The University Of Sheffield.

PIPELINE INSPECTION WITH AUTONOMOUS SWARM ROBOTICS.

RICHARD MOLYNEUX

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Philosophy

The University of Sheffield

Faculties of *Mechanical Engineering* and *Automatic Control and Systems Engineering*

Date: 14/01/2023.

Abstract

Underground water pipelines require frequent inspection to prevent decay and avoid costly repairs. The use of robots to inspect pipelines is well documented but the uniquely hostile environment of subterranean water networks means pipes require continual inspection. The idea of autonomous inspection robots that can provide continuous coverage has shown promise, but existing methods do not proactively aim to overcome a variety of diverse challenges. Specifically, extreme variability in the pipe conditions and the dense surrounding earth limit the communication capabilities of the robots, while dynamic water flows and power issues have a detrimental effect on their movement. This thesis presents the implementation of a range of path planning algorithms of varying levels of autonomy as governing swarm behaviours, each with a focus on overcoming some of the specific challenges inherent in underground water networks, with the goal of improving the efficiency with which the swarm can inspect a network. The Greedy Walk uses stochastic processes to plan a locally optimal path, the novel Ad Hoc algorithm aims to provide cyclic coverage, with robots moving as a fluid net throughout the network, and the k-Chinese Postman Problem solution explicitly plans optimal paths round subsections of a network. The thesis examines the performance of these behaviours against existing methods and anticipates obstacles in their real world implementation. The thesis then presents tailored versions of the path planning behaviours that include the introduction of more advanced methods focused on circumventing these issues. Specifically, the algorithms are developed to incorporate Gaussian Process Regression models to analyse strong water flows and use the data to plan intelligently, mitigating the detrimental effects of the flow. The flow analysis also provides a platform from which a novel Simultaneous Localisation and Mapping algorithm is presented, alongside a Multi-Objective Genetic Algorithm with the focus of increasing inspection frequency and conserving robot charge. The thesis shows evidence that an approach to pipeline inspection with autonomous swarm robotics based in path planning algorithms can help overcome the likely physical limitations of real world implementation.

Acknowledgments

First and foremost I wish to thank my supervisor, Kirill Horoshenkov, for his continued patience and mentorship throughout my academic journey. His knowledge and guidance have given me the freedom to pursue my research path, and have undoubtedly helped shape the direction of my thesis. Not only that, but his calm demeanour and professionalism have helped keep me sane and on track through peculiar and difficult times, and for that I am immensely grateful. Both in and out of academia, I wish him nothing but the best.

Likewise I give my thanks to my secondary supervisor, Sean Anderson, for his technical insight and willingness to help. There was never a call where I did not come away with a new idea or direction. I wish him luck in his future research.

A special thanks must go to Christopher Parrott for his tutelage and friendship. Outside of the sheer volume of time he put into my development, both as a coder and a researcher, I appreciate his help getting me through the long days, from sharing his projects to our collective ramblings. He has undoubtedly had the biggest impact on my research experience and I wish him the best of luck in his new career.

Thanks go to all my friends who have helped make my time in Sheffield special. In my eight years here I have been fortunate enough to meet some wonderful people who have enriched my experience in so many ways. To each of you I am grateful for the memories you have given me.

In addition, I wish to thank my lifelong friend Jack Evans. His steadfast friendship is rare and I appreciate every act of kindness he has shown to me and those around him. His consistency has kept me grounded, particularly during my research, and I will always welcome his advice.

To Antonia I give my greatest thanks. Her love and support have helped me endure my hardest times and I will be forever grateful to her for the patience she has shown through all my studies. I can only look forward to our future together.

Finally, I wish to thank my family for their love and unconditional support. It is impossible to express my gratitude for everything they have done for me — without them I simply would not be who I am today. To them I dedicate this work.

Contents

1. Introduction

Water networks are invaluable to humans. They are essential to sustaining modern life as we know it and feature in almost every aspect of our lives somewhere.

Despite their necessity, underground pipeline networks are increasingly in disrepair. Many of these pipes are old, built before sufficient documentation was proper practice. As such, not only are they in poor condition, but often their supposed location is not precise and they frequently overlap with other, newer pipes. The harsh environment in which they are placed accelerates decay, and their inaccessibility provides little opportunity for maintenance. It has become clear that current inspection methods are at best ineffective, so water companies are looking to transition to a more reliable, cost-effective solution.

Most water companies desire to increase their inspection frequency with minimal change to infrastructure, providing a new area to which autonomous sensing might be applicable. By introducing mobile inspection robots with the ability to navigate the networks, it is possible to deploy the robots from a central hub into the pipes, to inspect the surrounding area. This circumvents vast infrastructure changes and provides a non-destructive, proactive inspection process capable of finding leaks and blockages before they become problematic. Given the typical size of UK water networks, it is likely that multiple robots will be required to service a single area. With proper implementation, these *swarms* have the advantage of communicating and working together autonomously to increase the efficiency of the inspection process.

Autonomous swarms are extremely capable and easy to implement, and have subsequently branched out into a vast variety of fields. Unfortunately, swarm behaviours typically require frequent, if not constant, communication. Unlike most swarm settings, underground pipe networks are a highly challenging communication environment with extremely limited communication ranges. This is due to the extreme variability in the pipe conditions and the relatively high attenuation

of acoustic and radio waves with which robots would normally communicate. This impacts the swarm's ability to make quick, informed decisions as each agent is working with partial, or out of date information.

Additionally, water networks have the unique problem of dynamic water flows. A high flow has the potential to slow or halt a robot, which provides particular difficulties in ensuring a consistent inspection process. As well as this, the disruption from strong flows limit the frequency with which robots in the swarm pass one another, further compounding the communication issues. In addition, to cover the large size of the water networks with a continuous inspection process requires the robots to be untethered. Given that the robots are continuously inspecting, fighting flow and attempting to communicate to one another, their power demands vastly outweigh their capabilities.

There are many approaches that can be taken to address these issues, from actively governing the swarm to encouraging biologically inspired simple behaviours. Each has the potential to counteract the problems in different ways, with the ultimate goal of creating an efficient pipeline inspection process to sustain the water networks we rely on.

1.1. **Motivation.** Given its recent emergence as a field, swarm robotics is already relatively advanced due to its vast array of capabilities with simple application. Typically, robots require little information to influence the swarm and, similar to many natural systems, the group is able to make complex decisions based on a multitude of smaller bits of information collected by the swarm. The ability to formulate efficient solutions with minimal computational or sensing requirements proves advantageous in many fields, from nanotechnology to search and rescue robots. Unfortunately, underground pipeline networks are a uniquely challenging environment with various stipulations that limit the potential of typical swarm behaviours, which can be detrimental to the efficiency of autonomous swarm inspection. The simple nature of swarm algorithms means they struggle to make use of other potentially beneficial, intelligent solutions that can be created from the data collection. Additionally, pipeline environments do not favour swarm systems

as the efficiency of swarm behaviour is highly dependent on frequent, instantaneous communication. As such, the opportunity presents itself to examine the rigour of other existing methods to determine both their efficiency against swarm behaviours, and whether they can be tailored to include more advanced techniques. Path planning algorithms of varying degrees of autonomy are commonplace and are used in many fields, from navigation to video games. They require more direction to function efficiently than swarm algorithms however they are able to take advantage of much more information than swarm algorithms to create solutions to more complex problems. It is hoped that by introducing path planning algorithms, methods specifically focused on mitigating the effects of the environment can be implemented to help improve the efficiency of the autonomous swarm inspection process.

1.2. **Problem Definition.** Inspection robots have gained traction as viable solutions to pipe maintenance in recent times. Though the field is new, there is a multitude of evidence that shows the potential of robots in a pipeline environment. However, the step up to an autonomous platform is enormous. Immediate considerations surrounding the physical limitations of an autonomous robot range from the power requirements to the communication methods and processing power of a necessarily small robot. To this end, what is needed is an exploration of potential systems that are capable of providing an efficient inspection process, whilst minimising the physical demands of the robot. A well-documented field surrounding network navigation is rooted in graph theory, specifically path planning algorithms. Path planning algorithms have the ability to create intelligent routes for a robot and can have varying degrees of autonomy. Their capacity to consider information and plan ahead allows them to determine optimal paths with multiple objectives. As the field progresses, and the anticipated physical obstacles of autonomous inspection become more apparent, an assortment of behaviours that can circumvent these issues could aid in reducing certain hardware requirements. To be sure of an optimal and realistic inspection process, an exploration into path planning algorithms and their potential in subterranean water networks must be considered.

1.3. **Aims and Objectives.** The overall objective of this research project is to develop a swarm behaviour based in path planning capable of handling the unique technical challenges posed to autonomous pipeline inspection robots. The work of Parrott, C. et al. [44] lays the foundation of a simulation platform and the stigmergy behaviour [16], against which the efficiency of path planning algorithms can be benchmarked. To counteract the evident detrimental effect of dynamic water flows, the work aims to integrate Learning Algorithms, with which the agents can model the water flow, with path planning algorithms to navigate the network efficiently. This work is to build upon the existing simulation to introduce other mechanics that represent issues in implementing autonomous swarms that have not been considered previously, and build in adapted algorithms to achieve optimal solutions in a more complex simulation. Contributions to the simulation should consider realistic situations such as strong flows overwhelming robots, pushing them back and the robots becoming 'lost', or the effects of adding power limitations to the robots with limited charging stations. This provides the opportunity for the algorithms to be developed further, in response. For example, the work aims to utilise information gained from the flow analysis to improve a novel SLAM implementation for lost agents. As these considerations increase, the aim is to similarly adapt any algorithms or behaviours with additional solutions to mitigate the effects of these changes and keep consistently low Time Between Inspections. It is hoped that a variety of path planning algorithms will provide a strong basis to be adapted in response to each new stipulation. At the conclusion of the thesis, the resulting algorithms should incorporate many different solutions to exhibit an adaptive approach to autonomous pipeline inspection robots that sustains low Time Between Inspection values and high network coverage, regardless of the conditions imposed.

1.4. **Publication.** This thesis represents the authors own work and has led to the following peer-reviewed publication:

Molyneux, R. , Parrott, C. , Horoshenkov, K. (2019), 'An Application of Path Planning Algorithms for Autonomous Inspection of Buried

Pipes with Swarm Robots', World Academy of Science, Engineering and Technology, Open Science Index 153, International Journal of Mechanical and Materials Engineering, 13(9), 555 - 563.

1.5. **Thesis Outline.** The thesis is structured as follows:

- Chapter 2 explores the literature surrounding autonomous robotic inspection and potential fields of application. Particularly, it explores biologically inspired swarm behaviours and path planning algorithms and how they can be applied to pipeline inspection. First and foremost, existing inspection methods and robotic capabilities are explored to examine the areas of challenge in implementing autonomous swarms. The literature examines the strengths of current acoustic and ultrasonic communication techniques and investigates fields in artificial intelligence and machine learning. An emphasis is placed on potential applications in path planning before concluding with analysis of genetic algorithms and their applications.
- Chapter 3 begins by providing an in-depth detailing of the simulation platform presented by Parrot, C. et al. [44]. This is important as the work in this thesis is built off adaptations and additional work that is built on the simulation platform. The detail informs the reader of the workings of the original simulation, and enables more coherent explanations of the changes as they are added. The Chapter goes on to introduce three different path planning algorithms that govern the decision making process of the agents in a network, before demonstrating the simulation output to analyse their inspection efficiency against the existing stigmergy behaviour. The Chapter concludes by adding acoustic and ultrasonic communication methods to the simulation before comparing the effects on the behaviours.
- Chapter 4 uses the simulations existing ability to model dynamic flow patterns to analyse the effects of strong flows on the path planning algorithms. It goes on to add a mechanic to the simulation — the ability for an agent to become overwhelmed by the flow and become lost. A simple but effective SLAM

algorithm capable of localising quickly with limited topological information is then presented and the adapted simulation is used to evidence it's efficiency. The Chapter then presents an approach for stochastic modelling of the flow patterns with the intent to learn the cyclic flow of a network, to aid in intelligent traversal. To this end, Gaussian Regression models and the speed at which they can learn the network flow are illustrated. The Chapter finishes by examining the effects of path planning with knowledge of the flow and concludes with the idea for a novel SLAM algorithm, capable of utilising the flow data to improve the rate at which it is able to localise.

- Chapter 5 details additional changes to the simulation platform surrounding charging and power. Specifically, agents are given a battery percentage and two methods of charging, as well as adding conditions on a powerless robot. The Chapter then presents a Multi-Objective Genetic Algorithm with the goal of finding Pareto optimal solutions that balance the power needed to inspect a path, against the reduction in Time Between Inspections.
- The final Chapter 6 discusses the works and contributions of the thesis and concludes with a specific vision for future work that extends from lessons learned during the research project.

2. Literature Review.

This Chapter will provide an analysis of existing literature relevant to autonomous swarm inspection. The chapter begins with a brief history of existing inspection methods and recent developments in inspection robotics before presenting novel work surrounding biological swarm behaviours and autonomy. This is followed up by a literature review of path-planning algorithms and their potential applications in subterranean networks. In anticipation of the methods this thesis presents, the review then extends to other areas of relevance, such as Simultaneous Localisation and Mapping algorithms, Deep Learning algorithms and Multi Objective Algorithms to provide context as to their uses and appropriateness to work in tandem with path-planning algorithms.

2.1. **Existing Inspection Methods.** Underground water pipeline networks are uniquely difficult to inspect frequently — the subterranean nature of the pipes alongside the size of the networks makes proactive manual inspection near impossible. This is a problem as water pipeline networks are increasingly in disrepair, and water companies have limited knowledge of the conditions of their pipes. As such, faults in the system are only exposed *after* they have caused a leak, leading to consumer disruption and costly repairs — an estimated 3,031 million litres of water are leaked each day [15].

The most manual inspection process use human inspection to either enter a pipe or surveying the conditions surrounding a pipe. For severely damaged pipes, the deterioration can be indicated by examining the properties of the surrounding soil [35]. The pH value of soil above a pipeline can provide an insight into potential corrosion alongside other indicators such as the moisture levels in the soil. Changes in the electrolyte levels in the soil surrounding a pipe can be surveyed to ascertain whether gaps exist in the coating of ferrous pipework. Though non-invasive, this is a time consuming approach to examing the surface conditions of pipework. In a similar manner, sediment sample analysis can be implemented to assess their chemical constituents and risk to water quality [26].

In spite of this, existing methods do exist that provide a measure of assurance to water companies, though they are limited. Commercial inspections using remote CCTV cameras that inspect the internal bore are commonplace. The system comprises of a CCTV camera and lighting apparatus that is moved throughout the network [33], and can be fitted with multiple cameras to achieve a 360 degree view of the pipe [24]. The method of implementation is either to fit the CCTV underground or insert it at the end of a rod, similar to an umbilical cord system, to explore the surrounding area [25]. The rod insertion also provides additional inspection opportunities by attaching hydrophones in tandem with the cameras for accurate in-pipe leak detection. Unfortunately both elements become expensive when considering the size of the networks typically seen in the UK. A fitted CCTV unit requires invasive drilling and given the topography of pipeline networks, with

extremely long pipes and bends, is inefficient in a fixed position to examine an entire network. Similarly, the range on the rod insertion is around 200 meters. An average size water network is explored as part of the research in this thesis, the size of which is detailed in the Looping network in Table 1. The cumulative length of this network is nearly 30,000 metres, with limited natural entry points. When considering the 348,723km of water pipework in the UK [15] it is extremely difficult to consider the rod able to sufficiently inspect a network without a high quantity of manual labour and the use of invasive drilling. The requirement to be close to an entry point is a common obstacle in ensuring coverage of a network and is a contributing factor to the push for autonomous inspection.

Another well documented inspection method surrounds the use of acoustic detection in water. The wide applicability of acoustic detection has allowed it to develop into many commercial products, with one that leans heavily on the use of acoustic sensors being the LeakFinder system [21], focused on pinpointing leaks in a length of pipe. The system is comprised of transmitters and sensors that can sense leak-induced vibrations and sound. A signal is transmitted through the pipe to two receivers at different points of the pipe. The technology couples both primary acoustic sensors, accelerometers and hydrophones, to calculate the time lag between the two sensors [33] and derive an accurate location of the leak. Though the inspection returns accurate results, the system still requires an operator and is a reactive response to being aware of the existence of a leak.

Another commercially viable acoustic technology is that of the SmartBall [22]. The SmartBall is a free-swimming device acoustic device designed for leak detection. What differentiates SmartBall is that it passes through a water pipeline network with the flow of the network. As it traverses the network, SmartBall makes use of multiple acoustics sensors alongside ultrasonic transmitters and temperature sensors to continuously record data and the position of the ball within a network. By emitting acoustic pulses and examining the feedback, the sensors are able to determine approximate locations of air pockets and leaks in a non-invasive manner. However, as the SmartBall is propelled along

by the flow of the network, the user has very little control over the direction or speed the ball takes. As such, frequency analysis is often required to properly determine the nature of an anomaly. Additionally, the lack of directional control means that to gain a proper coverage of the network, the ball must be placed at different entry points so as to avoid simply repeating the same few paths.

2.2. **Robotic Inspection Methods.** Given the limited ranges that can be achieved via current manual inspection processes, it is unsurprising that vast leaps have been made in the areas of robotic pipeline inspection. Robotic inspection has particular application in Non-Destructive Testing and can be implemented to reduced work time, reach areas of difficulty, or provide a safer inspection process. The field is well documented, particularly in gas pipeline networks. Typically, the inspection revolves around inserting a tethered robot into a pipe, equipped with visual or acoustic sensors, and traversing each direction from the entry point whilst continuously scanning [5]. The tethered nature of the robots removes any requirement for power considerations, and provides a fast, reliable feedback of inspection data to the receiver. However, the range that can be inspected is of course limited to the length of the tether.

Perhaps the most advanced field when considering robotic pipeline inspection is centred around gas pipelines. Gas and oil leaks pose a much more serious threat than water leaks, and as such extensive research has been invested to ensure a thorough inspection process. Early robotic systems have been presented that use articulated structures and tether cables to develop snakelike robots capable of navigating complex pipeline configurations [48]. Pipeline inspection gauges or gadgets, commonly referred to as pigs, can be propelled by a pressure-driven flow at speeds to gather fast inspection data [42] [13]. Unfortunately this is not a complete method for all pipelines as obstacles and physical barriers can disrupt this inspection method. As such, advances have been made in *crawler* technology to create inspection robots of higher maneuverability. Crawlers are typically mounted on wheels or tracks and enable the user to assess a wide range of pipeline conditions due to

their independent movement and sensor functionality. They are typically equipped with cameras and lights, alongside acoustic or ultrasonic sensors for leak detection [38]. By moving independently to the flow they are able to move slower and steadier to generate a more accurate depiction of the pipeline. Additionally, they can be built smaller than the pigs, and as such provide an option to inspect smaller pipes that were previously inaccessible.

There are many different designs for robotic inspection, typically in response to the pipes in question [27]. Because of the strength of certain dynamic flows in water networks, water inspection robots tend to follow a locomotion design that anchors them to the wall [41]. Usually this is done by compressing the robots upon entry to the network, then applying outward force until the wheels/tracts/movement mechanism is snug to the pipe wall [47]. Therefore, one of the main considerations in the design stage is the pipe diameter, as this provides the biggest obstacle with respect to the physical size the robots can be, and therefore their capabilities. Additionally, topological constraints such as sharp corners or obstacles in the pipe are necessary to circumvent to gain full use of the tethered range. As such, a favourable design in water pipelines is that similar to that of a caterpillar. Elongating the robot enables the system to fit inside pipes of small diameters without compromising on functionality, spreading out necessary components such as motors and sensors. The conditions of many pipelines mean pipes frequently have internal deposits lining the pipe wall that can inhibit movement methods of other robots — those relying on magnetic attraction to a ferrous surface to grip the pipe walls can falter when brought into contact with high quantities of sludge. The See Snake [45] is a commercial adaptation for water pipelines focused on non-contact Non-Destructive Testing. The See Snake aims to circumvent the effects of the pipe wall lining by attaching wheels to multiple modular bodies at different degrees and linking the bodies together in the caterpillar design. Additionally, the modular split provides the robot with enhanced movement capabilities, and enables it to traverse sharp bends. This is particularly useful in water networks where pipes can meet at sharp junctions and unusual angles. Despite it's maneuverability, the

See Snake is again a tethered robot that requires manual instruction to navigate a network.

Advances in robotic pipeline inspection have demonstrated that through various innovative designs, robots are capable of maneuvering across the network and around sharp bends. Additionally, they are loaded with sensors that already collate valuable inspection data. This helps support the proposal that robotic swarms can inspect autonomously. However, the field has struggled to break away from the anchor of tethered systems, with no system capable of continuously inspecting on it's own. Untethered robots lack the high charge capacity needed to sustain inspection for long, and there is no mention of subterranean charging capabilities.

2.3. **Autonomous Inspection Methods.** It has been established that the concept of utilising robots to inspect pipelines is well documented. What differentiates the field of autonomous inspection is the goal of creating a system that does not require an operator. Implementing autonomous inspection circumvents many of the problems that exist with pipeline inspection — continuous inspection can occur without the need for invasive or destructive manual inspection techniques. An autonomous robot that is continually inspecting would be able to collate larger amounts of data, generating a much more accurate depiction of pipeline networks with a much greater coverage. By extension, introducing multiple robots into a network would inevitably increases the frequency with which pipes can be inspected as this increases the cumulative distance of pipe inspected by the swarm.

