of one component does not prevent the remainder of the system from operating. For example, in a system of four independent robots, each executing the same task in parallel, the failure of one robot will not prevent the remaining three from completing the task. In this case, redundancy alone is sufficient to ensure the continued operation of the system. However, as highlighted by [194], if a system contains many interacting elements, and the failure of one component can have an effect on the others, then redundancy alone may not suffice, and alternative approaches to fault tolerance may be required.

The explicit approach to fault tolerance can be thought of as a two step process, first involving the detection and diagnosis of a fault, and subsequently followed by an appropriate response that serves to remove, isolate or mitigate the effects of the fault. For example, in a mobile robot, if a fault causes one wheel to turn at a lower rate than the others, resulting in the robot moving in a circle, an explicit fault tolerance mechanism may detect this fault and respond by reducing the speed of the robot’s other

motors, allowing the robot to move in a straight line again.

Whilst the definition of ‘fault tolerance’ given above is widely accepted in literature [27, 29, 83, 193] there is sometimes confusion in the definition of the terms: fault, failure, error and anomaly. In this thesis, a classification based upon that of Laprie [103] and Carlson and Murphy [19] is used. In this classification, a fault is the *cause*, an error (or anomaly) is the *state* and a failure is the *event*. Consider a small mobile robot with an array of infrared (IR) range sensors, performing simple obstacle avoidance. If a fault occurs in one of the IR range sensors, it may cause that sensor to return a value which is biased by a constant amount away from its true value, this is an error. If the control system then reads the value of the faulty sensor it will no longer be able to effectively perform obstacle avoidance, and hence, the fault has manifested itself as a failure in the control system.

Whilst these terms appear tightly linked, it is important to note that, although by definition, a fault will always lead to an error, an error will only result in a failure if it becomes ‘activated’. If for example, the control system of the imagined robot never reads the value of the faulty sensor, the failure will not occur. Using the terminology of [103] an error that is not yet activated is referred to as a *latent* error.

It is hard to remove a fault without replacing the faulty component, and an error will persist as long as the fault is present, however, if the fault can be isolated or the value of the error can be corrected, then it is entirely possible to reduce or completely remove a failure. In this thesis, both implicit and explicit methods of removing failures are presented. In chapter 5, an explicit method of detecting anomalies in infrared sensor data is described and in chapter 6, a compatible strategy for removing failures by isolating the faulty modules is presented. Meanwhile, in chapter 7, an implicit mechanism for repairing multi-robot structures is reported.

1.3 Symbiotic Evolutionary Robot Organisms

The aim of the SYMBRION and REPLICATOR projects is to investigate novel approaches to the design and control of self-reconfigurable modular robotic systems. There is a significant amount of crossover between the two projects, with core tasks such as self-assembly [36, 89, 108, 148, 196] and collective locomotion [68, 69, 87, 115] targeted by members of both. There are, however, certain topics which help to distinguish the two projects. Within SYMBRION there is greater emphasis placed upon so-called ‘bio-inspired’ approaches, with methods from the field of evolutionary robotics in particular forming a major part of the project [6, 13, 67, 70, 123, 190]. Elsewhere, investigations into the provision of fault tolerance, inspired by the function of the vertebrate immune system [11, 119, 120, 178], further highlight this bio-inspired theme. REPLICATOR, on the other hand, though not devoid of biological influence, may be thought of as pursuing a more ‘traditional’ approach to robotics. Focal topics include world modelling and sensor fusion [147, 184], along with various approaches to vision based navigation

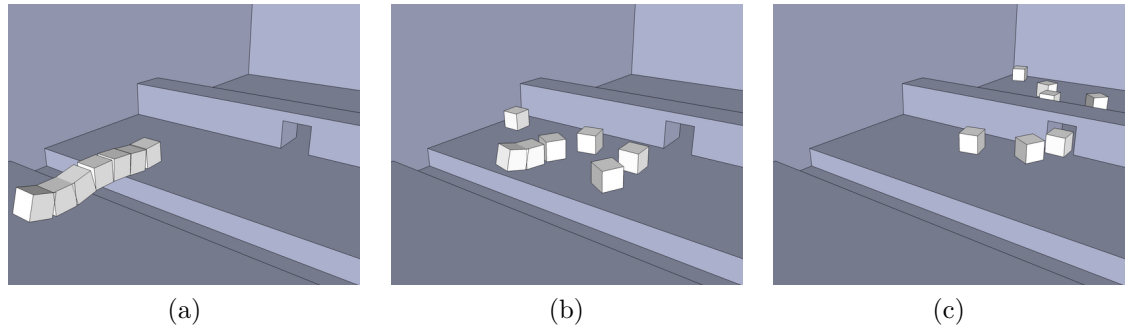


Figure 1.1: An example scenario for the Symbicator robots

[2, 3, 94, 95]. A more exhaustive shared list of publications can be found online¹ and an extensive overview of both projects, published in 2010, is provided by [105].

Though divergent in terms of their scientific focus, the two projects shared in the design of a common hardware platform, referred to hereafter as *Symbicator* (SYMBRION and repliCATOR). An example of how the completed system may be expected to operate is provided in figure 1.1. A multi-robot structure is shown climbing over a ditch (a), a task which an individual robot, or an structure arranged in a different morphology, may find impossible. The structure is then shown to disassemble into its constituent parts (b), allowing the individual modules to travel through a small gap (c), another task that would be difficult in any other configuration. The central challenge of the SYMBRION and REPLICATOR projects is how to design controllers that are able to exploit the capabilities of this highly flexible platform and demonstrate the kind of adaptability depicted in figure 1.1, whilst at the same time demonstrating extreme tolerance to environmental changes, both external (lighting conditions, resource availability) and internal (energy level, component failures).

To assess the performance of this platform, the SYMBRION and REPLICATOR projects have proposed the ‘100 Robots 100 Days’ grand challenge (introduced in section 3.1.3). Towards the end of the projects, using the various approaches developed throughout, this grand challenge will be used to demonstrate adaptation and evolution within robotic structures. Through their situation in a dynamic environment, it is envisaged that, to survive, individual robots and larger robotic structures will need to exhibit the properties of self-organisation, self-reconfiguration and self-repair; co-evolving and cooperating with each other to ensure the continued survival of the collective.

1.4 Thesis Outline

This thesis is focused on improving the long-term autonomy of self-reconfigurable modular robotic systems through the study and development of distributed approaches to fault tolerant morphogenesis. The thesis may be split into two main parts. The

¹<http://www.symbrion.eu/publications>

first part involves analysing an existing approach to morphogenesis and adapting it to provide fault tolerance. This problem is tackled in three stages:

1. **Reliability Analysis** - In the first stage, the reliability of an existing approach to morphogenesis is analysed. Some of the ways in which the system is likely to fail are identified and areas in which improvements need to be made are highlighted.
2. **Anomaly Detection** - Based upon the outcome of the reliability analysis, in the second stage, an existing anomaly detection algorithm is extended and optimised, before being applied to the task of detecting errors in infrared (IR) sensor data.
3. **Failure Recovery** - In the final stage, a recovery strategy is developed for efficiently reconfiguring robotic structures in which individual modules have failed.

The second part of this thesis involves the design of a platform extension which transforms an existing type of swarm robotic system into a self-reconfigurable modular robotic platform. New algorithms are developed for the platform which demonstrate, in a simplified setting, the types of self-assembling and self-reconfiguring properties that are necessary to ensure the long-term survival of a collective robotic system.

The following general hypothesis is used to guide this thesis. In later sections, more concrete hypotheses are presented to analyse specific parts of the problem.

Hypothesis: *The long-term autonomy of self-reconfigurable modular robotic systems can be improved through the study and development of distributed approaches to fault tolerant morphogenesis. This may be achieved through the design of new robotic platforms, through the study of existing systems, through the development of algorithms for detecting faults in robotic modules, and through the development of strategies for recovering from failures.*

The remaining chapters of this thesis are summarised below:

Chapter 2 - Fault Tolerant Autonomous Robotics This chapter reviews existing work into fault tolerant autonomous robotics. It begins by looking at some of the different methods that have been used to analyse the reliability of robotic systems, motivated by the desire to understand when and where faults are likely to occur, and what effects they may have on a system. The chapter then moves on to review some of the existing approaches that have been developed for detecting faults in robotic systems. Finally, the chapter concludes by examining some of the different strategies that have been developed for recovering a robotic system from a failed state.

Chapter 3 - Self-reconfigurable Modular Robotics This chapter introduces the robots and simulators used throughout the remainder of this thesis and provides an extensive review of the general field of self-reconfigurable modular robotics. Based upon the taxonomy of [207], the review is split into four sections, each covering one of the four main classes of self-reconfigurable modular robotic system.

Chapter 4 - Reliability Analysis and Morphogenesis This chapter begins by introducing a morphogenesis controller previously developed for the SYMBRION robots. The controller is then analysed using two different techniques from the field of reliability engineering. The techniques, Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA), are used to compare two different variants of the morphogenesis controller. The reliability of the controller is then discussed and areas where improvements could be made are identified. The chapter concludes by discussing the suitability of techniques such as FMEA and FTA within the fields of swarm and self-reconfigurable modular robotics.

Chapter 5 - Energy Foraging and Anomaly Detection This chapter presents an adapted version of an immune-inspired anomaly detection algorithm for detecting errors in infrared sensor data. An energy foraging controller is also introduced that allows robots to dock with and share energy with one another. The anomaly detection algorithm is optimised using multi-objective optimisation and its performance is compared with a state-of-the-art Support Vector Machine (SVM) based approach. In a simplified version of the 100 Robots 100 Days grand challenge, it is demonstrated how, with the integration of a basic fault recovery mechanism, the anomaly detection algorithm and energy foraging behaviours can benefit the long-term survival of a self-reconfigurable modular robotic system.

Chapter 6 - Self-repairing Robotic Structures This chapter presents an extension to the morphogenesis controller introduced in chapter 4 which allows the system to ‘self-repair’ if a fault is introduced into one of the modules. After describing the self-repair strategy, results are presented from both simulated and real robot experiments and the strategy is shown to be effective at ensuring that the system successfully recovers from failures. Some of the interesting properties of the strategy are analysed in detail and its limitations are discussed.

Chapter 7 - Self-assembling and Self-reconfiguring Robotic Structures This chapter presents the design of a new, low-cost, structural extension for the e-puck robot [122]. An algorithm for controlling the collective locomotion of a group of robots equipped with the extension is introduced and the implicit self-assembling and self-reconfiguring properties of the system are analysed. Finally, by examining a form of environment driven self-reconfiguration, the behaviour of the system is observed in a more complex environment.

Chapter 8 - Conclusions This chapter summarises the main findings and conclusions from chapters 2-7, before revisiting the general hypothesis introduced at the start of this section and highlighting some potential areas of future work.

1.4.1 Contribution

The primary contributions of this thesis are outlined below:

Chapter 4 presents an analytical study of the reliability of a previously developed morphogenesis controller. The study makes novel use of two techniques which are traditionally utilised within the manufacturing industry. This is the first known application of these techniques to the field of self-reconfigurable modular robotics.

Chapter 5 introduces several improvements to an existing anomaly detection algorithm and its supporting framework. The classification accuracy of the adapted algorithm is shown to be significantly better than the original. The chapter also details the development of a new energy foraging and energy sharing strategy. The strategy allows multiple robots to simultaneously recharge themselves at power sockets or provide energy for other modules. During long-term survival experiments, systems which include the anomaly detection and energy sharing behaviours are shown to significantly improve the robots chances of survival. Performance is shown to be comparable to a state-of-the-art alternative and significantly better than systems with no anomaly detection and no energy sharing.

Chapter 6 presents a new self-repair strategy for providing fault tolerant morphogenesis to self-reconfigurable modular robotic systems. The strategy relies on the robots isolating, removing and replacing failed modules with functional spares. In contrast to other similar approaches, the strategy is designed to allow the system to recover from the failure of individual modules, rather than large structural disturbances. A new metric is introduced for classifying robotic structures and is shown to be a good predictor of how well the self-repair strategy will perform on a given structure. An implementation of the strategy for the Symbicator platform and demonstrations of its effectiveness in simulation and using physical robots are also presented.

Chapter 7 introduces a new low-cost platform extension for investigating the interesting properties of self-reconfigurable modular robotic systems. The platform is both cheaper and more accessible than the current alternatives. All of the designs for the extension are freely available online. A novel approach to collective locomotion is presented which relies on inferring the orientation of neighbouring robots based upon the strength of the infrared signal received when exchanging messages. A similar strategy is used to demonstrate a new ‘seedless’ approach to self-assembly, in which robots initially form pairs or triples, before combining to create larger structures. New strategies for self-disassembly and self-reconfiguration are also presented.

CHAPTER 2

Fault Tolerant Autonomous Robotics

In order to detect faults in autonomous robots, or to design strategies for mitigating the effects of failures, it is important to understand when and how faults and failures may occur. This chapter begins by reviewing previous work into *analysing* the failure modes and reliability of mobile robots. In chapter 4, based upon some of the techniques described here, the reliability of a group of Symbricator robots is analysed. The current chapter then reviews some of the existing approaches to *detecting* robot failures. In chapter 5, this is followed up by examining the performance of an anomaly detection algorithm during an energy foraging task. Finally, some of the existing strategies developed for *recovering* a mobile robotic system from a failed state are discussed, including some methods of self-repair and self-assembly that are similar in concept to the strategies introduced by this thesis in chapters 6 and 7.

2.1 Reliability Analysis

To help determine the type and frequency of failures that can be expected within mobile robots, Carlson et al. analysed the usage logs of robots at the University of South Florida's Center for Robot-Assisted Search and Rescue (CRASAR) [18, 20]. A total of three years worth of data was collected from 13 robots, gathered under both 'laboratory' and 'field' conditions, in which the robots carried out a mixture of 'urban search and rescue' and 'military operations in urban terrain' style tasks. The measures of *Mean Time Between Failure* (MTBF) and *Availability* (the probability that a system will be error free at some given point in time [20]) were used to assess the robot's reliability.

In Carlson et al.'s studies, failures were classified according to the taxonomy shown in figure 2.1. Since the focus of this thesis is on autonomous robotic systems, only the five types of physical failure in figure 2.1 are relevant, that is: effector, power, control system, sensor and communication failures. Repairability, which in Carlson et al.'s taxonomy refers to repairs carried out by a human operator, is also not relevant here.

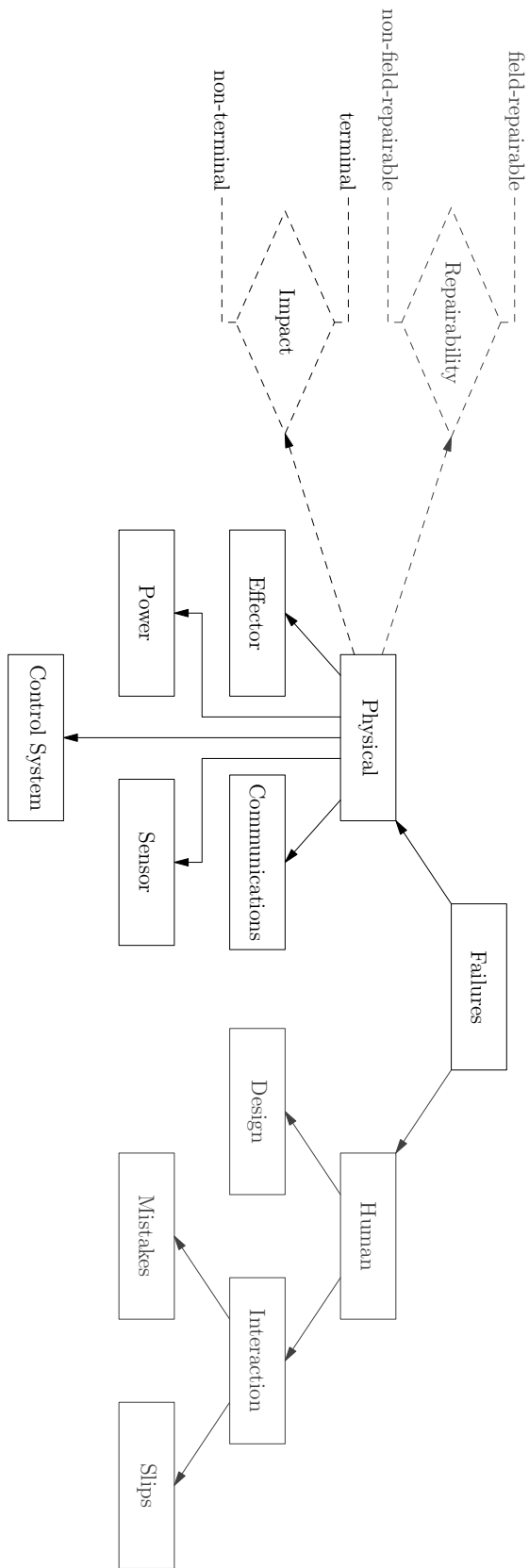


Figure 2.1: The taxonomy of robot failures introduced by [20]

The analysis of [18] and [20] showed the MTBF to lie somewhere between 8 and 24 hours, and availability to be around 54%. For both measures, robots were seen to be much less reliable under field conditions. In terms of the types of failures observed, the authors of [18] found failures of effector functions to be the most common, with examples including tracks slipping and wheels warping due to heat. In [20] however, which extended the previous study with an extra year's worth of data, failures of the control system were found to be the most common.

In a meta-study of ten different surveys into the reliability of field operating Unmanned Ground Vehicles (UGV) (including data from the original CRASAR studies) Carlson and Murphy once again found effector and control system components to be the least reliable [19]. In the same survey, it was also observed that tracked robots failed more often than wheeled robots. The least likely components to fail were found to be sensors and power systems. Carlson and Murphy suggest that these findings may be due to the fact that control systems and effectors, such as tracks and wheels, are complex and highly specific to their associated platforms, whereas power systems and sensors are relatively simple, technologically well established, and mass produced.

The analysis of the original CRASAR studies [18, 20] and the meta-study of [19] considered only individual robots, operating alone or semi-autonomously with human interaction. Of greater relevance to this thesis is the reliability of collective autonomous robotic systems. Since early advocates such as [14], it has long been claimed that the inherent redundancy of multi-robot systems can lead to greater reliability, but as highlighted by [174], there is little quantitative evidence to support this argument. To help provide some evidence for this claim, Stancliff and Dolan modelled failures in multi-robot teams, carrying out a planetary exploration mission, and assessed the trade-off between: the length of operation, the reliability of components and the cost of the system [174]. All of the experiments run using Stancliff and Dolan's models supported the claim that, at an equal or reduced cost, larger multi-robot teams of unreliable individuals may be more efficient and more reliable than smaller robot teams with more reliable individuals. Winfield et al. modelled the behaviour of a robotic swarm at a much more detailed level [195] and whilst their main intention was not to investigate fault tolerance, such a model could also have applications in this area.

It should be noted, that Stancliff and Dolan's models do not take into account what would happen if a failure occurred mid-task, and furthermore, failures are modelled as all-or-nothing, if a single subsystem fails, the whole robot fails. These simplifications are not realistic and do not allow the model to take into account the effect that a partial failure or a partially completed task may have on the other robots within the system.

Using a very different approach to Stancliff and Dolan, Winfield and Nembrini [193] take into account the effects of partial failures. Building upon their notion of a 'dependable swarm' [194] and utilising the Failure Mode Effect Analysis (FMEA) methodology, Winfield and Nembrini analysed the effects that various failures have on the behaviour of a robotic swarm whilst the robots carried out a simple containment task. Six types of failure were considered: motor, communications, avoidance sensor, beacon sensor, control system and complete system failures. Following analysis, Winfield and Nembrini

concluded that robotic swarms exhibit the property of robustness, allowing them to be thought of as reliable systems, however, whilst they are very tolerant to complete failures, they are far less tolerant to partial failures. A motor failure in the containment task, for example, may lead to an anchoring of the swarm, caused by the fact that the failed robot will still be able to communicate with its neighbours, and therefore affect their behaviour, but will not be able to move with them [193].

In other theoretical work, Bererton and Khosla modelled the reliability of multi-robot teams in which the robots are capable of repairing one another [10]. Using the ‘Aggregate’ Mean Time To Failure (AMTTF) as a metric, Bererton and Khosla compared repairable and non-repairable teams and showed the AMTTF to be greater with the repairable model. Kannan and Parker further discussed the need for metrics that measure the fault tolerance of robot teams, suggesting that system performance should be evaluated based based upon ‘efficiency’, ‘robustness’ and ‘learning’ [83].

Although multi-robot systems may have the potential to be more reliable than single-robot alternatives, it is important to realise that, in designing a collective robotic system, it cannot simply be a case of adding more robots with the hope of increasing reliability. The effects that potential failures—and in particular partial failures—may have on a system must be taken into account. To help determine these effects, several authors advocate the use of modelling and analysis [10, 83, 174, 193] but this only tells the designer what *might* go wrong, to ensure that it does not, behaviours must also be incorporated to detect and then recover from failures.

2.2 Fault Detection

In the development of fault-tolerant control systems, such as those used in aviation and other safety critical systems, detection and diagnosis are often referred to as part of the same Fault Detection and Diagnosis (FDD) process [213]. There are a number of different approaches to fault detection and diagnosis. In [213], Zhang and Jiang provide a classification that categorises methods as either model-based or data-based. The term model-free is also sometimes used to refer to non model-based approaches [75].

Model-based approaches require the construction of a model that describes the normal behaviour of the system. At run-time, the difference between the actual system output and that of the model can then be used to detect the presence of faults. In contrast, with data-based approaches, a training period is typically required, in which the output of the system (data) is used to define what normal behaviour is. As with the model-based approach, anomaly detection is performed by comparing the actual system output with the expected normal behaviour.

As highlighted by Zhang and Jiang, in the field of fault tolerant safety-critical control systems, the majority of approaches to FDD are model-based. Of greater relevance here, however, are the methods applied to mobile and collective robotic systems. In [214], Zhuo-hua et al. surveyed various methods of fault detection and diagnosis in wheeled robots and found that model-based approaches were once again well represented.

Various authors have utilised neural networks and supervised learning techniques to perform fault detection and diagnosis. In the novel *exogenous* fault detection approach of Christensen et al., a ‘leader’ robot was able to detect faults in a ‘follower’ using a time-delay artificial neural network [27]. The reverse, however, was not investigated and it was not determined whether, using the same strategy, the follower would be able to detect faults within the leader. In [28], the same approach was applied to two further tasks: docking with another robot and following a perimeter. A similar neural network based approach is described by Wang et al., in which a recurrent neural network is trained to model the sensors of an underwater robot, and detection is performed at run-time by comparing the output of the network with the sensor outputs [189].

Hashimoto et al. describe an approach to fault detection based upon voting between redundant sensors [73]. Specifically, three laser range sensors and one dead reckoning module are used to estimate the current velocity of a robot and vote accordingly. Any sensor that is not in agreement with the others is considered to be faulty. Hashimoto et al.’s approach has also been extended to a multi-robot system [74], in which both the velocity and position of robots within the group are estimated. A potential drawback of this approach is the extra cost associated with fact that the robots must possess multiple sensors capable of measuring the same information.

In [29], Christensen et al. describe a fault detection algorithm inspired by the synchronous flashing behaviour of fireflies. Christensen et al.’s approach is based upon the assumption that a failed robot will not be able (or will choose not) to flash its LEDs. Any robot that does not flash in synchrony with its neighbours is considered faulty. The approach was shown to be successful at detecting faults but is limited slightly by the requirement that robots should constantly flash their LEDs, preventing them from being used for other purposes.

In other ‘bio-inspired’ work, several authors have proposed using Artificial Immune Systems (AIS) [38] for detecting and diagnosing faults in mobile robots. Inspired by the immune theory of self/nonself discrimination, Canham et al. describe an algorithm that learns ‘normal’ behaviour during a fault free training period and detects abnormal behaviour as any deviation thereof [17]. One disadvantage of this approach is that it requires that what is considered ‘normal’ remains the same over the entire operating period of the system. Addressing this issue, Owens et al. present an immune-inspired algorithm that is able to adapt over time to gradual changes [142]. Using the algorithm from [142], Lau et al. developed a ‘collective self-detection’ scheme [104] in which robots exchange information with their neighbours in order to detect faults. Meanwhile, in [120], Mokhtar et al. describe an algorithm for detecting anomalies in sensor data which is loosely-inspired by the way in which the cells of the innate immune system detect pathogenic activity. One limitation of this approach is its use of artificial sensor data. To address this and other issues, an adaptation of the algorithm from [120] is presented in chapter 5 of this thesis.

Another disadvantage of many of the approaches described above is that they require a-priori knowledge of normal behaviour. To address this issue, [177] presents an immune inspired model for detecting abnormal behaviour in multi-agent systems, based upon the

assumption that behaviours which are ‘persistent and abundant’ should be tolerated and those which are ‘rare’ should be identified as abnormal. The approach showed promise in its ability to detect abnormal behaviour, however, it was only demonstrated within an abstract agent-based simulation which included assumptions such as global communication. It remains to be seen whether the technique would perform as well within a more realistic robotic simulator, or using physical robotic hardware.

2.3 Failure Recovery

In a fully autonomous system, it is often impossible to repair or replace low-level components such as sensors or actuators without some form of human intervention. However, if enough information is known about the fault, it may be possible to reduce the effects of a failure by correcting for, or minimising the error. If for example, a faulty sensor exhibits an anomaly that causes it to return a value that is a (known) constant amount away from what it should be, the error can be corrected simply by removing the bias.

Alternatively, at a higher level, in systems composed of many redundant components, the simplest form of recovery may be to isolate or remove the failed component. Such a strategy is particularly well suited to collective robotic systems in which, as a group, the robots possess multiple copies of the same components and, for many scenarios, individual robots may be considered expendable. The majority of the strategies discussed in this section fall into this category.

With the aim of resolving failures at a sub-system level, the authors of [8] and [1] present two different robot designs that allow sub-systems which contain faulty components to be removed and replaced autonomously by other robots.

In [8], Bererton and Khosla describe a ‘repairable’ robot in which the processing, communications and power components are grouped into replaceable sub-systems. Each robot has a forklift mechanism and each sub-system has a forklift reciprocal that allows other robots to dock with them. In [8], the authors demonstrate the robot’s ability to remove and replace faulty sub-systems by remote control, whilst in [9] they extend this work with the development of an autonomous, vision-based docking approach.

In [1], Ackerman and Chirikjian describe a hexagonal-shaped robot named Hex-DMR (Hexagonal Distributed Modular Robot). The robot consists of a star-shaped chassis and six replaceable sub-systems which fit between the arms of the structure. The replaceable parts include the power management system (and battery); the control and communications elements; and three drive units, each containing a single omnidirectional wheel. A further ‘manipulator’ sub-system, to which an optional end-effector may be added (akin to the forklift from [8]), is used to remove and replace the sub-systems of other robots. Ackerman and Chirikjian tested the repairing capabilities of their robot both in simulation and with remote controlled physical hardware.

The necessity for a modular design in which sub-systems can easily and quickly be exchanged, and the requirement for a manipulator that can reliably swap devices, means that the development of repairable robots such as [8] and [1] can become very

complicated. Especially as the number and complexity of the individual robots is increased. Although, as suggested by [8], not every robot in a system may necessarily need dedicated repair capabilities, and a system could survive with only a sub-population equipped to repair the remainder, a far simpler approach is to treat failures at the robot level and discard any module which contains a faulty sub-system. As discussed below, this approach has been successfully employed by several previous authors.

In one of the earliest examples of such a system, Tomita et al. describe a distributed self-repair process [179] for the self-reconfigurable modular robotic platform, Fracta [124]. Fracta robots are able to physically connect with one another using a combination of permanent and electro-magnets. In [179], the authors describe an algorithm for assembling structures of a particular shape and an extension that allows structures to be repaired if they are damaged. If part of a structure is removed, the robots are able to self-repair by degenerating to an earlier state of assembly and rebuilding the original structure with redundant modules.

Using a different type of modular robot, Christensen describes a fault tolerant self-reconfiguration strategy that uses simulated attraction points to stimulate growth within 3D structures [30]. Christensen evolves artificial neural networks for the ATRON robots [80] to control the assembly of different types of structure. The structures are used to bridge gaps between two planes and support platforms. When attraction points are embedded within a structure itself, and the structure is damaged by removing modules, the robots demonstrate robustness in their ability to reform a shape resembling the original structure [30]. In later work, Christensen also demonstrates the ability of the system to tolerate the failure of robots that remain within the structure [31]. One potential limitation of the approaches of Tomita et al. [179] and Christensen [30] is that the robotic structures must carry redundant modules with them at all times.

In a swarm robotic setting, Cheng et al. describe an implicit approach to self-repair for a ‘formation control’ task [24]. In simulation, robot agents are supplied with the coordinates of a shape and the task of the robots is to occupy the space within the shape’s boundary. The majority of agents are not told their position within the world and must determine it through the use of proximity sensors and local communication. Agents behave differently depending upon whether they believe they are located within or outside of the shape. Robots within the shape act like gas particles in a closed container, whilst those outside simply wander randomly. Any disruptions created by failed robots or environmental changes are automatically repaired as the agents spread out to fill the gaps.

In a similar approach to that of Cheng et al., Rubenstein and Shen describe a fault-tolerant assembly strategy in which, rather than act like gas particles, the robots follow gradient fields which define target shapes within a shared coordinate system [153, 156]. Unlike [24], in Rubenstein and Shen’s approach, during repair, structures are automatically scaled according to the number of available agents.

One disadvantage of the approaches mentioned so far is that there is no discrimination between partially and fully failed robots. Removing a robot that only has a small failure may be seen as a rather drastic approach. If a partially failed robot is able to

complete tasks or fulfil roles for which its faulty components are not required, then it may still be useful to the collective.

To address this problem, the ALLIANCE framework of Parker provides a fault tolerant distributed mechanism of action selection, in which the behaviour of an individual robot is influenced by the current requirements of its task, the behaviour of its neighbours, the state of the environment, and the state of the robot itself [145]. If a fault prevents a robot from completing its current task, the framework ensures that it will automatically switch to another task and be replaced by a more capable robot.

The majority of the approaches discussed above have only been demonstrated in simulation. Two notable counter examples are the real robot experiments performed by O’Grady et al. and Yim et al., using s-bots [137] and CKBots [208] respectively.

If a robot has failed in such a way that it cannot move by itself, it may cause problems by interfering with or acting as an obstacle to other robots. To address this problem, O’Grady et al. describe a distributed algorithm for transporting failed modules to a specialised ‘repair’ zone [137]. Using the s-bot platform of [121] O’Grady et al.’s approach involves dedicated repair robots docking with and attempting to pull failed robots to the repair zone. If the failed robot is part of a larger connected group, then a single robot will not be able to rescue it, in this scenario the rescue robot recruits others by presenting itself as a failed robot. Using real s-bots, O’Grady et al. demonstrated scenarios in which two broken robots were rescued in parallel, a pair of connected robots were retrieved by two rescue robots, and deadlock was resolved when the rescue robots were spread too thinly to effectively recover the failed robots [137].

The recovery mechanisms discussed thus far have all been triggered by relatively benign failures which affect only a few robots within the local vicinity. In order to handle a more drastic disruption to a robotic system, Yim et al. present a strategy for re-assembling a robotic structure that has been broken into several pieces by an ‘explosive’ event [208]. Using the CKBot modular robot, Yim et al. describe a weakly bound structure composed of three strongly bound sub-structures, each containing five CKBot modules. The explosive event—in this case, someone kicking the structure—causes the weak bonds to break and results in the creation of three separate structures of five modules each. The sub-structures then use cameras and LEDs to locate one another and, after moving closer, are able to dock and re-form the original structure.

2.4 Summary

To help increase the understanding of when and how robotic systems may fail, Carlson et al. analysed the usage logs of several different types of mobile robot, in a variety of different scenarios [18, 19, 20]. Carlson et al. found the MTBF to lie somewhere between 8 and 24 hours and that effector and control system components were the most likely to fail. Using different techniques, Bererton and Khosla [10], Stancliff and Dolan [174], and Winfield and Nembrini [193] all considered the effects of failures in collective robotic systems. All three groups concluded that redundancy could provide

benefits with regards to fault tolerance, but Winfield and Nembrini also warned of the dangers that partial failures could have in swarm robotic systems.

Approaches to fault detection and diagnosis can generally be classified as either model-based or data-based. In fault-tolerant safety-critical control systems, the majority of approaches to FDD are model-based. In the mobile robotics field, approaches based upon artificial neural networks [27, 28, 189], voting [73], synchronised communication [29] and artificial immune systems [17, 104, 120, 142, 177] have all been proposed.

In collective robotics systems, approaches to failure recovery have included the design of robots that can physically exchange faulty components [1, 8] and control strategies that allow robots to change roles when failures dictate that they can no longer execute their current task [145]. Using real robotic hardware, [208] demonstrated the ability of a robotic structure to recover from an explosive event and [137] showed how rescue robots could be used to drag failed modules to a repair zone. However, the most popular recovery strategy used by collective robotic systems is simply to rely on the redundancy of the system and replace the failed robots with spares [24, 30, 153, 179].

CHAPTER 3

Self-reconfigurable Modular Robotics

Self-reconfigurable modular robotic systems, are highly adaptable kinematic machines, composed of several, relatively simple, physically connected modules. The main feature of such systems is their ability to dynamically alter their morphology in order to suit their task or environment [207]. In this chapter, several of the most relevant platforms, from what is a very expansive field, are reviewed. Focus is placed primarily on the robotic hardware, rather than the algorithms that have been developed for controlling them. The purpose of this review is to help position the research from this thesis within the wider field of self-reconfigurable modular robotics, and to lay the foundations for the design of the new platform extension introduced in chapter 7.

In 2007, Yim et al. produced a comprehensive review of the field of self-reconfigurable modular robotics [207]. The review included a ‘taxonomy of architectures’ which classified platforms as either: *chain*, *lattice*, *hybrid* or *mobile*, according to the manner in which they reconfigure themselves. In chain-based architectures modules are connected to one another in series but may branch to form tree like structures or fold and reconnect to form loops within continuous three-dimensional (3D) space. Contrastingly, with lattice architectures, modules occupy discrete positions within a conceptual grid. Reconfiguration in lattice-based systems typically only involves the movement of modules between neighbouring grid positions. Architectures that combine elements from both chain and lattice based systems are described as hybrids, and architectures in which the modules reconfigure themselves by moving through their environment as individuals, are described as mobile.

When assembled, robots from a mobile architecture may be viewed as conforming to a chain or lattice type structure. However, it should be noted that, according to the taxonomy of Yim et al., it is primarily the method of reconfiguration, rather than the eventual arrangement, which determines the classification of a system. In this context, the property of mobility supersedes that of the modules geometric arrangement.

Platforms may further be classified according to the number of degrees of freedom that the individual units possess, the number of dimensions in which structures can

be formed, the type of docking mechanism utilised, and whether reconfiguration is performed in a *deterministic* or *stochastic* manner.

This chapter reviews several different types of self-reconfigurable modular robots. The review is divided into four separate sub-sections, each accounting for one of the four different types of architecture from the taxonomy of Yim et al. [207]: chain, lattice, hybrid and mobile. Photographs of the most relevant and influential systems from each class are included in each section. The main properties of all of the reviewed platforms are summarised in tables 3.2-3.5. The tables include details of the connection method used by the platform, the principal author, the year of the first publication referenced in this review and the number of controllable degrees of freedom (DOF) that the individual units possess. When considering mobile platforms, the DOF responsible for individual module locomotion are omitted, furthermore, in the case of heterogeneous systems, where different module types may have different DOF, the largest number of DOF present on a single type of unit is referenced. The tables were generated from a combination of the information gathered whilst researching this survey, and from similar tables found in [140, 207, 209].

Within the wider field of modular robotics, there exist several platforms that are not capable of *self*-reconfiguration. In such systems, robots must be connected by hand and research is focused on the control of fixed structures, rather than reconfiguration. Examples include the *Tetrobot* [71, 72] of Hamlin and Sanderson; the *YaMoR* platform [117, 118] of Moeckel et al. and the *Y1* [58] and *GZ-I* [211] family of robots from González-Gómez et al.. As the focus of this thesis is on self-repair and fault tolerance through self-reconfiguration, systems in which modules do not possess the functionality necessary to autonomously reconfigure their structure are omitted from the review.

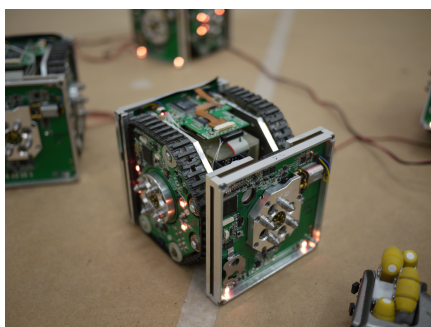
The review itself begins in section 3.2, before which, a detailed overview of the Symbricator platform, which is used extensively in chapters 4-6 of this thesis, is presented.

3.1 Symbricator

The Symbricator platform was developed as part of the SYMBRION and REPLICATOR projects. In this section, the three different types of Symbricator robot, the two simulators, and the ‘100 Robots 100 Days’ grand challenge are all introduced in detail.

3.1.1 Platform

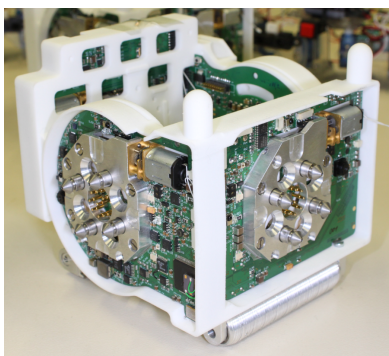
The Symbricator robots are pictured in figure 3.1. The robots are named according to their primary purpose, in figures 3.1a-c a single ‘Scout’, ‘Active Wheel’ and ‘Backbone’ robot are shown. In figure 3.1d an example of a ‘passive module’ is also pictured. To remain compatible with each another, and to aid in the development and manufacturing processes, the robots and the passive module share several common elements [90]. In this section, after introducing the common elements, the basic functionalities of all four modules are briefly described. For a more detailed overview of the platform see [88].



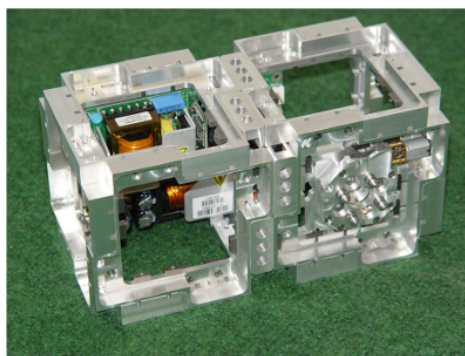
(a)



(b)



(c)



(d)

Figure 3.1: Images of a Scout module (a), an Active Wheel (b), a Backbone robot (c) and a passive module (d)

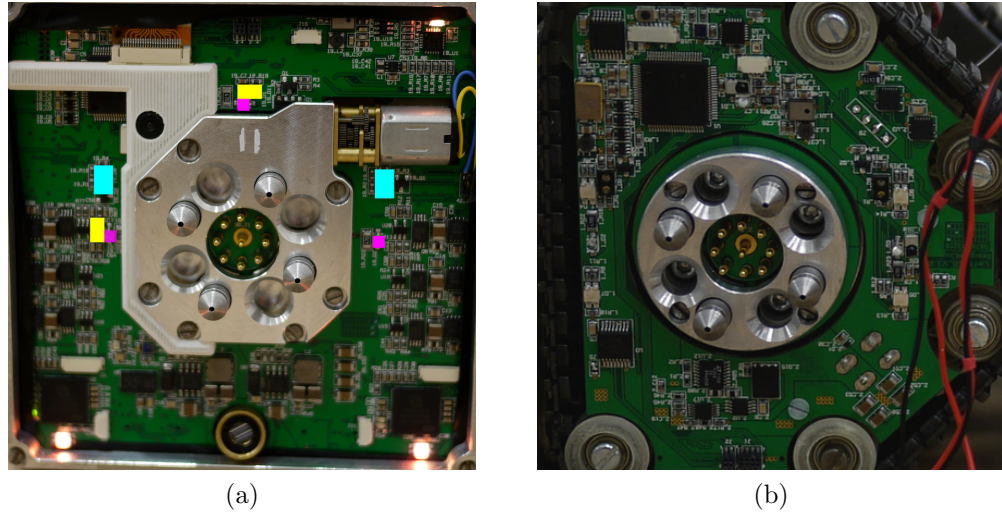


Figure 3.2: An active docking element (a) and a passive rotating docking element (b) on the sides of a Scout robot. In (a) the positioning of the infrared LEDs, sensors and receivers are marked in magenta, cyan and yellow respectively

Common Elements The set of elements shared by the Symbicator modules include communication systems such as the infrared (IR) sensors [107] and the Ethernet bus; the docking interface and docking elements [106]; the power management system [77]; and the electronics architecture and software framework [112]. Of particular relevance to this work, and the main focus of this section, are the docking interfaces and the set of IR components which surround them.

The purpose of the docking interface is twofold. Firstly, it allows modules to connect with one another to form larger structures, and secondly, once connected, it allows them to symbiotically share energy and computational resources. The docking element itself, which is responsible for securing the connection between two modules, is known as the Cone Bolt Locking Device (CoBoLD) [106] and was designed specifically for the Symbicator robots. As shown in figure 3.2, each element consists of four cone-shaped bolts and four complementary holes. There are two versions of the device, an ‘active’ version (a) in which the bolts of one device can be secured in place by the ‘locking wheel’ (not visible in figure 3.2) of another, and a ‘passive’ version (b) in which the locking wheel and the motor which drives it are not present. To guarantee that a connection between two modules is strong enough to facilitate 3D actuation, and reliable enough for a data or energy bus connection to be established, at least one of the modules must possess an active docking element. For added flexibility, the passive version of the device may be mounted on a rotating disk, therefore allowing modules with different orientations to dock with each other.

There are three main infrared components: IR LEDs which are capable of emitting IR signals, an IR sensor package which consists of an emitter and a receiver, and a dedicated remote receiver package. Two different types of receiver are required in order

Functions	Sensor		IR LED (emitter)	Remote receiver
	emitter	receiver		
Proximity	✓	✓		
Docking		✓	✓	
Communications			✓	✓

Table 3.1: The various infrared components on-board the Symbricator robots, along with their primary functions. Reproduced from [107]

to detect different frequencies of IR signal. Figure 3.2a shows the placement of infrared components around the active docking element of a Scout robot. Although the number and positioning of the IR components varies slightly between the different types of robot, their purpose remains the same. Together, the various components provide: short range proximity detection, mid-range beacon detection and longer range data communication. Figure 3.2a shows three IR LEDs (magenta), which allow the robot to act as a beacon or broadcast messages to its neighbours; two IR sensors (cyan), for obstacle and robot proximity detection; and two IR remote receivers (yellow), for detecting messages sent by other robots. The various IR components, and the functions they provide, are summarised in table 3.1.

Backbone The Backbone robot (shown in figure 3.1c) is a cubic module, with edges of length 12 cm. Its name derives from the fact that—more so than its counterparts—the Backbone is specialised to operate as a core element within a multi-robot structure.

The robot’s body consists of two main sections, joined at the centre by a hinge that provides the robot with a single rotational degree of freedom (DOF). On each of its four vertical faces the robot possesses a docking interface with an active docking element. To further suit its function, the Backbone robot has a very strong and stable structure, with actuators that are powerful enough to lift several other docked units. Given these properties, it is easy to imagine how a Backbone robot may operate in the spine of snake-like configuration, or in the joint of a larger multi-legged structure.

As an individual, the Backbone robot has a novel form of locomotion. As can be seen in figure 3.1c, the robot possesses two ‘screw drives’ on its underside, which allow the robot to move omnidirectionally, depending upon the direction and rotational speed applied to each screw. The choice of screw drives as a method of planar locomotion was also guided by the Backbone’s main function. In allowing the robot to move omnidirectionally, the task of aligning and docking with other modules, on any of the robot’s four docking sides, is theoretically made much easier.

Scout The Scout robot (shown in figure 3.1a) specialises in fast 2D locomotion and high-fidelity sensing. As its name suggests, the robot was designed specifically with scouting and surveillance tasks in mind.

Though similar in size and shape to the Backbone robot, the Scout is both lighter

and more agile. Two caterpillar tracks, controlled by a differential drive system, allow the individual robots to operate well on a variety of different types of terrain. Although less versatile than the Backbone as a member of a multi-robot structure, due to the Scout's superior sensing and locomotive abilities, it may still be used as a specialised 'head' or 'foot' module in larger robotic collectives.

Like the Backbone, the Scout robot has four docking interfaces, each containing a single docking element. The two elements on the front and back of the robot are active, whilst the two side elements are passive. The Scout cannot rotate its entire body like the Backbone, but it does possess a moving arm. Although much weaker than the hinge mechanism of the Backbone, the Scout's arm is still capable of supporting the weight of a single module. The ability of the Scout to lift or pull another robot means that, further to its role as a scouting module, it may also be used to help transport the lesser-mobile Backbone robots.

Active Wheel The Active Wheel (shown in figure 3.1b) is an example of what is referred to as a 'tool' module [88]. A tool module is a robot designed primarily to carry out a specific task, in this case, that task (from which the name 'Active Wheel' originates) is the transportation of other modules. A combination of a high ground clearance, powerful actuators and omnidirectional locomotion, enable a pair of Active Wheels, docked either side of a Backbone or Scout robot, to lift and transport the module over large distances. This allows the transported modules to travel further, and across rougher terrain than they otherwise would be able to, whilst at the same time expending less energy.

The Active Wheel can also support the Backbone and Scout modules in other ways. For example, due to its larger size, the Active Wheel can hold more energy, and therefore may serve as a store at which the other modules can recharge. Furthermore, due to its strong actuators and rotating docking elements, the Active Wheel may assist modules that have fallen into a position from which they cannot right themselves, or transport them to areas in the arena which as an individual they would not be able to reach.

The structure of the Active Wheel is very different to that of the Scout and Backbone robots. In figure 3.1b, an early prototype with a symmetrical 'S' shaped design is shown. The prototype consists of two identical segments, each equipped with two omnidirectional wheels that are positioned perpendicular to one another. The segments are joined at the centre by a hinge which is flanked by two passive rotating docking elements. In figure 3.3, a more recent 'T' shaped version is shown. The design was updated in order to improve traction and sensor coverage, the main alteration being the removal of one wheel, and the repositioning of the remaining three at 45° offsets.

Passive Module The purpose of passive modules is to help fulfil roles that are not covered by the other modules. For example, a passive module may carry a sensor that the other modules do not possess, or dedicated hardware that can be used for computationally intensive processing. Though they share a similar function to tool modules such as the Active Wheel, passive modules are not independently mobile and

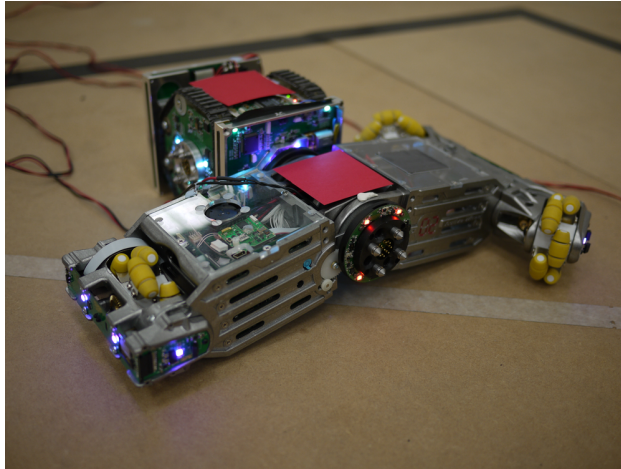


Figure 3.3: The newer ‘T’ shaped Active Wheel module, docked with a Scout robot

do not operate autonomously, they simply respond to the instructions of other modules.

Passive modules can take on a number of different forms. To remain compatible with the other Symbricator robots, their only requirement is that they possess a mechanism through which the other modules can communicate or interact with them. Figure 3.1d shows the frame and basic electronics of a passive module. The module approximates the dimensions of two Backbone robots, but since it contains none of the Backbone’s actuators, it is both lighter and has more internal space. Assuming that the module is also equipped with the necessary energy sharing and communication systems, by placing this module within a larger robotic structure and utilising its internal space to store more battery packs, such a module could be used to greatly increase the operating period of a collective structure.

3.1.2 Simulators

In this section, the two Symbricator simulators used throughout the remainder of this thesis are introduced. The first is the ‘Robot3D’ simulator [197], designed as part of the SYMBRION and REPLICATOR projects, this simulator provides accurate 3D models of all three Symbricator robots. The second is a modified version of the popular ‘Stage’ simulator [186], adapted to more closely emulate the docking and energy sharing capabilities of the Symbricator platform.

Robot3D Simulator The Robot3D simulator [197] includes 3D models of all three of the Symbricator robots. Several of the robots’ sensors and actuators are simulated, including actuators for 2D and 3D locomotion, docking elements and infrared (IR) sensors; as well as radio, IR and Ethernet communication. As shown in figure 3.4, Robot3D is able to simulate the behaviour of robots as individuals (a) and as part of collective robotic structures (b).

Robot3D uses the Open Dynamics Engine (ODE) to provide rigid body dynamics

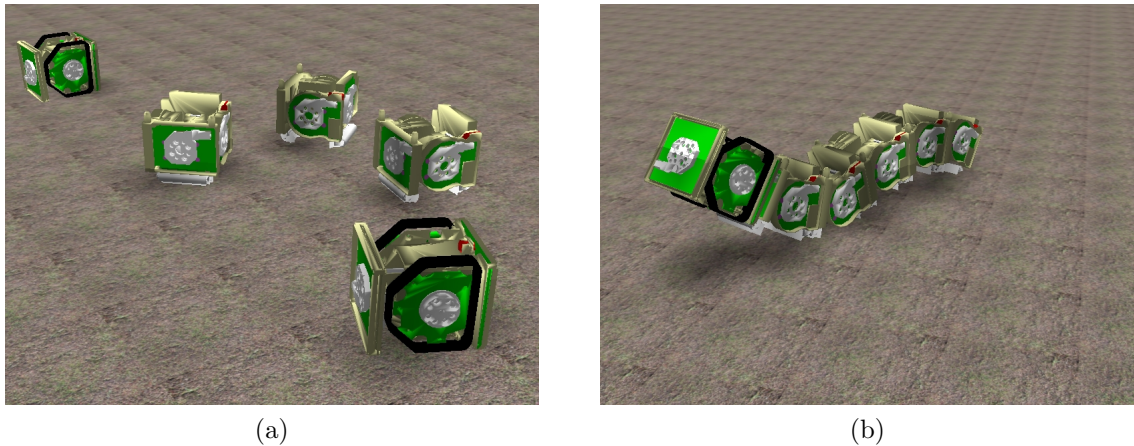


Figure 3.4: Screenshots from the Robot3D simulator, showing the behaviour of both individual modules (a) and a multi-robot structure (b)

simulation and collision detection. Simulation of visual sensors is implemented using Open Scene Graph (OSG). The mass and friction of the robot bodies are accurately simulated and simplified geometric models are used for collision detection. Various different actuators are implemented, including: a hinge joint, an omni-directional wheel and a screw drive. Two different methods of simulating the connection between docked robots are provided. The first uses an ODE fixed joint and allows the forces and torques that one robot applies to another to be calculated, however, under certain conditions this method can make the simulator unstable. The second solution merges the bodies of two connected robots, leading to a less accurate but more stable simulation. In all of the experiments reported in this thesis the second method is used.

The simulator serves two main purposes within the SYMBRION and REPLICATOR projects: firstly, it is used to aid in the design of robot controllers before their deployment on the real Symbicator hardware, and secondly it is used for research into the evolution of robot controllers and morphologies, which is naturally better suited to simulation. Originally known under the title ‘Symbicator3D’ [198] over recent years the simulator has undergone significant revisions. For use outside of the SYMBRION and REPLICATOR projects, Robot3D has now been released as an open source project¹.

Stage Simulator The Stage simulator [186] is a 2D robotics simulator, commonly used in conjunction with the Player plugin². Noted for its speed, scalability and ease of use, Stage supports the simulation of large numbers of independent robots, with basic sensors and actuators.

Stage simulates the environment as an array of cells (the size of which is configurable) and provides only first-order motion simulation (ignoring dynamics). Collision detection

¹<https://launchpad.net/robot3d>

²<http://playerstage.sourceforge.net>

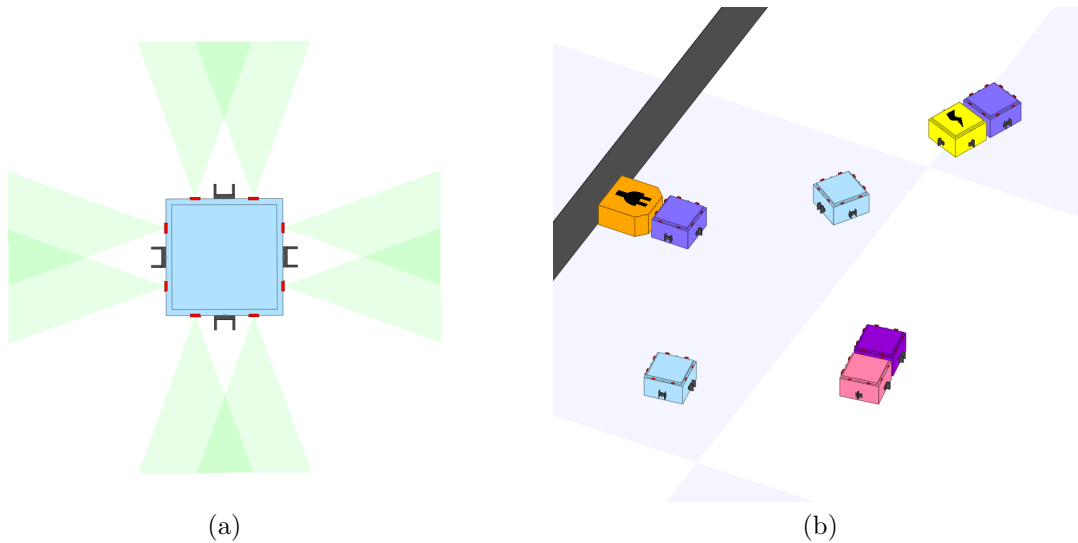


Figure 3.5: Screenshots from the Stage simulator, showing the original Symbricator model from [108] (a) and the updated models used within this thesis (b)

and range sensor data is computed using ray tracing with the granularity determined by the size of the underlying cells. The default cell size is 0.02 m, however, in the experiments described in this thesis, a resolution of 0.001 m was used.

The simulator was extended for use in the SYMBRION project with the addition of a model that more closely resembles the shape, size and functionality of the Symbricator robots [108]. The model includes accurate representations of the robot’s IR sensors, LEDs and remote receivers, based upon data gathered from real components [108]. To allow robots to dock with one another on a 2D plane, the model also includes a simplified representation of the robot’s docking elements. A screenshot from the adapted simulator is shown in figure 3.5a.

The simulator was further extended for this thesis with the addition of models for representing power sockets and passive modules, further details of which are provided in chapter 5. The original Symbricator model was also updated to allow robots to donate energy to each other, or to recharge themselves when connected to power sockets or passive modules. Figure 3.5b shows all three models in the updated simulator.

3.1.3 100 Robots 100 Days

First proposed by [88], the 100 Robots 100 Days grand challenge was designed to assess the ability of a group of robots to survive (remain functional) autonomously for long periods of time, without any human interaction. This property of long-term autonomy is believed to be essential if robotic systems are ever to fulfil their potential in tasks such as autonomous space exploration, environmental monitoring or search and rescue.

A sketch of the challenge from [88], is shown in figure 3.6. The essence of the

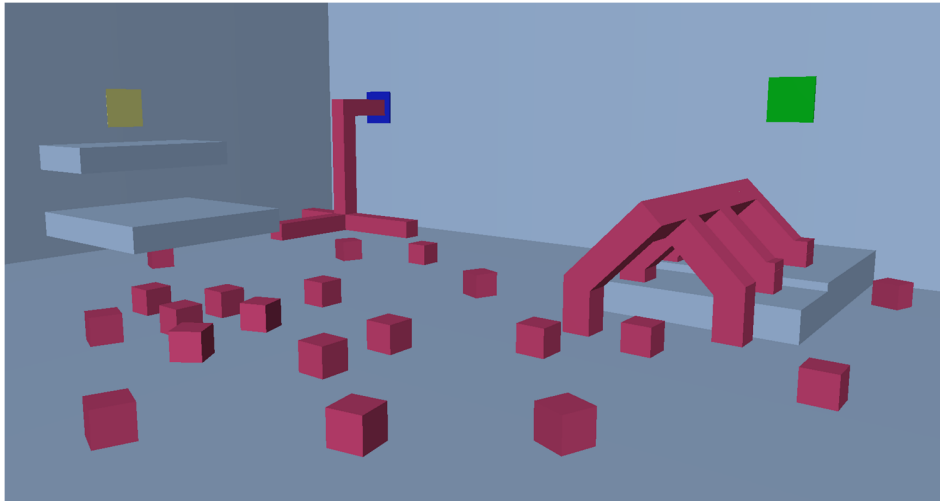


Figure 3.6: A graphical representation of the 100 Robots 100 Days challenge [88]

challenge is as follows. A large group of robots, assumed to be of the order of 100, are placed in a complex but structured environment, that is slowly changing over time. In the beginning, the environment is rich in energy with multiple power sources available. Some sources, in the form of power sockets, will be positioned in areas that are easy to reach for individual robots. Whereas others will be placed in more challenging regions, for example, several centimetres above the surface of the arena, or beyond obstacles. Over time, power sockets may turn on and off, altering their availability. To survive, the robots must monitor the availability and distribution of energy and adapt their behaviour to meet their demands. As time progresses further, stronger environmental pressure will force further adaptation. Those power sockets in easy to reach places will become fewer. To survive, the robots must cooperate and adapt to form larger robotic structures that can scale obstacles and utilise the sockets in the harder to reach regions. The challenge will end after a set period of time, in the order of 100 days.

During a period as long as 100 days, it is inevitable that some robots will run out of energy, whilst others will suffer electrical or mechanical failures. Throughout the challenge, there will be minimal human interaction, this means that the robots themselves will have to detect when a module has failed. Furthermore, when failures are detected, the robots should react appropriately, for example, by removing or replacing the failed individuals. At the end of the challenge, metrics such as the proportion of robots still functioning (and to what extent) as well as the amount of human interaction that was required, may be used to assess the performance of the system.

3.2 Chain

The main properties of the systems reviewed in this section are summarised in table 3.2. Images of the most influential of these platforms are shown in figure 3.7.

System	DOF	Connection Mechanism	Author	Year
Polypod	2 3D	Pin/Hole, SMA	Yim	1993
PolyBot	1 3D	Pin/Hole, SMA	Yim et al.	1998
CONRO	2 3D	Pin/Hole, SMA	Will and Shen	1998
CKBot	1 3D	Magnets	Yim et al.	2007
ModRED	4 3D	Pin/Hole, Solenoid	Dasgupta et al.	2010

Table 3.2: The main properties of each of the ‘chain’ platforms reviewed in this chapter

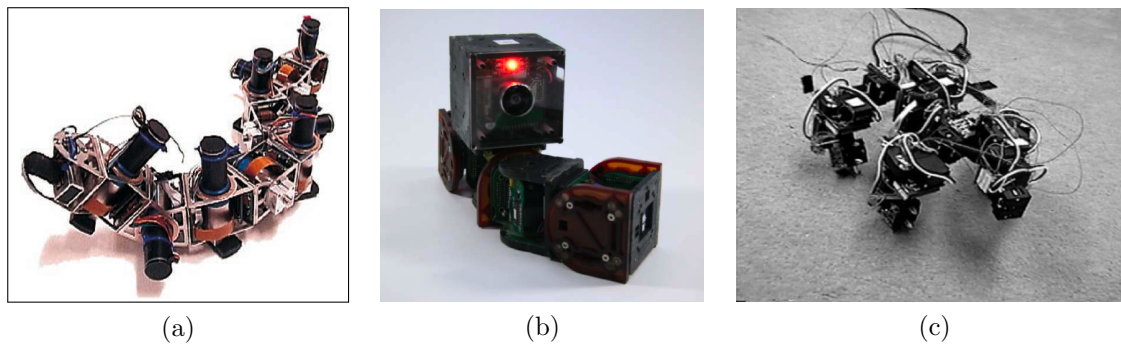


Figure 3.7: Three examples of chain-based self-reconfigurable modular robotic systems. From left to right, (a) shows eight PolyBot robots arranged in a snake configuration [203], (b) shows a group of four CKBots with a camera module attached [208] and (c) shows a quadruped structure assembled from CONRO robots [23]

One of the earliest examples of a chain-based modular robot was the *Polypod* system of Yim [199, 201]. The platform consists of two different types of module, a flexible *segment* that provides actuation, and a static *node* that supplies power and serves as a branching point within structures. Segments have two degrees of freedom and two connection plates for physically and electrically connecting separate modules. Nodes have no degrees of freedom but possess six separate connection plates. With this platform, Yim demonstrated several different forms of ‘statically stable’ locomotion [200]. Whilst apparently possessing the necessary sensors and actuators, it is not clear from the literature whether Yim was also able to successfully demonstrate autonomous docking with the Polypod platform. Therefore, it is not clear whether the platform can truly be described as *self*-reconfigurable, however, as a forerunner of the highly-influential *PolyBot* robot [41, 202], its importance should not be understated.

The PolyBot system also consists of two types of module, again referred to as nodes and segments. Multiple versions of the PolyBot robots have been developed, each one an incremental refinement of the last [204, 206]. The basic shape and functionality of the robots, however, has remained the same. Each segment possesses two connection plates which can be rotated using a single DOF between -90° and 90° . Upon each plate, the robots have both mechanical and electrical connectors. In later versions of the hardware, infrared transmitters and receivers allow the modules to autonomously dock with one another [205]. The robots have been used to demonstrate several different types of gait, including snake, caterpillar, spider, lizard and rolling loop motion [203, 212]. The platform is notable for being the first to demonstrate the automatic transition between two different modes of locomotion (rolling and snake-like motion) and for its repeated deployment within unstructured environments [41, 203, 206].

With colleagues from the GRASP Laboratory at the University of Pennsylvania, Yim et al. later developed another type of chain-based modular robot, known as the *CKBot* (Connector Kinetic roBot) [208]. The CKBot platform also consists of two different types of module, but unlike the Polypod and PolyBot platforms, both types of module possess a single DOF. The modules share similar electrical and mechanical components but have subtly different kinematic structures which allow them to be assembled into a greater number of arrangements than would be possible with only a single module type. Modules can be connected by hand using screws or autonomously connect with one another using magnetic interfaces [208]. When connected, the modules may share power and communicate using a CAN (Controller Area Network) bus. Each robot is also equipped with infrared receivers and transmitters that can be used for proximity detection and communication. In [144] the authors used these infrared sensors to provide communication fault tolerance during a distributed control task. Over recent years, the CKBot platform has been augmented with numerous extensions, including passive wheels [4], compliant legs [163], camera modules [208] and even a ‘foam generation device’ [152]. Using the foam generation device Revzen et al. demonstrated that a group of CKBots were able to modify their environment and form new robotic structures by creating links between existing CKBot groups ‘on-the-fly’. The platform is also well known for its demonstrations of ‘dynamic rolling’, which tested the ability

of loops containing between 8 and 12 modules to move both up-hill and down-hill at various speeds [164]. Of particular relevance to this thesis is the demonstration by Yim et al. of a group of 15 modules that is able to autonomously self-assemble after an ‘explosive’ event which breaks the system into three sub-structures [208].

At around the same time that Yim et al. were developing PolyBot, Will and Shen created the *CONRO* robot [22, 23]. CONRO modules are composed of three linearly connected segments. The central segment houses the core electronics, whilst the two outer segments contain docking connectors and infrared sensors for both intra-structure and inter-structure communication. The two outer segments are joined to the central unit by independent joints, which are perpendicularly offset to provide the module with both pitch and yaw DOF. One of the outer segments contains a single passive ‘male’ docking connector with protruding pins, and the other contains three active ‘female’ connectors which incorporate a latch for securing the connection between two modules. Docking has been successfully demonstrated between two modules within the same structure [166] as well as two modules within separate structures [154]. Methods for determining the configuration of a structure [21] and distributed hormone-inspired approaches for controlling assembled structures [161, 167] have also been proposed.

More recently, in 2010, Nelson et al. introduced *ModRED* (Modular Self-reconfigurable Robot for Exploration and Discovery) [32, 132]. The individual ModRED robots are composed of four linearly connected segments, the two outer segments contain docking brackets, whilst the two central segments contain actuators and drive-train components. Two rotational degrees of freedom join the docking brackets to the central segments, whilst a single translational degree of freedom—allowing for expanding and contracting motion—and a further rotational degree of freedom, separate the two central segments. Despite possessing more DOF than most other chain-based modular robots, the fact that the modules have only two docking connectors severely limits the type of structure that modules may form. Most early work with the ModRED system has been conducted in simulation. Although the problem of determining when and how robotic structures should be reconfigured has formed a large part of this early work [43, 149], the authors have yet to demonstrate autonomous docking with real robotic hardware.

3.3 Lattice

The main properties of the systems reviewed in this section are summarised in table 3.3. Images of the most influential of these platforms are shown in figure 3.8.

Over the last 25 years, a large number of lattice platforms have been developed. A variety of different module designs have been proposed, ranging from the very small to the reasonably large. A number of different connection mechanisms have been developed for physically joining modules and a variety of different actuation methods have been proposed for controlling and reconfiguring systems.

Examples from the smaller end of the scale include the *Miche* [53, 54] and *Smart Pebble* systems [51, 55] developed at the MIT Computer Science and Artificial Intelligence

System	DOF	Connection Mechanism	Author	Year
Fracta	3 2D	Magnets and Electromagnets	Murata et al.	1994
Metamorphic	6 2D	Mechanical Clamp	Chirikjian et al.	1996
Molecule	4 3D	Electromagnets	Kotay and Rus	1998
3D Fracta	6 3D	Mechanical Clamp & Cuff	Murata et al.	1998
I-Cube	3 3D	Mechanical Lock and Key	Ünsal and Khosla	2000
Crystalline	1 2D	Mechanical Lock and Key	Rus and Vona	2000
Telecube	6 3D	Magnets	Suh et al.	2002
ATRON	1 3D	Mechanical Hooks	Stoy et al.	2004
Prog. Parts	0 2D	Magnets	Klavins	2005
Catom	0 2D	Electromagnets	Goldstein et al.	2005
Miche	0 3D	Magnets	Gilpin and Rus	2007
Smart Pebble	0 3D	Electromagnets	Gilpin and Rus	2010
Fluidic Assembly	0 3D	Fields metal	Neubert et al.	2010

Table 3.3: The main properties of each of the ‘lattice’ platforms reviewed in this chapter

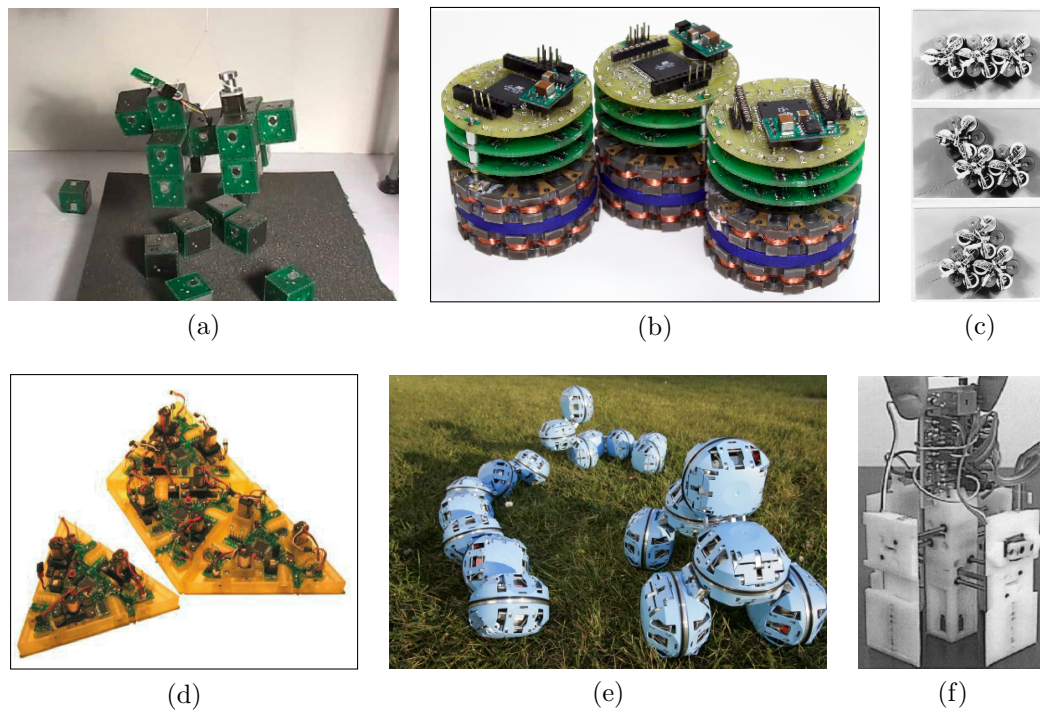


Figure 3.8: Six examples of self-reconfigurable lattice platforms. In the top row, (a) shows a 15 module ‘dog’ assembled from 27 Miche robots [53], (b) shows three Catom robots [91] and (c) shows three Fracta modules transforming from a linear shape into a more densely packed arrangement [126]. In the bottom row, (d) shows three Programmable Parts performing self-assembly [12], (e) shows three different structures assembled from ATRON modules [140] and (f) shows a single Crystalline robot [158]

Laboratory. Envisaged as a test bed for future systems of *programmable matter*, using magnetic connectors these small, immobile, cube shaped modules may self-assemble with the help of an external stochastic force, for example, a vibrating table. Once assembled, a distributed strategy of self-*disassembly* is used to ‘sculpt’ the desired object from the densely packed robotic substrate [50, 52, 56].