The literature in Section 2.2 evidences that aside from the charging and power considerations, appropriate robotic platforms exist that can navigate and inspect a network. A milestone in evidencing the feasibility of *autonomous* inspection is the work of Parrot, C., Dodd, T., Boxall, J.B. and Horoshenkov, K. titled 'Simulation of the Behaviour of Biologically-Inspired Swarm Robots for the Autonomous Inspection of Buried Pipes' [44]. The work details a simulation platform that explores how a simple swarm intelligence can be applied to autonomous

robots to create a powerful inspection process based off of indirect coordination. Specifically, a simulation environment was designed to model the structure of real life water networks and place robots, or *agents*, within the simulation capable of traversing the network. By assuming continual inspection as the agents moved throughout the network, the simulation was capable of determining an average Time Between Inspections or *TBI* for each pipe and was able to evidence improved efficiency to existing methods. In addition to this, the simulation makes use of EPANET software files to construct the corresponding water flows within the network across a 24 hour period. By assuming that strong water flows might inhibit the movement speed of robots in a pipeline, the paper was able to illustrate the detrimental effects of existing flow cycles to both the coverage and frequency of inspections.

The work extends further, seeking to prove not only the effectiveness of autonomous inspection, but to provide a governing intelligence for the swarm of robots to improve the coordination of the swarm. To govern the movement of the swarm, the work by Parrott, C. et al. [44] digitised the Stigmergy [16] swarm behaviour to implement an *implicit* form of cooperation between the agents, based solely off choosing a pipe when the agent reaches a junction. Provided there was communication between the agents, the agents were able to share their 'memories' of the network — their individual inspection histories. In doing so, two agents sharing their inspection history were able to update their own awareness of pipes that had been inspected most recently to inform decisions. Each pipe was therefore anointed with a *virtual pheromone* that enabled agents at a junction to choose the pipe which most required inspection. The paper goes on to illustrate the benefits of increased communication ranges and coordination to the efficiency of the Stigmergy implementation, particularly with respect to network coverage.

The work of Parrott, C et al. [44] provides a fantastic foundation for future research into autonomous inspection. The work proves the feasibility of autonomous pipeline inspection using swarms by evidencing a behaviour capable of adapting to the expected physical obstacles the might surround the implementation of real robots into a network.

Building on this, it is possible to consider other aspects relevant to autonomous inspection such as intelligent solutions that seek to proactively mitigate the negative affects illustrated in the work. The simulation results have shown both strong water flows and limited communication capabilities to be complications unique to underground water pipeline inspection and as such, potential areas to be navigated to improve the efficiency of the swarm. Additionally, it must be noted that continuous inspection was assumed and the real life power stipulations inherent in robotics were not modelled.

Other notable contributions to autonomous inspection includes the work of Li, X., Yu, W., Lin, X. and Iyengar, S.S. titled 'On Optimizing Autonomous Pipeline Inspection' [32]. The work proposes a solution to *gallery guarding*, a problem aiming to find the smallest set of points inside a region from which all the boundary points are visible. In the application of pipeline networks, the work assumes the robots are fitted with a camera and lighting and can inspect everything they see. The work implements a hierarchical integer linear programming (HILP) algorithm to find the optimal formulation of points on a network from which coverage can be guaranteed. The work details a 3D model evidencing leaks and blockages found during the simulation, and provides a method with which an individual robot could ensure traversal of the network and complete inspection — by simply moving from point to point.

The application of the work of Li, X. et al. [32] into water networks requires some consideration. The system is reliant on the initial geometry of the network to be modelled before any computation can occur. Water companies in the UK do not necessarily have the information required to build the 3D model — pipes have frequently been built on top of other pipes and before proper documentation was implemented. Assuming the information is available, the work details how the pipeline wall needs to be marked in order for the robot to localise at the exact location needed to ensure complete coverage. Applying the markings to water networks is a task in itself; given the size of water networks in the UK and the relative range a robot can assume to be able to inspect from one spot, it is likely that an extremely high number of markings

would be required to run efficiently and is likely to be a manual task in itself. Finally, water networks have the unique issue of dynamic flow rates which are capable of blocking off large subsections of network at a time. This would make it difficult to consistently reach the spots required for optimal inspection without coordination.

Continuous autonomous inspection of water networks is undoubtedly desirable but, due to hardware capabilities and power requirements, it seems a long way off. However, if the ability to continuously inspect can be assumed, it is now evident that robots could be applied for this purpose. To that end, as with the work of Parrott, C. et al. [44], through the medium of simulations it is possible to begin analysing efficient ways to get the robots to work together and build and optimal inspection process.

2.4. **An Introduction to Swarm Intelligence.** Swarm intelligence can be considered as the emulation of some biological system to achieve a collective goal. Bonabeau [4] offers a specific definition that swarm intelligence is 'any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of social insect colonies and other animal societies'. The field is technically a subset of Artificial Intelligence (AI) algorithms and has gained increasing popularity in the last two decades due to it's wide variety of applications. Swarm algorithms typically use small amounts of data to inform a rudimentary choice that, in tandem with the rest of the swarm, often contribute to offering a versatile solution. An example of their strength lies in NP-hard problems — a classification of polynomial-time problems that are not solvable by a non-deterministic Turing machine in polynomial time. They are particular difficult to solve, and as the problem grows, so too does the solution — exponentially. As such it is frequently difficult to solve them exactly in real time. Swarm intelligence algorithms have been applied to find solutions sufficiently close to the global optima in a much more viable time frame [9]. This helps illustrate how powerful proper application of swarm intelligence can be.

Perhaps the best example of a simple swarm intelligence is *stigmergy*, a form of indirect communication used by ant colonies to leave relevant

information for other members of the colony in their environment [16]. The concept was first introduced in 1959 by Pierre Paul Grasse [55] and linked together the behaviour of individual ants with that of the swarm. In essence the worker ants would leave pheromone signatures on their environment as they continue their work. For instance, when traversing a food trail and discovering all the food has been harvested and there is none left, a single ant would leave a pheromone signature to let other workers know. As more and more ants begin to leave similar signatures, the group eventually reaches a *quorum*, a state in which they have sufficient information to agree that trail is no longer a viable food root. This was shown to be used to share indicators of danger or new resources. The evidence of what could be achieved with such simple information and implementation is the basis for *Ant Colony Optimization* algorithms.

It is unsurprising then that swarm intelligence has been applied in a variety of fields with notable success. Biologically inspired algorithms are not limited to insects, and range from glow worm to bat and lion based algorithms [9]. They have huge applications and can simplify complex problems into manageable operations. Of note are Particle Swarm Optimization algorithms, built to mimic fish schooling and bird flocking [11]. PSOs communicate not only their position but also their velocity, enabling the members to manage their own positions and move together intelligently. This has lead to applications in UAV swarms in search and rescue operations [10], tremor analysis for the diagnosis of Parkinson's disease [19] and auto-tuning for telecommunication systems [18].

The application of swarm intelligence in subterranean networks is limited. Despite their affluence above ground, swarm behaviours are highly reliant on being able to communicate with one another, directly or indirectly. As soon as a member becomes isolated, their efficiency and contributions to the swarm objective will drop. Given the hostile communicative environment of underground networks, it is therefore unsurprising that the implementation of swarm behaviours in pipeline inspection is difficult. Despite this, the work of Parrott, C. et al. [44] mentioned in Section 2.3 digitised the stigmergy swarm behaviour to

assign *virtual pheromones* to pipes in a network. This helped formulate the governing behaviour of the swarm, informing decisions implicitly to enforce an impressive indirect coordination method. The work did explore the effect of different communication ranges and was more efficient when the agents were able to communicate more frequently.

Mandal, S. K., Chan, F. T. and Tiwari, M. K. [34] also found a way to implement swarm intelligence in pipeline inspection by combining an Artificial Bee Colony (ABC) algorithm with a Support Vector Machine (SVM) algorithm to develop a pipeline leak detection system. ABC algorithms are typically concerned with optimisation for problems that are deterministic or stochastic in nature, and are inspired by the perceived foraging behaviour of honey bees in their search for nectar [29]. SVMs are supervised learning algorithms centred on regression analysis. The work uses the split of Employed Bees focused on investigating food sources and updating their memory to complete their search and share this information in the *dance area*. Subsequently, Onlooker Bees decipher information from the dance area to choose a food source based on the best estimated returns. In this instance, a Genetic Algorithm is applied to select 'food sources' for the Onlookers. Finally, the Scouts are Employed Bees who have exhausted their food sources to randomly search out another food source. Combining the two approaches results in a behaviour where, during the search, employed bees and onlookers modify their positions while the scouts exchange abandoned solutions with more fruitful ones. The paper goes on to show the benefits of using this search process in tandem with an SVM and the SVM's improved ability to obtain high detection rates of leaks.

Swarm intelligence is clearly a powerful and well established field. However, the underlying commonality between all swarm behaviour is a fast rate of communication — it is essential to ensuring intelligent contribution to a common goal. Without proactively seeking to increase the rate of communication exchange, extrapolating the work to a subterranean environment is unlikely to yield similarly brilliant results as it does above ground.

2.5. **An Introduction to Path Planning.** Path planning is a crucial component to many robotic and automated systems. The field is based in geometry and is an extension of *graph theory* in mathematics, with the objective of calculating the best way to traverse a graph with one or more secondary objectives. Gasparetto, A., et al. gives a straightforward definition: 'find a collision-free motion between an initial (start) and a final configuration (goal) within a specified environment' [23]. Contrary to the indirect approach of swarm intelligence, path planning algorithms aggressively seek optimal solutions but are typically reliant on higher computer processing power to use data to inform their decisions. Literature in the field is well documented as the goals of the algorithm can vary drastically, from finding the minimum energy route to an end point, to planning a path that visits allows traversal of an edge only once. The latter is the famous Konigsberg bridge problem which questions whether it is possible to walk over the seven bridges of Konigsberg only once each and return to the same spot [58].

Dijkstra's algorithm is a complete method for finding the shortest path between two vertices on a graph. A graph is simply a mathematical representation of relationships and is built with vertices and edges [6]. Modelling water networks in this manner allows for the introduction of other path planning algorithms. It works by dividing all nodes (vertices) into unvisited and visited sets and iteratively refining the sets, moving nodes to the visited set once the shortest path to the visited node has been found. Eventually, the destination vertex is visited and the algorithm is complete. The algorithm is widely accepted as the default, optimal solution for shortest path search problems which has lead to implementations in navigation systems such as Google Maps [31], intelligent fire evacuation systems [61] and soft computing [14]. However, the method is computationally expensive and becomes unsuitable as the size of the graph increases [39]. This lead to the development of the A* algorithm, a heuristic approach to shortest path problems [54].

Another well documented example of a path planning algorithm is the Chinese Postman Problem [20]. The Chinese Postman Problem is a heavily studied topic within graph theory, and hence has a huge back

catalogue of relevant literature of adaptations, typically involving min-max approximations [1]. The Chinese Postman Problem itself focuses on finding the optimal route around every edge in a graph such that every edge is passed at least once [43]. The problem originally revolved around a single postman servicing a network of streets delivering post whilst trying to minimise the total time it took to complete a route, always returning to the depot node. This problem is highly relevant to the problem of pipe inspection with an autonomous robot swarm as the swarm network can be modelled as a graph, with junctions as vertices and the pipe links as edges [56]. A CPP solution would provide an optimal traversal route for the agents in the network. The CPP has a counterpart routing problem, the Traveling Salesman Problem [17]. The TSP attempts to traverse a graph while touching every vertex at least once instead. Both of these algorithms constitute path-planning algorithms as they create a connected path between two points with some optimization function.

The Vehicle Routing Problem (VRP) is an advanced research topic within path planning and has an overlap with swarm intelligence [2]. It is an optimisation problem concerned with minimising or maximising some goal but for *multiple* vehicles. The earliest example is the Truck Dispatching Problem [12], which required that a fleet of oil trucks leaving a central hub traverse the minimal amount of distance as a fleet, whilst still fulfilling the demand for oil from customers diffused around the network. This has been increasingly adapted to incorporate more worldly considerations such as time windows for delivery slots and dynamic information [28]. Research is also driven forward in a solo capacity for vehicles — Global Positioning Systems (GPS) are inherent in most vehicles and smartphones nowadays. To ensure the fastest route, GPS routes take much more into account than the topography of UK road networks. Traffic jams, roadworks and even user selected options such as avoiding motorways are all necessary considerations to advise on the best route. Maaref, M. and Kassas, Z. M. introduce statistical bounds that guarantee the probability of the absolute position error exceeding a set threshold cannot be larger than the integrity risk [36]. The work shows that the additional considerations created a route outside of the shortest path that was able to outperform it.

The k-CPP is an extension of the Chinese Postman Problem, introducing multiple postmen to service an area [43]. As opposed to swarm behaviours, ideal solutions in this instance revolve around the postmen actively planning their paths together. The most time consuming method and inefficient method would have k-1 postmen simply remain at the depot node, whilst the remaining postman serviced an area on their own. On the other hand, an optimal algorithm would see the workload distributed relatively evenly with the overall route time decreasing linearly as k, the number of working postmen, increases. The process of finding CPP solutions revolves around ensuring that an the graph is Eulerian — that is to say, no vertices in the network have an odd degree. An elegant, albeit inefficient method for then extracting a route around the graph is Fleury's algorithm. Fleury's algorithm works by iteratively removing edges from the network who's deletion does not disconnect the graph [49]. The process repeats from connecting vertices until the last edge has been removed, and the Eulerian cycle is the order in which the edges were removed.

It is hoped that the literature presented can be adapted to offer an alternative consideration for the governing intelligence of the swarms in underground pipeline inspection. The network components in roads which allow for the application of path planning algorithms correlate with those in water networks — roads and junctions serve as appropriate reflections of pipes and their intersections. The necessary step is to present the network as a *graph*. Upon doing so, many applications of path planning can be implemented, such as shortest path algorithms or multiple agent planning algorithms such as the k-CPP and VRP solutions.

2.6. **Simultaneous Localisation and Mapping Algorithms.** Simultaneous Localisation and Mapping algorithms, or SLAM algorithms, are typically used in unknown environments to localise themselves based on features in their surroundings, often landmark features [3]. The field is broad with many applications from large scale mapping of urban structures [57] to autonomous mine mapping [40]. Solutions to SLAM problems must be considered when attempting to make truly

autonomous robots. With respect to pipeline networks, it could be desirable for robots to be able to map out the network. As aforementioned, the information surrounding some networks is unreliable. Previous methods of mapping out networks have included Branch and Bound algorithms [7]. However, these techniques use brute force and are exhaustive. They are therefore undesirable when considering the size of UK water networks. The potential for robots to map their own environment before traversing them provides a newer definition of autonomy.

An example of gas pipeline is presented by Zhang, S. and Dubljevic, S. [64]. The work details how a pipeline crawler can navigate small-diameter gas pipelines whilst simultaneously mapping out the topology of the network. The system uses an inertial measurement unit and odometers to generate kinematic and measurement models that can accurately relay pipeline features and a complete topology.

Once a system has been mapped, SLAM algorithms are typically employed into robotic systems, which provides them with the ability to quickly localise themselves. Worley, R., Yu, Y. and Anderson, S. describe a method of acoustic echo-localisation for pipe inspection robots [60]. Underground pipeline networks are extremely dark, and as such other visual SLAM (vSLAM) methods lose their appeal [53]. Similarly, pipelines are often devoid of landmark features and as such become a difficult environment with which to truly be certain that a robot has localised. The echo system presented detects distant features and uses acoustic processing to make measurements of the robots position. The acoustic data is refined before a state estimation is declared, leaving the method with a high success rate.

With strong flows in the network it is necessary to consider the possibility of robots getting pushed around. If a robot is pushed too hard it might lose its bearings. In this instance, a Simultaneous Localisation And Mapping algorithm can assist in localising the robot again. Relying on acoustic echo-localisation compliments the acoustic communication and inspection methods as the robots will be necessarily

small, so any equipment that can be shared will significantly lighten the weight of the robot.

2.7. **Applications of Deep Learning Algorithms.** Machine learning is a subset of artificial intelligence focused on obtaining specific information without being explicitly programmed to. Supervised learning requires human interaction for input data, and requires feedback on the accuracy after the output [50]. On the contrast, unsupervised learning algorithms require no input from the user. Though supervised learning algorithms tend to be simpler than their counterparts, they are more powerful in their ability to output data based on previous experiences. However, the strength of a supervised model is entirely reliant on the efficiency of the training data set. Incoherent data leads to incoherent learning.

Deep Learning Algorithms are a subset of machine learning algorithms that use multiple layers of neural networks to build computational models [63]. They are able to train deep neural networks and can be used in diverse tasks such as image classification, object detection and image retrieval [62]. Often it is the case that they have several algorithms capable of handling specific tasks better than others. The deep learning refers to the use of layers in the network, and learning can be split into supervised, semi-supervised or unsupervised.

Gaussian Process Regression models are a subset of deep learning artificial intelligence that are simple to implement, flexible, fully probabilistic models [8]. They use a non-parametric, Bayesian approach to regression that allows for powerful prediction analysis on datasets [51]. This means that instead of calculating a probability distribution using data, the PDF is calculated using functions that fit the data. Gaussian Processes are particularly adept at solving unknown function curves that map inputs to outputs [52]. In contrast to supervised learning methods, this allows the relationship between an input and output to be solved as a function curve using very little data. As the data set matures, the Gaussian Process refines the curve to better represent the new distribution function. This is an extremely powerful method for early analysis when little is known but the data set. This could

have applications in pipeline swarms, building up a database of flow readings to fit a curve that accurately depicts flow cycles which could inform decision making surrounding path planning.

3. Simulation Platform and Communication Methods.

3.1. **Introduction.** In order to provide a competitive inspection process, it is desirable that the autonomous robots are able to work effectively together as a swarm. Given pipe diameters can be as little as 30cm tall, the robots are necessarily small and are therefore unlikely to be computationally powerful. As such, the robots will likely lack the means to think intelligently individually so must be able to make complex decisions from limited information in order to achieve optimal results independent of manual instruction. To make informed decisions the robots must overcome an oppressive subterranean environment which limits their communication and movement.

Underground networks provide some unique difficulties for autonomous robots. Strong water flows inherent in pressurised water networks can slow, or even halt a robot. Network size demands the robots be untethered which introduces power requirements that currently surpass their capacity. Leaks and blockages can obstruct access to whole segments of network, rendering them incapable of inspection. Autonomous inspection robots must take these considerations into account, which forces both swarm and path planning algorithms to become more reliant on a consistent feed of up-to-date information. Above ground these issues can be mitigated but in an underground network they are compounded by strenuous communication capabilities.

A strong communication method helps swarms share information about their environment. Typically this is a simple process that can be done with acoustic exchanges or pheromones, both of which can convey a simple message quickly. Unfortunately, underground pipeline networks provide a particular challenge in this regard. Because of the density of the surrounding earth and the variability of the pipe conditions, typical communication methods are rendered ineffective. Without the range customarily afforded to swarms, the robots in the system are unable to make use of what little information they can gather as they inspect a network. By demonstrating an effective communication process, improvements in the robots decision making processes and ability to work together are made evident.

This chapter details the custom simulation platform presented by Parrott, C. et al. [44] for autonomous inspection of pipeline networks. Three path planning algorithms are presented to provide a basis to explore efficiency solutions, and the effects of communication ranges are explored in a manner consistent with the existing stigmergy results. Finally, realistic ultrasonic and acoustic communication types and ranges are added to the simulation, to get a realistic representation of the frequency of communication in larger networks.

3.2. **Simulation.** The nature of pipeline inspection creates difficulties in exploring proposed solutions — the scale of the networks demands high investment for calculable results and testing must be non-invasive. Similarly, implementing robots in water pipelines requires hardware development and potential infrastructure upgrades. Given the abstract nature of the problem, simulation platforms are the only medium capable of demonstrating the feasibility of autonomous inspection. Modelling the robots in a network as point agents removes the necessity for immediate hardware expansion while an accurate network reconstruction allows for the exploration of solutions.

The simulation detailed here, in Section 3.2, continuing until Section 3.3, is an in depth description of the simulation described in the work of Parrott, C., Dodd, T., Boxall, J.B. and Horoshenkov, K. (2020) 'Simulation of the Behaviour of Biologically-Inspired Swarm Robots for the Autonomous Inspection of Buried Pipes' [44]. The work is detailed here to inform the reader of the technicalities of the simulation as the work is adapted and expanded throughout the remainder of the thesis (Section 3.5 onwards). Additionally, it should support the reader in understanding context surrounding the structure of the code, which allows for separate behaviours to be kept in separate classes, how the Time Between Inspection metric is generated and how movement, flow and communication are modelled.

The presented simulation was built in *C++*, an extension of the *C* programming language, and has been compiled using Microsoft Visual Studio 2017. The advantage of C++ is that it supports object-oriented programming, a programming paradigm that allows the manipulation

of objects and classes. Not only does this enable the simulation to model dynamic systems, but it ensures cohesiveness of results. The simulation is simply compiled beforehand and parameters are chosen via a user interface which allows core attributes of the simulation to be kept separate from the selected parameters. In addition, the simulation uses EPANET, a water distribution modelling software. This allows the simulation to read in **.inp** files which contain data from real water networks for reconstruction. The reconstruction is rendered using the built-in Windows Application Programming Interface — Figure 1 illustrates the EPANET render alongside the simulation counterpart with point agents for the Net2 network.

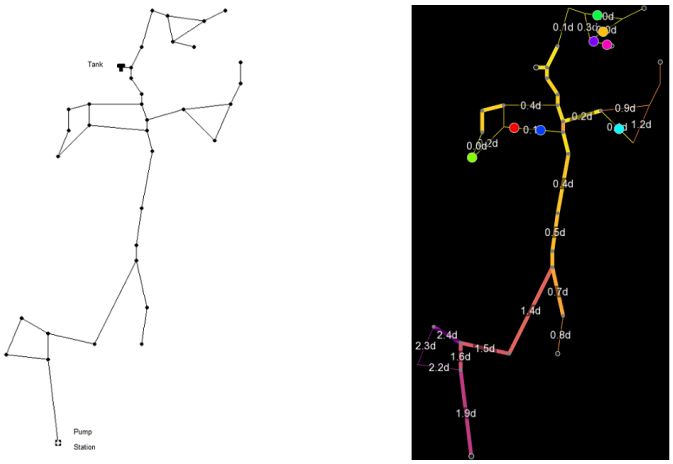

FIGURE 1. The EPANET render and simulation render of the Net2 network respectively.

The network itself is reconstructed as an *undirected graph* wherein vertices and edges represent junctions and pipes respectively. The data provided by EPANET consists of network positions for each junction alongside connectivity information which suffices to form the graph. Additionally, the edges themselves have an ID alongside information such as their length, diameter and roughness. The dynamic water flow is represented by the *demand* of water at each junction, for which there is sufficient data for a 24 hour cycle.

Table 1 details four presented networks and their number of junctions, pipes and cumulative pipe length. The Net2 (Figure 1) and Net3 (Figure 2) networks are artificial whereas the Branching (Figure 3)

TABLE 1. A table detailing the topography of multiple networks.

Network	Junctions	Pipes	Total Length (m)
Net2	35	40	360.00
Net3	92	117	2157.12
Branching	168	175	2858.50
Looping	510	560	28765.51

and Looping (Figure 4) networks are real pipeline networks in England. These networks are also present in the paper detailing the simulation by Parrott, C. et al. [44]. This is because the networks were the only ones available at the time of the research project.

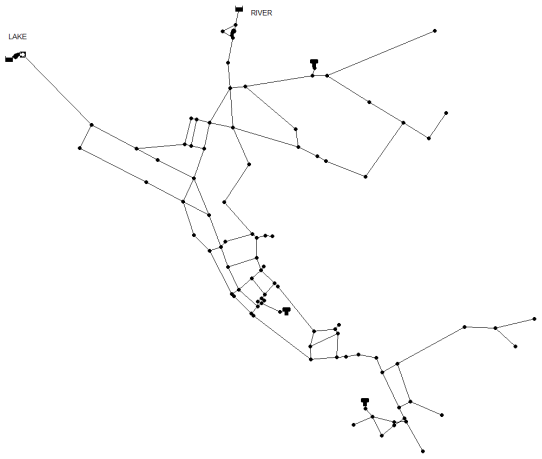

FIGURE 2. The EPANET render for the Net3 network.

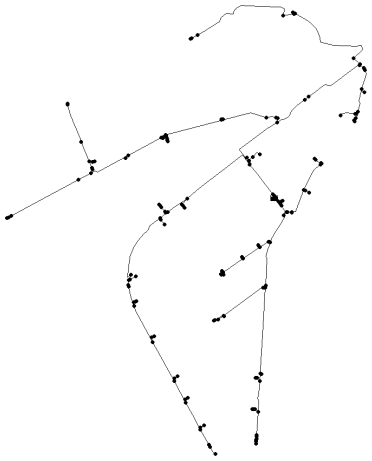

FIGURE 3. The EPANET render for the Branching network.

FIGURE 4. The EPANET render for the Looping network.

After the simulation stores the network, the user interface allows for parameter specification and creates the subsequent output files. Once the parameters have been inputted, the simulation creates the agents and their starting positions in the network. From there, the agents traverse the network and are assumed to be continuously inspecting, with the behaviour they are following making the decisions based on accumulated information. A necessary distinction to make is though the behaviour decides the optimal paths, it is the agents which actively move through the network and follow the strict laws in place within the system. For example, a behaviour might tell an agent to enter a pipe for which the flow is too strong to overcome, in which case the agent in question will forcibly stop moving and enter a rest state until the flow allows traversal. If specified in the parameters, the simulation can replicate dynamic water flows or impose charge requirements on the agents. As the agents traverse the network, their communication methods are simulated and each is provided with an individual belief of the inspection history of the network which assists in the decision making process. Once the simulation has concluded, the normalised mean and standard deviation of the average *Time Between Inspections* (*TBI*) is outputted alongside any other specified metrics.

The Time Between Inspections is a natural measure of the efficiency of the inspection process. For pipeline inspection the objective is to

provide global, continuous inspection. The TBI is defined as the average amount of time between two successive inspections of a specific pipe. At the conclusion of the simulation, it is possible to find the mean value and the standard deviation of the network's TBI. The two in tandem portray how efficient or biased a behaviour is — a low mean demonstrates intelligent, continuous inspection, whereas a low standard deviation shows an unbiased, equal distribution of inspection.

3.2.1. *Simulation Detail.* The simulation has been designed with multiple parameters and experiments in mind. As such there are many functions and applications that require further explanation to relate meaningful results to real networks. Similarly, due to the experimental nature of the simulation certain assumptions have been made which require justification. It is therefore beneficial to provide key elements of the simulation with more detail and address the reasoning behind specific implementations.

The base simulation platform presented by Parrot, C. et al. [44] begins by opening an interface and loading in the following parameters for the simulation:

- Simulation Duration — the duration in milliseconds a single simulation will run for. Typically set to 2,419,200,000 to represent a 28 day month.
- Beginning and End Number of Robots — a simulation set is formed of a single simulation for each sequential number of agents.
- Number of Iterations — the number of simulations sets in the *batch*.
- Robot Driving Speed — the base speed at which the agents will move.
- Communication Type — selects the communication method the agents will use.
- Dynamic Flow System — turns the water flow in the network on or off.
- Visual Render — determines whether the render will be displayed.

As elements of the simulation are built on or adapted (Section 3.3 onwards), the following parameters are required to be set also.

- Overwhelming Flow — turns the ability to become lost on or off.
- SLAM Algorithm — sets the SLAM algorithm to use if lost.
- Flow Intelligence — selects the level of flow intelligence the agents implement.
- Power System — turns the power system on or off.
- Power Intelligence — selects the level of power intelligence the agents implement.

Once verified the parameters fall within the bounds of the system, the program creates directories and files for the results. From here, a *simulation batch* is created — a grouping of simulation sets and instances depending on the iterations specified in the interface. To lower computation time, the program supports *threading*, a technique that allows separate pieces of code to run concurrently instead of consequentially. In this instance, multiple simulations are able to run in parallel on individual cores and processors of the computer. As a core finishes a simulation, it is assigned a new one from the remaining batch. Upon conclusion of the batch the results are compiled and the normalised values are printed on the results files for the reader.

A simulation itself begins by reconstructing the selected network from the .inp file and storing the copy in a class. From here it is possible to access the pipes and their intersections as *links* and *junctions* respectively, with each link having a start and end junction, and each junction having at least one connected link. If dynamic flow is specified, the flow information is stored in a separate class as the water demand of a junction as a function of time.

After the network is stored, the starting positions for the agents are randomised and the agents created. The agents are assigned an index and a *behaviour* which will govern their decision making processes. Whereas the assigned behaviour uses information to decide a course of action, the agent class controls the traversal across the network, from

physically entering junctions and links to calculating movement speeds in the presence of flow. In addition, the agent class contains functions that enable the agents to broadcast and receive *transmittables*, as well as an ability to measure the water flow of the pipe it is in.

At this point the simulation is executed and the agents placed in the network. For each link in the network, the *Time Since Last Inspection* (*TSLI*) is initialised to zero. The Time Since Last Inspection value for a link represents the time passed since it was last inspected by an agent. As the agent traverses the network, the TSLI value for each link is continuously updated and is only reset to zero upon inspection. Figure 1 demonstrates how the render converts the TSLI value into a link colour, to illustrate the current pipes most in need of inspection. A recently inspected pipe is represented by a bright yellow which steadily gets cooler the longer it goes without inspection.

As the agents traverse the network, a *PipelineInspectionLogger* class responsible for recording inspections is continuously called for the duration of the simulation. As an agent enters a pipe, the logger records the time of the simulation and stores it for each pipe such that at the culmination of every simulation, the time and location of every single inspection has been recorded. This enables the TBI value to be calculated at the end of the simulation, independent of changes that can occur from fluctuating patterns in the behaviours.

To help the behaviours make informed decisions, each agent is given an *individual memory*. This memory contains a list of every pipe and it's own perceived TSLI value of when that pipe was last inspected. At the start of the simulation every pipe is given a TSLI of zero, which increases linearly with the simulation time so long as it is not inspected. On entry to a new pipe, the agent sets the TSLI for that pipe in it's own memory back to zero. This information allows the agents to inadvertently keep track of pipes they have inspected recently, while actively seeking to inspect pipes in need of inspection. The agents are assumed to be capable of communicating their memories to one another, from which they update each pipes TSLI with the most recent of the two memories. This helps the agents to provide equal inspections to pipes

that would naturally not be frequented as much as others. Figure 5 illustrates the different beliefs a group of eight agents have for the TSLI values in a network. Each cube segment in the memory bar represents a pipe in the network and, as with the network render, the colour correlates to the TSLI value. The Actual Time Since Last Inspection bar at the top shows the cumulative memory for the network, and is a direct translation of the true values represented on the network render.

FIGURE 5. The individual memories of a network of a group of inspection robots.

Similarities in the agent memories point towards a recent exchange of memory. In Figure 5, Agents 2 and 4 have a near identical memory, whereas Agent 6 shares the pattern, albeit much cooler. This informs the user that Agents 2 and 4 have recently passed one another and communicated their respective beliefs of the network, and that at least one of them communicated with Agent 6 a little further back. Though the render is purely visual and has no bearing on the simulation, it aptly illustrates how the agents are able to exchange information in order to avoid inspecting recently traversed pipes.

The communication itself is controlled by a communication class with a broadcast function that can be called as an agent leaves a junction to enter the next link. It is done so in this manner as at this point, the agent's behaviour must have decided on the next link to follow, and thus the broadcast includes the link about to be inspected in the

broadcast. Not only does this provide all recipients with the most up-to-date information, but it inadvertently avoids detrimental behaviours such as agents on the same link syncing up and following one another. The broadcast is called when the behaviour signals to the agent that there is a *transmittable* to share at the next junction. A transmittable is individual to each behaviour, and can be any information the behaviour needs to communicate in order to inspect as a swarm effectively. Typically this includes the aforementioned agent memories, but can also include relevant path planning information.

The platform supports multiple communication types with different properties and ranges. Depending on the communication type selected, the agent is able to enter a *transmission* state, pausing the agent in place for a prespecified amount of time while it communicates. The transmission mechanic of the simulation is a separate addition to the base simulation presented by Parrott, C. et al. [44] as this is introduced in Section 3.9.1. In addition, the communication class is responsible for calculating all agents in range of the broadcasting agent who can therefore receive the broadcast. This calculation differs depending on the communication type and range selected.

The platform can access the flow data from the .inp file to influence the movement speed of the agents in a link. The flow data available constitutes a 24 hour cycle that changes every half hour. The flow in a link is calculated as a demand of the two junctions it connects from which the *flow impact* on the movement speed is calculated. If the impact is against the agent direction it will slow the agent, and if it goes with the agent direction it will speed up. In addition, if the flow impact is against the agent direction and exceeds the movement speed, the agent enters a *sleep behaviour*. This behaviour stops the agent moving for half an hour, before waking to check if the new flow velocity is traversable. This behaviour is repeated until the flow velocity subsides enough to move again, at which point the behaviour returns to the original planning behaviour. The behaviour represents a hardware assumption that robots are able to clamp themselves to a pipe wall and rest, conserving energy instead of fighting the flow. It is assumed that the robots can be equipped with flow sensing capabilities

and as such, all behaviours are gifted the ability to measure the flow velocities for the connected links at the agent's current junction. This enables the agents to avoid entering a link that immediately traps it in a sleep cycle.

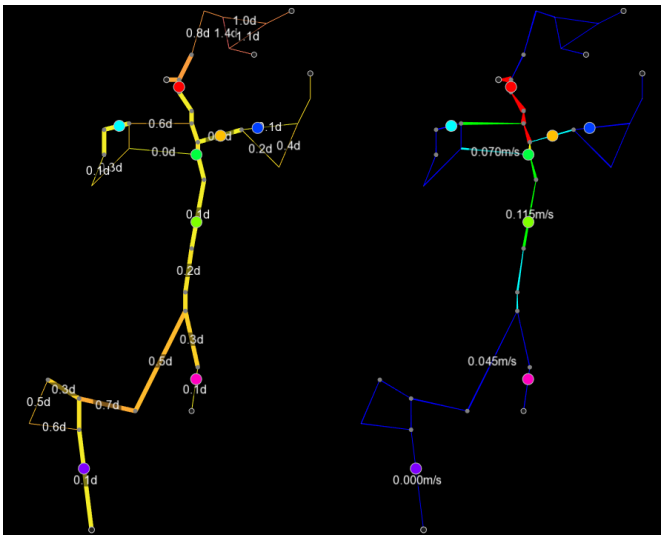

FIGURE 6. The simulation render of Net2 showing the TSLI spectrum and the dynamic flow rates.

Figure 6 shows the simulation render of the TSLI map against the flow velocity map. The links in the flow velocity render are small triangles, with the thicker end representing the direction the flow is heading. The thickness and colour of the triangle represents the flow strength. As is evident from the TSLI map, the topmost section has gone the longest without inspection as it has the coolest colours. The flow map advises this is due to the strength of the flow in the connecting links slowing agents reaching the subsection.

Once the simulation is complete the Time Between Inspection metric is calculated. The time between each inspection event is calculated from the *PipelineInspectionLogger* records before being averaged and normalised with respect to the cumulative network length, L, and the nominal driving speed of the agents, S. This allows for comparison of TBI values regardless of the network in question. The normalised Time Between Inspection value is therefore defined as

$$T_N = \frac{T_{BI} \times S}{L}$$

where T_{BI} represents the raw TBI value and T_N represents the normalised value. Both the mean and standard deviation are calculated, with low values indicating a continual inspection and an impartial inspection respectively.

Figure 7 depicts a state machine of the platform. A finite-state machine is a behaviour model which illustrates the transition between building blocks of states and functions. In this instance it is used to outline the raw design of the simulation process.

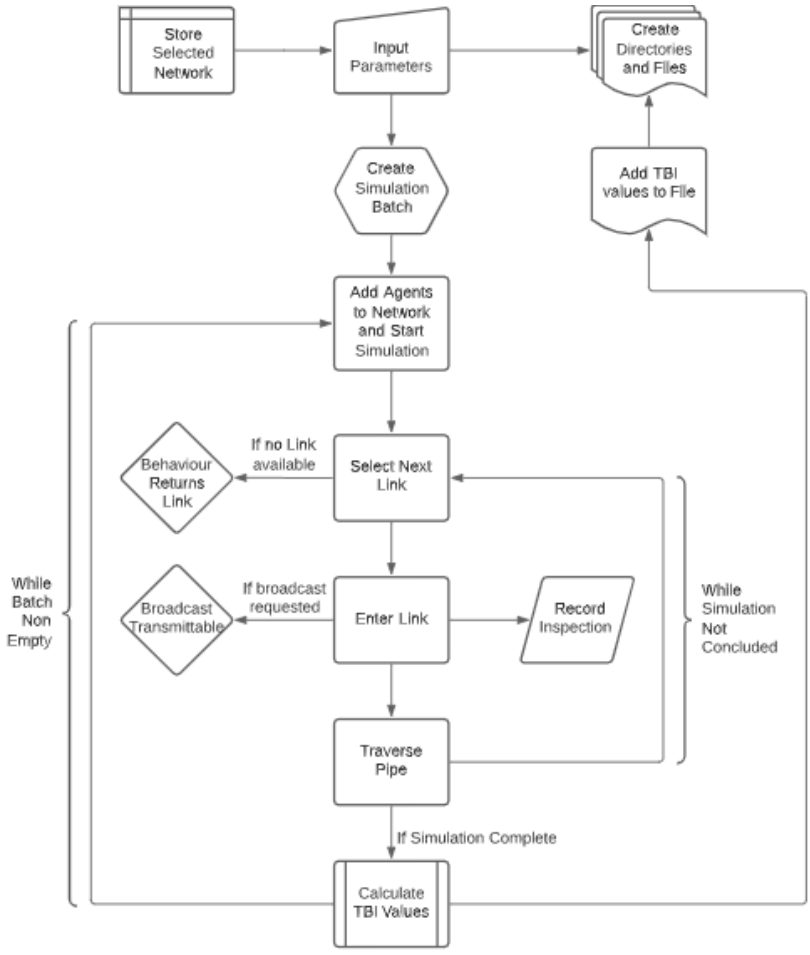

FIGURE 7. The simulation state machine.

3.3. **Behaviours.** In the context of the simulation, a behaviour is a governing intelligence of some description that is designed to improve the inspection process by calculating the best path an agent can take. They can be very simple or complex, and will be adapted through the

course of this thesis to make use of additional information that becomes available to them in the hopes of becoming more efficient. There are four presented behaviours, each with a unique focus or justification. The first is *Stigmergy*, as presented in the work of Parrott, C. et al. [44], a simple behaviour inspired by ant colonies that relies on indirect coordination to lower inspection times. The second, *Greedy Walk* is a computationally inexpensive and fluid stochastic process that plans and broadcasts it's path to encourage a more direct coordination. The *Ad Hoc* behaviour is an extension of the Greedy Walk that uses graph theory to plan routes that culminate in close proximity of one another to circumvent communication issues. Finally, the multiple *Chinese Postman Solution* is a computationally expensive approach that finds optimal routes around individual subsections of the network. This section of the thesis details the behaviours further, alongside their advantages and pseudocodes.

The Stigmergy indirect coordination method is surprisingly powerful for such a simple method. The agents traverse a pipe, each with their own memory of the network and the last time each individual pipe was inspected. As an agent reaches a junction, it determines which pipes have been inspected most recently, choosing the least recent one. Only once a robot comes within communication range of another robot will it exchange information, updating their maps with the others inspection route — there is no planned path to encounter other agents and the interactions occur by chance. The process is repeated each time another robot is encountered so that there is a continuous process of information exchange between robots and corrections to their respective future routes. By following this rudimentary behaviour, unbiased pipe coverage is achieved as the robot cares little about the topology of the network and is solely interested in the immediate TSLI values.

The Greedy Walk behaviour aims to introduce forward planning whilst still being computationally inexpensive. The agent simulates different *random walks* that last a pre-specified time, before choosing the walk which inspected the highest cumulative TSLI. In graph theory, a random walk is a stochastic process that consists of successive random steps, while a *greedy* strategy is focused on pursuing the highest value

at every possibility. The Greedy Walk therefore seeks to find the most optimal solution it can without compromising it's simplicity and lack of bias. The random nature of the walks simulated ensure solutions do not just tend to local optima, but global. As the agent follows the specified walk, it broadcasts not only the inspection memory, but the path it is currently undertaking. This informs other agents of pipes that have been allocated to be inspected in the near future, and therefore aren't in need of inspection.

The Ad Hoc behaviour constitutes multiple agents traversing the network in close proximity to one another. An ad hoc network is a self-sufficient network that does not require full knowledge of an infrastructure. The core concept of the 'sweeping net' is to plan paths for each robot together such that they are within range of another robot at the predetermined time of a communal communication exchange, at which point the path for each is planned again. The robots work together to ensure that they are not traversing similar trails around one another, and the whole group tends towards a cyclic net sweeping around the network. The behaviour uses *Dijkstra's* algorithm to ensure cyclic coverage and aggressively seek outlying junctions that might otherwise be less frequently inspected. This implementation provides a rudimentary solution to communication problems, whilst allowing complete autonomy and a generally even coverage. The efficiency of the algorithm is highly dependent on the number of robots, increasing as the number of robots does. In short, a bigger net covers more surface area.

The multiple Chinese Postman Problem is an ancient graph theory problem wherein a postman wishes to find the quickest route to deliver post to every street on their route, which is solvable by making the graph *Eulerian*. By splitting a network into regions using a simple cluster analysis, it is possible to reduce the pipeline graph into a series of distinct smaller sub-graphs, each with their own individual robot. Once a robot has been allocated a sub-graph area, it will find a complete solution traversing a much smaller area, reducing computational complexity whilst completely bypassing communication issues. Despite the computational complexity, the CPP ensures complete coverage of a

network as each link is part of a cluster an individual robot is responsible for, for which the solution dictates each link must be inspected at least once in a cycle.

3.4. **Stigmergy.** The Stigmergy behaviour detailed presents the adaptations of biological Stigmergy from Bonabeau, E. et al. [4] as implemented by Parrott, C. et al. [44] as a virtual swarm intelligence. At this point the behaviour is unaltered from the presented work, and is included as a benchmark with which subsequent algorithms will be compared. Additionally, the pseudocode is presented.