Other examples from the field of programmable matter include Goldstein et al.’s *Catom* [57] and Neubert et al.’s *fluidic assembly* modules [133]. Designed as part of the Claytronics project³ the *Catom* is a small cylindrical module which relies on an array of electromagnets to connect modules on a two-dimensional plane. By coordinating which magnets are on and which are off, modules can reconfigure by revolving around one another. A novel design developed for a more recent prototype allows the same magnetic interfaces to be used for power transfer and communication [91].

Neubert et al.’s fluidic assembly modules employ a very different approach to self-reconfiguration. Modules are suspended in a fluid filled chamber with a single seed attached to an external power source. Guided by the turbulent flow of liquid within the chamber, new modules are attracted to the seed and connect to it using a novel mechanism which effectively solders the robots together. Connected modules share power with one another and are able to channel the flow of liquid through their bodies by adjusting a set of values. The flow of liquid through a structure alters the turbulence in the chamber and therefore allows the modules to direct the assembly process.

Most early work into lattice-based modular robotics focused on 2D systems. The *Fracta* modules of [124] and the *Metamorphic* robots described by [143] are two good examples. *Fracta* modules have a six-lobed structure that approximates a hexagon. To reconfigure they employ a combination of permanent magnets and electromagnets, allowing modules to move around one another in a manner not dissimilar to that of Goldstein et al.’s *Catoms* [57]. Tomita et al. describe a self-assembly algorithm that allows a group of *Fracta* to autonomously reconfigure into a predefined shape [179]. Extending the self-assembly method Tomita et al. also demonstrate how the system may self-repair in three steps by (1) removing failed modules, (2) degrading to an earlier state of assembly and (3) re-initiating the assembly process.

Pamecha et al. describe two different types of *Metamorphic* robot [143]. The first of which consists of six articulated links, arranged in a hexagon shape. The robots are able to dock with one another using a clamping mechanism and locomotion is achieved by deformation of the hexagon shape, allowing the modules to effectively ‘roll’ around one another. The second type of robot is square and uses a shuttle and jaw connecting mechanism, allowing the robots move like pieces in a sliding block puzzle.

Another, more recent, example of a 2D-lattice system are Bishop et al.’s *Programmable Parts* [12]. Bishop et al.’s ‘parts’ are triangular and rely on magnets to connect with one another. Self-assembly is performed in a stochastic manner by randomly ‘mixing’ the modules on an air table. Modules join with one another following chance collisions, but decide whether to remain connected using a set of predefined

³<http://www.cs.cmu.edu/~claytronics/>

graph grammar rules.

Following on from their 2D work with Fracta, Murata et al. created one of the earliest examples of a 3D modular lattice platform. The modules in Murata et al.'s *3D Fracta* [125] system are roughly cubic in shape, with a single rotatable arm protruding from each face. Mechanical grippers at the end of each arm allow the modules to connect, communicate, and share power with one another. Expanding upon their earlier work in two dimensions, Murata et al. also present a 3D self-assembly algorithm.

The *ATRON* platform from [80, 140] represents another good example of a 3D system. Each ATRON module consists of two rotating hemispheres and connects with other modules using a mechanical hook mechanism. The robots have been used to demonstrate behaviours including collective locomotion [139], self-reconfiguration and self-repair [30].

In contrast to the systems introduced thus far, the *Molecule* [93] and *I-Cube* [182] platforms may be described as 'bi-partite' in structure. Both systems are made up of a collection of cubic modules that are joined to each other by a set of bonds, or 'links'. In the case of the Molecule system, these bonds are rigid and are always permanently attached to two modules. Each module, referred to as an 'Atom', has five electromagnetic connectors, one of which possesses a single rotational degree of freedom, and four of which are static. To further aid reconfiguration, the Atom's have a second rotational degree of freedom about the point at which they are connected to the bond. In [92] Kotay and Rus present a language for describing the motion of Molecule robots, and using it, present a series of algorithms for performing various primitive behaviours.

In contrast to the Molecule platform, in the I-Cube system, the modules and links are physically independent. Despite their autonomy, the two parts are mutually reliant upon one another in order to perform reconfiguration. Links are joined to modules, referred to as 'cubes', using a lock-and-key style mechanism. To enable reconfiguration, links are articulated at the centre and have one rotational degree of freedom at either end. Links provide actuation whilst cubes provide power, computation, and sensing. Ünsal et al. describe experiments with prototype hardware that demonstrates the movement of a link from one face of a cube to another, the passing of a link between two cubes, and the passing of a cube between two links [183].

Inspired by the expanding and contracting behaviour of muscles and amoebas, the creators of the Molecule robot later produced the *Crystalline* system [157, 158]. Crystalline modules are square in shape with each side connected to a common actuator that allows them to expand and contract across a 2D plane. Docking is performed using a lock-and-key mechanism, with each module possessing two active 'key' connectors and two passive 'lock' connectors. In [158] Rus and Vona describe how structures may move using an inchworm like motion and how reconfiguration may be performed by transporting individual modules from one area of the lattice to another. In contrast to most other lattice systems, in which the modules reconfigure themselves by moving over or around the lattice, in the Crystalline system, the modules use an expansion/contraction behaviour that repeatedly shifts modules by one place in the grid, effectively allowing modules to move 'through' the lattice.

System	DOF	Connection	Author	Year
M-TRAN I	2 3D	Magnets/SMA Coil	Murata et al.	2002
M-TRAN II	2 3D	Magnets/SMA Coil	Murata et al.	2003
Molecubes	1 3D	Magnets/Pin and Socket	Zykov et al.	2005
M-TRAN III	2 3D	Mechanical Hooks	Kurokawa et al.	2006
SuperBot	3 3D	Mechanical Clamp	Shen et al.	2006
Roombots	3 3D	Mechanical Latch	Sproewitz et al.	2008

Table 3.4: The main properties of each of the ‘hybrid’ platforms reviewed in this chapter

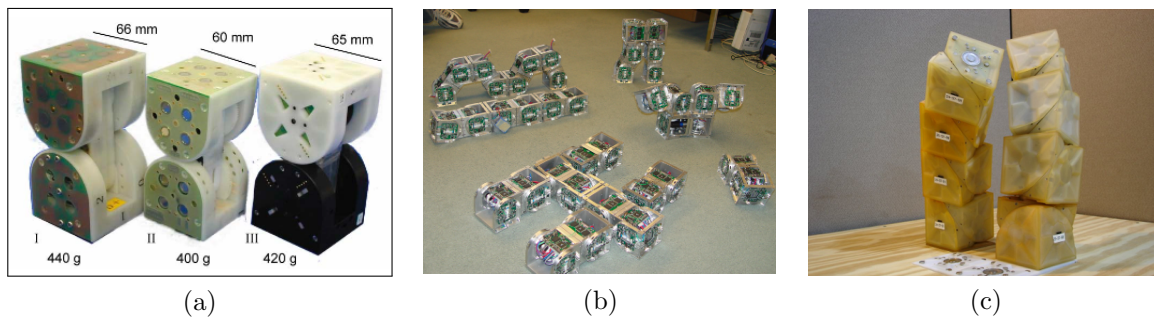


Figure 3.9: Three examples of self-reconfigurable hybrid platforms. From left to right, (a) shows the three different versions of the M-TRAN robot [99], (b) shows multiple SuperBot modules arranged in various different configurations [26] and (c) shows eight Molecube robots performing self-replication [217].

Using their *Telecube* robot [175], Suh et al. later realised the expandable module concept in three dimensions. In the *Telecube* system, the expansion of each side of a module is controlled by an independent piston. Each face plate contains two permanent switching magnets for connecting with other modules; an infrared transmitter and receiver for inter-robot communication; and electrical contacts for power transfer. In [185] the authors describe a reconfiguration algorithm in which control is abstracted to the level of ‘meta-modules’, each of which is composed of eight *Telecubes*.

3.4 Hybrid

The main properties of the systems reviewed in this section are summarised in table 3.4. Images of the most influential of these platforms are shown in figure 3.9.

The earliest example of a modular robotic system that combined aspects from both lattice and chain architectures was the Modular TRANSformer (M-TRAN) platform of Murata et al. [127]. Through its innovative design, the M-TRAN system combines the ease with which lattice type systems may be reconfigured, with the flexibility of locomotion found in chain type systems [127]. Since 1999, three different versions of the

M-TRAN system have been developed. The modules of the first version (*M-TRAN I*) are formed of two cubic segments. The segments are separated by a single link with one controllable DOF at either end. One segment is referred to as the ‘active box’ and the other as the ‘passive box’. Both segments contain three (permanent) magnetic docking interfaces, but the active box also contains a spring-loaded detachment mechanism, controlled by a shape memory alloy (SMA) coil. The two hardware revisions that followed *M-TRAN I* maintained the same two-segment structure as the original but improved upon both the sensing and actuation capabilities of the robots.

In the second version of the M-TRAN system, *M-TRAN II* [97], the overall size of the modules was reduced and the individual robot’s computational power was increased, as was both the effectiveness and efficiency of their (un-)docking mechanism. With these modules, Kurokawa et al. successfully demonstrated the transformation between quadruped and caterpillar structures [97], as well as the (remotely operated) assembly of a single structure from two smaller ones [98]. Meanwhile, in [210] and [82] the authors presented methods for coordinating the motion of such structures, using controllers evolved and optimised with genetic algorithms (GA).

In the third and most recent version of the M-TRAN hardware [100] the magnetic docking connectors were replaced with motor driven hooks to provide a faster, stronger and more reliable connection between modules. Despite being slightly larger and heavier than the previous version, the *M-TRAN III* robots consume less power. The modules also possess infrared sensors, although these were present on some versions of the M-TRAN II hardware [82], they were not widely utilised. With the M-TRAN III robots, the infrared sensors were used to provide a simple form of local communication [99]. With the addition of a specialised camera module, the transmitters of the IR sensors were used as markers for a vision based docking approach [128].

Inspired by a combination of Murata et al.’s M-TRAN modules [97, 100, 127], and their own previously developed CONRO robots [22], in 2006, Shen et al. introduced the *SuperBot* platform [162, 168]. Like the M-TRAN robots, the SuperBot modules are composed of two cubic segments, each of which possesses a single DOF. However, unlike M-TRAN, the link that joins the two segments of a SuperBot can rotate, providing the robots with a degree of freedom in each of the three principal axes of rotation: yaw, roll and pitch. The other main difference between the SuperBot and M-TRAN systems is found in the robots docking mechanisms. Whereas the M-TRAN modules possess both ‘male’ (active) and ‘female’ (passive) connectors, the connectors used by the SuperBot robots are described as ‘genderless’ [169], meaning that every side of every module is identical, and resulting in a greater number of possible structural configurations. Furthermore, the unique design of the SuperBot connector means that one robot may undock from another without that module’s cooperation. This one-sided disengagement is useful to prevent failed modules from adversely affecting the remainder of a robotic structure from which they cannot be removed. A large amount of work with the SuperBot robots has been focused on the task of locomotion, and in particular on that of controlling the motion of a group of robots arranged within a loop structure. In such a configuration, the SuperBot robots have demonstrated the

System	DOF	Connection	Author	Year
CEBOT	1 2D	Mechanical Hooks	Fukuda et al.	1988
Swarm-bot	3 2D	Mechanical Gripper	Mondada et al.	2004
Symbrictor	3 3D	Mechanical Latch	Kernbach et al.	2010
Sambot	1 3D	Mechanical Hooks	Wei et al.	2010
X-Cell	1 2D	Electromagnets	Hong et al.	2011
Swarmanoid	3 2D	Mechanical Gripper	Dorigo et al.	2013
SMORES	4 3D	Magnets	Davey et al.	2012

Table 3.5: The main properties of each of the ‘mobile’ platforms reviewed in this chapter

ability to roll continuously for 54 minutes, covering a distance in excess of 1 km [26], as well as robustness in their ability to ‘self-recover’ if the loop falls onto its side [25]. Like the PolyBot robots, the SuperBot platform is notable for having demonstrated its capabilities within challenging real-world environments, including climbing sand dunes [26] and both vertical and horizontal ropes [150].

Two final hybrid systems, which both employ a similar design, are Zykov et al.’s *Molecubes* platform [215] and Sproewitz et al.’s *Roombots* system [170, 172]. Both systems contain units that are roughly spherical in shape and like the modules of the ATRON platform, each unit is composed of two rotating hemispheres. In the Roombots system, each module is made up of two of these primitive units, permanently joined on one face by a single rotating joint, therefore providing each complete module (two units) with a total of three DOF. Each Molecubes module, on the other hand, contains only a single unit, and therefore possesses only a single DOF. The platforms also differ in terms of their connection method. Roombots utilise a mechanical latch [171], whereas Molecubes favour a pin and socket style connector. Early Molecube prototypes were shown to be capable of a basic form of self-reproduction [216, 217], whilst Roombots are envisaged as a future platform for self-reconfigurable furniture [173].

3.5 Mobile

The main properties of the systems reviewed in this section are summarised in table 3.5. Images of the most influential of these platforms are shown in figure 3.10.

One of the earliest forms of mobile self-reconfigurable robot, and one of the earliest forms of modular robot in general, was the *CEBOT* (cellular structured robot) platform of Fukuda et al. [47, 48]. The platform may be described as *heterogeneous*, consisting as it does of multiple different types of module. In [47] the authors describe three different types of robot ‘cell’, one capable of bending, one capable of rotation and one that is independently mobile. Each cell type contains compatible docking interfaces, based upon a bolt and hole mechanism, and locked by an SMA spring. In [48] an improved docking mechanism is described that uses a cone shaped docking connector

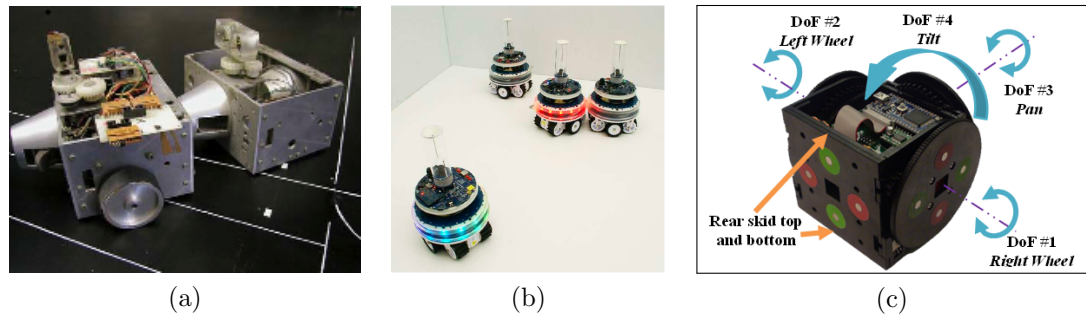


Figure 3.10: Three examples of self-reconfigurable mobile platforms. From left to right, (a) shows two CEBOT modules (image from [60]), (b) shows a group of four ‘foot-bots’ from the Swarmanoid project [40] and (c) shows a single SMORES module with each of its four independent DOF marked [37].

with hooks either side to hold the connection in place. Improved sensing capabilities are also introduced, including infrared and ultrasonic sensors. In [46], the authors describe their vision of the CEBOT platform as an intelligent universal manipulator and describe some of the methods that are required to achieve this function.

Despite Fukuda’s pioneering work, in early modular robotics research, the concept of a mobile self-reconfigurable platform was largely ignored, with most researchers tending to focus on chain and lattice based systems. In recent years, however, a number of different mobile platforms have emerged. This resurgence was led by the *s-bot* platform [121] from the Swarm-bot project⁴, and the *foot-*, *hand-* and *eye-* bots [40] of the succeeding Swarmanoid project⁵. Each s-bot possesses a combined track and wheel (‘treel’) differential drive system, an array of sensors, an omnidirectional camera and grippers for physically connecting with neighbouring modules. As an evolution of the s-bot, the foot-bot possesses a similar set of sensors and actuators, but housed in a more robust and slightly larger platform. The hand- and eye-bots are designed to supplement the foot-bot with additional sensors and actuators, resulting in a highly flexible heterogeneous swarm. Although unable to create arrangements as complex as those produced by other modular robotic systems, the hand- and foot-bots may still physically connect to form simple 2D structures. The Swarm-bot and Swarmanoid platforms have been used to developed several distributed methods of self-assembly [62], self-reconfiguration [135] and collective recovery [137].

The Symbricator platform [88] from the SYMBRION and REPLICATOR projects, introduced in section 1.3, represents another good example of a heterogeneous self-reconfigurable modular robotic system. Like the modules from the Swarm-bot and Swarmanoid projects, the individual robots are independently mobile. However, unlike the s-bot and its derivatives, the modules are also designed to be capable of forming complex 3D structures. The platform consists of three unique, yet complimentary,

⁴<http://www.swarm-bots.org/>

⁵<http://www.swarmanoid.org/>

modular forms, each possessing its own set of functionalities, and accordingly, its own set of sensors and actuators. Although structurally, mechanically and electronically distinct, all three types of module share a common mechanical docking interface. As summarised in section 1.3, research within the two projects has included the development of evolvable distributed controllers for collective locomotion tasks [68, 70], strategies for self-assembling robotic structures [89, 108] and immune-inspired algorithms for fault detection [120] and diagnosis [11].

The *Distributed Flight Array* (DFA) of [141] represents a very different type of mobile self-reconfigurable system. The DFA modules are hexagonal and measure approximately 25 cm in length. At the centre of each module is a single 3-blade propeller and surrounding it are three omnidirectional wheels. The modules may move along the floor as individuals, but when connected, form a multi-rotor vehicle that is capable of autonomous flight. Protruding features of the robots chassis allow for passive alignment and docking, and magnets ensure that docked modules remain connected. The strength of the magnets was chosen to allow modules to break apart only if sufficient force is applied, for example when a flying structure falls to the floor. Early experiments by [141] verify the ability of the modules to dock and fly in a connected group.

Two other examples of mobile systems are Wei et al.'s *Sambot* [191] and Hong et al.'s *X-Cell* [76]. Both systems approximate cubes with 8 cm sides, and both are made mobile by a two wheeled differential drive system. The X-Cell modules consist of two parts, a main unit which houses the core electronics and battery, and a square faced shell upon which sensors and docking connectors are mounted. A servo motor allows the shell to rotate independently of the base, in the range of $0 - 180^\circ$. The Sambot modules possess four passive docking sides and one active side, mounted on a rotating arm. In contrast to X-Cell, which may only form structures within a 2D grid, the rotating arm of the Sambots allows them to form 3D chain-like structures. Using the Sambot platform, Wei et al. have demonstrated basic alignment and docking, quadruped assembly and caterpillar motion [192]. Like Wei et al., using the X-Cell platform, Hong et al. have also demonstrated basic infrared-based alignment and docking [76].

One final, more recent, mobile system is the *SMORES* (Self-assembling MOdular Robot for Extreme Shape-shifting) platform of Davey et al. [37]. The SMORES system promises the ability to perform reconfiguration using lattice, chain and mobile strategies. Each module is articulated at the centre and has four docking connectors, one of which is passive and three of which can actively rotate. The active docking connectors also serve as wheels for individual module locomotion. Robots are connected using permanent magnets and disconnect by rotating their docking elements until the magnets repel one another. At present, the robots are not equipped with any sensors, but have successfully demonstrated the ability to assembly and reconfigure under remote control.

3.6 Summary

Over the past five years, the SYMBRION and REPLICATOR projects have overseen the development of three new robotic platforms and two new simulators. The hardware platforms, collectively referred to under the title ‘Symbricator’, and known individually as the Backbone, Scout and Active Wheel, each have their own set of specialist functionalities. The Backbone was designed to serve as the core component of multi-robot structures, the Scout was designed to perform surveillance and sensing tasks, and the Active Wheel was designed to provide support for the others. The Robot3D simulator, which contains 3D models of the Symbricator robots, was developed specifically for SYMBRION and REPLICATOR projects. Meanwhile, in order to provide a fast 2D simulation environment, the Stage simulator was adapted for use within the projects.

Members of the SYMBRION and REPLICATOR projects have proposed the 100 Robots 100 Days grand challenge, which will be used to assess the ability of self-reconfigurable modular robotic systems to survive autonomously, for long periods of time, without any form of human interaction. Over such a long period of time, it is anticipated that some robots will suffer failures, and therefore, to survive, the system must be capable of handling faulty robots. That is to say, the robots must be capable of demonstrating fault tolerance. To do so will require techniques for detecting faults in individual robots and strategies for recovering a system that contains failed individuals.

According to the taxonomy of Yim et al. [207], there are four main types of self-reconfigurable modular robotic system: chain, lattice, hybrid and mobile. These different types of platform are distinguished according to the manner in which they reconfigure themselves, and may be further be classified according to the number of degrees of freedom that the individual modules possess, the type of docking mechanism they utilise and whether reconfiguration is performed in a deterministic or stochastic manner.

Chain-based systems include the Polypod [199, 201] and succeeding PolyBot [204, 206] platforms of Yim et al.. The latter of which is notable for being the first platform to demonstrate the transformation between two separate modes of locomotion, and for its repeated deployment in real-world unstructured environments. Of particular relevance to this thesis, is the CKBot platform [208], which has been used to demonstrate the ability to self-repair, following an event which breaks the system into several parts.

The majority of self-reconfigurable modular robotic systems belong to the lattice category. Relevant systems include the Fracta [124] and ATRON [80, 140] platforms, which have both been used to demonstrated different forms of self-repair. Other interesting examples include the Programmable Parts [12] and Catom robots [57], which, despite the individual modules not possessing any independent degrees of freedom—and in the case of Catom, no moving parts—are still capable of demonstrating complex self-reconfiguration on a 2D plane.

Hybrid platforms are those which combine elements of chain and lattice based systems. One of the earliest such systems was the M-TRAN series [97, 100, 127] of Murata et al. Said to combine the ease with which lattice systems can be reconfigured, with the flexibility of locomotion that chain type systems provide, the M-TRAN robots

have been used to demonstrate several forms of locomotion, self-assembly and self-reconfiguration. The equally influential SuperBot [162, 168] platform of Shen et al., is memorable for its ‘genderless’ docking connector, long-distance rolling experiments (including self-recovery) and its placement within challenging real-world environments.

The Symbicator robots, used in chapters 4-6 of this thesis, and the new platform extension introduced in chapter 7, both fall into the category of mobile self-reconfigurable modular robotic systems. Other members of this category include the robots from the Swarm-bot and Swarmanoid projects [40, 121], which have both been used to demonstrate highly relevant behaviours such as fault detection, self-assembly and collective self-recovery. Meanwhile, the mobile DFA [141] and X-Cell [76] platforms, although bearing less relevance to the Symbicator robots, share several commonalities with the new platform described in chapter 7.

This chapter showcased the wide range of self-reconfigurable modular robotic systems that have been developed, but also highlighted the high complexity and cost of many of these systems. Due to their cost and complexity, modular robots are typically very limited in their accessibility. Very few platforms are available to buy or have been released as open hardware projects. This is in direct contrast with the field of swarm robotics in which several low-cost, low-complexity platforms have been developed and are available either commercially, or as open source projects. The cost and limited availability of modular robotic systems was one of the main motivations for the development of the platform introduced in chapter 7. As an extension to an existing swarm robot, the platform combines benefits from both swarm and self-reconfigurable modular robotic systems in order to provide a simple, low-cost system, that may be used to investigate the interesting properties of self-reconfigurable modular robotics from a simplified level.

CHAPTER 4

Reliability Analysis and Morphogenesis

In this chapter, the reliability of a morphogenesis controller—originally developed by Liu and Winfield [108] as part of the SYMBRION project—is analysed using two different techniques from the field of reliability engineering. The techniques, Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA), are used to analyse and compare two variants of the controller. Following analysis, the reliability of the system is discussed and areas where improvements could be made are suggested. The techniques of FMEA and FTA are also compared with one another and their usefulness as aids to the design of fault tolerant robotic systems is considered.

The reliability study described in this chapter was performed using the Symbicator platform, and specifically the Backbone robots. These robots were chosen due to their availability within the SYMBRION project and their use in the original work by Liu and Winfield. It is highlighted, however, that the analysis techniques described here are sufficiently general that they could be applied to other types of self-reconfigurable modular robotic system, and to different forms of controller.

This chapter continues in section 4.1 by describing the morphogenesis controller in detail. In sections 4.2 and 4.3, the FMEA and FTA techniques are introduced and then used to analyse a system of Symbicator robots running the morphogenesis controller. In section 4.4, the outcomes of the analysis are discussed and the FMEA and FTA procedures themselves are compared. Finally, in section 4.5, a summary is presented and some potential avenues of future work are suggested.

4.1 Morphogenesis Controller

The morphogenesis controller of Liu and Winfield [108] was designed to allow a group of Symbicator robots to self-assemble into a robotic structure of predetermined size and shape. Once assembled, the structure may either completely disassemble, or partially disassemble, allowing for the efficient transformation into a different configuration.

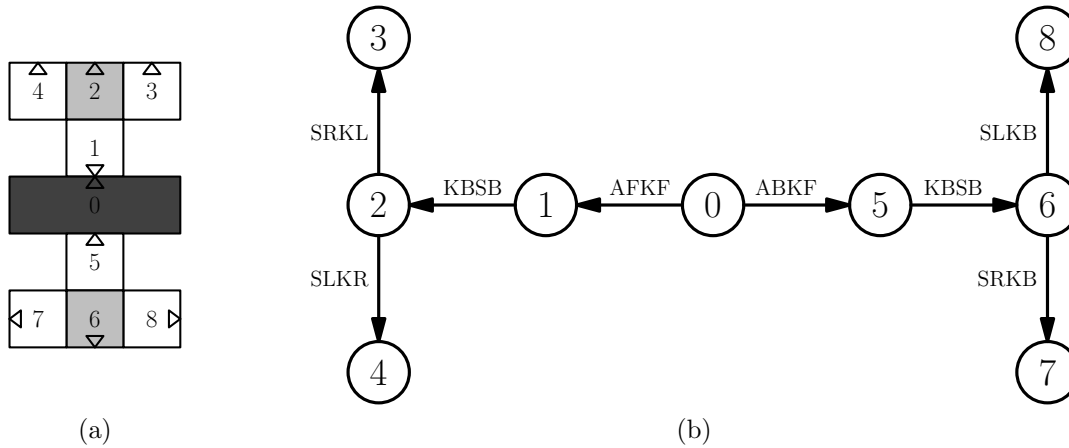


Figure 4.1: A top-down view of a multi-robot structure (a), alongside its corresponding graph representation (b). The structure in (a) is composed of 6 Backbone modules (shown in white), two Scout robots (light grey) and one Active Wheel (dark grey). The small arrows in (a) signify the robots' headings. The symbols on the edges of the graph in (b) describe the manner in which the two modules associated with the edge are connected to one another

In this chapter, it is assumed that the target system is composed entirely of Backbone robots, however, in chapter 6, structures containing Backbone, Scout and Active Wheel modules are investigated. It is recalled from section 3.1.1 that each Backbone robot possesses four active docking elements and that for a reliable connection to be established between two robots, at least one of the pair must have their docking element locked. The robots also possess a range of infrared (IR) devices which are used for both navigation and communication (full details of which are provided in section 3.1.1). Specifically, the morphogenesis controller introduced in this section makes use of the IR sensors, for obstacle and robot proximity detection; the IR LEDs, to allow robots to act as beacons or broadcast messages to their neighbours; and the IR remote receivers, for detecting the messages sent by others.

The morphogenesis process is initiated and coordinated by a single 'seed' module which possesses a plan for the structure that the robots are required to assemble. However, before introducing the controller itself, it is important to first describe the way in which robotic structures are internally represented by the individual modules.

4.1.1 Structural Representation

The method that the robots use to encode structural configurations has undergone several revisions since it was first published in [108]. The most up to date method represents structures as directed acyclic graphs (DAGs) and is visualised in figure 4.1.

Figure 4.1a shows a top-down view of a potential arrangement of 9 Symbricator robots, whilst figure 4.1b shows the same structure represented as a graph. Each vertex

Robot types		Docking sides	
K	Backbone	F	Front
A	Active Wheel	R	Right
S	Scout	B	Back
		L	Left

Table 4.1: Symbols used to represent different types of robot and different docking sides

in the graphical representation corresponds to a robot, and each edge to a connection between two robots. The direction of an edge represents which of the two modules is responsible for recruiting the other. For example, in figure 4.1b, as shown by the direction of the edges which connect their vertices, module 6 is responsible for recruiting modules 7 and 8. In every graph, there will be a single node with an indegree of zero, the corresponding module (for which no other robot is responsible for recruiting) represents the seed robot. The seed of the structure shown in figure 4.1, therefore, is module 0.

Every edge of a structure’s graph contains a string of four characters which represents the type and connected sides of the two corresponding modules. For example, in figure 4.1, the edge joining vertices 0 and 5 is labelled ‘ABKF’. The first two symbols in this string represent the type and recruiting side of module 0, whilst the second two represent the type and docking side of module 5. The symbols ‘A’ and ‘K’ represent Active Wheel and Backbone modules respectively, whilst the symbols ‘B’ and ‘F’ are used to represent the back and front sides of modules. Therefore, the string ‘ABKF’ represents the fact that an Active Wheel (A) is required to recruit, on its rear side (B), a Backbone module (K), docked using its front side (F). The symbols used to encode the other sides and module types are summarised in table 4.1.

Internally, the robots represent graphs using a string of symbols. The string which corresponds to a particular graph can be constructed by performing a depth first traversal of the graph (starting with the seed). For every edge that is explored, the corresponding symbols are copied to the string, and for every backtracking step that is performed, four NULL symbols are added. The result, as shown in figure 4.2, is the construction of a string in which every set of four characters has a matching set of NULL symbols, and everything between the two represents one branch of the structure.

4.1.2 Controller

Every robot in the system runs the same behavioural controller. A finite state machine (FSM) for the controller is shown in figure 4.3. The behaviours above the dashed line are executed by modules that are currently part of the structure being assembled, and the behaviours below the line are executed by physically independent robots.

Whilst figure 4.3 shows the behaviours of the individual robots, it is also possible to identify ‘system level’ behaviours which arise from the *interactions* of robots executing the individual behaviours. At least three such system level behaviours may be iden-

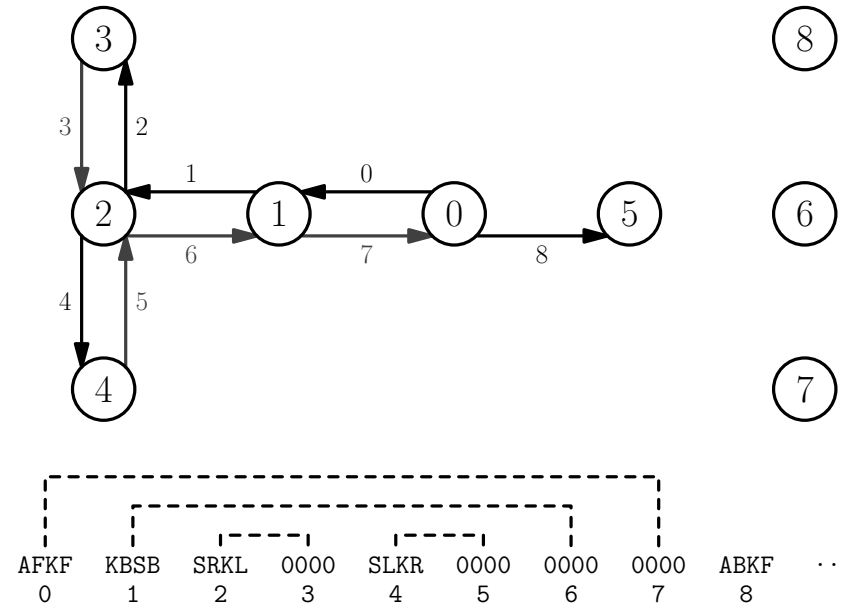


Figure 4.2: The partial construction of a string-based representation of the structure from figure 4.1a. Arrows show the order in which vertices are encountered during a depth first traversal of the graph (top). Black arrows represent exploratory steps, whilst grey arrows show backtracking steps. The order in which nodes are traversed corresponds directly to the order in which symbols are added to the string (bottom)

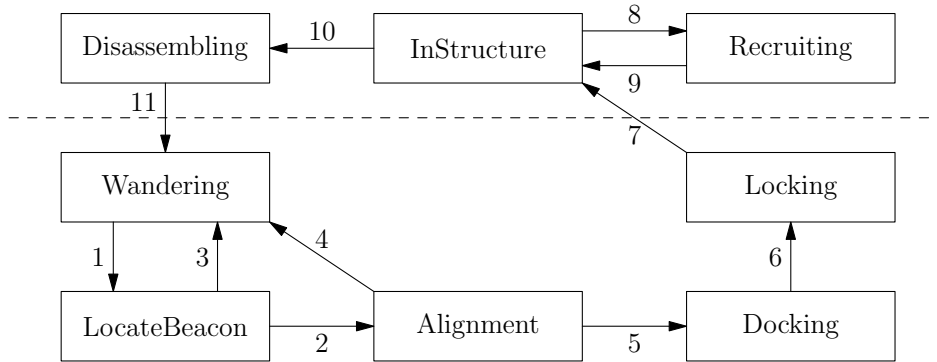


Figure 4.3: A finite state machine for the morphogenesis controller of [108]. Each state corresponds to a single behaviour. The behavioural states positioned above the dashed line are executed by modules which belong to a larger robotic structure. The behaviours positioned below the dashed line are performed by independent unconnected robots, which are not considered to be part of a robotic structure

tified: Exploration, Self-assembly and Self-disassembly. Exploration simply describes the ability of robots to explore their environment whilst avoiding obstacles and other robots. Exploration is essential to ensure that robots are able to locate the site of the forming structure. Self-assembly is the main behaviour observable in the system and incorporates all of the mechanisms through which individual robots combine to form a new structure. Finally, Self-disassembly, involves all of the methods through which robots coordinate their behaviour in order to disassemble a robotic structure. These three behaviours together allow the system to exhibit the property of autonomous morphogenesis. Each of the three system level behaviours is now introduced in turn, with reference to the individual behaviours that are combined to give rise to them.

Exploration

Exploration is provided by the interactions of robots in the Wandering state. In this behaviour, robots simply move forward and, if an obstacle is encountered, they change direction to avoid it. This strategy is sufficient to ensure good coverage in an enclosed arena with no internal obstacles. It is acknowledged, however, that for more complex environments, a different strategy may be required. As suggested in [108], this behaviour can be replaced by other swarm behaviours (such as ‘foraging’) without affecting the system’s ability to perform morphogenesis.

Self-assembly

Self-assembly arises from the interactions of robots in almost all of the individual behavioural states. After deciding to form a new structure, the seed robot immediately enters the Recruitment state and begins to broadcast both long range ‘recruitment’ messages and short range beacon signals on the docking sides at which it is required to recruit other modules. The detection of ‘recruitment’ messages causes robots in the Wandering state to transition into the LocateBeacon state (figure 4.3, transition 1) and subsequently, if the beacon signal is detected, into the Alignment state (figure 4.3, transition 2).

If the LocateBeacon or Alignment states are unsuccessful, robots will transition back to the Wandering state (figure 4.3, transitions 3-4). If both are successful, the robots will transition to Docking (figure 4.3, transition 5) and in order to establish a reliable connection, will execute a simple docking protocol. At the same time as it locks its own docking element, the docking robot instructs the recruiting module to do the same by sending a ‘docking-ready’ message using its IR LEDs and transitioning to the Locking state (figure 4.3, transition 6). After a short delay (in the order of a few seconds) allowing time for the connection to be established, the docking robot transmits a ‘docking-complete’ message through the wired channel. Finally, the recruiting module responds by transferring information regarding the shape of the current structure.

The controller also incorporates two precautions against the interference that may occur as multiple robots attempt to align with the same recruiting robot. As soon

as a robot enters the Alignment state it broadcasts an ‘in-range’ message to notify the recruiting robot that it is within range of the beacon signal. When a recruiting robot receives an ‘in-range’ message from an aligning robot, it stops broadcasting its availability, therefore helping to prevent other robots from entering the LocateBeacon state. Furthermore, whilst aligning, robots broadcast ‘expelling’ messages to notify other nearby robots of their presence. When an aligning robot receives an ‘expelling’ message from another module, it switches back to the Wandering state, helping to prevent two robots from aligning with the same recruiting module. Both of these actions reduce the chances of multiple robots interfering with the docking process.

When a new robot joins a structure it enters the InStructure state (figure 4.3, transition 7). When a module is require to recruit, it transitions to Recruitment (figure 4.3, transition 8), and when finished transitions back to the InStructure state (figure 4.3, transition 9). The decision as to which robots enter the Recruitment state at which point in time, and upon which sides they signal for other robots to join, is determined by a ‘recruitment strategy’. Two different recruitment strategies have been developed. The first of which is described in [108] and referred to as ‘single entry recruitment’. In this strategy, only one robot may occupy the Recruitment state at any moment in time, and may only recruit upon one docking side. The second strategy [110], is referred to as ‘multiple entries recruitment’. This strategy removes the restrictions upon the number of robots that may recruit simultaneously, and the number of sides at which they may do so. Both strategies are described in more detail below.

Single Entry Recruitment In the single entry recruitment strategy all robots hold the same library of structural shapes. When a new robot joins a structure it is told which shape is being constructed and the current number of robots in the structure. Messages are then propagated throughout the structure to allow every robot to update its own internal counter of the number of robots present. For every shape, there is a predetermined order in which the modules must be added. Since every module in the structure knows when it joined and how many robots are currently in the structure, individuals may work out when it is their turn to recruit simply by referencing these values with their knowledge of the shape being constructed.

Multiple Entries Recruitment With the multiple entries recruitment strategy, robots are not required to store a record of all of the possible structural configurations. Instead, the seed robot holds the entire shape (encoded using the method described in section 4.1.1) and the relevant sub-branches are passed on to new modules as and when they join the structure. When a new robot joins, depending upon which side it docks, it is passed the corresponding sub-branch. If the joining robot is not required to recruit any further modules, it is simply passed an empty string. Any robot that receives a non-empty string immediately enters the Recruitment state. By examining the string they receive, robots in the Recruitment state can determine which docking sides they are required to recruit at. The process continues until every module has finished recruiting, at which point, all robots will occupy the InStructure state. Note

that with the multiple entries strategy, since the order in which robots may be added is less restricted, modules do not need to know the current size of the structure in order to determine whether or not they should enter the Recruitment state. Any robot that is required to recruit new modules will attempt to do so immediately.

Self-disassembly

Regardless of the recruitment strategy that is employed, the Self-disassembly behaviour remains the same. Self-disassembly can be initiated by any module. When and why a robot would choose to initiate the disassembly process is considered to be beyond the scope of this work, but examples may include the robots encountering a gap or crevasse that is too small or narrow for a larger structure to navigate through. Self-disassembly begins with a single module sending disassembly messages to all of its neighbours. As soon as an individual decides, or is notified, that it is time to disassemble it moves into the Disassembly state (figure 4.3, transition 10) and unlocks its docking element. Robots that are only connected to one other module then remove themselves from the structure by reversing away and re-entering the Wandering state (figure 4.3, transition 11). Robots that are connected to more than one other module, propagate the disassembly message and wait until all but one of their neighbours has disconnected itself. This process repeats until all of the robots have left the structure. It is assumed that disassembly begins only once the 3D structure has reconfigured itself into a 2D planar structure, but how the structure arrives in this arrangement is considered to be outside of the scope of this work.

4.2 Failure Mode and Effect Analysis

Failure Mode and Effect Analysis (FMEA) is a well established procedure for analysing the safety and reliability of a product or process [114]. The technique is widely used within the manufacturing industry, but in a departure from its normal usage, was also utilised by Winfield and Nembrini to analyse the reliability of a robotic swarm [193].

To perform FMEA, the analyst first derives a list of specific ‘failure modes’—the things that can go wrong—for every component in the system under study. They then attempt to identify all of the effects that these failures may have on the system, in the process building up a general overview of the system’s reliability. Because of this progression from specific failures to general effects, FMEA may be described as an *inductive* approach to system analysis. Further details of the procedure can be found in [34] and [114].

In order to perform FMEA on the morphogenesis controller introduced in section 4.1, a similar approach to that of Winfield and Nembrini [193] is followed. The individual components of the system are considered to be the system level behaviours introduced in section 4.1. The failure modes, meanwhile, correspond to the complete or partial failure of an individual robot.

Hazard	Description	Hazard	Description
H_1	Motor failure	H_M	Motor failure
H_2	Communications failure	H_R	IR remote receiver failure
H_3	Avoidance sensor(s) failure	H_S	IR sensor failure
H_4	Beacon sensor failure	H_L	IR LED failure
H_5	Control systems failure	H_T	Total systems failure
H_6	Total systems failure	H_D	Docking element failure
		H_W	Wired communication failure

(a)

(b)

Table 4.2: Hazards investigated by [193] (a) and those analysed in this study (b)

4.2.1 Failure Modes

Failure modes or ‘hazards’, as they are otherwise referred to, may be classed as either *internal* or *external* (environmental). Internal hazards are those which originate from a single robot. The failure of an individual component or subsystem, for example, would be considered an internal hazard. External hazards originate from the *interactions* of multiple robots and their environment, sensor interference is a good example of such a hazard. In this study, like that of Winfield and Nembrini, only internal hazards are considered. The six hazards identified by Winfield and Nembrini (reproduced in table 4.2a) form the basis of this work.

The majority of the hazards considered in Winfield and Nembrini’s study correspond to the failure of an independent subsystem. Although the Symbricator modules share some of the same functionalities as the robots in [193], subsystems that are considered independent in Winfield and Nembrini’s study, may not necessarily be considered independent here. For example, as highlighted in section 3.1.1, the avoidance sensing subsystem of the Symbricator robots shares components with the beacon sensing subsystem, a fault within an infrared sensor—a component which both subsystems share—could result in the failure of both obstacle avoidance and beacon sensing. Specifically, in this study, the subsystems responsible for communications, avoidance sensing and beacon sensing are all inter-linked. As shown in table 4.2b, because of this interdependence, hazards corresponding to the low level components that make up each subsystem, rather than the subsystems themselves, are studied. As in [193], motor and total systems failures are also considered. Additionally, in this study, two new hazards are examined which relate to the ability of the robots to dock with one another and communicate through a wired connection.

4.2.2 Effects

In section 4.1, three system level behaviours were identified: Exploration, Self-assembly and Self-disassembly. When an individual robot suffers a failure, the effect that it has

on the system may differ depending upon which of the system level behaviours the robot was contributing to when the failure occurred. During self-assembly, the choice of recruitment strategy may also influence the effects of any failures. In order to compare the two recruitment strategies, during analysis, the recruitment strategy decision processes are extracted from the Self-assembly behaviour and considered as separate system level behaviours. The methods through which recruiting robots attract new modules to a structure and the interactions between the robots within and outside of the structure are still considered part of the Self-assembly behaviour. The difference is that the mechanisms through which robots decide whether or not to enter the Recruitment state, and upon which docking sides to recruit, are now considered separately under the headings of the two recruitment strategies.

Whilst performing FMEA, four different effects were identified, two of which are described as serious and two of which are described as non-serious. All four are listed below, with uppercase lettering used to denote the serious effects:

- e₁ - reduction in the number of capable robots*
- e₂ - delay in the formation of a structure*
- E₁ - stall in the formation of a structure*
- E₂ - stall in the disassembly of a structure*

This section now considers each of the hazards introduced in section 4.2.1 in turn and outlines how they may affect the system, depending upon the system level behaviour which the afflicted robot is contributing to. Failures are assumed to be permanent and to occur in only a single robot at any one moment in time.

H_M - Motor

In this chapter, a motor failure is defined as any failure of the motor subsystem which causes the affected robot to remain stationary. This definition does not cover partial failures in which a robot can move, but does so in an uncoordinated fashion. Whilst a robot that has suffered a motor failure will remain stationary, it will not be prevented from communicating with other modules.

The effect that a motor failure has on the system depends largely upon where the robot was located when the failure occurred. The failure of a robot that is located outside of communication range of the structure, thus contributing only to Exploration, will have very little effect on the system. Since there is no explicit communication between robots during Exploration, a failed robot will not interfere significantly with others. The only effect will be a reduction in the number of capable robots, *e₁*. Provided that the number of active robots is still greater than the number of robots required to create the desired shape, the task of forming a structure may still be completed.

Robots which are contributing to the Self-assembly behaviour, but are not yet part of the structure, will be located closer to the site of assembly. The effect of a motor

failure in this instance will be far more severe. A robot that stops moving whilst in the Alignment state, for example, will not only fail to join the structure itself, but by transmitting ‘expelling’ messages and physically blocking the path of others, may actively prevent further robots from doing so. If robots are prevented from joining the structure at one or more recruitment sites, the formation of the structure will be permanently stalled, effect E_1 .

Analogously, during Self-disassembly, a robot that has suffered a motor failure may act as a physical obstacle, preventing itself and others from leaving the structure and resulting in a stall in disassembly, or effect E_2 .

H_R - IR Remote Receiver

An IR remote receiver failure is defined as a failure of the communications subsystem which prevents a robot from receiving wirelessly transmitted messages through one or more of its IR remote receivers. This definition does not account for failures in which a device may still receive partial messages or corrupt information.

IR receivers are not used by either of the recruitment strategies, nor are they used during Self-disassembly. However, the failure of this device may affect the system if it occurs in a robot that is contributing to Exploration or Self-assembly.

During Exploration, the failure of one or more of a robot’s remote receivers will reduce the chances of the robot detecting any messages broadcast by recruiting modules. Without being able to detect ‘recruiting’ messages, a robot will be unable to transition to the Alignment state, and therefore will not be able to join a structure. Although the affected individual will be unable to help form a structure itself, it will not prevent other robots from doing so. It will simply remain in the Wandering state. The only effect will be e_1 , a reduction in the number of active robots.

During Self-assembly, IR receivers are used by aligning robots to detect ‘expelling’ messages and by recruiting robots to detect both ‘in-range’ and ‘docking-ready’ messages. The inability of a robot to detect these messages is unlikely to prevent the formation of a structure, since they are not required for a robot to successfully dock, but it may slow the assembly process. Whilst aligning, robots which are less likely to detect ‘expelling’ messages are more likely to cause interference near the recruitment site. Meanwhile, a recruiting robot that cannot detect ‘in-range’ messages will continue to attract robots to the recruitment site, regardless of whether there are already other robots within range of its beacon. An increase in the number of robots near the recruitment site, will naturally lead to an increase in interference. The effect, in both scenarios, is a delay in the formation of the structure, e_2 . A recruiting robot that does not receive a ‘docking-ready’ message will not be able to respond by locking its docking element, however, provided the docking robot is able to lock its own element, this alone is not sufficient to prevent the completion of the docking process. The inability of a recruiting robot to receive a ‘docking-ready’ message will therefore have no effect on the system.

H_S - IR Sensor

IR sensors can fail in a variety of ways. In this study, no single type of failure is considered, but rather the focus is on the general effects of a breakdown in the correlation between true and sensed values. IR sensors are used for both proximity detection and beacon detection. The failure of one or more IR sensors will reduce the robots' ability to perform both of these functions. Since the recruitment strategies do not require these functions, they will not be affected, the other behaviours, however, will be.

Without the ability to effectively perform proximity detection, a robot is likely to collide with obstacles or its neighbours. This is not a problem if the robot is contributing only to Exploration, in which case the effect will simply be a reduction in the number of capable robots, effect e_1 . If a robot is contributing to Self-assembly or Self-disassembly, however, the inability to perform proximity detection may lead to the robot colliding with the structure. By physically blocking the path of other robots, in the worst case, an IR sensor failure may cause both the assembly and disassembly processes to stall, effects E_1 and E_2 .

Without the ability to perform beacon detection, robots will be unable to align with the recruiting module, this may simply lead to the robot returning to the Wandering state, but in the worst case, it may instead lead to the robot colliding with the structure, again resulting in effect E_1 .

 H_L - IR LED

An IR LED failure is defined as the failure of one or more infrared LEDs, that will result in the affected LEDs remaining permanently off. This definition does not account for failures in which an LED remains permanently on, or flashes uncontrollably. An IR LED failure will limit the ability of a robot to broadcast messages or transmit beacon signals. IR LEDs are not used by either of the recruitment strategies, nor are they used during Self-disassembly or Exploration. The Self-assembly behaviour, on the other hand, relies heavily on modules with functioning LEDs to provide communication.

As part of self-assembly, recruiting modules use their LEDs to send both long range recruitment messages and short range beacon signals. Without this ability it will be impossible for robots to locate and align with recruiting modules. Leading to a stall in the formation of the structure, or effect E_1 . LEDs are also used by aligning robots to transmit 'expelling' messages and by docking robots to transmit 'docking-ready' messages. The effect of a robot that is unable to transmit 'expelling' messages will be an increase in interference and a delay in the formation of the structure, e_2 . However, the severity of this effect is far outweighed by the inability of a recruiting module to attract new robots. The effect of a robot that is unable to transmit 'docking-ready' messages is not significant, since this alone is not sufficient to prevent the completion of the docking process.

H_T - Total System

A total systems failure—which may be considered equivalent to a robot running out of energy—will result in the shutdown of all of a robot’s subsystems. A robot that suffers a total systems failure will be completely immobilised and be unable to communicate with other modules.

During Exploration, the effects of a total systems failure are negligible. Any robot contributing to Exploration that suffers a total systems failure will simply act as a static obstacle to other members of the group. The only effect will be a reduction in the number of active robots, e_1 . Whilst participating in the recruitment strategy decision process, or contributing to the Self-assembly or Self-disassembly behaviours, the consequences of a total system failure will be more severe.

For similar reasons to when a motor failure occurs, a robot that suffers a total systems failure whilst blocking the path to or from a structure, may cause both the assembly and disassembly of the structure to stall, effects E_1 and E_2 . Note, however, that unlike the effects of H_M , there will be no interference due to the robot sending ‘expelling’ messages. A total systems failure will also prevent recruiting modules from attracting new robots and disassembling modules from unlocking their docking mechanisms, further precursors of effects E_1 and E_2 .

Robots which fail whilst executing either recruitment strategy will be unable to transition from the InStructure state to the Recruitment state when required. Furthermore, in the case of single entry recruitment, modules will be unable to propagate messages when a new robot joins the structure. Again leading to effect E_1 .

H_D - Docking

A docking element failure will force the affected mechanism to remain in the state that it currently occupies. An unlocked element will be unable to lock itself and a locked element will be unable to unlock. This definition does not cover failures that occur whilst an element is transitioning between the locked and unlocked states.

Docking elements do not change state during Exploration, or during the execution of either recruitment strategy, a failure in these behaviours, therefore, will have no effect.

Assuming that all robots start with their docking elements unlocked and remembering that the docking protocol introduced in section 4.1 ensures that only one locked docking element is required for a reliable connection to be established, if a single element fails in the unlocked state, Self-assembly will not be affected.

If the docking element of an existing member of the structure fails-locked, however, it will prevent the affected robot (and any connected neighbours) from un-docking. Therefore leading to effect E_2 , a stall in the disassembly of the structure.

H_W - Wired Communication

The failure of wired communications will prevent the affected robot from sending or receiving any information through the wired channel. This definition does not account

<i>Behaviour</i>	H_M	H_R	H_S	H_L	H_T	H_D	H_W
Exploration	e_1	e_1	e_1	-	e_1	-	-
Self-assembly	E_1	e_2	E_1	E_1	E_1	-	E_1
Self-disassembly	E_2	-	E_2	-	E_2	E_2	E_2
<i>Recruitment strategy</i>							
Single entry	-	-	-	-	E_1	-	E_1
Multiple entries	-	-	-	-	E_1	-	-

Table 4.3: A summary of the FMEA study. Each row corresponds to one of the three system level behaviours and two recruitment strategies. Each column corresponds to one of the failure modes introduced in section 4.2.1. Each cell shows the effect that a particular failure mode may have when it occurs during the execution of a particular behaviour or recruitment strategy

for failures in which a robot may receive partial messages or corrupt information.

An affected robot will be unable to send or receive messages using wired communications and will be unable to act as a node in a wired network. Not only will the failure have a local effect on the failed robot, but any two robots in the structure whose sole path of communication passes through the failed module will also be prevented from exchanging messages.

Wired communications are not utilised during Exploration, nor are they used during the decision process of the multiple entries strategy, in these scenarios, such a failure will have no effect.

During Self-assembly, the failure of wired communications in either a recruiting robot or a newly docked robot, will prevent the new module from receiving information about the shape of the structure being constructed. Therefore, the assembly of the structure will stall with effect E_1 .

During Self-disassembly, if a module is unable to propagate messages when it is time to disassemble, sections of the structure may be unaware that it is time to do so. This will result in a stall in the disassembly of the structure, or effect E_2 .

Similarly, in order to determine when a robot should enter the Recruitment state, the single entry recruitment strategy requires that messages are propagated throughout the structure when a new module joins. If this is not possible then the formation of the structure will stall with effect E_1 .

4.2.3 Analysis

The effects that all seven of the hazards may have on the performance of the system are summarised in table 4.3. The three system level behaviours and the two recruitment strategies are considered separately.

As shown in table 4.3, of the 21 possible combinations of hazard and behaviour (ignoring for now the recruitment strategies), all but 6 will have some effect on the

system. Of the 15 which have an effect, 10 of the effects are described as serious. The most detrimental of the seven hazards is a total systems failure, which has a negative effect on the system in every behaviour, as well as during the execution of both recruitment strategies. The failure of an infrared remote receiver is the least harmful, in no scenario will this hazard have a serious effect on the system. LED failures and docking element failures are the next least critical, each only leading to a single serious effect, in a single behaviour.

In comparing the two recruitment strategies, the only difference that arises is the effect of a wired communications failure. Systems employing both strategies are reliant upon wired communications to ensure that newly docked robots receive information about the shape of the structure being constructed. After a robot has joined the structure, however, only the single entry strategy uses wired communications for determining which robots should enter the Recruitment state. With the single entry strategy, the failure any robot that prevents messages being propagated to the required modules will cause the formation of the structure to stall. Meanwhile, with the multiple entries strategy, only a failure during the initial exchange of information will stall assembly.

4.3 Fault Tree Analysis

In the previous section, the effect that a particular failure may have on a system were inferred using the inductive Failure Mode and Effect Analysis (FMEA) technique. In contrast to FMEA, Fault Tree Analysis (FTA) [187] is described as a *deductive* approach to system analysis. Beginning with a general system failure, the analyst attempts to identify all of the potential causes of the failure event and the logical sequence of secondary events leading up to it, these together make up the fault tree.

The tree for a particular system failure is constructed by repeatedly breaking the event down into more specific intermediate events. The relationships between events at different levels in the tree are specified using Boolean logic, where the outputs of functions at lower levels serve as the inputs to those at higher levels.

Following the construction of a fault tree, and relying on the fact that the underlying structure of the tree may be described in terms of Boolean algebra, the analyst may perform both qualitative and quantitative analysis (again differentiating FTA from FMEA, in which only qualitative analysis is possible). Qualitatively, the analyst is able to identify the various combinations of component failures which may cause a system failure, as well as their relative importance. Quantitatively, if the failure rates of the components are known, it is possible to calculate the probability of a system failure.

In the remainder of this section, a fault tree is constructed for the morphogenesis controller introduced in section 4.1. The tree is then used to analyse some of the potential causes of a stall in the formation of a multi-robot structure. Fault trees are represented graphically using an extension of standard logic gate notation. In this work, only a subset of the available notation is used, with symbols introduced as and when they are required, for a full list the reader should consult [187]. Following construction,

the fault tree is analysed qualitatively. The combinations of component failures which may lead to a stall in the assembly process are identified and their relative importance is discussed. Comments are made upon the levels of fault tolerance in the current system and areas in which improvements could be made are identified.

4.3.1 Construction

Before beginning construction it must be stated more precisely what is meant by a ‘stall in the formation of a structure’. A stall in the formation of a structure can be said to have occurred if at least one robot has been unable to recruit the modules that it was required to (within some acceptable time limit). This means that one or more robots have either become stuck in, or failed to transfer to, the Recruitment state. When investigating the multiple entries recruitment strategy, only the scenario in which a robot becomes stuck in the Recruitment state needs to be considered, since the time spent in the InStructure state (before a robot has completed its recruiting duties) is negligible. With the single entry strategy, both cases must be considered. Here, only the first scenario is examined, that a robot has become stuck in the Recruitment state.

The only condition that will cause a robot to leave the Recruitment state is the receipt of confirmation (through wired communications) that another robot has docked with it. As shown in figure 4.4, where $R1$ is the robot that is awaiting confirmation, this statement constitutes the topmost event of the fault tree. In fault tree notation, events are represented using rectangular boxes.

The reasons why $R1$ has not received confirmation are now considered. There are two options, either no robot has docked with $R1$, or a robot ($R2$) has docked with $R1$ but for some other reason $R1$ has not received confirmation. These options are separated using an OR gate, representing the fact that the output of this gate will occur if at least one of the inputs occurs. In this context, two robots are considered to have ‘docked’ once they have aligned and moved into the docking position, communication between the pair and the engagement of either docking mechanism are not considered necessary for docking to be said to have taken place. The scenario in which no robot has docked with $R1$ is not pursued further, but may immediately be recognised to contain situations such as other robots blocking the path to $R1$, or $R1$ suffering an LED or total systems failure which prevents it from attracting other robots. Using fault tree notation, this ‘undeveloped event’ is presented in a diamond shaped box. As discussed further in section 4.4, this event highlights a potential weakness with the FTA approach, specifically, the difficulty in has in modelling complex, stochastic processes.

Focusing on the case in which another robot has docked with $R1$. In this scenario, there are two reasons why $R1$ would not receive confirmation of docking from $R2$. The first is that $R2$ has not sent confirmation, the second is that $R2$ has sent confirmation but $R1$ has not received it. Considering the first leads to the first basic cause of a stall in the assembly of a structure, that $R2$ has suffered a wired communications failure. Such ‘basic fault events’ are represented by circles in fault tree notation. The second case, that $R2$ has sent confirmation but $R1$ has not received it, requires further development.

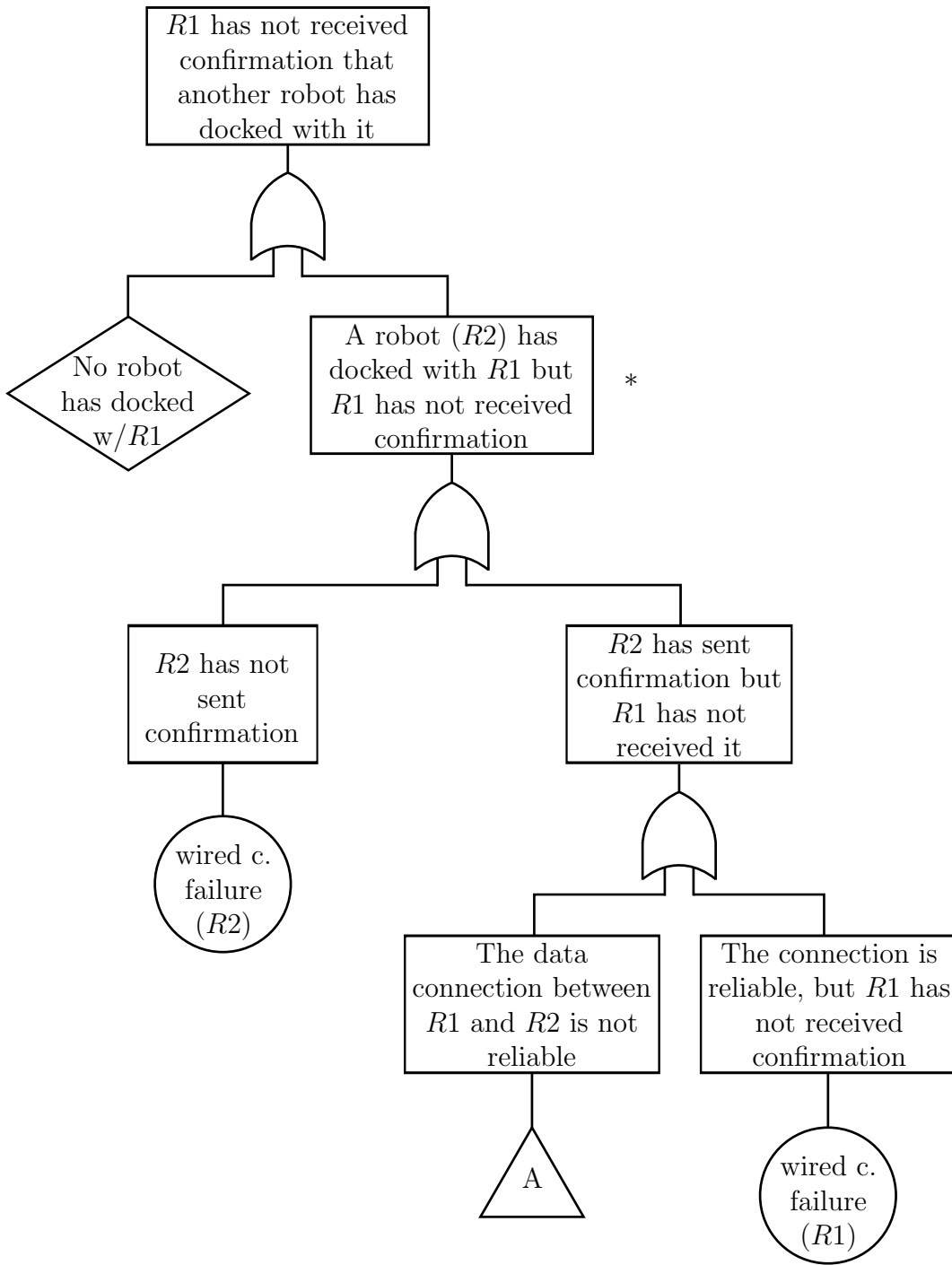


Figure 4.4: A fault tree examining the causes of a stall in the formation of a robotic structure. The rectangular boxes represent developed events, the diamond shaped box represents an undeveloped event, the logic gates describe the relationships between events, the circles represent basic fault events and the triangle signifies the transition to a different fault tree diagram

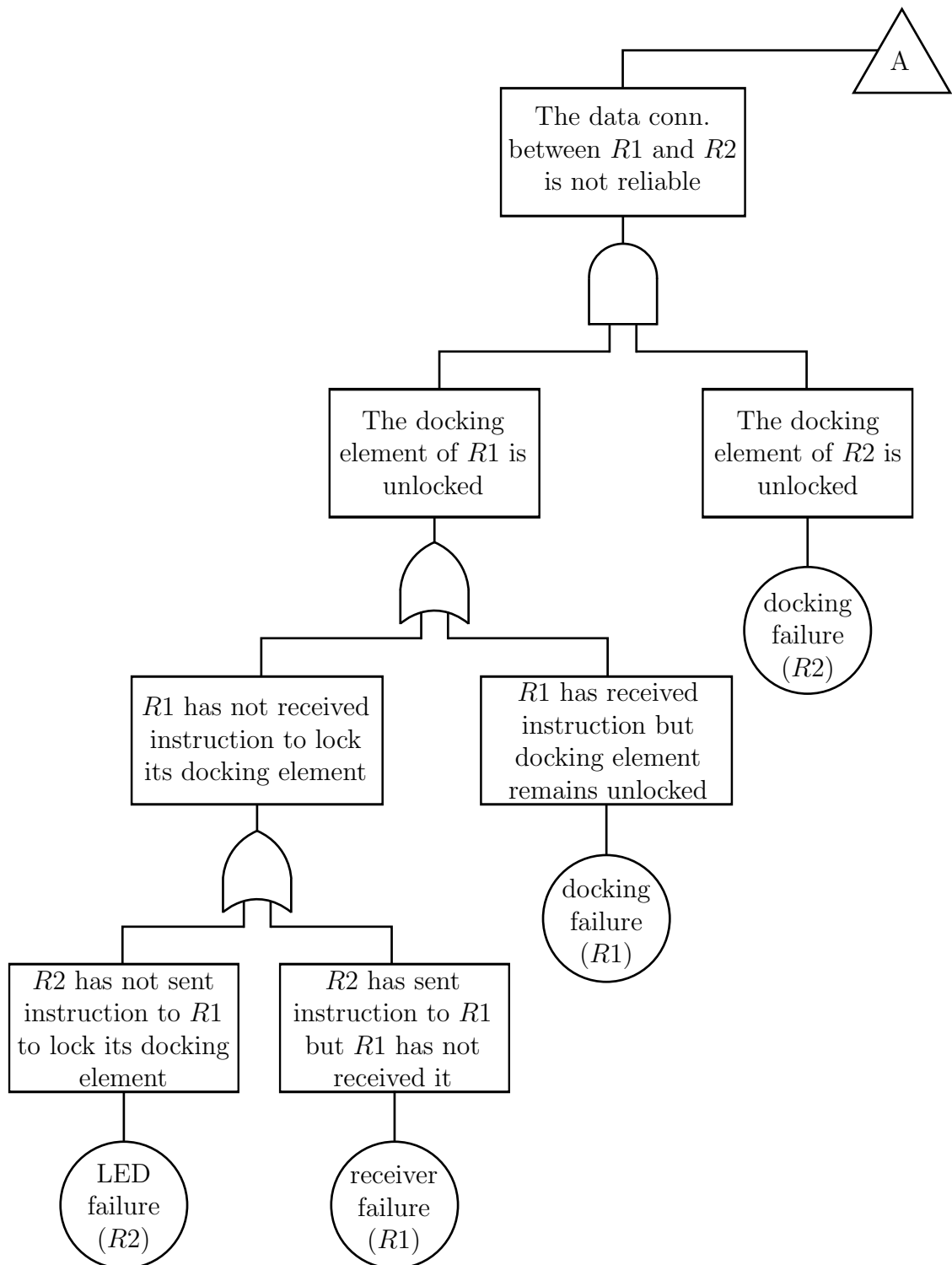


Figure 4.5: A continuation of the fault tree from figure 4.4. The rectangular boxes represent events, the logic gates describe the relationships between events, the circles represent basic fault events and the triangle signifies the transition from a different fault tree diagram

There are two possibilities why, although $R2$ has sent confirmation, $R1$ has not received it. The first is that the data connection between the two robots is unreliable. The second is that although the data connection is reliable, $R1$ has still not received confirmation. In this second scenario, given that confirmation was successfully sent by $R2$, the problem must lie with the wired communications of robot $R1$. The case of an unreliable data connection is developed further in figure 4.5, triangular ‘transfer symbols’ are used to connect the two diagrams.

Using an AND gate, the output of which will occur only if both inputs do, the first part of figure 4.5 shows that for an unreliable connection to be formed between $R1$ and $R2$, the docking elements of both robots must be unlocked. Remembering the docking protocol introduced in section 4.1. Since $R2$ is not reliant on any communication with $R1$ in order to lock its docking element, the only reason that it would remain unlocked is a failure of the device itself. $R1$ on the other hand will only lock its element once it receives instruction from $R2$ through an IR receiver, or when a wired connection is already established. Since it is assumed that a reliable wired connection has not been established, there are two reasons why $R1$ ’s docking element remains unlocked, either $R1$ has not received instruction to lock it, or it has received instruction, but in attempting to lock it has been prevented by the presence of a docking element failure.

The scenario in which $R1$ has not received instruction to close its docking element leads to the final two basic causes of a stall in assembly, and to the end of the fault tree’s construction. For $R2$ to instruct $R1$ to close its docking element both the LED of $R2$ used to send the message and the IR receiver of $R1$ used to receive it must be functioning. If either of these components have failed $R1$ will not receive the instruction to close its docking element. This is shown by the final level of the fault tree in figure 4.5. The construction of the fault tree is complete and the focus now turns to analysis.

4.3.2 Analysis

The first step of analysing a fault tree involves representing it in terms of Boolean equations. Ignoring the undeveloped event, figures 4.4 and 4.5 can be minimally represented as figure 4.6. The basic fault events WX , DX , LX and RX correspond respectively to wired communications, docking element, LED, and receiver failures of robot ‘ X ’. The equivalent Boolean equations of the tree are shown alongside it in figure 4.6.