As previously stated, the Stigmergy behaviour is considered the purest of the swarm algorithms in this research. The algorithms strength lies in its use of *indirect coordination* — the process of achieving sychronized, efficient behaviour by sharing small bits of information. The most common example of this is found within ant colonies. As ants venture from their nests to forage for food, they leave small trace pheromones along the paths they travel. Each distinct pheromone conveys a simple message, for example if a path has a bountiful supply of food, or if there is an enemy colony nearby. Another ant sensing these pheromones is able to make a quick decision, on its own, without the help of the swarm. This process allows armies of ants to draw the same conclusions and make seemingly simultaneous decisions.

To implement this efficiently, the algorithm utilises a 'virtual pheromone' system, the Time Since Last Inspected values previously mentioned. No agent relays instructions to another but by sharing one another memories, the agents are able to achieve indirect coordination. Outside of the communication exchange, the agents follow a very simple algorithm. They travel along a pipe and when they reach a junction they check their memory. They eliminate the pipe they have just traversed as an option, unless they are at a dead end, and choose the pipe which has been inspected least recently. If two pipes are equally in need of inspection, the agent chooses one of them at random. When dynamic flow is introduced, the agents follow the same algorithm with the distinct difference that they eliminate any path with too strong a

flow to traverse as an option. If no options are available at a junction, for example at a dead end where strong flow now keeps the agent trapped, the agent will enter the sleep behaviour before waking and trying again.

Algorithms 1 and 2 details the pseudocode used in the simulation for the Stigmergy behaviour.

Algorithm 1 Stigmergy Algorithm

Require: $CurrentJunction \neq null, DrivingSpeed > 0$

$LastLink \leftarrow null$
loop
  $NextLink \leftarrow$ SelectNextLink $(CurrentJunction, DrivingSpeed, GetTime(), LastLink)$
  if $NextLink \neq null$ **then**
    $TimeSinceLastInspection$ of $NextLink \leftarrow 0$
    Broadcast all known $TimeSinceLastInspection$ values to nearby agents
    Travel along $NextLink$ at $DrivingSpeed$
    $CurrentJunction \leftarrow$ opposite junction of $NextLink$
    $LastLink \leftarrow NextLink$
    if Only 1 link is connected to $CurrentJunction$ **then**
      Synchronise $TimeSinceLastInspection$ values with Node
    end if
  else
    Sleep for 30 minutes
  end if
end loop

The Stigmergy behaviour provides an unbiased approach to pipeline inspection. By considering only the immediate links connected to the junction it inadvertently achieves a high impartiality which, when coupled with the indirect coordination, results in a highly robust and even inspection process. The limitation of not thinking ahead means the performance has the potential to suffer when flow is introduced, getting stuck behind high flows it could have avoided. However, the behaviour is resiliently fair and capable of achieving consistently low TBI values. Often the behaviour becomes cyclic and segmented towards the end of the simulation, as certain groups of agents disperse to certain areas.

Algorithm 2 Stigmergy SelectNextLink Function

```
1: function SelectLink((Junction, DrivingSpeed, CurrentTime, LastLink))
2:     CandidateList ← empty
3:     BackTrackingCandidate ← null
4:     for all Links connected to Junction do
5:         if DrivingSpeed > FlowVelocity of Link at CurrentTime
   then
6:             if Link ≠ LastLink then
7:                 Add Link to CandidateList
8:             else
9:                 BackTrackingCandidate ← Link
10:            end if
11:        end if
12:    end for
13:    if CandidateList ≠ empty then
14:        SelectionList ← empty
15:        MaxTimeSinceLastInspection ← 0
16:        for all Links in CandidateList do
17:            if TimeSinceLastInspection of Link >
   MaxTimeSinceLastInspection then
18:                MaxTimeSinceLastInspection ←
   TimeSinceLastInspection of Link
19:                SelectionList ← empty
20:            end if
21:            Add Link to SelectionList
22:        end for
23:        return A random link from SelectionList
24:    else if BackTrackingCandidate ≠ null then
25:        return BackTrackingCandidate
26:    end if
27:    return null
28: end function
```

This does not seem to hinder the performance of the agents as they seem to settle in even areas with little overlap.

3.5. **Greedy Walk.** Random walks are a stochastic process of sequential random steps with the end goal of finding a solution within some prespecified bounds. The Greedy Walk behaviour relies on simplicity and forward planning in an attempt to find locally optimal paths. The behaviour begins with multiple iterations of random walks in the surrounding area and calculates the cumulative TSLI value they inspect.

The iterations are random, with each walk a collection of random decisions at each junction. However, the behaviour estimates the time each link takes to inspect and the path concludes once the cumulative time exceeds a prespecified value. By weighting each link with the time it will take to inspect, the Greedy Walk is able to include small outlying pipes or clusters of small pipes in the path as they take little time to traverse, but provide a large TSLI gain.

In addition to planning ahead, the Greedy Walk behaviour is able to broadcast the path it is following to surrounding agents. Though most behaviours share the inspection memory in a similar manner to Stigmergy, the addition of including a future path in the broadcast enables the agent to avoid future crossover as well. As such, agents in close proximity either directly inspect together efficiently to clear a small section, or one of the agents finds another route and they split to inspect independent areas. This helps reduce the time between inspections as the agents are able to plan around one another and prioritise pipes that might otherwise have been left.

Algorithm 3 details the pseudocode used in the simulation for the Greedy Walk behaviour.

Algorithm 3 Greedy Walk SelectNextLink Function

```
function GREEDY WALK((CurrentJunction, LastLink))
Require: CurrentJunction ≠ null
    if PathList non-empty then
        if DrivingSpeed < PathList Front (FlowVelocity) then
            Clear PathList
        end if
    end if
    if PathList empty then
        PathPlan(CurrentJunction, LastLink)
    end if
    NextLink = PathList Front
    PathList delete Front
    return NextLink
end function
```

3.6. **Ad Hoc.** The Ad Hoc behaviour is inspired by the implementation of ad hoc networks in natural disasters and their ability to remain

Algorithm 4 Greedy Walk PathPlan Function

```
1: function PathPlan((CurrentJunction, LastLink))
2:     bestCumulativeTSLI ← 0
3:     for all Iterations do
4:         while PathTime < MaxWalkTime
5:         for all Links connected to CurrentJunction do
6:             if DrivingSpeed > FlowVelocity of Link at CurrentTime then
7:                 if Link ≠ LastLink then
8:                     Add Link to CandidateList
9:                 end if
10:            end if
11:        end for
12:        if CandidateList non-empty then
13:            NextLink ← random Candidate
14:            Add NextLink to IterationsList
15:            CurrentJunction ← NextLink end junction
16:            CumulativeTSLI = CumulativeTSLI + TimeSinceLastInspection of Link
17:        end if
18:        end while
19:        if CumulativeTSLI > bestCumulativeTSLI then
20:            bestCumulativeTSLI ← CumulativeTSLI
21:            bestPathPlan ← IterationsList
22:        end if
23:    end for
24:    return bestPathPlan
25: end function
```

cohesive with little to no infrastructure. The focus of this algorithm is to have the agents in close proximity to one another at an appointed time intervals where a communication exchange will occur. By overcoming the communication dilemma the algorithm aims to actively have agents work together, with each agent sharing their planned path. The agents follow a hierachy, with the first planning a route and sharing it's chosen route with the next agent. This agent then calculates a path that ends in communication range of the former, before passing on *both* plans to the next agent. Each agent will subsequently choose a path that compliments the swarms current plan. The end result is a unique swarm behaviour that sees a net of agents initially expand as they inspect the network before contracting in the concluding stages

of their path plans so as to be in communication range for the next 'meeting'.

The Ad Hoc behaviour relies heavily on *Dijkstra's* algorithm — a method of finding the shortest path on a graph between two vertices. Using Dijkstra's, the first agent is able to find all the junctions in the network it can reach in time for the next communication 'ping'. Though computationally more expensive, unlike the $A*$ heuristic, Dijkstra's is complete and will always find the shortest path. It works by producing the *shortest-path tree* — fixing a source node, spreading from this to calculate the shortest path to every node in the network which ultimately includes the second node in question. Once the first agent has found all junctions within range, it selects the junction with the largest cumulative TSLI value. From here, the agent will calculate which of the surrounding junctions are within communication range, depending on the type and range selected. Once calculated, the agent broadcasts a memory, a path it will follow and the candidate list of junctions in range of it's own new end point.

Upon receiving a broadcast, the following agent will examine the candidate list for any junctions it can reach in time. If this exists it will follow the same process the first agent and add to the candidate list. If it cannot reach one of the candidate junctions in range, it will instead find the highest TSLI junction it can reach with Dijkstra's. By iteratively passing a junction candidate list through the agents and implementing a greedy Dijkstra's approach, the agents tend to clusters of agents actively inspecting together with rare satellite agents that aggressively seek outlying nodes. It should be noted, that whilst it seems nonsensical to calculate the shortest paths using Dijkstra's to then choose the most expensive of those options, the alternative is to use a much more computationally expensive *longest-path* algorithm before selecting a path that can be completed in time, a shorter one.

Algorithms 5 and 6 detail the pseudocode used in the simulation for the Ad Hoc behaviour and Dijkstra's algorithm.

The efficiency of the Ad Hoc algorithm is reliant on the number of robots in a network — too few and the swarm struggles to maintain

Algorithm 5 Ad Hoc Algorithm

```
1: procedure AD HOC((CurrentJunction, NodesInRange, CommunicationTime))
2:     if NodesInRange empty then
3:         for all Junctions do
4:             ShortestTravelTime ← Dijkstras(CurrentJunction, Junction)
5:             if ShortestTravelTime ≤ CommunicationFrequency then
6:                 Add Junction to CandidateList
7:             end if
8:         end for
9:         BestValue ← 0
10:        for all CandidateList do
11:            if PathValue ≥ BestValue then
12:                BestValue ← PathValue
13:            end if
14:        end for
15:        TimeSinceLastInspection of PathLinks ← 0
16:        for all Junctions do
17:            if InCommunicationRange(EndNode, Junction) = true then ▷ This function checks if a junction is in communication range of the node at the next exchange. Add Junction to NodesInRange
18:            end if
19:        end for
20:        Broadcast all known TimeSinceLastInspection values and NodesInRange to nearby agents
21:    else
22:        for all NodesInRange do
23:            ShortestTravelTime ← Dijkstras(CurrentJunction, Junction)
24:            if ShortestTravelTime ≤ CommunicationTime then
25:                Add Junction to CandidateList
26:            end if
27:        end for
28:        BestValue ← 0
29:        for all CandidateList do
30:            if PathValue ≥ BestValue then
31:                BestValue ← PathValue
32:            end if
33:        end for
34:        TimeSinceLastInspection of PathLinks ← 0
35:        for all Junctions excluding NodesInRange do
36:            if InCommunicationRange(EndNode, Junction) = true then Add Junction to NodesInRange
37:            end if
38:        end for
39:        Broadcast all known TimeSinceLastInspection values and NodesInRange to nearby agents
40:    end if
41: end procedure
```

Algorithm 6 Dijkstras Algorithm

```
1: function DIJKSTRAS((StartJunction, EndJunction, Graph))
2:     while Not at EndJunction
3:     for all Links at CurrentJunction do
4:         if OptimalDistance of CurrentJunction + LinkLength <
   OptimalDistance of OtherJunction then
5:             OptimalDistance of OtherJunction ←
   OptimalDistance of CurrentJunction + LinkLength
6:         end if
7:         if LinkLength < ClosestNeighbour and Unvisited then
8:             ClosestNeighbour = LinkLength
9:             NextJunction ← CurrentJunction
10:        end if
11:    end for
12:    if NextJunction index is invalid then
13:        NextJunction ← VisitedSet Front
14:        Pop VisitedSet Front to the back
15:    else
16:        Add CurrentJunction to VisitedSet
17:    end if
18:    Set CurrentJunction as Unvisited
19:    CurrentJunction ← NextJunction
20:    end while                                              ▷
   Now every Junction has an optimal distance from StartJunction
   and Dijkstra's follows this back from the EndJunction
21:    while Not at StartJunction
22:    for all Links do
23:        if OptimalDistance of OtherJunction < ShortestPath
   then
24:            ShortestPath = OptimalDistance of OtherJunction
25:            NextJunction ← OtherJunction
26:        end if
27:    end for
28:    CurrentJunction ← NextJunction
29:    end while
30:    return OptimalDistance of EndJunction
31: end function
```

form, or does so with agents following similar paths. On the other hand, too many robots causes the 'net' to saturate, with those lower in the hierachy seeing increasingly low TSLI returns on their paths. A sweet spot exists, where the network is sufficiently saturated with agents to maintain a cyclic net, but not so many agents that the latter

agents are surplus. In spite of this, the goal of the Ad Hoc approach is to limit the frequency of communication required, particularly as transmission times are introduced into the network. Where other behaviours requirement for consistent transmission compounds the effect of longer transmission times, the Ad Hoc approach suffers only a fraction of the detriment. Similarly, because of the fluidity of the behaviour — planning around only a sole end point, the behaviour is able to retain consistent inspection times when flow and communication issues are examined in tandem.

3.7. **Multiple Chinese Postmen Problem.** The multiple Chinese Postmen Problem investigates the possibility of autonomous inspection without communication. The concept has evolved around the Chinese Postman Problem, a very old mathematical dilemma that attempts to find the optimal path around a graph such that every edge is traversed at least once. This can be extrapolated to pipeline inspection, creating a cyclic path wherein every pipe is inspected by a robot at least once a cycle. The solution revolves around making the graph *Eulerian* — that is, every vertex must have an even degree. This is done by matching up pairs of odd degree vertices and creating an additional artificial path between them consisting of current real edges that can be traversed twice. Once the network is Eulerian, it is possible to create a shortest cycle that traverses each real and artificial path once using Fleury or Hierholzer's algorithms — algorithms for finding paths around Eulerian networks.

The issue for the Chinese Postman Solution revolves around it's computational expense in larger graphs. As the graph size increases, so too does the number of odd degree vertices. The combination of odd degree vertices that gives the optimal pairings is iteratively calculated by examining the *lexicographic* order — a technique for ordering and swapping variables in a manner that ensures every possible combination is examined. Therefore, the combination of odd degree vertex pairings that finds the optimal articial path introductions increases exponentially as the network grows. 12 odd degree vertices means there are $12! = 479,001,600$ lexicographic combinations of odd pairings. Given

that the Looping network in Figure 4 has 296 odd degree vertices, a smarter implementation is necessary.

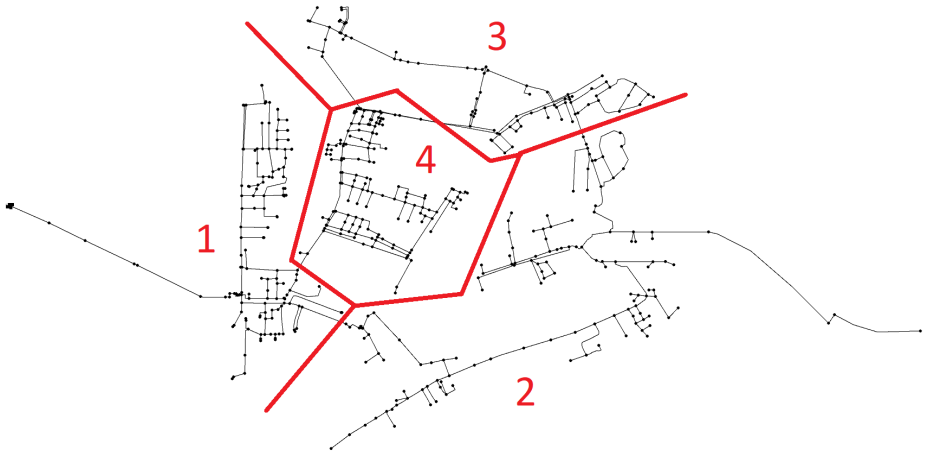

FIGURE 8. An example of how a DMA split is introduced to divide the Looping network with four agents.

Figure 8 illustrates how the network can be divided using District Metred Area analysis. At the beginning of a simulation, each junction and link in the network is assigned to a robot based off its proximity to the robot's starting position — it joins the cluster of the robot it is closest to using Dijkstra's. Once the clusters have been assigned, the boundary links between different clusters are added to both clusters they join by adding the other junction end to each cluster. This ensures every link is accounted for, and every cluster is a connected graph. Not only does this help to reduce the computational complexity of a network but it ensures consistent, local coverage for each cluster.

In addition, a heuristic approach can make use of additional, advantageous information provided by EPANET. Typically, the ID numbers that junctions are assigned are ordered. That is, a junction who's ID is close to another junction's ID is likely to be close to that junction in the network. Extending this logic allows us to create the first lexicographic list wherein every odd degree vertex ID is in size order. By coupling off junction pairs with close IDs it is possible to assume that the cumulative additional artificial paths is close to the optimal, as the paths introduced will, in the majority, be between junctions close together. By introducing this simple heuristic approach whenever the

number of odd degree vertices in a cluster is deemed excessive, it is possible to rapidly calculate CPP solutions that otherwise make this method unusable.

Below the pseudocodes used in the simulation for the CPP behaviour algorithm are detailed:

Algorithm 7 Multiple CPP SelectNextLink Function

function SELECTLINK($(Junction, CurrentTime, LastLink)$)
  if $PathList = empty$ **then**
    $PathList \leftarrow$ MCPP($CurrentJunction, ClusterAssignment$)
    while $PathListFirstEntry \neq CurrentJunction$ **do**
      Remove $PathListFirstEntry$ from $PathList$ and add it to the back of $PathList$
    end while
  end if
  $NextLink \leftarrow PathListFirstEntry$
  Remove $PathListFirstEntry$ from $PathList$ and add it to the back of $PathList$
  return $NextLink$
end function

Algorithm 8 MCPP Function

procedure MCPP($(CurrentJunction$,$ClusterAssignment)$)
  $PathPlanned \leftarrow false$
  while $PathPlanned = false$ **do**
    $EulerianCluster \leftarrow$ Eulerian($ClusterAssignment$) ▷ This function takes the cluster assignment and makes it Eulerian.
    $SolutionFound \leftarrow false$
    while $SolutionFound = false$ **do**
      $PathList \leftarrow$ FleurysAlgorithm ($CurrentJunction, EulerianCluster$)
    end while
    $PathPlanned \leftarrow true$
  end while
  return $PathList$
end procedure

This algorithm requires no communication to be effective, and the computational costs of running once a path is found are minimal. Without flow, the algorithm guarantees consistent, cyclic coverage of every pipe in the system. However, the starting computations required to find the paths can become extremely expensive without a heuristic approach

Algorithm 9 Eulerian Function

```
1: procedure EULERIAN((ClusterAssignment))
2:     for all Junctions do
3:         if Junction degree is odd then
4:             Add Junction to OddDegreeList
5:         end if
6:     end for
7:     for all OddDegreeList do ▷ Creates initial lexicographic list.
8:         Arrange in size order depending on JunctionID
9:     end for
10:    Define InverseLexicographic
11:    while Lexicographic ≠ InverseLexicographic
12:    for all LexicographicPairs do
13:        AdditionalPaths = AdditionalPaths + Dijkstras(LexicographicPair)
14:    end for
15:    if AdditionalPaths < bestCumulativeAdditionalPaths then
16:        bestCumulativeAdditionalPaths = AdditionalPaths
17:        bestArtificialPathsAdded = ArtificialPathsAdded
18:    end if
19:    Lexicographic(LexicographicList)
20:    end while
21: end procedure
```

and, given its rigid one-path solution, it struggles to adapt to flow changes. Though realistic anyway, it becomes necessary to spread out the agents at the beginning of the inspection process intelligently. This improves the behaviours efficiency by ensuring that the region distribution, and therefore the work distribution, are even.

3.8. **Simulations.** To analyse the efficiency of the proposed behaviours, a set of base simulations were run to provide a benchmark comparison. This simulation was run for a 28 day period 20 times for 1 to 32 robots. The 28 day timestep is chosen to counter any bias from the chosen starting positions, giving the agents time to break out of their respective areas, whilst inspecting long enough to provide meaningful results. The 20 iterations aims to eliminate any anomalies that can occur in single simulation runs. To determine how efficiently the agents function as a swarm, the number of agents increases after each set of simulations, from 1 to 32. This allows us find a *critical point* at which

Algorithm 10 Lexicographic

```
1: function LEXICOGRAPHIC(((LexicographicList)))
2:     for all LexicographicList going backwards do
3:         if LexicographicsList[Index] < LexicographicList[Index + 1] then
4:             FirstCharacter ← LexicographicList[Index]
5:             FirstIndex = Index
6:         end if
7:     end for
8:     for all LexicographicList above Index do
9:         if LexicographicsList[Index] < FirstCharacter then
10:            CeilingCharacter ← LexicographicList[Index]
11:            CeilingIndex = Index
12:            break loop
13:        end if
14:    end for
15:    Swap FirstCharacter and CeilingCharacter
16:    for all LexicographicList above FirstIndex do
17:        Sort into size order.
18:    end for
19: end function
```

Algorithm 11 Fleury's Algorithm

```
1: function FLEURYSALGORITHM((CurrentJunction, EulerianCluster))
2:     CycleSize ← NumberLinksInEulerianCluster
3:     for CycleSize do
4:         for all ConnectedLinks do
5:             if Link also has an ArtificialLink then
6:                 Add Link to ViableOptions
7:             end if
8:         end for
9:         if ViableOptions is empty then
10:            for all ConnectedLinks do
11:                Add Link to ViableOptions
12:            end for
13:        end if
14:        Choose random Link from ViableOptions
15:        CurrentJunction ← LinkOtherJunction
16:        PathList add Link
17:    end for
18:    return PathList
19: end function
```

the introduction of new agents provides little additional benefit as well as determining which behaviours function better in sparser or denser environment. For these simulations the driving speed has been kept *consistent* so as not to interfere with the *TBI*.

The simulations were run without communication or dynamic flow in order to ascertain a base level of inspection. The results have been normalised and plotted on a log-log scale to illustrate the relationship between the number of robots and TBI clearer.

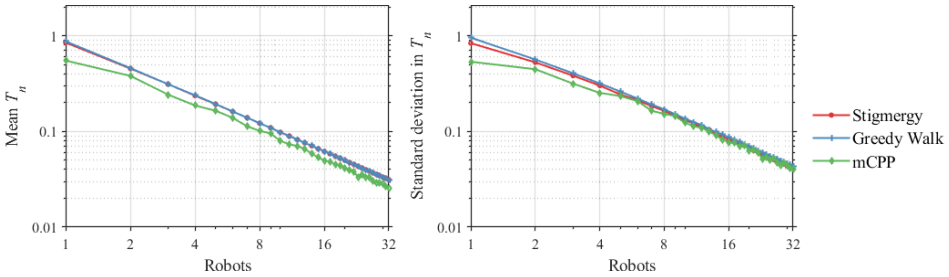

FIGURE 9. A graph showing the relationship between the Time Between Inspection and number of robots in the Looping Network.

Initially it is obvious that the more agents in a system, the more frequent the inspections. This is shown by the generally linear downhill slope of mean and standard deviations in Figure 9. As the number of agents in a network is increased, the frequency of inspections rises as there are simply more agents traversing the network at one time. Early additional agents provide a much higher return likely due to low numbers of agents in a system being unable to cover the entire system. Therefore additional agents provide a bigger benefit than when the swarm is already large. As the number of agents in the network reaches 16, the introduction of more agents starts to see diminishing returns, as is evident by the congested mean values. The network has become *saturated* as agents find themselves frequently in close proximity to one another, inspecting pipes that have just been inspected. The effect is mirrored in the standard deviation of the TBI, where additional early agents provide a much more significant benefit than those introduced in the latter stages. Again, this indicates a critical point where the current swarm is able to inspect the entire network sufficiently.

The mean values for each behaviour are extremely close together, especially as the network saturates with agents. However, this is to be expected when the simulation is run without a dynamic flow present. Without flow, the agents in a network are continuously moving and inspecting regardless of the paths they take. As such, the mean values tend to be quite close together. For flow-less simulations, a better indicator of performance is the standard deviation. The standard deviation of the TBI represents the disparity between the frequency that individual pipes are inspected. A low standard deviation indicates that most pipes in the network are inspected with the same frequency, whereas a high standard deviation highlights a discrepancy in the inspection process.

With regards to the behaviours themselves the multiple CPP solution seems to perform the best. This is unsurprising — the algorithm specifically constructs an optimum path around a cluster, which in itself ensures consistent coverage. As the network is split into clusters, the mCPP can ensure consistent, cyclic coverage. In addition, the agents can only overlap at the cluster borders which helps contribute to the low mean TBI. However, the mCPP is the most rigid of all the behaviours and therefore most susceptible to flow and other considerations.

The Stigmergy and Greedy Walk behaviour are extremely consistent, as demonstrated by the perfectly linear lines in both the mean and standard deviation plots. Though they have higher mean values than the mCPP, the behaviour outputs competitively low standard deviation values. Without communication, the agents have only their own TSLI memory to utilise. Therefore, the agents inspect the areas and pipes they personally have not inspected yet and as such, tend toward a cycle around the network. This means the introduction of new agents has no weight on the decision process of other agents, and subsequently each follows their own cyclic route of the network. This helps account for the extremely linear relationship between the TBI and the number of robots as introducing a new agent is simply introducing a new cycle of coverage into the system.

Due to issues with the code during the research project, the Ad Hoc behaviour has not been included in the simulations, and is instead left as Future Work in Section 6.3.

3.9. **Implementing Communication Between Robots.** The ability to communicate is a necessary and powerful tool in biological systems. Sharing information about the surrounding habitat, usually regarding food or danger, is beneficial to each member of the society. An ideal swarm environment relies on this, with each individual fulfilling a task and communicating to the swarm both to convey and receive information as necessary. Unfortunately, due to the extreme variability of the pipe conditions and the density of the surrounding earth, an underground pipeline network attenuates common communication waves robots can communicate with. Creating a swarm behaviour with limited communication capabilities is a challenge for the field as ultimately the robots are forced to compromise the routes they can take to exchange the necessary information to achieve optimal performance.

It is possible to demonstrate the detrimental effect of a lack of communication with a simple addition to the simulation. As aforementioned, the communication is controlled by a communication class with a broadcast function that is called. It is therefore easy to add a basic communication process capable of exchanging the relevant information agents need to make informed decisions. For the purpose of this specific simulation, the broadcasting and receiving of a message are considered instantaneous by any robot within range. Though subsection 3.9.1 introduces a transmission delay to reflect the time needed to communicate, this simulation is concerned only with the effect of the *range*.

In this instance, the range constitutes the sum of the length of the pipes a signal can travel. That is, two agents are deemed in range of one another if the length of the shortest path between them does not exceed the range. The communication class calls a basic exploratory *Branch and Bound* algorithm, Algorithm 12 to ascertain all junctions and links within range. Though the simplest experiment would have detailed the range by a linear radius with the broadcasting agent at its epicentre, the environment of the network dictates the signals must

also traverse the network, as they cannot pass through the pipe walls or surrounding earth.

Algorithm 12 Communication Branch and Bound Algorithm

```
1: function BRANCH AND BOUND((Junction, InitialBroadcastRange))
2:     if At a Junction then
3:         Add (Junction, InitialBroadcastRange) to BroadcastJunctions
4:     else
5:         DistanceFromLastJunction = LinkTravel
6:         DistanceToNextJunction = LinkLength − LinkTravel
7:         for all Agents in Link do
8:             if Agent in broadcast range then
9:                 Add Agent to AgentsToBroadcastTo
10:            end if
11:        end for
12:        Add (StartJunction, InitialBroadcastRange − DistanceFromStart) and (EndJunction, InitialBroadcastRange− DistanceFromEnd) to BroadcastJunctions
13:    end if
14:    while BroadcastJunctions non empty
15:    Broadcast ← BroadcastJunctions Front
16:    BroadcastJunctions remove Front
17:    if Broadcast has not been broadcast to then
18:        Broadcast set to broadcast to
19:        for all BroadcastConnectedLinks do
20:            if Agents in Link in range then
21:                Add Agent to AgentsToBroadcastTo
22:            end if
23:            BroadcastRemainingRange = Broadcast range −LinkLength
24:            if BroadcastRemainingRange > 0 then
25:                Create Iterator until BroadcastRemainingRange expires
26:                Add Agents to AgentsToBroadcastTo
27:            end if
28:        end for
29:    end if
30:    for all AgentsToBroadcastTo do
31:        Agent receive transmittable
32:    end for
33: end function
```

In order to study the effects of the communication range of the robots, each simulation was run with four different communication ranges as

a proportion of the cumulative length of the network. The frequency of a communication exchange in a larger network is much lower than a smaller network with the same communication range so by implementing the range as a percentage of the network length, the TBI values are normalised for every network. The ranges used are No Communication (0%), 1%, 10% and Infinite (100%). Figures 10 and 11 demonstrate the effect that different communication lengths have on the Time Between Inspection for behaviours in the Looping network as an increasing number of robots are introduced to the network. As the multiple Chinese Postman solution was specifically chosen as a comparison because of it's indifference to communication, the effects can only be analysed on the Stigmergy and Greedy Walk behaviours. It must be noted that though both Figure 10 and Figure 11 have been run on the same simulation version for parity, the Stigmergy behaviour and base simulation remain the same as those presented by Parrott, C. et al. [44] and as such, Figure 10 will match these results closely.

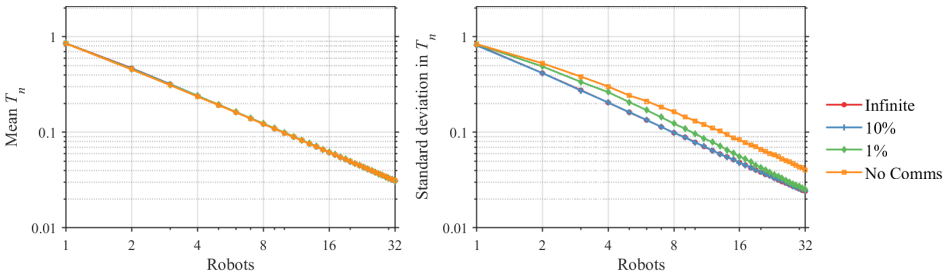

FIGURE 10. The effects different communication ranges have on the Stigmergy Time Between Inspection in the Looping network.

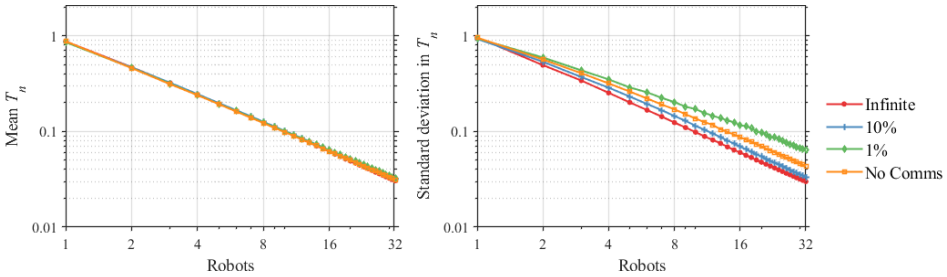

FIGURE 11. The effects different communication ranges have on the Greedy Walk Time Between Inspection in the Looping network.

The most important thing to note when examining the communication results is the distinction between the mean and standard deviation TBI results. Whereas the mean is concerned with the overall average time it took to inspect the network, the standard deviation describes the consistency of coverage throughout the network. At this stage it is unsurprising that the mean TBIs remain consistent throughout — the simulation is not considering flow and therefore the agents are constantly moving at the same arbitrary speed. This leads to identical mean TBI results as the mean is concerned with the frequency at which pipes are inspected, and the agents are constantly inspecting regardless of their communication ranges.

To investigate the effects of a simple communication process it is therefore much more appropriate to examine the standard deviation of the TBIs. This illustrates how effectively agents are working together to attain consistent coverage. A high standard deviation is undesirable as it indicates a discrepancy between the frequency at which individual pipes are inspected.

Examining the standard deviation values in Figure 10, it is apparent that Stigmergy's coverage remains relatively consistent. As expected, the behaviour follows the trend that the longer the communication range, the more effective the behaviour. This is because a larger communication range is directly proportional to a more frequent exchange of communication. As such, the agents are able to make better informed decisions as to the state of the network. Interestingly the 10% range is comparable to the infinite communication. Due to the size of the network (Table 1), the Stigmergy agents are just as affected by the 10% range as the infinite. The 10% range is more than sufficient for them to make locally optimal decisions, and by the time the network saturates the network is sufficiently covered by the 10% range regardless. The consistency stems from Stigmergy's relatively simple decision making process — only considering the immediate surrounding pipes at a junction unconsciously provides Stigmergy with an unbias approach to inspection. As communication is introduced, before the agent can traverse the whole network, it is communicated that a specific region has already been done and as such the agents implicitly tend toward a

network partition, with each group providing an even coverage of their area.

A similar scenario is evident in the Greedy Walk standard deviation in Figure 11. Instead however there is the unique case where a 1% communication range performs worse than the No Communication range. This problem is specific to the Greedy Walk behaviour, and is a reflection on it's desire to inspect locally optimal routes. A 1% communication range encourages an exchange of information which statistically occurs mid-plan. As such, at the conclusion of the current path, two agents again create a greedy path, that inevitably concludes with them each inspecting the same paths from their shared memory. As the communication range becomes sufficiently large for the agents to communicate without overlap, the performance increases again.

Figures 10 and 11 illustrate the necessity for an effective communication system. An irresponsible communication process can inadvertently reduce the inspection efficiency, whilst a suitable one will inevitably improve the consistency of inspections. As additional factors such as water flow or robot power are introduced it is imperative that robots are able to communicate effectively to negate the additional detrimental effects.

3.9.1. *Acoustic and Ultrasonic Communication.* It is evident that an optimal inspection process requires sufficiently frequent communication for robots to work together effectively. The main difficulty the communication process must overcome is the variability of the pipes, from material to condition of the pipe. The attenuation of radio waves with which robots might communicate is too high, struggling to convey a signal that details large volumes of information inspection history and planning information. Other obstacles include multi-path propagation and limited available bandwidth. In contrast, there is a vast array of literature detailing the advancements in underwater acoustic communication.

Acoustic waves have an extended range that can travel through water, but also propagate round corners well despite their slow propagation

speed. This is advantageous as the robots do not necessarily have to be in sight of one another to communicate, meaning robots can communicate from further distances around problematic topological features. In addition, acoustic methods are well documented and adaptable, with hydrophones small and cheap.

As well as acoustic sensing, ultrasonic methods show surprising promise for underwater communication. By operating in the smaller wavelengths of ultrasonic frequencies, transmitters and receivers operate on a lower energy consumption because of their size. This allows for frequent communication exchange, albeit it at a significantly smaller range. Ultrasonic waves struggle to propagate round corners and for the purposes of pipeline communication, require a line of sight to the receiver.

The benefits of each of these types of communication is that they coincide with common inspection techniques and provide relatively cheap and small additions to an inspection robot. Similarly, both have documented applications in pipes of a range of typical materials, from cast iron to PVC and concrete. The simulation will implement both methods of communication as viable techniques as each behaviour may benefit from a differing communication type.

3.9.2. *Acoustic and Ultrasonic Implementation.* When implementing communication in the simulation it is important to keep the behaviour and communication separate. By keeping the communication in a separate class, the behaviour is able to choose *when* to call the communication type to share information, as opposed to being at the mercy of when the communication class might choose to communicate. This provides a more intelligent approach as each behaviour shares information a different way. The simulation is hard-coded with transmission times for both communication types, calculated by the amount of information to be exchange over the speed at which the communication type can exchange information. As is likely to be the case in a real network, the agents pause the calculated amount of time to communicate and receive this information.

For the size of the information to be exchanged, it is assumed the Inspection Memory is the only transmittable passed in. It is also assumed that the transmittable is communicated via binary code as this is the quickest and most reliable method. Therefore, to convey time as a function of days, hours, minutes and seconds requires $28, 24, 60, 60$ binary options or $11100, 11000, 111100, 111100$. As such, for each link in the network, the information to be transmitted is of the size $5+5+6+6 = 22$ bits.

For acoustic communication, the aim is to provide a stream of reliable data at a long range. Chirp Linear Frequency Modulation is the acoustic method that provides the longest range in a pipe at 50 metres. However, the communication exchange is slow, with a rate of approximately 7 bits per second. In addition, the method requires a 10 second preamble before the message pad to inform the receivers they are about to receive a broadcast, as well as a 10 second post-amble to inform them the broadcast is over. Examining Table 1, we see that were the agents to communicate an entire network's inspection history in every broadcast, the transmission time might take just under half an hour. However, another acoustic method, Amplitude Shift Keying provides a much more competitive bit transmission speed of 6350 bits per second. This greatly reduces the transmission time to approximately 2 seconds, though it still requires the pre and post amble. As such, the acoustic communication is modelled with a 22 second transmission time, a 50 metre range and the ability to go around corners. The range given corners is again calculated with the Branch and Bound algorithm — Algorithm 12.

On the contrary, ultrasonic provides a much quicker transmission at the expense of a competitive communication range. Because of the speed and size of ultrasonic waves, the range struggles to exceed 10 metres in a pipe and does not attenuate well round corners. However, the speed at which information is communicated is rapid even compared to the Amplitude Shift Keying acoustic method. As such the ultrasonic method is modelled with *negligible* transmission time, but with a 10 metre range and an inability to go round corners.

3.9.3. *Acoustic and Ultrasonic Simulations.* To analyse the effectiveness of acoustic and ultrasonic waves requires a slightly different approach to the normalised communication ranges. Because the ranges of acoustic and ultrasonic waves are finite, it is not realistic to model them as a normalise value dependent on the cumulative size of a network. In addition, the size of the network is an important factor — the same number of robots in a large network are likely to be spread further out than those in a small network. As such, because the communication range is specified, by definition the frequency with which the robots will communicate will be higher in a smaller network.

To compare the two ranges, the simulations have been run on the Stigmergy behaviour in both the Looping and Net2 networks. The Stigmergy behaviour has been selected because it is the behaviour that communicates most frequently, at the exit of every junction. Therefore the differences in the communication type, including transmission time, will be highlighted the most. The Looping and Net2 networks were selected for these simulations as they are the largest and smallest networks respectively.

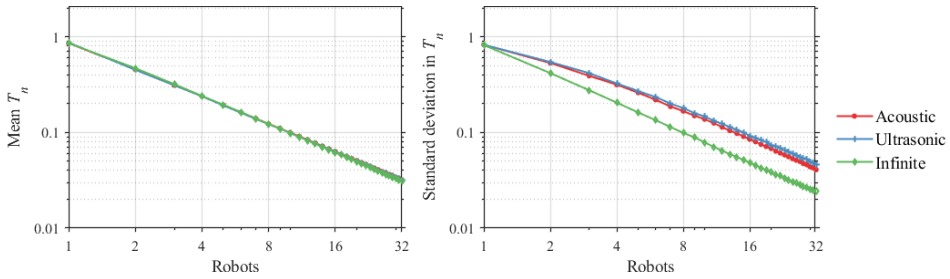

FIGURE 12. The effects of Acoustic and Ultrasonic Communication for the Stigmergy behaviour on the Looping network.

Immediately it is obvious that there is little difference between the two ranges. As the behaviours are the same, the mean values being identical is predictable. However, the similarities between the standard deviations is telling. Specifically, at every intersection the acoustic communication is waiting for 22 seconds while the ultrasonic transmission time is negligible. The fact the standard deviations are so similar means that the acoustic range is supplementing the deficit of the transmission time with a more intelligent swarm inspection. On the other

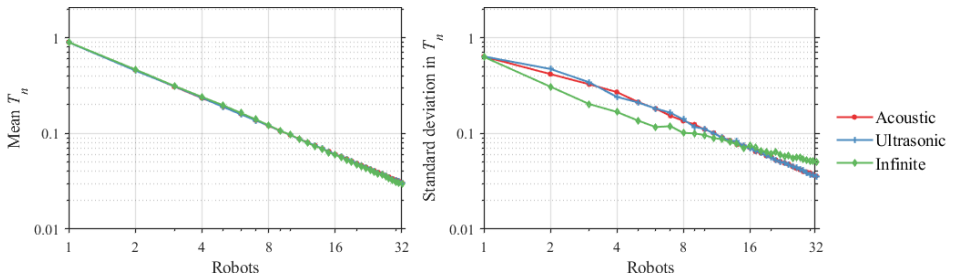

FIGURE 13. The effects of Acoustic and Ultrasonic Communication for the Stigmergy behaviour on the Net2 network.

hand, the ultrasonic method allows the agents to be continuously moving to help overcome the disadvantage of not signals not attenuating around corners.

The standard deviation in Figures 12 and 13 illustrate the effects of network size with relation to both ranges. As expected, the smaller network has the lower standard deviation out of the two as a whole. This is due to the range sizes physically filling a larger portion of the network to allow more frequent communication. Interestingly in the larger network both signals are closer to the No Communication seen in Figure 10 than the infinite communication range, which is a strong indication of the insufficient range size appropriate for a network that large. As suspected, in this instance the acoustic method seems to be better by the smallest of margins, as the reach outweighs the additional distance the ultrasonic agents are required to cover to find one another in more sparsely populated network. Similarly, the Looping network has more corners round which the ultrasonic waves cannot attenuate. Because of the scale of Net2, ultrasonic waves can reach other junctions which aids in increasing communications frequency. For a network the size of the Looping network, it as apparent that ultrasonic agents have to be in almost the exact same place to communicate effectively.

A final comment must address the infinite communication crossover in the Net2 network. Despite having no transmission time, the infinite behaviour is overtaken by the acoustic and ultrasonic methods as the network saturates. This illustrates the necessity of a disparity between

the individual beliefs of agents. In a larger network infinite communication is good as the agents are spread out and almost inspect their own areas, or follow a similar cycle out of sync for optimal effectiveness. However, in a small network the agents are condensed and each of them having the exact same memory can be detrimental as they become increasingly aware every pipe is being inspected at all times. The behaviour can then turn to random selection or agents following one another and making similar decisions. The effect of having at least some different beliefs is that not every agent will then make the same decision, and at least some semblance of the planned behaviour remains.