With the tree converted into Boolean equations, the next task is to obtain the ‘minimal cut sets’. In fault tree analysis, a cut set is any combination of basic events which, if they all occur, will cause the top event to occur. A cut set is described as *minimal* if removing any of the elements from the set produces a collection which is no longer a cut set. The minimal cut sets can be obtained simply by reducing the equations of the fault tree until only basic fault events are left. By applying the laws of Boolean algebra, the equations in figure 4.6 are easily converted into the form shown in equation 4.1.

$$T = (D1 \bullet D2) + (D2 \bullet L2) + (D2 \bullet R1) + W1 + W2 \quad (4.1)$$

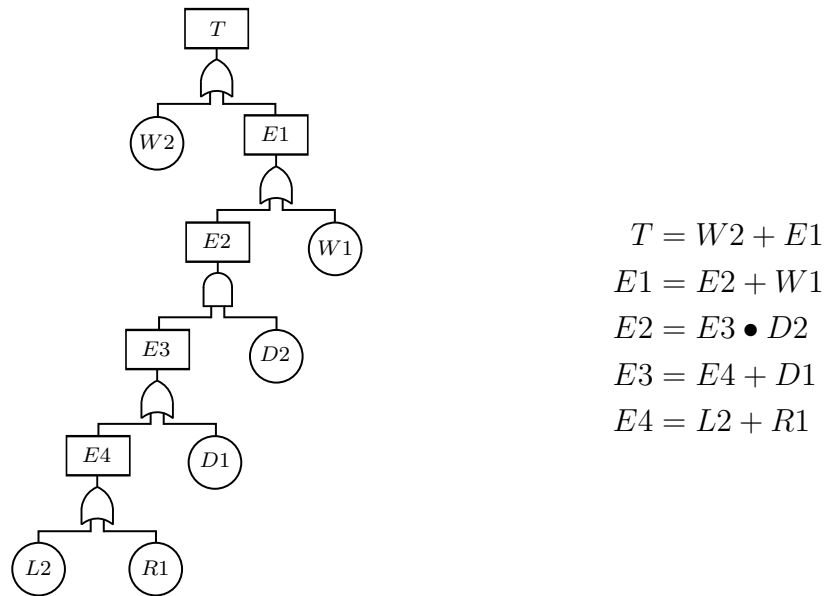


Figure 4.6: A minimal version of figures 4.4 and 4.5, with equivalent Boolean equations. The rectangular box labelled T represents the top event, the rectangular boxes labelled $E1$ - $E4$ correspond to intermediate events, and the circles correspond to basic fault events. For simplicity, the events which are not separated by a logic gate in figures 4.4 and 4.5 are combined

The minimal cut sets then, are simply: $\{D1, D2\}$, $\{D2, L2\}$, $\{D2, R1\}$, $\{W1\}$ and $\{W2\}$. Where, for example, $\{D1, D2\}$ represents the scenario in which a recruiting module and a newly docked robot both suffer docking element failures.

For this example, deriving the minimal cut sets was trivial. However, it should be remembered that this is only a subsection of a much larger fault tree. To determine the minimal cut sets of more complex fault trees requires greater effort. Fortunately, software exists to aid the analyst during both the construction and analysis of fault trees. One example of FTA software which was used to help develop the fault trees in this work is OpenFTA [5].

Even though only a subsection of the overall tree has been considered, it is still possible to make some interesting qualitative observations. From the point of view of reliability, single component minimal cut sets are undesirable, simply because they imply the failure of a single component may cause the entire system to fail. Even without reducing figure 4.6 to the minimal cut sets of its Boolean equations, it is clear from the lack of AND gates that the system will contain single component cut sets. In this case, those cut sets are $\{W1\}$ and $\{W2\}$, corresponding to the scenarios in which either a recruiting robot or a newly docked robot has suffered a wired communications failure. Figure 4.6 does contain one AND gate, and its presence gives rise to the three remaining two-component cut sets, all of which, it may be noted, contain $D2$ —the failure of a newly docked robot’s docking element. The design of the hardware and the

simple docking protocol, in this situation, ensure that the single failure of one of the involved components does not lead to a stall in the assembly process.

As a general rule, it is desirable to have AND gates positioned as high up the fault tree as possible. With this in mind it is possible to suggest some improvements to the system. A weak point in the fault tree of figure 4.4 may be identified at the '*'. Although recruiting modules are provided with information about the presence of newly docked robots in the form of both wired and wirelessly transmitted messages, as well as the values returned by their IR proximity sensors, they are critically dependent upon the wired channel to determine their behaviour. Recall that the only time at which a robot will leave the Recruitment state is when it receives confirmation, through wired communications, that another robot has docked with it. It is suggested, therefore, that rather than relying solely on wired communications to dictate their behaviour, recruiting modules should make better use of the data from their IR receivers and sensors. If robots react appropriately to the information (or lack thereof) that all three channels provide, the chances of a robot becoming stuck in the Recruitment state, following the docking of another robot, may be reduced. Detecting the presence of another robot without receiving confirmation through wired communications may signal to the recruiting robot that there is something wrong. Therefore providing the robot with another way out of the Recruiting state by, for example, initiating the processes of repairing or replacing either itself or the docking robot. The fault tree of a controller adapted in this manner, no longer being singularly reliant on wired communications, would be expected to contain more AND gates, located further up the tree. The system, therefore, would exhibit greater tolerance to the failure of individual components.

4.4 Discussion

This section begins by discussing the suitability of FMEA and FTA to the analysis of self-reconfigurable modular robotic systems. The relative merits and weaknesses of the two approaches are outlined and the usefulness of their application during the design of fault tolerant robotic systems is considered. The main findings of the analysis performed in sections 4.2 and 4.3 are then discussed. Finally, the fault tolerance of the morphogenesis controller introduced in section 4.1 is examined and the areas in which improvements could be made are identified.

Whilst FMEA is an inductive approach to failure analysis, FTA is deductive. When performing FMEA, the analyst begins with a list of specific hazards and attempts to identify what effect they may have on the system. Conversely, with FTA, the analyst begins with a general effect and gradually traces it back to its root causes, the hazards. In practical terms, this means that by employing FTA, the analyst is forced to focus on identifying increasingly specific causes. Resulting in a very complete understanding of the chain of events that lead from a component failure to a system failure.

One advantage of the precise knowledge that the FTA procedure provides, is that it may help the analyst to identify specific weak points in the system. This was shown in

section 4.3 where the over reliance of recruiting robots on wired communications was identified as a weakness. What FMEA lacks in terms of its depth of detail, it makes up for with the breadth of information that it provides. In section 4.2, the FMEA procedure covered four different effects of seven different single component failures. In a similar amount of space, with significantly greater effort, the FTA procedure covered only a subsection of a single effect, and revealed only one single component cause.

What the FTA procedure does reveal with ease, which may be less obvious when performing FMEA, are the different *combinations* of component failures which may lead to a system failure. Furthermore, although in general, FMEA is better suited to the task of exhaustively listing possible hazards, the two approaches both have the potential to identify hazards overlooked by the other.

Another potential advantage of FTA, which was not exploited in this work, is the ability to perform quantitative analysis. One of the reasons why quantitative analysis was not carried out is because it is reliant upon failure rate data, which was not available. In this instance, another more pressing reason, however, indicative of a much larger problem with the application of FTA to self-reconfigurable modular robotic systems, is that the systems themselves may be too complex to model in sufficient detail. When constructing the fault tree in section 4.3, assumptions were made about the state of the system, in particular with regard to the interactions between robots and their environment. The tree presented in section 4.3 examined the system in a relatively static state, considering the interactions between only two robots. The scenario in which the formation of a structure had stalled because no robot was able to dock with the recruiting module was ignored as an ‘undeveloped event’. There are numerous reasons why a robot may be unable to dock with a structure, but due to the dynamic nature of these systems as well as the large numbers of agents and the locational dependency of their interactions, the FTA approach is far less well suited to describing them. Examining these more dynamic components would require increasingly limiting assumptions to be placed on the system. Allowing for a more complete understanding, but at the expense of detail in the model. Without modelling the interactions between robots more accurately, it is not possible to produce accurate quantitative results.

The desire to model the reliability of complex fault tolerant systems is widespread. The difficulties that arise in accounting for the high levels of redundancy, fault recovery mechanisms and sequentially dependent failures that such systems possess are not unique to robotic systems. Efforts to help solve these problems have led to the development of Dynamic Fault Trees (DFT) [42]. In constructing DFTs, additional ‘dynamic’ gates are used to account for the extra complexity. Analysing DFTs then essentially involves combining the standard fault tree approach with Markov Chain models. Thus, the simplicity of the standard approach is augmented by the flexibility of Markov models [42].

The analysis carried out in section 4.2 revealed that 15 of the 21 possible combinations of hazard and behaviour will have some effect on the system, 10 of which will be serious. This is in contrast to the results of the study by Winfield and Nembrini [193], in which there were observed to be only 6 serious effects out of a possible 30.

While performing FMEA, a pessimistic view was taken in assuming that all components of a single subsystem will fail simultaneously. Whilst in some cases, the effect of a failure will depend upon how many and which components of a subsystem fail, in others, the failure of a single component may be equally as detrimental as the failure of multiple components. An obstacle avoidance controller, for example, that aggregates the values from several IR sensors in order to determine the speed of two motors, may be severely hampered even if only a single sensor fails. The poor outlook, therefore, cannot be attributed solely to pessimism. Neither is this evidence sufficient to reject the long held belief that swarm robotic systems inherently provide fault tolerance. It is reminded that the Symbicator platform is not a pure swarm robotic system, but rather a self-reconfigurable modular robotic system. The modular aspect brings with it far greater dependence between robots. If the system is examined closer, it is observed that it is during the interactions with modules already in the structure where the effects of a failure are most severe. Note that in table 4.3 there are no serious effects when a failure occurs during Exploration, the only behaviour that involves robots operating exclusively as individuals.

To endow a modular robotic system with levels of fault tolerance similar to those found in swarm robotic systems, greater plasticity in the conformation of the system is required. Whereas in swarm robotic systems, faulty robots may simply be ‘left behind’, in modular robotic systems, more explicit methods of replacing or repairing modules are required. The Symbicator platform was designed with this in mind, supporting the ability of robots to recharge or directly power their connected neighbours. As a result, in the pursuit to design fault tolerant modular robotic systems, it is encouraging that the fact that what appears to be the most detrimental hazard, a total systems failure, is also the easiest to repair (if assumed to be caused by a robot running out of energy).

In this analysis, the effects of transient and external hazards were not considered. If transient hazards were considered, or if it were possible to repair or replace failed modules, the multiple entries strategy could be identified as possessing a significant advantage. With the multiple entries strategy, even though the assembly of a structure may stall at one point, it may continue at others. A system utilising the multiple entries strategy is therefore able to recruit and repair simultaneously. In considering external hazards, such as the interference observed near recruitment sites, the single entries recruitment strategy possesses the advantage. However, with the multiple entries strategy, the increased level of interference is offset by the fact that multiple robots may recruit simultaneously. The time saved by the multiple entry strategy directly translates to a saving in energy, which may be critical to the long-term survival of the system.

4.5 Summary and Future Work

In this chapter, the FMEA and FTA procedures were applied to a morphogenesis controller designed for the Symbicator robots. The analysis revealed several scenarios in which even a single failure may have serious consequences, the worst case being the

occurrence of a total systems failure. FMEA and FTA were each identified to have their own advantages. Whereas FMEA provides a good general overview of the system under analysis, FTA reveals specific details about the chain of events that may lead to a system failure. A combined approach, as demonstrated here, is advocated during the design of fault tolerant autonomous robotic systems. In future work, further analysis of the both the current controller and any improved systems may be performed. For which purpose, DFTs and Markov chain models have been identified as promising techniques.

CHAPTER 5

Energy Foraging and Anomaly Detection

In chapter 4, a total system failure—which may be considered equivalent to a robot running out of energy—was identified as the hazard that would be most detrimental to a group of robots performing autonomous morphogenesis. The failure of infrared (IR) sensors, resulting in an increase in the number of collisions between robots, was also identified as a hazard. In this chapter, an energy foraging and energy sharing strategy is presented which aims to reduce some of the negative effects that may appear as a result of robots running out of energy. Furthermore, an anomaly detection algorithm is presented which targets the failure of IR sensors and aims to support the energy foraging strategy by reducing the chances of robots colliding with one another.

The anomaly detection algorithm, known as the ‘modified’ Dendritic Cell Algorithm (mDCA), was originally developed by Mokhtar et al. [120], as an adaptation to the Dendritic Cell Algorithm (DCA) of Greensmith et al. [59]. In this chapter, improvements are made to the mDCA and its parameters are optimised using multi-objective optimisation. The performance of the system is compared to a state of the art Support Vector Machine (SVM) based approach and the relative merits of each system are discussed.

The mDCA and SVM based approaches are then combined with the energy foraging strategy. The ability of these systems to survive for long periods of time, despite the presence of faulty individuals, is investigated within a simulated setting. The systems are shown to perform significantly better than a system which is not capable of detecting anomalies, and on a comparable level to a system which utilises ideal anomaly detection.

This chapter continues in section 5.1 by introducing the original mDCA approach and describing the improvements that were made in this thesis. In section 5.2, the energy foraging and energy sharing strategy is described and comparisons are made with similar existing systems. In section 5.3, the parameters of the mDCA and SVM based approaches are optimised and the two systems are compared. In section 5.4, the long term survival of robotic collectives which utilise the combined energy foraging and anomaly detection systems are investigated. Finally, in section 5.5, the main findings of this chapter are summarised and some potential areas of future work are suggested.

5.1 The ‘modified’ Dendritic Cell Algorithm

The DCA [59] and mDCA [120] are inspired by the function of the vertebrate immune system. Specifically, they take inspiration from a novel immunological theory known as the *danger model* [113]. The danger model proposes that the ability of the immune system to detect and mount a response against invading pathogens may be related to the presence of special ‘danger’ signals that are released when cells are damaged. This contrasts with the previously dominant *self-non-self model* in which the physical matching between immune cell receptors and antigens was considered to be the primary initiator of an immune response.

In the new model, ‘danger’ signals, and other molecular patterns indicative of the state of the environment, are detected and processed by *dendritic cells*. How dendritic cells respond to the signals that they receive determines whether or not a threat is perceived and whether the rest of the immune system takes action. It is this decision process of the dendritic cells that the DCA and mDCA aim to replicate. Whether or not anomalous behaviour is flagged by the DCA or mDCA is somewhat equivalent to whether or not dendritic cells initiate an immune response against an invading pathogen.

The mDCA was first introduced by Mokhtar et al. [120], who applied it to the task of detecting faults in the IR sensors of simulated e-puck [122] robots. In unpublished work, the approach was subsequently used to monitor the actuation and power subsystems of simulated Symbricator modules. Later, as a step towards creating a ‘self-sufficient’ modular robotic system, the task of monitoring robots’ power subsystems was investigated by Humza et al. [78].

In this chapter, improvements are made to both the mDCA itself and the supporting experimental framework. Whereas early investigations used simple robot models, in this chapter, more realistic models from the adapted version of the Stage simulator (introduced in section 3.1.2) are used. The updated simulator includes more realistic models of the robots, as well as more authentic sensor data—derived from measurements of real IR components—and more realistic failure data.

5.1.1 Overview

The mDCA is an anomaly detection algorithm for detecting errors within timeseries data. When applied to the task of detecting anomalies in the infrared sensors of Symbricator robots, the algorithm is run separately on each of the robots’ eight sensors. For each sensor, the output is a value between 0 and 1, which indicates the likelihood that the corresponding sensor is faulty. A value of 1 signifying that the data received from the sensor is anomalous and a value of 0 signifying normality.

For each sensor that is being monitored (i), the mDCA first calculates the output of the weighted sum shown in equation 5.1. The terms $A_i(t)$, $B_i(t)$ and $C_i(t)$ are abstract analogies of the signals which are detected by dendritic cells. In the danger model, a variety of signals are said to provide dendritic cells with information about the current state of their environment, and may be representative of both normal and abnormal

states. The dendritic cells aggregate this information and then decide whether or not to contribute to an immune response. In the mDCA, $A_i(t)$, $B_i(t)$ and $C_i(t)$ are features that have been extracted from a sliding window of recent sensor data and provide information about the current state of sensor i . In analogy to the behaviour of the dendritic cells, the features are aggregated using equation 5.1 and the output used to help determine whether or not data is flagged as anomalous or normal.

$$\sigma_i(t) = \frac{w_a A_i(t) + w_b B_i(t) + w_c C_i(t)}{w_a + w_b + w_c} \quad (5.1)$$

As an example, the feature $A_i(t)$ may represent the difference between sensor i and sensor $i - 1$ at time t . Because neighbouring sensors can be expected to return similar values, a large value of $A_i(t)$ may be considered anomalous in this case.

It is noted that equation 5.1 could easily be extended to include more than three features. Increasing the number of features would potentially increase the classification accuracy. However, it would also increase the computational complexity and the size of the parameter search space, which in turn would increase the time taken to optimise the system. For this reason, it was decided that only three features would be used.

Each of the three features has an associated weight: w_a , w_b and w_c , the values of which determine how much influence each feature has over the output of equation 5.1. At each timestep, the output of the weighted sum from equation 5.1 is passed through the thresholding function shown in equation 5.2.

$$\phi_i(t) = \begin{cases} 0 & \sigma_i(t) < \tau \\ 1 & \sigma_i(t) \geq \tau \end{cases} \quad (5.2)$$

Equations 5.1 and 5.2 together define a linear classifier in 3-dimensional space. These two equations alone could be used to classify incoming data as anomalous or normal, however, to add some level of confidence to the classification, the value of $\phi_i(t)$ is averaged over ω previous timesteps using equation 5.3.

$$c_i(t) = \frac{1}{\omega} \sum_{k=t-\omega}^t \phi_i(k) \quad (5.3)$$

The final output of the algorithm for sensor i at time t is then: $0 \leq c_i(t) \leq 1$. A value of 1 indicates that it is likely that the corresponding sensor is faulty and a value of 0 indicates that it is unlikely. From this brief description of the mDCA it can be seen that there are five main parameters which define the behaviour of the system, and therefore determine the classification accuracy. These five parameters are shown in table 5.1 and correspond to: the weights of the three features (w_a , w_b and w_c), the threshold value (τ) and the window over which to average the classification output (ω).

w_a	w_b	w_c	τ	ω
-------	-------	-------	--------	----------

Table 5.1: The five main parameters of the original mDCA

5.1.2 Improvements

This section describes the improvements that were made to the mDCA. The improvements that were made to the sensor data and supporting experimental framework are described separately in section 5.2.

In previous work, the following three features, or variants thereof, were extracted from the sensor data to represent the terms $A_i(t)$, $B_i(t)$ and $C_i(t)$ in equation 5.1:

1. The difference between the value of the sensor being monitored and the value of its nearest neighbour, at time t
2. The difference between the value of the sensor being monitored at time t and the average of the values returned by that sensor over a short recent period
3. The result of applying a binary threshold to the output of feature 2, returning ‘1’ for values greater than the threshold and ‘0’ otherwise

There are some issues with the above features which limit the capabilities of the mDCA, particularly when more realistic sensor data is used. Feature 1 was chosen to represent abnormal behaviour, based upon the assumption that neighbouring sensors should return similar value. This assumption is generally true, however, the calculation can not identify whether the difference between the value of one sensor and its neighbour is the result of a failure within the sensor itself or in its neighbour. In the original algorithm, this led to a large number of false positives, with sensors believing themselves to be faulty when there was actually a fault present in a neighbouring sensor.

Another issue with the previous feature extraction process is highlighted by feature 3. This feature only ever returns 0 or 1, effectively turning what would be a 3D classifier into a pair of 2D classifiers that are switched between depending upon the value of this feature. It is suggested that a better classifier may be constructed if three real-valued features are used, creating a true linear classifier in 3D space.

A more general issue with all three of the above features is their complexity, or lack thereof. All of the features use very simple arithmetic operations and are calculated over short periods of time. The obvious advantage of their simplicity is that they guarantee that the algorithm will be able to run on a resource limited system, as the Symbricator robots are considered to be. However, if the features are too simple, they may not provide enough information to accurately classify the status of the sensors.

In this chapter, some of the restrictions of the previous feature extraction process are removed and improvements are made. To address the lack of complexity in the existing features, a number of new features have been designed. The complete set of features used in this chapter is displayed in table 5.2. Attention is drawn to the feature named

$distN$, which is similar to feature 1 from above, but takes the difference between the target sensor and the average of both of its neighbours, one of which will be positioned on the same side of the robot (the immediate neighbour) and one of which will be positioned on an adjacent side. The design of the $distN$ feature was motivated by the fact that, comparing the target sensor's value with both of its neighbours (rather than just one), should help to differentiate between the scenario in which the sensor itself is faulty and the scenario in which one of its neighbours is faulty.

It is important to highlight that, at any one moment in time, the mDCA may only use three of the features from table 5.2, retaining the classifiers restriction to three-dimensional space. Which features the algorithm uses is determined by three new parameters: f_a , f_b and f_c , each of which corresponds to one of the ten features listed in table 5.2. To allow the algorithm to be tuned further, the size of the sliding window over which the features are calculated is also made adjustable. The size of the window corresponding to each of the three features f_a , f_b and f_c is specified by the value of three new parameters: t_a , t_b and t_c . The complete set of parameters used by the new version of the mDCA is displayed in table 5.3. In section 5.3, multi-objective optimisation is used to find the optimal values of these parameters and in section 5.4, the long-term survival of the optimised system is analysed.

5.2 Energy Foraging Controller

The energy foraging controller described in this section was designed to reduce the chances of an individual robot experiencing a total systems failure. The controller ensures that total systems failures are avoided by allowing modules to recharge themselves at power sockets and share energy with other modules. The controller is based upon the morphogenesis algorithm introduced in section 4.1, but rather than a single module initiating the construction of a predetermined shape, robots form structures only as and when then need to. Robots may dock with power sockets in order to recharge themselves, or form ad-hoc structures with other modules in order to donate or harvest energy. To further improve the reliability of the system, the controller also incorporates a recovery mechanism which alters the behaviour of the robots if a failure is detected in one of their infrared sensors.

5.2.1 Robots

The controller was developed for the adapted version of the Stage simulator introduced in section 3.1.2. The simulated robots are functionally similar to the Symbricator Backbone modules. Each robot possesses eight infrared sensors, four two-way communication devices, four LEDs, eight light detectors and four active docking elements. The IR sensors are used for detecting the presence of obstacles; the IR communication devices are used for both long and short range (line-of-sight) communication; the LEDs and light detectors are used to accurately identify the position of objects; and the docking

Name	Description	Formula
$stdDev$ (σ)	The standard deviation of recent data	$\sigma = \sqrt{\frac{1}{(N-1)} \sum (x_i - \bar{x})^2}$
$distAve$	The difference between the current sensor value and the average of recent data	$distAve = x - \mu $
$bDistAve$	The difference between the current sensor value and the average value of recent data, with a binary threshold (τ) applied. In all of the experiments reported here τ took the value 0.	$bDistAve = \begin{cases} 0 & distAve \leq \tau \\ 1 & distAve > \tau \end{cases}$
$distN$	The average difference between the current sensor and its two neighbours	$distN = \mu_a - (\mu_b + \mu_c)/2 $
$distNN$	The average difference between the current sensor and its nearest neighbour	$distNN = \mu_a - \mu_b $
$skew$	The ‘skewness’ of recent sensor data	$skew = \frac{1}{N} \sum ((x_i - \mu)/\sigma)^3$
$mean$ (μ)	The mean value of recent sensor data	$\mu = \frac{1}{N} \sum x_i$
$distStdDev$	The difference between the standard deviation of the sensor being monitored and that of its nearest neighbour	$distStdDev = \sigma_a - \sigma_b$
$range$	The range of the recent sensor data	$range = \max(X) - \min(X)$
$pairDist$	The cumulative difference between consecutive pairs of recent data	$pairDist = \frac{1}{N} \sum x_{i+1} - x_i $

Table 5.2: The names, descriptions and formulae of the ten features that may be used by the mDCA. The formula column uses the following terminology: subscript letters a , b and c indicate an association with one of three neighbouring sensors; N is the number of data points in the window across which the feature is extracted; X is the set of all data points in the current window; x are individual data points; σ is shorthand for the $stdDev$ feature; and μ is shorthand for the $mean$ feature

f_a	f_b	f_c	t_a	t_b	t_c	w_a	w_b	w_c	τ	ω
-------	-------	-------	-------	-------	-------	-------	-------	-------	--------	----------

Table 5.3: The 11 parameters of the new version of the mDCA

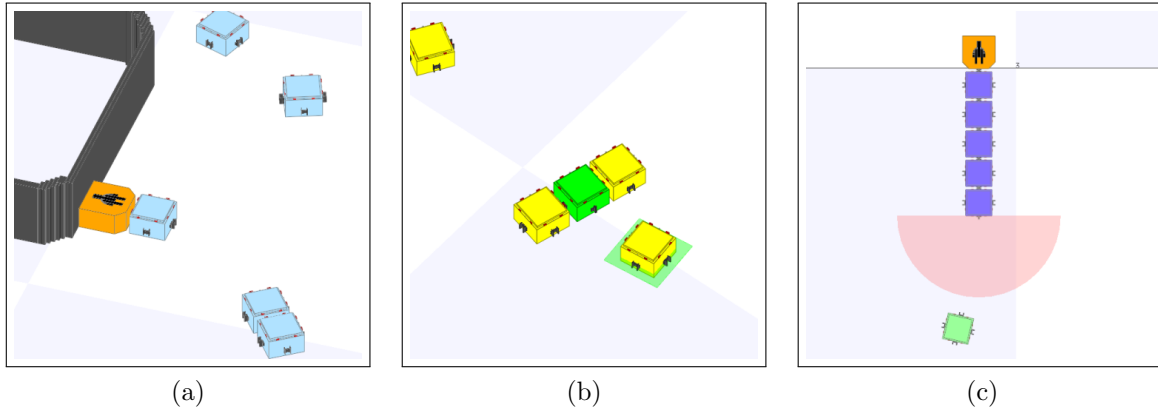


Figure 5.1: Screenshots of the new models within the Stage simulator

elements are used to secure the connection between two docked robots.

As shown in figure 5.1, in this thesis, the Stage models were extended to allow modules to share energy with one another and to recharge themselves at power sockets. Figure 5.1a shows the newly created model of a power socket with one module recharging itself. Figure 5.1b shows three modules docked and sharing energy whilst a fourth robot approaches. Figure 5.1c shows extra functionality which demonstrates how, when docked to a power socket, robots may recruit other modules and form chains that allow multiple robots to be recharged simultaneously.

5.2.2 Sensor Data

A direct benefit of using the adapted version of the Stage simulator are the improvements that are made to the realism of the sensor data. In previous work [120], sensor values were returned as a distance in millimetres from the robot to any obstacle in the way of that sensor. In reality, the proximity sensors do not return a measure of distance, but a raw value which scales non-linearly with the actual distance between the robot and an obstacle. By using the Stage simulator, not only is it possible to access these raw values but, based upon data collected by [108], it is also possible to add realistic noise to the sensors and create more realistic fault models for injecting anomalies.

There are numerous possible sources of anomaly in the data stream of an infrared proximity sensor. Examples include: interference from other robots or docking stations, the varying reflective properties of different materials, or a fault within a sensor itself. In all of these situations, anomalies may manifest themselves transiently or permanently. In this work, only transient anomalies are considered. The proposed source of these anomalies are faults within the sensors, however, the fault model employed is sufficiently

Fault	Description
Stuck-at-value	the value returned is always the same
Sensor noise	the value returned is a random amount away from the ideal
Sensor bias	the value returned is a fixed amount away from the ideal

Table 5.4: The three different types of fault simulated in this chapter

general that the resulting anomalies may share characteristics with other situations. For this reason, the mDCA may be thought of as a general anomaly detection algorithm, and is not just limited to the detection of faults. The fault model used in this chapter defines three different types of fault. These faults are listed in table 5.4.

In past applications of the mDCA, only two of these three faults were investigated, those being: stuck-at-value and sensor noise. The model of sensor noise used in previous experiments simply returned a random value, irrespective of the actual value of the sensor. In this work, a different model is used in which values are returned from a normal distribution, centred around the ideal value. It is important to note that sensor noise is present in most real world systems and is often not considered a fault. However, when the noise exceeds that of a device’s normal operating characteristics, the device may be declared faulty. Most systems are able to tolerate a small amount of noise, but performance may degrade when the level of noise is increased.

In the past, the mDCA was shown to work well in the extreme case (equivalent to using a normal distribution with a large standard deviation), however, as the standard deviation is reduced and the level of noise in the system decreases, the task of detecting anomalies becomes harder. Eventually there will come a point at which the amount of noise is so low that, from the perspective of the mDCA, normal and abnormal data is indistinguishable. Whether or not this location is acceptable is determined by how tolerant the behavioural controller is to the amount of noise present. If the robots are able to continue operating without deviating significantly from their normal behaviour, then it does not matter if abnormal sensor data is incorrectly labelled as normal. In fact, if there is a cost associated with responding to the detection of an anomaly, then it may be beneficial for the system to ignore anomalies which do not significantly affect behaviour. The question of whether the cost of ignoring a fault out-weighs the cost of recovery is returned to section 5.4.

5.2.3 Controller

A finite state machine for the energy foraging and energy sharing controller is shown in figure 5.2. The LocateBeacon, Align, Dock and Lock states (and their associated transitions) are similar to their namesakes from figure 4.3. However, whereas the states in figure 4.3 allow modules to align and dock with other robots, in figure 5.2, these states also allow robots to dock with power sockets. The controller described in this section further differs from that introduced in section 4.1 with regards to the circumstances

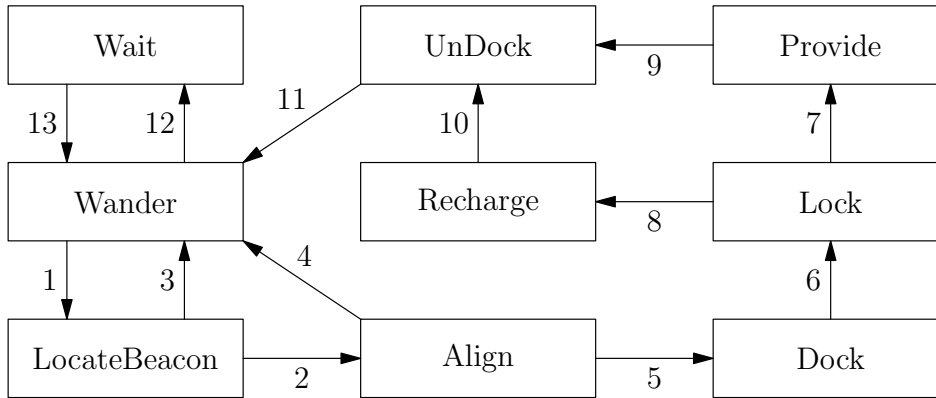


Figure 5.2: A finite state machine for the energy foraging controller

Symbol	Value	Name
τ_w	0.20	Wait threshold
τ_a	0.25	Avoid power socket threshold
τ_p	0.40	Approach power socket threshold
τ_r	0.50	Approach distressed robot threshold

Table 5.5: The four different thresholds of the energy sharing strategy

which cause robots to initiate the alignment and docking processes, and the behaviour of modules that have successfully done so.

The default behaviour of the robots is wandering. In the Wander state, robots simply move around the environment, avoiding obstacles and other robots. Which states the robots subsequently transition into is determined by their current energy level and whether or not any of their infrared sensors are considered to be faulty. The robots’ ‘energy sharing’ and ‘recovery’ strategies determine which transitions are made.

Energy Sharing Strategy

In the experiments reported in section 5.4, robots have a limited supply of energy, and so to survive, must be able to recharge themselves. Recharging may occur either at power sockets, such as the one shown in figure 5.1a, or through docking with other modules, as shown by figure 5.1b. A simplified energy model is used in which energy is transferred between docked robots at a constant rate and is consumed by individual robots at a rate which is linearly proportional to the speed at which the robots are travelling. For the collective as a whole to survive, robots must decide when to be altruistic and when to be selfish, that is, when to recharge themselves and when to provide energy for others. The energy sharing strategy described in this section utilises a threshold-based approach. The four threshold values used here are listed in table 5.5 (as a proportion of the robots’ maximum energy capacity).

In normal operation, robots perform random wandering with obstacle avoidance. If a robot has sufficient energy ($e > \tau_r$, where e is the robot's current energy level) and it detects the presence of another module that requires energy, the robot will transition through the aligning and docking stages (figure 5.2, transitions 1, 2, 5 and 6) and into the Provide state (figure 5.2, transition 7). Whilst in the Provide state, robots will transfer up to 50% of their available energy to the module that they are docked with.

If a robot with a moderate need for energy ($e < \tau_p$) detects a power socket, it will approach, align and dock with the socket before entering the Recharge state (figure 5.2, transition 8). Robots remain in this state until their energy level reaches 100%.

During testing, it was observed that, with a large number of robots and a comparatively small number of docking stations, congestion around power sockets often led to robots crashing and blocking the socket from future use. To help reduce this problem, as shown in figure 5.1c, functionality was added to allow multiple robots to recharge at the same power socket. After a robot has entered the Recharge state, it begins to signal on its rear side as if it were a power socket, attracting more modules to the socket which may then dock with the signalling module and simultaneously recharge. To further reduce the congestion problem, robots which have insufficient energy to reach a socket ($e < \tau_a$), or have no need to recharge ($e > \tau_p$), will actively avoid sockets.

When a robot in the Provide state has donated more than 50% of its available energy to the module that it is docked with, or it is no longer required to supply energy, it will transition to the UnDock state (figure 5.2, transition 9). Likewise, when a robot is fully recharged, or its power source is no longer able to provide for it, it will also transition to the UnDock state (figure 5.2, transition 10). Any robot in the UnDock state which is only docked with one other module (or socket) will then return to the Wander state (figure 5.2, transition 11).

If a wandering robot is desperately in need of energy ($e < \tau_w$), it will stop moving and conserve what energy it has by transitioning to the Wait state (figure 5.2, transition 12). Whilst in the Wait state, modules signal to nearby robots that assistance is required. Up to four robots (one on each side) may simultaneously dock with a waiting robot and provide energy for it. Once the robot is recharged, it notifies any providing modules, stops signalling and transitions back to the Wander state (figure 5.2, transition 13).

Recovery Strategy

When an anomaly is detected in a sensor, regardless of whether it is the result of a fault in that sensor, interference from other robots, or some other unknown factor, a response is required that will prevent the anomaly from adversely affecting the behaviour of the robot. An anomaly which causes the value of a sensor to remain fixed at a low value, for example, may prevent the robot from detecting an obstacle and lead to a collision which, as highlighted in chapter 4, could have wide reaching consequences for the system.

It is important to realise that, without prior knowledge of the system and without placing assumptions on the expected output, it is difficult to design a recovery strategy that can retrieve all of the information that is lost due to an anomaly. Recovery

strategies may be envisaged that are able to bring back some of the lost information, for example, by removing noise or offsetting bias in the data, but for the most part, the best that a recovery strategy can achieve is to minimise the effects of the failure.

In this work, when an anomaly is detected in a sensor, the behavioural controller responds by assuming that the anomalous sensor is returning the same value as its immediate neighbour, based upon the assumption that these values should be similar. When a robot is only performing obstacle avoidance or wandering, this strategy is generally sufficient. However, during more precise tasks such as aligning and docking, more accurate sensor values are required. Furthermore, since these tasks take place in critical areas of the arena, as suggested in section 4.2, the system as a whole may be far less tolerant to robots crashing or failing when executing these behaviours.

To address this issue, robots which have detected an anomaly in the data stream of one of their front two sensors—those which are most influential over the robots' behaviour—are prevented from entering the `LocateBeacon` state. Furthermore, if a robot is already in the `LocateBeacon` or `Align` state when it detects the presence of a failure, it will immediately transition back to the `Wander` state (figure 5.2, transitions 3 and 4) and remain in this state for as long as the failure is present.

There is one special case that must also be considered, that being, what happens when both a sensor and its neighbour are thought to be returning anomalous data. In this situation, the strategy utilised here is to assign both sensors the same small value. This strategy is based upon the assumption that a small value will not be able to drastically affect the behaviour of the robot, but will be enough to prevent the robot from colliding with obstacles on the side at which the failed sensors are located.

A couple of deficiencies can be identified in this recovery strategy which it is important to highlight. Firstly, the assumption that the values of neighbouring sensors are similar will not always hold, and when it doesn't, the consequences can be as bad, or even worse, than if no response was attempted at all. Secondly, because the recovery strategy prevents robots from recharging both themselves and other modules whilst an anomaly is present, the entire system, and not just the erroneous individual, is more vulnerable.

5.2.4 Comparisons

In chapter 2, several different types of fault detection system were reviewed. Those most relevant to this chapter are the other approaches which were inspired by the function of the vertebrate immune system. For example, both [11] and [120] (from which this work follows) were inspired by the danger model theory introduced in section 5.1. Meanwhile, [17] takes inspiration from the more traditional self-non-self model. Whilst in [142], the authors describe an anomaly detection system inspired by the interactions between immune cell receptors and the molecules with which they bind. The approach of [142] was also later used by [104] as part of a collective self-detection scheme. Finally, in [177], the authors describe an approach to fault detection which does not require a-priori knowledge of what should be considered 'normal' or 'abnormal' behaviour.

The task of ‘swarm foraging’ is well studied within the field of swarm robotics. Inspired by the collective behaviour of social insects such as ants and bees, the task of swarm foraging involves a group of agents searching for an object or resource within their environment. Once located, the resource is often transferred to some central location, or ‘nest’. The controller described in this chapter was designed to perform the related sub-task of ‘energy foraging’. In this task, the resource being collected is energy, and to succeed, agents must balance the energetic cost of foraging with the reward received for successfully retrieving objects, that is to say, they must maximise the net energy income [111].

Many authors have described approaches to swarm foraging in which energy efficiency is of central importance [15, 86, 101, 111]. In [101], Labella et al. describe a foraging controller in which the global division of labour is automatically adjusted based upon whether individual robots are successful or unsuccessful at locating and retrieving ‘prey’ from their environment. In Labella et al.’s strategy, every individual has a certain probability of switching between resting and searching behaviours, in robots which are successful at foraging, this probability is increased, whilst in unsuccessful robots, this value is decreased. The result is a system in which the number of foragers and resters automatically adapts according to the availability of prey. In similar work, Campo and Dorigo describe a decision algorithm in which robots alter their behaviour according to the amount of energy that they estimate is obtainable by the group as a whole [15]. Meanwhile, in [111], Liu et al. describe a system in which the number of foraging and resting robots is balanced based on internal cues (whether or not the individual is successful), environmental cues (how many other robots are currently foraging) and social cues (whether or not other individuals are successful).

The three systems mentioned above all utilise an abstract definition of ‘energy’ which does not fully appreciate the real world consequences of an individual robot running out of power. In this chapter, a more literal interpretation of ‘energy foraging’ is followed in which the resource being collected directly influences the amount of energy stored within the individual. Furthermore, whereas in the systems described above, the resource is retrieved from the environment and returned to a nest, in this work, there is no central store and energy is always fully distributed about the system. In this respect, the approach is similar to that of Kernbach and Kernbach, in which a large-scale robot swarm demonstrates the ability to survive over long periods by allowing the individual robots to recharge themselves at docking stations [86]. In a similar manner to the strategy described in section 5.2, each robot monitors its own energy level and decides when, and for how long, it should recharge itself. However, unlike the strategy described in this chapter, as well as monitoring their own energy levels, the robots estimate the collective energy of the swarm. Using this information, the robots can infer when the swarm is highly active and avoid bottlenecks at the docking stations by reducing the amount of time they spend recharging. Conversely, if the swarm is not active, robots may recharge for longer without the risk of causing interference.

The biggest difference between the controller described in section 5.2 and the approaches reviewed above is the fact that, in the system introduced in this chapter,

robots are able to transfer energy between one another. Whilst a lot of previous research has focused on the task of energy foraging, far fewer approaches have focused on energy sharing. In [134], Ngo and Schioler describe the design of a new mobile robotic platform in which modules can carry multiple batteries and may exchange them with other robots depending up their energetic needs. This method of physically transferring batteries differs from the energy sharing strategy described in section 5.2—and other similar energy sharing systems [79, 96, 116]—in which each robot has a single battery, but is able to donate or receive energy from other modules through recharging. In [96] and [116], Kubo and Melhuish discuss the idea of ‘robot tropholaxis’ and describe a simple threshold based strategy through which robots may request energy from, or donate energy to, other modules. Meanwhile, in [79], Ismail and Timmis describe an algorithm inspired by the immunological process of granuloma formation, in which robots surround, isolate and recharge other modules which are in need of energy.

Whilst similarities are observed between the controller introduced in this chapter and all of the above systems, what distinguishes this work is the manner in which it combines fault detection, energy foraging and energy sharing into a single system.

5.3 Multi-objective Optimisation

Systems with multiple objectives are inherently more difficult to optimise than those with only one, especially if some of the objectives are conflicting. The canonical example of a multi-objective optimisation problem is that faced by the automotive industry. Car manufacturers must (amongst other things) maximise performance and strength (safety), whilst minimising fuel consumption and weight. Clearly a trade-off is required. Rather than produce a single optimal solution, multi-objective optimisation allows the designer to produce a set of *Pareto* optimal solutions, from which they may select the solution which best fits their current requirements. With regards to the problem of anomaly detection, there are several objectives that the designer may wish to optimise. For example, maximising the true positive rate, minimising the false positive rate, maximising the speed of detection and, of particular importance to resource limited systems such as autonomous robots, minimising the computational cost.

In this section, the parameters of the mDCA are optimised in order to find the best individual with which to investigate the task of long-term survival. An alternative approach to classification which utilises Support Vector Machines (SVMs) is also introduced and optimised. After optimising the parameters of the two types of system, the approaches are compared and analysed in terms of their classification accuracy.

5.3.1 Support Vector Machines

Support Vector Machines are a form of supervised learning model, which are commonly used in classification tasks. SVM models are built during a training phase in which, given a set of labelled data, a training algorithm constructs a hyperplane that maximises

the distance between itself and the training data of two different classes. After training, the SVM model may be used to classify new data points according to which side of the plane they fall on. As well as performing linear classification, by first transforming the feature space, SVMs may also be used to classify data that is not linearly separable.

More formally, given a set of n labelled points (\mathbf{x}_i, y_i) , where \mathbf{x}_i is a real vector and y_i is a class label of 1 or -1 . The aim is to find a hyperplane which separates the set of points for which $y_i = 1$ and the set of points for which $y_i = -1$. This hyperplane can be written as the set of points \mathbf{x} which satisfies:

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \quad (5.4)$$

If the two classes are linearly separable, several such hyperplanes may exist. The objective then is to find the *optimal* hyperplane. That is, the hyperplane for which the margin between itself and the closest data points of the two classes is maximised. This margin may be defined by two further hyperplanes:

$$\mathbf{w} \cdot \mathbf{x} - b = 1 \quad (5.5)$$

$$\mathbf{w} \cdot \mathbf{x} - b = -1 \quad (5.6)$$

The set of points which lie on these planes are known as the *support vectors*. Finding the optimal hyperplane is an optimisation problem which involves finding the set of support vectors and the hyperplanes for which the separating margin is maximised.

In this chapter, following a feature extraction process similar to that used by the mDCA, SVMs are used to classify IR sensor data as anomalous or normal.

5.3.2 Experimental Setup

A common experimental setup was used during all of the optimisation experiments. In the following sub-sections, the parameters that were optimised, the objective functions that were evaluated and the simulation environment in which the experiments were conducted are all introduced in turn.

The parameters were optimised using the state-of-the-art evolutionary multi-objective optimisation algorithm, NSGA-II (Non-dominated Sorting Genetic Algorithm II) [39]. The implementation of NSGA-II from the Kanpur Genetic Algorithms Laboratory¹ was used throughout all of the experiments. Beyond the description of the genome (provided in the following sub-section) the Kanpur implementation of NSGA-II requires at least six other parameters to be specified. In the experiments conducted in this chapter, these parameters were selected based upon informal testing and were not themselves optimised. The parameters required, and the values they took in all of the optimisation experiments, are displayed in table 5.6.

¹<http://www.iitk.ac.in/kangal/codes.shtml>

Parameter	Value
Population size	48
Number of generations size	100
Crossover probability	0.6
Mutation probability	0.3
Crossover distribution index	20
Mutation distribution index	20

Table 5.6: The NSGA-II parameters used during the optimisation experiments

						mDCA-I				
mDCA-II										
SVM										
f_a	f_b	f_c	t_a	t_b	t_c	w_a	w_b	w_c	τ	ω

Table 5.7: The parameters optimised in the mDCA-I, mDCA-II and SVM experiments

Parameters

Two different approaches to optimising the mDCA parameters were investigated, each focusing on the optimisation of two different sets of parameters. In the first approach, referred to as mDCA-I, only a subset of the 11 mDCA parameters from table 5.3 were optimised. Specifically, in resemblance to the original mDCA experiments performed in [120], the parameters representing the three features (f_a , f_b and f_c) were fixed to: *distNN*, *bDistAve* and *distAve*. Furthermore, the window over which *distNN* was calculated (t_a) was restricted to a single timestep. The windows over which the other two features were calculated (t_b and t_c), as well as the weights of the three features (w_a , w_b and w_c), the threshold value (τ) and the averaging window (ω) were all optimised. In the second approach, referred to as mDCA-II, all 11 of the parameters from table 5.3 were optimised. The full set of parameters optimised in the mDCA-I and mDCA-II experiments are shown by top two rows of table 5.7.

The mDCA is a linear classifier. The algorithm defines a flat plane in three-dimensional space which attempts to separate two, potentially overlapping, classes as best as possible. In the mDCA-I experiments, the parameters which define the feature space are partially evolved, alongside the parameters which define the separating plane. With the mDCA-II, both the feature space and the separating plane are fully evolved.

SVMs may similarly be used as three-dimensional classifiers, however, as well as being able to separate classes linearly, SVMs may also classify data that is not linearly separable. In this chapter, an SVM-based approach is used to classify IR sensor data as normal or abnormal. The method through which features are extracted from the data is analogous to the process employed by the mDCA, but how these features are classified differs. The third row of table 5.7 shows the parameters of the SVM feature extraction process which are optimised in this section. All of the SVM experiments

Parameter	f_a	f_b	f_c	t_a	t_b	t_c	w_a	w_b	w_c	τ	ω
Range	(0, 9)			(1, 500)			(-100, 100)			(-100, 100)	(1, 500)

Table 5.8: The ranges over which the evolvable parameters are restricted

reported in this chapter use the LIBSVM library².

The same set of features from table 5.2 were used in all of the mDCA and SVM experiments. The parameters optimised in each of the three experiments are shown in table 5.7 and the ranges over which they were restricted to are shown in table 5.8.

Objective Functions

NSGA-II may be used to optimise any number of objectives. For classification tasks, there are several potential objectives that may be used to assess performance. Examples include maximising the true positive rate (TPR) and classification accuracy (ACC), or minimising the false positive rate (FPR) and speed of classification. Table 5.9 shows how these, and other, statistical measures of performance may be calculated. In the experiments described in this chapter, two objectives were chosen, one of which relates to the TPR and one of which relates to the FPR. As shown in table 5.9, in order to calculate the TPR and FPR, the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) must be known.

The output of the mDCA is a real value between zero and one, to provide a classification, this value must be thresholded—as is necessary when implementing the recovery strategy described in section 5.2. Whilst it would be possible to calculate the TPR and FPR from the mDCA by first thresholding the output, in order to make use of the extra information that the mDCA provides, a slightly different approach is employed. Rather than measure the TPR and FPR directly, in this set of experiments, objectives were chosen which calculate the real-valued difference between what would be the ideal output and the actual output of the algorithm, first for when anomalous data is being read (*obj1*) and second for when normal data is being read (*obj2*). The assumption here being that solutions in which these distances are small will be more robust, and hence better classifiers, when thresholding is later applied.

Since the output of the SVM-based approach is binary, in optimising the SVM parameters, objectives *obj1* and *obj2* took their values from the TPR and FPR directly. It should be noted that the NSGA-II implementation utilised here only supports the minimisation of objective functions and so in reality *obj1* was calculated as 1-TPR.

Simulation Environment

All of the experiments were conducted using the Stage simulator. For every generation, each individual defined the parameters of the anomaly detection system on-board a single robot. In this set of experiments, the robots did not consume energy. At the

²<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

Name	Description	Formula
True Positive Rate (TPR)	Also known as the <i>sensitivity</i> , the TPR measures the percentage of anomalies that are correctly identified	$\frac{TP}{(TP+FN)}$
True Negative Rate (TNR)	Also known as the <i>specificity</i> , the TNR measures the percentage of normal instances that are correctly identified	$\frac{TN}{(TN+FP)}$
False Positive Rate (FPR)	One minus the TNR	$1 - TNR$
Positive Predictive Value (PPV)	Otherwise known as the <i>precision</i> , the PPN measures the percentage of instances labelled as anomalous which truly are	$\frac{TP}{(TP+FP)}$
Negative Predictive Value (NPV)	The percentage of instances labelled as normal which truly are	$\frac{TN}{(TN+FN)}$
Accuracy (ACC)	The proportion of all instances that were correctly classified, either positively or negatively	$\frac{TP+TN}{(TP+FP+TN+FN)}$

Table 5.9: Statistical measures for assessing the performance of the mDCA and SVM based anomaly detection systems. TP refers to the number of true positives, that is: anomalies which are correctly identified as such; TN refers to true negatives: normal data that is correctly identified as such; FP refers to false positives: normal data that is incorrectly identified as anomalous; and FN refers to false negatives: anomalous data that is incorrectly identified as normal

start of each run, the robots were randomly positioned within a large obstacle filled arena and left to perform random wandering with obstacle avoidance for one simulated hour. Three of each of the different types of anomaly described in section 5.2.2 (nine in total) were injected into each robot over the course of each run. To avoid bias in the evaluation, the total time in which a robot experienced each particular type of anomaly was balanced, however, the length of the individual anomalies themselves were not necessarily the same. At the end of each run *obj1* and *obj2* were calculated as previously described.

Because SVMs require training before they may be used as classifiers, in the SVM experiment, an extra step was needed before evaluating the individuals within the simulator. Prior to the optimisation experiment itself, raw sensor data was generated from 50 runs involving 50 robots operating for one hour, in which time nine anomalies were injected into each robot. During optimisation, for each generation, the data from a different one of the 50 runs was used to train the SVM classifiers. For each individual, training data was generated by sampling the raw sensor data of every robot in the run using a sliding window. The size of the window was equal to the size of the individual's largest feature window and the step size was set to half of this value. Samples in which the final two thirds of the data was not either exclusively anomalous or exclusively normal were discarded. Using the parameters of the individual, for each sample, three feature values were calculated and labelled accordingly. After the data was collected and the feature values were calculated, the set of points labelled as normal was reduced to the size of the set of points labelled as anomalous and combined to produce the training set. The training data was then used to train the SVM model before evaluation.

5.3.3 Results

This section presents the results from the mDCA and SVM optimisation experiments. For each experiment, the evolutionary progress is discussed and the classification performance of the evolved system is analysed in terms of the true positive and false positive rates. In the case of the mDCA-II and SVM experiments, the frequency with which each of the individual features was selected by the optimisation process is also presented.

mDCA with Fixed Features (mDCA-I)

The mDCA-I experiment was designed to emulate the original mDCA system introduced by [120]. The parameters that determine which features are used were fixed to values that replicate the features of the original experiments, whilst the remaining parameters were optimised freely.

Figure 5.3a shows the evolutionary progress of the experiment over time. For every individual member of each generation, the value of *obj1* is plotted against the value of *obj2*. Members of the initial population are shown in light grey whilst members of the final population are shown in black. Members of the intermediate populations are shown in varying shades between light grey and black.

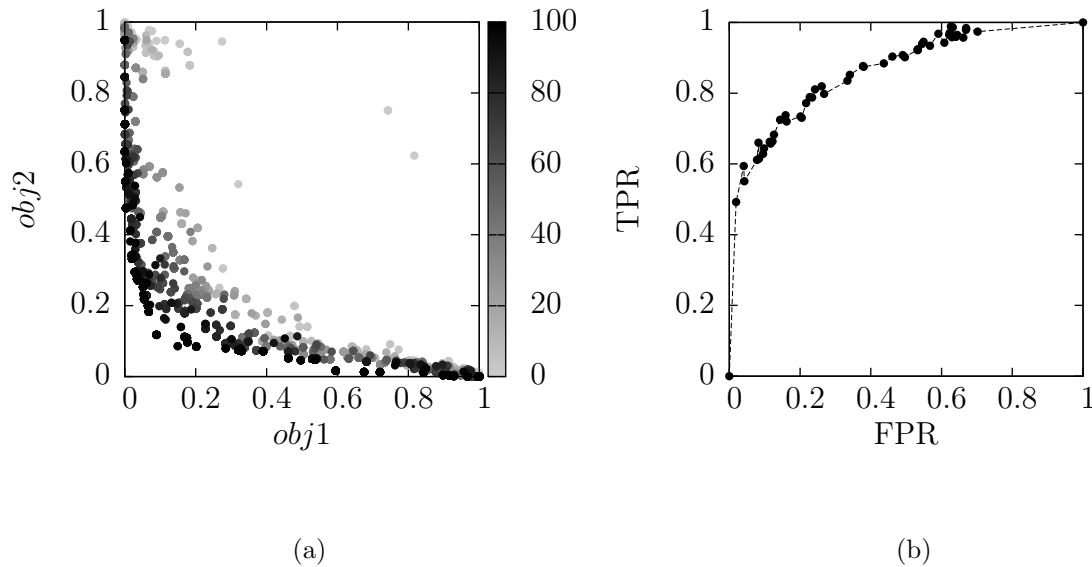


Figure 5.3: Plots showing the evolutionary progress of the mDCA-I experiment (a) and the trade-off between the TPR and FPR of an evolved system (b)

In figure 5.3a, it can be observed that the members of the final population lie closer to y-axis than they do to the x-axis. The positioning of the members of the final population demonstrate that it was easier to minimise *obj1* than it was to minimise *obj2*, implying that the system may be skewed towards producing solutions which favour maximising the TPR rather than minimising the FPR.

To analyse the performance of the optimised system further, fifty of the best individuals from the optimisation process were identified and subjected to further tests. In a multi-objective task, what constitutes the ‘best’ individual may be difficult to define. One method of defining the best individuals is consider the set of *Pareto optimal* solutions. A Pareto optimal solution is a solution for which no objective can be improved without decreasing the value of another objective, because of this property, Pareto optimal solutions are also known as ‘non-dominated’ solutions. Together, the set of Pareto optimal (or non-dominated) solutions define the ‘Pareto front’. In theory, all members of the Pareto front may be considered equal. However, when optimising solutions to a classification task, where it is desirable to maximise the TPR but minimise the FPR, the Pareto front may include solutions in which the TPR and FPR are both either maximised or minimised, neither of which is desirable.

For this experiment, the fifty ‘best’ individuals were selected by calculating which solutions laid closest to the origin in figure 5.3a. Selecting solutions in this manner confers an equal importance to the TPR and FPR. In comparing different systems, this approach provides a convenient common starting point. However, as highlighted

	<i>stdDev</i>	<i>distAve</i>	<i>bDistAve</i>	<i>distNN</i>	<i>distN</i>	<i>skew</i>	<i>mean</i>	<i>distStdDev</i>	<i>range</i>	<i>pairDist</i>
Count	5	0	31	17	3	1	18	0	26	43
# Ind.	5	0	30	17	3	1	18	0	19	38

Table 5.10: A summary of features used during the final generation of the mDCA-II optimisation experiment. The second row of the table shows the number of times each feature appeared within the final population, whilst the third row shows the number of individual within the final population that contained each feature

in section 5.5, there are scenarios in which it would be beneficial if the TPR and FPR were not equally weighted. It is in these scenarios that determining the Pareto optimal set would be beneficial and the advantages of the NSGA-II approach are observed.

Through repeated re-evaluation, the selected individuals were compared with one another. Each individual was re-evaluated 50 times and the average of the two objectives was calculated. From these results, a single best individual was selected and its classification performance was evaluated. To perform classification with the mDCA, the output of the algorithm must be thresholded. If the output of the mDCA is greater than the chosen threshold, the algorithm signals that the input belongs to one class, and if the output is less than or equal to the threshold, it signals a different class.

To evaluate the algorithm’s performance, the threshold value was varied between 0 and 1 in increments of 0.02. For each threshold value, a single experimental run, containing 50 identical individuals, was performed. As with the previous optimisation experiments, each run lasted for one simulated hour, during which time nine faults were injected into each robot.

Figure 5.3b plots the average TPR and FPR for each threshold value, producing what is often referred to as a receiver operating characteristic (ROC) curve. A common metric used to compare the performance of binary classifiers is the area under a ROC curve (AUC). A perfect classifier would have an AUC of 1, whilst a random classifier would have an AUC of 0.5. The AUC of the curve in figure 5.3b is 0.877.

mDCA with Evolved Features (mDCA-II)

In the mDCA-II experiment, all of the mDCA parameters were allowed to evolve freely, including the features and the time windows over which the feature values were calculated. Figure 5.4a shows the evolutionary progress of the experiment over time. Figure 5.4b shows a ROC curve created by varying the threshold value of a single individual, selected in the same manner as was done with the mDCA-I experiment. The AUC of the curve in figure 5.4b is 0.894. A video of one of the best individuals from the mDCA-II experiment is provided in the accompanying material (video 5.1).

Figure 5.5 shows how many times each feature was present in each generation, and how the features evolved over time. Table 5.10 summarises the number of times each feature was present in the final generation and the number of individuals within the final generation which contained each feature.

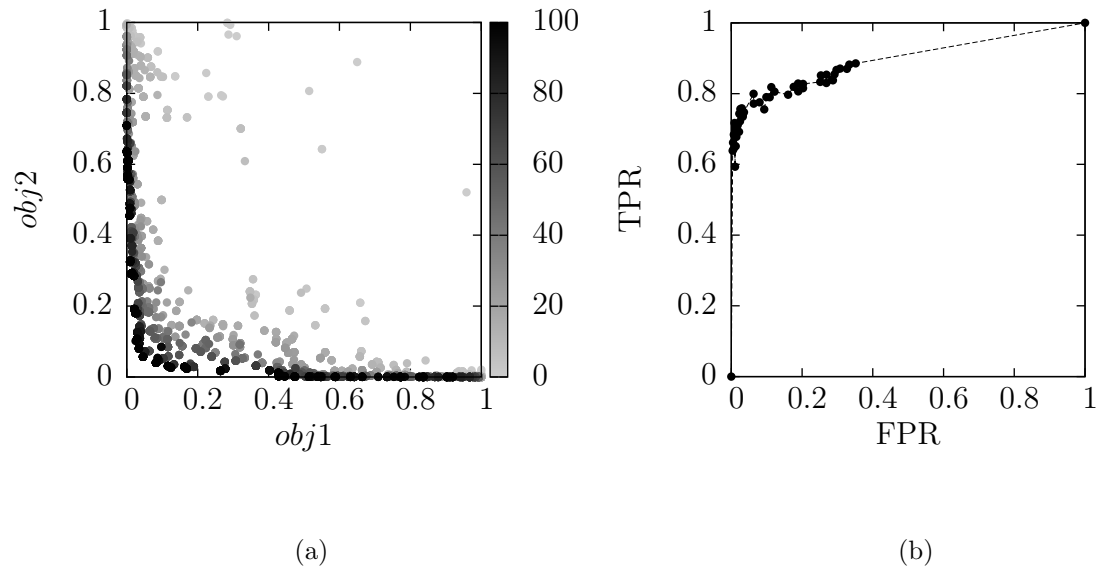


Figure 5.4: Plots showing the evolutionary progress of the mDCA-II experiment (a) and the trade-off between the TPR and FPR of an evolved system (b)

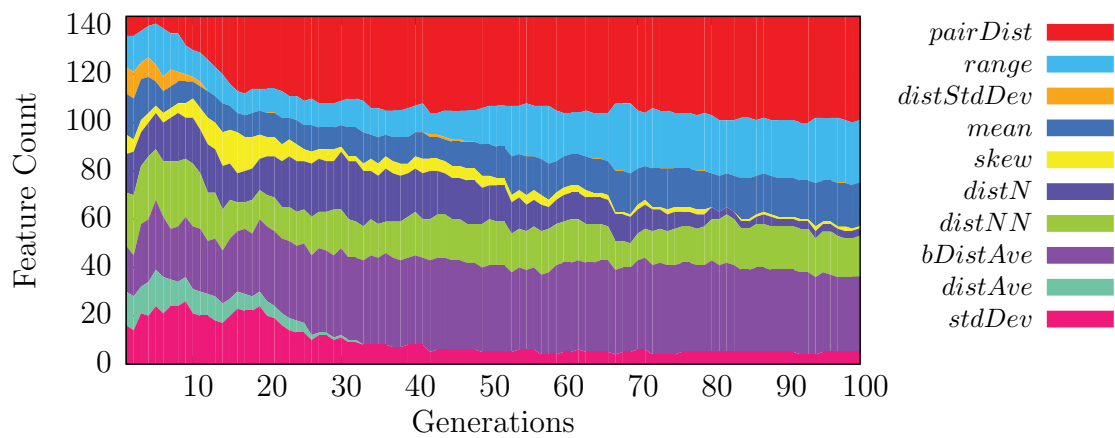


Figure 5.5: The number of times each feature was used within each generation of the mDCA-II optimisation experiment

	<i>stdDev</i>	<i>distAve</i>	<i>bDistAve</i>	<i>distNN</i>	<i>distN</i>	<i>skew</i>	<i>mean</i>	<i>distStdDev</i>	<i>range</i>	<i>pairDist</i>
Count	23	13	1	3	5	14	42	4	8	31
# Ind.	22	13	1	3	5	12	35	4	7	27

Table 5.11: A summary of features used during the final generation of the SVM optimisation experiment. The second row of the table shows the number of times each feature appeared within the final population, whilst the third row shows the number of individuals within the final population that contained each feature

As shown by figure 5.5 and table 5.10, the most selected feature was *pairDist*, which appeared 43 times in the final generation and was used by 38 of the 48 individuals. The next most popular was *bDistAve*, followed by *range*.

It is interesting to note that the *distN* feature, which was introduced in order to combat the perceived deficiencies of the original *distNN* feature, was found only 3 times in the final generation. In comparison, the *distNN* feature was present 17 times. Whilst *distNN* was only the fourth most selected feature, it was present in what was by far the most popular *combination* of features. Along with *bDistAve* and *pairDist*, *distNN* appeared in 17 of the individuals from the final generation. The next most popular combination of features was the *mean*, *range* and *pairDist*, which appeared together in nine individuals.

Of the original features used by the mDCA, *bDistAve* was the most popular, appearing 31 times in 30 individuals. Meanwhile, *distAve* was the least popular of the original features and along with *distStdDev* was the least popular of all the available features, both of which were not found in any member of the final population.

SVM with Evolved Features

In the SVM experiment, only the features and the ranges over which their values were calculated were evolved. Figure 5.6a shows the evolutionary progress of the experiment over time. Both objectives are observed to converge very quickly to optimal values. The fast convergence may, at least partially, be attributed to the fact that each individual SVM classifier was trained prior to evaluation. In this experiment, evolution was not used to define the entire classifier, but merely the features upon which it performs classification. Figure 5.6b shows a ROC curve created in a similar manner to figures 5.3b and 5.4b by first selecting the best individual and then varying its discrimination threshold. The AUC of the curve in figure 5.6b is 0.983.

Figure 5.7 shows how many times each feature was present in each generation of the SVM optimisation experiment and table 5.11 summarises how many times each feature was present in the final population.

The frequency with which each feature appears is very different to the outcome of the mDCA-II experiment. As can be seen in table 5.11, the most popular feature was the *mean* which appeared 42 times in the final generation and was used by 35 of the 48 individuals. The next most popular were *pairDist*, appearing 31 times in 27

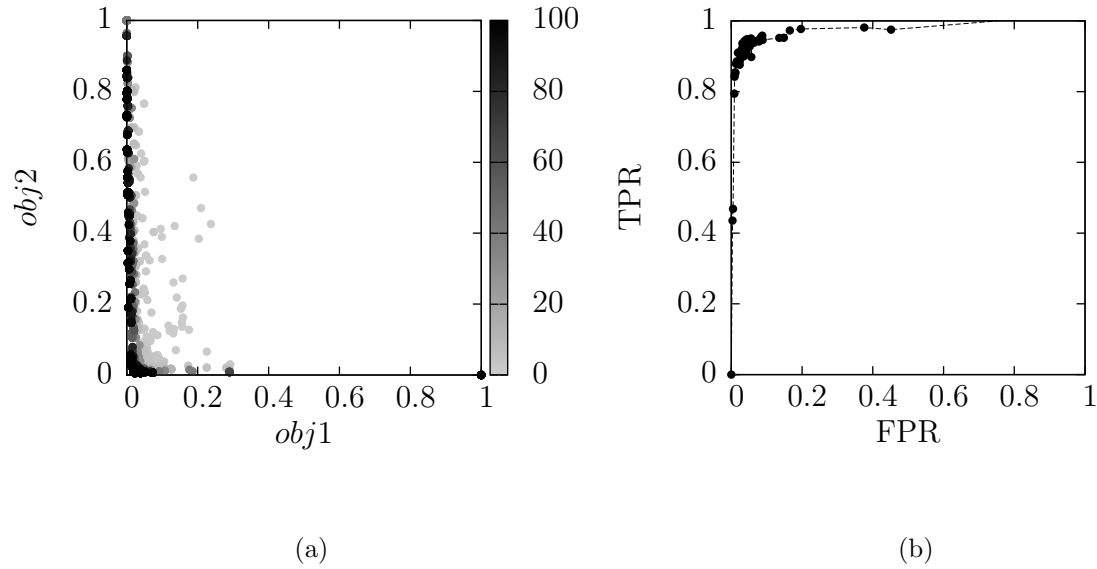


Figure 5.6: Plots showing the evolutionary progress of the SVM experiment (a) and the trade-off between the TPR and FPR of an evolved system (b)

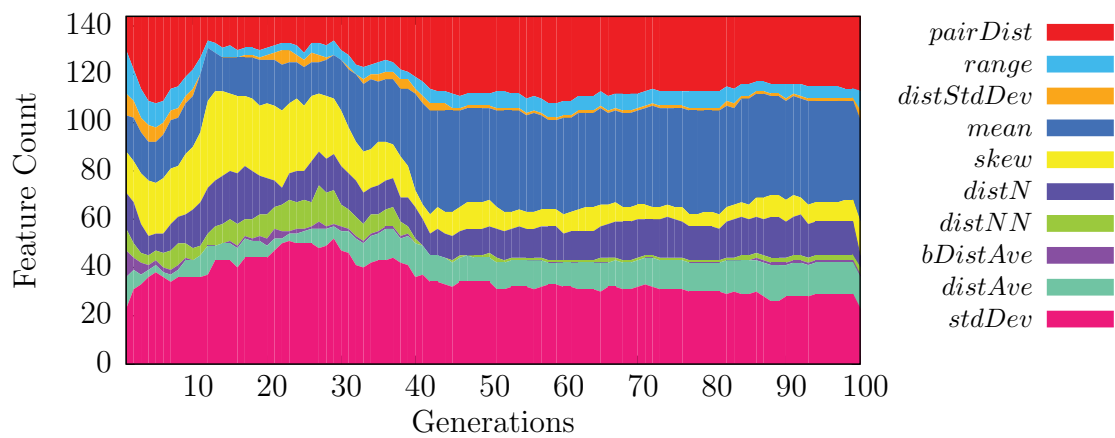


Figure 5.7: The number of times each feature was used within each generation of the SVM optimisation experiment

individuals and *stdDev* appearing 22 times in 13 individuals.

None of the features used by the original mDCA were utilised very often and, in a reversal of the findings from the mDCA-II experiment, the *distNN* feature was found to be more popular than *distN*. Unlike the mDCA-II results, no particular combination of features stood out amongst the others. The most frequent combination contained the three most popular features: *mean*, *pairDist* and *stdDev*, but appeared only seven times in the final population.

5.3.4 Analysis

In this section, the results of the optimisation experiments are analysed and comparisons are made between the mDCA-I, mDCA-II and SVM systems. Specifically, the evolutionary progress, classification accuracy and speed of the systems is analysed.

Evolutionary progress

Figure 5.8a compares the objective values of members of the final population, during the mDCA-I (grey) and mDCA-II (black) experiments. Both objectives were minimised further in the mDCA-II experiment than they were in the mDCA-I experiment. This result demonstrates that the newly introduced features, and the loosening of the restrictions upon the older features, brought an evolutionary advantage to the mDCA-II experiment, in spite of the increase in the size of the search space.

In analysing the features selected during the mDCA-II experiment, it can be observed that only one of the original three features, *bDistAve*, was still widely utilised in the final generation. Meanwhile, the *distN* feature, which was introduced to target a perceived deficiency in the *distNN* feature appeared to bring no evolutionary advantage and was present fewer times in the final generation of the mDCA-II experiment than the original *distNN* feature was. The *distN* feature relies on comparing the values of three neighbouring sensors, two of which are positioned on the same side of a robot, and one of which is positioned on an adjacent side. The feature was chosen based upon the assumption that neighbouring sensors will return similar values (regardless of which side they are positioned on). The fact that the *distN* feature was selected fewer times than the *distNN* feature—which only compares sensor values from one side of a robot—may indicate that this initial assumption was incorrect.

Because of the subtly different objective functions that they utilise, it is not possible to fairly compare the evolutionary progress of the mDCA and SVM experiments. However, it is possible to compare the features that were selected.

Figure 5.9 compares the number of times that each feature was present in the final generation of the mDCA-II and SVM experiments. As shown in figure 5.9, the features which worked well for the SVM-based system are very different to those which worked well for the mDCA. The *bDistAve*, *distNN* and *range* features, whilst well represented in the mDCA-II experiment, were far less commonly found in the final generation of the SVM experiment. Meanwhile, the *stdDev*, *distAve* and *skew* features, whilst not

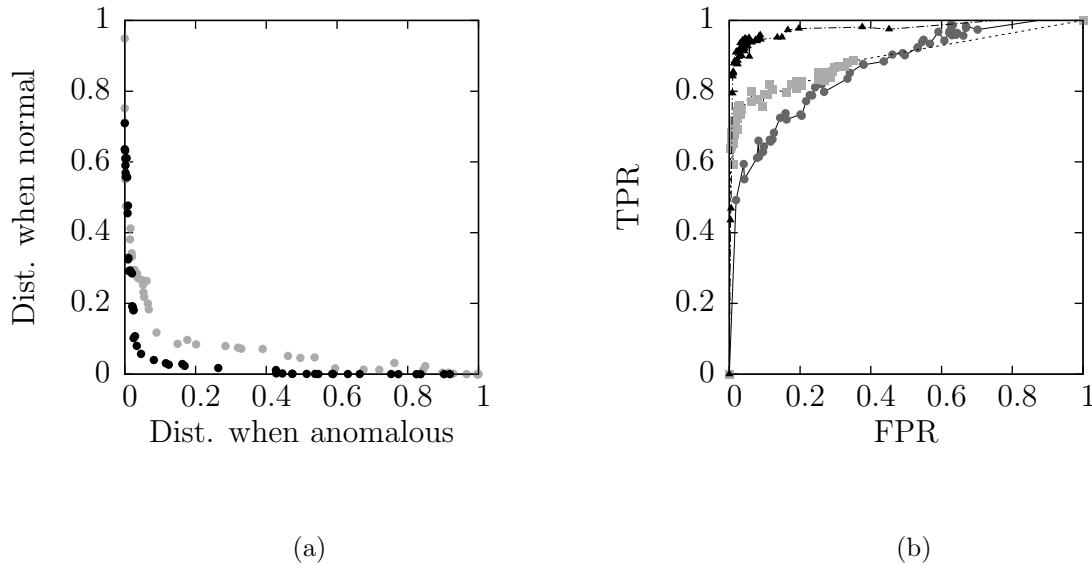


Figure 5.8: Graphs comparing the objective function scores of the mDCA-I (grey) and mDCA-II (black) systems (a) and the TPR and FPR of the mDCA-I (dark grey circles), mDCA-II (light grey squares) and SVM (black triangles) systems (b)

Exp.	mDCA-I	mDCA-II	SVM
AUC	0.876539	0.894397	0.983085

Table 5.12: A comparison of AUC values from the mDCA-I, mDCA-II and SVM systems

uncommon in the SVM experiment, were rarely found in the mDCA-II experiment. One exception to this pattern is the *pairDist* feature, which was well represented in the final generation of both experiments.

Classification accuracy

Figure 5.8b compares the ROC curves from the mDCA-I, mDCA-II and SVM experiments, whilst table 5.12 compares the AUC values of the three systems. Out of the three systems, the SVM-based approach is observed to be the best classifier, as demonstrated by the fact that it has the largest AUC value, and its curve lies closest to the point (0,1). The mDCA-II based system narrowly outperforms that of the mDCA-I, with the two systems obtaining AUC scores of 0.8944 and 0.8765 respectively. In comparison to the mDCA-I system, the curve of the mDCA-II system is skewed towards the left-hand side of the graph, indicating that at low false positive rates, the mDCA-II approach will have a higher true positive rate.

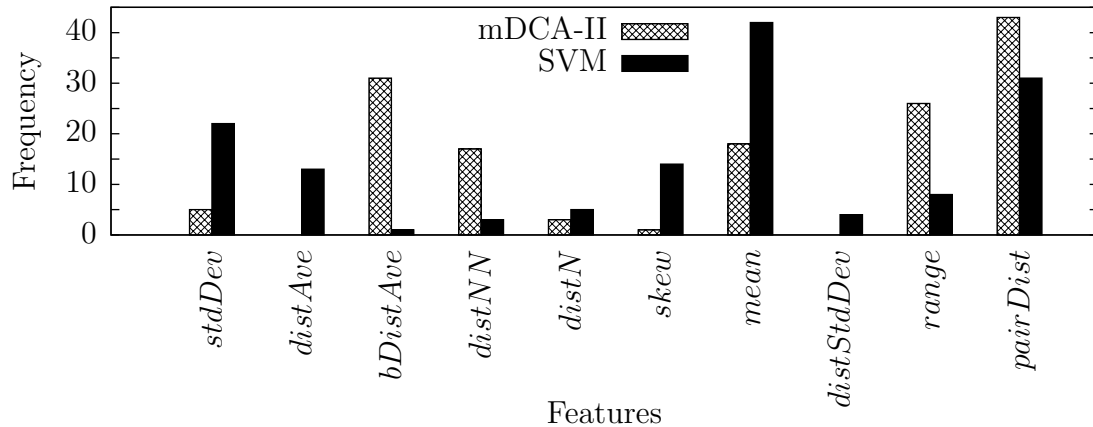


Figure 5.9: A histogram comparing the number of times each feature was present in the final generation of the mDCA-II and SVM optimisation experiments

Exp.	mDCA	SVM
Time (s)	947.44 [6.675]	1423.92 [16.116]

Table 5.13: The mean run-time (and std dev.) of the mDCA and SVM experiments

Speed

As an informal measure of the complexity of the mDCA and SVM based systems, empirical data was gathered regarding the run-time speed of the two approaches. For each approach, 25 runs were executed, each containing 50 robots and lasting for one simulated hour, during which time nine anomalies were injected into each robot. The simulator was configured to execute the simulation as fast as possible and the real time taken was measured using the Linux/UNIX `date` command. The experiments were carried out on a desktop computer. In order to ensure that each run was performed under similar conditions and obtained a similar proportion of CPU time, networking was turned off and all other applications were closed.

The results of these experiments are presented in table 5.13. The table compares the mean time taken to complete a single run. From this table it can be seen that the mDCA system significantly outperforms the SVM-based approach in terms of speed. On average, the mDCA runs were just under eight minutes shorter, or 1.5 times quicker.

5.4 Long-term Survival

In all of the experiments described so far, the robots were provided with an unlimited supply of energy, and were not required to perform any of the recharging or energy sharing behaviours introduced in section 5.2.3. Here, this simplification is removed and

the full controller, along with its associated recovery strategy, is utilised.

In chapter 1, it was suggested that for self-reconfigurable modular robotic systems to be useful in tasks such as autonomous search and rescue, they would need to demonstrate the ability to survive for long periods of time, without any form of human interaction. It was further stated that, to survive autonomously for long periods of time, the robots would need to demonstrate high levels of fault tolerance. In the same chapter, the 100 Robots 100 Days grand challenge was presented as a test-bed for assessing whether a system possessed the necessary characteristics for long-term survival.

Using a simplified version of the 100 Robots 100 Days challenge, this section investigates the extent to which anomalous sensor data may affect the long term survival of a collective robotic system. Furthermore, this section considers whether the introduction of an anomaly detection system and recovery strategy can reduce or remove any of the adverse effects that result from the presence of anomalous sensor data.

After describing the experimental setup, results are presented which document the long-term survival of three different types of system. Survival is measured both by the number of robots that remain active at the end of a run and the total amount of energy stored throughout. Results from a control experiment in which no anomalies were introduced and two baseline experiments in which no anomaly detection and no energy sharing took place, are also presented.

5.4.1 Experimental Setup

Two different sets of experiments were designed in order to investigate the long-term survival of a collective robotic system in the presence of anomalous sensor data. The first set of experiments focused on comparing the long term survival of various different types of system, some of which were capable of anomaly detection and others of which were not. The second set of experiments considered how varying the FPR of systems which are capable of anomaly detection affects their performance.

In the first set of experiments, the performance of six different systems was compared. The first experiment in this set was a control. In the control experiment, the robots used the behavioural controller described in section 5.2.3 but no anomalies were injected into their sensor data. The second and third experiments were used to determine a baseline level of performance. Both involved introducing anomalies into the proximity sensors of robots which possessed no method for detecting or responding to them, however, in the third experiment, the robots' energy sharing capabilities were also removed. The fourth and fifth experiments were used to determine whether the pairing of an anomaly detection system and appropriate recovery strategies could improve the performance of a system containing anomalous sensor data. Specifically, the two anomaly detection systems were defined by the parameter sets of the best individuals from the mDCA-II and SVM optimisation experiments. The sixth and final experiment involved the use of an 'ideal' anomaly detection system, which was able to respond immediately to every anomaly.

For each experiment, ten individual runs were carried out. In each run, 50 robots

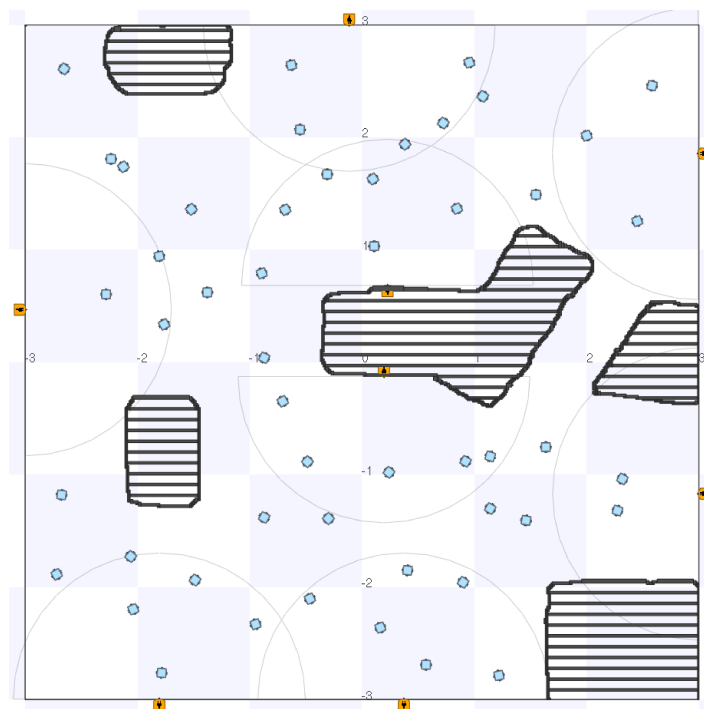


Figure 5.10: The environment used during the long-term survival experiments. The 50 small blue squares are robots and the eight, slightly larger, orange objects are power sockets. The range and area over which the power sockets may communicate with the robots is shown by the semi-circles. The black striped areas are obstacles

were simulated and 12 anomalies were injected into their sensor data over a period of ten simulated hours. At the start of each run, the robots were positioned at random within a large obstacle filled arena and assigned a random energy level between 30% and 100% of their maximum. The arena used is pictured in figure 5.10. At each timestep, the total energy stored in every robot and the total number of surviving robots (robots with an energy level greater than zero) was recorded. Robots which ran out of energy were removed from the environment immediately. Table 5.14 presents 10 null hypotheses which are used in section 5.4.3 to compare the performance of the different systems.

In the mDCA and SVM experiments discrimination thresholds were set to values that would ensure a low false positive rate of around 0.01. In the second set of experiments, to investigate the effects of altering the FPR of these approaches, four further experiments were performed. For both of the approaches, new discrimination thresholds were chosen that would ensure false positive rates close to 0.1 and 0.2. As with the experiments in which the FPR was set to 0.01, ten individual runs were completed, each lasting for ten simulated hours and involving the injection of 12 anomalies.

$H5.1_0$	There is no difference in the amount of stored energy present at the end of an experiment using the ideal anomaly detection system and an experiment using the SVM-based approach
$H5.2_0$	There is no difference in the number of surviving robots present at the end of an experiment using the ideal anomaly detection system and an experiment using the SVM-based approach
$H5.3_0$	There is no difference in the amount of stored energy present at the end of an experiment using the ideal anomaly detection system and an experiment using the mDCA-based approach
$H5.4_0$	There is no difference in the number of surviving robots present at the end of an experiment using the ideal anomaly detection system and an experiment using the mDCA-based approach
$H5.5_0$	There is no difference in the amount of stored energy present at the end of an experiment using the mDCA anomaly detection system and an experiment using the SVM-based approach
$H5.6_0$	There is no difference in the number of surviving robots present at the end of an experiment using the mDCA anomaly detection system and an experiment using the SVM-based approach
$H5.7_0$	There is no difference in the amount of stored energy present at the end of an experiment using no form of anomaly detection system and an experiment using the SVM-based approach
$H5.8_0$	There is no difference in the number of surviving robots present at the end of an experiment using no form of anomaly detection system and an experiment using the SVM-based approach
$H5.9_0$	There is no difference in the amount of stored energy present at the end of an experiment using no form of anomaly detection system and an experiment using the mDCA-based approach
$H5.10_0$	There is no difference in the number of surviving robots present at the end of an experiment using no form of anomaly detection system and an experiment using the mDCA-based approach

Table 5.14: Hypotheses used to compare the long-term survival of different systems

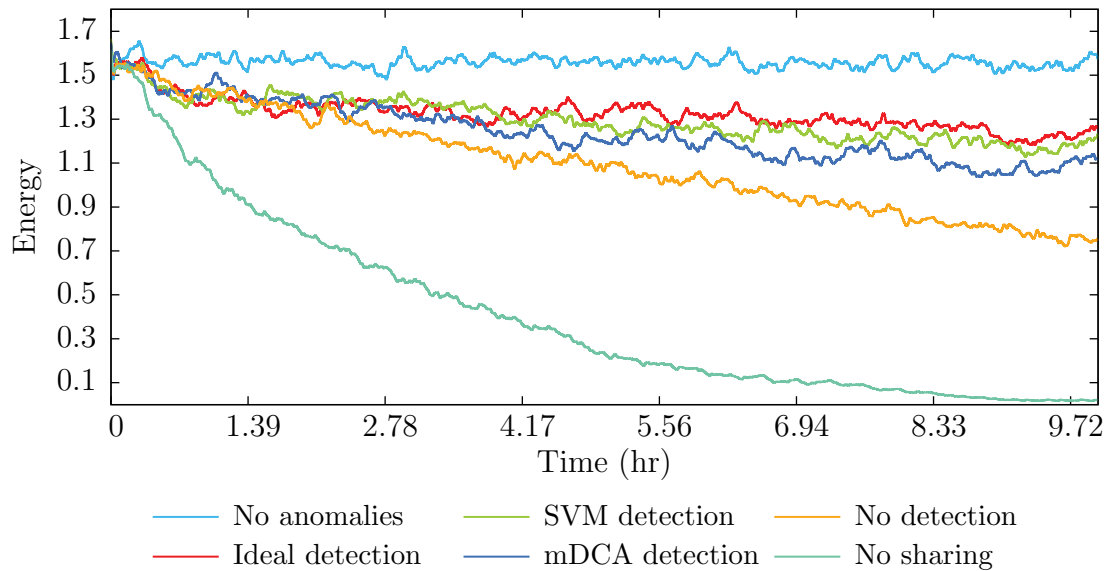


Figure 5.11: The total stored energy of 50 robots operating over a ten hour period. Each line represents the mean outcome of ten independent runs

5.4.2 Results

Figure 5.11 shows how the total energy stored by all of the robots in the system changes over the time. Each line corresponds to a single experiment and shows the mean value from ten runs. A video showing a short segment of one of the runs is provided in the accompanying material (video 5.2). In the experiment in which no anomalies were present (light blue line), the total stored energy remains relatively constant. Contrastingly, in the experiment in which anomalies were present but there was no anomaly detection (orange line), the total energy stored steadily decreases over time. Meanwhile, in the experiment in which no sharing took place (teal line), the total stored energy drops quickly at first, before levelling out as the amount approaches zero. The three experiments in which anomaly detection, energy sharing and recovery did take place all follow a similar trend. Initially, the perfect detection, SVM and mDCA based approaches all follow the same linear decrease as the experiment with no detection. However, after about one hour, the energy levels of the systems with anomaly detection begin to level out and although they continue to decrease for the remainder of the simulation, they do so at a lower rate than the experiment with no detection, one potential explanation for this behaviour is provided in section 5.4.3.

Figure 5.12 shows the mean amount of energy stored at the end of each of the experiments, represented using boxplots³. Results from the experiment in which no

³All of the boxplots presented in this thesis use the following drawing conventions: the horizontal line represents the median; the bottom and top of the boxes represent the first and third quartiles; and the upper and lower whiskers represent the highest and lowest data points that are found no further than 1.5 times the inter quartile range from the edges of the boxes.

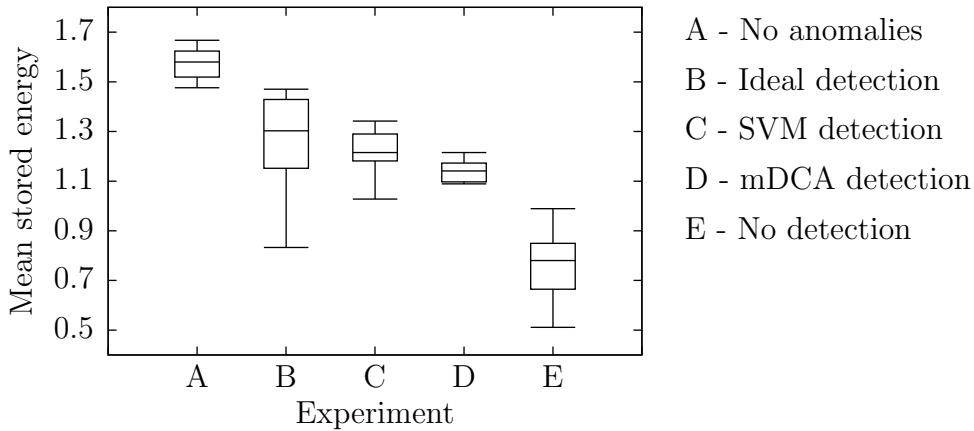


Figure 5.12: The amount of energy present at the end of each experiment

	B	C	D	E
A	0.000157	0.000157	0.000157	0.000157
B	-	0.650147	0.069642	0.000381
C	-	-	0.058781	0.000157
D	-	-	-	0.000507

Table 5.15: The p-values obtained when comparing the amount of stored energy present at the end of experiments involving systems A-E, using a Wilcoxon rank-sum test with a significance level of 0.05. The values in grey highlight where there is a significant difference between two systems, after a Bonferroni correction factor is applied

energy sharing took place are omitted from this figure since, in all but three of the runs, every robot had ran out of energy by the end of the run. The remaining stored energy was greatest at the end of the experiment in which no anomalies were present and lowest in the experiment in which no anomaly detection took place. In comparing the three systems in which anomaly detection was present, it can be observed that the system with ideal anomaly detection outperformed the SVM-based system, which in turn outperformed the mDCA-based system. It should be remembered, however, that it was only possible to conduct ten repeated runs. As shown by the p-values in table 5.15, using a Wilcoxon rank-sum test with a significance level of 0.05 and applying a Bonferroni correction factor to account for multiple comparisons, the difference between the final stored energy of all three of the systems with anomaly detection cannot be said to be statistically significant. However, all of the other systems can be said to significantly outperform the system in which no detection took place.

Figure 5.13 shows how the total number of active robots changed over time. Due to the strong relationship between the amount of energy in the system and the number of surviving robots, the lines in figure 5.13 follow a similar trend to those in figure 5.11.

Figure 5.14 shows the mean number of active robots present at the end of each

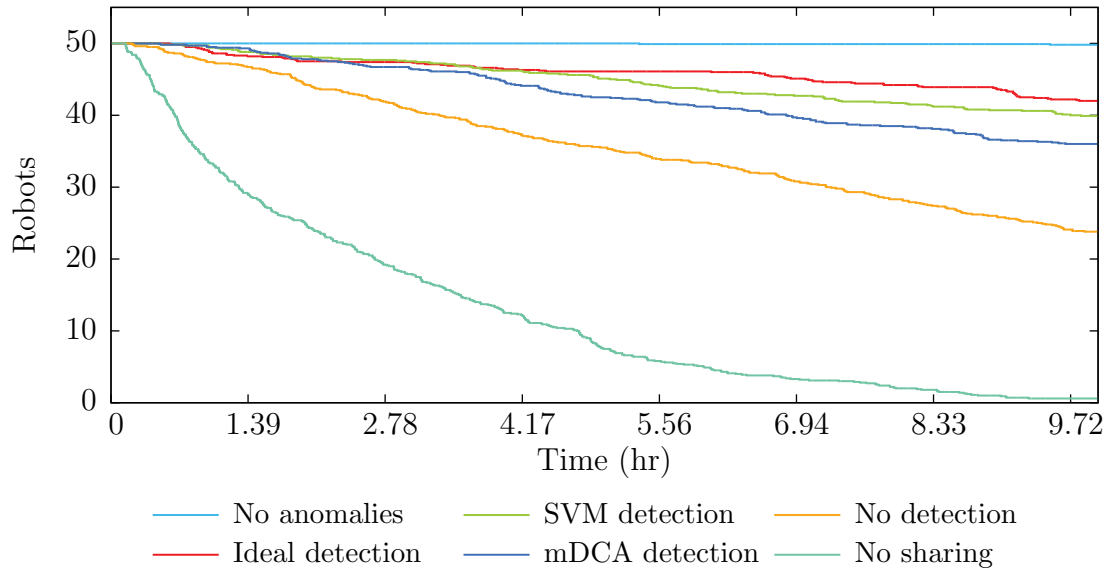


Figure 5.13: The total number of operational robots over a ten hour period. Each line represents the mean outcome of ten independent runs

	B	C	D	E
A	0.000157	0.000157	0.000157	0.000108
B	-	0.185877	0.012611	0.000108
C	-	-	0.037635	0.000108
D	-	-	-	0.000287

Table 5.16: The p-values obtained when comparing the number of surviving robots present at the end of experiments involving systems A-E, using a Wilcoxon rank-sum test with a significance level of 0.05. The values in dark grey highlight where there is a significant difference between two systems before a Bonferroni correction factor is applied and the values in light grey highlight where there is still a significant difference after applying a Bonferroni correction factor

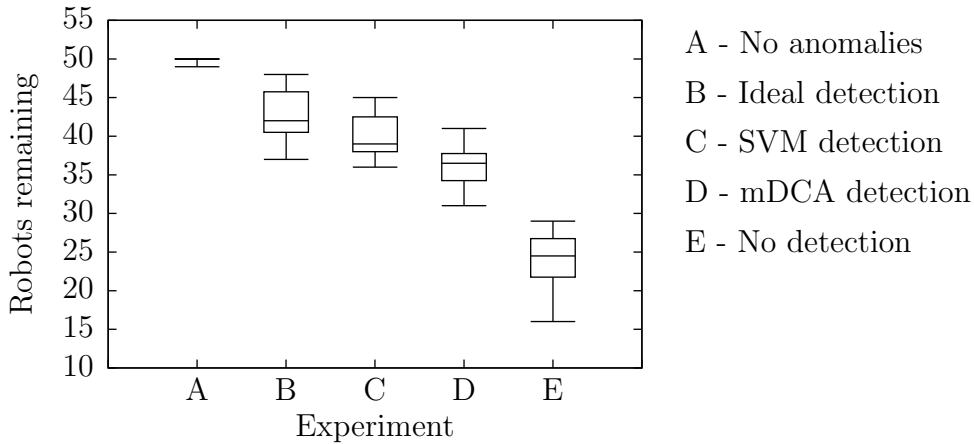


Figure 5.14: The number of robots present at the end of each experiment

of the five experiments which involved energy sharing. The results once again reflect those which recorded the amount of stored energy. The number of surviving robots was found to be greatest in the experiment in which no anomalies were introduced and lowest in the experiment with no detection. The ideal system still outperformed the SVM-based approach, which in turn outperformed the mDCA-based system. Using a Wilcoxon rank-sum test with a significance level of 0.05, it can be stated that there is no significant difference in the number of surviving robots in the ideal and SVM-based systems. As shown in table 5.16, before the Bonferroni correction is applied, there is a significant difference between the mDCA system (D) and both the ideal (B) and SVM (C) systems, however after applying the Bonferroni correction, the difference can no longer be said to be statistically significant. As was the case when comparing the amount of stored energy, all of the other systems can be said to significantly outperform the system in which no detection took place.

Figures 5.15 and 5.16 show how altering the false positive rate of the mDCA and SVM anomaly detection systems affected the amount of stored energy and number of remaining robots in the long-term survival experiments. As reported above, with a FPR of 0.01, the SVM-based approach significantly outperforms the mDCA-based approach in terms of the number of surviving robots. Whilst the amount of stored energy is also greater with the SVM-based approach, the difference is not statistically significant.

Interestingly, in the experiments in which the FPR was set to 0.1 and 0.2, the findings are reversed. Despite the SVM-based approach consistently having a greater TPR, it is outperformed by the mDCA-based approach both in terms of the amount of energy stored and the number of surviving robots. It should be highlighted, however, that although the median value from the mDCA experiments is greater than the median from the SVM experiments, the difference is not statistically significant.

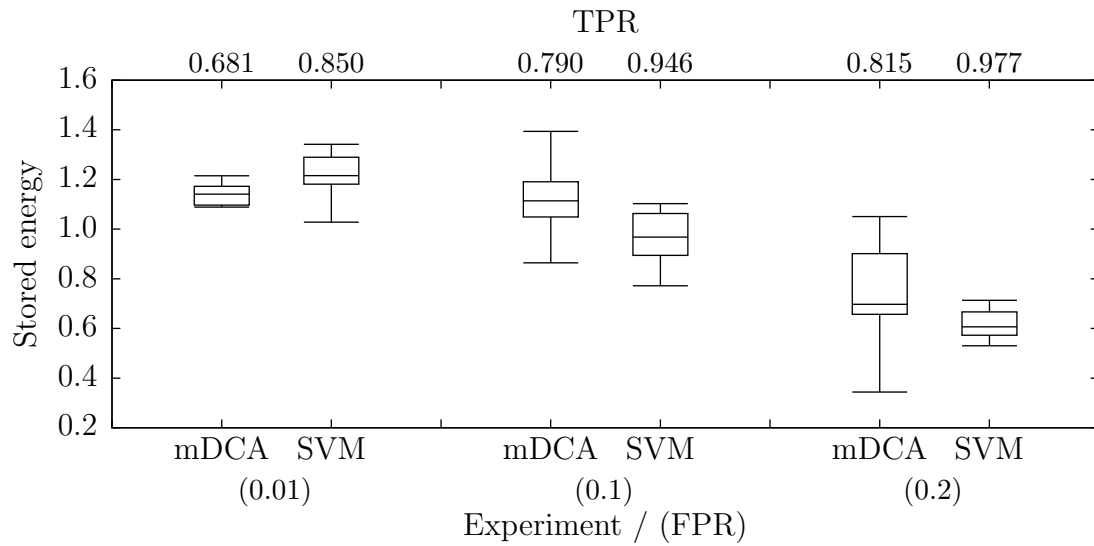


Figure 5.15: Boxplots showing the effect of varying the false positive rate (FPR) on the amount of energy present at the end of each experiment

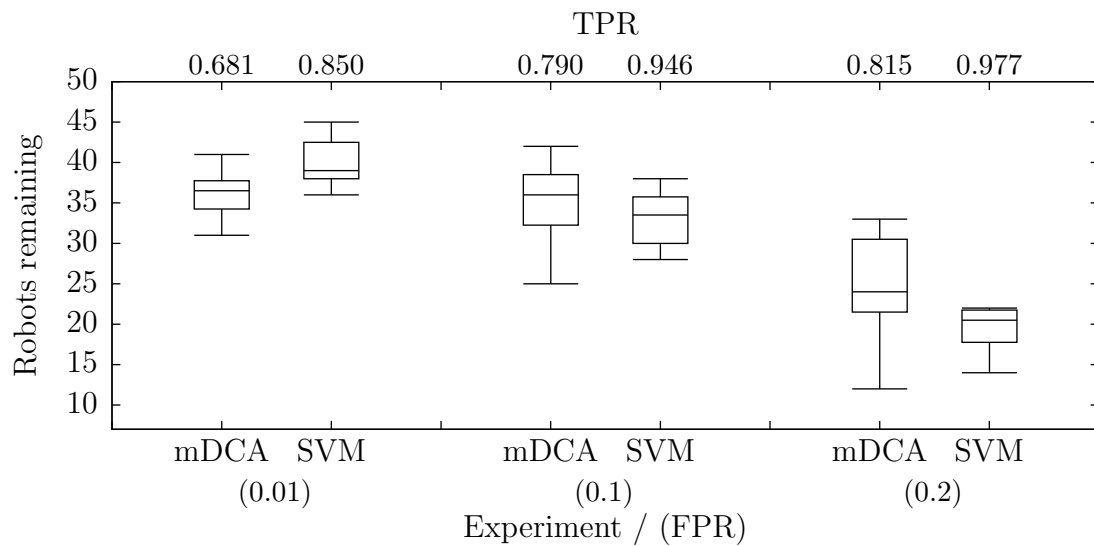


Figure 5.16: Boxplots showing the effect of varying the false positive rate (FPR) on the number of robots present at the end of each experiment

5.4.3 Analysis

In this section, the results of the long-term survival experiments are analysed and each of the ten hypotheses from the table 5.14 is examined. It is important to highlight that, given the low number of repeated runs, these results must be interpreted with caution.

During the course of the long-term survival experiments, the system in which no anomalies were injected performed consistently better than the others. The performance of the three systems which utilised anomaly detection was similar. However, during the second half of the experiments in particular, the system which employed ideal anomaly detection consistently outperformed the SVM-based system, which in turn outperformed the mDCA-based approach. All of the other systems outperformed the two which did not utilise any form of anomaly detection. From the two systems which did not perform anomaly detection, the system in which robots were able to share energy with one another performed consistently better than the system in which no sharing took place, highlighting the benefit of the energy sharing behaviours.

Intuitively, one might expect the performance of the ideal detector to match that of the experiment in which there were no anomalies. The reason that this is not so is because the recovery strategy is not perfect. It was only the anomaly detection in this experiment that was ideal. The basic response to the detection of an anomaly in a sensor is to assume that the sensor is returning the same value as its neighbour, unfortunately, this assumption is not always valid and in situations where two neighbouring sensors differ by a large amount, the true value may in fact be closer to the anomalous value than that of the neighbour. Furthermore, if an anomaly is detected in one of a robot's front two sensors, that robot is prevented from recharging both itself and its fellow robots, it becomes a burden upon the group, able to take from other robots but not able to give back. This is why the ideal system does not perform as well as the case in which there were no anomalies.

In comparing the ideal system and the SVM-based approach, it can be said that there is no significant difference in the amount of stored energy present at the end of each experiment and no significant difference in the number of surviving robots. Based upon this evidence it is not possible to reject $H5.1_0$ or $H5.2_0$ and it is accepted that there is no difference in the performance of a system using ideal anomaly detection and a system using the SVM-based approach.

In comparing the ideal system and the mDCA-based approach, it can be said that there is no significant difference in the amount of stored energy present at the end of each experiment and, after applying the Bonferroni correction factor, no significant difference in the number of surviving robots. On this evidence, it is not possible to reject either $H5.3_0$ or $H5.4_0$.

In comparing the mDCA and SVM-based systems, it can be said that there is no significant difference in either of the performance measures. It is not possible to reject either $H5.5_0$ or $H5.6_0$ and it is accepted that there is no difference in the performance of a system that uses the SVM-based system and a system that uses the mDCA approach.

In comparing the SVM-based approach with the system in which no detection took

place, it is observed that there is a significant difference in both the amount of stored energy and the number of surviving robots present at the end of each experiment. It is therefore possible to reject both $H5.7_0$ and $H5.8_0$ and state that the SVM-based system performs significantly better than a system in which no anomaly detection takes place.

Finally, in comparing the mDCA-based approach with the system in which no detection took place, it can again be stated that there is a significant difference in both the amount of stored energy and the number of surviving robots present at the end of each experiment. Based upon this evidence it is possible to reject both $H5.9_0$ and $H5.10_0$ and accept that the mDCA-based system performs significantly better than a system in which no anomaly detection takes place.

In the latter part of section 5.4.2, results were presented in which the FPR of the SVM and mDCA-based system were varied in order to study what effect this had on the long-term survival of the systems. It was observed that with a low FPR of 0.01 the SVM based approach outperformed the mDCA both in terms of the amount of stored energy and the number of surviving robots (although only the difference between the number of surviving robots was significant). With an FPR of 0.01 the SVM-based approach has a TPR of around 0.85, whilst the mDCA has a TPR of 0.681. It is suggested that the SVM-based approach is able to perform better because its higher TPR allows it to detect more anomalies and prevent adverse effects such as robots colliding with power sockets or other modules. However, when the FPR is increased to 0.1 and the true positives rates of the mDCA and SVM-based approach increase to 0.79 and 0.95 respectively, the performance of the SVM-based approach is worse than that of the mDCA (albeit not significantly so). When the FPR is increased to 0.2, the same is true again and despite having a larger TPR, the SVM-based approach performs worse than the mDCA-based system.

The reason for this surprising result could be related to the same deficiencies in the recovery strategy which meant that the ideal system did not perform as well as the system in which no anomalies were present. To clarify this point, consider the case where a sensor-noise anomaly increases the amount of noise in one of the front two sensors of a robot by 1%. Since 1% is only a small increase, it is likely that the anomaly would be missed by a system with a low TPR, but detected by a system with a larger TPR. Consider now the behaviour of two such systems immediately after the anomaly has been introduced. The system which detected the anomaly would take the value of a neighbouring sensor, whilst the system which did not would continue to read the anomalous value. Because the size of the anomaly is so small, it is possible that the anomalous value is actually closer to the true value than that of the neighbouring sensor. Furthermore, robots utilising the system with the larger TPR, which did detect the small anomaly, would be prevented from recharging or sharing energy. Robots utilising the approach with the lower TPR, meanwhile, would continue to recharge and share energy and given the small size of the anomaly would be unlikely to be adversely effected. This, it is suggested, may be the reason why the SVM based system is outperformed by the mDCA approach when the FPR is increased. By ignoring anomalies which do not adversely affect the behaviour of the robot, less modules are

taken out of the pool of ‘active’ robots—those which may recharge either themselves or others—and so the system as a whole is better equipped to survive.

The same behaviour may also explain why, in the experiments which included anomaly detection, the total stored energy initially follows the same trend as the experiment with no anomaly detection (figure 5.11), in spite of the fact that, as shown in figure 5.13, the number of operational robots decreases at a much lower rate. The suggested reason for this is that, if the robots are able to detect anomalies, then there are guaranteed to be less active robots in the system at any one moment in time. With fewer robots able to recharge or share energy, it is expected that the average proportion of energy stored by each robot will decrease. Once this new average value is approached, however, there is a switch in the dynamics of the system and the total stored energy then decreases in line with the numbers of operational robots.

5.5 Summary and Future Work

In this chapter, an energy foraging and energy sharing strategy was presented which aimed to reduce some of the negative effects that were identified by the reliability study documented in chapter 4. The strategy took inspiration from the morphogenesis controller introduced in chapter 4 and allows robots to dock with power sockets and form ad-hoc multi-robot structures within which modules can harvest or share energy.

An anomaly detection algorithm, inspired by the function of the vertebrate immune system, was also introduced. The algorithm, known as the ‘modified Dendritic Cell Algorithm’ (mDCA), uses sliding windows to extract features from time-series data and classifies the data as anomalous or normal using a linear weighted sum. Weaknesses in the original mDCA implementation were highlighted and suggested improvements were made to both the algorithm itself and the supporting experimental framework. The parameters of two different versions of the mDCA were optimised using the evolutionary multi-objective optimisation strategy, NSGA-II. In the first version, which was designed to emulate the original mDCA approach, the features that were extracted from the data were fixed. In the second version, a wider set of possible features were provided and the ones used by the algorithm were selected by evolution. The two versions of the mDCA were compared in terms of their evolutionary progress and classification accuracy. In both cases, the new version of the mDCA was shown to outperform the original. Comparisons were also made with a system that utilised Support Vector Machines (SVM) to perform classification. The feature extraction process of the SVM-based system was optimised using the same multi-objective approach that was used to select the mDCA parameters. The SVM approach was shown to outperform both versions of the mDCA in terms of classification accuracy, but was slower than the mDCA in terms of run-time speed.

The anomaly detection algorithms were then combined with the energy foraging controller and used to investigate the task of long-term survival. The performance of the combined systems were measured in terms of the number of surviving robots and

the amount of stored energy present at the end of a ten hour simulation. The mDCA and SVM based approaches were shown to perform significantly better than systems in which no detection took place and on a comparable level to a system which utilised an idealised form of anomaly detection. The SVM-based system outperformed the mDCA in terms of the number of the surviving robots and the amount of stored energy, but not significantly so. The benefit of the energy sharing strategy was demonstrated by the fact that all of the systems in which energy sharing took place consistently outperformed the system in which it did not.

In order to investigate what effect increasing the false positive rate would have on the performance of the system, the discrimination thresholds of the SVM and mDCA based approaches were then varied. Intuitively, one might expect that the system with the lowest FPR and greatest TPR would perform best, however, the results revealed that with equal FPRs, the SVM-based approach did not benefit from having a larger TPR, and in fact performed slightly worse than the mDCA. It was suggested that the reason for this behaviour is related to the fact that when a small anomaly is present, the cost of recovery may be greater than the cost of ignoring the anomaly. A system with a lower TPR, which ignores small anomalies, may have an advantage in these scenarios. However, more experimental results would be required in order to confirm this hypothesis.

These observations help to highlight one of the main benefits of a multi-objective approach to optimisation. In the scenario described above, there is an uneven weighting between the desire for a high TPR and a low FPR. Where the desired relative weightings of the objectives are unknown, a multi-objective approach to optimisation can be beneficial in discovering a variety of solutions in which different weightings are applied.

In this chapter, only two objectives were optimised, in future work, a greater number may be investigated, including, for example, the run-time speed or memory requirements of an approach. To investigate scenarios in which the relative weightings of the objectives may change over time, the optimisation of the anomaly detection systems may also be performed in an online manner. In order to further investigate what effect varying the true positive and false positive rates has on the performance of the system, more long-term survival experiments may be performed. Furthermore, in order to help reduce the costs of recovery, new recovery strategies may be developed.

The work in this chapter was focussed on the detection of and recovery from failures which occur whilst robots are operating as individuals. This work highlighted the importance of designing effective and efficient recovery strategies. In the following chapter, the task of recovering from a failure that occurs whilst a robot is part of a larger robotic structure is investigated and a new form of recovery strategy is introduced.

CHAPTER 6

Self-repairing Robotic Structures

In this chapter, a new *self-repair* strategy is presented which aims to improve the fault tolerance of the morphogenesis controller introduced in chapter 4. Following the failure of an individual module within a robotic structure, the strategy ensures that the faulty module is isolated and removed, and that the structure is rebuilt in an efficient manner.

The strategy focuses on failures that occur within robots which are already part of a robotic structure. With relation to the reliability analysis study performed in chapter 4, the strategy aims to tackle the serious effect E_1 : a stall in the formation of a structure. However, as well as targeting partially assembled structures, the strategy may be used to repair structures in which modules fail after the assembly process has completed. Assuming the presence of a fault detection system, such as that introduced in chapter 5, a system which uses this strategy is capable of recovering from all seven of the hazards introduced in chapter 4. Learning from the FTA analysis performed in chapter 4, in which the wired communications channel was identified as a single point of failure, the reliability of the system is further improved by introducing an extra level of redundancy to inter-module communication.

The generality of the approach is highlighted in its implementation on all three of the Symbicator platforms. The performance of the strategy is analysed using two different simulators, as well as physical robotic hardware. In simulation, the performance of the system is compared with a naive strategy that responds to the detection of a failure by disassembling and restarting the entire assembly process from scratch, and with a perfect system in which no failures occur. The new self-repair strategy is shown to outperform the naive approach in the majority of cases and performs particularly well on configurations that contain many repeated segments. Using real robots, as a proof of principle, the behaviour of the system is analysed in four typical scenarios.

In section 6.1, the self-repair strategy itself is introduced, the main assumptions made during its design are presented, and comparisons are made to other similar techniques. In section 6.2, the specific implementation of the strategy on-board the Symbicator robots is described and its limitations are discussed. In sections 6.3-6.5 results from both simulated and physical robot experiments are provided. Finally, in section 6.6, a summary is provided and potential areas of future work are discussed.

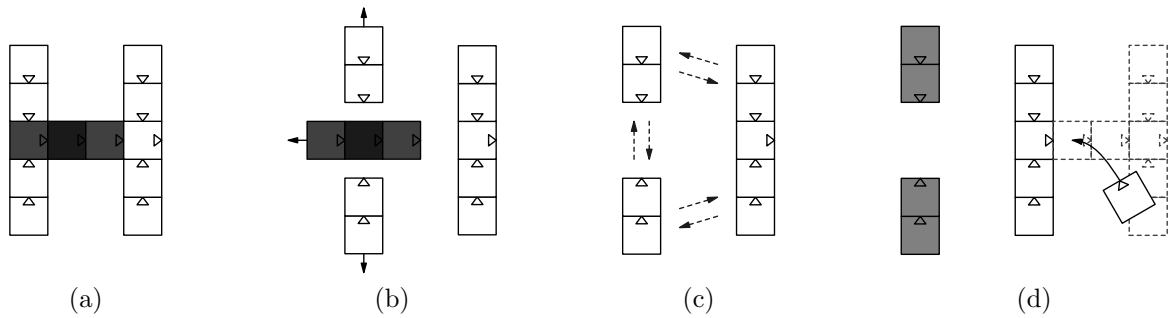


Figure 6.1: The four stages of the self-repair strategy. The failed module is shown as a dark grey square, the support modules are represented by the light grey squares and the repair modules are represented by the remaining white squares

6.1 Strategy

The strategy described in this chapter can be used to repair an assembled or partially assembled robotic structure which contains a single failed module. The strategy works in the same way, regardless of the state of assembly. The strategy begins by isolating the failed individual and dividing the structure into separate sub-structures. Once the structure has been divided, the failed robot can be removed, either by itself, or if the robot is immobile or unaware that it has failed, by neighbouring support modules. Each of the individual sub-structures then assigns itself a score, based upon knowledge of its own morphology and that of the desired target shape. By broadcasting and listening for messages, the separate structures then negotiate with one another which is in the best position to rebuild the target structure. Following negotiation, the ‘winning’ sub-structure restarts the assembly process, whilst the other sub-structures disassemble.

During the self-repair process, every module may be considered to take on one of three roles: *failed module*, *support module* or *repair module*. The failed module is that which is responsible for initiating the self-repair strategy. The support modules are those which neighbour the failed module and are responsible for removing it from the structure. The remaining robots are repair modules, whose job it is to determine how to re-build the structure in the most efficient manner. Note that, provided the failed robot is capable of undocking and removing itself from the structure, and that it is capable of communicating its intentions to the remainder of the structure, support modules may not always be required. If support modules are not required, the neighbours of a failed module would also belong to the group of repair modules.

As visualised in figure 6.1, the strategy can be split into four stages. In the first stage, shown in figure 6.1a, a failure is detected by either the failed module or its neighbouring support modules. Messages are then propagated to the remaining modules so that the entire structure is aware that the recovery process has been initiated.

As shown by figure 6.1b, in the second stage, the structure is split into several sub-structures, allowing the support modules to move the failed module away from the

group. For each sub-structure, this retreating phase is led by the repair modules which neighbour with support modules, assuming the structure does not contain any loops, there will be one such module in every sub-structure.

In stage three, using local message passing, the repair modules establish the shape of the sub-structure that they belong to. This shape is compared to a known target and used to calculate the ‘score’ of each sub-structure. The details of how this score is calculated are implementation dependent but must, in some way, represent how similar the sub-structure is to the desired target shape.

As shown in figure 6.1c, every module in each sub-structure then broadcasts its score and a unique ID, whilst listening for messages from the other sub-structures. If a sub-structure detects a score higher than its own, it notifies its neighbours and begins to broadcast the higher score instead. To resolve conflicts, if a sub-structure detects a score that is the same as its own, it broadcasts the score with the lower associated ID. Eventually, a consensus will be reached and all sub-structures will be broadcasting and receiving the same messages.

In the final stage, as shown in 6.1d, the sub-structure which did not detect any scores higher than its own assumes itself to be in the best position to continue assembly, and declares itself as the winner. The other sub-structures are simply disassembled. After removing modules which do not belong to the target shape, the winning sub-structure may continue to assemble the target. Note from figure 6.1d, that a modules role within the new structure need not be the same as its role in the original structure.

6.1.1 Assumptions

This section summarises the main assumptions made during the development of the self-repair strategy. Where appropriate, the assumptions are justified and details of the consequences of removing them are outlined.

Structures will contain at most one failed module At any single moment in time, a structure will not contain more than one failed module. This is the most severe limitation of the self-repair strategy and is especially problematic when considering systems which contain a very large number of individual modules. In large systems, there is more chance that multiple robots will fail at the same time, reducing the validity of this assumption. However, in small to medium sized systems, where there is less chance of multiple faults occurring simultaneously, the assumption more reasonable. In many cases, even if multiple failures did manifest themselves within the same structure, the strategy would still be able to cope. The strategy would respond to multiple failures by first initiating self-repair to resolve the first failure, and upon completion, re-initiating the process to resolve the second failure. For the approach to be successful, the second failed module must possess sufficient capabilities to allow the first repair process to be completed successfully, otherwise the approach will fail. It is also important to highlight that the system as a whole may still recover from the occurrence of parallel failures, but only if they occur within separate structures.

Robots will not suffer control system failures The system is only designed to recover from (detectable) hardware failures. The failure of a robot's controller may result in undesired behaviour from which the system cannot recover. For example, a robot which incorrectly transitions from the Disassembling state to the Supporting state could severely disrupt the disassembly process and place the system in a state from which it cannot repair itself.

Modules are capable of omnidirectional locomotion All robots are capable of some form of omnidirectional locomotion. Assuming that every robot can move in every direction simplifies the description of the self-repair strategy. Given that two of the three Symbricator robots are capable of omnidirectional locomotion, this is not an unreasonable assumption. However, in order to demonstrate the generality of the approach, in section 6.2, an implementation of the strategy is presented in which this assumption is removed.

All robots have some knowledge of a target shape Every robot has some knowledge of a target shape. This knowledge need not be 100% accurate and different robots may, in fact, have vastly different target shapes. It is only necessary that each module has some way of comparing the shape of the sub-structure to which it belongs, with some target, and in doing so is able to assign itself a score which will accurately rank the potential of the sub-structure to which it belongs, relative to the other sub-structures. How these scores are calculated is considered to be implementation dependent, in section 6.2, one possible mechanism used in experiments involving the Symbricator robots is presented.

All robots are capable of detecting failures Every robot possesses some mechanism for detecting failures within itself, or within neighbouring robots. This assumption is justified by the existence of the fault detection methods reviewed in chapter 2 and the anomaly detection algorithm presented in chapter 5. The self-repair strategy is only triggered after the discovery of a failure. Therefore, without a fault detection system, the strategy would be ineffective.

There is a sufficient number of spare robots Following the removal of a failed module, there are enough spare robots present within the vicinity of the structure in order to rebuild the target shape. The nature of self-reconfigurable modular robotic systems and the fact that they are designed to contain many redundant modules, justifies this assumption. This is also an assumption of the original morphogenesis controller, if there is not a sufficient number of spare robots, in its current form, the assembly process will stall. In future work, a minor alteration could resolve this issue by causing the assembly process to timeout if no new robots had joined after a certain period.

There is a mechanism for transforming a 3D structure to 2D The system possesses a mechanism for transforming a 3D structure into a planar 2D structure, from which the self-repair process can be initiated. Without first transitioning

into a 2D configuration, the self-repair strategy would not work, but precisely how this transformation is achieved is considered to be beyond the scope of this work. This assumption highlights a potential limitation of the approach in that, if a module fails whilst forming part of an assembled 3D structure, and cannot revert to its 2D state, the self-repair strategy would be unlikely succeed.

Robots can only dock as individuals Multi-robot structures are not capable of autonomous docking, only individual modules can dock. Allowing multi-robot structures to dock with one another could greatly improve the speed and efficiency of self-repair. However, from experience with the Symbricator robots, even single module docking was found to be a very difficult task. The perceived difficulty of implementing multi-robot docking justifies this assumption.

Structures cannot contain loops There is only one unique path between every pair of modules, which is to say, structures can not contain loops. This assumption is imposed by the original morphogenesis controller which represents configurations as DAGs, and therefore cannot contain cycles. It is further imposed by the Symbricator hardware which, due to the networking design, does not currently support loops. Removing this assumption would require changes to be made to both the original morphogenesis controller and the self-repair strategy.

6.1.2 Controller

This section details how the self-repair strategy is integrated with the morphogenesis controller from chapter 4. Figure 6.2 shows an updated version of the finite state machine from figure 4.3. The original states and transitions from figure 4.3 are shown in grey, whilst the new ones are displayed in black. For clarity, the state occupied by failed modules is omitted from this diagram.

The controller requires that robots have at least two channels of communication. One that provides direct communication between members of the same structure, and one that provides wireless communication between members of separate structures. The various different types of message that modules send during self-repair, and the data that they carry, are summarised in table 6.1.

In the remainder of this section, each of the new states and transitions in figure 6.2, as well as the ‘Failed’ state, are described in detail. Throughout this section, a running example is used to demonstrate the process of self-repair. Figure 6.3 shows the example structure that is used, alongside its corresponding graph and string representations.

Failed

A module enters the Failed state either when it detects a failure within itself, or when a neighbour detects a failure within it. If a failure is detected by a neighbouring robot, it informs the failed module by sending an M_{KILL} message. However, if because of the failure, it is not possible for the robots to communicate this information to the failed

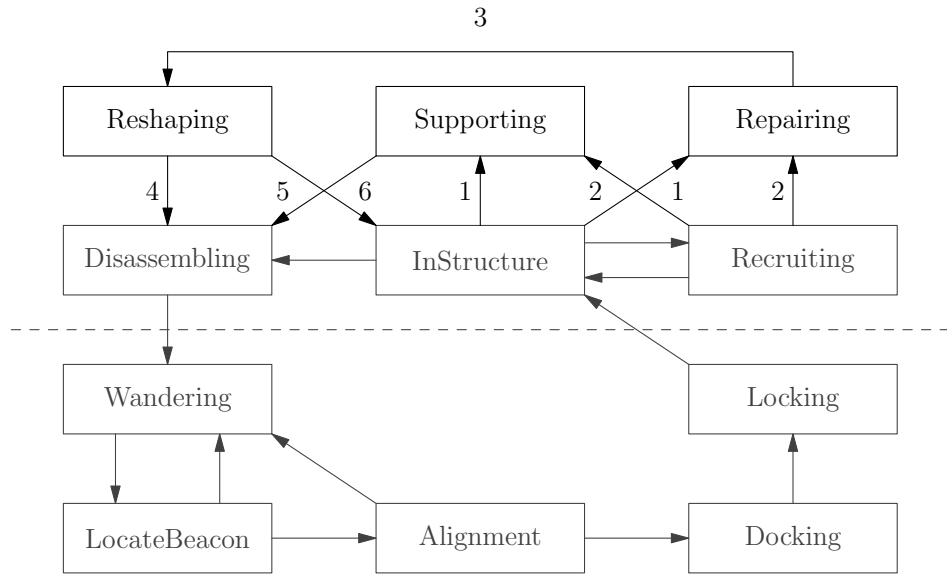


Figure 6.2: A finite state machine for the morphogenesis controller from chapter 4, extended to include the self-repair strategy described in this chapter

Message	Description
M_{FAILED}	Sent by modules in the Failed or Supporting states to indicate the presence of a failure and notify whether the sending module requires assistance in removing itself from the structure
M_{KILL}	Sent by a module which detects a failure within its neighbour to inform the neighbour that it has failed
M_{SHAPE}	Sent by modules in the Repairing state to determine the shape of a sub-structure. Every message sent includes a (possibly incomplete) string-based representation of a structural configuration
M_{SCORE}	Sent by modules in the Repairing state to discover the highest scoring sub-structure. Every message sent includes a score and an ID
$M_{RETREAT}$	Sent by the lead repair module to inform others to start retreating
M_{STOP}	Sent by the lead repair module to inform others to stop retreating
$M_{RESHAPE}$	Sent by the lead repair module to inform others to enter Reshaping
$M_{DISASSEM}$	Sent by Reshaping modules to inform others to enter Disassembling
M_{BRANCH}	Sent by modules in the Reshaping state to provide information about the new structure. Every message sent includes a string-based representation of one segment of the new structure

Table 6.1: The different types of message sent between robots during self-repair

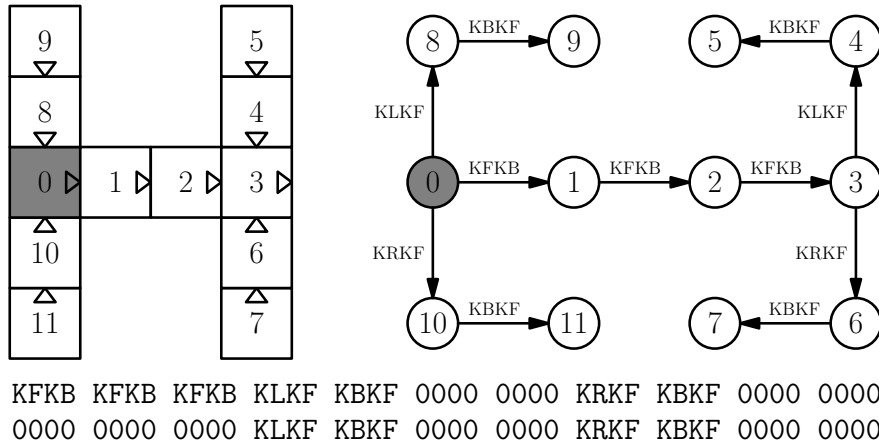


Figure 6.3: An example structural configuration alongside its graph representation, with the corresponding string representation shown below. The seed module is coloured grey

module, the module will not enter the Failed state and instead will remain ignorant of the failure. A module that is not aware that it has failed can still be safely removed from a structure by support modules, however, these modules must then sacrifice themselves in order to prevent the failed module from interfering at a later stage.

In theory, a module may enter the Failed state from any of the other states in figure 6.2. However, based upon the assumption from section 6.1.1 that a structure will contain no more than one failed module, the transitions from each of the three new self-repair states, in figure 6.2, can be ignored.

If a module detects a failure within itself whilst it is not part of a robotic structure, that is, whilst it is occupying any state below the dashed line in figure 6.2, it responds by transitioning immediately to the Failed state. Once in the Failed state, the module may either immobilise itself, or move to a specialised repair zone.

If a module detects a failure within itself, or is notified that it has failed, whilst it is part of a larger robotic structure, it responds by sending an M_{FAILED} message to each of its neighbours and entering the Failed state. The M_{FAILED} message includes details of whether the module needs assistance in removing itself from the structure. If it is able to remove itself from the structure, it unlocks any docking elements and waits until there is room for it to leave the area. Otherwise, it simply waits to be removed by support modules.

Supporting

A module will enter the Supporting state in two scenarios. Firstly, if it receives an M_{FAILED} message in which it is specified that help is required, and secondly, if it detects a failure within a neighbouring module and, after notifying the module, no M_{FAILED} response is received (figure 6.2, transition 1). Immediately after entering the

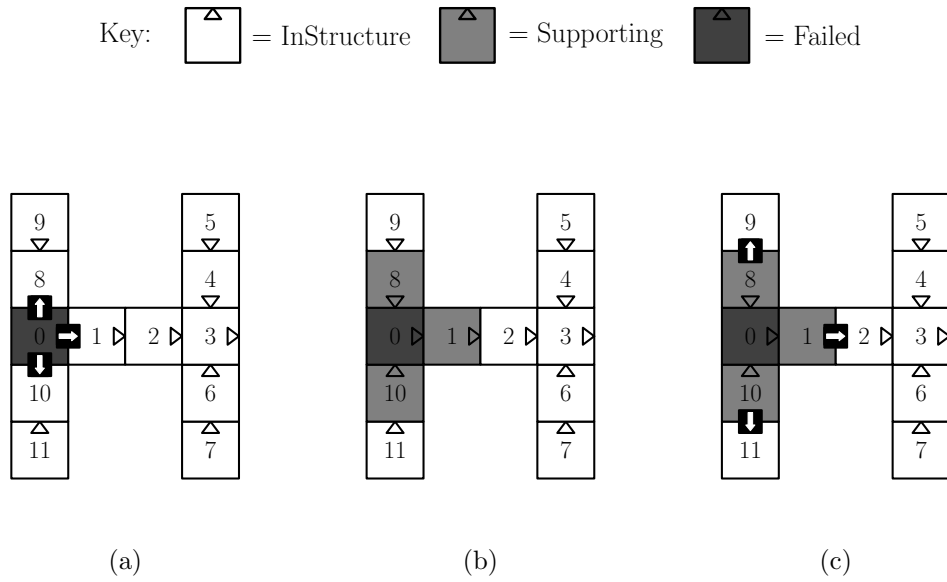


Figure 6.4: An example of modules entering the Supporting state. The shade of each module represents the state that it is currently occupying. The white arrows within black boxes represent the passing of M_{FAILED} messages between neighbouring modules

Supporting state, modules send an M_{FAILED} message to every other neighbour (apart from the failed one). To ensure that their neighbours do not also become support modules, in this new M_{FAILED} message, they specify that no assistance is required.

For example, as shown in figure 6.4, if module 0 was to fail, and required help in removing itself from the structure, it would send an M_{FAILED} message to modules 1, 8 and 10, requesting their assistance (a). Modules 1, 8 and 10, which previously occupied the InStructure state, would then enter the Supporting state (b) and send another M_{FAILED} message to modules 2, 9 and 11 (c), this time not requesting help.

Support modules are responsible for removing the failed module from a structure. To do so, they first wait until the structure has been split into sub-structures. They then transport the module to a safe region and either disassemble (figure 6.2, transition 5), if it is possible and safe to do so, or remain connected to the failed module in order to neutralise any further problems that the module may cause.

The most difficult part of this behaviour is coordinating the motion of the support modules. Precisely how this is done is implementation dependent, and relies upon the type of sensors, actuators and communication channels available on the platform. If the failed module is capable of communicating with the support modules, the task is equivalent to the 2D collective locomotion problem, which has been solved by many self-reconfigurable modular robotic systems [7, 130, 181]. If the failed module is not able to communicate, the task is equivalent to the related problem of collective transport, which again has been solved on numerous occasions [16, 45, 63].

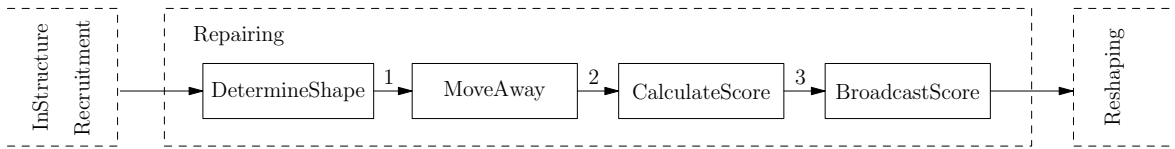


Figure 6.5: A finite state machine for the Repairing behaviour. Modules enter the Repairing state from either the InStructure or the Recruitment state and, after transitioning through the four sub-states, exit by switching to the Reshaping behaviour

Repairing

The majority of the self-repair process is performed by modules in the Repairing state. Every member of a sub-structure enters the Repairing state at some point in time. The first module of a sub-structure to enter the state is assigned as the ‘lead’ of that structure and is responsible for coordinating the behaviour of the other modules. Each sub-structure may only contain a single lead module. Modules will enter the state either when they receive an M_{FAILED} message or an M_{SHAPE} message (figure 6.2, transition 2). If a module receives an M_{FAILED} message, which does not specify that the sender requires help removing itself from the structure, the receiver enters the Repairing state and assigns itself as the lead of a new sub-structure. If a module receives an M_{SHAPE} message, and is not already in the Repairing state, it enters the Repairing state, but does not assign itself as the lead module.

As shown in figure 6.5, it is convenient to consider the Repairing state as a smaller, four step, finite state machine. The first state (DetermineShape) is used by the modules to discover the configuration of the sub-structure to which they belong. In the second state (MoveAway), the modules coordinate their motion in order to move their sub-structure away from the failed module. In the third state (CalculateScore), the modules determine the score of their sub-structure. Finally, in the fourth state (BroadcastScore), the modules determine whether or not they belong to the winning sub-structure.

In the remainder of this section, algorithms 1-3 are used to describe the DetermineShape, MoveAway and BroadcastScore states. Each algorithm contains three procedures, one that is executed immediately before a module enters the state, one that is executed once for every timestep that a module remains in the state, and one that is executed after a module leaves the state. An overview of the CalculateScore state is also provided, but since this behaviour will differ depending upon how the sub-structure scores are calculated, full details are omitted from this section. In section 6.2, a potential implementation of the CalculateScore state is described.

DetermineShape This state serves as the entry point for the Repairing behaviour and is used by modules to determine the shape of the sub-structure to which they belong. The behaviour of robots in this state is outlined by algorithm 1.

Starting with the lead module, the robots effectively perform a distributed depth-first traversal of their sub-structure, in the process constructing a string representation

of the sub-structure's graph. M_{SHAPE} messages are used to pass a *Shape* string between neighbouring modules. Every time a message is sent, one or both of the sending and receiving modules will add relevant symbols to the sequence. Which symbols are added depends upon the type of the modules involved, and the sides at which the messages are sent or received.

Returning to the example from figure 6.4, when modules 2, 9 and 11 receive the M_{FAILED} messages sent by modules 1, 8 and 10, they enter the Repairing state and assign themselves as lead modules. There are now three sub-structure, one led by each of modules 2, 9 and 11. Considering the sub-structure led by module 2, figures 6.6 and 6.7, along with algorithm 1, help to show how the shape of this structure is determined.

As shown in algorithm 1, on lines 2-3, upon entering the Repairing state, modules initialise a reference to the robot which caused them to enter the state (their *Parent*) and populate a queue containing references to all of their other neighbours (*DockedQ*). The lead module then creates an empty *Shape* string (line 5) which will subsequently be passed between every module in the sub-structure and updated accordingly until the shape of the structure has been determined.

From the example in figure 6.6, robot 2 will populate a queue containing its only other neighbour, module 3. After initialising the *Shape* variable, robot 2 will append the symbol representing its type 'K' and the symbol representing the side at which its first neighbour (module 3) is docked 'F' (algorithm 1, line 11). As shown in figure 6.6a, the *Shape* string is then sent to module 3 within an M_{SHAPE} message (algorithm 1, line 12). Upon receipt of this message, module 3 will enter the Repairing state and append its own type 'K' and the side at which the message was received 'B' to the *Shape* string (algorithm 1, line 8). As shown on the first line in the left of figure 6.7, after the first message has been sent, the *Shape* string will contain the symbols: 'KFKB'. Module 3 will then populate its queue with references to its two neighbours, modules 4 and 6, before sending an M_{SHAPE} message containing the symbols 'KL' to its first neighbour, module 4 (figure 6.6b). At this point, modules 2 and 3 have entered the main DETERMINESHAPE procedure of algorithm 1 (line 15), and will continue to executed this procedure as long as their queue of neighbours is not empty.

The passing of the *Shape* string continues and as more modules enter the Repairing state (figure 6.6), more symbols are added to the *Shape* string (figure 6.7). As shown in figure 6.6c, eventually the *Shape* string reaches module 5, but since this module has no neighbours other than its *Parent*, its queue will be empty and it will transition immediately to the next sub-state of the Repairing behaviour, MoveAway (algorithm 1, lines 16 and 27). However, on exiting, module 5 will first append four NULL symbols to the *Shape* string (algorithm 1, line 32) and send a message back to its *Parent*, module 4 (figure 6.6d). Upon receiving this message (algorithm 1, line 17), module 4 will remove module 5 from the front of its queue, update the *Shape* string and append the relevant symbols (algorithm 1, line 18-20). Since the queue of module 4 is now empty, it will also transition to the MoveAway state and send the *Shape* string back to module 3 (figure 6.6e). Module 3 will process the message in the same way, but since, even after removing module 4, its queue will not be empty (algorithm 1, line 21), module 3 will

Algorithm 1 Procedure for determining the shape of a sub-structure




```

1: procedure ONENTRY
2:    $Parent \leftarrow$  the module which caused this robot to enter the Repairing state
3:    $DockedQ \leftarrow$  all of the robots docked with this module (excluding  $Parent$ )
4:   if this is the lead module then
5:      $Shape \leftarrow$  empty string
6:   else
7:      $Shape \leftarrow$   $Shape$  received from  $Parent$ 
8:      $Shape.APPEND$ (own ‘type’ + ‘side’ at which  $Parent$  docked)
9:   end if
10:  if not  $DockedQ.EMPTY$  then
11:     $Shape.APPEND$ (own ‘type’ + ‘side’ at which  $DockedQ.FRONT$  docked)
12:    send  $M_{SHAPE}$  message containing  $Shape$  to  $DockedQ.FRONT$ 
13:  end if
14: end procedure

15: procedure DETERMINESHAPE
16:  if not  $DockedQ.EMPTY$  then
17:    if  $M_{SHAPE}$  message received from  $DockedQ.FRONT$  then
18:       $DockedQ.DEQUE$ 
19:       $Shape \leftarrow$   $Shape$  received in  $M_{SHAPE}$  message
20:       $Shape.APPEND$ (own ‘type’ + ‘side’ at which  $M_{SHAPE}$  message received)
21:      if not  $DockedQ.EMPTY$  then
22:         $Shape.APPEND$ (own ‘type’ + ‘side’ at which  $DockedQ.FRONT$  docked)
23:        send  $M_{SHAPE}$  message containing  $Shape$  to  $DockedQ.FRONT$ 
24:      end if
25:    end if
26:  else
27:    transition to MoveAway
28:  end if
29: end procedure

30: procedure ONEXIT
31:  if not the lead module then
32:     $Shape.APPEND$ (‘0000’)
33:    send  $M_{SHAPE}$  message containing  $Shape$  to  $Parent$ 
34:  end if
35: end procedure

```

Key:  = InStructure  = MoveAway  = DetermineShape

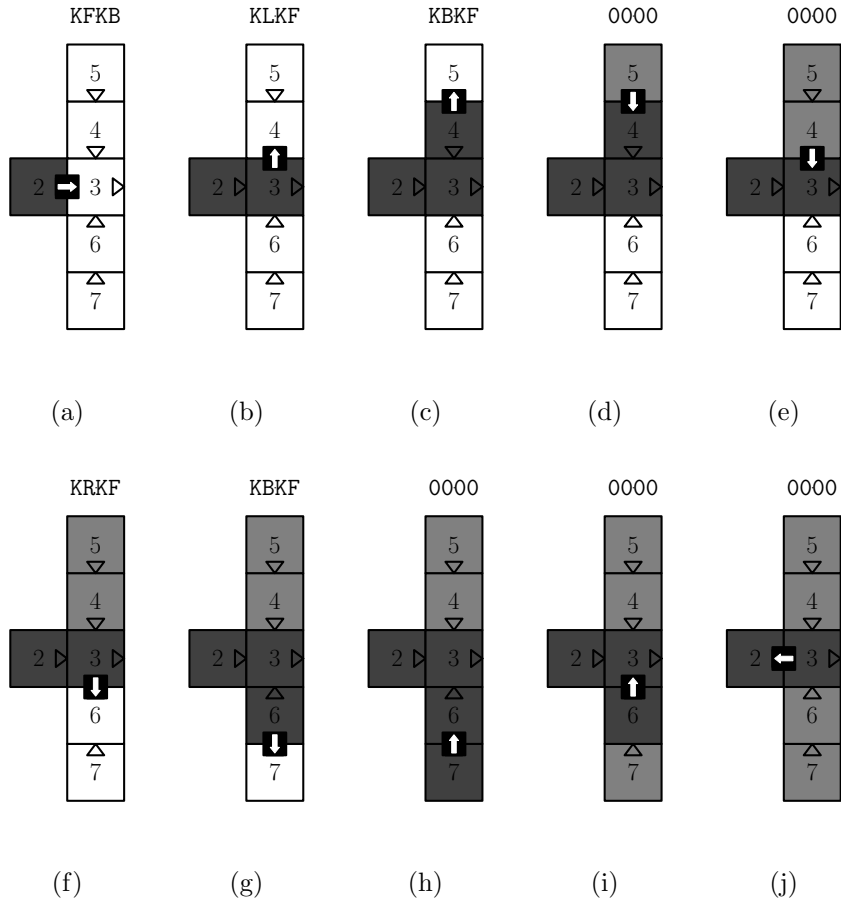


Figure 6.6: The sequence of messages sent between modules in the DetermineShape state. The shade of each module represents the states that it is currently occupying. The white arrows within black boxes represent messages being sent between neighbouring modules. The four symbol sequence above each figure represents the part of the sub-structure string that is created when the corresponding message is passed

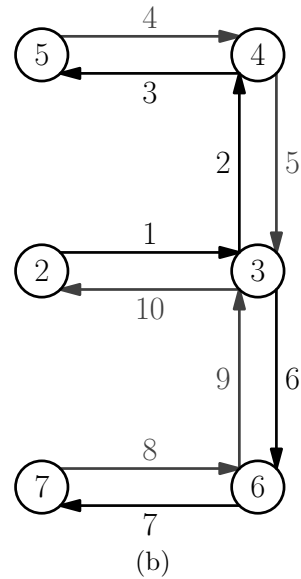
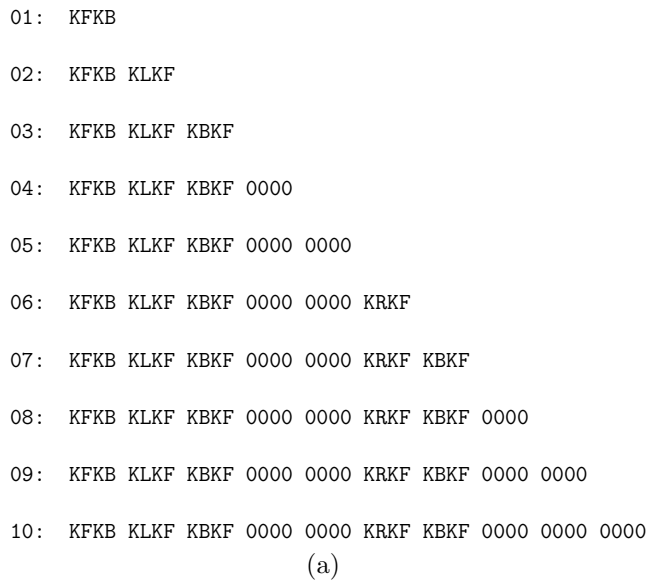


Figure 6.7: The construction of the sub-structure string (left) with the order in which messages are sent superimposed upon the sub-structure graph (right). The messages which trigger a new module to enter the Repairing state and contain module information are shown in black, whilst the messages that contain NULL symbols are shown in grey

send the *Shape* string to the new module at the front of its queue, module 6 (algorithm 1, line 23). This process will continue (figure 6.6f-j) until only module 2 remains in the DetermineShape state, at which point, the *Shape* string will be complete (figure 6.7, line 10) and module 2 will enter the MoveAway state.