3.10. **Discussion.** The focus of this chapter was to detail the simulation platform presented by Parrot, C. et al. [44] and why it is an appropriate mechanism to model autonomous pipeline inspection. By creating a system that can take in real networks and generate point agents with limited hardware assumptions it has been able to give a low-cost insight into the challenges of subterranean inspection, particularly highlighting the communication difficulties faced in this particular environment. In addition, the platform has laid the foundation from which the rest of the thesis is built by creating a simulation capable of easy manipulation without interfering with the core mechanics of the system. This was exhibited in the adaptation of the simulation to include ultrasonic and acoustic communication methods, the transmission state and three new planning algorithms.

The chapter has been able to provide a preliminary investigation into the aptitude of multiple algorithms in a network. The behaviours presented each aimed to circumvent some pipeline specific issue, but most importantly provide different truly autonomous methods. Importantly, the scale of the number of robots gives an insight into network saturation and can hint toward an appropriate number of robots required for a specified level of inspection. One of the main outcomes of this thesis is to prove the feasibility of autonomous inspection and the simulation has shown that continuous inspection is, at least in theory, a possible and efficient solution.

With regards to the behaviours themselves, it was apparent that the benchmark implementation of the behaviours gave favour to the multiple CPP solution. However, as additional considerations were taken into account, in this instance communication, the other algorithms had opportunity to improve. Unfortunately the CPP is a rigid solution and fares increasingly worse the more considerations it must make. It is evident that an efficient communication method is beneficial to inspection performance and contributes to creating a robust swarm intelligence, as demonstrated by the normalised communication ranges in Figures 10 and 11. Similarly, the simulations in Figure 12 and 13 illustrate how a real application of signal processing is still able to achieve consistent inspection levels. This is important because, as the thesis progresses, it is apparent that autonomous inspection becomes more reliant on gathering and exchanging additional information to remain effective.

4. Implementing a Responsive Flow Intelligence.

4.1. **Introduction.** Water networks provide a unique obstacle to inspection robots, with dynamic flows hindering the movements of the robots. Not only does this physically limit the speed at which inspections can be completed but, given the already limited communication ranges between robots, reduces their ability to cooperate effectively. This results in higher times between inspection for pipes, increasing the windows in which cracks and blockages can form. A solution to this is to develop a responsive artificial intelligence, capable of analysing flow patterns in order to reduce the impact of strong flows on the robots. Additionally, strong flows have the potential to overcome an agent, pushing it back down pipes with the strong flow. At this point an agent can become lost, ceasing to make optimal decisions as it cannot localise itself to the memory of the network.

This chapter begins by illustrating the detrimental effects dynamic flow has on the planning algorithms, using the simulation. The chapter then adapts the simulation to enable strong flows to overwhelm an agent, putting them in a 'lost' state. Subsequently, a simple Simultaneous Localisation And Mapping algorithm is introduced that the agents can implement as they become lost to localise quickly. The algorithm requires only the knowledge of how many connected links there are at a junction to be effective.

This chapter goes on to present a learning algorithm based off of Gaussian Processes for Regression problems to predict flow rates. Incorporating a measure of flow intelligence allows the behaviours to mitigate the detrimental effects of flow, and the use of a *deep-learning* Bayesian method improves this process as time goes on. It is common for water companies to have a rough knowledge of the network flows, but there are often cases where flows are unknown or are not accurate enough to be meaningful. The two cases are considered — where robots have prior knowledge of the networks flow, and when they do not. Implementing a *supervised learning* algorithm helps circumvent these issues to provide a solution to flow.

The chapter concludes by briefly outlining a novel Flow-Based SLAM algorithm that uses additional information from the Gaussian Process to improve the rate at which the existing SLAM algorithm localises.

4.2. **Introducing Dynamic Flow.** Dynamic flow is undoubtedly the biggest obstacle autonomous robots face in the inspection process. To analyse their impact, the individual network's dynamic data can be implemented in a 24 hour flow cycle in the simulation. The required data is available in the EPANET export of a network that is read into the simulation, but unless specified in the user interface, has no influence on the agents. With the implementation of flow, the agents follow the same decision making algorithms as before; the simple difference is that the robot driving speed is now proportionally influenced by the flow.

Unfortunately the flow data itself is discrete and changes hourly, which can cause instant fluctuations as an agent is traversing a pipe. However, though not a discrete change, it is common for flow to suddenly fluctuate with the demands of human routine. For example, the collective early morning shower can cause a spike in flow demand, not too distant from a discrete leap. Similarly, though the flow data available in these simulations is repeated every 24 hours due to the availability of the data, this fortunately mirrors the monotony of the working week. In short, the flow data provided for these simulations is real and, though not ideal, a fair representation of flow demands and properties.

To explore the effects of dynamic flow, Figure 14 shows the Greedy Walk behaviour run on the Looping network both with and without flow, for a variety of communication ranges. The communication ranges have been set to percentages of the cumulative network size again as this allows the detrimental effects of flow to be isolated from the communication issues it compounds.

Figure 14 demonstrates a clear loss of performance when flow is introduced. The mean represents the overall speed at which agents are inspecting the network and in each instance the introduction of flow

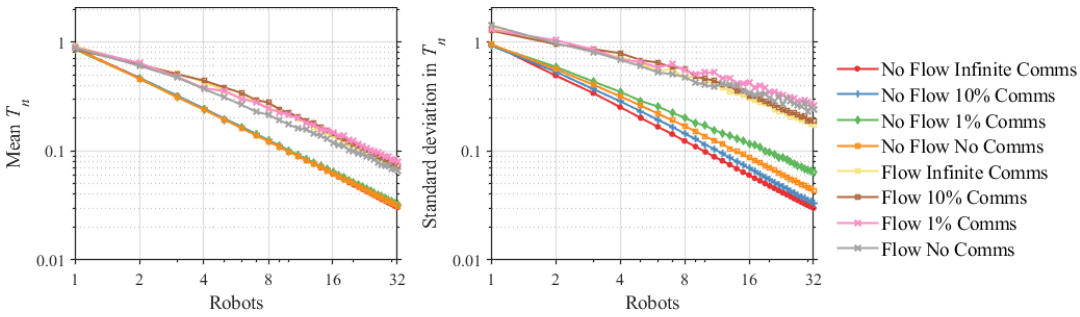

FIGURE 14. The effects of introducing flow to the Greedy Walk behaviour for the Looping network.

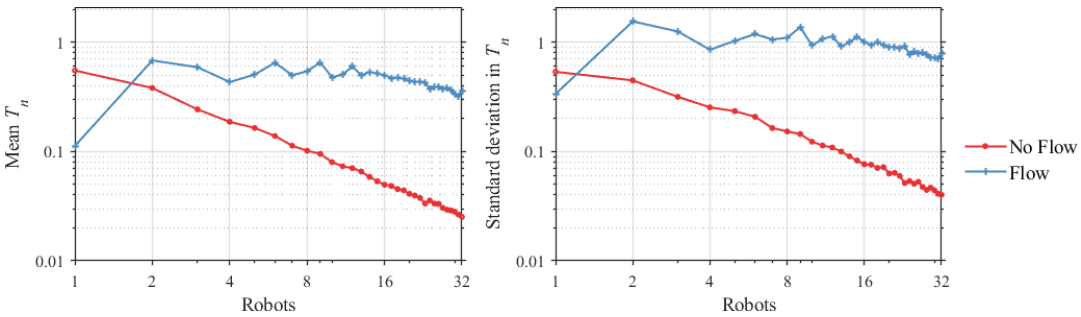

FIGURE 15. The effects of introducing flow to the CPP behaviour for the Looping network.

greatly reduces how mobile the agents are. The mean is increased simply because the agents are slower at traversing the network as a whole, and therefore reach new pipes for inspection much less frequently. In addition, Figure 14 highlights how the Greedy Walk behaviour specifically could be improved with the addition of flow; as the algorithm uses forward planning, so long as the flow is traversable, it will continue to push on despite the slow rate. It is conceivable that introducing a break function when the flow exceeds a certain value to reanalyse the future path and instead choose a path of lesser resistance could help improve the behaviour. Despite this, the behaviour is more fluid than the CPP solution, which is shown to struggle further with flow in Figure 15.

The CPP algorithm struggles so much when flow is introduced due to the algorithms stubborn nature. By concerning itself with a topologically optimal path, the algorithm fails to take precautions for dynamic flows and their effects on the agents. By splitting up the network, the algorithm is sure to eventually inspect all of its cluster at some point.

However, a cluster who's junctions are surrounded by strong flows will inevitably waste vast amounts of time getting to those junctions. In addition, the standard deviation sees a vast increase as areas with low flow are still able to function somewhat effectively. The contrast to those in high flow areas helps formulate this discrepancy.

In addition, the flow has a huge impact on the standard deviation of the inspection for the Greedy Walk. This is to be expected, as particular pipes can have consistently high flows, with extremely infrequent lulls in which an agent can traverse it. Without accounting for the flow it is difficult to then achieve similarly low standard deviation values, as the small windows in which an agent can enter a pipe are not being prioritised. As such, whole areas of a network can be cut off save for the rare moment an agent happened to being inspecting nearby during a lull. The Looping network particularly has topological obstacles, such as bottlenecks to large segments of the network that are compounded by an unusually high flow. Without any flow intelligence, a behaviours ability to navigate flow in a network is dependent on the topology of the network. Similarly, areas with low flow are likely to be inspected every time they are passed, which helps cement the disparity between the inspection frequencies.

The introduction of flow reaffirms the necessity for a behaviour to be adaptive — the Greedy Walk behaviour suffers but not in the devastating manner of the CPP algorithm. This can be done indirectly, with behaviours that frequently re-plan their paths, moving in tandem with the flow. For path planning algorithms, a more beneficial response is to actively assimilate flow data in the hopes of using it to plan around the flow.

4.3. **Localization.** Introducing flow also introduces another issue for robots in a network. As previously stated, in the instance that the flow becomes too powerful for an agent, overcoming a realistic moving speed, the agent will enter the sleep state, becoming immobile and conserving 'charge' before waking every 30 minutes to see if the flow has subsided enough for it to move again. However, it is not realistic to assume that the clamping mechanism can withstand any determinable force

imaginable. There is a likely scenario wherein the flow can become *too* powerful, overwhelming the clamped robot and pushing it back down the pipes with the flow. When this happens, it is important to assume the agent is aware of the strength of the flow, and the possibility it has been moved. In this instance, when the agent awakes it will enter a *lost* state, traversing aimlessly — that is, with no governing behaviour, until it enters the communication range of another agent. At this point the agent receives a broadcast and position of the broadcasting agent, and is assumed to be able to localise from that, returning to the previous behaviour.

Without a flow intelligence it is all too frequent that an agent enters a long pipe as the flow is traversable, only to be rendered immobile or pushed back by a discrete switch mid-way through the pipe. This is detrimental to the swarm inspection process, as once a robot becomes lost, not only will that single robot be unknowingly inspecting a less than optimal path, but it could potentially be communicating incorrect inspection and flow data to the rest of the swarm. To avoid this it is necessary to use the agents flow sensor to ascertain when the flow is too high, and will have pushed it back. For the purposes of the simulation, the agent enters a sleep behaviour when the flow velocity against it is equal to it's driving speed. Therefore the lost state is triggered when the flow velocity is over double the driving speed of the agents. Triggering the lost state prevents the agent from broadcasting information or inspecting efficiently until it has been 'found'. Therefore it is necessary to consider a *localisation* process that can quickly evaluate the surrounding area and inform the robot of it's location in order to quickly return to function.

Localisation algorithms are a subset of Simultaneous Localisation And Mapping algorithms and typically revolve around ascertaining a precise location given surrounding landmarks and features by increasing the certainty of a position as additional information is gathered. Given the monotony of water pipe networks, it is difficult to to distinguish distinct features to provide clarity. Though it is possible to analyse the cracks, leaks or wear of pipes to identify them, this requires high levels of hardware and an intelligent analyses system. Fortunately,

most water networks have a unique and random topological map. This makes topological localisation well suited to inspection robots, as a quick localisation process can be achieved simply by considering the topology of the network — in this case, the junctions of a network. By simply analysing the number of pipes at a junction and comparing them to the topology of the network, an efficient localisation algorithm can be implemented that uses little information and allows the robot to simultaneously inspect as it finds itself.

4.3.1. *No SLAM.* We first consider the case where there is no proper localisation algorithm as a benchmark. When an agent becomes lost, they must still be able to traverse the network in search of an agent. If the flow were to push an agent to an outlying pipe it could be a long time before another agent passes within communication range. In the meantime, the agent would be providing no discernible benefit to the swarm and would simply be prolonging the time before it can return to an inspection process. By continuously moving the agent improves it's chances of statistically finding another agent. However, when lost, an agent has no local network knowledge. Therefore, it must rely solely on what it can see which in this instance, is either the pipe it is moving forwards in, or the intersection it reaches at the end. Without a localisation algorithm, the lost behaviour defaults to an agent picking links at random when faced with a choice at a junction.

Though not specifically a behaviour, the simulation requires a governing behaviour for the agent to move around a map. As such, the NoSLAM $SelectNextLink$ function necessary for movement simply returns a random $LinkIndex$ of those links connected to the current junction. The concept is rudimentary and serves only to fill the absence of a proper behaviour.

4.3.2. *SLAM Algorithm.* Underground pipeline networks give the agents a big advantage when it comes to localisation — their topology. Topology based SLAM algorithms can face challenges when they encounter symmetrical or cyclic problems as repeated identical topological features provide no real information that can be used to localise. Because

pipeline networks are typically designed in response to demand, particularly surrounding housing expansion, there is little natural symmetry. Similarly, there is though cycles exist, they are rarely confined and exist within surrounding clusters with their own features. As such it is possible to view pipeline networks as graphs, with their topologies and accompanying sub-structures appearing uniquely complex. Without symmetry or cycles, chaining subsequent vertex degrees begins to form a unique topological landmark, much like a bar code. This provides opportunities for reliable localisation with the only assumption being that at an intersection, a robot is able to count the number of pipes connected to it.

The SLAM algorithm itself is inspired by the work of Worley, R. and Anderson, S.[59] which presents a Hidden Markov Model localisation model that uses the number of links at a junction to influence state transitions and likelihood models to achieve localisation. Because of the information available to the agents in the simulation environment, it is beneficial to make use of the junction information in a similar manner, except in a deterministic fashion. Dealing only in absolutes is sufficient in this instance for the agents to localise quicker and helps reduce the computational power of the simulations.

The algorithm itself utilises the stored map and junction information to correlate to the current junction it is at. As the agent continues through the SLAM process, it can store the index of each link it chooses for future reference. An important distinction to be made is though the simulation assigns a connected link an index with relation to the junction, this is unavailable to the lost agents. Because they are lost, they do not know the direction they arrive at the junction from, so cannot localise the 'correct' index assigned by the simulation. The index is a matter of perspective, however, and by storing the index of each link from the direction the agent arrived at the junction, upon conclusion of the SLAM algorithm it is possible to reconstruct the path taken from when the agent initially became lost. Not only is a perspective based index important for the SLAM behaviour, it also allows the agent to continue inspecting as it progresses through the SLAM process. As the agents can reconstruct their entire path, simply

storing the local TSLI values allows them to update their memory upon conclusion of the algorithm. Though while actively localising the agent is unable to broadcast reliable TSLI values, upon completion it can still contribute to the swarm memory of the system.

Though recording the index entries remains important for reconstructing the SLAM path at the end, the algorithm itself is also concerned with the degree of the junctions it passes. Once lost, the algorithm begins by having the agent count the degree of the junction it is initially at, and removing any junctions who's degree does not match from a potential starting junctions list. It then chooses a link who's flow it can traverse and follows that path until it reaches the next junction. The agent counts again, and removes any junction from the starting list who is not connected to a junction of a matching degree. This concludes the initial step — once the agent reaches the second junction, it has a bearing and begins the second phase.

Having a bearing means the agent is able to concretely assign a perspective index to each link as it arrives at a junction based off the link it has arrived from. For example, upon arrival to the junction it would assign the leftmost link the index 0. This allows the algorithm to store a temporary local map of the link taken at a junction. The process is iterated, with the the link index and junction degree continuously recorded. At the arrival of each junction, the agent is then able to examine the initial starting positions list and ask which are connected to links which, when they follow the same index path, tick off the same vertex degrees along the way. This rapidly reduces the list of potential starting positions as, though the very first link index is unknown, the connected junctions have a strict path and therefore criteria to follow. Once the starting positions list has reached one, the process is complete. The agent is able to work out the missing link — the second junction, and from there reconstruct the entire path from the only starting position back to the current position.

Algorithm 13 details the pseudocode for the SLAM algorithm:

Algorithm 13 SLAM Algorithm

Require: $CurrentJunction \neq null$
for all **do***Junctions*
  if *Junction* degree = *CurrentJunction* degree **then**
    PotentialStartingJunctions add *Junction*
  end if
end for
Choose random *Link*
for all *PotentialStartingJunctions* **do**
  $JunctionMatch \leftarrow false$
  for all *PotentialStartingJunctions* connected *Junctions* **do**
    if *ConnectedJunction* degree = *CurrentJunction* degree **then**
      $JunctionMatch \leftarrow true$
    end if
  end for
  if $JunctionMatch = false$ **then**
    Remove from *PotentialStartingJunctions*
  end if
end for ▷ End of Initial Phase
while $PotentialStartingJunctions > 1$
Choose random *Link*
IndexList add *IndexOfLink*
At *NextJunction* add *NextJunction* degree to *JunctionDegreeList*
for all *PotentialStartingJunctions* **do**
  $JunctionMatch \leftarrow true$
  for all *ConnectedJunctions* **do**
    for IndexList **do** size
      Iterate through *IndexList*
      if *IterationJunction* degree $\neq$ *JunctionDegreeList* **then**
        $JunctionMatch \leftarrow false$
      end if
    end for
  end for
  if $JunctionMatch \leftarrow false$ **then**
    Remove from *PotentialStartingJunctions*
  end if
end for
end while
for all *OnlyStartingJunction ConnectedJunctions* **do**
  $JunctionMatch \leftarrow true$
  Iterate through *IndexList*
  if *IterationJunction* degree $\neq$ *JunctionDegreeList* **then**
    $JunctionMatch \leftarrow false$
  end if
  if $JunctionMatch = true$ **then**
    $JunctionTwo = ConnectedJunction$
  end if
end for

4.4. **SLAM simulations.** In order to examine the effectiveness of the SLAM algorithm it has been implemented in two behaviours. The first, Stigmergy, is demonstrated in Figures 16 and 17 with ranges constituting 1 and 10% of the network respectively. Additionally the algorithm has been added to the mCPP solution in Figure 18. The two behaviours represent opposite sides of the autonomous spectrum, with Stigmergy effective inspecting anywhere and the CPP solution strictly designated to one cluster. As such, the CPP requires an additional couple of tweaks for the condition in which it localises outside of it's own cluster. Given the path it plans is dependent on the cluster passed in, it is necessary to return it to it's own. As such, it simply implements Dijkstra's algorithm to work out which of it's cluster junctions is closest, before returning and beginning it's cycle again. In a similar fashion, if the CPP agent wanders into another cluster, either due to the flow or the localisation algorithm, it is necessary to be able to localise from the communication of other agents. Otherwise, in the No SLAM case, there would be no means by which the behaviour could localise. As such the CPP agents are able to communicate their locations alone, in the range of 1% of the network.

The Looping network has been selected to demonstrate the efficiency of the SLAM algorithm as, numerous junctions of different degrees and complex topological map, it would be the most difficult to localise in.

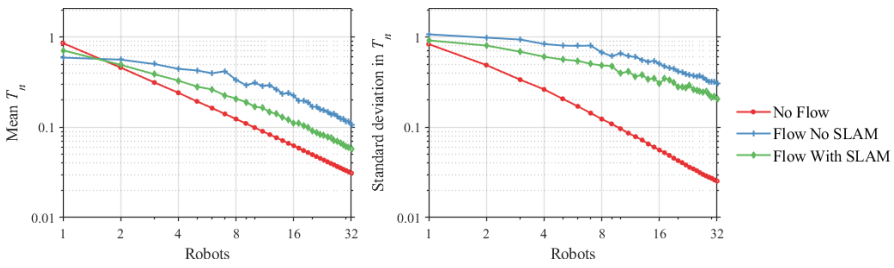

FIGURE 16. The effects of implementing a SLAM algorithm with Stigmergy and a 1% communication range in the Looping network.

When analysing the efficiency of the SLAM algorithm, the most notable feature of the simulation runs is the SLAM application in Figure 18. While the Stigmergy's TBI values drastically drop when using the

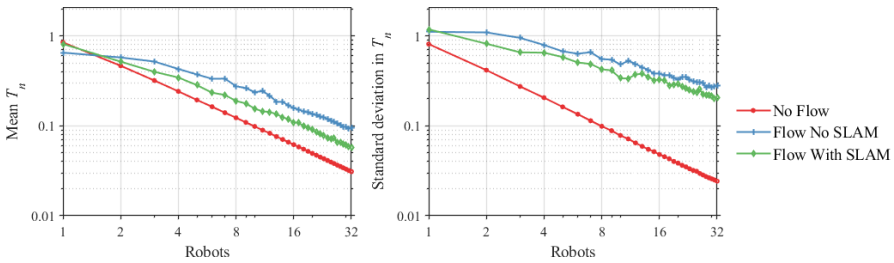

FIGURE 17. The effects of implementing a SLAM algorithm with Stigmergy and a 10% communication range in the Looping network.

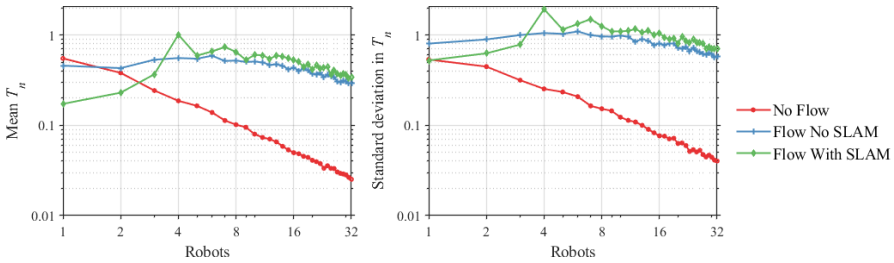

FIGURE 18. The effects of implementing a SLAM algorithm with the mCPP in the Looping network.

SLAM algorithm over random behaviour, the CPP actually finds benefit in not localising at all. As the SLAM behaviour inevitably localises the agent quicker than a No SLAM approach, the only conclusion that can be drawn is when flow is introduced, the random No SLAM behaviour is actually a more adequate inspection process than the CPP solution. This is unsurprising when Figure 15 is recalled — the ability to move fluidly around the network, only piecing together the TSLI values once localised, is still a better approach than localising quickly to return to stagnating in a strong flow.

The Stigmergy then provides a better insight into the efficiency of the SLAM algorithm. As is evident, for both communication ranges, the SLAM algorithm returns a better TBI. This is because of the speed at which it can localise to return to the Stigmergy behaviour — a functionally better method of inspection. Interestingly there is very little difference between the two communication ranges and their SLAM line. As part of the SLAM algorithm is localising once within communication range of another agent, it would be conventional to see a larger deviation based on the communication range. This indicates that the SLAM

algorithm frequently self-localises without the assistance of other agents in the system.

The topology based SLAM algorithm must therefore be considered an effective approach to solving the problem of localisation in underground networks. Not only is it able to make use of the simplest available information to localise effectively, but it does so quickly and in a manner where the agent is able to store TSLI values so inspection time is not wasted. Given that in large networks the acoustic and ultrasonic communication struggles more, it is also beneficial that there is little reliance on communication range to be effective. In tandem with an appropriate flow intelligence, the algorithm can help minimise the wasted inspection time introduced by dynamic flows.

4.5. **A Reactive Flow Intelligence.** Before developing a flow intelligence it is necessary to examine the information available to the robots, from which they can build a portrayal of the network. Collating live data as the robots naturally traverse the network can be used to provide a clearer indication of flow patterns and is particularly useful for *unknown* networks. Working with the flow will help navigate the softer flows, moving quicker and lowering the mean TBI, and is crucial in keeping the standard deviation TBI low in outlying areas. Additionally, as with the simulation data itself, many water companies have accessible, if not complete, flow data, which readings can reaffirm or contradict depending on its validity. It is therefore desirable that the flow intelligence be able to build a clear understanding of the flow of the network regardless of the data it begins with.

In a similar sense, it is apparent the robots must have a basic understanding of the topology of the network. Specifically, if a robot is to make use of the flow patterns of the network, it must be able to differentiate between pipes. Not only does this enable the robot to traverse the network more efficiently, but it allows the robot to take accurate readings to further improve the data. It is sufficient to assume the robots have knowledge of the degree of each junction in a network and the respective connections to other junctions. This enables the robots

to distinguish between individual pipes and consequently, the flow data specific to that pipe.

The presented flow intelligence revolves around modelling individual flow patterns for each pipe in a network. Once a robot is able to differentiate between pipes, it can begin to learn the individual flow patterns, using this information to negate the impact of the flow. The most common flow fluctuations that must be navigated mirror the demand for water — for example, a sharp increase in the flow strengths in the early mornings as people shower before work. Similarly, there is a long lull through the night as people sleep, resulting in a low flow. The most appropriate interpretation of the flow data then is to model it over a 24 hour cycle as this best represents the demands on the network, and will thus prove more beneficiary as the robots learn the flow patterns.

The proposed flow model uses Gaussian Process (GP) methods to define the relationship between the flow strength and the time of day. Gaussian regression methods are a stochastic process used in supervised learning which focus on generalizing probability distributions to a function. Using training sets it is possible to fit a continuous model to discrete data. The principle behind the presented flow intelligence system is to partition the network into individual pipes and fit a regression model to each, representing the flow cycle throughout a 24 hour day. As the robots traverse the network, they can use the regression model to predict the flow in each pipe, whilst simultaneously collecting *training data* that can improve the model. The most important property of the process is that, by implementing a regression model, the Gaussian process can model the flow as a *continuous* function. This is an important stipulation as it contradicts the discrete data provided by water companies for this thesis. The data provided is an hourly approximation of the flow in individual pipes and is mirrored as such in the simulation. In real life however, spikes in flow are not instantaneous and on the hour but rather fluctuate gradually. As robots traverse the network, a continuous models gradient will better describe the strength of the flow than a discrete jump.

4.6. **The Gaussian Process.** Gaussian processes are a non-parametric, Bayesian approach to regression that assume that every finite collection of random variables in a stochastic process have a multivariate normal distribution. The process makes use of prior distribution, typically assumed to be Gaussian, in tandem with a training set to calculate a posterior distribution using Bayes theorem. From this, a *predictive distribution* can be calculated, which can be used to obtain a point prediction for unknown data. This gives the added benefit that each prediction carries a calculable uncertainty measurement, as the prediction is calculated using the mean and variance of a Gaussian distribution.

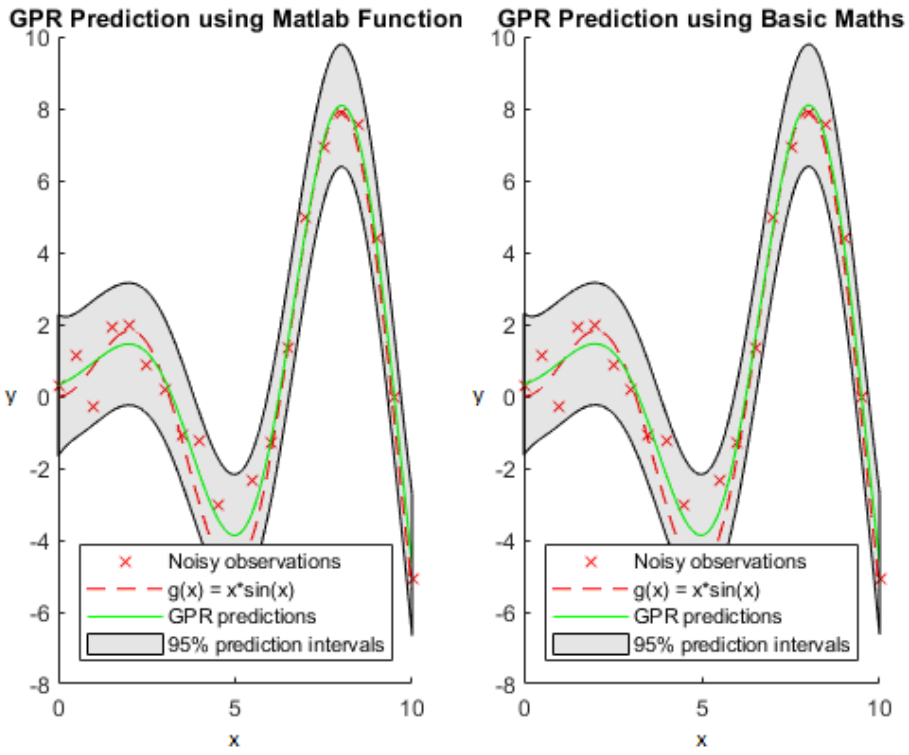

FIGURE 19. A comparison of the predictive distribution obtained via Gaussian processes and the distribution for the training data.

Figure 19 illustrates an application of Gaussian processes. The red dotted line represents the distribution with which the 20 points of training data was created, $g(x) = xsin(x)$, and is therefore the target replica function. The green line represents the predictive distribution, with the grey shaded region representing the expected variance. By inferring a

probability distribution, the predictive distribution is able to get remarkably close to the target function for such a small sample size, with suspect areas of few data points represented by an increased variance region.

Removing the requirement to learn exact data as other machine learning algorithms do makes the process efficient with small data sets, as well as reducing the computational power required for inference. This is particularly beneficial to pipeline inspection when there is limited prior information about the flow of the network as it allows the robots to quickly learn a fundamental pattern, where other methods would take longer to provide a beneficial intelligence. In addition, pipes with strong flows have infrequent opportunities to be inspected, so making use of limited data sets is essential in modelling the harsher flows too.

4.6.1. *Creating the Predictive Distribution.* Figure 19 details two similar predictive distributions, one created with MATLAB's in-built GPR function and a simpler, version which makes various noise assumptions for computational ease. Though MATLAB offers an in-built GPR function, it is desirable that the custom simulation be self-contained to better represent the computational challenges a robot in a network would face. It is unlikely that having a robot repeatedly return to one of the limited contact points with the surface every time it was necessary to update the GPR prediction would be very efficient. Figure 19 demonstrates that with relatively few data points it is possible to recreate an extremely close GPR match by eliminating some noise requirements and using 'basic' maths. Implementing this in the simulation allows the agents to sufficiently predict the flow of the network to make near-optimal decisions, without having to compromise the routes they take to outsource the computations.

The equations detailed below are presented by Rasmussen, C. E. and Williams, C. K. I in their work 'Gaussian Processes for Regression' [46].

To create a predictive distribution it is intuitive to take the *weight-space* view, a Bayesian treatment of the linear model. The development of a standard regression model in tandem with Bayes rule and some

Gaussian noise assumptions is sufficient to create an effective predictive model. The following demonstrates how a predictive distribution is built.

The Gaussian process begins with a Bayesian analysis of the standard linear regression model;

$$f(\mathbf{x}) = \mathbf{x}^\intercal \mathbf{w}, \quad y = f(\mathbf{x}) + \epsilon \tag{1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the vector of parameters of the linear model, f and y are the function value and observed value respectively. The difference in y and the function values $f(\mathbf{x})$ differ by additive noise which is assumed an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2:

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2). \tag{2}$$

To calculate the posterior distribution, we must employ Bayes Rule:

$$posterior = \frac{likelihood \times prior}{marginallikelihood}, \quad p(\mathbf{w}|\mathbf{y}, X) = \frac{p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|X)} \tag{3}$$

where X is the matrix of training inputs — the design matrix. The prior must express the beliefs about the parameters before the observations are examined, and as such a simple zero mean Gaussian prior with covariance matrix Σ_p will suffice:

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p). \tag{4}$$

The likelihood is derived directly from the noise assumption and can be factored over cases in the training set to reduce the model to

$$p(\mathbf{y}|X, \mathbf{w}) = \mathcal{N}(X^\intercal \mathbf{w}, \sigma_n^2 I). \tag{5}$$

The normalizing constant, or marginal likelihood, is independent of the weights and is given by

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})d\mathbf{w}. \tag{6}$$

Completing the square we recognise that the posterior distribution is a Gaussian distribution with mean $\tilde{\mathbf{w}}$ and covariance matrix A^{-1}:

$$p(\mathbf{w}|X,\mathbf{y}) \sim \mathcal{N}(\tilde{\mathbf{w}} = \frac{1}{\sigma_n^2}A^{-1}X\mathbf{y}, A^{-1}) \tag{7}$$

where $A = \sigma_n^{-2}XX^\intercal + \Sigma_p^{-1}$.

To make predictions using the posterior probability, all possible parameter values are averaged before being weighted by their posterior probability. Therefore, the final predictive distribution is given by:

$$\begin{aligned} p(f_*|\mathbf{x}_*, X, \mathbf{y}) &= \int p(f_*|\mathbf{x}_*, \mathbf{w})p(\mathbf{w}|X,\mathbf{y})d\mathbf{w} \\ &= \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^\intercal A^{-1}X\mathbf{y}, \mathbf{x}_*^\intercal A^{-1}\mathbf{x}_*). \end{aligned} \tag{8}$$

4.6.2. *Function Space and the Predictive Function.* Though the weight-space view allows a direct derivation for the predictive distribution, the *function-space* view considers inference in function space to predict expected values. The calculations for mean, covariance and expected values required for the robots to understand the flow intelligently are much more computationally efficient due to their reliance on the *kernel function.* The function-space view works under the definition that a Gaussian process is a collection of random variables, any finite number of which have a joint distribution. This means a process is completely specified by its mean function and covariance function and therefore

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{9}$$

where $m(\mathbf{x}) = E[f(\mathbf{x})]$ and $k(\mathbf{x}, \mathbf{x}') = E[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$. Specifically, the covariance function used is the *squared exponential* covariance function, defined as follows:

$$k(\mathbf{x}_p, \mathbf{x}_q) = exp(-\frac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2). \tag{10}$$

In this particular instance, it is possible to assume that the observations obtained by the simulation will be noise free — they are in fact simulated values, representing discrete time and flow. However, in real life the robots are likely to have noise surrounding their observations,

ϵ which we assume is independently, identically distributed Gaussian with variance σ_n^2. As such, the prior on noisy observations is represented by

$$cov(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq} or \quad cov(\mathbf{y}) = K(X, X) + \sigma_n^2 I \tag{11}$$

where $K(X, X_*)$ denotes the $n \times n_*$ matrix of covariances evaluated at all pairs of training and test points. With the noise term it is now possible to examine the joint distribution of the target values and function values under the prior and, in deriving the conditional distribution, the predictive equations can finally be simplified to:

$$\tilde{f}_* = \mathbf{k}_*^\intercal (K + \sigma_n^2 I)^{-1} \mathbf{y} \tag{12}$$

$$Var[f_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\intercal (K + \sigma_n^2 I)^{-1} \mathbf{k}_* \tag{13}$$

The above equations give the predicted mean and variance for an unknown point under the predictive function. By consistently updating the training data with flow data and remodeling the predictive distribution, robots are able to make more accurate predictions as the variance drops. This *pinching* effect is illustrated where the variance surrounding training data has reduced, though it remains large in areas with little training data. As the Gaussian process manipulates a distribution as opposed to exact data, the computational complexity to calculate the mean and variance values is low, even as the volume of test data increases. Therefore the model simply improves as time progresses, with little cost to the robots.

4.7. **Simulation Implementation.** As aforementioned, the flow in the network is represented by dividing the network into individual pipes and using a Gaussian process to define the pipes 24 hour flow cycle. From this, the robots are better able to calculate the time it takes to traverse a pipe, and can subsequently plan the most efficient paths to reduce the average Time Between Inspection. The network of the flow can be considered as n separate Gaussian processes, as illustrated

in Figure 21 which shows the EPAnet representation of Net2's upper pipes in Figure 20.

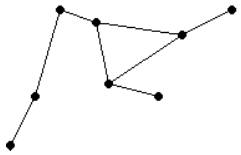

FIGURE 20. The upper part of Net2.

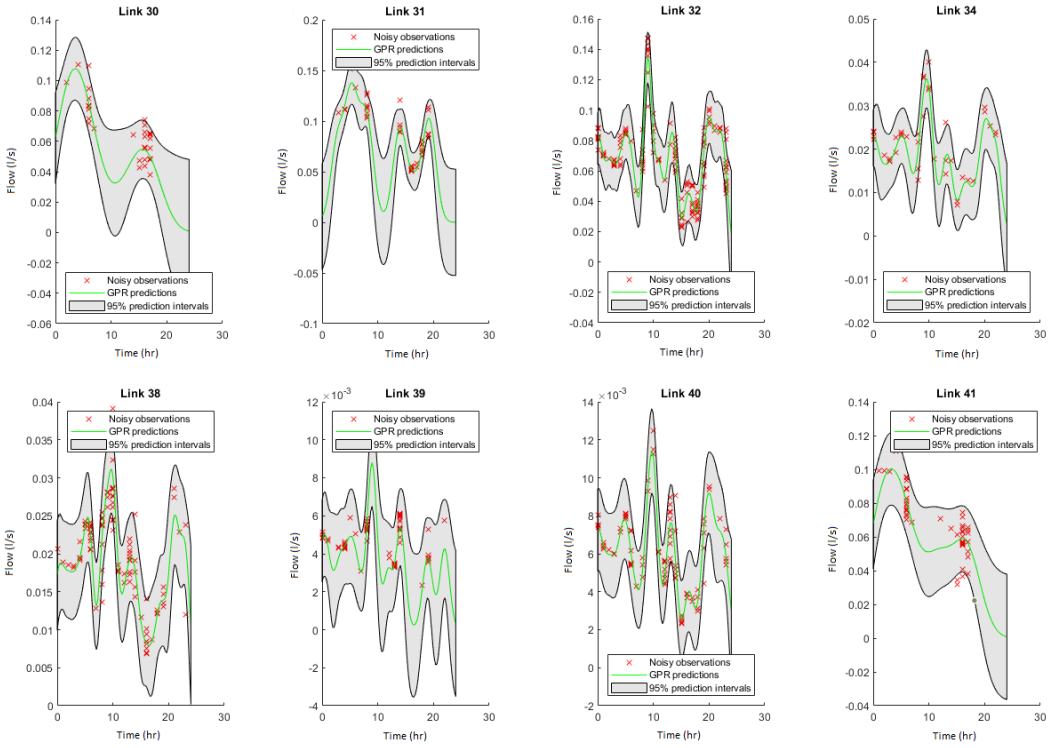

FIGURE 21. The flow of the upper part of Net2 represented as a group of Gaussian priors.

Figure 21 was created with three months worth of inspection data from the simulation. Obtaining an appropriate amount of training data is fundamental to the efficiency of the model and repeated recomputations of the model are likely to slow the robots considerably. It is already assumed that robots in the network have a means of sharing their inspection data, either by passing a communication point in the network and uploading to a cloud or by exchanging as they pass one another. It is therefore realistic to consider the robots also sharing the flow data they have measured. Upon doing so, the models can be

updated with the new training data, before the new predictive distribution is relayed for consideration. The mean and variance flows are then equally calculable from Equations 12 and 13 respectively.

The flow intelligence implementation in the simulation follows a similar format to that of the TSLI memory. Upon entry to a pipe, an agent will record the time and the flow of the pipe. The time can be reversed engineered to its 24 hour time using a 24 hour modulo operation. Throughout the simulation an agent will continuously store the data point and upon completion will 'upload' this batch of new training data to 'the cloud'. At this point, the data is added to the existing pool of training data and the model recalculated. The next simulation is able to work off the improved model, as it itself collects new data and as such, each simulation sequentially improves, lowering the average TBI values.

Given the typically short simulation time (28 days), for convenience the simulation updates the training data at the end of a simulation, as this allows a comparison of complete simulations with better or worse training sets. Alternatively, it is possible to introduce a class that can export the training data for a MATLAB reconstruction of the predictive distribution, though due to its continuous nature, the calculations of expected point values must be done within the simulation; the class must contain the necessary equations to calculate the expected value of the flow, which the agents call as they calculate their paths.

Figures 22, 23 and 24 illustrate the development of the Gaussian model of Pipe 38 in Net2 over three months of training data. Each data point represents a the time in a 24 hour cycle it was recorded and the flow of the link at that time. Given each month has 50 data points, it is remarkable how similar to the third month the first month is able to get. This illustrates the efficiency of the model in working with small datasets. Similarly, we see the training data mature in the second month, as it becomes almost indistinguishable from the third months data, despite a 50 point separation.

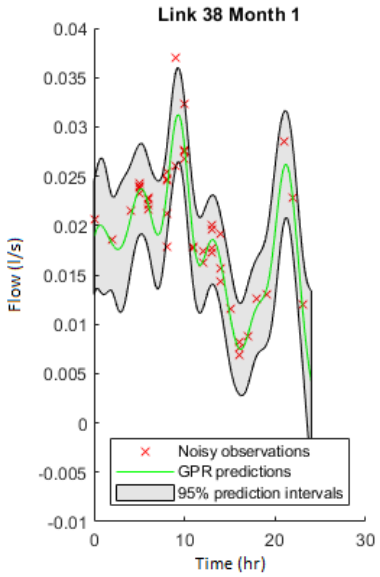

FIGURE 22. The GPR model obtained within the simulation for pipe 38 using one months worth of collected data.

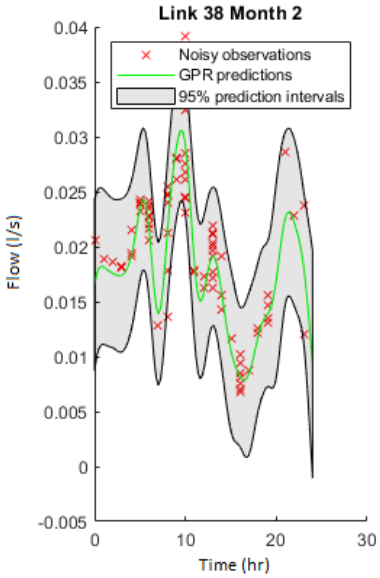

FIGURE 23. The GPR model obtained within the simulation for pipe 38 using two months worth of collected data.

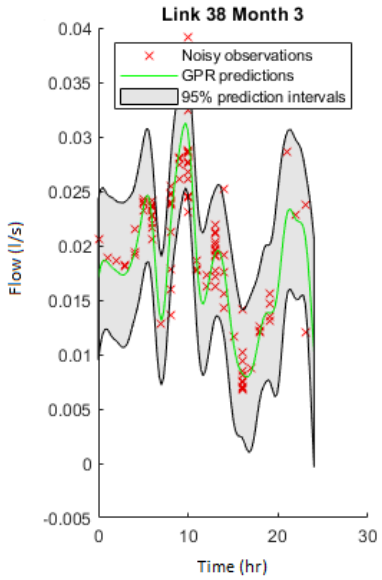

FIGURE 24. The GPR model obtained within the simulation for pipe 38 using three months worth of collected data.