MoveAway This state is used to move sub-structures away from failed modules. The behaviour of robots in this state is outlined by algorithm 2. In this description, the modules simply head in the opposite direction to the failed module for a pre-set period of time. In certain scenarios, more complex controllers may be required in order to ensure that sub-structures do not collide with one another, or with other obstacles.

Algorithm 2 Procedure for moving a sub-structure away from a failed module

```

1: procedure ONENTRY
2:   Heading  $\leftarrow$  the direction this module should move
3:   if this is the lead module then
4:     send  $M_{RETREAT}$  message to all other robots in sub-structure
5:   end if
6: end procedure

7: procedure MOVEAWAY
8:   if timeout or  $M_{STOP}$  message received then
9:     stop moving
10:    transition to CalculateScore
11:  else if this is the lead module or  $M_{RETREAT}$  message received then
12:    move towards Heading
13:  end if
14: end procedure

15: procedure ONEXIT
16:  if this is the lead module then
17:    send  $M_{STOP}$  message to all other robots in the sub-structure
18:  end if
19: end procedure

```

Before moving, each module must be aware of the direction in which it should head. Although globally, members of the same sub-structure must head in the same direction, relative to their own orientation, it is likely that different modules will need to move in different directions. The desired (local) heading of each individual is stored in a *Heading* variable. Upon entering the Repairing state, the lead module sets its *Heading* to be the opposite of the side at which its *Parent* is docked. Since the lead module is aware of its own *Heading*, and of the orientation of its neighbours, as shown

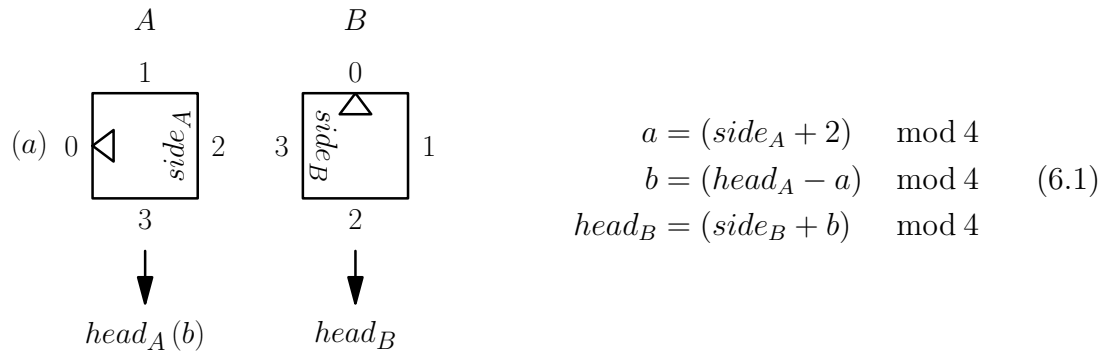


Figure 6.8: Given the *Heading* of robot A ($head_A = 3$) and the sides at which robots A and B are docked with one another ($side_A = 2$ and $side_B = 3$), this figure shows how, using equation 6.1, robot A may calculate the *Heading* of robot B ($head_B = 2$)

in figure 6.8, it is able to calculate its neighbours' headings using equation 6.1. As soon as the other robots have been informed of their own *Heading*, they too can calculate their neighbours' headings. Although omitted for clarity from algorithm 1, the most convenient time for robots to transmit *Heading* values to their neighbours is when they first send an M_{SHAPE} message to them.

Once modules are aware of the direction in which they are required to move, the MoveAway procedure is simple. Upon entering the MoveAway state, the lead module sends an $M_{RETREAT}$ message to all other robots in the sub-structure (algorithm 2, line 4). When this message is received, every other module in the MoveAway state starts to move towards their *Heading* (algorithm 2, line 12). After a pre-set period of time, the lead module stops moving and exits the MoveAway state, at the same time it sends an M_{STOP} message to the other robots (algorithm 2 line 17). Upon receipt of this message, the other robots also stop moving and leave the state (algorithm 2, lines 8-10).

CalculateScore This state is used to calculate the score of each sub-structure. The calculation can be performed in a number of different ways but there are two requirements that must be met. Firstly, each individual module must calculate a personal score which rates how well this module believes it would serve as the seed of a new structure. Secondly, the lead of each sub-structure must be aware of the highest score calculated by any of the other members of the sub-structure. In section 6.2, one potential implementation of this state is provided for use with the Symbicator robots.

BroadcastScore This state is used to determine which sub-structure contains the module that is in the best position to restart the assembly process. The behaviour of robots in this state is outlined by algorithm 3. Upon entering this state, each individual module initialises the two Boolean variables *BestModule* and *BestStructure* to *false* (algorithm 3, lines 2-3). The *BestModule* variable signifies whether or not this module

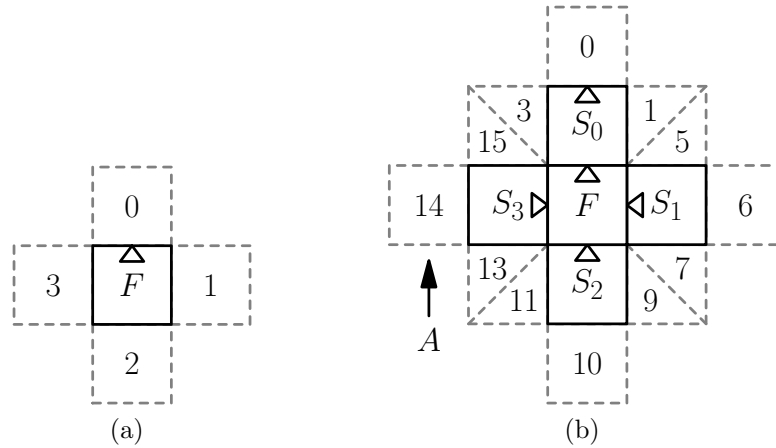


Figure 6.9: How the *StructureID* may be determined in two different scenarios, one that does not involve support modules (a) and one which does (b). Failed modules are labelled F and support modules are labelled S_n . The grey boxes show potential locations at which the lead module of a sub-structure may be docked and the values within these boxes show the *StructureID* for that location

is in the best position to continue assembly, whilst the *BestStructure* variable signifies whether or not this module is part of the sub-structure which contains the *BestModule*. The *StructureID* variable is then initialised to the unique ID of the structure to which this module belongs, and the *ModuleScore* variable is initialised to the score calculated during the CalculateScore state (algorithm 3, lines 4-5).

The ID of a sub-structure can be calculated in a number of ways. If every module in the system has a unique ID, the *StructureID* may simply take the ID of the lead module. However, if unique IDs cannot be guaranteed, then IDs can be generated according to the sides at which modules in the Repairing and Supporting states are docked with the failed module. As shown in figure 6.9a, if there are no supporting modules, there will be a maximum of four sub-structures (one docked on each side of the failed module) and IDs can be generated based upon the side at which each lead module is docked with the failed module. If supporting modules are present, as shown in 6.9b, there are 16 positions in which sub-structures may form (one docked on each of the three available sides of the four supporting modules). The scores for the sub-structure positions in figure 6.9b may be generated by multiplying the side at which the failed module is docked to the supporting module by four, and adding the side at which the lead module is docked with the supporting module. For example, at position A in figure 6.9b, if there is a lead module docked with side 2 of the support module S_3 , which in turn is docked with side 3 of the failed module F , the *StructureID* would be calculated as $3 \times 4 + 2 = 14$. As with the *Heading* variable used in algorithm 2, the most convenient time for the *StructureID* value to be transmitted is when a module first enters the Repairing state.

Algorithm 3 Procedure for broadcasting and listening for sub-structure scores

```

1: procedure ONENTRY
2:    $BestModule \leftarrow false$ 
3:    $BestStructure \leftarrow false$ 
4:    $StructureID \leftarrow$  the ID of this sub-structure
5:    $ModuleScore \leftarrow$  the score of this module
6:   if this is the lead module then
7:      $BestID \leftarrow StructureID$ 
8:      $BestScore \leftarrow$  the best score of any module in the sub-structure
9:     send  $BestScore$  and  $BestID$  to all other robots in the sub-structure
10:  end if
11: end procedure

12: procedure BROADCASTSCORE
13:  broadcast  $BestID$  and  $BestScore$  in  $M_{SCORE}$  message
14:  if new  $score$  and  $id$  received in  $M_{SCORE}$  message then
15:    if  $score > BestScore$  or ( $score == BestScore$  and  $id < BestID$ ) then
16:       $BestID \leftarrow id$ 
17:       $BestScore \leftarrow score$ 
18:    end if
19:    if message was received from another sub-structure then
20:      send  $score$  and  $id$  to all other robots in this sub-structure
21:    end if
22:  end if
23:  if timeout or  $M_{RESHAPE}$  message received then
24:    transition to Reshaping
25:  end if
26: end procedure

27: procedure ONEXIT
28:  if  $BestID == StructureID$  then
29:     $BestStructure \leftarrow true$ 
30:    if  $BestScore == ModuleScore$  then
31:       $BestModule \leftarrow true$ 
32:    end if
33:  end if
34:  if this is the lead module then
35:    send  $M_{RESHAPE}$  message to all other robots in the sub-structure
36:  end if
37: end procedure

```

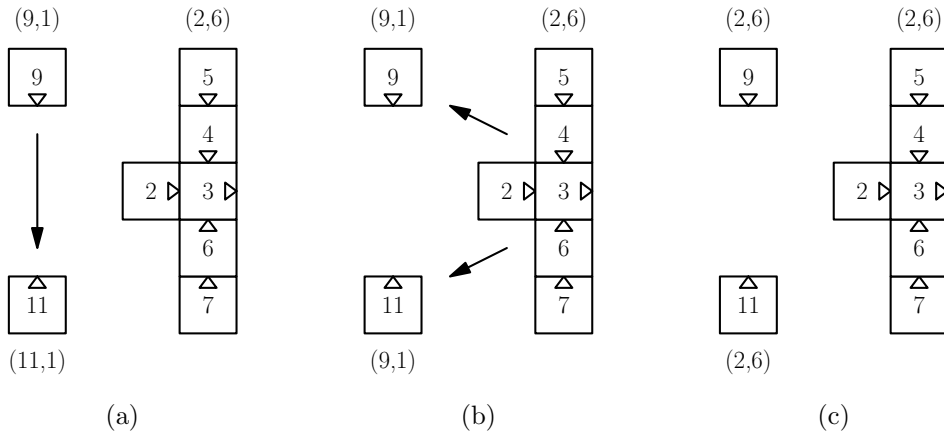


Figure 6.10: How a consensus is reached between three separate sub-structures. The numbers in brackets (x,y) show the *BestID* (x) and *BestScore* (y) values of each sub-structure and the black arrows represent the broadcasting of these values. In the first step each sub-structure has different values (a), but as more messages are broadcast (b) a consensus is eventually reached (c)

In the running example from figure 6.3, it was said that after the failure of module 0, the structure would be split into three sub-structures, led by modules 2, 9 and 11. Let it be assumed that the best score within the sub-structure led by module 2 is ‘6’, and that the best score within the sub-structures led by modules 9 and 11 is ‘1’. Furthermore, as shown in figure 6.10, let the *StructureID* of each of these sub-structures correspond to the IDs of their lead modules: ‘2’, ‘9’ and ‘11’. Upon entering the BroadcastScore state, the lead modules initialise a *BestID* variable to the value of their *StructureID*, and a *BestScore* variable to the value of the best score of any module in the sub-structure (algorithm 3, lines 7-8). To ensure that all robots have the same information, these values are then sent to every other module in the sub-structure (algorithm 3, line 9).

For as long as modules remain in the BroadcastScore state, they will broadcast their *BestID* and *BestScore* values (algorithm 3, line 13) and listen for the values from other modules (algorithm 3, line 14). If a module receives a new *score* that is greater than the module’s own *BestScore*, or the scores are equal but the new *id* is less than the current best, the *BestID* and *BestScore* variables are updated (algorithm 3, lines 14-17). For example, as shown in figure 6.10a, if module 11 detects a message broadcast by module 9, since the scores of both modules are equal, but module 9 has a lower ID, module 11 will update its *BestID* and *BestScore* variables and start to broadcast the same values as module 9. At some time later, if modules 9 and 11 detect the messages broadcast by the sub-structure led by module 2 (figure 6.10b), since the *BestScore* of modules 9 and 11 is lower, they will update their *BestID* and *BestScore* variables. Eventually, a consensus will be reached and every module will be broadcasting and receiving the same *score* and *id* values (figure 6.10c). Depending upon the method of communication

used, it is possible that some broadcast messages will not be received by every module. To ensure that all modules have the most up-to-date information, every time new *score* and *id* values are received from a robot within a different sub-structure, they may be forwarded to the other members of the structure (algorithm 3, lines 19-20).

Modules continue to broadcast and receive M_{SCORE} messages for a pre-set period of time. The duration of this period must be chosen carefully to ensure that a consensus has been reached before moving on to the next state. In section 6.2, one potential method of determining the broadcasting period is described. After the broadcast period has passed, the lead module will transition to the Reshaping state (algorithm 3, lines 23-24) and send an $M_{RESHAPE}$ message to the other members of the sub-structure (algorithm 3, line 35), causing the other modules to leave the state. Upon exiting the state, each module compares its *BestID* and *StructureID* values to determine whether it belongs to the winning sub-structure (algorithm 3, lines 28-29), and if so, compares its *BestScore* and *ModuleScore* values to determine whether or not it is the module that is best positioned to restart the assembly process (algorithm 3, lines 30-31).

Reshaping

The Reshaping behaviour is generic and is not intrinsically tied to the self-repair strategy. Given an existing structure, containing a single *Seed*, the purpose of the Reshaping behaviour is to initialise the transformation of the structure into a new *Target* shape. The process begins with the *Seed* module checking each of its neighbours and instructing them either to enter the Disassembling state (figure 6.2, transition 4) if they do not form part of the *Target* structure, or the InStructure state if they do (figure 6.2, transition 5). Every module that does not enter the Disassembling state then performs the same action on each of its own neighbours, until the assembly process is restarted.

When used during self-repair, modules enter the Reshaping state after the broadcasting period of the BroadcastScore state has ended, or after having received an $M_{RESHAPE}$ message from the lead of their sub-structure (figure 6.2, transition 3). As shown in lines 29 and 31 of algorithm 3, before modules leave the Repairing state, they first set their *BestStructure* and *BestModule* flags to signify respectively whether they are part of the winning sub-structure and whether they are in the best position to restart the assembly process. These flags are used in algorithm 4 to determine the behaviour of robots in the Reshaping state.

In the first part of the Reshaping behaviour, the *BestModule* is assigned as the *Seed* of the new structure (algorithm 4, line 2). The *Seed* then splits its *Target* shape into separate branches, one for each of the outgoing edges of its corresponding graph (algorithm 4, line 4). For each side of the module, the *Seed* then checks whether there is a neighbour currently docked, and if there is, whether it is of the correct type and orientation (algorithm 4, line 6-7). If there is a neighbour docked, but there shouldn't be, or if the neighbour is of the wrong type or orientation, an $M_{DISASSEM}$ message is sent on the corresponding side (algorithm 4, lines 10 and 13). If a module in the Reshaping state receives an $M_{DISASSEM}$ message it will propagate the message to all of

Algorithm 4 The Reshaping behaviour

```

1: procedure RESHAPING
2:    $Seed \leftarrow BestModule$ 
3:   if  $Seed$  or  $M_{BRANCH}$  message received then
4:     split  $Target$  or  $M_{BRANCH}$  message contents into separate branches
5:     for each  $side$  of the robot do
6:       if there is a neighbour docked and there should be then
7:         if the neighbour is of the correct type and orientation then
8:           send  $M_{BRANCH}$  message to  $side$ 
9:         else
10:          send  $M_{DISASSEM}$  message to  $side$ 
11:        end if
12:       else if there is a neighbour docked but there shouldn't be then
13:         send  $M_{DISASSEM}$  message to  $side$ 
14:       end if
15:       transition to InStructure
16:     end for
17:   end if
18:   if not  $BestStructure$  or  $M_{DISASSEM}$  message received then
19:     propagate  $M_{DISASSEM}$  message to all other neighbours
20:     transition to Disassembling
21:   end if
22: end procedure

```

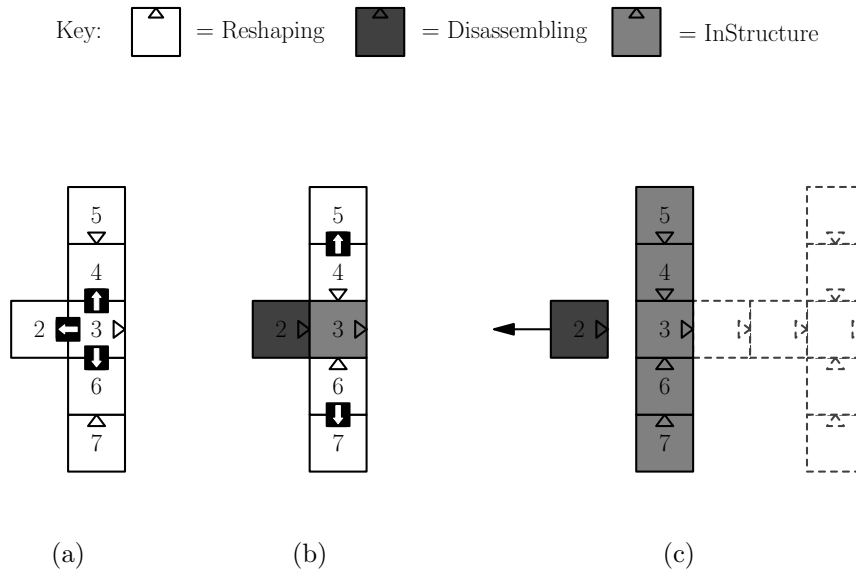


Figure 6.11: An example of modules executing the Reshaping behaviour. The shade of each module represents the state that it is currently occupying. The white arrows within black boxes represent the passing of M_{BRANCH} and $M_{DISASSEM}$ messages between neighbouring modules. The black arrow signifies movement and the dashed grey boxes highlight the positions in which modules will eventually be recruited

its other neighbours and transition to the Disassembling state (algorithm 4, lines 19-20). Meanwhile, if there is a neighbour which *is* of the correct type and orientation, then an M_{BRANCH} message will be sent on that side, containing the relevant information for that branch of the graph. Any module that receives an M_{BRANCH} message (algorithm 4, line 3) will repeat the above process for each of its neighbours, using the branch information that it received.

Considering the running example from this section, in figure 6.10, it was shown that at the end of the BroadcastScore state, every module should be aware that sub-structure led by module 2 is the winning structure. Therefore, because the *BestStructure* flags of modules 9 and 11 will be set to *false*, after entering the Reshaping state, they will transition immediately to the Disassembling state (algorithm 3 lines 18-20). Assuming that within the remaining sub-structure, module 3 is the *BestModule* and has a *Target* shape equivalent to that of the original structure from figure 6.3, the remainder of the *Reshaping* behaviour will proceed as shown in figure 6.11.

Because modules 4 and 6 fit within the *Target* shape, module 3 will send them each an M_{BRANCH} message containing the relevant branch information (figure 6.11a). As shown in figure 6.3, there should be no module docked on the rear of the *Seed* module. Module 2, therefore, does not fit within the *Target* shape of module 3, and consequently is sent an $M_{DISASSEM}$ message. After receiving the M_{BRANCH} messages, modules 4 and 6 will in turn send M_{BRANCH} messages to modules 5 and 7, meanwhile, when module 2 receives the $M_{DISASSEM}$ message, it will enter the Disassembling state

(figure 6.11b). Finally, as shown in figure 6.11c, whilst module 2 disassembles, modules 3-7 will enter the InStructure state and continue to assemble the new structure.

6.1.3 Comparisons

In chapter 2, several different forms of fault tolerant robotic system were reviewed. In this section, some of these earlier systems are compared with the self-repair strategy introduced in this chapter.

In contrast to the likes of Ackerman and Chirikjian [1] and Bererton and Khosla [8], which aim to repair the sub-systems of an individual robot by physically replacing failed components, the approach described in this chapter aims to repair a multi-robot structure by physically replacing failed robots. Unlike the framework of Parker [145], in its current form, the strategy described here also does not consider whether a failed module would be able to take on a different role within the system. Irrespective of the type of fault, failed modules are simply removed and replaced with functioning ones.

The reliance on the availability of redundant modules unites this approach with those of Christensen [30], Cheng et al. [24] and Rubenstein and Shen [153]. However, several differences between these systems and the self-repair strategy described in this chapter can also be identified. Firstly, in the experiments reported in this chapter, it can be observed that far fewer robotic modules are utilised. Secondly, in the strategy described in this chapter, the structural configuration of a group of robots is precisely defined by the interconnections of the individual units. Contrastingly, in the alternative systems, configurations are defined from a high level description of the general pattern that the group of modules form. Finally, whereas in the systems of [30], [24] and [153], failures are catastrophic events which involve removing or displacing several individual units, in this work, the events that the system is designed to recover from are far less drastic, involving only the failure of an individual module.

The approach described in this chapter most closely resembles that of Tomita et al. [179]. Following the introduction of a failure, both systems rely on reverting to an earlier state of assembly, before rebuilding the new structure. However, despite their superficial similarities, the manner in which the systems are reverted differ greatly. In the case of this work, the earlier state of assembly is reached by splitting the current structure into sub-structures, about the point at which the failure occurred. Contrastingly, in [179], the system is gradually degraded by removing individual units, until the point at which the failed modules were first added to the structure. The systems also differ in terms of where the replacement modules are recruited from. In the strategy described in this chapter, free individuals are recruited from the surrounding environment. Whereas, in [179], spare modules are recruited from other parts of the original structure.

The similarities between modules in the Supporting state and modules undertaking collective locomotion [7, 130, 181] or collective transport tasks [16, 45, 63] have already been highlighted. More specifically, parallels may be drawn between this behaviour and the approach described by O’Grady et al. [137] for transporting failed modules to a specialised ‘repair’ zone. Other similarities can be seen in the work of Yim et al. [208],

in which, after an ‘explosive event’ a group of modules is split into several separate sub-structures. Like the modules executing the MoveAway behaviour, these sub-structures are capable of independent locomotion, however, whereas in the work of [208], sub-structures are able to dock with one another in order to reform the original structure, in this work, it is assumed that modules can only dock as individuals.

6.2 Symbricator Implementation

This section describes a platform specific implementation of the self-repair strategy, designed for use with the Symbricator robots. Four important changes are made to the general controller from section 6.1.2. Firstly, in order to allow the strategy to work with Scout robots, an adaptation is made to the DetermineShape state which removes the assumption from section 6.1.1 that modules must possess omnidirectional locomotion. Secondly, an implementation of the CalculateScore state is provided which complements the original morphogenesis controller described in chapter 4. Thirdly, a method for calculating the duration of the broadcasting period of modules in the BroadcastScore state is presented. Finally, the reliability of the system is improved by introducing redundancy to communications. In the remainder of this section, each one of these four alterations is described in detail. The section concludes with a discussion on some of the limitations of this implementation.

6.2.1 DetermineShape Adaptation

In section 6.1.2, when considering the example structure from figure 6.3, it was said that the failure of module 0 would result in the formation of three separate sub-structures, one led by each of modules 2, 9 and 11. As shown in figure 6.12a, during the MoveAway sub-state of the Repairing behaviour, a sub-structure led by module 2 would be required to move towards the right. However, if any of modules 4-7 were Scout robots, movement in this direction, perpendicular to the headings of these robots, would not be possible.

To resolve this issue, the DetermineShape state is extended so that, before a module is instructed to enter the Repairing state, its neighbour first checks whether it will be able to move in the required direction. If it will not be able to move in the same direction as the other members of the sub-structure, the robot is instructed to form a new sub-structure, which will move perpendicular to that of the original. This behaviour is realised by replacing the sending of M_{SHAPE} messages on lines 11-12 and 22-23 of algorithm 1, with the new procedure from algorithm 5.

Using equation 6.1, the procedure first determines the *Heading* that the module at the front of the robot’s *DockedQ* will need to travel in (algorithm 5, line 2). If the module is capable of moving in this direction, there is no change in behaviour and, as before, the module is sent an M_{SHAPE} message containing the *Shape* string (algorithm 5, lines 3-5). However, if the module is not capable of moving in this direction, it must be pruned and a new sub-structure formed. A new sub-structure is created by sending

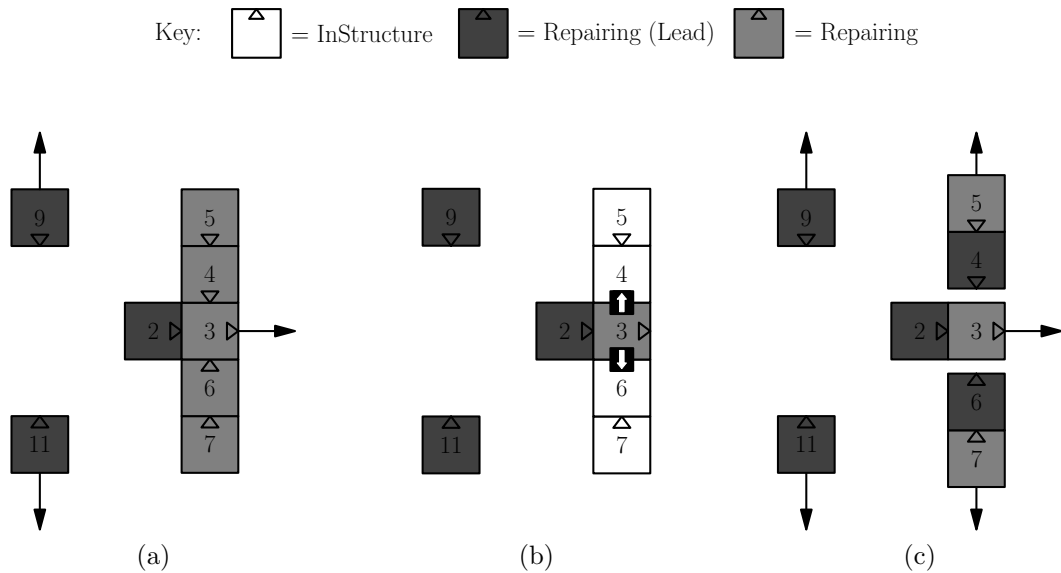


Figure 6.12: The behaviour of modules in the original and adapted DetermineShape states. Figure (a) shows that, in this scenario, the original DetermineShape implementation would produce three sub-structures. Figures (b-c) show that, if modules 4 and 6 were Scout robots, the adapted implementation would produce two further sub-structures. The shade of each module represents the state that it is currently occupying. The white arrows within a black boxes represent messages being sent between two neighbouring modules

Algorithm 5 Procedure to check whether a robot can move in the required direction

```

1: procedure CHECKBEFORESEND
2:   NeighbourHeading  $\leftarrow$  direction that DockedQ.FRONT should move
3:   if DockedQ.FRONT can move in direction of NeighbourHeading then
4:     Shape.APPEND(own ‘type’ + ‘side’ at which DockedQ.FRONT docked)
5:     send  $M_{SHAPE}$  message containing Shape to DockedQ.FRONT
6:   else
7:     send  $M_{FAILED}$  message to DockedQ.FRONT, not requesting help
8:     DockedQ.DEQUEUE
9:   end if
10: end procedure

```

the module an M_{FAILED} message, in which help is not requested (algorithm 5, line 7). When this message is received, it will cause the receiving module to enter the Repairing state as the lead module of a new sub-structure. Since the lead of the new sub-structure is not considered part of the original sub-structure, to prevent the pruning robot from waiting for a response from this module, it is removed from the front of the robot’s *DockedQ* (algorithm 5, line 8). The Repairing behaviour then continues as normal in both the original and the new sub-structures.

In the example from figure 6.12, if modules 4-7 are assumed to be Scout robots, when module 3 enters the DetermineShape state, rather than send M_{SHAPE} messages to these robots, it will send M_{FAILED} messages (figure 6.12b). Robots 4 and 6 will then lead two new sub-structures, which will move perpendicularly to the structure led by module 2. Resulting in the formation of five separate structures, all of which are able to move away from the failed module, without hindering one another (figure 6.12c).

There is one special case in which this solution will not work. If a module cannot move in the necessary direction, but pruning it would still require the module to move in this direction, it will not be possible to form a new sub-structure. For example, in the example structure from figure 6.3, assuming that module 7 was to fail and a sub-structure containing the remaining modules, led by module 6, was created. The desired *Heading* of this structure would be directly up, away from module 7, but if module 3 was a Scout robot, this movement would not be possible. Neither, would it be possible to prune module 3, because to do so would still require module 3 to move sideways. Fortunately, based upon the assumption from section 6.1.1 that robots can only dock as individuals, and therefore that Scout modules can only dock using their front or rear sides, this scenario can only ever arise in one part of a structure at a time. The only time in which this scenario could occur twice within the same structure would be if there existed a chain of modules connecting the left or right sides of two Scout modules, and one of the modules within this chain failed. However, as shown by the proof of theorem 1, such a configuration is not possible within a valid structure. Therefore, to resolve this issue, it is sufficient to allow the sub-structure which contains the immobilised Scout robot to remain stationary.

Theorem 1. *There cannot exist a chain of modules in which both the first and last modules within the chain are connected to the left or right sides of a Scout robot.*

Proof by contradiction. Assume that there exists a chain of modules, at either end of which two Scout robots, S_A and S_B , are docked using their left or right sides. Since Scout robots can only dock themselves using their front or rear sides, any module which is docked on the side of a Scout robot, must have joined the structure later than the Scout. That is to say, every member of the chain that starts on the side of S_A , up to and including S_B , must have joined the structure after S_A . Likewise, every member of the chain that starts on the side of S_B , up to and including S_A , must have joined the structure after S_B . Implying both that robot S_B joined the structure after robot S_A and that robot S_A joined the structure after robot S_B , a contradiction. \square

6.2.2 CalculateScore Implementation

There are many different ways of implementing the CalculateScore state. In this section, one particular implementation is described that was designed specifically with the Symbricator platform and the original morphogenesis controller in mind.

In this implementation, it is assumed that the *Target* of every module is identical to the original structure. The implementation involves each module within a sub-structure considering itself as the *Seed* of a new structure and calculating the size of the largest common subgraph that it shares with the *Target* shape. The score of a sub-structure is the size of the largest subgraph that any of the modules within the sub-structure share with their *Target*. Because the sub-structure with the highest score will later go on to re-construct the remainder of the *Target*, and to do so will use the original morphogenesis controller, this subgraph must contain the *Seed* of the *Target*. Figure 6.13, shows a *Target* shape (a) and three potential sub-structures with their scores assigned (b-d). Note that, although the structure in figure 6.13c contains more modules than that of figure 6.13d, its score is lower because not all of its modules belong to the largest subgraph which contains the *Seed*. This limitation is discussed in section 6.2.5.

The behaviour of robots in the CalculateScore state is outlined by algorithm 6. In a similar manner to that of the procedure from algorithm 1, the robots collectively perform a distributed depth first traversal of their sub-structure. At each stage, modules compare the size of the largest common subgraph that they and their *Target* share, with that of the largest found so far by any other module. The greater of these two values is passed on to the next module. At the end of the traversal, the lead module will know the size of the largest subgraph found by any individual, and this value is subsequently assigned as the *BestScore* of the sub-structure.

The *Shape* string constructed by a group of modules in the DetermineShape state, describes the configuration of the sub-structure to which the modules belong, from the perspective of the lead module. The string constructed in figure 6.7, for example, describes a structure from the perspective of module 2, that is, with module 2 in the position of the *Seed*. Before a module can compare its sub-structure with its *Target*, it must first transform the sub-structure graph to its own perspective. As shown in

Algorithm 6 Procedure for calculating the *Score* of a sub-structure

```

1: procedure ONENTRY
2:   Parent  $\leftarrow$  the module which caused this robot to enter the Repairing state
3:   DockedQ  $\leftarrow$  all of the robots docked with this module (excluding Parent)
4:   if not the lead module then
5:     wait for  $M_{SHAPE}$  message from Parent
6:     Shape  $\leftarrow$  a transformed version of the Shape received from Parent
7:   end if
8:   ModuleScore  $\leftarrow$  size of the largest common subgraph of Shape and Target
9:   BestScore  $\leftarrow$  ModuleScore
10:  if not the lead module and score received from Parent  $>$  ModuleScore then
11:    BestScore  $\leftarrow$  score
12:    ModuleScore  $\leftarrow$  0
13:  end if
14:  if not DockedQ.EMPTY then
15:    send  $M_{SHAPE}$  containing Shape and BestScore to DockedQ.FRONT
16:  end if
17: end procedure

18: procedure CALCULATESCORE
19:  if not DockedQ.EMPTY then
20:    if  $M_{SHAPE}$  message received from DockedQ.FRONT then
21:      DockedQ.DEQUE
22:      if score received in  $M_{SHAPE}$  message  $>$  BestScore then
23:        BestScore  $\leftarrow$  score
24:        ModuleScore  $\leftarrow$  0
25:      end if
26:      Shape  $\leftarrow$  a transformed version of the Shape received from Parent
27:      if not DockedQ.EMPTY then
28:        send  $M_{SHAPE}$  containing Shape and BestScore to DockedQ.FRONT
29:      end if
30:    end if
31:  else
32:    transition to BroadcastScore
33:  end if
34: end procedure

35: procedure ONEXIT
36:  if not the lead module then
37:    send  $M_{SHAPE}$  message containing Shape and BestScore to Parent
38:  end if
39: end procedure

```

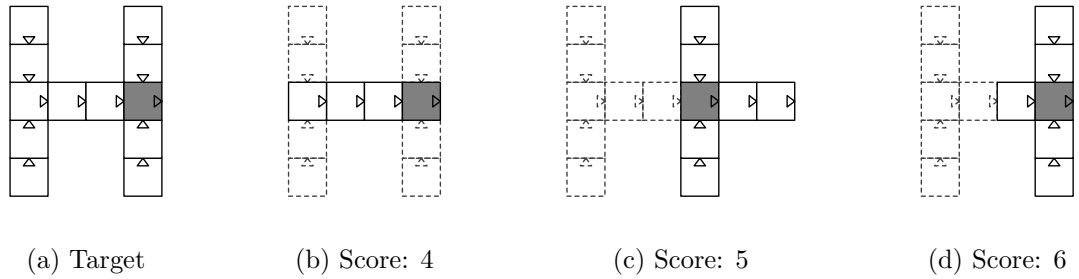


Figure 6.13: A target shape (a) and examples of the scores assigned to three potential sub-structures (b-d). In each figure, the *Seed* is shown in grey. In figures b-d, the footprint of the target is included to highlight the area of the largest common subgraph

figure 6.14, to transform a graph from the perspective of one module to that of its neighbour, all that is required is for the direction of the edge that joins the two modules to be reversed, and for the first pair of symbols in the edge label to be swapped with the second. In terms of string manipulation, as shown in figure 6.15, this equates to swapping the first two pairs of symbols, moving them to the position of their matching four NULL symbols, and moving the NULL characters to the end of the string.

Upon entering the CalculateScore state, the lead module in each sub-structure calculates its *ModuleScore* as the size of the largest common subgraph shared by its *Shape* and *Target* (algorithm 6, line 8). This value is assigned as the *BestScore* found so far (algorithm 6, line 9), and along with the *Shape* string, is sent within an M_{SHAPE} message to the module's first neighbour (algorithm 6, line 15). Every other module that enters the CalculateScore state first waits until it receives the *Shape* string and *BestScore* values sent by its *Parent*, and then, using the method demonstrated in figure 6.15, transforms the *Shape* string to its own perspective (algorithm 6, lines 5-6). After calculating its own *ModuleScore*, each robot then compares its score with the value received from its *Parent* (algorithm 6, line 10). If the *score* received from a module's *Parent* is greater than its own, the *BestScore* variable takes this value and the *ModuleScore* variable takes the value 0 (algorithm 6, lines 11-12). The *Shape* string and *BestScore* are then propagated to the next module in the sub-structure and the process repeats. Every time a new message is received, the *score* and *BestScore* values are compared (algorithm 6, line 22), the *Shape* string is transformed (algorithm 6, line 26) and a message is sent to either a new neighbour (algorithm 6, line 28), or if non remain, back to the module's *Parent* (algorithm 6, line 37).

By comparing the graphs in figure 6.14 with that in figure 6.3, and observing the amount of overlap, it is possible to determine the *ModuleScore* values of robots 2, 3 and 4. For example, the subgraph in figure 6.14a which contains modules 2 and 3, is identical to the subgraph in figure 6.3 involving modules 0 and 1, therefore resulting in a score of '2' for module 2. Meanwhile, the subgraph in figure 6.14b containing modules 3-7 is identical to the subgraph in figure 6.3 involving modules 0, 8, 9, 10

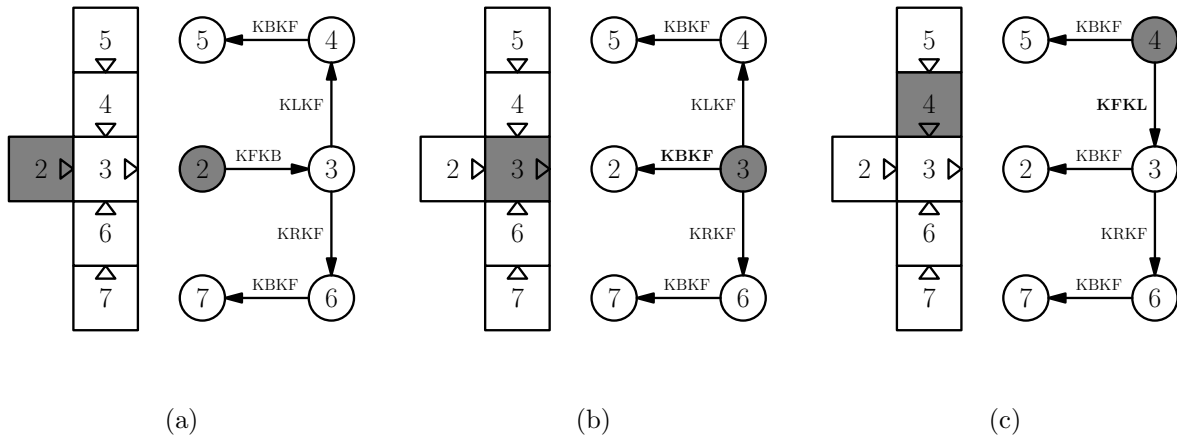


Figure 6.14: A demonstration of how the graph representation of a structure can be manipulated to transform the *Seed* from one module, to its neighbour. The *Seed* is represented by the dark grey square and the label of the edge which is manipulated during each transformation is highlighted in bold

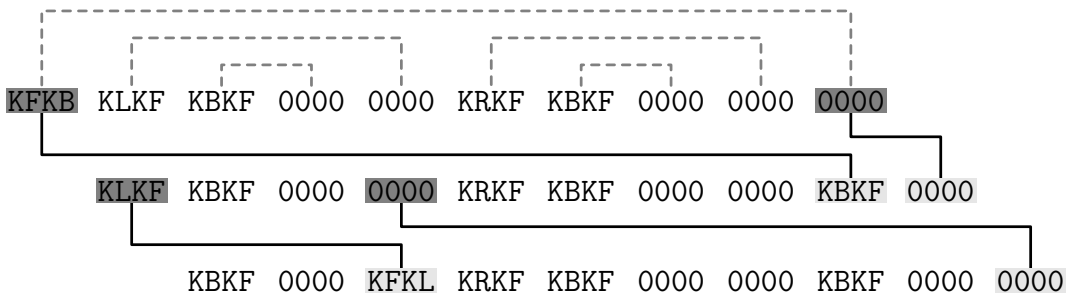


Figure 6.15: A demonstration of how the string representation of a structure can be manipulated to transform the *Seed* from one module, to its neighbour. The dashed grey lines show the different branches of the structure. The highlighted symbols and the solid black lines show the parts of the string which are manipulated during each transformation

and 11, resulting in a score of ‘5’ for module 3. Since there are no common subgraphs containing more than one module (and including the *Seed*) in figures 6.14c and 6.3, the score for module 4 is simply ‘1’. The score obtained by module 3 (5) is the largest of any module within the sub-structure and so, after a full traversal of the sub-structure, the *BestScore* variable of the lead module would hold this value. After transitioning to the BroadcastScore state, this *BestScore* value would be the initial value that all of the modules within the sub-structure would broadcast.

6.2.3 Broadcasting Period

The length of time for which sub-structures broadcast and listen for each others scores must be chosen carefully. The longer the broadcasting period, the longer the self-repair process will take, therefore, to maximise efficiency, the duration should be as short as possible. However, if the duration is too short, the modules of one sub-structure may have left the BroadcastScore state before the modules of another have entered, preventing the robots from reaching a common consensus.

Modules will only enter the BroadcastScore state after having first determined the shape and score of the sub-structure to which they belong. Since both of these procedures rely on performing a depth first traversal of the sub-structure, the time taken for modules to enter the BroadcastScore state differs depending upon the size of the structure to which they belong. Meaning that, modules within smaller sub-structures will enter the BroadcastScore state quicker than those within larger sub-structures.

In this implementation, since it is assumed that all modules have the same *Target*, every robot knows how many modules the *Target* contains. Using this common reference, sub-structures may dynamically set the duration of their broadcasting period in order to minimise the chances that one sub-structure will stop broadcasting before the others start. Specifically, each sub-structure may choose its broadcasting period by calculating the difference between the number of modules in the *Target* and the number of modules in its sub-structure. Therefore, sub-structures with fewer modules will broadcast for longer, whereas sub-structures with more modules will broadcast for less time. The result is that, although the modules from different sub-structures will enter the BroadcastScore state at different moments in time, they will all leave at approximately the same time.

6.2.4 Communication Redundancy

In chapter 4, the over reliance on a single channel of communication by the original morphogenesis controller was identified as a potential weakness. Furthermore, from experience working with the Symbricator robots, it was found that the Ethernet communication channel was sometimes unreliable. Occasionally and temporarily, a loose Ethernet connection would result in messages being lost. To address this issue, and further improve the reliability of the morphogenesis controller, in the experiments described in section 6.5, redundancy is introduced to the robots’ communications.

Rather than relying solely on wired communications to pass information between connected modules, the infrared (IR) components of the robots are also used to send and receive messages. In the adapted controller, every time a message is sent to a neighbouring robot, two copies are created, one that is sent using IR and one that is sent using wired communications. Furthermore, every time a robot receives a message, it is abstracted so that the robot will respond in the same way regardless of which channel it arrives on. To prevent robots reacting to the same message twice, as soon as one message has been received, all subsequent copies of the same message are ignored.

The wired communications channel can transmit messages at a much higher rate than those sent using infrared, therefore, in the majority of cases, robot will respond only to messages sent using wired communications. However, in early experiments with the real robotic hardware, it was found that, due to loose connections, the wired communications channel was not always reliable. The use of infrared, therefore, may serve as a back-up, in cases where a message sent using the wired channel does not arrive. Whilst it is possible that a messages sent on either of the channels will be lost, it is unlikely that both channels will fail simultaneously. By introducing this level of redundancy to the communications systems, the reliability of the system can be greatly improved, without requiring any alterations to the underlying strategy itself.

6.2.5 Limitations

This section is used to highlight some of the limitations of the implementation of the self-repair strategy developed for the Symbricator platform.

Every member of a losing sub-structure must be removed Following the identification of winning and losing sub-structures, the fact that every losing sub-structure must be completely disassembled, may be highlighted as inefficient. A better solution would be to allow the winning sub-structure to first recruit replacements for the failed and supporting modules, and then for each of the remaining sub-structures to rejoin in its original location. However, as highlighted in section 6.1.1, accurately controlling the movement of a group of connected modules during alignment and docking is a difficult task. Therefore, it is assumed that modules can only dock as individuals, and consequently, that before rejoining a structure every robot must first return to the Wandering state.

The structure must be rebuilt using the original plan In calculating the score of a sub-structure, when comparing the graph of a sub-structure with its *Target*, the condition that the largest common subgraph must include the *Seed* module imposes a further limitation. If any module within the winning sub-structure does not belong to the largest common (*Seed* containing) subgraph that the structure and its *Target* share, then it must be pruned from the new structure. For example, in the scenario from figure 6.11c, it was said to be necessary to remove module 2 before restarting the assembly process.

This limitation is present because, when assembling a new structure, modules are required to follow the *Target* specification exactly. Every description of a structural configuration partially defines the order in which modules must be added to the structure. There will exist different structural configurations, with the *Seed* module placed in different locations, which result in the construction of the same morphological structure, but impose different orderings over module recruitment. If it were possible to define a new *Target* shape that was structurally equivalent to the original, but positioned the *Seed* module at a different location, then a new *Target* could be defined for each sub-structure, in which every member of the sub-structure also belonged to the *Target*. That is to say, a *Target* could be created which contains a subgraph that is identical to the entire sub-structure. In such a scenario, it would never be necessary to remove modules from the winning sub-structure, and the score of each sub-structure could be calculated simply by counting the number of modules that it contains.

Such a strategy would not be difficult to implement and could, for example, use the same technique shown in figures 6.14 and 6.15 for transforming the perspective of a structure from that of one module to that of its neighbour. The reason why it is not implemented is because it is anticipated that the ordering in which modules are recruited may carry some importance, and changing this ordering by transforming the *Target*, could cause problems. For example, in a structure which contains Scout modules, since these robots cannot move sideways, any robot which is docked to the side of a Scout, must join the structure later than it. Similarly, a particular ordering may have been chosen to minimise interference during docking. Furthermore, the order in which modules are added to a structure may have already been designed with efficiency in mind and altering this order could result in a less efficient assembly time. Sticking with the original plan, therefore, is considered to be the safest course of action.

The shape of each sub-structure must be discovered The time taken for modules to determine the shape of the sub-structure to which they belong scales linearly with the sub-structure size. In this implementation, the *Target* of each module is set to be the same as the original structure, therefore, it may be queried why the discovery phase is needed at all, and whether the shape of a sub-structure can be determined in constant time. The reason why the shape cannot be determined in constant time, is because although every module is aware of the *Target*, it is not necessarily aware of its position within the *Target*. When a new module joins a structure it is only provided with information about the branches of the structure for which it is the root. Therefore, it is only aware of its position relative to the modules which it is responsible, directly or indirectly, for recruiting. Furthermore, it is reminded, that in other implementations it may not always be guaranteed that the *Target* of each module is the same as the original structure. In every sub-structure, the module which joined the original structure earliest will have been responsible (directly or indirectly) for recruiting all of the others. It may

be asked why the information held by this module cannot be used to determine the sub-structure shape quicker than the depth first traversal approach currently used. The reason is that, although the oldest member of the sub-structure is aware of every module that it is required to recruit, it is not necessarily aware how many of these modules have been successfully recruited, and whether those which have been recruited are part of the same sub-structure. The current approach was chosen in order to ensure that the self-repair process would work even if a structure has not been fully assembled.

It would be possible to implement a version of the original morphogenesis controller in which, during recruitment, modules are provided with more information about the shape of the structure they are joining, and their position within it. However, in this work, such an approach was avoided because of the desire to extend the original approach, rather than redesign it.

6.3 Stage Experiments

This section presents the results from a set of preliminary experiments, conducted using the Stage simulator. In these experiments, simulated Backbone robots were used to assemble two different types of robotic structure. During assembly, artificial faults were injected into the robots and the self-repair strategy introduced in section 6.2 was used to isolate, remove and replace the failed modules. In this section, the performance of the self-repair strategy is compared with that of a fault free system and a system in which a naive recovery strategy is employed. Whilst this set of experiments is limited to the investigation of two structural configurations, in section 6.4, this work is expanded to examine 47 different types of robotic structure.

6.3.1 Experimental Setup

The experiments described in this section use the same robot models as those in chapter 5. The simulated modules are functionally similar to the Symbicator Backbone robots. Like the Backbone robots, the modules are cubic in shape and possess one docking connector on each side. They also possess a range of simulated IR components which are used for communication and proximity detection. However, unlike the physical Backbone robots, the simulated modules cannot easily move omnidirectionally. Because of this limitation, target structural shapes were purposefully chosen in which the modules were only required to dock using their front sides.

Figure 6.16 shows screenshots of the two different structural configurations investigated in this section. The first configuration, presented in figure 6.16a, is a small, simple structure, referred to here as structure *A*. The structure is made up of 17 individual modules and contains two branch points. The second configuration, shown in figure 6.16b, is a larger, more complex structure, referred to as structure *B*. This structure is made up of 35 modules and contains three branch points.

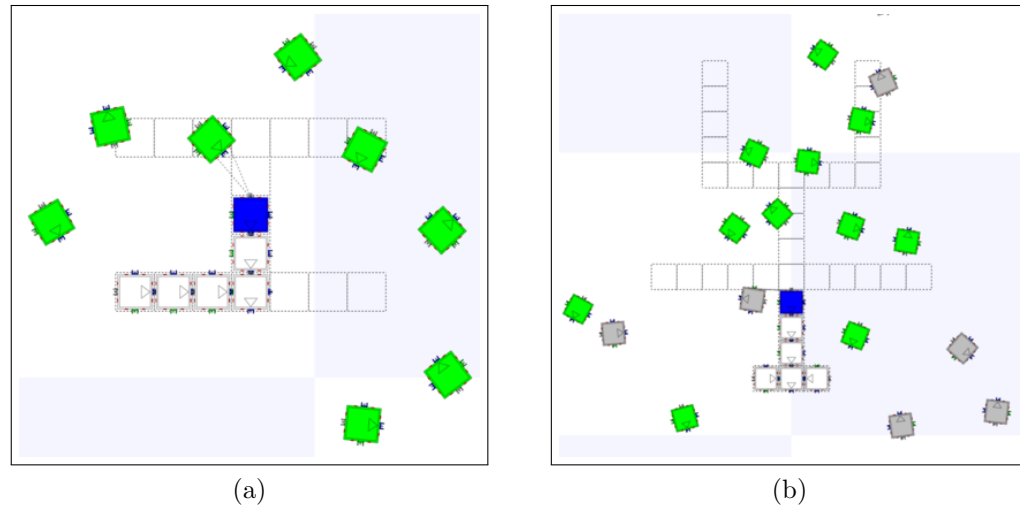


Figure 6.16: Two partially assembled structures within the Stage simulator

Each experimental run involved 49 individual modules. The robots were evenly distributed about an empty arena, with a single seed robot placed directly in the centre. During each run, after a random number of robots had joined the forming structure, a single fault was introduced into a single member of the structure. It is assumed that the modules possess an anomaly detection system which, after a short delay, is able to detect the presence of a fault with 100% accuracy. The type of fault which is injected is not specified but it is assumed that the failed robot will always require assistance from support modules in order to remove itself from the structure. Due to the limitations of the simulator, it is not possible to easily coordinate the collective motion of a group of connected modules, therefore, the presence of a navigation controller that allows the support modules to transport the failed robot away from the vicinity of the structure is also assumed. In the experiments described in this section, once a structure has been split into sub-structures, the failed robot and connected support modules are immediately removed from the arena.

In each run, the total time taken to complete the assembly of the structure was recorded. The performance of a system employing the self-repair strategy described in this chapter (SR) is compared with two other systems. The first is a naive recovery strategy (NR) in which, following the introduction of a failure, the structure is completely disassembled and the morphogenesis process is restarted from scratch. The second is a baseline system (BL), in which no robots fail and no form of recovery is performed, this serves as a benchmark for the other two strategies. For each of the two different structural configurations, 30 independent runs were performed. To assess the performance of the systems, the following two null hypotheses are presented:

$H6.1_0$: In a system containing a single failed module, there is no difference in the time taken to assemble structure A by a system employing the self-repair strategy (SR) and a system employing the naive strategy (NR).

H6.2₀ : In a system containing a single failed module, there is no difference in the time taken to assemble structure *B* by a system employing the self-repair strategy (SR) and a system employing the naive strategy (NR).

6.3.2 Results

The robots were able to successfully complete the assembly process in all 90 of the experimental runs. In the 60 runs in which faults were injected, both recovery strategies were consistently successful in isolating and removing the failed module. Videos are provided in the accompanying material showing the successful repair of structure *A* when a fault is injected during (video 6.1) and following (video 6.2) assembly. Figure 6.17 shows the behaviour of the self-repair strategy, during the assembly of structure *B*. In figure 6.17a a failure is introduced into the red robot, which is subsequently detected by the neighbouring yellow support modules. In figure 6.17b, the structure is split into two sub-structures and the failed module is removed from the arena. As shown in figure 6.17c, after negotiation, it is clear that the larger sub-structure is in the best position to continue assembly. Figure 6.17d shows the continuation of the assembly process after most of the extraneous modules have been pruned. Note that, due to the fact that certain structural motifs are repeated throughout the configuration, the members of the winning sub-structure are able to take on different roles in the new structure.

In figure 6.18, the performance of the self-repair strategy introduced in this chapter (SR) is compared with a baseline system (BL) and the naive recovery strategy (NR). The graphs show the time taken to assemble structure *A* (6.18a) and structure *B* (6.18b). Each boxplot contains the data from 30 individual runs. The ‘*’ symbol is used to represent where there is a significant difference between the distributions of two experiments, according to a Wilcoxon rank-sum test with a significance level of 0.05.

For structure *A* (figure 6.18a), it is observed that both recovery strategies take significantly longer to complete the assembly process than the fault free system. It is also observed that the naive recovery strategy is quicker, on-average, than the self-repair strategy (albeit not significantly so). On this evidence, it is not possible to reject hypothesis *H6.1₀*.

The reason why the naive approach performs slightly better than the self-repair strategy may be attributed to the fact that, in certain scenarios, it is quicker to disassemble and start again, rather than deliberate about which sub-structure is in the best position to initiate re-assembly. Specifically, if the time that the naive strategy takes to disassemble and re-form to the size of the best placed sub-structure is shorter than the time taken by the self-repair strategy to determine which sub-structure is best placed, then the naive strategy will perform at least as well as the self-repair strategy.

Taking this into consideration, it may be anticipated that the self-repair strategy will perform better than the naive strategy on larger structures, in which the cost of disassembly is larger, but the cost of negotiation remains the same. The results from the experimental runs involving the larger, more complex, structure *B*, help to strengthen this argument. For structure *B* (figure 6.18b), it is observed that although

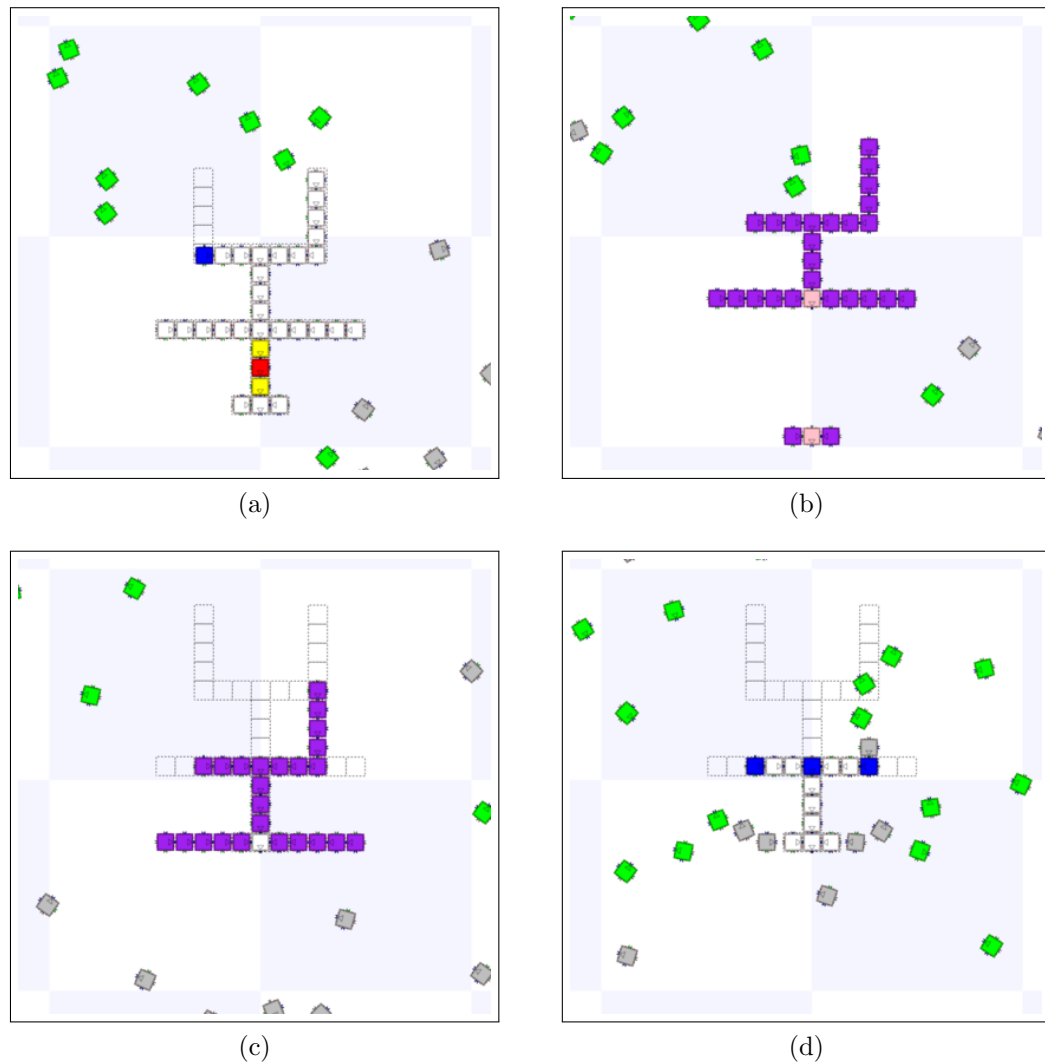


Figure 6.17: An example of the self-repair process during the assembly of structure B . The white modules represent robots in the InStructure state; the blue modules represent robots in the Recruiting state; the red modules represent robots in the failed state; the yellow robots represent modules in the Supporting state; the pink and purple modules represent robots in the Repairing state; and the green and grey modules represent robots in the Wandering, LocateBeacon and Alignment states

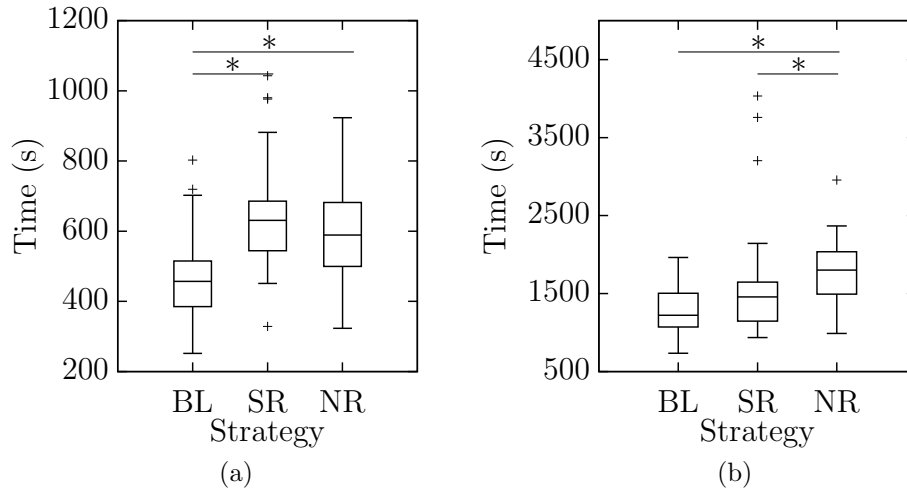


Figure 6.18: Boxplots showing the time taken to complete the assembly of structures *A* (a) and *B* (b), for each of the three different systems

the self-repair strategy is still slower than the fault free system, the difference is no longer statistically significant. Furthermore, the self-repair strategy shows a significant improvement in performance over the naive strategy. Based upon this evidence it is possible to reject $H_{6.2_0}$ and state that there is a difference in the time taken to assemble structure *B* by a system employing the self-repair strategy and a system employing the naive recovery strategy.

6.3.3 Analysis

In this section, the time taken to assemble two different types of robotic structure was investigated using three different systems, with and without the introduction of artificial faults. For a small, simple structure, a system employing the self-repair strategy was shown to perform significantly worse than a fault free system and worse (but not significantly so) than a system employing a naive recovery strategy. For a larger, more complex structure, the difference in performance between the self-repair strategy and the fault free system was no longer significant and, in a reversal of the findings from the smaller structure, the self-repair strategy performed significantly better than the naive strategy.

It is suggested that the reason for these different outcomes may be related to the shape and size of the structures involved. Both the self-repair strategy and the naive recovery strategy carry an associated cost. With the self-repair strategy, this cost is the time taken by the system to split into sub-structures and to determine which sub-structure is in the best position to continue assembly. With the naive strategy, the cost is the time taken to disassemble and re-assemble to the point at which the recovery process was started. Based upon the preliminary experiments described in this section,

for the smaller structure, the cost of disassembling appears, on average, to be less than the cost of negotiation. For the larger structure, meanwhile, the reverse is true.

Another factor which may also play a part in determining which strategy performs best, is the amount of repetition found within the target structure. With the self-repair strategy, the winning sub-structure is that which is found to share the largest common, seed-containing, sub-graph with the target shape. In structures with little repetition, modules will be less likely to share large sub-graphs with one another. Therefore, the size of the largest common sub-graph shared by the winning sub-structure and the target is likely to be smaller on average. This means the point that the system must revert back to before re-starting assembly will be earlier, and therefore that the time taken to complete assembly will be greater.

6.4 Robot3D Experiments

In order to analyse the behaviour of the self-repair strategy in greater detail, in this section, results are presented from experiments using a variety of different structural configurations. Seven new hand-designed structures are introduced, as well as 40 randomly generated configurations. To avoid some of the previously discussed limitations of the Stage simulator, in this section, the more realistic ‘Robot3D’ simulator is used. To aid analysis, a metric is introduced to classify different types of structure, and the relationship between this value and the speed of the self-repair strategy is discussed.

6.4.1 Experimental Setup

The experiments described in this section use simulated Backbone modules from the Robot3D simulator (introduced in section 3.1.2). The modules are based upon realistic 3D models of the physical Symbricator robots and, unlike the robots from the Stage simulator, are capable of full omnidirectional locomotion. Figure 6.19a shows a close up of two of the Backbone modules from the Robot3D simulator and figure 6.19b shows a topdown view of the arena in which the experiments took place.

The performance of the self-repair strategy is compared with a system utilising a naive recovery strategy and a baseline system in which no faults were injected. In each experiment, the mean time taken to assemble the target structure was recorded. Each experimental run lasted for one simulated hour and involved 21 individual modules. At the start of each run, the robots were evenly distributed throughout the arena at random orientations. In each run involving the self-repair and naive recovery strategies, a single fault was injected into a single robot. The faults were injected at a random point in time between when the second module joined the structure and the average time taken to assemble the structure in the baseline experiment. The Robot3D simulator uses a more realistic physics model than the Stage simulator and is therefore slower. Consequently, fewer robots were used than in the Stage experiments. Due to the limited number of robots, it was assumed that the faulty robots would not require assistance

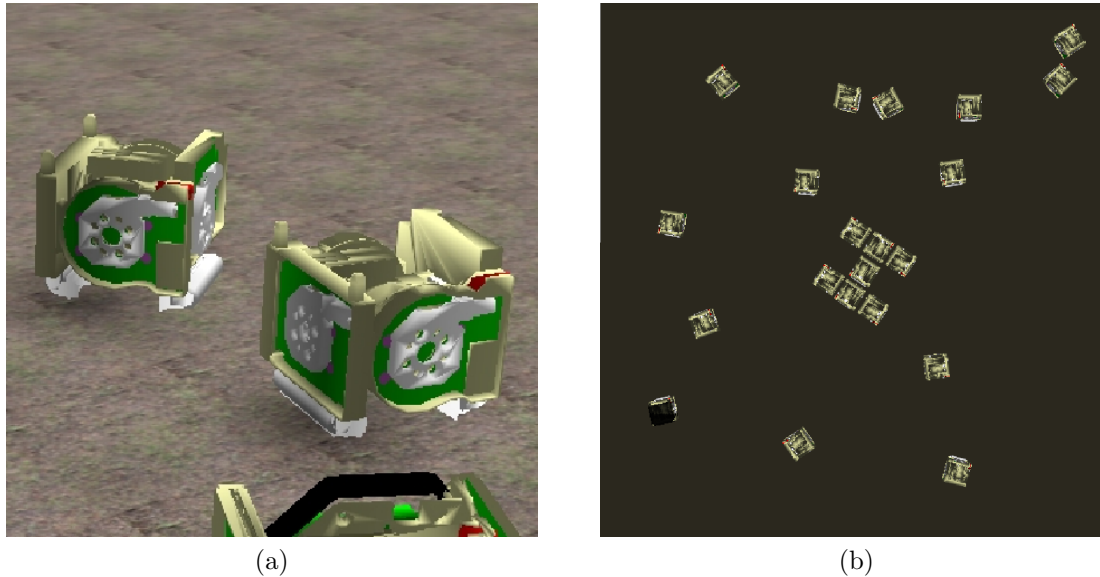


Figure 6.19: A close up view of the Backbone modules in the Robot3D simulator (a) and a top down view of the arena used for the experiments reported in this section (b)

in removing themselves from the structure and that shortly after removing themselves, would become fully functional again.

Data was collected from experiments involving 47 different structural configurations. For each configuration, and each of the three systems, 100 independent runs were performed. Each of the 47 structures contained between 6 and 12 modules. Seven of the structures were designed by hand. These structures, shown in figure 6.20, were chosen based upon what were perceived to be desirable characteristics of a multi-robot structure: stability, symmetry and repetition. The remaining 40 structures were generated at random. Four sets of ten structures, each containing 6, 8, 10 and 12 modules were generated. Each structure was screened to ensure that it did not include any loops or other motifs which would make it physically impossible to assemble. Some examples of the randomly generated structures are shown in figure 6.21.

To help classify different structural configurations a new metric, referred to as the ‘repair potential’, is introduced. As shown in algorithm 7, the repair potential of a configuration is determined by iterating through the sub-graphs produced when each edge of the structure’s graph is removed. For each sub-graph, the size of the largest common (seed containing) sub-graph shared with the original graph is calculated. This value relates directly to the sub-structure score calculation introduced in section 6.2.2. The sum of these values is then divided by the product of the number of edges (connections) and the number of vertices (robots) in the graph to produce the repair potential score.

The maximum repair potential of a configuration is 1. A score of 1 is obtained when cutting the graph of a structure at any edge results in the creation of two sub-structures which both have scores equal to the number of modules they contain. Configurations

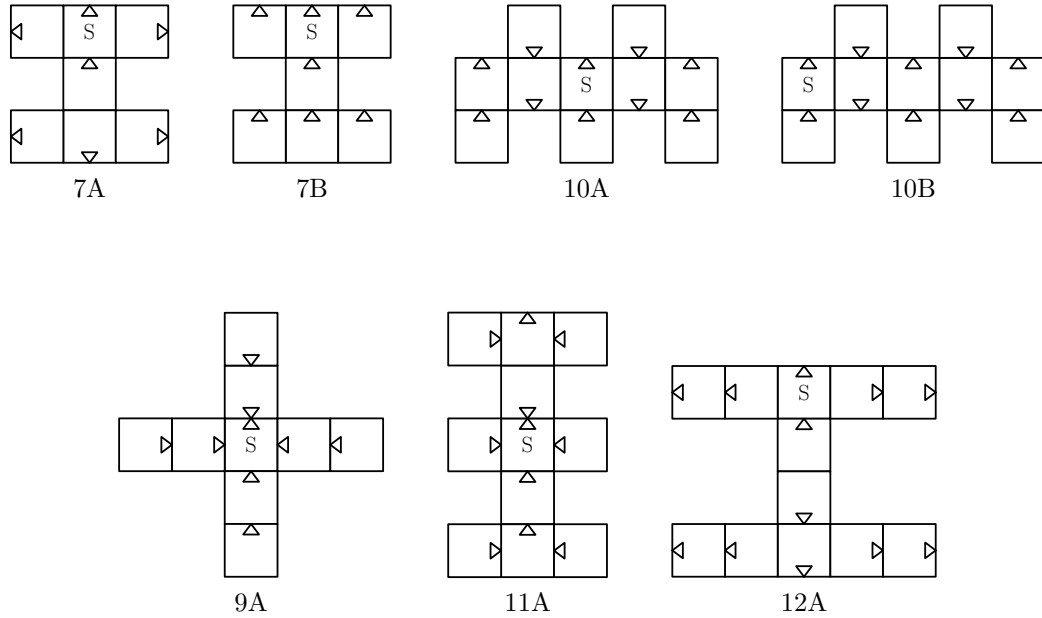


Figure 6.20: The seven hand-designed structures used in this section

Algorithm 7 Procedure for calculating *repairPotential* of a structure

- 1: **procedure** REPAIRPOTENTIAL(*structure*)
 - 2: $G = (V, E) \leftarrow$ the graph of the *structure*
 - 3: *potential* $\leftarrow 0$
 - 4: **for each** edge $e \in E$ **do**
 - 5: $A, B \leftarrow$ the two connected graphs created by removing e from G
 - 6: *potential* \leftarrow *potential* + SCORE(A, G) + SCORE(B, G)
 - 7: **end for**
 - 8: **return** *potential* $\div (|E| \times |V|)$
 - 9: **end procedure**

 - 10: **procedure** SCORE(A, B)
 - 11: **return** the size of the largest common (seed containing) sub-graph of A and B
 - 12: **end procedure**
-

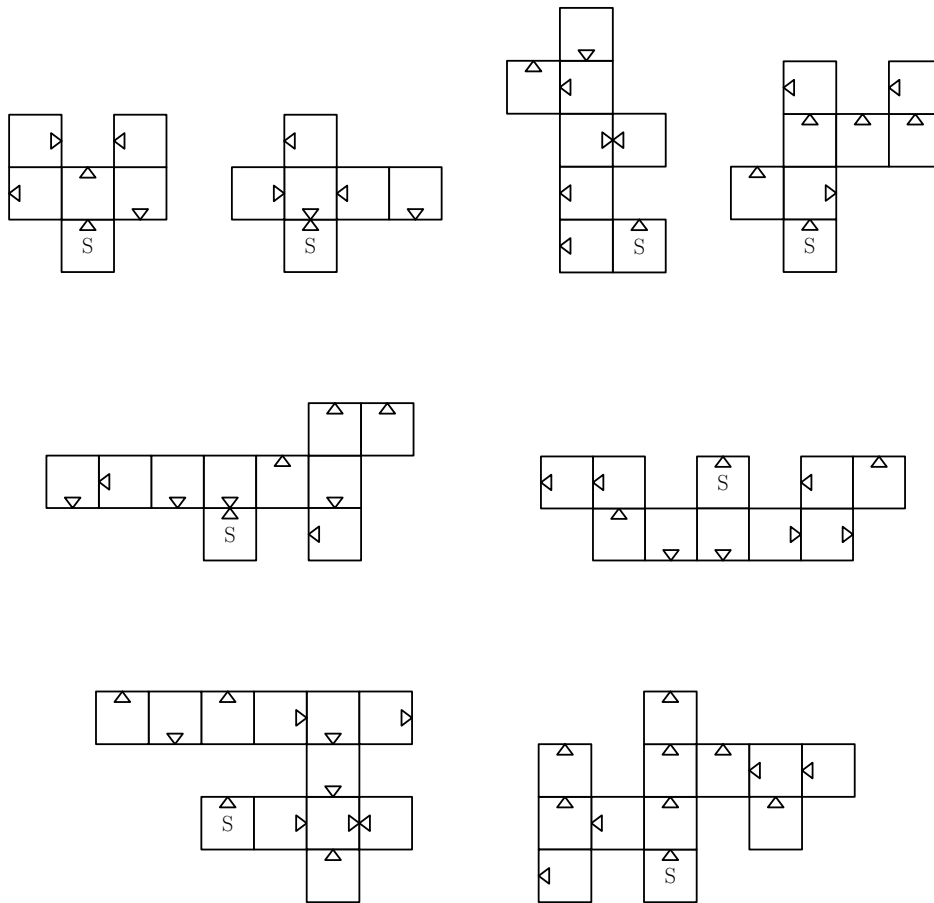


Figure 6.21: Some of the randomly generated structures used in this section

9A and 10A are examples of such structures. An important property of structures with a repair potential of 1 is that, during self-repair, regardless of which module fails and which sub-structure is selected to continue assembly, no pruning of unused modules is required. Interestingly, although structure 10B is topologically identical to 10A, it does not have the same repair potential. This can be confirmed by considering the case in which the connection between the seed module and its right neighbour is cut. This will produce two sub-structures, one containing two modules which obtains a score of two and one containing eight modules but which only receives a score of six.

The minimum repair potential of a structure is 0.5. A score of 0.5 would, for example, be the score of a two module structure which contained one Backbone and one Scout robot. Regardless of which of the two modules was designated as the seed, cutting the connection between them will always result in the formation of one structure that has a score of 1 (the seed) and one structure that has a score of 0.

To further aid analysis, the time that the system spent recovering was also measured. When a fault is introduced into the system, the number of robots in a structure is guaranteed to decrease by at least one. The recovery time was calculated as the period between when a fault is introduced and the point at which the number of modules in the structure returns to the amount it was at before the fault was introduced. Note that the configuration of the structure immediately before and after the recovery period need not be identical, it is sufficient that the structure contains the same number of modules. Along with the overall assembly time, using the following two null hypothesis, the recovery time is used to compare the performance of the self-repair strategy with the naive recovery approach:

H6.3₀ : There is no difference in the assembly time of the self-repair strategy (SR) and the assembly time of the naive recovery strategy (NR).

H6.4₀ : There is no difference in the recovery time of the self-repair strategy (SR) and the recovery time of the naive recovery strategy (NR).

In section 6.3.3, it was suggested that the self-repair strategy would perform better on larger structures, whilst the naive recovery strategy would perform better on smaller structures. To test this theory, the following two null hypothesis are presented:

H6.5₀ : There is no correlation between the recovery time of the self-repair strategy (SR) and the size of the structure under study

H6.6₀ : There is no correlation between the recovery time of the naive recovery strategy (NR) and the size of the structure under study

It was also noted that different characteristics of the structures, such as the amount of repeated patterns, may also affect performance. To test this theory, the ‘repair potential’ is used and the following null hypothesis is presented:

H6.7₀ : There is no correlation between the recovery time of the self-repair strategy (SR) and the repair potential of the structure under study

	Assembly (s)			Recovery (s)		Success	
	BL	SR	NR	SR	NR	SR	NR
7A	234.07	336.16	361.82	107.93	130.50	1.000	1.000
7B	221.73	316.89	371.80	82.805	126.69	1.000	1.000
9A	191.44	266.46	312.28	72.921	114.93	1.000	1.000
10A	257.24	349.17	395.90	86.478	137.74	1.000	1.000
10B	364.35	464.27	589.06	111.81	210.24	1.000	1.000
11A	287.60	391.97	468.84	89.400	169.99	1.000	1.000
12A	480.57	607.45	754.72	135.80	277.25	1.000	1.000
6R	293.22	457.27	469.29	137.23	171.61	0.998	1.000
8R	350.17	494.80	558.07	130.43	201.30	1.000	1.000
10R	460.90	641.51	734.03	161.94	258.82	0.988	0.997
12R	549.12	744.00	861.41	169.16	304.02	0.983	0.983

Table 6.2: A summary of the results from the Robot3D experiments. From left to right the columns show the target structure; the mean assembly time during the baseline (BL), self-repair (SR) and naive recovery (NR) experiments; the mean recovery time during the self-repair (SR) and naive recovery (NR) experiments; and the success rate of the self-repair (SR) and naive recovery (NR) strategies. The bottom four rows display the combined results from all randomly generated structures of the same size

6.4.2 Results

Table 6.2 summarises the results of the experiments reported in this section. The first seven rows of table 6.2 show the results from the experiments involving the hand designed structures, each row summarises the results from 100 independent runs. The final four rows show the results from the experiments involving randomly generated structures and are grouped according to the size of the structures involved. In the final four rows, each group represents the outcome from 1000 independent runs. For every structure, the mean assembly time was found to be shortest during the baseline experiments, followed by the self-repair experiments and finally the naive recovery approach. The mean recovery time of the self-repair strategy was always found to be less than that of the naive approach.

In the experiments involving hand designed structures, the robots were always able to successfully recover from the introduction of a fault and complete the assembly of the structure. In the accompanying material, video 6.3 shows robots utilising the self-repair strategy to successfully recover from a failure which occurs following the assembly of structure 12A. In the experiments involving randomly generated structures, whilst the robots were successful in the vast majority of cases, they were occasionally prevented from completing assembly. All of the unsuccessful runs resulted from scenarios in which, following the introduction of a fault, and the subsequent partial re-assembly, the new positioning of the target structure meant that it was difficult or impossible for one or

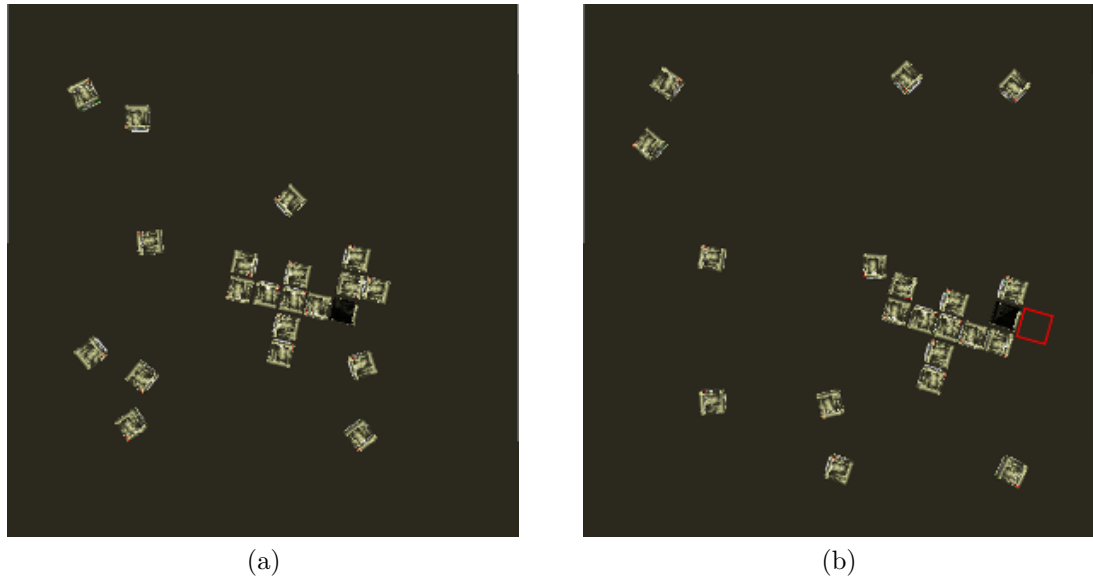


Figure 6.22: A complete 12 module structure (a) and an incomplete version (b) in which assembly was halted by the close proximity of the structure to the arena boundary (which, in this figure, corresponds to edge of the image)

more robots to find or dock with the structure. This scenario is demonstrated in figure 6.22. Figure 6.22a shows the shape and positioning of one of the randomly generated 12-robot structures, following a successful recovery attempt. Figure 6.22b shows an unsuccessful recovery attempt on the same structure in which one module is missing from the target. The red box in figure 6.22b shows where the final module should be docked. Due to the close proximity of the structure to the arena wall, no module was able to successfully navigate to this location within the 1 hour time limit. Whilst the seed of a structure will always start in the centre of the arena, after a fault has been introduced, any module within the structure may become the new seed and may consequently move the location of the structure. As reflected by the results in table 6.2, this problem is more likely to affect larger, elongated structures, the extremities of which are more likely to reach the arena edges during a recovery attempt.

To avoid bias, runs in which repair was not successful are omitted from the recovery time statistics used in this section. As are the results from runs in which a fault was introduced into a 2-module structure. The reason being that, when a fault is introduced into a structure containing 2 modules, the two recovery strategies are indistinguishable and both respond simply by removing the failed module and restarting assembly.

Figure 6.23 shows the number of independent structures present during typical runs of experiments involving configurations 7A (a), 10A (b) and 12A (c). Each figure shows the outcome from a single run involving the self-repair strategy and a single run involving the naive recovery strategy. In order to compare the two approaches, in each figure, runs were selected in which the fault was introduced at a similar point in time,

when the structure had reached a similar stage of assembly. In all three cases, after the introduction of a fault, the number of structures from the run involving the naive approach is observed to increase to a maximum as the structure is disassembled and then gradually decrease to a minimum as the structure is rebuilt. In the runs involving the self-repair strategy, the number of structures still increases as the structure is partially disassembled. However, since the self-repair strategy does not need to revert as far as the naive approach, it is able to restart assembly from a more advanced position and consequently is able to complete assembly faster.

In figure 6.24, the assembly time of the self-repair strategy is compared with that of the baseline and naive recovery experiments. Each boxplot shows the results from 100 independent runs. For each of the seven hand designed structures, the same trend is observed. The baseline experiment is always the fastest, followed by the self-repair strategy and finally the naive recovery approach. For each of the seven structures, using a Wilcoxon rank-sum test with a significance level of 0.05, it can be stated that there is a significant difference between the assembly time of all three systems.

In figure 6.25, the recovery time of the self-repair strategy is compared with that of the naive recovery approach. Runs in which a fault was introduced into a two module structure are omitted and from the remaining runs a random sample of size 70 was selected. The same trend is observed for each of the seven structures. The recovery time of the self-repair strategy is always observed to be lower than that of the naive approach. Furthermore, the difference between the two approaches appears to increase as the size of the target structure increases. For structure 7A, using a Wilcoxon rank-sum test with a significance level of 0.05, it can be stated that there is no significant difference between the recovery time of the self-repair and naive approaches. However, for the remaining six structures, using a Wilcoxon rank-sum test with a significance level of 0.05, the difference in performance can be said to be significant.

It was noted that, in figure 6.25, as the size of the hand designed structures was increased, the difference between the recovery time of the naive and self-repair approaches also increases. In figure 6.26, data from the experiments involving randomly generated structures is added and the median recovery time is plotted against the size of the target structure involved in each experiment. The graph shows that, with a Pearson correlation coefficient value of 0.75, there is a strong positive correlation between the structure size and the recovery time of the naive strategy. Meanwhile, in the self-repair experiment, which obtains a Pearson's r value of 0.24, there is a weak positive correlation. These results indicate that, as the number of modules in a structure is increased, the recovery time of the naive strategy increases at a much greater rate than that of the self-repair strategy.

In section 6.3.3 it was suggested that the self-repair strategy may perform better on structures which contain repeating patterns. In section 6.4.1, a metric referred to as the 'repair potential' was introduced for categorising structures based upon their shape of their underlying graph representation. Figure 6.27 plots the repair potential of the 47 different structures investigated in this section against the median recovery time of the self-repair strategy. The hand designed structures are shown as black points,

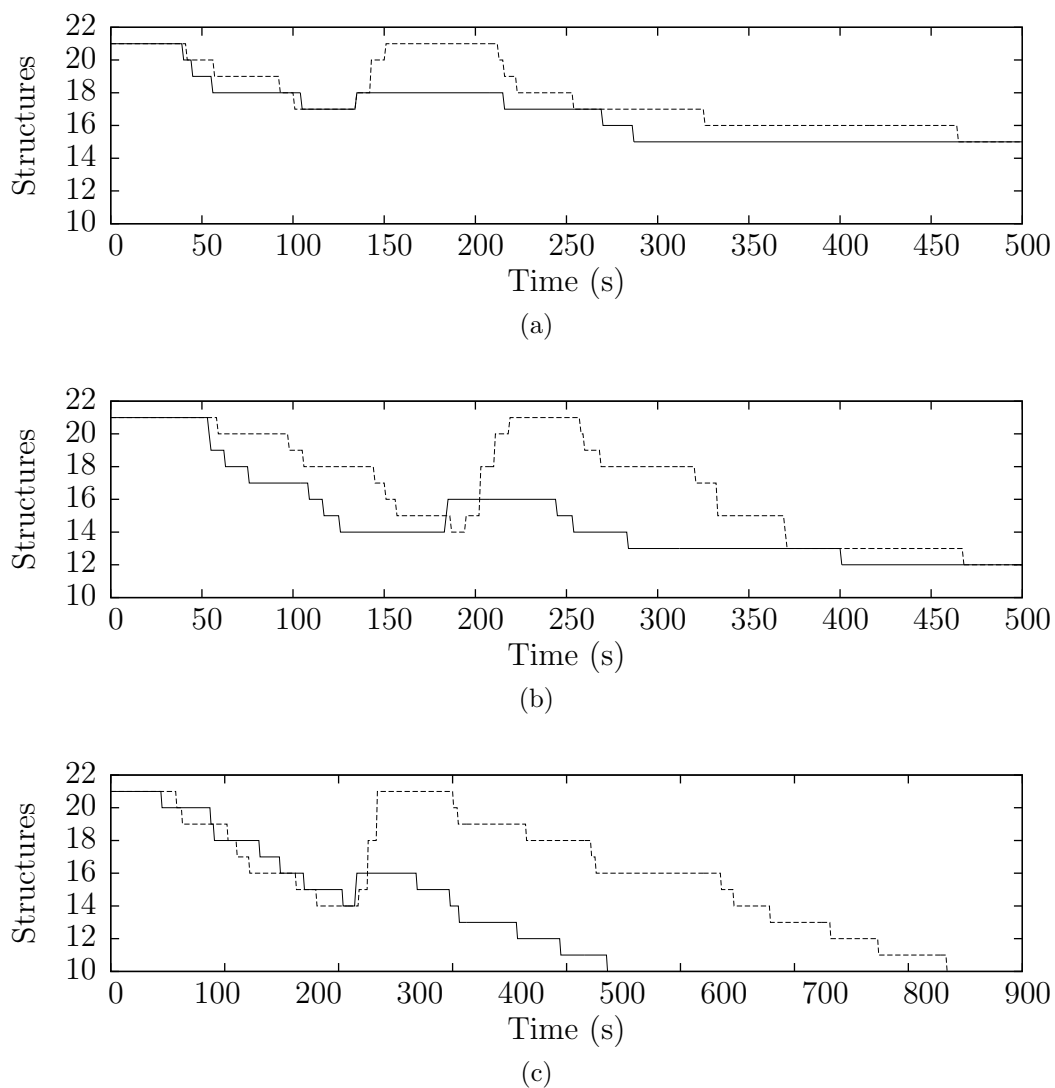


Figure 6.23: A comparison between the number of separate structures present during the assembly of shapes 7A (a), 10A (b) and 12A (c). Each line represents a single run of the naive recovery approach (dashed line) or the self-repair strategy (solid line)

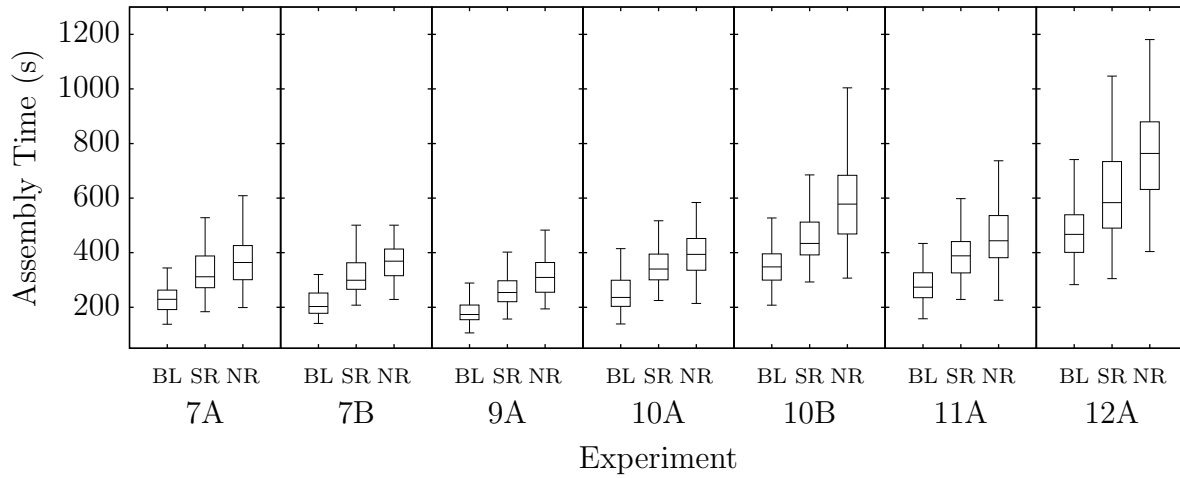


Figure 6.24: The assembly time during the baseline (BL), self-repair (SR) and naive recovery (NR) experiments

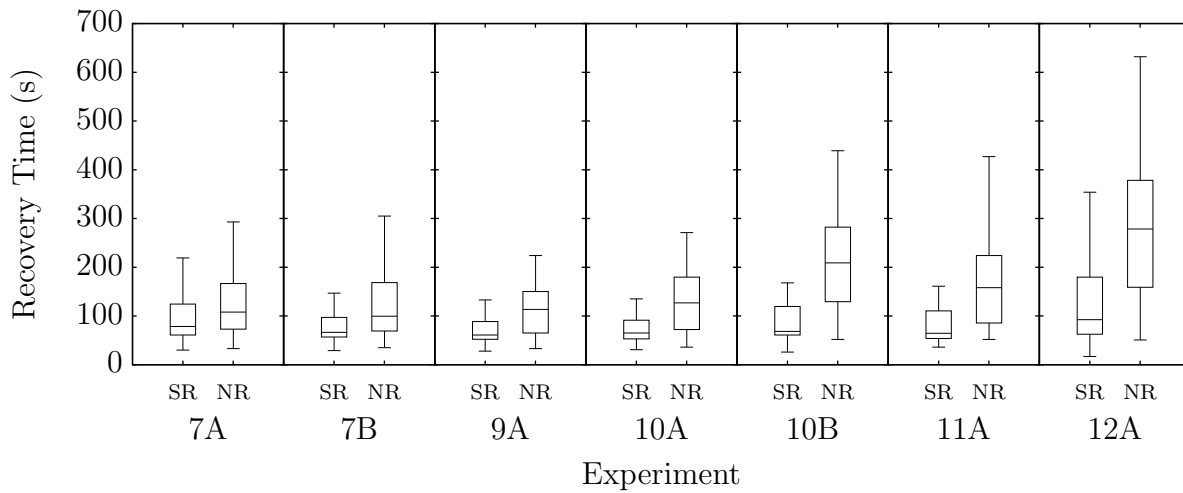


Figure 6.25: The time spent recovering during the self-repair (SR) and naive recovery (NR) experiments

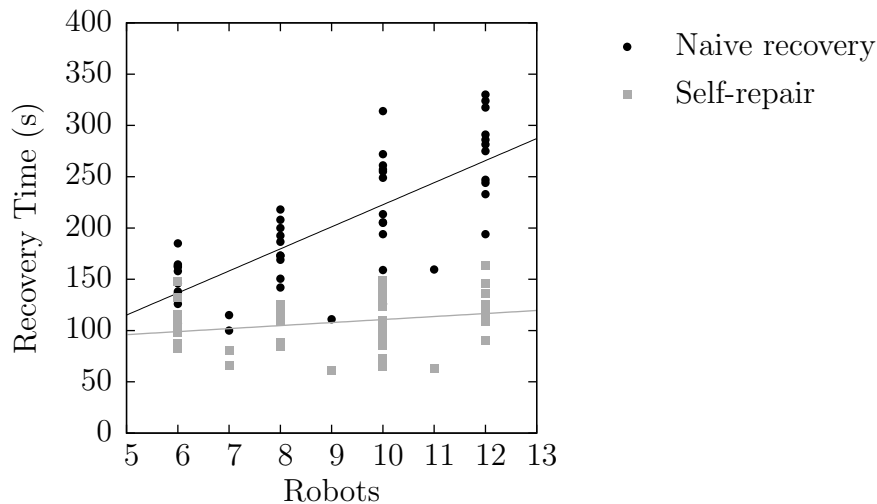


Figure 6.26: The recovery time during the naive (black) and self-repair (grey) experiments plotted against the number of robots in the target structure

whilst the randomly generated structures are shown in grey. It can be observed that, in general, the hand designed structures have a greater repair potential score and a lower recovery time than the randomly generated structures. When considering both types of structure together, it can be stated that, with a Pearson's r value of -0.75 , there is a strong negative correlation between the repair potential and the recovery time of the structures. Which is to say, when utilising the self-repair strategy, structures which have a high repair potential tend to have a low recovery time.

6.4.3 Analysis

In this section, the results from the Robot3D experiments are analysed and each of the five hypotheses introduced in section 6.4.1 is examined.

The results strengthen the conclusions from section 6.3 and show that the self-repair strategy is capable of outperforming a naive recovery approach both in terms of the overall time taken to assemble the target structure and the time spent recovering. Furthermore, whilst the recovery time of both approaches is shown to increase as the size of the structures involved increases, the recovery time of the self-repair approach does so at a much lower rate. It was also demonstrated that other structural properties may also affect performance and a strong correlation was found between the 'repair potential' of a structure and the time spent in recovery.

Whilst presenting the results, a deficiency of both the self-repair and naive recovery strategies was highlighted. Following the introduction of a failure and during the re-assembly of a robotic structure, if the structure is located too close to an obstacle, it was observed the assembly process can stall. Akin to effect E_1 , from section 4.2.2. To resolve this problem, in the future, it will be necessary for the recovery strategy to

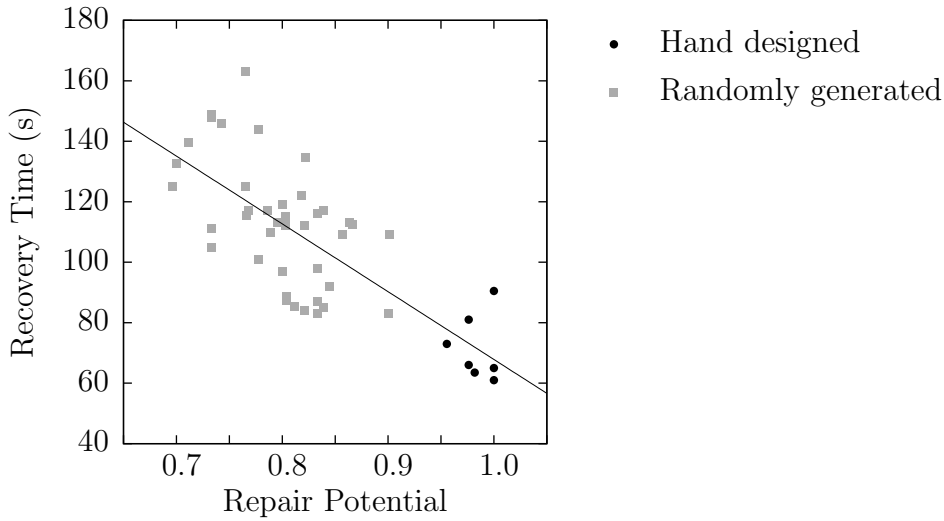


Figure 6.27: The recovery time during the self-repair experiment plotted against the repair potential of the target structure. Results are included for both hand designed (black) and randomly generated (grey) structures

incorporate better awareness of the surrounding environment.

Returning to the hypotheses introduced in section 6.4.1. In comparing the assembly time of a system which utilises the self-repair strategy and one which uses the naive recovery approach. It can be stated that there is a significant difference between the two systems, and therefore that it is possible to reject $H6.3_0$ and accept that the self-repair strategy outperforms the naive recovery approach in terms of assembly time.

In comparing the time spent recovering, it can be observed that, with the exception of structure 7A, there is a significant difference between the performance of a system utilising the self-repair strategy and one utilising the naive recovery approach. Therefore, excluding structure 7A, it is possible to reject $H6.4_0$ and state that the self-repair strategy outperforms the naive recovery approach in terms of recovery time.

As the number of robots within a structure is increased, the recovery times of both the self-repair and the naive recovery approaches are also observed to increase. The rate of increase, however, is much greater with the naive recovery strategy. When evaluated using the Pearson correlation coefficient, only in the naive approach can there be said to be a strong correlation between the size of the structures and the time spent recovering. Therefore, whilst it is possible to reject $H6.6_0$ and state that there is a correlation between the recovery time of the naive strategy and the size of the structures involved, it is not possible to reject $H6.5_0$. Until more data is available it must be accepted that there is no correlation between the recovery time of the self-repair strategy and the size of the structures involved.

Finally, when considering the repair potential of the 47 structures introduced in section 6.4.1, it can be observed that the potential of the hand designed structures is consistently greater than that of the randomly generated structures. Furthermore,

the recovery time of the hand designed structures tends to be lower. By plotting the repair potential against the recovery time and evaluating the results using the Pearson correlation coefficient, it is possible to state that there is a strong negative correlation between the two values. Based upon this evidence it is possible to reject $H_{6.7_0}$ and state that there is a correlation between the recovery time of the self-repair strategy and the repair potential of the structures under study.

6.5 Real Robot Experiments

This section presents results from experiments conducted using the real Symbricator hardware. Due to limitations in the availability and capabilities of the hardware, it was not possible to fully replicate the broad range of experiments that were carried out in simulation. Instead, as a proof of principle, four specific scenarios were designed to showcase the main features of the self-repair strategy.

In the remainder of this section, the general experimental setup used throughout all of the experiments is introduced, along with specific details of the four chosen scenarios. Following which, the results from the experiments are presented and analysed.

6.5.1 Experimental Setup

All of the experiments reported in this section use at most one Active Wheel, and either two or three Scout modules. In every scenario, the robots begin in a disconnected state, before assembling into a single structure. Following assembly, an artificial fault is injected into one member of the structure. It is assumed that the faulty robot is able to immediately detect the fault, and that it does not require assistance in removing itself from the structure. After the faulty robot notifies its neighbours, the self-repair strategy is executed.

Due to the limited number of robots, as soon as the faulty module has removed itself from a structure it is no longer considered to be faulty, and if required, may form part of the new structure. Batteries were not available and therefore all of the experiments were conducted whilst the robots were tethered to a power supply. The necessity to tether the robots restricted their movements and therefore, to prevent the robots wires from becoming tangled, every scenario was set up so that robots were positioned within communication range of the module that they were required to dock with. The robots were forced to remain stationary whilst in the Wander state and only started to move upon entering the LocateBeacon state. This restriction greatly simplifies the aligning and docking procedures, but has little bearing upon what is the main focus of this chapter, the self-repair strategy itself.

For each scenario, ten independent runs were conducted in an arena measuring approximately 165×120 cm. An overhead camera, mounted directly above the arena, was used to record the experiments and track the positions of the robots over time. Each one of the four scenarios is now described.

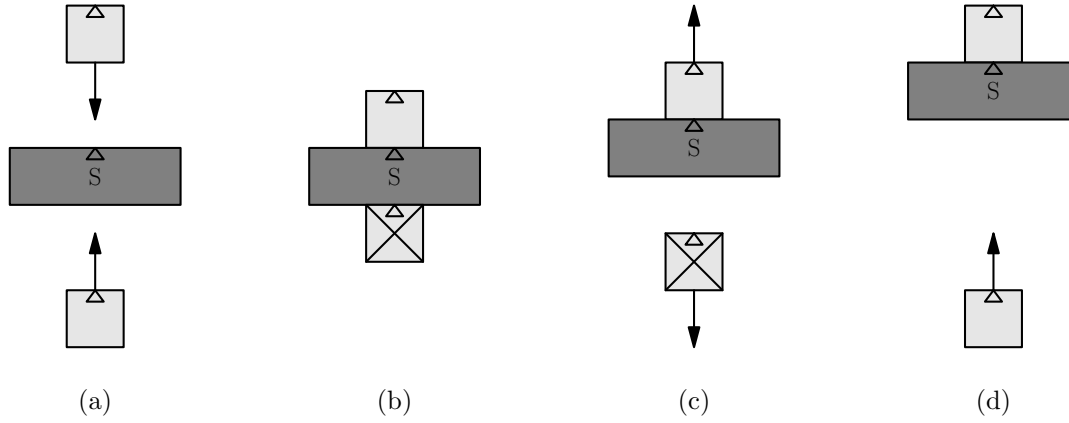


Figure 6.28: A diagram outlining scenario A. The *Seed* robot is marked with an ‘S’ and the failed robot is marked with a cross. Solid arrows are used to represent the movement of modules

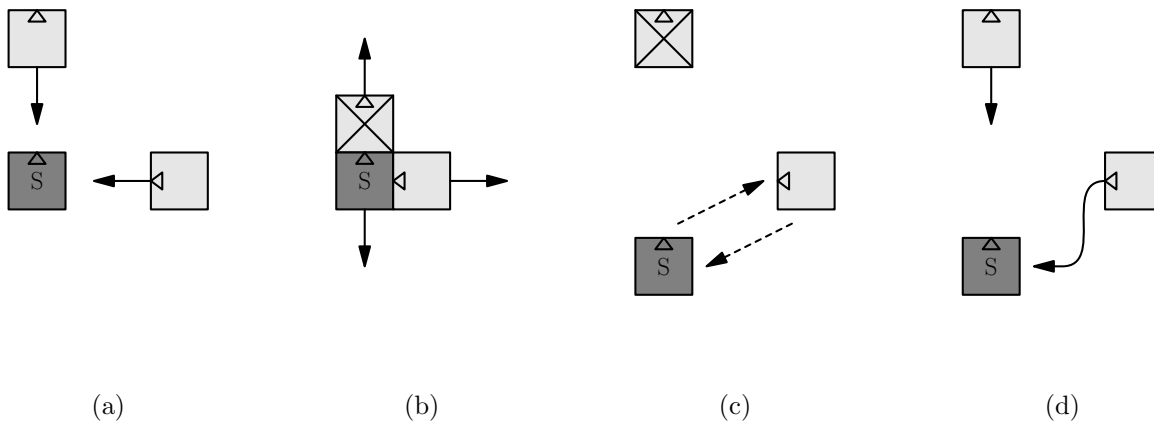


Figure 6.29: A diagram outlining scenario B. The *Seed* robot is marked with an ‘S’ and the failed robot is marked with a cross. Solid arrows are used to represent the movement of modules and dashed arrows are used to represent communication

Scenario A The first scenario is visualised in figure 6.28. This scenario demonstrates the coordinated movement of a 2-robot sub-structure and the removal and replacement of a failed module. The scenario involves two Scouts and one Active Wheel. As shown in figure 6.28a, the Active Wheel serves as the seed and begins by recruiting one Scout module on each side. One of the Scout robots then fails (figure 6.28b) and removes itself from the structure, whilst the remaining two-module sub-structure retreats (figure 6.28c). By default, the sole sub-structure obtains the highest score and then restarts the assembly process by recruiting the previously failed module (figure 6.28d). To demonstrate robustness, in each experimental run this procedure is repeated four times.

Scenario B Scenario B is shown in figure 6.29. This scenario was primarily designed to demonstrate the adaptation described in section 6.2.1 which accounts for the fact that Scout modules cannot move sideways. However, it is also used to demonstrate the negotiation between two separate sub-structures. As shown in figure 6.29a, the scenario involves three Scout robots and begins with the formation of an ‘L’ shape. One of the modules then fails and removes itself from the structure. Due to their relative orientations, the remaining two modules are forced to form two separate sub-structures (figure 6.29b). The repair process continues with two the sub-structures broadcasting their own score and ID (figure 6.29c). Since both structures contain only a single module, the individual with the lowest ID is declared the winner and takes responsibility for restarting the assembly process (figure 6.29d).

Scenario C An outline of scenario C is shown in figure 6.30. This scenario is used to demonstrate the fact that, following self-repair, robots may take on different roles within the new structure. Furthermore, the fact that it is sometimes necessary to prune modules which do not belong to the new structure is also demonstrated. Like scenario A, a three module structure containing two Scout robots and one Active Wheel is formed. However, as shown in figure 6.30a, in this case, the seed is a Scout robot. After forming a structure, the seed module fails, causing the structure split and the seed to be removed (figure 6.30b). By default, the remaining Scout becomes the new seed, however, since according to the original plan, the Active Wheel is docked using the wrong side, this module is instructed to remove itself (figure 6.30c) before assembly can continue (figure 6.30d).

Scenario D Scenario D is visualised in figure 6.31. This scenario involves four robots and demonstrates collective sub-structure locomotion and inter sub-structure negotiation. As shown in figure 6.31a, a structure containing three Scouts and one Active Wheel is assembled. The Active Wheel then fails, leading to the formation of two sub-structures (figure 6.31b). After negotiation (figure 6.31c), the larger of the two sub-structures is declared the winner and begins to reassemble the original structure (figure 6.31d).

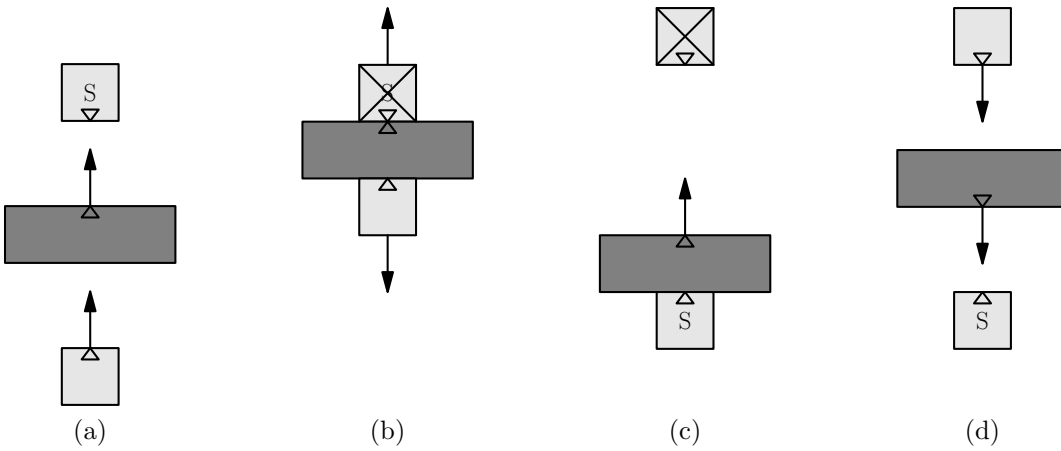


Figure 6.30: A diagram outlining scenario C. The *Seed* robot is marked with an ‘S’ and the failed robot is marked with a cross. Solid arrows are used to represent the movement of modules

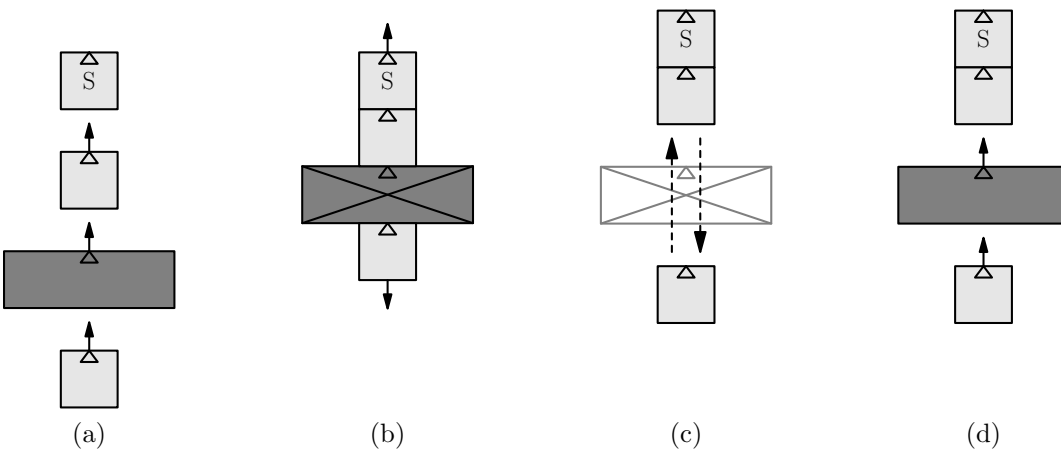


Figure 6.31: A diagram outlining scenario D. The *Seed* robot is marked with an ‘S’ and the failed robot is marked with a cross. Solid arrows are used to represent the movement of modules and dashed arrows are used to represent communication

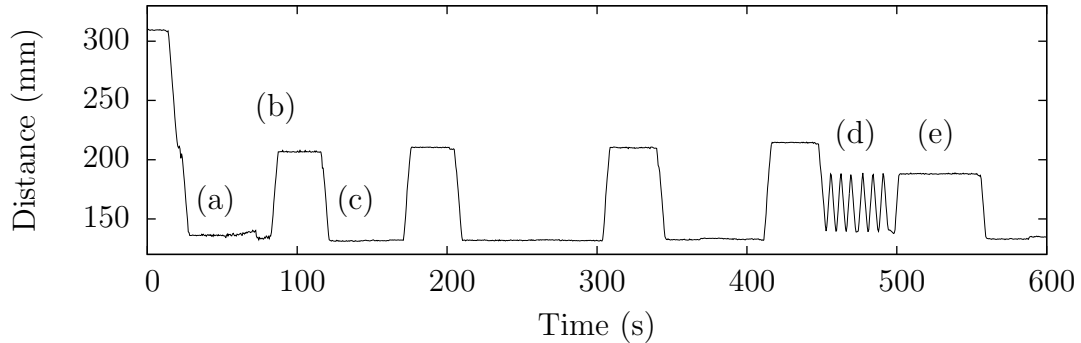


Figure 6.32: The average distance separating three robots during scenario A

6.5.2 Results

The results from the experiments performed using each of the four scenarios introduced above are now presented. Videos of all of the experiments described in this section are provided in the accompanying material (videos 6.4-6.7).

Scenario A In seven of the ten runs that were performed using scenario A, the robots were able to successfully assemble into a single structure and repeatedly self-repair following the introduction of four consecutive failures. In the remaining three runs, a problem with the logic of the recruiting behaviour prevented the seed robot from recognising that another robot had docked with it, leading to a stall in assembly.

To provide an example of the typical behaviour of the robots, figure 6.32 shows the average distance separating the three modules over time, during one of the successful runs. At the beginning of the run, when all three robots are disconnected, the distance is at its largest. At point (a) in figure 6.32, all three robots have assembled into a single structure and the distance between them is at its minimum. At point (b) a fault is introduced into one of the robots, causing the structure to split and the average separating distance to increase. At point (c) the structure is repaired and the distance once again is minimal. This process repeats two more times until point (d) is reached. At point (d), the docking approach is initially unsuccessful and requires several attempts, as shown by the rapid fluctuations in distance. Eventually, after a brief pause at point (e) whilst waiting for the recruiting robot, the structure is repaired for the final time.

Scenario B In all ten of the runs that were performed using scenario B, the robots were able to self-assemble into a single structure. Following the introduction of a failure, in nine of the ten runs, the robots successfully identified the fact that one of the Scout modules would not be able to move in the required direction and therefore should be pruned. After splitting into two separate sub-structures and negotiating which sub-structure was best placed to continue assembly the robots were able to successful repair

the original structure. The one case in which self-repair was not successful resulted from one of the robots pushing and displacing the seed as it attempted to dock with it, thereby preventing the other module from locating it. Pushing was also observed in three of the successful runs but, in these cases, was not sufficient to disrupt the self-repair process.

Figure 6.33 shows four composite images taken from a video of a single experimental run. The modules are arbitrarily labelled with a number by the tracking software and green links signify when two modules have docked. Note that the robot's labels do not necessarily correspond to their IDs. In figure 6.33a, the final module is shown docking with the other two in order to complete the assembly of the original structure. In figure 6.33b, module 2 has failed, this event is detected by module 1 which subsequently retreats. Since module 0 is oriented perpendicular to module 1, it is instructed to form a separate sub-structure and also retreats. In figure 6.33c, modules 0 and 1 are broadcasting and listening for each others scores. Both modules have the same score, but in this case module 1 has a lower ID and therefore, as shown in figure 6.33d, takes responsibility for rebuilding the structure.

Scenario C In all ten of the runs that were performed using scenario C, the robots were able to self-assemble into a single structure. After a failure was introduced into the original seed, in nine of the ten runs, the robots successfully selected a new seed and identified the fact that the Active Wheel was docked using the wrong side. In the one case which failed at this point, an undiagnosed problem with the Active Wheel prevented it from removing itself when instructed. Out of the nine runs which reached this stage, eight went on to successfully complete the self-repair process. In the remaining run, assembly stalled when one of the modules was unable to align itself properly and, whilst attempting to retry, subsequently moved outside of communication range of the recruiting module.

Figure 6.34 shows the removal and re-docking of the Active Wheel during one of the successful runs. In figure 6.34a, the *Seed* robot (bottom) is shown instructing the Active Wheel (middle) to remove itself from the structure whilst the failed module (top) waits. In figure 6.34b, the *Seed* robot is shown sending recruiting messages on the side at which the Active Wheel was previously docked. Upon receipt of these messages, in order to orientate itself correctly, the Active Wheel is shown to begin turning. In figure 6.34c, the Active Wheel is shown aligning with the *Seed* and finally, in figure 6.34d, the previously failed Scout module is shown to approach the new structure.

Scenario D In eight out of the ten runs that were performed using scenario D, the robots were able to successfully assemble into a single structure. In all eight cases, following the introduction of a failure, the robots were then able to split into two separate sub-structures, negotiate which was best placed to restart the assembly process and finally, repair the original structure. In the final two runs, a real hardware fault in the IR remote receiver prevented the runs from being completed.

Figure 6.35 shows still images from a video of one of the successful runs. In figure

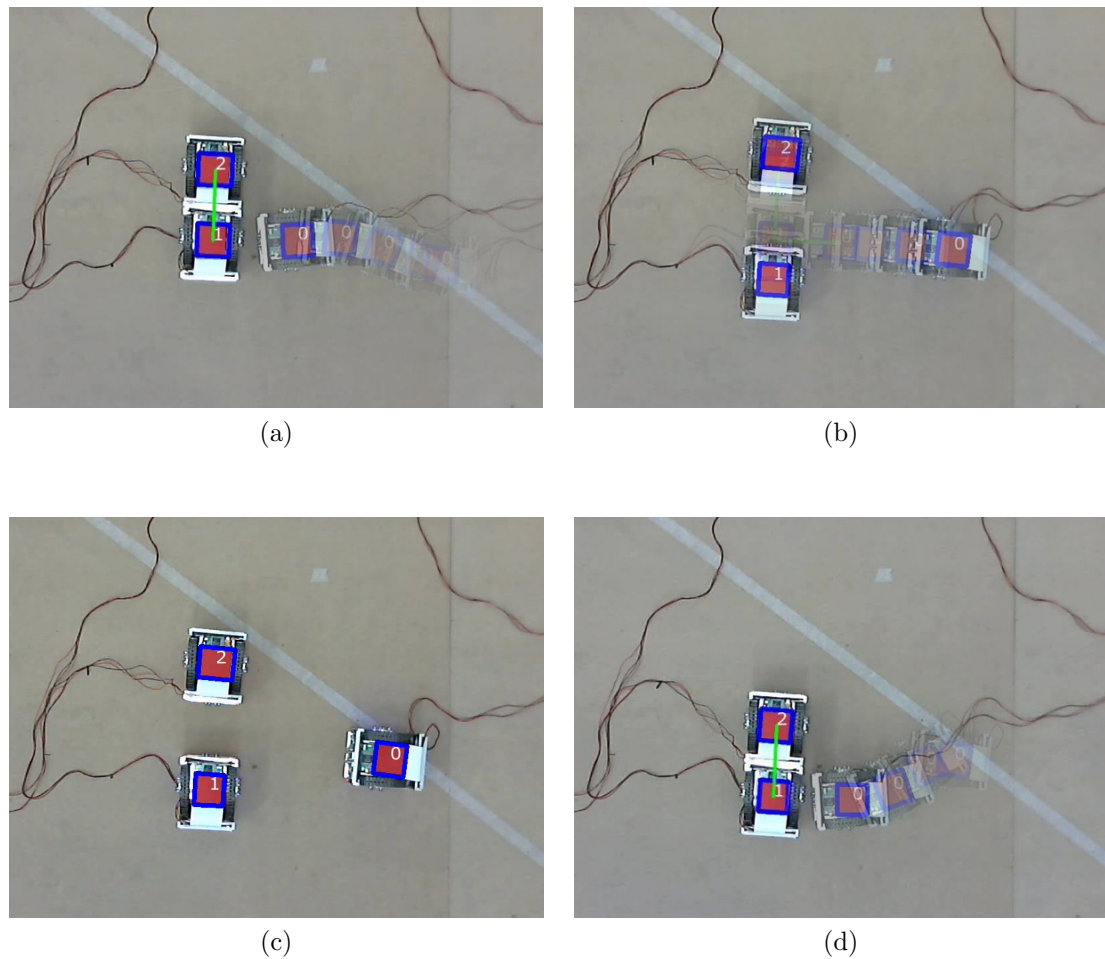


Figure 6.33: Composite images from a video of scenario B with information from the tracking software overlaid

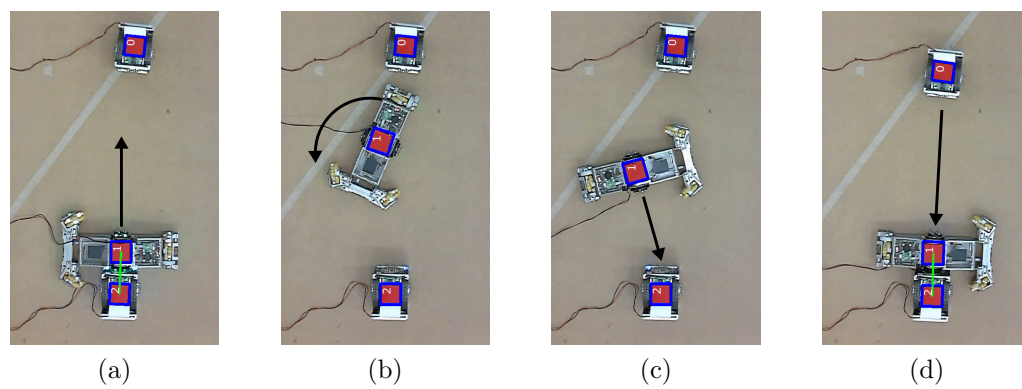


Figure 6.34: Still images from a video of scenario C. Solid black arrows are used to visualise the movement of the individual modules

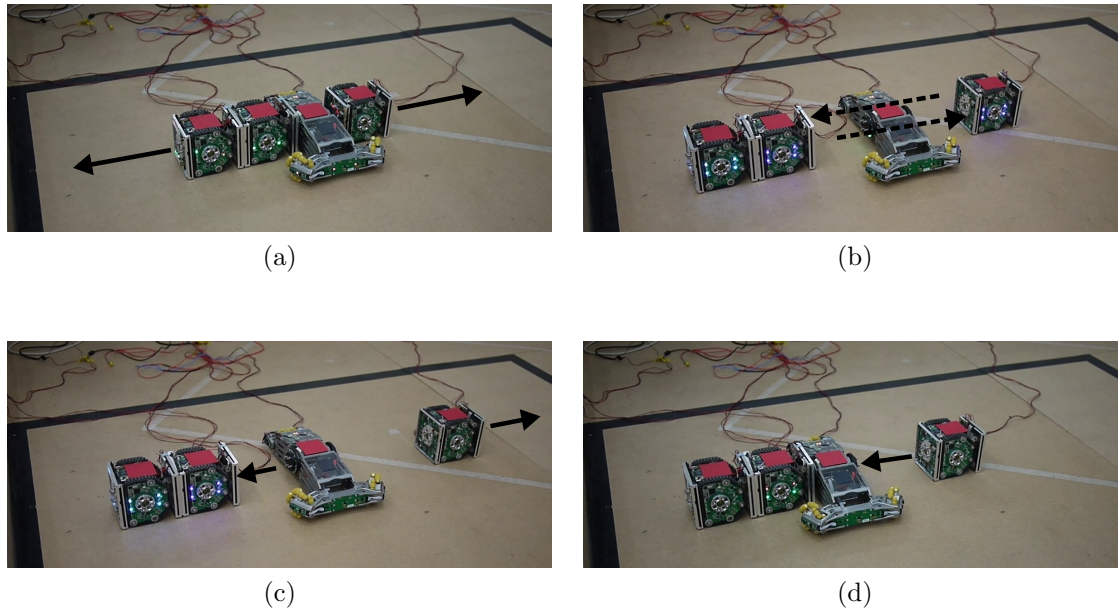


Figure 6.35: Still images from a video of scenario D. Solid black arrows are used to visualise the movement of the individual modules and dashed arrows are used to represent communication

6.35a, a fault is introduced into the Active Wheel and the structure is shown to split into two sub-structures. In figure 6.35b, the two sub-structures are shown negotiating. The larger of the two sub-structures is declared the winner and in figure 6.35c is shown to restart the assembly process whilst the losing sub-structure is ‘disassembled’. In figure 6.35d, the final module is recruited and the structure is completed.

6.5.3 Analysis

The main purpose of the experiments described in this section was to demonstrate the implementation of the self-repair strategy on-board real robotic hardware. Scenarios were chosen to reflect some of the main components of the self-repair strategy. Specifically, scenario A was used to demonstrate the coordinated movement of multi-robot sub-structures; scenario B was used to show how sub-structures containing modules that cannot move omnidirectionally are pruned and how negotiation takes place between two separate sub-structures; scenario C was used to demonstrate the fact that, following self-repair, modules may take on different roles within the new structure and that it is sometimes necessary to remove modules which do not belong to the structure; finally, scenario D was used to further showcase the collective locomotion of sub-structures and inter sub-structure negotiation.

Ten repeated runs were performed with each scenario. As summarised in table 6.3, in 32 out of the 40 runs the robots were able to successfully complete the self-repair process.

	<i>Success</i>	Recruitment	Alignment	Hardware	Unknown
A	7	3	-	-	-
B	9	-	1	-	-
C	8	-	1	-	1
D	8	-	-	2	-

Table 6.3: The number of successful runs and the reasons for failure during unsuccessful runs, for each of the four scenarios

Table 6.3 also summarises the reasons for why self-repair was not successful in the eight remaining runs. Three of the runs failed due to a problem with the Recruitment state from the original morphogenesis controller, two failed due to problems with the Alignment behaviour, two could not be completed due to hardware failures and one run failed due to an undiagnosed problem.

It is important to highlight that none of the eight incomplete runs resulted from problems with the self-repair strategy itself (with the possible exception of the undiagnosed failure). Furthermore, if the system had possessed a method for detecting hardware failures, and there were a sufficient number of spare robots, the two runs which were halted by hardware problems may also have been completed. The fact that hardware failures occurred in two of the runs highlights the need for a self-repair strategy, but the fact that the runs could not be completed, emphasises the fact that without an effective fault detection system any form of explicit recovery strategy is useless.

Even with a fault detection system, the five runs which could not be completed due to problems with the Recruitment and Alignment behaviours would still have stalled. This highlights the importance of designing reliable and functionally correct controllers but also serves as a reminder that control system failures may be equally as detrimental as hardware failures. In the reliability analysis performed in chapter 4 only hardware related hazards were considered. Furthermore, the anomaly detection system introduced in chapter 5 was designed only to detect hardware failures. To further improve the reliability of self-reconfigurable modular robotic systems, the effects of control system failures, and methods for detecting their presence, should also be studied.

6.6 Summary and Future Work

In this chapter, a new self-repair strategy was presented which aims to improve the fault tolerance of the morphogenesis controller introduced in chapter 4. The strategy may be used to repair assembled or partially assembled robotic structures containing a single failed module. The strategy works by isolating the failed module, removing it from the structure and re-assembling the structure in the most efficient manner. Local communication is used by the modules to determine the shape of the structure that they belong to and to coordinate which modules should take part in the recovery process.

To address one of the issues highlighted in chapter 4, inter-module communication is performed using two separate channels, increasing the reliability of the approach and removing a potential single point of failure. The generality of the approach was demonstrated in its deployment on all three of the Symbicator platforms, both in simulation and using real robotic hardware.

The first set of experiments were performed with the Stage simulator, using simulated Backbone modules. The performance of the self-repair approach was compared with a baseline system in which no faults were injected and a naive recovery strategy in which, following the introduction of a fault, the structure is completely disassembled. In this set of experiments, two different structural configurations were investigated. In every case, the robots were able to complete the assembly process and were consistently successful in isolating, removing and replacing the failed module. The self-repair strategy was shown to outperform the naive approach in the experiment involving the larger of the two structures but, with the smaller structure, no significant difference in performance was observed.

In analysing the results from the Stage experiments, the shape and size of the structures involved was highlighted as an important factor in how well the self-repair strategy would perform. To investigate the properties of the self-repair strategy further, several more experiments were performed using the more realistic Robot3D simulator. Once again, Backbone modules were simulated, however, in this set of experiments, a wider range of structural configurations were investigated. Specifically, seven hand designed structures and 40 randomly generated structures were used. To aid analysis, a metric referred to as the ‘repair potential’ was introduced. The metric was calculated by iteratively comparing the graph representation of a structure with each of its sub-graphs and accumulating a score based upon their similarity.

The self-repair strategy was shown to significantly outperform the naive recovery approach in terms of both the overall assembly time and the time spent recovering. In the majority of cases, both strategies were able to successfully recover from the introduction of a failure, but were occasionally prevented from doing so if the target structure was assembled too close to the arena wall.

For each of the 47 different structural configurations, 100 independent runs were performed. In analysing the results from these runs, there was observed to be a strong positive correlation between the recovery time of the naive approach and the number of robots in the target structure. That is to say, as the number of robots was increased, the recovery time of the naive strategy also tended to increase. Whilst the recovery time of the self-repair strategy also increased in line with structure size, it did so at a much lower rate than the naive approach. In analysing the repair potential of the different configurations, it was observed that the hand designed shapes tended to have both a higher repair potential and a lower recovery time. Generally, a strong negative correlation was found between the repair potential and recovery time. As the repair potential increased, the recovery time tended to decrease.

In the final set of experiments, the self-repair strategy was deployed on the physical Symbicator hardware. Experiments were performed using both the Active Wheel and

Scout modules. Four typical scenarios were designed to showcase some of the main components of the strategy. Ten repeated runs were performed with each scenario and in the majority of cases the robots were able to successfully complete the repair process. On two of the occasions in which repair was unsuccessful, hardware failures were to blame. These failures highlighted the importance of the self-repair strategy itself but also emphasises the need for effective fault detection systems such as that introduced in chapter 5. In five of the unsuccessful runs, repair was unsuccessful due to problems with the behavioural controller and the nature of the physical hardware. Problems such as this, which were overlooked in the reliability analysis study performed in chapter 4, highlight the importance of verifying algorithms on real robotic hardware and designing reliable and functionally correct controllers.

In future work, further experiments may be performed with both the physical Symbricator hardware and within the the Robot3D simulator. With the physical hardware, similar experiments to those conducted in simulation may be performed, whilst in simulation, further analysis of the structural properties favoured by the self-repair strategy may be conducted. To gain further insight, mathematical analysis of the system dynamics may also be performed. To address some of the limitations discussed in this chapter, improvements may be made to the strategy itself. For example, the potential for having multiple robots dock as a combined unit could be investigated. To improve the efficiency of the approach and prevent unnecessary pruning, alternative methods of calculating the sub-structure score could also be developed. Furthermore, to prevent the scenario in which assembly is stalled when a structure is positioned too close to an arena wall, the robots could utilise information about the state of the environment when determining which module should restart the assembly process. Investigations into the suitability of the current strategy for handling multiple failures, and the development of any necessary extensions to handle such scenarios would also be interesting.

In the following chapter, a new form of morphogenesis is introduced and several more experiments are conducted. A different physical platform is utilised and the focus shifts from explicit self-repair to implicit forms of self-assembly and self-reconfiguration.

CHAPTER 7

Self-assembling and Self-reconfiguring Robotic Structures

In chapter 4 it was suggested that, by following the example of swarm robotics, the fault tolerance of self-reconfigurable modular robotic systems could be improved if greater plasticity was afforded to the conformation of systems. To provide such plasticity, in chapter 6, a self-repair strategy was presented which allows faulty modules to be removed from a robotic structure and replaced by functional ones. The strategy was successful in improving the fault tolerance of the system, but scenarios in which the dependence between connected modules could still cause problems were also identified. For example, if a module failed whilst locked to another functional robot, and it was not possible to remove the failed module, then it would be necessary to also sacrifice the functional robot in order to repair the system.

In chapter 1, the disparity between the availability of swarm and self-reconfigurable modular robotic platforms was also highlighted. This disparity may, at least partially, be attributed to the differing complexity of the required hardware. Swarm robots are purposefully simple units, whereas modular robots, although simple in comparison to the structures that they may form, require complex electrical and mechanical hardware to facilitate the processes of docking, reconfiguration and inter-robot communication.

To address these issues, in this chapter, the design of a new structural extension is presented for the e-puck robot [122]. The e-puck is a small mobile robot, equipped with basic sensors and actuators, including a colour camera, a ring of LEDs and an array of infrared sensors. The extension described in this chapter, hereafter referred to as the *modular e-puck extension*, consists of a plastic frame and four passive magnetic docking interfaces. The extension allows a group of e-pucks to physically join with one another, transforming what is traditionally a swarm robotics platform into a 2D self-reconfigurable modular robotic system. By combining benefits from swarm and self-reconfigurable modular robotic systems, the modular e-puck extension provides a simple, low-cost platform, with a high-degree of plasticity, that may be used to

investigate the interesting properties of self-assembling and self-reconfiguring systems.

Later in this chapter, an algorithm for controlling the collective locomotion of a group of robots equipped with the modular e-puck extension is also introduced. As well as demonstrating collective locomotion, without alteration to the underlying algorithm, robots are shown to be capable of demonstrating a novel form of implicit self-assembly which allows a group of robots to reform when broken apart. After investigating how the performance of the collective locomotion behaviour is affected when using structures of different shapes and sizes, the properties of the system which lead to the exhibition of self-assembly are analysed in detail. Finally, by combining self-assembly and collective locomotion with a new form of self-disassembly, the task of self-reconfiguration under changing environmental conditions is also investigated.

The remainder of this chapter is structured as follows. In section 7.1, comparisons are made with existing platforms and related control strategies. In section 7.2, the modular e-puck extension itself is introduced. In section 7.3, the collective locomotion algorithm is described. In section 7.4, the common setup used during all of the experiments is detailed. In section 7.5, the results from collective locomotion experiments are presented. In section 7.6, the self-assembling properties of the system are analysed. In section 7.7, a form of environment driven self-reconfiguration is investigated. Finally, in section 7.8, a summary is provided and potential areas of future work are discussed.

7.1 Comparisons

The modular e-puck extension may be used to transform the e-puck robot into a mobile-lattice self-reconfigurable modular robotic system. Each module is independently mobile but may connect with up to four other neighbours within a two-dimensional grid. A large number of lattice and mobile self-reconfigurable robots have been developed, several of which were reviewed in chapter 3. In this section, some of the similarities and differences between the modular e-puck extension and related platforms are highlighted. Following this, some of the previous approaches to controlling the locomotion, assembly, and reconfiguration of such systems are reviewed.

7.1.1 Platforms

One of the most important factors in the design of a self-reconfigurable modular robotic system, is the method by which the individual robots connect with one another. Several different methods have been proposed for connecting mobile and lattice modular robots. Platforms such as Symbricator [88], the Swarm-bot and Swarmanoid robots [40, 121], Metamorphic [143], ATRON [80] and Sambot [191], rely on purely mechanical connection mechanisms. Many other systems favour a magnetic approach, for example, the Programmable Parts [12], DFA [141], Miche [53, 54] and Telecube [175] systems all use permanent magnets. The X-Cell [76], Catom [57] and Smart Pebble systems [51, 55], on the other hand, all favour electromagnets, whilst the Fracta [124] system utilises a

combination of permanent magnets and electromagnets. The modular e-puck extension uses only permanent magnets. Mirroring the approach of the DFA platform, the strength and positioning of the magnets on the modular e-puck extension were chosen such that the connection between two modules is stable when it needs to be, but not so strong that the modules are prevented from disconnecting when it is desired.

In terms of their structure, robots equipped with the modular e-puck extension closely resemble the X-Cell modules reviewed in chapter 3. Both systems possess a separate square frame that rests on top of their main body, allowing them to form stable structures within a 2D lattice, and both systems utilise a two-wheeled differential drive system. However, two important differences between the X-Cell modules and robots equipped with the modular e-puck extension can be identified. Firstly, the positioning of the electromagnets on the X-Cell shell and the provision that no two electromagnets may connect with one another, limits the number of possible configurations that the robots may form. With the modular e-puck extension, there are no limitations upon the number of configurations that the robots can be arranged in—other than those imposed by the lattice grid. Secondly, the rotation of the X-Cell shell is controlled by a servo motor and may only move in the range of $0 - 180^\circ$. The frame of the modular e-puck extension, on the other hand, is completely independent of the e-puck robot, allowing for free rotation about the robot's body. This leads to a far more flexible system, albeit at the expense of controllability.

7.1.2 Control Strategies

When coordinating the motion of a collective robotic system, it is important that the individual robots reach a consensus regarding the goal of the collective. This may be achieved in a centralised fashion, whereby a designated leader propagates commands to the other modules, or in a decentralised manner, in which no single robot is responsible. Owing to the perceived lack of a 'single point of failure', decentralised methods of control are often favoured.

In [65], Gutiérrez et al. describe a decentralised method for synchronising the headings of a group of stationary e-pucks, which relies on exchanging relative bearings using infrared (IR) communication. In related work, Trianni et al. incorporate information from ground sensors and demonstrate coordinated motion with hole avoidance [181]. Using a similar approach, Baldassarre et al. evolve controllers for the s-bot platform which use information from the robots' traction sensors to synchronise the headings of a group of robots [7]. Baldassarre et al.'s controllers were verified on real robots, under various conditions, and although it was not their main focus, the authors also demonstrate the robot's ability to complete the related task of collective transport.

In another example that uses the s-bot's traction sensor, Groß et al. describe a solution to the collective transport task that allows a collective to efficiently transport an object to a goal location, even if one or more of the robots cannot perceive the goal [63]. In [61], Groß and Dorigo evolve controllers for a similar task but start with the robots in a disconnected state, requiring that the robots first self-assemble before performing

collective transport. Campo et al. and Ferrante et al. describe two hand-coded solutions to the collective transport task, both of which include explicit negotiation between robots [16, 45]. Ferrante et al.’s approach relies on infrared communication, whilst Campo et al. utilise the robots’ LEDs and vision systems.

The approach to collective locomotion described in this chapter builds upon that of [65]. Like [65], IR communication is used, and like both [16] and [45], explicit negotiation takes place between modules. Unlike [63], [181] and [7], in the absence of a traction sensor, the robots must determine their heading based solely on their perception of the environment and communication with their neighbours.

In [60], Groß and Dorigo review several different types of self-assembling system and classify them as either self-propelled or externally-propelled, depending upon the manner in which the individual modules move. Due to the independent mobility of the e-puck, in Groß and Dorigo’s taxonomy, the modular e-puck extension may be viewed as a self-propelled system.

Many of the systems reviewed in chapter 3 are capable of self-propelled assembly. The CEBOT [48], X-Cell [76], Sambot [191] and Symbicator robots [108], have all demonstrated alignment and docking approaches based upon the use of infrared sensors. The s-bot platform, on the other hand, has been used to develop several distributed methods of self-assembly [62, 136] and self-reconfiguration [135] which make use of the robots’ LEDs and vision systems. In [136], for example, O’Grady et al. describe a distributed approach to self-assembly in which structures are grown by recruiting new modules to specific locations, based upon local interaction rules. A common property of these approaches is that they rely on the presence of a seed module to initiate self-assembly, and in most cases ([136] excepted), require a target shape to guide the assembly process. In contrast, the approach used by the DFA system in [141] requires neither a seed nor a target. The DFA modules simply move around the arena until they collide with other robots, at which point permanent magnets and protruding features of the robot’s chassis cause them to join together. Since it negates the need for complex alignment procedures, this form of random passive docking—which is more commonly found in externally-propelled forms of self-assembly—can be beneficial.

The approach to self-assembly described in this chapter resembles that of the DFA platform. Self-assembly is not directed and does not require a seed, however, unlike the DFA approach, communication between nearby robots does produce a tendency for modules to move towards one another.

In other related work, Groß et al. describe a system of heterogeneous modules, that are capable of responding to changes in environmental conditions, whilst demonstrating self-replication and externally-propelled self-assembly [64]. Two similar tasks are described by [180] and [190], in which controllers are evolved to perform self-assembly in response to environmental triggers. Inspired by this work, in section 7.7, a similar task is used to investigate self-reconfiguration with the modular e-puck extension.

7.1.3 Summary

According to the classification of Yim et al. [207], the modular e-puck extension may be described as a mobile-lattice modular robotic system. Although, structurally, robots equipped with the extension resemble the X-Cell modules from [76], they are far more flexible in terms of the different configurations they may form. Like the Programmable Parts, DFA, Miche and Telecube systems, the extension uses a connection mechanism based upon permanent magnets, and like the DFA system, the strength of the magnets was carefully chosen to allow for both self-assembly and self-reconfiguration.

The approach to collective locomotion described here builds upon that of [65], and uses explicit negotiation in a manner similar to that of [16] and [45]. The approach to self-assembly places the system within the ‘self-propelled’ category of Groß and Dorigo’s taxonomy [60], but unlike many other such systems, the behaviour is not directed, and does not require a seed module. Neither, on the other hand, is it fully stochastic like the approach employed by the DFA platform.

7.2 Modular e-puck Extension

As an open hardware project, the e-puck robot is very well suited to modification. Over recent years, a number of extensions have been developed, including an omnidirectional vision turret [122], a range-bearing board [65], colour LEDs, a ZigBee radio module [33], and even an embedded Linux implementation [109].

In this section, an extension is described that may be used to transform the existing e-puck platform into what can be described as a mobile-lattice modular robotic system. Robots equipped with the extension remain independently mobile, but through passive magnetic docking interfaces may physically connect with other modules within a 2D grid. Whilst not possessing the same functionality as some of the more sophisticated modular robotic systems, the platform may serve as an entry point, or stepping stone, to more advanced work. Providing a low-cost alternative for investigating the interesting properties of self-reconfigurable modular robotic systems, at an abstract level.

As shown in figure 7.1a, the extension consists of three parts: a circular base plate which sits directly on top of the e-puck, a central frame which rests on top of the base plate, and a second circular cover which sits on top of the frame to help secure the extension together.

The base plate is positioned on top of the default extension board using three 15 mm hexagonal spacers. To remain compatible with other extension boards, larger spacers may be used. A small overhang on the base plate allows the inner ring of the central frame to rest on the base plate, without being permanently attached. This lip allows the frame to rotate unhindered around the central axis of the e-puck. To enable separate modules to connect with one another, two magnets are fitted on each internal edge of the central frame, with opposing poles facing outwards. The strength and positioning

¹http://www.elec.york.ac.uk/research/projects/Modular_e_puck_Extension.html

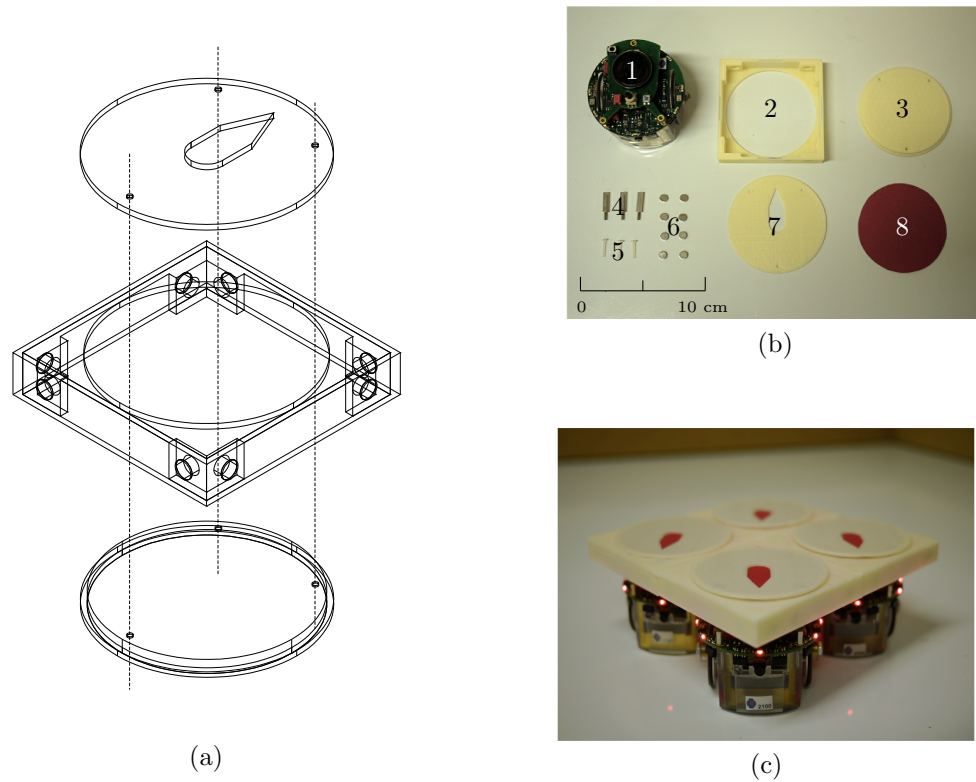


Figure 7.1: A schematic of the main structural components of the modular e-puck extension (a) and photographs of unassembled and assembled prototypes (b-c). The designs for the structural components are made freely available online¹

Part	Label	Description	Quantity	Cost (€)
e-puck	1	The e-puck mobile robot	1	-
Structure	2,3,7	3D-printed ABS plastic, ~ 25 g	1	0.90
Spacers	4	Brass/nickel stand-off pillars, M2	3	0.50
Screws	5	Nylon set screw, slotted, M2, 12 mm	3	0.20
Magnets	6	Neodymium N38 disc magnets, 6×2 mm	8	4.00
Card	8	Coloured card for tracking (optional)	1	-
Total				5.60

Table 7.1: The list of parts required to construct a single e-puck extension. The Labels in the second column reference the parts shown in figure 7.1b

of the magnets were chosen such that, if connected modules coordinate their motion, they will remain attached, but if they do not, they will break apart. This approach ensures that the extension provides a suitable platform for investigating both collective behaviour and self-reconfiguration. Screws which pass through the two circular plates secure the extension to the e-puck and an arrow shaped window in the top cover allows the current heading of the robot to be easily recognised. To aid computer tracking, a piece of coloured card may be positioned between the base plate and the top cover.

The parts required to construct a single extension are listed in table 7.1, and labelled in figure 7.1b. Figure 7.1c shows a potential arrangement of four e-pucks equipped with the fully assembled extension. The three main structural parts were fabricated using a MakerBot thing-o-matic 3D printer². The total cost is estimated to be around €5 per unit, of which the magnets take up the majority. With the exception of the e-puck itself, together, all of the parts in table 7.1 weigh less than 50 g.

7.3 Algorithm Description

In this section, an algorithm for controlling the collective locomotion of a group of e-pucks equipped with the modular e-puck extension is described. Through a behaviour-based approach, every robot in the group is motivated to move forward, to align with its neighbours and to avoid obstacles. The summation of these three objectives determines the speed of the robot's motors. Regardless of their position within the larger structure, each robot runs the same controller and uses only local communication.

Figure 7.2 shows a high level overview of the controller. The parts of the controller corresponding to the alignment, obstacle avoidance and forward motion of the robots are labelled. The inputs h_0, h_1, \dots, h_N represent the relative headings of a robot's neighbours and the positions of nearby obstacles. The outputs M_L and M_R correspond to the speed of the robot's left and right motors. The boxes labelled (7.1), (7.2) and (7.3) represent equations of the same name which are introduced in the forthcoming sections.

²<http://www.makerbot.com/support/thingomatic/>

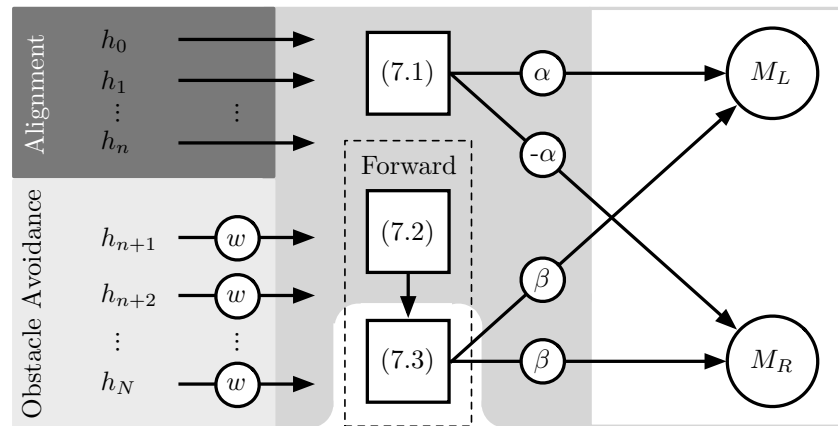


Figure 7.2: A high-level overview of the collective locomotion controller. For clarity, details of the inter-robot communication components are omitted. The alignment behaviour is introduced in sections 7.3.1-7.3.2, the obstacle avoidance behaviour is introduced in 7.3.3 and the forward behaviour is introduced in 7.3.4

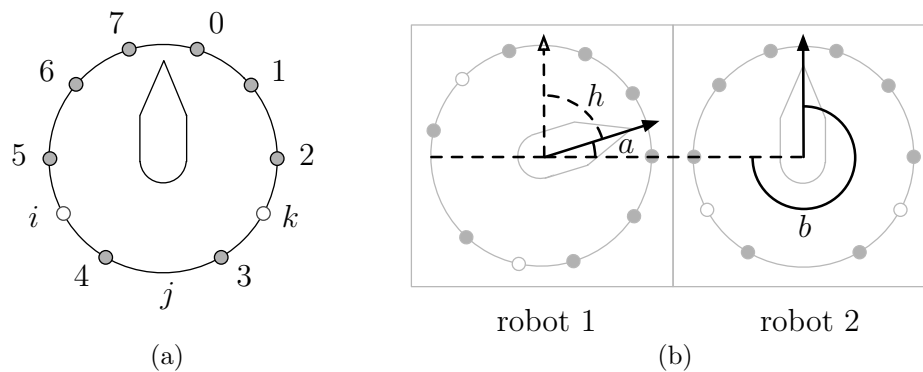


Figure 7.3: The positioning of the infrared sensors on board an e-puck robot (a) and the mechanism for exchanging relative bearings between two modules (b)

As a collective, the robots are able to exhibit continuous coordinated motion within an enclosed arena, whilst at the same time demonstrating robustness to perturbations in their overall structure. Following the removal of one or more modules, whether deliberate or accidental, the system is able to reconfigure and re-form either the original structure, or an entirely new one. This process of self-assembly is not pre-programmed but emerges due to a combination of factors including: the design of the structural extension, the design of the controller, and the nature of the robots' environment.

The controller makes extensive use of the e-puck's IR sensors. The arrangement of the eight sensors on a single e-puck is shown in figure 7.3a, the points marked 'i', 'j' and 'k' correspond to blind spots at which no sensors are present. The obstacle avoidance behaviour uses the IR sensors for proximity detection whilst, with the help of the *LibIrcom* library [66], the alignment behaviour uses them for short-range communication.

The alignment behaviour is based upon the same principle of exchanging relative bearings as both the *LibIrcorn* library’s ‘synchronize’ example, and the alignment technique described by [65]. This method of alignment, which was originally designed to synchronise the headings of stationary robots, is described shortly and hereafter referred to as *static synchronisation*. When considering non-stationary alignment, some problems with the static synchronisation approach are identified and whilst introducing a new alignment behaviour, some solutions are proposed. Following which, the obstacle avoidance and forward bias behaviours are described and how the three parts of the system are combined to produce the desired overall locomotion is detailed.

7.3.1 Static synchronisation

By exchanging relative bearings with one another, the static synchronisation approach allows a group of stationary robots to converge to and maintain a common heading. The approach relies on every robot broadcasting a unique ID and listening for the IDs of others. Based upon the sensor at which a message is received, robots are able to estimate the position of their neighbours as an angle relative to their own heading. For every ID that a robot receives, a message is sent to the corresponding neighbour, notifying it of the angle at which it was detected. As shown in figure 7.3b, using the angle at which robot 2 was detected ‘ a ’, and the angle at which robot 2 detected robot 1 ‘ b ’, robot 1 may calculate the relative heading of robot 2 as $h = a + \pi - b$. The relative heading of each of a robot’s neighbours is used to incrementally update the robot’s own desired heading, which consequently determines whether a robot should turn left, turn right, or remain stationary at each control cycle.

The approach is effective at synchronising the alignment of stationary modules, but in preliminary experiments with mobile robots, it was observed that if the robots were not able to converge to the same heading quickly enough, they had a tendency to break apart. Two potential causes of this problem were identified, both of which relate to the positioning of the sensors on the e-puck.

Firstly, because the angle between neighbouring sensors ranges from around 30° to 60° , unless two sensors are perfectly aligned, the estimate of angles a and b is often inaccurate. Although the static synchronisation approach incorporates mechanisms for reducing this uncertainty, it is still present.

The second problem arises due to the large gaps between sensors 2, 3, 4 and 5. When two robots are connected, the close proximity of the modules and the large gaps between the sensors can create blind spots in some orientations (marked i , j and k in figure 7.3a). As a result of these blind spots, in certain configurations, the time taken to converge to a common heading is increased.

These two problems are further highlighted in figure 7.4. When sending messages via infrared, it is possible to estimate the distance between the sending and receiving sensors by measuring the intensity of the light received. Figure 7.4a maps the intensity of the IR signal for messages sent between two robots, arranged at various orientations. The setup used to gather this data is shown in figure 7.4b, where robot 1 is the receiving module

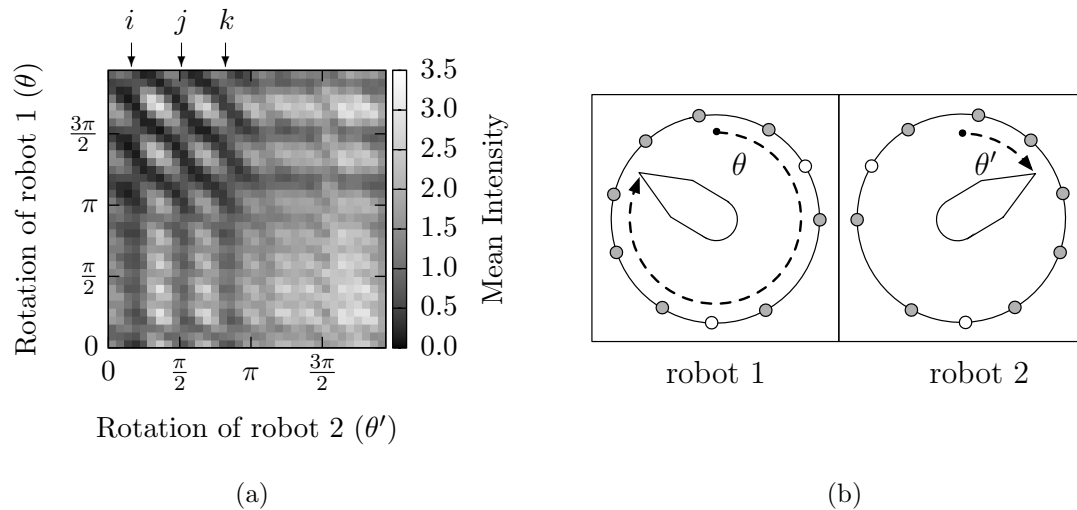


Figure 7.4: An intensity heatmap of messages sent between two modules at various orientations (a) and a diagram of the setup used to gather the data (b)

and robot 2 is the sender. The intensity of the signal associated with five received messages was recorded at 10° intervals for every 1296 (36×36) possible configurations of the two robots. Where no message was received within a certain time limit, an intensity of 0 was assigned. The mean value of the five measurements is plotted.

It can be noted from figure 7.4a that, due to the distribution of the sensors, when the two robots are facing each other (bottom right) the intensity of the received signals is high, but when two robots are facing away from each other (top left) the intensity is often low. A high intensity value indicates that the sending and receiving sensors are closely aligned, so when two robots are facing each other the measurement of angles a and b is likely to be more accurate than when they are facing away.

7.3.2 Sensor-aware alignment

The new approach to alignment aims to tackle the problems identified in the previous section, through greater awareness of the positioning and behaviour of the e-puck's sensors. Like the static synchronisation approach, robots still communicate with and track the relative orientation of their neighbours. However, as well as making use of the content and direction of the messages they receive, the signal intensity also influences their behaviour.

In figure 7.4a, the lines at $x = i$, $x = j$ and $x = k$ correspond respectively to the configurations at which the blind spots i , j and k of robot 2 are directly aligned with robot 1. As shown in figure 7.4a, the intensity values of the messages received along and adjacent to the lines i , j and k are low. It is possible to make use of this fact to infer

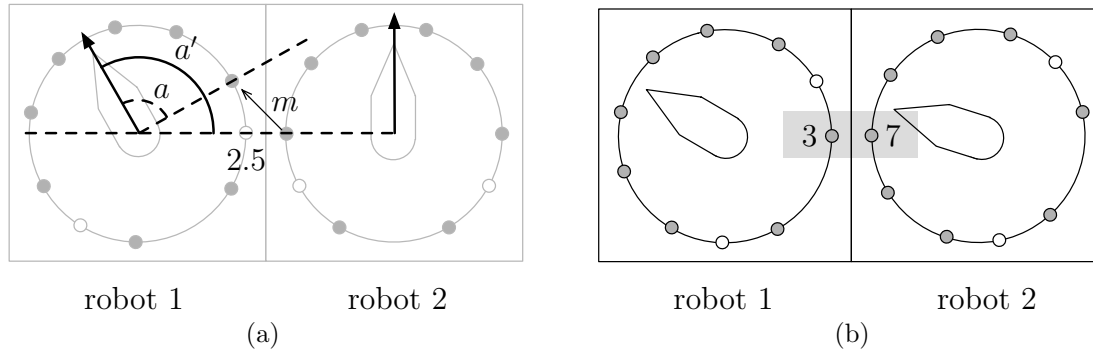


Figure 7.5: Diagrams showing the strategy for correcting the misalignment of two connected robots using both ‘virtual’ sensors (a) and paired sensors (b)

when the blind spot of a robot is aligned with its neighbour, and hence to determine the position of the neighbour more accurately. Specifically, as shown in figure 7.5a, when the message m received at sensor 2 reports a low intensity, it can be inferred that blind spot k of robot 1 is facing robot 2. Whilst it is true that sensor 2 will also report low intensity values when the point between sensors 1 and 2 is aligned with robot 2, because this gap is smaller, these values will never drop as far they do in blind spot k .

A similar inference can be applied to blind spot i and its relation to sensor 5. Notionally, it is possible to define two *virtual* sensors ‘2.5’ and ‘4.5’ which lie between sensors 2-3 and 4-5 respectively. As shown in figure 7.5a, if a message is detected at sensor 2.5, rather than be assumed to have originated from a point at an angle a , it can more accurately be assumed to have originated from an angle a' , halfway between sensors 2 and 3. Note that it is not possible to define a virtual sensor ‘3.5’ which lies between sensors 3 and 4, because from the perspective of these sensors, the blind spots i , j and k are indistinguishable. In figure 7.3a, the virtual sensors are represented by the white circles at points i and k .

It should be noted that, using intensity values alone, it is difficult for a robot to differentiate between scenarios in which its own blind spot is facing its neighbour, its neighbour’s blind spot is facing it, or both blind spots are facing each other. This is not a major concern, however, since in either scenario the reaction is the same, the robots will turn towards each other. Furthermore, it may be noted that messages received from neighbours that are not directly connected, i.e. neighbours positioned at a diagonal, will always have lower intensity values. Since the *LibIrcorn* library preferentially processes high intensity messages, the proportion of messages received from indirect neighbours, and thus the influence they exert, will be lower than that of direct neighbours. In the worst case scenario, robots will over eagerly turn towards each other, but as is suggested in section 7.6, this is not always detrimental.

To further improve the time taken for the robots to converge upon a common heading, a new method for translating the relative headings of neighbours into motor commands is also implemented. The method begins by calculating the average heading of

all of the robot's most recently detected neighbours. This value (η) is calculated using equation 7.1, where $(h_i, h_{i+1}, \dots, h_n)$ are the relative headings of the robot's neighbours, $k = n$ is the number of neighbours and γ_i is a weight, which in this case is set to 1.

$$\eta = \arctan 2 \left(\sum_{i=0}^{i=k} \sin(h_i) \cdot \gamma_i, \sum_{i=0}^{i=k} \cos(h_i) \cdot \gamma_i \right) \quad (7.1)$$

The average heading η , which will always be in the range $-\pi < \eta \leq \pi$, is used to determine the angular speed of the robot's motors. As shown in figure 7.2, before being applied to the motors, η is multiplied by a synchronisation parameter α , the larger this value, the more aggressively the robots will attempt to turn. For the motor on the right (M_R), the α parameter is inverted, this means that for values of $\eta < 0$ the robot will turn left and for values of $\eta > 0$ will turn right. For values of $\eta = 0$ and for control cycles in which no messages are received, the turning speed of the robot's motors is set to zero.

In communicating the relative angle at which a neighbour was detected, robots transmit the number of the sensor, rather than the angle itself. Furthermore, if $|\eta| > \frac{\pi}{2}$, to pre-empt the fact that the robot is about to make a fast turn, the sensor number that is transmitted is incremented or decremented by one—depending upon whether the robot is turning left or right.

In a further adaptation, based upon the knowledge that a high intensity signal is indicative of a close alignment between two sensors, sensor pairings are defined which, when the intensity of the signal is high, should not influence the movement of the robots. For example, in figure 7.5b, if robot 1 receives a high intensity message on sensor 3, that was sent from sensor 7 of robot 2, the relative heading of robot 2 will be set to 0. Note that although the alignment between robots 1 and 2 in this scenario is not perfect, it is considered 'good enough' for the task at hand, and preferential to the robots continuously changing direction.

7.3.3 Obstacle Avoidance

Every sensor that has not received a message within the last few seconds, and is not positioned next to a sensor that has received a message, contributes to obstacle avoidance. Each contributing sensor which detects an obstacle closer than the avoidance threshold τ creates a new desired heading in the opposing direction to the obstacle. The distance to the detected object is used to assign a weight w_i in the range $(0, 1)$ to each of these new headings. The closer the obstacle is, the larger the weight. These headings are added to the relative headings of the robot's neighbours using equation 7.1 with γ_i set to w_i and k set to N , where N is the total number of headings from both the obstacle avoidance and alignment behaviours. As described in section 7.3.2, the average heading η is used to determine the speed of the robot's motors.

This approach is equivalent to assuming that there is a neighbouring robot facing every sensor that has detected an obstacle. The robots attempt to align with these

imaginary neighbours in the same way that they align with real robots. For example, if an obstacle was detected on the left hand side of a robot, the robot would react as if there was another module directly facing this side, it would attempt to align with it, and in doing so would turn right, therefore avoiding the obstacle. The only difference between the robots reaction to obstacles and neighbouring robots, is that headings derived from the presence of obstacles are weighted according to how far away they are, with closer obstacles exerting a greater influence over alignment.

7.3.4 Forward Bias

In preliminary experiments, to ensure that the robots always moved forward, a small positive bias was added to the speed of the robots motors. For structures containing only a few robots, this bias, combined with the obstacle avoidance behaviour described above, was sufficient to ensure that the robots remained synchronised whilst still obtaining good coverage of the arena. However, when the number of robots was increased, a more adaptive approach was required.

When the number of modules was first increased, it was observed that the forward motion of the robots could have a negative effect on the robot's alignment. In an unsynchronised structure, it is not desirable for all of the individual robots to attempt to move forwards. The forward motion of the robots reduces their turning rates, which results in an increase in synchronisation time. Furthermore, in a structure containing many modules with different headings, the forward motion of the individuals increases the risk of the group breaking apart.

To handle this problem, the algorithm was modified so that the robot's forward bias is scaled in proportion to how well they believe themselves to be aligned—based upon the known relative headings of their neighbours. Using equation 7.2, a *synchronisation score* s is calculated by taking the average magnitude of the N headings (h_i, h_{i+1}, \dots, h_N) provided by the obstacle avoidance and alignment behaviours, this value will be low for well aligned robots and high otherwise. The score is then scaled using function $B(s)$ (7.3) and multiplied by the forward speed parameter β to determine how much bias to add to the speed of the robot's motors.

$$s = \frac{1}{N} \sum_{i=0}^{i=N} |h_i| \quad (7.2)$$

$$B(s) = \begin{cases} 0 & : s \geq \pi \\ \frac{\pi-s}{\pi} & : s < \pi \end{cases} \quad (7.3)$$

Robots with a low score, that is those which are well aligned, are consequently assigned a larger forward bias than those with a high score. In a poorly synchronised structure the scores will be high, meaning the forward motion will be less disruptive and the robot's rotary motion will dominate. Leading to faster alignment and reducing the chances of the structure breaking apart.

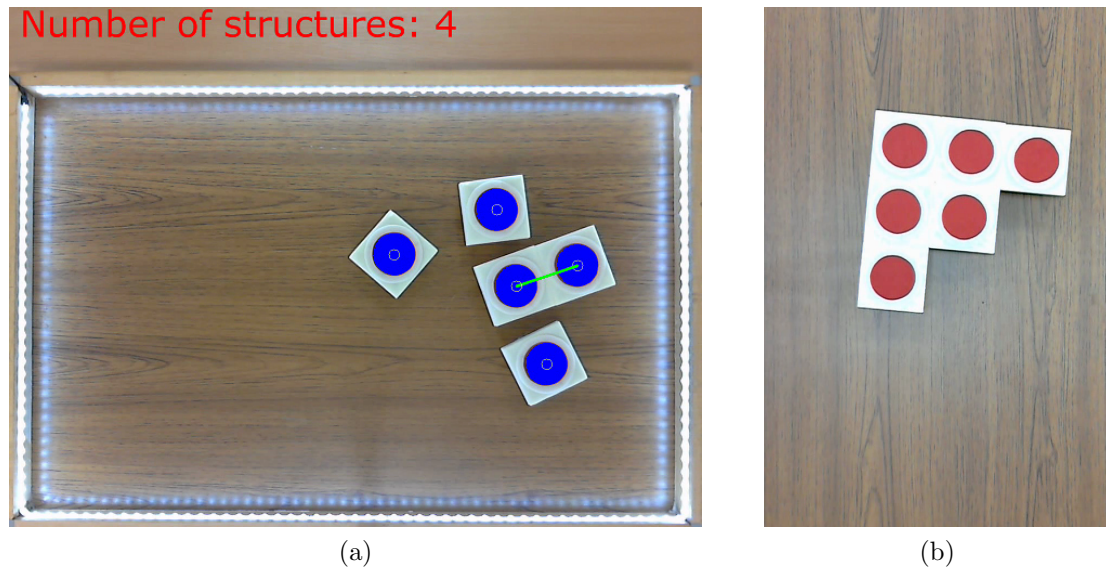


Figure 7.6: Top down views of the arena used for the experiments, both with (a) and without (b) information from the computer tracking software superimposed

7.4 Experimental Setup

The experiments detailed in the following sections were all conducted using the same setup. Every experiment was performed within a small table-top arena, measuring approximately $86 \text{ cm} \times 56 \text{ cm}$. The arena, shown in figure 7.6a, is surrounded by a ring of white LEDs, however, in this work, the LEDs were only used during the final experiment of section 7.7. The base of the arena is wooden, which provides good traction for the e-puck’s rubber coated wheels, but is not so resistive that it will completely prevent robots from ‘skidding’ if lateral forces are applied. An overhead camera was used to record the experiments and tracking software was used to extract positional and connectivity information from the scene. Some of the output from the tracking software is shown in figure 7.6a, the blue circles highlight the robots currently being tracked and the green links signify which robots are connected. The number of independent structures detected is shown in the top left corner of the view.

The experiments reported here may be split into two classes, those in which the robots start in a fully connected state and those in which they start disconnected. Section 7.5, and the first part of section 7.7, cover those of the former class, whilst section 7.6, and the remainder of section 7.7, handle the latter. The 15 starting configurations that were used for the first class of experiments are pictured and labelled in figure 7.7. In the experiments where the robots began disconnected, the modules were evenly distributed within the arena and their headings were randomised at the start of each run. In the experiments where the robots began connected, the structure was always placed in the centre of the arena at the same orientation, but the underlying robots were aligned and rotated at 10 different starting angles, each offset by 36° . In total,

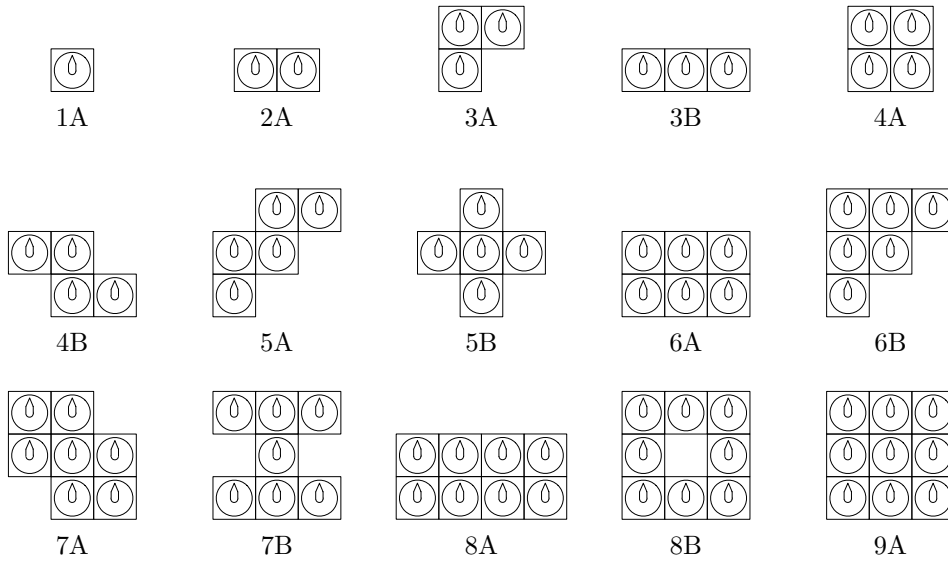


Figure 7.7: The various different configurations used in the experiments

Parameter	Description	Value	
		A	B
α	Synchronisation weight	100	125
β	Forward speed weight	20	50
τ	Avoidance threshold	30	30

Table 7.2: The two parameter sets used in the experiments

280 independent runs were performed and ~ 45 hours worth of data was collected.

The parameters used in the experiments are displayed in table 7.2. Parameter set ‘A’ was used only for the stationary alignment experiments reported in section 7.5.1 and the preliminary collective locomotion experiments described at the start of section 7.6. Following these experiments, the parameters were tweaked to produce set ‘B’, which was then used for the remainder of the experiments. The values were chosen by trial and error. Once satisfactory behaviour was observed, no further attempt was made to optimise the parameters.

7.5 Collective Locomotion

In this section, the task of collective locomotion is investigated. In order to assess the effectiveness of the alignment behaviour introduced in section 7.3.2, this section begins by examining the sub-task of stationary alignment. The section then moves on to investigate the full collective locomotion task.

7.5.1 Stationary Alignment Results

In this set of experiments, the modified alignment behaviour introduced in section 7.3.2 is compared with that of the original static synchronisation strategy from the *LibIrcorn* library. Experiments were conducted using groups of two, three and four stationary robots, arranged as shown in figure 7.8a.

For each controller and each arrangement, 20 individual runs were conducted. The orientation of the robots was randomised at the start of each run and the absolute heading of each robot was recorded at one second intervals over a period of 100 seconds.

To assess the robots' ability to converge towards a common heading, the polarisation metric from [65] is used. As shown by equation 7.4, the polarisation P of a group of robots G is defined as the sum of the distance between the heading of every robot and its angular nearest neighbour θ_{ann} .

$$P(G) = \sum_{i \in G} \theta_{ann}(i). \quad (7.4)$$

Figures 7.8b-d plot the mean polarisation of the two approaches, for each of the three module configurations. As is evident by the eventual low polarisation values in all of the figures, in every experimental run, the modules were observed to converge to and maintain a common heading. In comparing the two approaches, using the Wilcoxon rank-sum test with a significance level of 0.05, it can be said that there is no significant difference in polarisation over the final 20 seconds of each experiment. However, in every configuration, it can be observed that convergence is faster for the experiments utilising the new approach to alignment. Furthermore, during the convergence phase, the variance in the polarisation of the static synchronisation approach is greater. In the accompanying material, video 7.1 provides a comparison of the two different approaches to stationary alignment.

7.5.2 Collective Locomotion Results

After integrating the obstacle avoidance and forward bias behaviours, the task of controlling the collective locomotion of a group of mobile robots is now considered. For each of the 15 starting configurations shown in figure 7.7, 10 independent runs were performed, each lasting 10 minutes. If one or more modules became disconnected from a structure, the run was terminated. As reported in the following section, 29 out of 150 runs ended in this manner. In every run, the positions of the individual modules, the number of structures, and the number of modules in each structure was recorded. This information was used to calculate the *success rate*, *coverage*, *cohesion* and *speed* for each of the different configurations. The results of these experiments are summarised in table 7.3, and analysed in more detail in the remainder of this section. Following on from these experiments, the *scalability* of the approach is examined by considering linear structures containing between one and six modules. Video 7.2 from the accompanying material shows a variety of different structures performing collective locomotion.

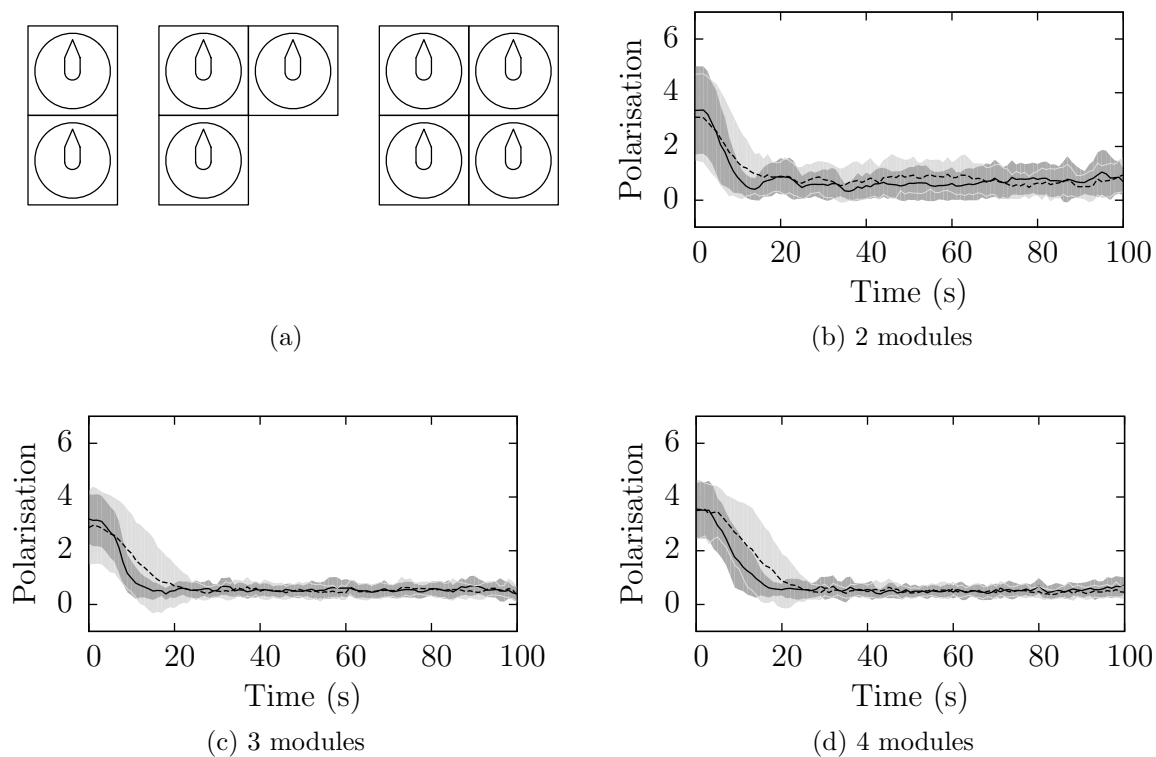


Figure 7.8: Graphs (b-d) plot the mean polarisation \pm one standard deviation, for each of the three configurations in (a). The static synchronisation approach is represented by the dashed line and the lighter grey region, and the new approach is represented by the solid line and the darker region

Shape	Moore	von Neumann	Speed (cm/s)	Coverage	Cohesion	Runs
1A	0.00	0.00	0.57 [3.0e-5]	0.56	1.00	10/10
2A	1.00	1.00	0.47 [4.9e-4]	0.60	1.00	10/10
3A	2.00	1.33	0.39 [1.1e-4]	0.64	0.91	7/10
3B	1.33	1.33	0.35 [2.9e-4]	0.63	1.00	10/10
4A	3.00	2.00	0.34 [1.4e-4]	0.67	1.00	10/10
4B	2.50	1.50	0.33 [3.2e-4]	0.68	1.00	10/10
5A	2.80	1.60	0.26 [3.1e-3]	0.62	0.51	1/10
5B	3.20	1.60	0.25 [3.5e-4]	0.59	0.66	5/10
6A	3.67	2.33	0.23 [3.5e-4]	0.65	1.00	10/10
6B	3.33	2.00	0.23 [7.2e-4]	0.64	0.62	2/10
7A	4.00	2.29	0.20 [1.2e-4]	0.70	1.00	10/10
7B	2.86	1.71	0.22 [7.3e-4]	0.69	0.85	6/10
8A	4.00	2.50	0.20 [3.0e-4]	0.68	1.00	10/10
8B	3.00	2.00	0.19 [2.3e-4]	0.67	1.00	10/10
9A	4.44	2.67	0.16 [4.3e-4]	0.70	1.00	10/10
Total						121/150

Table 7.3: A summary of the results from the collective locomotion experiments. From left to right, the columns display the name of shape; the average number of Moore and von Neuman neighbours; the speed, coverage and cohesion scores (described in the text); and the total number of runs successfully completed

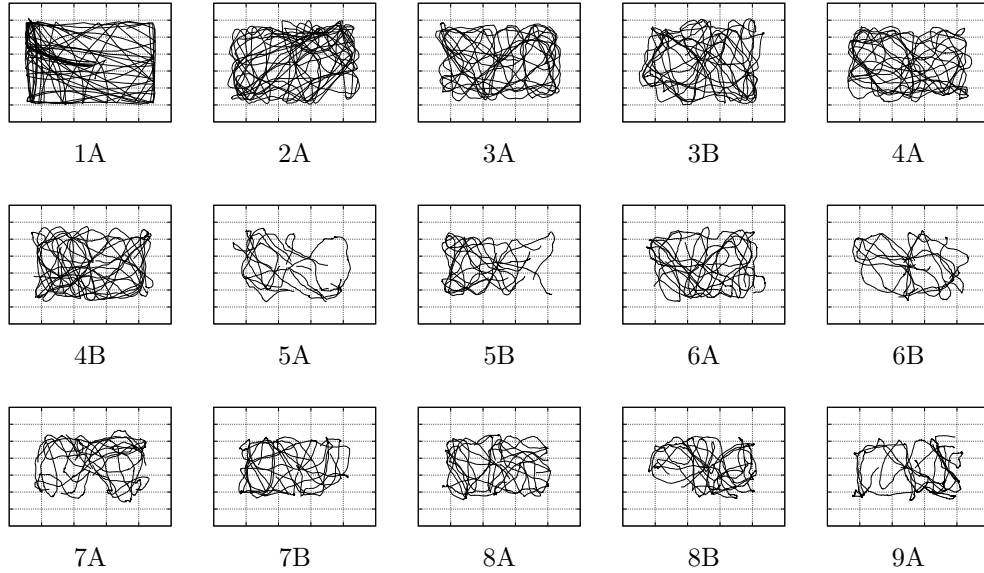


Figure 7.9: Plots tracing the centre of mass of each structure in the collective locomotion experiments. Each plot combines data from 10 runs

Success rate

In this section, ‘success’ is defined as the ability of a structure to navigate the arena for the full 10 minute duration of a run, whilst all of the modules remain connected to one another in the same configuration in which they started. To account for any inaccuracies in the tracking software, and to allow for minor breaks that are quickly repaired, a break in the link between two modules is only acknowledged if it exists for greater than one second. The results show that the approach is successful in the vast majority of scenarios, even with structures containing up to nine modules. Out of 150 runs, across all 15 configurations, it was found that the robots were successful on 121 occasions and that 10 of the 15 configurations were successful 100% of the time.

Coverage

Figure 7.9 shows the area covered by the centre of mass of each structure across all 10 runs. It is observed that, in the single module configuration (1A), the robot is able to obtain an even coverage of the entire arena³. As the number of robots is increased, it naturally becomes harder for the centre of mass of the structures to reach the edges of the arena. Despite not being able to reach the arena perimeter, and although coverage is increasingly sparse, a relatively even coverage is still observed with configurations of up to nine modules.

³For 1A, there is a bias towards the left-hand side of the arena. This is due to the fact that, in this experiment, the heading of the robot was kept the same across all runs.

To allow coverage to be analysed more quantitatively, the arena is split into 672 equal squares, each measuring approximately 7.18 cm^2 . The coverage value in table 7.3 is calculated as the percentage of these squares that the centre of mass of any module in the structure has passed through⁴. It was expected that coverage would increase inline with an increase in the number of modules, by virtue of the increase in surface area. However, whilst there is a general trend of increasing coverage between 1A and 9A, it is not as striking as one might expect. This is highlighted by the fact that the two 4 module structures obtain the same coverage score as the two 8 module structures, despite having half as many modules. In the following sub-section, it is suggested that the reason coverage does not scale directly with size, is related to the speed at which structures of various sizes travel.

Speed

Speed is calculated by measuring the average distance travelled by each module in 0.5 seconds, summed over the course of a run and divided by the duration of the run. The value in table 7.3 is averaged over all runs and the variance is shown alongside. In general, it is observed that as the number of modules in a structure is increased, the average speed at which the structure travels decreases. This may help explain why the coverage of larger structures is lower than expected.

Figure 7.10a plots the mean average speed for structures of different sizes. The results from configurations which contain the same number of modules are combined into the same sample. Whiskers are used to show the extent of the mean values plus or minus one standard deviation. Using the Wilcoxon rank-sum test with a significance level of 0.05, it can be said that there is a statistically significant difference between all pairs of samples in figure 7.10a.

It is suggested that the reason why larger structures travel slower than smaller ones is because, in a larger structure, it takes the robots longer to synchronise their headings. The problem is compounded by the fact that a larger structure will need to turn to avoid obstacles more often than a smaller one, increasing the time in which the robots will be out of synchronisation with one another. Furthermore, because the robots can only ever determine the approximate heading of their neighbours, there will always be noise in their measurements. Since the approach relies heavily on infrared communication, it is also highly susceptible to errors caused by reflections or interference. With a greater number of robots, the probability of such errors occurring will be larger, increasing the level of noise and reducing the accuracy in the estimation of a neighbours heading. With less accurate measurements, the time taken to reach a consensus can be expected to increase further.

⁴It would be easy to achieve ‘better’ coverage by using fewer squares, however, the purpose of this experiment was to compare the coverage of different configurations, not to measure the overall performance. Therefore a value was deliberately chosen that was large enough to ensure that no configuration was able to achieve a score of 1.0.

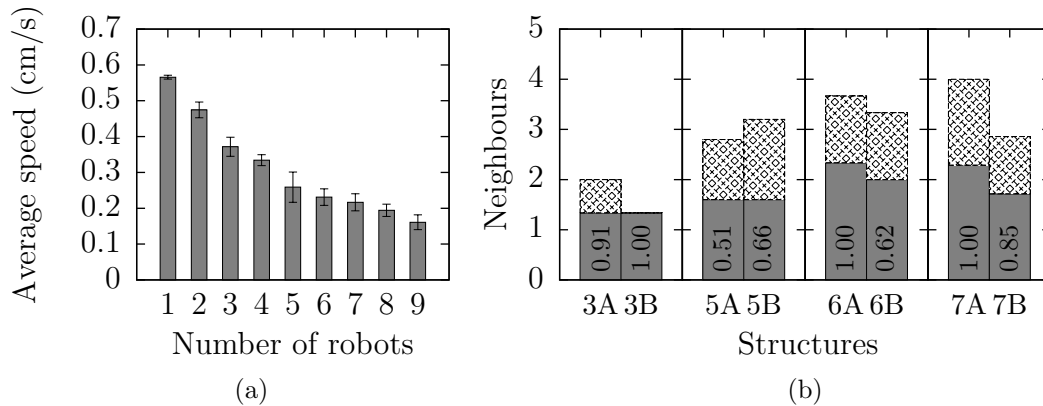


Figure 7.10: Graphs showing the mean average speed (\pm one standard deviation) for structures of different sizes (a) and the average number of von Neumann (dark grey box) and Moore neighbours (dark grey box plus hatched box) in pairs of structures containing 3, 5, 6 and 7 robots (b)

Cohesion

In order to analyse the ability of robots to remain connected with one another in more detail, a ‘cohesion’ value is defined. The cohesion value is calculated by dividing the total time in which the modules remain connected by the maximum available time.

Out of the 15 different structures, 10 were able to remain connected for the full duration of every run, and are therefore assigned a cohesion value of 1.0. From the remaining configurations, structures 5A and 6B were observed to be the least stable, obtaining cohesion scores of 0.51 and 0.62 whilst remaining connected for the full duration in only 1 and 2 of the 10 runs respectively. Structures 5B and 7B were the next least stable, with scores of 0.66 and 0.85. Of all the structures with a success rate of less than 100%, structure 3A was the most stable, remaining connected 91% of the time.

Structural similarities can be observed between all five configurations with a cohesion score of less than 1.0. Note that the 3A pattern is repeated in all of these shapes. However, the 3A pattern is also present in structures 4B, 7A and 8B, which each obtained cohesion scores of 1.0. This implies that the 3A pattern alone does not represent a universal motif of instability, and other factors such as the density and symmetry of a shape may also have a role.

It was not possible to find a single metric which accurately predicts the stability of a configuration purely from its geometric shape. Having attempted to use measures including: the length of the perimeter, the number of loosely connected modules (those with only one neighbour) and the elongation of the shape, it was eventually found that, in comparing two structures of the same size, the average number of neighbours provided the best estimate of stability.

In figure 7.10b, the average number of neighbours in a von Neumann neighbourhood (dark grey box) and the average number in a Moore neighbourhood (dark grey box plus

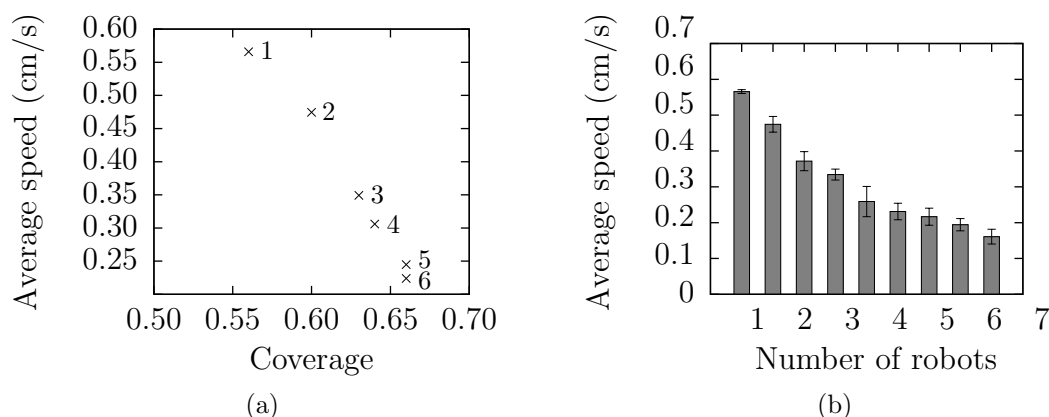


Figure 7.11: Graph (a) shows the relationship between the average speed and coverage scores for linear chains of between one and six robots, whilst (b) shows the mean average speed (\pm one standard deviation)

hatched box) is plotted for every configuration containing 3, 5, 6 or 7 modules. Within each box the cohesion score for the corresponding shape is printed. In comparing structures of the same size, it was found that in every case, the structure with the larger cohesion value has at least as many von Neumann neighbours as the structure with the lower cohesion value. When considering a Moore neighbourhood, it was found that in every case, apart from structures containing 3 modules, the configuration with the largest average number of neighbours has the largest cohesion score.

7.5.3 Scalability

To examine the scalability of the approach, three further experiments were performed involving linear chains of four, five and six modules. By combining these results with those of structures 1A, 2A and 3B, it is possible to observe how the system scales for linear configurations of between one and six modules.

As shown in table 7.3, for chains containing one, two and three modules, the robots remained connected for the full duration of every run. This was also true in the four module case. For the five and six module cases, in one and two runs respectively, the structures were not able to remain connected, resulting in cohesion scores of 0.95 and 0.92. It is observed that, as the number of modules is increased, the chances of them remaining connected decreases.

Figure 7.11a plots the average speed against the coverage scores of each of the linear structures. The graph shows that as the number of modules is increased, whilst the average speed decreases, the coverage continues to increase.

Figure 7.11b plots the average speed of linear chains containing up to six modules. As the number of modules is increased from one to three, a sharp drop in speed is observed, however, in chains containing between four and six modules, the decrease in

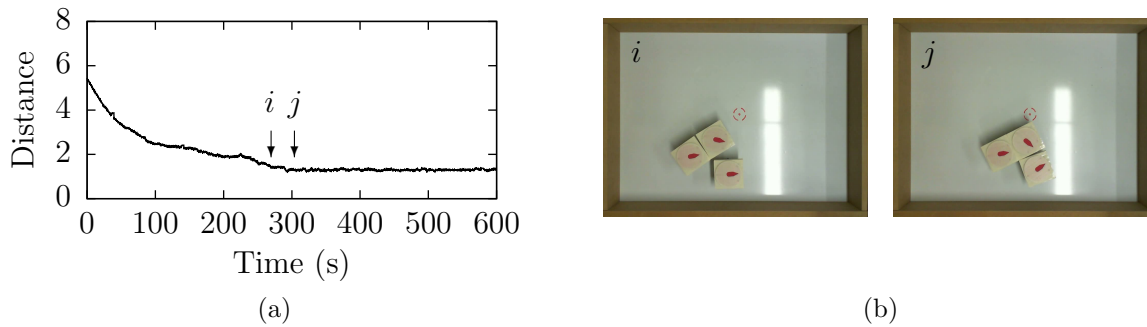


Figure 7.12: The pairwise distance between three robots recorded over a 10 minute period (a) and still images corresponding to the points in time i and j (b)

speed begins to level off. Based upon a Wilcoxon rank-sum test with a significance level of 0.05, it may be stated that the drop in performance between the four and six module chains is not statistically significant. That is to say, the performance of the system is not significantly affected by the addition of extra modules, indicating that the system scales well with respect to speed. It is noted, however, that due to the small size of the arena, whilst avoiding obstacles, larger structures spend a lot of their time stationary. Different insights into the scalability of the system may be gained by experimenting within a larger arena, where less time is spent avoiding obstacles.

7.6 Self-assembly

In preliminary collective locomotion experiments, using an earlier version of the algorithm introduced in section 7.3, an interesting self-assembling behaviour was observed which allowed a group of modules to reform if broken apart. In the earlier version of the algorithm, the link between boxes (7.2) and (7.3) in figure 7.2 was not present and the forward bias always took the same value as the forward speed parameter, equivalent to assigning the robots a synchronisation score of 0. To examine this self-assembling property further, three robots, each running the early version of the collective locomotion algorithm with parameter set ‘A’, were placed in different corners of a small arena and left to operate for 10 minutes.

In figure 7.12a, the average pairwise distance between each of the robots is plotted over a 10 minute period. As can be seen in figure 7.12a, the robots start far away from one another and gradually converge to a close proximity at around the 5 minute mark. For the remainder of the experiment they remain within close proximity of each other. As shown in figure 7.12b, at point i , two of the modules physically join together to form a two module structure. Shortly after, at point j , the third module joins to complete the $3A$ configuration from figure 7.7. The robots then remained in this configuration until the end of the run.

This self-assembling behaviour was not pre-programmed, it emerged purely due to the interaction of the robots and their environment. Specifically, it is proposed that it results from a combination of four factors. Firstly, the enclosed arena ensures that robots never stray too far away from one another. Secondly, the alignment behaviour ensures that the robots all head in a similar direction. Thirdly, the design of the e-puck extension ensures that if two robots come into close proximity, their magnetic docking interfaces will cause them to ‘snap’ together. Finally, although there is no explicit cohesion behaviour, the implementation of virtual sensors described in section 7.3.2 may cause robots to move towards one another when they mistakenly believe themselves to be aligned with the blind spot of another robot.

When robots are near to one another, it is stated that the virtual sensors have the effect of producing a more accurate and stable form of alignment, but that when the robots are far apart, the virtual sensors cause them to turn towards one another, resulting in a cohesive behaviour. In this section, a new experiment is described in order to test whether virtual sensors really do benefit self-assembly.

7.6.1 Experimental Setup

Beginning with a group of separate individuals, the time taken for the robots to assemble into a single structure is measured. The performance of three different systems is tested, one in which virtual sensors are used (*A*), one in which virtual sensors are not used but the rest of the alignment behaviour is present (*B*), and a control experiment in which no form of alignment is implemented and the robots simply perform wandering (*C*).

For each system, 20 experimental runs were performed, half of which involved five robots, and half of which involved ten. Data was recorded for ten minutes of every run, regardless of whether the robots successfully assembled into a single structure. The number of times in which the system was able to successfully assemble, and the time taken to do so was recorded. Unsuccessful runs were assigned the maximum completion time of 600 s. To test whether there is a significant difference between the time taken to self-assemble in systems which use the virtual sensor approach (*A*) and systems which do not (*B* and *C*), the following four null hypotheses are presented:

$H7.1_0$: There is no difference in the time taken by **five** robots using system **A** and five robots using **B** to self-assemble into a single structure

$H7.2_0$: There is no difference in the time taken by **five** robots using system **A** and five robots using **C** to self-assemble into a single structure

$H7.3_0$: There is no difference in the time taken by **ten** robots using system **A** and ten robots using **B** to self-assemble into a single structure

$H7.4_0$: There is no difference in the time taken by **ten** robots using system **A** and ten robots using **C** to self-assemble into a single structure

#	System	Time (s)			# Structures (final 300 s)			Runs
		mean	median	std.	mean	median	std.	
5	A	245.40	255.00	64.93	1.01	1.00	0.03	10/10
5	B	398.60	399.00	133.30	1.44	1.33	0.42	8/10
5	C	573.15	600.00	84.91	2.13	2.03	0.47	1/10
10	A	381.15	397.00	178.47	1.50	1.37	0.42	7/10
10	B	411.30	544.00	225.49	2.19	2.05	0.54	5/10
10	C	600.00	600.00	0.00	3.57	3.68	0.58	0/10

Table 7.4: A summary of results from the self-assembly experiments. From left to right the columns show the number of robots, the system used, the average assembly time, the average number of structures assembled during the final 300 s of each run, and the number of runs in which assembly was successful

7.6.2 Analysis

The number of times in which the assembly process was successfully, and the average time taken to do so, is reported in table 7.4. It is observed that, with both five and ten robots, the system in which virtual sensors are used always outperforms the other two, both in terms of the mean assembly time and the number of successful runs. In the five robot scenario, system *A* was able to assemble into a single structure during every run and did so on average 2.5 minutes quicker than system *B*. System *B* was successful in completing assembly on seven occasions, whilst system *C*, purely by chance, was able to fully assemble on one occasion. In the accompanying material, video 7.3 shows a group of five robots using system *A* to perform self-assembly. With ten robots, system *A* was able to assemble into a single structure on seven occasions, and did so on average 30 seconds quicker than system *B*, which completed assembly on five occasions. In the ten robot case, system *C* was never able to successfully complete assembly.

Figure 7.13 plots the assembly time for systems *A* and *B* (top), and the mean number of structures over time for all three systems, both for the five robot (a) and ten robot (b) scenarios. In general, it is observed that the mean number of structures present during the experiments of system *A* is lower than for systems *B* and *C*, however, the difference is much less in the ten robot setting than in the five robot setting, specifically during the first 200 seconds.

Behaviourally, systems *A* and *B* are first observed to form small structures. These structures combine with other robots to form larger groups, eventually resulting in the formation of a single coherent structure. With system *C*, chance collisions are observed that lead to the formation of static structures. Without an alignment behaviour the robots are not able to perform collective locomotion, but do not move with sufficient force to break apart. Therefore, after forming structures, the robots in system *C* simply remain stationary for the remainder of the run.

For the five robot scenario, using the Wilcoxon rank-sum test with a significance

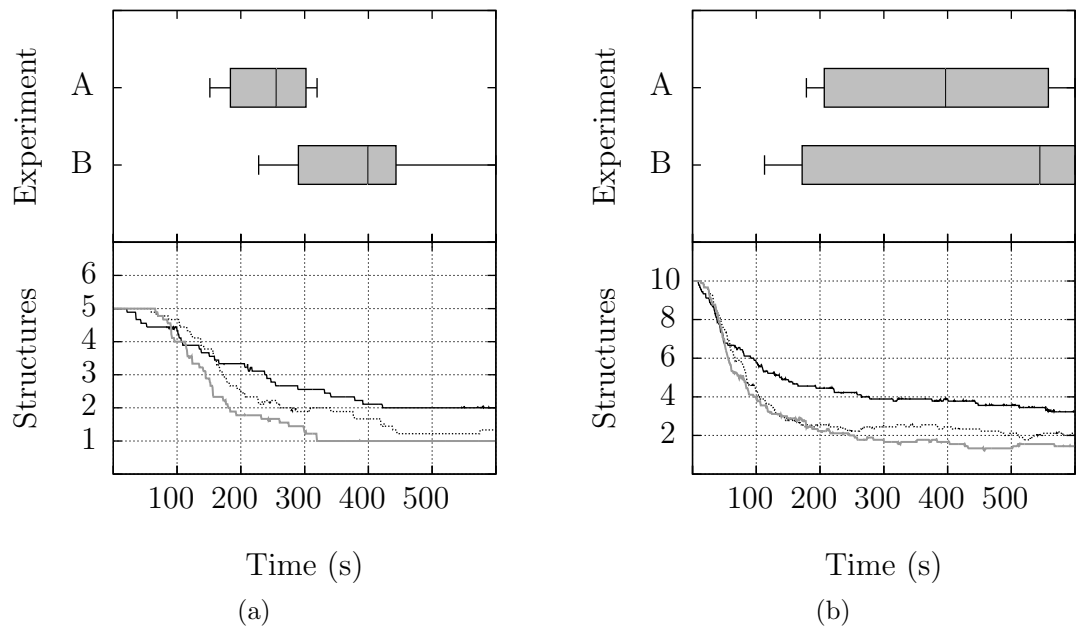


Figure 7.13: Boxplots showing the assembly time for systems *A* and *B* (top), and graphs showing the mean number of structures over time (bottom), both for experiments containing five (a) and 10 robots (b). In the lower charts, the grey line corresponds to system *A*, the dotted line to *B* and the black line to *C*

level of 0.05 it is possible to reject both $H7.1_0$ and $H7.2_0$, and accept the alternatives. That is to say, with five robots, there is a significant difference between the time taken by system A and systems B and C to self-assemble.

For the ten robot scenario, in comparing systems A and B , it is clear from the large overlapping of the boxplots in figure 7.13b that it is not possible to reject $H7.3_0$. This is confirmed using the Wilcoxon rank-sum test with a significance level of 0.05. Although the average (both mean and median) assembly time of system A is shown to be less than that of system B in table 7.4, the difference between the medians is not statistically significant. It is possible, however, to reject $H7.4_0$, and state that there is a significant difference in the time taken by systems A and C to self-assemble into a single structure.

In comparing systems A and B , it is suggested that the reason why a significant improvement in performance is only observed in the five robot scenario, may be due to the population density of robots within the arena. In the case of ten robots, with the modules packed twice as densely, the probability of two robots colliding purely by chance is greatly increased, potentially masking any advantage that the virtual sensors provide. This is supported by the graph in figure 7.13b in which, over the first 50 seconds, the mean number of structures follows the same trend for all three experiments. As the robots begin to cluster, however, the probability of chance collisions is reduced and as observed during the period from 300 seconds onwards, system A begins to outperform system B , which in turn outperforms system C .

Another reason that a significant difference between the assembly time of systems A and B is not observed when using ten robots, may be due to the fact that uncompleted runs are assigned a completion time of 600 s. Assigning a value of 600 is a very conservative estimate. Allowing the runs to continue for longer would reveal a more accurate representation of the average time taken to assemble a structure. Based upon the fact that after 600 s, system A was successful on more occasions than B , and that the mean number of structures in figure 7.13b is consistently lower for A , if all runs were allowed to continue until completion A would be expected to perform significantly better than B .

To avoid the problems discussed above, a new metric is devised to analyse the performance of groups of ten robots. Firstly, to prevent the problem that a high density of robots may simplify the task to the point of masking the benefits of the virtual sensors, the system is analysed only during the final 300 s of each run. Secondly, to allow the performance of uncompleted runs to be assessed more fairly, instead of measuring the time taken until completion, the average number of structures assembled at any point during the final 300 s of a run is measured. For this new metric, a fifth hypothesis is proposed:

$H7.5_0$: There is no difference in the mean number of structures assembled by ten robots during the final 300 s of runs involving systems A and B

As shown in table 7.4, in comparing groups of ten robots using systems A and B , it is observed that the groups using system A have both lower mean and median values

for the number of structures assembled during the final 300 s of each run. Using the Wilcoxon rank-sum test with a significance level of 0.05 it is possible to reject $H_{7.5_0}$ and confirm that there is a significant difference in the average number of structures assembled by groups of robots that use virtual sensors (A) and groups that do not (B).

7.7 Self-reconfiguration

In order to test the behaviour of the robots in a more complex setting, a dynamic component is introduced into the environment. As shown in figure 7.6a, this is realised as a ring of LEDs surrounding the arena. The LEDs may be viewed as a proxy for more complex real-world perturbations, such as changes in terrain or weather conditions. In order to cope with such changes, robots must be able to adapt their behaviour. In this simplified setting, the switch between self-assembling and self-disassembling behaviours is examined in response to a change in the state of the surrounding LEDs. This section begins by describing a self-disassembly behaviour, before going on to describe experiments, inspired by the work of [190], which combine self-disassembly with the previously introduced self-assembly behaviour, to produce a simple form of environment driven self-reconfiguration.

7.7.1 Self-disassembly

As with the self-assembly behaviour, in the self-disassembly strategy, the robots continuously broadcast their IDs and listen for the IDs of their neighbours. Robots can detect whether they are connected to another module based upon the intensity of the messages that they receive. If a robot determines that it is connected to at least one other module, it will, with equal probabilities, either set both motors to the maximum forward value, both to the maximum reverse value, or both to alternate maximum values. The resulting ‘shaking’ motion is normally sufficient to break the connection between two modules. If a robot determines that it is not connected to any other modules, it will simply wander in the arena, avoiding obstacles and other robots.

To test this behaviour a series of experiments were conducted, using a variety of different structures. Specifically, all of the configurations from figure 7.7 that contain either four, five or six modules were investigated. In each experiment, the number of structures, the average number of robots in each structure and the time taken by the robots to completely disassemble was measured.

Figure 7.14 plots the mean number of structures and the mean size of each structure over the first 150 seconds of each five minute run. In every case, the number of structures is observed to increase from one to N , where N is the number of robots in the initial structure. Naturally, as the number of structures increases, the average size of the structures can be observed to decrease.

As expected, with more robots, the time taken to fully disassemble is observed to be greater. This is because more connections must be broken between neighbouring robots

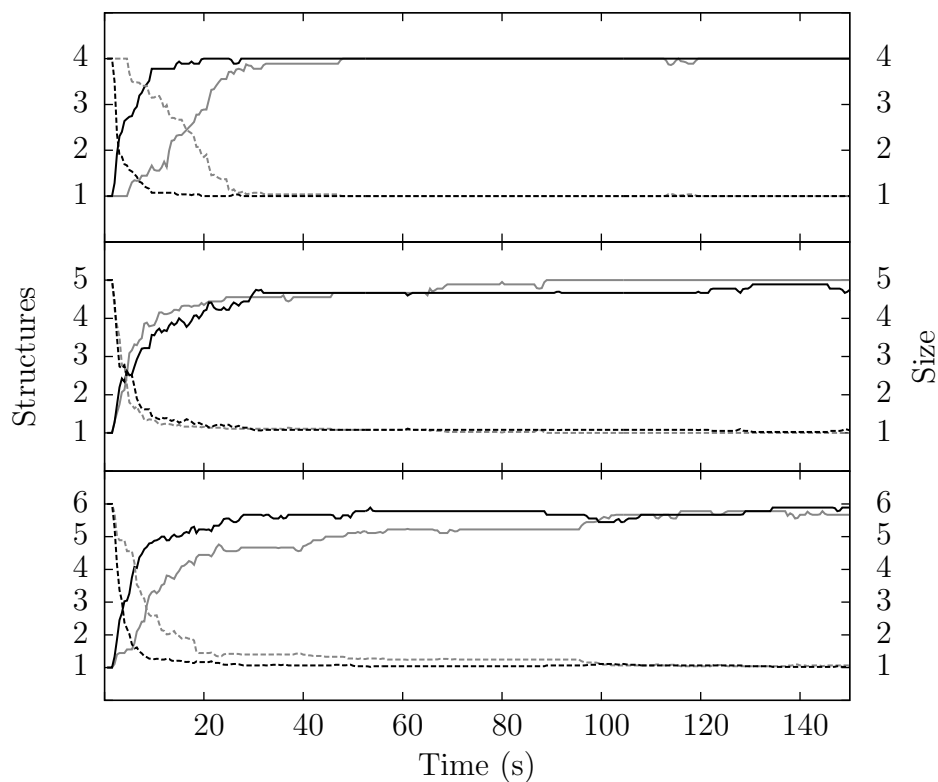


Figure 7.14: The average number of structures (solid line) and the average structure size (dashed line) over time. Results are presented for six different structures, two of each containing four, five and six robots. In each plot, the dark lines correspond to the ‘A’ shapes in figure 7.7, whilst the lighter lines correspond to ‘B’ shapes

and because, in a more densely packed environment, there is more chance of robots accidentally rejoining with other modules. In comparing structures of the same size, it is also observed that the structure with the greater number of von Neumann neighbours (corresponding to the number of physical connections) takes longer to disassemble. This is most evident in the case of structures 4A and 4B (figure 7.14 top).

Unfortunately, due to fabrication defects which led to variations in the positioning of the magnets, certain sides of certain robots appear to be more ‘sticky’ than others. On two occasions, in the experiments involving six robots, this led to runs in which the robots were not able to completely disassemble, as one pair of robots could not be separated. This introduced large outliers into the data. These outliers, many times larger than the mean, make statistical analysis difficult with such small samples, however it can still be observed from figure 7.14 that the general desired behaviour is present.

7.7.2 Self-reconfiguration

To investigate environment driven self-reconfiguration, 10 independent runs were performed, each lasting for 11 minutes and involving six modules. At the start of each run, the robots were spread evenly throughout the arena and began by executing the self-disassembly behaviour. After 50 seconds, the robots were triggered to switch from disassembly to assembly and remained in this state for a further five minutes. They then switched back to disassembly until the end of the run. The number of structures present was recorded over the duration of each run, the mean values are plotted in figure 7.15 \pm one standard deviation. For simplicity, repeatability, and to allow the experiments to be fully automated, the robots’ internal clocks were used to trigger the switch between assembling and disassembling. However, as shown in figure 7.16, this behaviour has also been demonstrated using the LED ring as a trigger.

The light grey area in figure 7.15 marks the five minute period during which the self-assembly behaviour was active. During this period, the average number of structures steadily decreases, flattening out at just below two. In 7 out of 10 runs, this five minute period was sufficient for all of the robots to assemble into a single structure, in the other three cases, the robots assembled into two separate structures. After the robots switched back to the disassembly state, in all but one of the runs, they were able to completely disassemble into six independent structures. After disassembling, the robots did not always remain disassociated until the end of the run, with chance collisions occasionally and temporarily bringing modules back together. This explains the variation in the mean number of structures over the last 200 s.

In contrast to the results presented in figure 7.15, where an internal timer was used to switch between assembling and disassembling behaviours, figure 7.16 provides screenshots from a video of an experiment in which the arena’s LED ring was used to trigger reconfiguration. The robots sense the LED ring by tracking changes in ambient lighting conditions using their IR sensors. Shortly after startup, a threshold value is established for each sensor by monitoring ambient light with the LED ring turned off. At each timestep, by comparing the ambient light with their threshold value, each

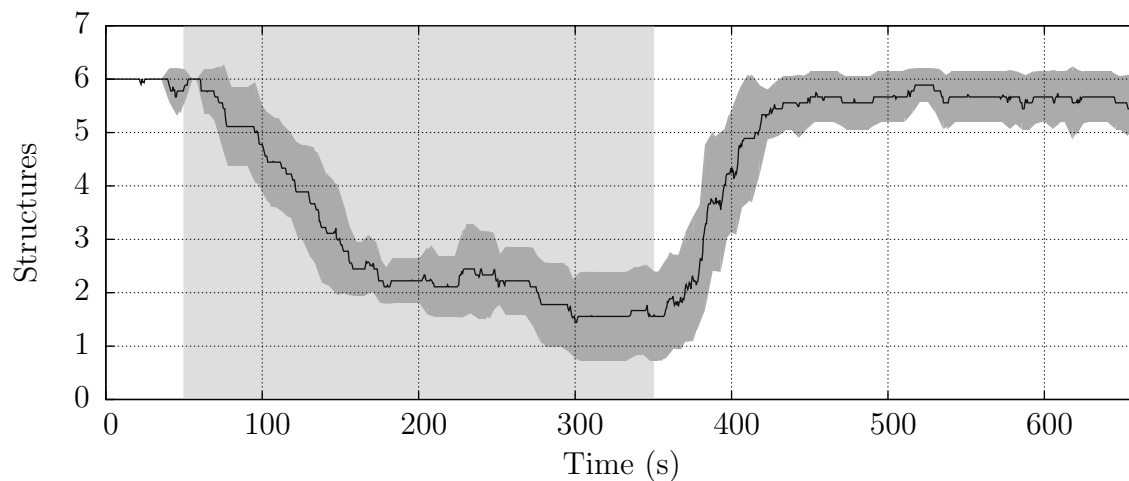


Figure 7.15: A graph showing the average number of individual structures over time. The dark shaded region shows the values \pm one standard deviation. The lighter region highlights the period of time during which the self-assembly behaviour was active

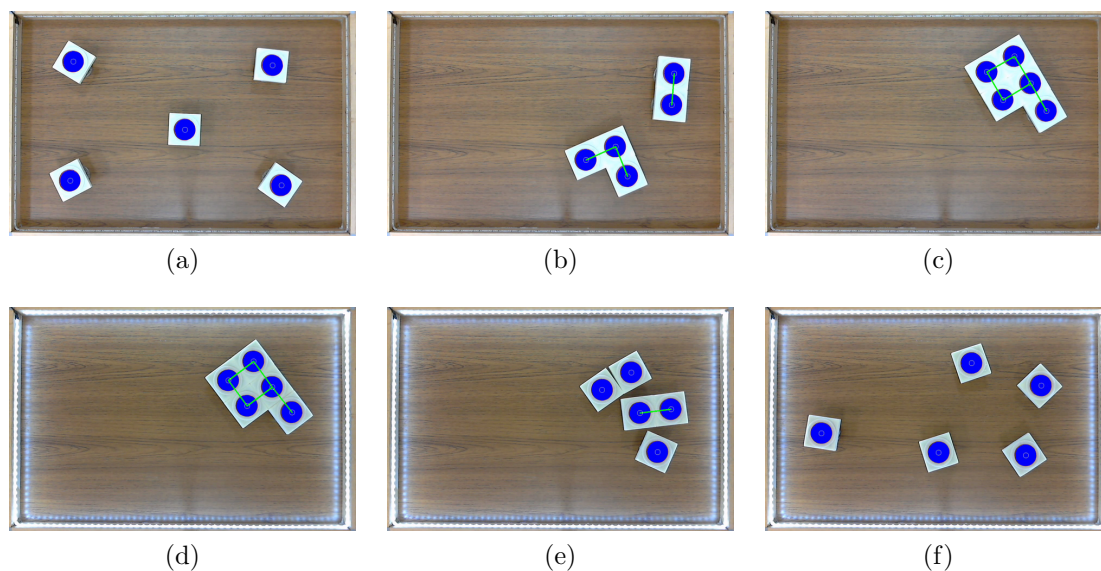


Figure 7.16: Still images from a video of environment driven self-reconfiguration

sensor provides one vote to determine whether or not the robot should switch between assembling and disassembling. The majority vote determines the robot's behaviour.

As shown in figure 7.16, the robots begin in a disconnected state, executing the self-assembly behaviour (a). At first they form two independent structures (b) which later collide to form a single structure (c). At which point the LED ring is turned on (d) triggering the switch from assembly to disassembly and causing the robots to split apart (e). Eventually, the robots reach a completely disassembled state (f), at which point the whole process may be restarted by turning off the LEDs. The video from figure 7.16 is available in the accompanying material (video 7.4) and can also be viewed, along with others, on the project website⁵.

7.8 Summary and Future Work

In this chapter, an extension was presented for the e-puck robot that transforms what is traditionally a swarm robotic platform into a mobile self-reconfigurable modular robotic system. An algorithm for controlling the collective locomotion of a group of robots equipped with the modular e-puck extension was also presented. As a consequence of the novel use of 'virtual sensors', without alteration, the same collective locomotion strategy was used to demonstrate self-assembling and self-reconfiguring behaviours.

In section 7.5, the collective locomotion performance of robots arranged in 15 different structural configurations was examined. In order to fully analyse the behaviour of the alignment strategy, a broad survey covering a variety of different types of structure was conducted. To minimise the risk of structures breaking apart, the speed of the individual robots was restricted, as a result, the speed at which the collective structures travel is relatively slow in comparison to the maximum speed of the e-puck. Despite their low speed, it was found that in the majority of cases, even with structures containing up to nine modules, the robots were able to successfully explore an enclosed arena whilst avoiding obstacles. Certain configurations were observed to be more stable than others and it was found that a good (though not universal) relative measure for the stability of similar-sized structures, is the average number of neighbours that the individual modules possess. The system showed promise with regards to its scalability, but it is acknowledged that further insight may be gained by conducting experiments within a larger arena, and using a greater number of modules.

In section 7.6, the robot's ability to perform self-assembly was analysed, focusing on how this behaviour relates to the implementation of 'virtual sensors'. Virtual sensors are imaginary sensors that are notionally located in the blind spots between real sensors and allow the position of neighbouring robots to be inferred more accurately by analysing the intensity of the messages received by the real sensors. In a sparse environment with five modules, robots utilising the virtual sensor approach were found to self-assemble into a single structure significantly quicker than robots that did not utilise virtual sensors. In a denser environment with ten robots, the advantage of virtual sensors was

⁵http://www.elec.york.ac.uk/research/projects/Modular_e_puck_Extension.html

less obvious, however, in analysing the number of structures formed during the final 300 s of each run, it was found that the robots were able to assemble into significantly fewer groups when using the virtual sensor approach. Whilst it is accepted that a more directed form of self-assembly could outperform the strategy from this chapter in terms of speed, this would likely be at the expense of simplicity. Investigating the possibility of more directed forms of self-assembly using the modular e-puck extension would represent an interesting area of future work.

Finally, in section 7.7, a self-disassembling behaviour was described. By combining this behaviour with self-assembly, the more complex task of self-reconfiguration within a changing environment was investigated. Using a ring of LEDs as an environmental trigger, it was demonstrated that the robots were able to successfully switch between self-assembling and self-disassembling behaviours when changes in the ambient lighting conditions were detected.

In comparison to similar platforms, such as the X-Cell [76], DFA [141], s-bot [121] and foot-bot [40] robots, the modular e-puck extension is one of the cheapest and simplest available. Furthermore, it is one of few modular robotic systems for which the designs have been made freely available online.

The simplistic design of the extension means that, like the DFA system, the platform is well suited to the type of passive self-assembly that is more commonly utilised by externally-propelled systems. An interesting consequence of which is that the robots do not require a seed module in order to self-assemble. Assembly is performed in a bottom-up fashion, in which the robots first form pairs or small groups, and only later combine to form single structures.

Simplicity, however, comes at a cost, and the reduced sensing and actuating capabilities of the platform present certain limitations. For example, although the freedom of the rotating frame is beneficial in terms of increasing the flexibility of the system, the lack of control and the absence of the ability to sense its current orientation make the formation of specific configurations difficult. In contrast to the modular e-puck extension, the X-Cell, s-bot and foot-bot platforms, which also possess rotating turrets, allow for fine-grained control and sensing of the orientation of their elements.

Another limitation of the platform is that, at present, the only method that has been developed to allow the robots to disassemble, is the random shaking approach described in section 7.7.1. Whilst this works well when it is desirable for an entire structure to be separated, it is harder to utilise this behaviour in order to remove only one or two modules, for example, during partial self-reconfiguration. The same problem is suffered by the DFA platform, but is circumvented by the X-Cell, s-bot, and foot-bot platforms, in which it is possible to actively control the connection between two modules without disrupting the connections of neighbouring robots. A related problem, as shown in section 7.5, is that it is not always possible to guarantee that the robots will remain connected, especially as the number of modules is increased. This problem is also solved by the X-Cell, s-bot and foot-bot modules through the use of active docking elements.

As highlighted in chapter 1, the design of the modular e-puck extension was partially motivated by the perceived lack of easily accessible platforms within the field of self-

reconfigurable modular robotics. In chapter 1, the importance of fault tolerance to the long-term survival of self-reconfigurable modular robotic systems was also highlighted. In chapter 4, it was suggested that for a system to demonstrate high levels of fault tolerance, greater plasticity in the conformation of the system is required. Through its use of passive docking, the modular e-puck extension is able to provide such plasticity.

In chapter 5, an energy foraging strategy was introduced in which robots were able to share energy by forming small ad-hoc structures. However, this work was conducted entirely within simulation. In the future, it would be desirable to test such behaviours on a physical robotic system. Given the simplicity and ease with which robots equipped with the modular e-puck extension may form ad-hoc structures, this platform represents a viable low-cost system for investigating such behaviours.

In chapter 6, a self-repair strategy was introduced that was designed to work with the Symbricator platform. Implementing a self-repair strategy for the modular e-puck extension would represent another interesting avenue of future work. However, for robots equipped with the modular e-puck extension to perform self-repair, they would first need to be able to demonstrate a more directed form of self-assembly and be capable of forming pre-determined configurations. This may be achieved in an implicit manner by using local alignment rules that have a natural tendency to produce one type of structure over another, or in an explicit manner in which the modules possess a detailed representation of their own structure and use signalling methods to recruit robots at the required locations. Depending upon the method employed, hardware extensions may be necessary. For example, a rotational sensor would make it easier for the robots to determine the relative heading of their neighbours. Furthermore, an actuator for controlling the rotation of the frame, and devices for signalling on specific sides of a module, would aid in directed self-assembly.

In summary, the modular e-puck extension was designed to transform the e-puck robot into a self-reconfigurable modular robotics system. A novel control strategy was developed for the platform which exploits the physical positioning of the sensors on board the e-puck robot. The extension has been used to demonstrate collective locomotion, self-assembly, self-disassembly and self-reconfiguration, both within static and dynamic environments. It has been shown that the platform is capable of providing meaningful experimental results to non-trivial problems and that the simple control strategies developed scale to structures containing up to ten modules. It is concluded that the modular e-puck extension represents a viable, low-cost, platform for research into self-reconfigurable modular robotics.

CHAPTER 8

Conclusions

In this chapter, the main findings and contributions of this thesis are summarised and its primary conclusions are presented. Specifically, in section 8.1, an individual summary of each of the preceding chapters is provided and the main contributions of each chapter are highlighted. In section 8.2, the general hypothesis introduced in chapter 1 is revisited and how the initial aims of this thesis were met is discussed. Finally, in section 8.3, some potential areas of future work are suggested.

8.1 Summary and Contribution

This thesis may be divided into three parts. Chapters 2-3 introduced the necessary background information and reviewed the topics of fault tolerance and self-reconfigurable modular robotic systems. Chapter 4 studied the reliability of an existing morphogenesis controller and laid the foundations for chapters 5 and 6, in which algorithms were developed for detecting faults and recovering from failures that may occur during morphogenesis. Finally, chapter 7 presented the design of a new self-reconfigurable modular robotic system which addressed some of the issues identified in chapters 4-6 and provided a new platform for investigating self-assembling and self-reconfiguring behaviours. Below, each of the preceding chapters (excluding the introduction) is summarised in detail and the main contributions of this thesis are identified:

Chapter 2 - Fault Tolerant Autonomous Robotics This chapter reviewed previous work into analysing the reliability of robotic systems, detecting faults within autonomous robots and recovering from failures within collective robotic systems. Of particular relevance to this thesis was the reliability study of [193], the anomaly detection algorithm of [120] and the self-repair strategies of [24, 179] and [153].

Contribution A review of fault tolerance in autonomous robotic systems. The review highlighted the previous approaches to fault detection, failure recovery and reliability analysis which inspired the approaches developed in chapters 5-7.

Chapter 3 - Self-reconfigurable Modular Robotics This chapter provided an extensive review of the field of self-reconfigurable modular robotics, focusing in particular on the robotic hardware. The review was split in to four sections, each covering one of the four types of system identified in the taxonomy of [207]: chain, lattice, hybrid and mobile. The Symbricator platform and the modular e-puck extension were both identified as mobile forms of self-reconfigurable modular robotic system. In this thesis, the work conducted using the Symbricator platform was said to bear greatest relevance to the Fracta [124] and CKBot [208] systems. The Catom [57], X-Cell [76] and DFA [141] robots, meanwhile, were identified as most pertinent to the modular e-puck extension.

Contribution A review of the field of self-reconfigurable modular robotics. The review identified the various different types of locomotion, communication and docking employed by existing self-reconfigurable modular robots, many of which influenced the design of the new platform introduced in chapter 7.

Chapter 4 - Reliability Analysis and Morphogenesis This chapter introduced a previously developed morphogenesis controller, designed to allow a group of Symbricator robots to autonomously assemble into a robotic structure of predetermined size and shape [108]. The reliability of the controller was analysed using two techniques from the field of reliability engineering: Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA). Both techniques were shown to be effective at highlighting when and how the system may fail and were each identified to have their own advantages and disadvantages. FMEA was said to be capable of providing a good overview of a system's reliability but did not easily reveal which combinations of component failures could lead to a system failure. FTA was observed to be capable of revealing the specific details of system failures but the complexity of self-reconfigurable modular robotic systems limited its applicability. In the controller under study, a total systems failure was identified to be the most detrimental hazard. Weaknesses were also identified in the system's over reliance on wired communications. More generally, the lack of plasticity in self-reconfigurable modular robotic systems was highlighted as a vulnerability.

Contribution A reliability study of the morphogenesis controller from [108], including the first known application of Failure Mode and Effect Analysis (FMEA) and Fault Tree Analysis (FTA) to a self-reconfigurable modular robotic system. A discussion on the suitability of applying such techniques in this context.

Chapter 5 - Energy Foraging and Anomaly Detection This chapter introduced a new energy foraging strategy which allows a group of robots to share energy with one another by forming ad-hoc multi-robot structures. An adapted version of the modified Dendritic Cell Algorithm (mDCA) from [120] was also presented. Improvements were made to the mDCA and its supporting framework and its parameters were optimised using multi-objective optimisation. The adapted mDCA was compared with a version designed to emulate the original algorithm and was

shown to significantly outperform the original both in terms of classification accuracy and evolutionary progress. The two systems were also compared with a state of the art Support Vector Machine (SVM) based approach. The SVM based approach performed better than both versions of the mDCA in terms of classification accuracy but was significantly slower than the mDCA in terms of its run-time speed. The energy foraging strategy and anomaly detection systems were then combined and the task of long-term survival was investigated using a simplified version of the 100 Robots 100 Days grand challenge. The mDCA and SVM based systems were shown to perform significantly better than systems with no anomaly detection and no energy sharing, and on a comparable level to a system with an idealised form of anomaly detection. An interesting result was observed when the FPR of the anomaly detection systems were varied. As the FPR was increased, despite the SVM based approach having a larger TPR, its performance started to drop below that of the mDCA. The reason for this, it was suggested, was due to the cost of recovery outweighing the cost of tolerating anomalies.

Contribution Improvements to the modified Dendritic Cell Algorithm (mDCA) and its supporting framework; increasing the number of features that that algorithm may use to classify data and improving the realism of the robot's sensors and fault models. Optimisation of the mDCA parameters using NSGA-II and a comparison of its classification performance with a Support Vector Machine (SVM) based approach. The development of an energy foraging and energy sharing strategy that allows multiple robots to simultaneously recharge themselves at power sockets or provide energy for other modules. The integration of energy foraging, anomaly detection and fault recovery into a single system and a demonstration that, during extended periods of operation, these behaviours can significantly improve the robots' chances of survival.

Chapter 6 - Self-repairing Robotic Structures This chapter introduced a recovery strategy to augment the morphogenesis controller studied in chapter 4. The strategy was designed to allow a self-reconfigurable modular robotic system to continue operating, despite the presence of failed individuals. Following the introduction of a failure, the strategy works by isolating and removing the failed module, before rebuilding the remainder of the structure in the most efficient manner. A Symbricator-specific implementation of the strategy was described and results from both simulated and real robot experiments were reported. In simulation, the strategy was shown to perform significantly better than a naive recovery approach in which the structures were completely disassembled before being rebuilt from scratch. The self-repair strategy was shown to be particularly advantageous when assembling larger structures. A metric was introduced for classifying structural configurations and the relationship between this value and the speed of self-repair was discussed. As a proof of concept, several of the key properties of the strategy were demonstrated using the Symbricator robots.

Contribution The development of a new self-repair strategy for providing fault tolerant morphogenesis to self-reconfigurable modular robotic systems. The strategy relies on the robots isolating, removing and replacing failed modules with functional spares, without having to completely disassemble the structure and without requiring the failed module to be aware that a failure has occurred. In contrast to other similar approaches the strategy produces an exact copy of the original structure, rather than just an approximation. The introduction of a new metric for classifying robotic structures which is shown to be a good predictor of how well the self-repair strategy will perform on a given structure. An implementation of the strategy for the Symbricator platform. A demonstration of the strategy using real physical robots. The development of a generic reshaping behaviour for transforming between any two structural configurations.

Chapter 7 - Self-assembling and Self-reconfiguring Robotic Structures This chapter described the design and development of a new extension for the e-puck robot. The extension transforms what is traditionally a swarm robotic platform into a self-reconfigurable modular robotic system. The extension consists of a square frame and four magnetic docking interfaces, which sits on top of an e-puck and is able to rotate freely around the central axis of the robot. An algorithm was developed for controlling the collective locomotion of a group of e-pucks equipped with the extension. The algorithm relied on the novel use of ‘virtual sensors’ which allowed to robots to infer the headings of the neighbours more accurately by measuring the intensity of the infrared signals that they received. The success rate, coverage, cohesion, speed and scalability of the approach was investigated in detail, using a wide range of different structural configurations. With no alteration to the underlying algorithm, the system was used to demonstrate self-assembly. With the addition of a self-disassembling behaviour, a form of environment driven self-reconfiguration was demonstrated. It was concluded that extension represents a viable, low-cost platform for investigating the interesting properties of self-reconfigurable modular robotic systems, at an abstract level.

Contribution The design of the *modular e-puck extension*, a new low-cost self-reconfigurable modular robotic system which extends the existing e-puck platform. The platform is cheaper and more accessible than any of the current alternatives and all of the designs are made freely available online. The development of a new collective locomotion behaviour which makes novel use the e-puck’s infrared sensors to infer the headings of other robots. The development of a new distributed self-assembly algorithm which does not require a seed module. In contrast to many other mobile self-reconfigurable systems, in order to assemble the robots first form pairs or triples and only later combine to generate larger structures. The development of a new self-disassembly behaviour which, when combined with the self-assembly algorithm, is used to demonstrate a new form of environment driven self-reconfiguration.

8.2 Concluding Remarks

At the beginning of this thesis, the task of autonomous search and rescue (SAR) was highlighted as a potential scenario which could, in the future, benefit from the use of self-reconfigurable modular robotic systems. Specifically, it was stated that, following a natural disaster, deploying a modular robotic system to search for survivors or monitor environmental conditions, could help to improve the survival rate whilst, at the same time, reducing the risks posed to rescue workers and reducing the financial costs.

One of the most commonly cited advantages of self-reconfigurable modular robotic systems is their high levels of adaptivity. In chapter 1, it was said that the ability to adapt would allow such systems to operate well in dynamic unstructured environments. However, it was also highlighted that if self-reconfigurable modular robotic systems are to be deployed in the real world, they will need to demonstrate the ability to survive autonomously for extended periods of time. To do so, will require the robots to exhibit high degrees of adaptivity to changes in their environment, and high levels of reliability with regards to the presence of failures and faulty individuals. Such adaptivity, it was stated, may be obtained through the development of fault tolerant approaches to self-reconfiguration and morphogenesis. The main aim of this thesis, captured within the following hypothesis, was to study and develop such behaviours.

Hypothesis: *The long-term autonomy of self-reconfigurable modular robotic systems can be improved through the study and development of distributed approaches to fault tolerant morphogenesis. This may be achieved through the design of new robotic platforms, through the study of existing systems, through the development of algorithms for detecting faults in robotic modules, and through the development of strategies for recovering from failures.*

In chapter 4, the reliability of an existing approach to morphogenesis was studied in order to highlight areas where its fault tolerance could be improved. In chapter 5, it was shown that the number of surviving robots and the amount of stored energy was consistently greater in systems equipped with methods for sharing energy, detecting faults and recovering from failures. In chapters 6 and 7, new approaches to collective locomotion, self-repair, self-assembly, self-disassembly and self-reconfiguration were developed and a new platform extension for investigating such behaviours was introduced.

Based upon these contributions, it is believed that all of the aims of this thesis have been met and it has been demonstrated that the long-term survival of self-reconfigurable modular robotic systems can be improved through the study and development of distributed approaches to fault tolerant morphogenesis.

8.3 Future Work

Throughout this thesis, several interesting areas of future work have been highlighted. These area can be categorised into three groups: experimental, theoretical and incremental. The experimental category concerns areas of future work that can be conducted

by performing further experiments with the existing algorithms and techniques. The theoretical group focuses on the insights that may be gained by studying the existing approaches from a theoretical standpoint. Finally, the incremental category concerns the suggested improvements that could be made to the existing algorithms and approaches. Some areas of future work from each one of these three categories are summarised below:

Experimental In chapter 5, it was suggested that further insight could be gained into the performance of the mDCA by optimising its parameters using a greater number of objectives. For example, different measures of classification accuracy could be used or the run-time speed and memory requirements of the approach could be considered.

It was also suggested that the analysis of the energy foraging and energy sharing strategy would benefit from further repeated runs of the long-term survival experiments. Due to the fact that the work in chapter 5 was conducted entirely within simulation, in chapter 7, it was later proposed that the modular e-puck extension could be used to test the energy sharing approach within a real-world environment. Parallels were drawn between the ability of robots equipped with the extension to form small ad-hoc structures and the groups formed by robots when sharing energy.

Similarly, in chapter 6, it was suggested that the self-repair strategy could be subjected to further experimental tests using the real Symbicator robots. Specifically, with the aim of replicating the Robot3D experiments using real hardware.

Finally, in chapter 7, it was proposed that the scalability of the collective locomotion controller could be analysed better if further experiments were conducted within a larger arena. Therefore preventing the time that the robots spend avoiding obstacles from dominating the results.

Theoretical In chapter 4, it was stated that the complexity of the morphogenesis controller made it difficult to study in detail using Fault Tree Analysis (FTA). To address this issue, it was suggested that Dynamic Fault Trees (DFTs), or other mathematical modelling techniques, could be used to help provided further insight. However, in order to perform such analysis, further information about the probability of particular failures occurring would need to be collected.

In chapter 6, mathematical modelling was also proposed as a method for better understanding the dynamics of the self-repair strategy, and in particular, how the shape of a particular structure affects its recovery. Creating a simple stochastic model of the strategy would allow the performance of the system to be tested using many more structural configurations, over a much shorter period of time.

Incremental In chapter 6, several incremental adaptations were suggested for improving the self-repair strategy. One major improvement would be to allow multiple robots to dock as a connected group. To achieve this would require the challenges of precisely coordinating the movement of a structure on a 2D-plane to be overcome.

Another area that would benefit from further investigation is the problem of what

happens if multiple robots fail simultaneously. It is anticipated that this would require changes to be made to the strategy to allow robots to detect failures during self-repair.

The development of an alternative method for calculating the sub-structure score was also proposed. It was suggested that greater efficiency could be achieved if the restriction that the seed robot remained within the same location when forming a new structure was lifted. Methods that allow environmental information to have a greater influence over the repair process were also discussed. For example, to prevent assembly from stalling when there is insufficient space within the arena, the robots could avoid assembling in certain locations or could sense their environment and ‘time-out’ when it becomes difficult or impossible for more robots to dock.

In chapter 7, the task of self-repair was highlighted as an interesting avenue of future work. However, as a precursor to self-repair, it was suggested that a more directed form of self-assembly would need to be developed and additional sensors or actuators may need to be integrated into the modular e-puck extension. These additions may include a sensor for detecting the current orientation of the turret, or actuators for controlling the turret’s rotation or the state of the docking interfaces.

Tasks such as search and rescue and the cleanup of hazardous waste are dangerous and financially expensive activities. In the future, to reduce the financial costs and reduce the risks posed to human workers, several authors have proposed using self-reconfigurable modular robotic systems to perform such tasks. To excel in these areas, systems must be capable of adapting to new circumstances, tolerating the presence of faults and surviving for long periods of time without any form of human interaction.

In this thesis, it has been demonstrated that, in an abstract setting, the long-term survival of self-reconfigurable modular robotic systems can be improved through the study and development of new and existing approaches to fault tolerant morphogenesis. Through further development of such strategies, future self-reconfigurable modular robotic systems may be developed which are deployable in real world scenarios such as search and rescue. Following natural disasters, it is envisaged that such systems will be capable of improving survival rates, reducing financial costs, improving the well-being and reducing the risks posed to future generations.

Bibliography

- [1] M. K. Ackerman and G. S. Chirikjian. Hex-DMR: A modular robotic test-bed for demonstrating team repair. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2012*, pages 4148–4153. IEEE, 2012.
- [2] M. S. Ahmed, R. Saatchi, and F. Caparrelli. Support for robot docking and energy foraging: a computer vision approach. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2012)*, Rome, Italy, February 2012.
- [3] M. S. Ahmed, R. Saatchi, and F. Caparrelli. Vision based obstacle avoidance and odometry for swarms of small size robots. In *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2012)*, Rome, Italy, February 2012.
- [4] D. Arney, S. Fischmeister, I. Lee, Y. Takashima, and M. Yim. Model-based programming of modular robots. In *Proceedings of the 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2010*, pages 66–74. IEEE, 2010.
- [5] Auvation. OpenFTA, 2011. [Computer software] <http://www.openfta.com>.
- [6] G. Baele, N. Bredeche, E. Haasdijk, S. Maere, N. Michiels, Y. Van de Peer, T. Schmickl, C. Schwarzer, and R. Thenius. Open-ended on-board evolutionary robotics for robot swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009*, pages 1123–1130. IEEE, 2009.
- [7] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Dorigo, and S. Nolfi. Self-organized coordinated motion in groups of physically connected robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):224–239, 2007.
- [8] C. Bererton and P. Khosla. Towards a team of robots with reconfiguration and repair capabilities. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2001*, volume 3, pages 2923–2928. IEEE, 2001.
- [9] C. Bererton and P. Khosla. Towards a team of robots with repair capabilities: a visual docking system. *Experimental Robotics VII*, 271:333–342, 2001.

- [10] C. Bererton and P. Khosla. An analysis of cooperative repair capabilities in a team of robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002*, volume 1, pages 476–482. IEEE, 2002.
- [11] R. Bi, J. Timmis, and A. Tyrrell. The diagnostic dendritic cell algorithm for robotic systems. In *Proceedings of the IEEE International Congress on Evolutionary Computation, CEC 2010*, pages 1–8. IEEE, 2010.
- [12] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Programmable parts: a demonstration of the grammatical approach to self-organization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005*, pages 3684–3691. IEEE, 2005.
- [13] N. Bredeche, J. Montanier, W. Liu, and A. Winfield. Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129, 2012.
- [14] R. A. Brooks and A. M. Flynn. Fast, cheap and out of control: A robot invasion of the solar system. *Journal of the British Interplanetary Society*, 42:478–485, 1989.
- [15] A. Campo and M. Dorigo. Efficient multi-foraging in swarm robotics. In *Advances in Artificial Life*, pages 696–705. Springer, 2007.
- [16] A. Campo, S. Nouyan, M. Birattari, R. Groß, and M. Dorigo. Negotiation of goal direction for cooperative transport. *Ant Colony Optimization and Swarm Intelligence*, 4150:191–202, 2006.
- [17] R. Canham, A. H. Jackson, and A. Tyrrell. Robot error detection using an artificial immune system. In *Proceedings of the NASA/DoD International Conference on Evolvable Hardware*, pages 199–207. IEEE, 2003.
- [18] J. Carlson and R. R. Murphy. Reliability analysis of mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2003*, volume 1, pages 274–281. IEEE, 2003. doi: 10.1109/ROBOT.2003.1241608.
- [19] J. Carlson and R. R. Murphy. How UGVs physically fail in the field. *IEEE Transactions on Robotics*, 21(3):423–437, 2005. ISSN 1552-3098. doi: 10.1109/TRO.2004.838027.
- [20] J. Carlson, R. R. Murphy, and A. Nelson. Follow-up analysis of mobile robot failures. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2004*, volume 5, pages 4987–4994. IEEE, 2004. doi: 10.1109/ROBOT.2004.1302508.

- [21] A. Castano and P. Will. Representing and discovering the configuration of conro robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2001*, volume 4, pages 3503–3509. IEEE, 2001.
- [22] A. Castano, W. M. Shen, and P. Will. CONRO: Towards deployable robots with inter-robots metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, 2000.
- [23] A. Castano, A. Behar, and P. M. Will. The CONRO modules for reconfigurable robots. *IEEE/ASME Transactions on Mechatronics*, 7(4):403–409, 2002.
- [24] J. Cheng, W. Cheng, and R. Nagpal. Robust and self-repairing formation control for swarms of mobile agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 59. MIT Press, 2005.
- [25] H. Chiu, M. Rubenstein, and W. M. Shen. “deformable wheel” - a self-recovering modular rolling track. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems, DARS 2008*, 2008.
- [26] H. C. H. Chiu, M. Rubenstein, and W. M. Shen. Multifunctional SuperBot with rolling track configuration. In *IROS 2007 Workshop on Self-Reconfigurable Robots & Systems and Applications*, pages 50–53, 2007.
- [27] A. Christensen, R. O’Grady, M. Birattari, and M. Dorigo. Exogenous fault detection in a collective robotic task. In F. Almeida e Costa, L. Rocha, E. Costa, I. Harvey, and A. Coutinho, editors, *Advances in Artificial Life*, volume 4648 of *LNCS*, pages 555–564. Springer, 2007.
- [28] A. Christensen, R. O’Grady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24: 49–67, 2008. ISSN 0929-5593.
- [29] A. Christensen, R. O’Grady, and M. Dorigo. From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4):754–766, 2009.
- [30] D. J. Christensen. Evolution of shape-changing and self-repairing control for the ATRON self-reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2006*, pages 2539–2545. IEEE, may 2006. doi: 10.1109/ROBOT.2006.1642084.
- [31] D. J. Christensen. Experiments on fault-tolerant self-reconfiguration and emergent self-repair. In *Proceedings of the IEEE Symposium on Artificial Life, ALIFE 2007*, pages 355–361. IEEE, 2007.
- [32] K. Chu, S. Hossain, and C. Nelson. Design of a four-DOF modular self-reconfigurable robot with novel gaits. In *Proceedings of the ASME International Design Engineering Technical Conference*, 2011.

- [33] C. M. Cianci, X. Raemy, J. Pugh, and A. Martinoli. Communication in a swarm of miniature robots: the e-puck as an educational tool for swarm robotics. In *Proceedings of the 2nd international conference on Swarm robotics, SAB 2006*, pages 103–115. Springer, 2007. ISBN 978-3-540-71540-5.
- [34] K. W. Dailey. *The FMEA Pocket Handbook*. DW Publishing Co., 2004.
- [35] L. T. Dauer, P. Zanzonico, R. M. Tuttle, D. M. Quinn, and H. W. Strauss. The Japanese tsunami and resulting nuclear emergency at the Fukushima Daiichi power facility: technical, radiologic, and response perspectives. *Journal of Nuclear Medicine*, 52(9):1423–1432, 2011.
- [36] M. Dauschan, R. Thenius, T. Schmickl, and K. Crailsheim. Using virtual embryogenesis in multi-robot organisms. In A. Bouchachia, editor, *Adaptive and Intelligent Systems*, volume 6943 of *LNCIS*, pages 238–247. Springer, 2011.
- [37] J. Davey, N. Kwok, and M. Yim. Emulating self-reconfigurable robots-design of the SMORES system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012*, pages 4464–4469. IEEE, 2012.
- [38] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag, London, UK, 2002. ISBN 978-1-85233-594-6.
- [39] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
- [40] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonan, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugnière, G. A. D. Caro, F. Ducatelle, E. Ferrante, A. Förster, J. M. Gonzales, V. L. J. Guzzi, S. Magnenat, N. Mathews, M. M. de Oca, R. O’Grady, C. Pinciroli, G. Pini, P. Rétoznaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stützle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 2013, *in press.*, 2013.
- [41] D. Duff, M. Yim, and K. Roufas. Evolution of PolyBot: A modular reconfigurable robot. In *Proceedings of the Harmonic Drive International Symposium, Nagano, Japan*, 2001.
- [42] J. Dugan, S. Bavuso, and M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992. ISSN 0018-9529. doi: 10.1109/24.159800.

- [43] A. Dutta, P. Dasgupta, J. Baca, and C. Nelson. A fast coalition structure search algorithm for modular robot reconfiguration planning under uncertainty. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems, DARS 2012*, 2012.
- [44] S. English, J. Gough, A. Johnson, R. Spanton, and J. Sun. Formica, 2012. <http://warrantyvoidifremoved.com/formica/> [Accessed 29 August 2013].
- [45] E. Ferrante, M. Brambilla, M. Birattari, and M. Dorigo. Socially-mediated negotiation for obstacle avoidance in collective transport. *Distributed Autonomous Robotic Systems*, 83:571–583, 2010.
- [46] T. Fukuda and Y. Kawauchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1990*, pages 662–667. IEEE, 1990. doi: 10.1109/ROBOT.1990.126059.
- [47] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1988*, pages 1581–1586. IEEE, 1988. doi: 10.1109/ROBOT.1988.12291.
- [48] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures - CEBOT. In *Proceedings of the IEEE International Workshop on Intelligent Robots*, pages 145 –150. IEEE, 1988. doi: 10.1109/IROS.1988.592421.
- [49] GCtronic. Elisa 3, 2012. <http://www.gctronic.com/doc/index.php/Elisa-3> [Accessed 29 August 2013].
- [50] K. Gilpin and D. Rus. Modular robot systems. *Robotics Automation Magazine*, 17(3):38–55, 2010. ISSN 1070-9932. doi: 10.1109/MRA.2010.937859.
- [51] K. Gilpin and D. Rus. Self-disassembling robot pebbles: New results and ideas for self-assembly of 3D structures. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 94–99. IEEE, 2010.
- [52] K. Gilpin and D. Rus. A distributed algorithm for 2D shape duplication with smart pebble robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2012*, pages 3285–3292. IEEE, 2012. doi: 10.1109/ICRA.2012.6225227.
- [53] K. Gilpin, K. Kotay, and D. Rus. Miche: Modular shape formation by self-dissassembly. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2007*, pages 2241–2247. IEEE, 2007.

- [54] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu. Miche: Modular shape formation by self-disassembly. *International Journal of Robotics Research*, 27:345–372, March 2008.
- [55] K. Gilpin, A. Knaian, and D. Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 2485–2492. IEEE, 2010.
- [56] K. Gilpin, K. Koyanagi, and D. Rus. Making self-disassembling objects with multiple components in the robot pebbles system. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2011*, pages 3614–3621. IEEE, 2011. doi: 10.1109/ICRA.2011.5980305.
- [57] S. C. Goldstein, J. D. Campbell, and T. C. Mowry. Programmable matter. *Computer*, 38(6):99–101, may 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.198.
- [58] J. González-Gómez, E. Aguayo, and E. Boemo. Locomotion of a modular worm-like robot using a FPGA-based embedded MicroBlaze soft-processor. In *Proceedings of the 7th International Conference on Climbing and Walking Robots, CLAWAR 2004*, pages 869–878. Springer, 2005.
- [59] J. Greensmith, U. Aickelin, and S. Cayzer. Introducing dendritic cells as a novel immune-inspired algorithm for anomaly detection. In C. Jacob, M. Pilat, P. Bentley, and J. Timmis, editors, *Artificial Immune Systems*, volume 3627 of *LNCS*, pages 153–167. Springer, 2005.
- [60] R. Groß and M. Dorigo. Self-assembly at the macroscopic scale. *Proceedings of the IEEE*, 96(9):1490–1508, 2008.
- [61] R. Groß and M. Dorigo. Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1-2):1–13, 2009.
- [62] R. Groß, M. Bonani, F. Mondada, and M. Dorigo. Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, December 2006. ISSN 1552-3098.
- [63] R. Groß, F. Mondada, and M. Dorigo. Transport of an object by six pre-attached robots interacting via physical links. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2006*, pages 1317–1323. IEEE, 2006.
- [64] R. Groß, S. Magnenat, L. Küchler, V. Massaras, M. Bonani, and F. Mondada. Towards an autonomous evolution of non-biological physical organisms. In *Proceedings of the 10th European Conference on Artificial Life, ECAL 2009*, volume 5777 of *LNCS*, pages 173–180. Springer, 2011.

- [65] A. Gutiérrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena. Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 3111–3116. IEEE, May 2009. doi: 10.1109/ROBOT.2009.5152456.
- [66] A. Gutiérrez, E. Tuci, and A. Campo. Evolution of neuro-controllers for robots alignment using local communication. *International Journal of Advanced Robotic Systems*, 6(1):25–34, March 2009.
- [67] E. Haasdijk, A. E. Eiben, and G. Karafotias. On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the IEEE International Congress on Evolutionary Computation, CEC 2010*, pages 1–7. IEEE, July 2010.
- [68] E. Haasdijk, A. Rusu, and A. Eiben. HyperNEAT for locomotion control in modular robots. In G. Tempesti, A. Tyrrell, and J. Miller, editors, *Evolvable Systems: From Biology to Hardware*, volume 6274 of *LNCS*, pages 169–180. Springer, 2010. ISBN 978-3-642-15322-8.
- [69] H. Hamann, J. Stradner, T. Schmickl, and K. Crailsheim. A hormone-based controller for evolutionary multi-modular robotics: From single modules to gait learning. In *Proceedings of the IEEE International Congress on Evolutionary Computation, CEC 2010*, pages 1–8. IEEE, July 2010. doi: 10.1109/CEC.2010.5585994.
- [70] H. Hamann, J. Stradner, T. Schmickl, and K. Crailsheim. Artificial hormone reaction networks: Towards higher evolvability in evolutionary multi-modular robotics. In *Proceedings of the ALife XII Conference*, pages 773–780. MIT Press, 2010.
- [71] G. J. Hamlin and A. C. Sanderson. Tetrobot modular robotics: Prototype and experiments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1996*, volume 2, pages 390–395. IEEE, 1996.
- [72] G. J. Hamlin and A. C. Sanderson. Tetrobot: A modular approach to parallel robotics. *Robotics & Automation Magazine*, 4(1):42–50, 1997.
- [73] M. Hashimoto, R. Kitade, F. Itaba, and K. Takahashi. Voting based fault isolation of in-vehicle multi-sensors. In *Proceedings of the SICE Annual Conference*, pages 1942–1946, 2008. doi: 10.1109/SICE.2008.4654979.
- [74] M. Hashimoto, T. Ishii, and K. Takahashi. Sensor fault detection and isolation for mobile robots in a multi-robot team. In *Proceedings of the 35th Annual Conference of Industrial Electronics, IECON 2009*, pages 2348–2353. IEEE, 2009. doi: 10.1109/IECON.2009.5415410.

- [75] G. Heredia, A. Ollero, M. Bejar, and R. Mahtani. Sensor and actuator fault detection in small autonomous helicopters. *Mechatronics*, 18(2):90–99, 2008. ISSN 0957-4158. doi: DOI:10.1016/j.mechatronics.2007.09.007.
- [76] W. Hong, S. Wang, and D. Shui. Reconfigurable robot system based on electromagnetic design. In *Proceedings of the 2011 International Conference on Fluid Power and Mechatronics, FPM 2011*, pages 570–575. IEEE, 2011.
- [77] R. Humza and O. Scholz. Energy autonomy and energy harvesting in recongruable swarm robotics. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms*, volume 7 of *Cognitive Systems Monographs*, pages 116–135. Springer, 2010.
- [78] R. Humza, O. Scholz, M. Mokhtar, J. Timmis, and A. Tyrrell. Towards energy homeostasis in an autonomous self-reconfigurable modular robotic organism. In *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, COMPUTATIONWORLD 2009*, pages 21–26, November 2009.
- [79] A. R. Ismail and J. Timmis. Towards self-healing swarm robotic systems inspired by granuloma formation. In *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2010*, pages 313–314. IEEE, 2010.
- [80] M. W. Jørgensen, E. H. Østergaard, and H. H. Lund. Modular ATRON: modules for a self-reconfigurable robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2004*, volume 2, pages 2068–2073. IEEE, 2004. doi: 10.1109/IROS.2004.1389702.
- [81] K-Team Corporation. K-Team Mobile Robotics, 2012. <http://www.k-team.com> [Accessed 29 August 2013].
- [82] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS 2004*, volume 3, pages 2370–2377. IEEE, 2004.
- [83] B. Kannan and L. E. Parker. Fault-tolerance based metrics for evaluating system performance in multi-robot teams. In *Proceedings of Performance Metrics for Intelligent Systems Workshop*, 2006.
- [84] S. Kawatsuma, M. Fukushima, and T. Okada. Emergency response by robots to Fukushima-Daiichi accident: summary and lessons learned. *Industrial Robot: An International Journal*, 39(5):428–435, 2012.
- [85] S. Kernbach. Jasmine swarm robot platform, 2012. <http://www.swarmrobot.org> [Accessed 29 August 2013].

- [86] S. Kernbach and O. Kernbach. Collective energy homeostasis in a large-scale microrobotic swarm. *Robotics and Autonomous Systems*, 59(12):1090–1101, 2011.
- [87] S. Kernbach, P. Levi, E. Meister, F. Schlachter, and O. Kernbach. Towards self-adaptation of robot organisms with a high developmental plasticity. In *Proceedings of the First International Conference on Adaptive and Self-adaptive Systems and Applications, ADAPTIVE 2009*, pages 180–187, 2009. doi: 10.1109/ComputationWorld.2009.11.
- [88] S. Kernbach, O. Scholz, K. Harada, S. Popesku, J. Liedke, H. Raja, W. Liu, F. Caparrelli, J. Jemai, J. Havlik, E. Meister, and P. Levi. Multi-Robot Organisms: State of the Art. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA10), workshop on “Modular Robots: State of the Art”*, Anchorage, Alaska, pages 1–10, 2010.
- [89] S. Kernbach, B. Girault, and O. Kernbach. On self-optimized self-assembling of heterogeneous multi-robot organisms. In Y. Meng and Y. Jin, editors, *Bio-Inspired Self-Organizing Robotic Systems*, volume 355 of *Studies in Computational Intelligence*, pages 123–141. Springer, 2011. ISBN 978-3-642-20759-4.
- [90] S. Kernbach, F. Schlachter, R. Humza, J. Liedke, S. Popesku, S. Russo, T. Ranzani, L. Manfredi, C. Stefanini, R. Matthias, C. Schwarzer, B. Girault, P. Alschbach, E. Meister, and O. Scholz. Heterogeneity for increasing performance and reliability of self-reconfigurable multi-robot organisms. IROS 2011, workshop on “Reconfigurable Modular Robotics”, San Francisco, 2011.
- [91] B. T. Kirby, B. Aksak, J. D. Campbell, J. F. Hoburg, T. C. Mowry, P. Pillai, and S. C. Goldstein. A modular robotic system using magnetic force effectors. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007*, pages 2787–2793. IEEE, 2007. doi: 10.1109/IROS.2007.4399444.
- [92] K. Kotay and D. Rus. Motion synthesis for the self-reconfiguring molecule. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1998*, volume 2, pages 843–851. IEEE, 1998. doi: 10.1109/IROS.1998.727304.
- [93] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1998*, volume 1, pages 424–431. IEEE, 1998. doi: 10.1109/ROBOT.1998.676452.
- [94] T. Krajník, J. Faigl, V. Vonásek, K. Košnar, M. Kulich, and L. Přeučil. Simple yet stable bearing-only navigation. *Journal of Field Robotics*, 27(5):511–533, 2010.

- [95] T. Krajník, J. Faigl, V. Vonásek, H. Szücssová, O. Fišer, and L. Přeučil. A visual navigation system for RoboTour competition. In *Proceedings of the 1st international conference on Robotics in Education, RiE2010*, pages 95–100. FEI STU, Slovakia, 2010.
- [96] M. Kubo and C. Melhuish. Robot trophallaxis: Managing energy autonomy in multiple robots. In *Proceedings of Towards Autonomous Robotic Systems, TAROS 2004*, 2004.
- [97] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, and S. Murata. M-TRAN II: Metamorphosis from a four-legged walker to a caterpillar. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems, IROS 2003*, volume 3, pages 2454–2459. IEEE, 2003.
- [98] H. Kurokawa, E. Yoshida, K. Tomita, A. Kamimura, S. Murata, and S. Kokaji. Self-reconfigurable M-TRAN structures and walker generation. *Robotics and Autonomous Systems*, 54(2):142–149, 2006.
- [99] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Self-reconfigurable modular robot M-TRAN: distributed control and communication. In *Proceedings of the 1st international conference on Robot communication and coordination*, page 21, 2007.
- [100] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata. Distributed self-reconfiguration of M-TRAN III modular robotic system. *The International Journal of Robotics Research*, 27(3-4):373–386, 2008.
- [101] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labor in a group of robots inspired by ants’ foraging behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):4–25, 2006.
- [102] N. Laframboise and B. Loko. Natural disasters: Mitigating impact, managing risks, 2012. IMF Working Paper.
- [103] J.-C. Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Proceedings of the 25th International Symposium on Fault-Tolerant Computing, ‘Highlights from Twenty-Five Years’*, page 2, 1995. doi: 10.1109/FTCSH.1995.532603.
- [104] H. Lau, J. Timmis, and I. Bate. Collective self-detection scheme for adaptive error detection in a foraging swarm of robots. In *Proceedings of the 10th International Conference on Artificial Immune Systems, ICARIS 2011*, pages 254–267. Springer, 2011.
- [105] P. Levi and S. Kernbach. *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, volume 7 of *Cognitive Systems Monographs*. Springer Berlin Heidelberg, 2010.

- [106] J. Liedke and H. Wörn. CoBoLD: A bonding mechanism for modular self-reconfigurable mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics, ROBIO 2011*, pages 2025–2030, December 2011.
- [107] W. Liu and A. Winfield. Implementation of an IR approach for autonomous docking in a self-configurable robotics system. In T. Kyriacou, U. Nehmzow, C. Melhuish, and M. Witkowski, editors, *Proceedings of Towards Autonomous Robotic Systems, TAROS 2009*, pages 251 – 258, September 2009.
- [108] W. Liu and A. F. T. Winfield. Autonomous morphogenesis in self-assembling robots using IR-based sensing and local communications. In *Proceedings of ANTS 2010, 7th international conference on Swarm intelligence*, pages 107–118. Springer, 2010. ISBN 3-642-15460-3, 978-3-642-15460-7.
- [109] W. Liu and A. F. T. Winfield. Open-hardware e-puck linux extension board for experimental swarm robotics research. *Microprocessors and Microsystems*, 35(1): 60–67, 2011. ISSN 0141-9331. doi: 10.1016/j.micpro.2010.08.002.
- [110] W. Liu and A. F. T. Winfield. Distributed autonomous morphogenesis in a self-assembling robotic system. In R. Doursat, H. Sayama, and O. Michel, editors, *Morphogenetic Engineering*, pages 89–113. Springer, 2012.
- [111] W. Liu, A. F. Winfield, J. Sa, J. Chen, and L. Dou. Towards energy optimization: Emergent task allocation in a swarm of foraging robots. *Adaptive Behavior*, 15(3):289–305, 2007.
- [112] R. Matthias and H. Wörn. Utilizing the full potential of a new flexible platform in modular self-reconfigurable mobile robotics. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics, ROBIO 2011*, pages 2712–2717, 2011.
- [113] P. Matzinger. Tolerance, danger, and the extended family. *Annual review of immunology*, 12(1):991–1045, 1994.
- [114] R. E. McDermott, R. J. Mikulak, and M. R. Beauregard. *The Basics of FMEA*. Productivity Press, 2008. ISBN 9781563273773.
- [115] E. Meister, S. Stepanenko, and S. Kernbach. Adaptive locomotion of multibody snake-like robot. In *Multibody Dynamics 2011, ECCOMAS Thematic Conference*, Brussels, Belgium, July 2011.
- [116] C. Melhuish and M. Kubo. Collective energy distribution: Maintaining the energy balance in distributed autonomous robots using trophallaxis. In *Distributed Autonomous Robotic Systems 6*, pages 275–284. Springer, 2007.

- [117] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. Ijspeert. YaMoR and Bluemove - an autonomous modular robot with bluetooth interface for exploring adaptive locomotion. In *Proceedings of the International Conference on Climbing and Walking Robots, CLAWAR 2006*, pages 685–692. Springer, 2006.
- [118] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. J. Ijspeert. Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with bluetooth interface. *Industrial Robot: An International Journal*, 33(4):285–290, 2006.
- [119] M. Mokhtar, J. Timmis, A. M. Tyrrell, and R. Bi. An artificial lymph node architecture for homeostasis in collective robotic systems. In *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2008*, pages 126–131. IEEE, October 2008.
- [120] M. Mokhtar, R. Bi, J. Timmis, and A. M. Tyrrell. A modified dendritic cell algorithm for on-line error detection in robotic systems. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009*, pages 2055–2062. IEEE, 2009.
- [121] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J.-L. Deneubourg, S. Nolfi, L. M. Gambardella, and M. Dorigo. Swarm-bot: A new distributed robotic concept. *Autonomous Robots*, 17:193–221, 2004. ISSN 0929-5593.
- [122] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. christophe Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65, 2009.
- [123] J.-M. Montanier and N. Bredèche. Embedded evolutionary robotics: The (1+1)-restart-online adaptation algorithm. In S. Doncieux, N. Bredèche, and J.-B. Mouret, editors, *New Horizons in Evolutionary Robotics*, volume 341 of *Studies in Computational Intelligence*, pages 155–169. Springer, 2011. ISBN 978-3-642-18271-6.
- [124] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation, ICRA 1994*, pages 441–448. IEEE, may 1994.
- [125] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1998*, volume 1, pages 432–439 vol.1. IEEE, 1998. doi: 10.1109/ROBOT.1998.677012.

- [126] S. Murata, H. Kurokawa, and S. Kokaji. Self-configurable machine. *Transactions of the Society of Instrument and Control Engineers*, 1(1):187–196, 2001. Reprinted/Translated from *Trans. SICE*, Vol.31, No.2, 254/262 (1995).
- [127] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-TRAN: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, 2002.
- [128] S. Murata, K. Kakomura, and H. Kurokawa. Docking experiments of a modular robot by visual feedback. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006*, pages 625–630. IEEE, 2006.
- [129] L. Murray, W. Liu, A. Winfield, J. Timmis, and A. Tyrrell. Analysing the reliability of a self-reconfigurable modular robotic system. In *Proceedings of the 6th International Conference on Bio-Inspired Models of Network, Information, and Computing Systems, BIONETICS 2011*, December 2011.
- [130] L. Murray, J. Timmis, and A. Tyrrell. Self-reconfigurable modular e-pucks. In *Proceedings of ANTS 2012, 8th international conference on Swarm intelligence*, volume 7461 of *LNCS*, pages 133–144. Springer, 2012.
- [131] L. Murray, J. Timmis, and A. Tyrrell. Modular self-assembling and self-reconfiguring e-pucks. *Swarm Intelligence*, 7(2-3):83–113, 2013.
- [132] C. Nelson, K. Chu, and P. Dasgupta. ModRED: A modular self-reconfigurable robot for autonomous extra-terrestrial exploration and discovery. In *Planetary Rovers Workshop, (colocated with the International Conference on Robotics and Automation, ICRA 2010), Anchorage, AK*, 2010.
- [133] J. Neubert, A. P. Cantwell, S. Constantin, M. Kalontarov, D. Erickson, and H. Lipson. A robotic module for stochastic fluidic assembly of 3D self-reconfiguring structures. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 2479–2484. IEEE, 2010. doi: 10.1109/ROBOT.2010.5509455.
- [134] T. D. Ngo and H. Schioler. An approach to sociable robots through self-distributed energy. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006*, pages 2192–2199. IEEE, 2006.
- [135] R. O’Grady, A. Christensen, and M. Dorigo. Autonomous reconfiguration in a self-assembling multi-robot system. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Sttzle, and A. Winfield, editors, *Proceedings of ANTS 2008, 6th International Conference on Ant Colony Optimization and Swarm Intelligence*, volume 5217 of *LNCS*, pages 259–266. Springer, 2008. ISBN 978-3-540-87526-0.

- [136] R. O’Grady, A. Christensen, and M. Dorigo. SWARMORPH: Multirobot morphogenesis using directional self-assembly. *IEEE Transactions on Robotics*, 25(3): 738–743, 2009. ISSN 1552-3098. doi: 10.1109/TRO.2008.2012341.
- [137] R. O’Grady, C. Pinciroli, R. Groß, A. Christensen, F. Mondada, M. Bonani, and M. Dorigo. Swarm-bots to the rescue. In G. Kampis, I. Karsai, and E. Szathmry, editors, *Advances in Artificial Life. Darwin Meets von Neumann*, volume 5777 of *LNC3*, pages 165–172. Springer, 2011. ISBN 978-3-642-21282-6.
- [138] K. Ohno, S. Kawatsuma, T. Okada, E. Takeuchi, K. Higashi, and S. Tadokoro. Robotic control vehicle for measuring radiation in Fukushima Daiichi nuclear power plant. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics, SSR 2011*, pages 38–43. IEEE, 2011.
- [139] E. Østergaard and H. Lund. Distributed cluster walk for the ATRON self-reconfigurable robot. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, and B. Kröse, editors, *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 291–298. IOS Press, 2004.
- [140] E. Østergaard, K. Kassow, R. Beck, and H. Lund. Design of the ATRON lattice-based self-reconfigurable robot. *Autonomous Robots*, 21:165–183, 2006. ISSN 0929-5593. 10.1007/s10514-006-8546-1.
- [141] R. Oung and R. D’Andrea. The distributed flight array. *Mechatronics*, 21(6): 908–917, 2011. ISSN 0957-4158.
- [142] N. Owens, A. Greensted, J. Timmis, and A. Tyrrell. T cell receptor signalling inspired kernel density estimation and anomaly detection. In *Proceedings of the 8th International Conference on Artificial Immune Systems, ICARIS 2009*, pages 122–135. Springer, 2009.
- [143] A. Pamecha, C.-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proceedings of the ASME International Design Engineering Technical Conference and Computers in Engineering Conference, Irvine, California*. ASME, 1996.
- [144] M. Park and M. Yim. Distributed control and communication fault tolerance for the CKBot. In *Proceedings of the ASME/IFTOMM International Conference on Reconfigurable Mechanisms and Robots, ReMAR 2009*, pages 682–688. ASME/IFTOMM, June 2009.
- [145] L. E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
- [146] Pololu Corporation. Pololu Robotics & Electronics, 2012. <http://www.pololu.com> [Accessed 29 August 2013].

- [147] L. Přeučil, P. Stepan, T. Krajník, K. Košnar, A. van Rossum, and A. Salden. Cognitive world modelling. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms*, volume 7 of *Cognitive Systems Monographs*, pages 165–183. Springer, 2010.
- [148] V. Rai, A. van Rossum, and N. Correll. Self-assembly of modular robots from finite number of modules using graph grammars. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011*, pages 4783–4789. IEEE, September 2011. doi: 10.1109/IROS.2011.6095038.
- [149] Z. Ramaekers, P. Dasgupta, V. Ufimtsev, S. Hossain, and C. Nelson. Self-reconfiguration in modular robots using coalition games with uncertainty. In *AAAI Workshop on Autonomous Action Planning for Autonomous Mobile Robots (PAMR)*, pages 57–63, 2011.
- [150] N. Ranasinghe, J. Everist, and W. M. Shen. Modular robot climbers. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007*. IEEE, 2007.
- [151] REPLICATOR. *REPLICATOR: Robotic Evolutionary Self-Programming and Self-Assembling Organisms, 7th Framework Programme Project No FP7- ICT-2007.2.1, grant agreement 216240*. European Communities, 2008-2012.
- [152] S. Revzen, M. Bhoite, A. Macasieb, and M. Yim. Structure synthesis on-the-fly in a modular robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011*, pages 4797–4802. IEEE, 2011.
- [153] M. Rubenstein and W. M. Shen. A scalable and distributed model for self-organization and self-healing. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1179–1182, 2008.
- [154] M. Rubenstein, K. Payne, P. Will, and W. M. Shen. Docking among independent and autonomous CONRO self-reconfigurable robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2004*, volume 3, pages 2877–2882. IEEE, 2004.
- [155] M. Rubenstein, N. Hoff, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. Technical Report TR-06-11, Harvard University, June 2011. <ftp://ftp.deas.harvard.edu/techreports/tr-06-11.pdf> [Accessed 11 February 2012].
- [156] M. Rubenstein and W. M. Shen. Scalable self-assembly and self-repair in a collective of robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 1484–1489. IEEE, 2009.

- [157] D. Rus and M. Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2000*, volume 2, pages 1726–1733. IEEE, 2000. doi: 10.1109/ROBOT.2000.844845.
- [158] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10:107–124, 2001. ISSN 0929-5593. 10.1023/A:1026504804984.
- [159] G. G. Ryland and H. H. Cheng. Design of iMobot, an intelligent reconfigurable mobile robot with novel locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 60–65. IEEE, 2010.
- [160] E. Şahin and W. M. Spears. *Swarm robotics: SAB 2004 international workshop, Santa Monica, CA, USA, July 17, 2004 : revised selected papers*. LNCS. Springer, 2005. ISBN 9783540242963.
- [161] B. Salemi, W. M. Shen, and P. Will. Hormone-controlled metamorphic robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 200*, volume 4, pages 4194–4199. IEEE, 2001.
- [162] B. Salemi, M. Moll, and W. M. Shen. SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006*, pages 3636–3641, 2006.
- [163] J. Sastra, W. G. B. Heredia, J. Clark, and M. Yim. A biologically-inspired dynamic legged locomotion with a modular reconfigurable robot. In *ASME Conference Proceedings*, pages 1467–1474. ASME, 2008. doi: 10.1115/DSCC2008-2402.
- [164] J. Sastra, S. Chitta, and M. Yim. Dynamic rolling for a modular loop robot. *International Journal of Robotics Research*, 28:758–773, June 2009.
- [165] E. Schweikardt. Modular robotics studio. In *Proceedings of the fifth international conference on tangible, embedded, and embodied interaction*, pages 353–356. ACM, 2011.
- [166] W. M. Shen and P. Will. Docking in self-reconfigurable robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2001*, volume 2, pages 1049–1054. IEEE, 2001.
- [167] W. M. Shen, B. Salemi, and P. Will. Hormone-inspired adaptive communication and distributed control for conro self-reconfigurable robots. *IEEE Transactions on Robotics and Automation*, 18(5):700–712, 2002.

- [168] W. M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh. Multimode locomotion via SuperBot reconfigurable robots. *Autonomous Robots*, 20(2):165–177, 2006.
- [169] W. M. Shen, R. Kovac, and M. Rubenstein. SINGO: a single-end-operative and genderless connector for self-reconfiguration, self-assembly and self-healing. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 4253–4258. IEEE, 2009.
- [170] A. Sproewitz, M. Asadpour, A. Billard, P. Dillenbourg, and A. Ijspeert. Roombots—modular robots for adaptive furniture. In *IROS Workshop on Self-Reconfigurable Robots, Systems and Applications*, 2008.
- [171] A. Sproewitz, M. Asadpour, Y. Bourquin, and A. J. Ijspeert. An active connection mechanism for modular self-reconfigurable robotic systems based on physical latching. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2008*, pages 3508–3513. IEEE, 2008.
- [172] A. Sproewitz, A. Billard, P. Dillenbourg, and A. J. Ijspeert. Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2009*, pages 4259–4264. IEEE, 2009.
- [173] A. Sproewitz, S. Pouya, S. Bonardi, J. Van den Kieboom, R. Mockel, A. Billard, P. Dillenbourg, and A. J. Ijspeert. Roombots: reconfigurable robots for adaptive furniture. *Computational Intelligence Magazine, IEEE*, 5(3):20–32, 2010.
- [174] S. B. Stancliff and J. M. Dolan. Mission reliability estimation for multi-robot team design. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006*, pages 2206–2211. IEEE, 2006.
- [175] J. W. Suh, S. B. Homans, and M. Yim. Telecubes: mechanical design of a module for self-reconfigurable robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002*, pages 4095–4101. IEEE, 2002.
- [176] SYMBRION. *SYMBRION: Symbiotic Evolutionary Robot Organisms, 7th Framework Programme Project No FP7-ICT-2007.8.2, grant agreement 216342*. European Communities, 2008-2012.
- [177] D. Tarapore, A. L. Christensen, P. U. Lima, and J. Carneiro. Abnormality detection in multiagent systems inspired by the adaptive immune system. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, AAMAS 2013*, pages 23–30. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

- [178] J. Timmis and A. Tyrrell. On homeostasis in collective robotic systems. In E. Hart, C. McEwan, J. Timmis, and A. Hone, editors, *Proceedings of the 9th International Conference on Artificial Immune Systems, ICARIS 2010*, volume 6209 of *LNCS*, pages 307–309. Springer, 2010. ISBN 978-3-642-14546-9.
- [179] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation*, 15(6):1035–1045, 1999. ISSN 1042-296X.
- [180] V. Trianni, E. Tuci, and M. Dorigo. Evolving functional self-assembling in a swarm of autonomous robots. In *From animals to animats 8: Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior*, pages 405–414. MIT Press, 2004.
- [181] V. Trianni, S. Nolfi, and M. Dorigo. Cooperative hole avoidance in a swarm-bot. *Robotics and Autonomous Systems*, 54(2):97–103, 2006.
- [182] C. Ünsal and P. K. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2000*, volume 2, pages 1742–1747. IEEE, 2000.
- [183] C. Ünsal, H. Kiliççöte, and P. K. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10: 23–40, 2001. ISSN 0929-5593.
- [184] A. van Rossum, S. McKibbin, A. Salden, and T. Schmidt. Emergent cognitive sensor fusion. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms*, volume 7 of *Cognitive Systems Monographs*, pages 183–203. Springer Berlin Heidelberg, 2010.
- [185] S. Vassilvitskii, M. Yim, and J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2002*, volume 1, pages 117–122, 2002. doi: 10.1109/ROBOT.2002.1013348.
- [186] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2: 189–208, 2008. ISSN 1935-3812.
- [187] W. E. Vesely and N. H. Roberts. *Fault Tree Handbook*. U.S. NRC, 1981.
- [188] K. Wada, T. Yoshikawa, T. Hayashi, and Y. Aizawa. Emergency response technical work at Fukushima Dai-ichi nuclear power plant: occupational health challenges posed by the nuclear disaster. *Occupational and environmental medicine*, 69(8):599–602, 2012.

- [189] J. Wang, G. Wu, Y. Sun, L. Wan, and D. Jiang. Fault diagnosis of underwater robots based on recurrent neural network. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics, ROBIO 2009*, pages 2497–2502, 2009. doi: 10.1109/ROBIO.2009.5420479.
- [190] B. Weel, E. Haasdijk, and A. Eiben. The emergence of multi-cellular robot organisms through on-line on-board evolution. In *Applications of Evolutionary Computation*, volume 7248 of *LNCS*, pages 124–134. Springer, 2012. ISBN 978-3-642-29177-7.
- [191] H. Wei, Y. Chen, J. Tan, and T. Wang. Sambot: A self-assembly modular robot system. *IEEE/ASME Transactions on Mechatronics*, 16(4):745–757, 2011.
- [192] H. Wei, Y. Cai, H. Li, D. Li, and T. Wang. Sambot: A self-assembly modular robot for swarm robot. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2010*, pages 66–71. IEEE, 2010.
- [193] A. F. T. Winfield and J. Nembrini. Safety in numbers: fault-tolerance in robot swarms. *International Journal of Modelling, Identification and Control*, 1(1): 30–37, January 2006.
- [194] A. F. T. Winfield, C. J. Harper, and J. Nembrini. Towards dependable swarms and a new discipline of swarm engineering. In E. Sahin and W. M. Spears, editors, *Swarm Robotics*, volume 3342 of *LNCS*, pages 126–142. Springer, 2005.
- [195] A. F. T. Winfield, W. Liu, J. Nembrini, and A. Martinoli. Modelling a wireless connected swarm of mobile robots. *Swarm Intelligence*, 2(2):241–266, 2008.
- [196] L. Winkler, A. Kettler, M. Szymanski, and H. Wörn. The robot formation language - a formal description of formations for collective robots. In *Proceedings of the IEEE Symposium on Swarm Intelligence, SIS 2011*, pages 1–8. IEEE, April 2011.
- [197] L. Winkler, V. Vonasek, H. Worn, and L. Preucil. Robot3D – a simulator for mobile modular self-reconfigurable robots. In *Proceedings of the IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI 2012*, pages 464–469. IEEE, 2012.
- [198] L. Winkler and H. Wörn. Symbricator3D a distributed simulation environment for modular robots. In M. Xie, Y. Xiong, C. Xiong, H. Liu, and Z. Hu, editors, *Intelligent Robotics and Applications*, volume 5928 of *LNCS*, pages 1266–1277. Springer, 2009.
- [199] M. Yim. A reconfigurable modular robot with many modes of locomotion. In *Proceedings of the International Conference on Advanced Mechatronics*, pages 283–288. Tokyo, Japan, 1993.

- [200] M. Yim. New locomotion gaits. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 1994*, pages 2508–2514. IEEE, 1994.
- [201] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University Mechanical Engineering Department, 1994.
- [202] M. Yim, D. G. Duff, and K. D. Roufas. PolyBot: a modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2000*, pages 514–520. IEEE, 2000. doi: 10.1109/ROBOT.2000.844106.
- [203] M. Yim, D. G. Duff, and K. D. Roufas. Walk on the wild side. *Robotics & Automation Magazine, IEEE*, 9(4):49–53, 2002.
- [204] M. Yim, Y. Zhang, and D. Duff. Modular robots. *Spectrum, IEEE*, 39(2):30–34, 2002.
- [205] M. Yim, Y. Zhang, K. Roufas, D. Duff, and C. Eldershaw. Connecting and disconnecting for chain self-reconfiguration with PolyBot. *IEEE/ASME Transactions on Mechatronics*, 7(4):442–451, 2002.
- [206] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, and S. Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2):225–237, 2003.
- [207] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics Automation Magazine, IEEE*, 14(1):43–52, March 2007. ISSN 1070-9932. doi: 10.1109/MRA.2007.339623.
- [208] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. J. Taylor. Towards robotic self-reassembly after explosion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007*, pages 2767–2772. IEEE, November 2007. doi: 10.1109/IROS.2007.4399594.
- [209] M. Yim, P. White, M. Park, and J. Sastra. Modular self-reconfigurable robots. In R. A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. Springer, 2009. ISBN 978-0-387-30440-3.
- [210] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji. Evolutionary synthesis of dynamic motion and reconfiguration process for a modular robot M-TRAN. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 2, pages 1004–1010. IEEE, 2003.
- [211] H. Zhang, J. Gonzalez-Gomez, Z. Me, S. Cheng, and J. Zhang. Development of a low-cost flexible modular robot GZ-I. In *Proceedings of the IEEE/ASME*

- International Conference on Advanced Intelligent Mechatronics, AIM 2008*, pages 223–228. IEEE, 2008.
- [212] Y. Zhang, M. Yim, C. Eldershaw, D. Duff, and K. Roufas. Scalable and reconfigurable configurations and locomotion gaits for chain-type modular reconfigurable robots. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 2, pages 893–899. IEEE, 2003.
- [213] Y. Zhang and J. Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252, 2008. ISSN 1367-5788. doi: DOI:10.1016/j.arcontrol.2008.03.008.
- [214] D. Zhuo-hua, C. Zi-xing, and Y. Jin-xia. Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey. In *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 3428–3433. IEEE, 2005. doi: 10.1109/ROBOT.2005.1570640.
- [215] V. Zykov, A. Chan, and H. Lipson. Molecubes: An open-source modular robotics kit. In *IROS 2007 Self-Reconfigurable Robotics Workshop*. IEEE, 2007.
- [216] V. Zykov, E. Mytilinaios, M. Desnoyer, and H. Lipson. Evolved and designed self-reproducing modular robotics. *IEEE Transactions on Robotics*, 23(2):308–319, april 2007. ISSN 1552-3098. doi: 10.1109/TRO.2007.894685.
- [217] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson. Robotics: Self-reproducing machines. *Nature*, 435(7039):163–164, 2005. ISSN 0028-0836.