4.8. **Flow Intelligence Simulations.** Real life implementation of flow intelligence demands a Gaussian Process Regression due to its low computational cost and ability to interpret small datasets so accurately. In addition, GPs have the ability to model continuous data, which flow readings outside of the simulation will form. However, the simulation data is discrete and cyclic. Additionally the simulation platform can be adapted to provide the agents with perfect data which would sufficiently replicate mature training data. The purpose of these simulations are to ascertain how flow knowledge can influence the inspection process. As such, for a simulation based approach, it is beneficial to work with a different model.

Given that the flow data is a repeated 24 hour cycle, the flow intelligence records the flow in the same manner a Gaussian Process would. The time is recorded in milliseconds mod 24 hours and each pipe is still represented by an individual flow memory of these recordings. However, when it comes to estimating a future flow value, if an agent has already inspected that pipe at the specified time, it is aware of the flow rate with certainty.

The model creates a list of each pipe and a 24 hour window. As it inspects the network, the model fills with certain flow data for future reference. As the agent plans paths and wishes to predict a future flow rate, it can cross check the pipes record. If a value is present at the necessary time, it is assigned as the predicted future flow. However, if no value is present, the agent seeks the nearest values either side of the time in question. If they are present, the agent will create a weighted average based on the time it is closest to, and use this to calculate a weighted difference between the pair. From here, an expected value is assigned.

This method allows the simulation to record data in the same manner as a Gaussian Process would. By iteratively filling out the timesteps of each pipe, it is able to assimilate to perfect knowledge in a similar manner to the GP model. The advantage remains that once a perfect copy of the flow has been established, longer simulations can be carried out that are similar to a Gaussian model with *mature* datasets. By using this model, the flow intelligence simulations in Figure 25 were run for the Net2 network spanning a month, three months, six months and a year.

The simulations were run using the Greedy Walk behaviour. With the exclusion of the Ad Hoc behaviour, the Greedy Walk is the only behaviour that can make use of the additional flow intelligence. The Stigmergy behaviour is solely concerned with the immediate flow which it already has access to, and as such has no need for the additional intelligence. In addition, the CPP is incapable of calculating a path outside of the topological optimum. As such, the intelligence was added to the Greedy Walk and ran for a year to demonstrate the effects of accumulating sufficient data. Net2 was chosen due to its volatile flow.

The simulation results in Figure 25 are initially extremely surprising. The No Intelligence case appears to closely resemble the flow intelligence when it has had One Months worth of training data, both in terms of mean and standard deviation. This is to be expected — Figure 22 has already demonstrated that the training data after a month has not yet 'matured', despite forming an effective model. Therefore

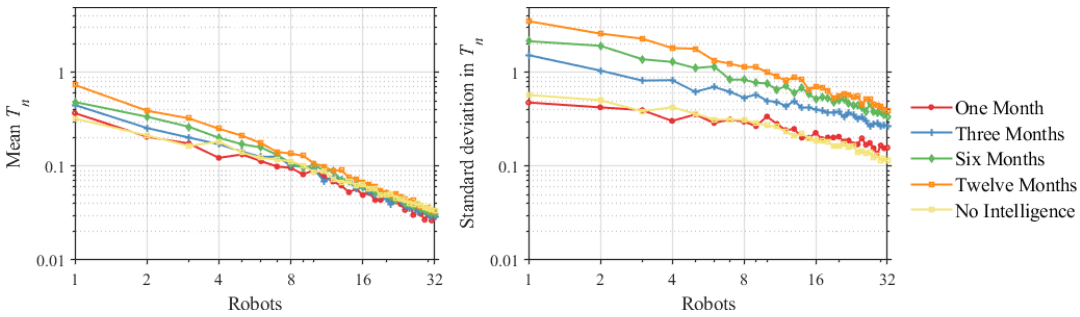

FIGURE 25. The effects of flow intelligence in the Greedy Walk behaviour over the course of a year for Net2.

it is unsurprising that the No Intelligence and One Month case should match as the model is still adjusting. However, the flow intelligence results then seem to follow a trend that indicates a relationship between increased training set size and worse means and standard deviation values. As the duration of the simulation increases, the mean TBI values slowly rise. In addition there is a drastic surge in the standard deviations of the TBI values.

Contrary to what these results initially indicate, it is apparent that the flow intelligence is effective. This is indicated by the discrepancy between the rise in mean values and the astronomical rise in standard deviation values. The standard deviation surge demonstrates that the agents have learnt the flow of the network, and use it to continuously traverse the *path of least resistance*. This explains why the mean value also rises — from a mathematical standpoint, continuously inspecting the same pipes does less to reduce the mean TBI than the aging pipes do to raise it. As such, there is a slow rise in mean value the longer the simulation progresses. It is likely the case that proportionally the 12 month intelligence inspects more pipes a month than the single month one, they just happen to be the same set of pipes along an easily traversable flow path. This compounds the considerable rise in standard deviation, as not only does the behaviour have a tendency to avoid in-need pipes, but it then actively seeks easy pipes to inspect.

The flow intelligence has been undeniably effective, as the standard deviations in Figure 25 are testament to. What is abundantly clear

however, is that a different application is necessary. Though the Greedy Walk has some merit in *adapting* to flow and communication types, it clearly is not a behaviour designed to *use* the flow. In learning the network the behaviour is able to exploit it's own greedy nature, to the detriment of high flow pipes. It was hoped that by implementing the Greedy Walk with a time based planning duration that additional flow knowledge would simply improve the amount of links it could reach within it's allotted time, by extension more frequently inspecting in-need pipes.

As a flow modelling technique, modelling the simulation flow in a discrete, learning manner has been effective. Similarly, a real world implementation of the Gaussian Process Regression model is likely to yield increasingly similar results with continuous data as the training set matures. As the models grow in depth it would benefit from a behaviour that is able to utilise the flow more intelligently, to ensure a fairer distribution of inspection. An alternative solution is proposed in Chapter 5, wherein the estimated flow velocity helps form part of Multi-Objective Genetic Algorithm metric that is weighted with different combinations to ensure an impartial and efficient inspection.

4.9. **Flow SLAM Algorithm.** The Flow SLAM algorithm is a potential extension of the SLAM algorithm detailed in Section 4.3 that uses the acquired flow data from the inspection process to localise quicker. Though the SLAM algorithm was efficient in the Looping network, there are real and artificial networks in which it would undoubtedly perform worse. As aforementioned, symmetric and cyclic topologies can introduce loops to the algorithm. To help counteract this, it is proposed that once the training data has matured, the expected flow of a pipe can be used to discount it in the localisation process. Using the information acquired from an existing metric would allow the SLAM algorithm to more quickly distinguish between pipes, as a further variable separates the pipes. Similarly, any computational expense has already been calculated in the construction of the initial GPR model.

The agents in a network have internal clocks to record accurate inspection times and flow readings. Therefore, as an agent is localising, it is

possible to determine the current flow of the link it is in and use that as an *identifying* trait, much like the number of links at a junction. If an agent measures the flow at it's current link, it can cross reference it with any link with sufficient training data in the network for the specific 24 hour time. If the current flow matches with a links expected flow, or within the realms of the Gaussian Process standard deviation, it can be considered. However, if the current flow is far outside the realms of the expected flow of the link in question, it can be safely discarded.

Though this is a theoretical algorithm with no basis for discussion, it is considered for any Future Work extending from this thesis in Section 6.3. If the SLAM algorithm and a Gaussian Process are implemented in a real life application of the system, it stands to reason that an additional step introducing readily available information might increase the efficiency of the SLAM algorithm. Given the speed with which the training data was collated in Figure 25, it is likely the data would quickly assimilate to an accurate model to assist with SLAM decisions with certainty.

4.10. **Discussion.** This Chapter aimed to demonstrate the unique issues that dynamic flows create when introduced into pipeline networks. The simple process of slowing an agent is sufficient enough to both lower the frequency of inspection, but also actively prevent specific pipes from being able to be inspected. In addition, the flow creates numerous issues for planning algorithms specifically, as seen with the huge detrimental impact on the CPP solution. The CPP serves as an example of a theoretically optimal approach unable to adapt to the sheer displacement of dynamic flows.

The introduction of overwhelming flow and a lost state also highlighted the necessity for a localisation algorithm in real applications. The presented SLAM algorithm is able to dampen the impact of strong flow by creating a quick localisation process. In addition, the algorithm relies solely on the topology of the network to localise, which aides in reducing the reliance on a strong communication range for this purpose. In spite of this, the performance of the SLAM algorithm does little to

increase the performance of the actual behaviours and an active flow intelligence is required.

The active intelligence is designed based on Gaussian Process Regression models and their ability to form complex models off little data. Given the low frequency with which certain pipes in the network can be inspected, this makes GPRs an ideal modelling platform for dynamic flows. Specifically, by dividing the network into individual pipes with their own flow history, a flow model of the entire network was constructed with the focus of recording and communicating this data. It is hoped that by providing agents with an advanced knowledge of the flow data, it might be possible to circumvent the effects of the flow and increase the raw frequency of inspections in a network. Though the models were efficient at creating a precise representation of the pipes, the Greedy Walk behaviour was unable to take advantage of the increased level of knowledge in the desired manner. A behaviour should be considered that could make active use of the flow data to seek out in need pipes to decrease the inspection discrepancies.

Finally, the presentation of a flow based SLAM algorithm seeks to capitalise on the success of both the SLAM and Gaussian Process efficiency. It is hoped that the two together can improve the speed at which an agent can localise by creating an additional variable with which other pipes can be distinguished.

5. Creating a Solution to Autonomous Power.

To continuously inspect underground networks of the size of water network, it is necessary for the robots to be *untethered* — that is, not connected to a power source. Not only does this allow the robots to travel past the range a tether might hold them to, but it eliminates the need for manual interaction. Additionally, the physical pipeline environment varies, and can often be rugged and sharp on the ground. Severing a power cord renders the agent stuck and unable to be retrieved non-invasively. It is therefore ideal that the robots in a pipeline network are untethered, which itself introduces issues.

Given the necessarily small size of the robots, fitting in pipes as small as 30cm, they are unable to hold a large charge. In addition, the agents are continuously inspecting, often against strong flows that hinder their movement, and are operating sensing and communication equipment. As such, the robots batteries quickly drain, rendering them unable to continue inspecting until they have recharged.

The Chapter begins by detailing the implementation of the power system and two charge states into the simulation. The Hydrant charge state makes use of specified junction ends in a network that are assigned as hydrants for a quick influx of charge. On the other hand, the Turbine state is based on the assumption that the robots in a network can clamp themselves to the wall still, this time with a small hydro-electric turbine that converts strong flow into charge. The process takes longer than the Hydrant charge, but is able to clamp anywhere.

This Chapter goes on to present a Multi-Objective Genetic Algorithm that aims to find a balance between the robots desire to continuously be inspecting, and the necessary requirement of frequently charging. Multi-Objective algorithms aim to find a mutually optimal *weighting* assigned to the metrics in consideration that together return the highest *Pareto optimal* value. A Pareto optimal set is a set of weightings where no weighting can be improved without worsening the performance of another metric[30]. By assigning weightings to the TSLI gain of a

decision, and the expected charge that decision will expend, the Genetic Algorithm component is able to find a Pareto optimal balance.

5.1. **Implementing the Power System.** In order to implement the power system, the agent class in the simulation was given an additional variable — charge. Without flow, the charge decreases at a steady rate as the agent traverses the network. Each agents power is initialised to 100% which has enough charge for eight hours of movement. When flow is introduced, the power decrease is calculated in another manner. The additional power required to push through a flow is calculated as the percentage of the flow against the driving speed. Implementing the power in this manner means traversing a link with a spike in flow is proportionally detrimental to the difficulty to overcome it.

In the instance that an agent's charge expires, it enters the Turbine state. Here the agent is assumed to have clamped itself in a similar manner to the Sleep state. The Turbine behaviour recharges the agent's battery at a rate of 100% per hour as opposed to calculating the charge by converting the links flow velocity into power as it is assumed that the flow is always sufficiently high to charge the battery at its maximum rate. However, when the Turbine state is forcibly entered, it remains immobile for the whole hour, representing a system reboot due to the shut down. On the contrary, if the agent chooses to enter the Turbine state, it is able to specify the time it wishes to remain in the state to gain sufficient charge.

Additionally, the agents are equipped with a Hydrant charging behaviour. Hydrants are a natural entry and exit point into subterranean networks because of their non-invasive access. As such it is assumed that agents at hydrant are accessible to the surface, in this instance for a quick-charge. At the start of the simulation various junctions in the network are assigned as hydrants. Specifically, the starting junctions are always assigned as hydrants as per simulation they represent the entry points to the network. Additionally multiple junctions who have only one connected link can serve as hydrant nodes if specified. When an agent enters a hydrant junction it charges to 100% in 10 minutes before it is placed back into the nework. As opposed to actively

removing the point agent from the system, it is modelled again as a sleeping behaviour, with no communication or movement capabilities. However, if the Hydrant system is selected, it will automatically set a charge percentage at which the agent exits the current behaviour and begins to search for a Hydrant charging point. This is a necessary mechanic to introduce as pipeline networks can be vast and it is not always a guarantee that a hydrant will be passed in one charge cycle. The behaviour searches for the nearest hydrant based off it's current junction and using Dijkstra's algorithm.

The simulation interface gives the user the option to specify which, or both, power systems it would like to implement.

5.2. **Creating a Multi-Objective Metric.** Once the power system has been specified, the Multi-Objective metric is created. Multi-Objective Genetic Algorithms (MOGAs) are optimization problems, aiming to find the best potential weightings to optimize the *shared fitness value*. By iteratively changing the weightings in different simulation runs and examining pairs that produced a low TBI value, the MOGA can hone in on a Pareto optimal pairing. In a simple two dimension Genetic Algorithm, a simple shared fitness function can be defined as:

$$f(\mathbf{x}, \mathbf{y}) = f_1(\mathbf{x}).w_1 + f_1(\mathbf{y}).w_2 \tag{14}$$

The shared fitness value, $f(\mathbf{x}, \mathbf{y})$ is the function to be optimized and determines the strength of the weighting. The $f_1(\mathbf{x})$ and $f_2(\mathbf{y})$ functions represent the functions in contention, while the w_1 and w_2 values represent the assigned weightings.

The MOGA is concerned with increasing the TSLI gain while simultaneously reducing the power expenditure. This translates to the following shared fitness function:

$$f(\mathbf{x}, \mathbf{y}) = (TSLIGained).w_1 - (ChargeExpended).w_2 \tag{15}$$

In order to have the shared fitness function influence behaviours, the TSLI metric from which they usually make their decisions is overriden

by the shared fitness function. For example, the Greedy Walk behaviour would iterate through its paths, accumulating the TSLI gains and the expected charge it would expend to inspect those pipes. Depending on the weightings assigned to the MOGA, the Greedy Walk chooses the path with the highest shared fitness function.

Once the shared fitness has been created, the initial weighting is assigned.

5.3. **Creating the Genetic Algorithm.** Genetic Algorithms are an evolutionary algorithm inspired by the natural selection process. They rely on the philosophy of survival of the fittest to keep *parents* that provide good *offspring*. In this instance, the parents are the weightings assigned to the shared fitness function, while the offspring is the normalised TBI value at the end of the simulation. If the TBI value is strong, the parents will be considered higher class than those weightings that caused a low TBI.

The essence of Genetic Algorithms is based on giving stronger parents a better chance of passing on their qualities. This is done with a range of methods, but most typically focus around *crossover* and *mutation*. At the end of a pairing when the offspring has been evaluated, the Genetic Algorithm must choose a new pair of weightings for the subsequent simulation. However, the algorithm will first skew the likelihood of the parents being chosen again, depending on the quality of their offspring.

This is enforced mainly through the crossover function. The crossover operator combines two parent weightings of good quality to form new offspring. The parents are selected from the population and their offspring evaluated. Evaluated offspring have a chance to be added to the parent population if their evaluated value is deemed high enough. By iterating through good combinations of parents and implementing a system where only acceptable pairings are reintroduced to the system, the population tends towards more optimal solutions.

In addition, each offspring value has a pre-specified chance to mutate to a different value. Therefore less than optimal values still have a chance

to be introduced to the population. Though this seems non-sensical it is extremely important as it gives the algorithm the opportunity to escape from tending towards a local optima.

Algorithm 14 is a pseudocode presented by Konak, A. Coit, D.W. and Smith, A.E in a MOGA guide titled 'Multi-Objective optimization using genetic algorithms: A tutorial' [30]. This illustrates the methodology clearly.

Algorithm 14 Genetic Algorithm

1: Set $t = 1$. Randomly generate N solutions to form the first population, P_1. Evaluate the fitness of solutions in P_1.
2: *Crossover*: Generate an offspring population Q_t as follows:
3: Choose two solutions $\mathbf{x}$ and $\mathbf{y}$ from P_1 based on the fitness values.
4: Using the crossover operator, generate offspring and add them to Q_t.
5: *Mutation*: Mutate each solution $\mathbf{x} \in Q_t$ with a predefined mutation rate.
6: *Fitness Assignment*: Evaluate and assign a fitness value to each solution $\mathbf{x} \in Q_t$ based on its objective function value and infeasibility.
7: *Selection*: Select N solutions from Q_t based on their fitness and copy them to P_{t+1}.
8: If the criterion is satisfied, terminate the search and return to the current population.

With regards to the simulation, the population values are calculated before the first simulation and updated at the end of a run. Because of the difficulties presented by concurrently updating the same population pool, the simulations run consequentially and avoid the threading and batching processes until the MOGA ends. Only then are the final weightings passed in to a usual batch of 20 runs to analyse the efficiency of the MOGAs final weightings.

Due to the extensive simulation time required to run a full simulation based MOGA cycle of sufficient size to be meaningful, in tandem with the complications of the COVID-19 pandemic, the power simulations have not been run to completion. This has been left as Future Work in Section 6.3.

6. Conclusion.

The success of the research project can be examined by considering the achievements of the thesis against the aims and objectives. The aims and objectives of this thesis were to anticipate issues in autonomous pipeline inspection for underground water networks and develop an advanced swarm behaviour based in path planning. Specifically, the final behaviour should proactively seek out solutions to the problems so as to keep high levels of inspection performance. Examining the contributions of the work presents an opportunity to explore discuss any achievements relating to the objectives. The Chapter presents a discussion into the success and contributions of the research project, potential areas of future work and concluding comments.

6.1. **Discussion.**

Exploring the literature it quickly becomes apparent that the idea of autonomous robotic swarm inspection is a relatively new field and there are no documented uses of the technology in underground pipeline networks. As such, the technical difficulties of implementing autonomous inspection robots are unknown. To gain an insight into the expected challenges the literature reviewed existing inspection methods and robotic capabilities in pipeline networks. The main obstacles to autonomous inspection robots were shown to be a hostile communication environment, strong dynamic water flows and a lack of existing autonomous systems capable of powering themselves to sufficiently inspect a network. Early examinations of swarms of inspection robots in simulations were able to provide an effective inspection process through swarm intelligence. However, the behaviour was reliant on good communication, suffered in performance with the introduction of dynamic flows and did not consider power and charge requirements. Given the unique combination of issues and their detrimental results, the research project aimed to develop a governing intelligence for the swarms that considered all major obstacles and could mitigate the effects. The scope shifted to examine path planning algorithms in an attempt to create a more intelligent swarm behaviour. It is these considerations that any contributions can be attributed to.

The contributions of the thesis are discussed below.

- Path Planning Algorithms: The focus of the thesis was to explore options in path planning for potential solutions to the inspection issues. To that end, the thesis presented three path planning algorithms of varying levels of autonomy.The CPP had early success in the simulation run without dynamic flow. In splitting up the network areas to assign robots individual subsections to inspect, the typical computational constraints of complete solutions was removed. As such, the CPP was simply implementing an optimal solution in a graph. The CPP's fall from grace proved the necessity for versatile behaviours as it was unable to adapt to the introduction of dynamic flows. On the other hand, the Greedy Walk behaviour proved to be robust in responding to dynamic flow. In the early stages of the research, it was a competitive inspection model with similar means and standard deviations to stigmergy and the ability to plan ahead. However the failings of the behaviour were illustrated when it was provided with detailed flow information, which it exploited to frequently inspect easy pipes. It did however serve to illustrate how a system could utilise perfect flow data to navigate a network in a more intelligent manner. At the very least, the two behaviours provided a comparison to the Stigmergy behaviour, further validating its use as a swarm intelligence. The Ad Hoc system was designed to circumvent the communication issues in underground networks and would in theory be able to utilise the flow data in a more effective way than the Greedy Walk. Unfortunately, issues with the code during the research project means the behaviour cannot be considered a contribution, and is instead left to future work.
- Simulation Adaptations: Adapting the simulation presented by Parrott, C. et al. [44] was necessary to better reflect the hardships of underground pipeline inspection. Implementing new mechanics tests the rigour of existing systems and provides insight into potential failings. Specific additional mechanics that were added to the simulation are the transmission time

of acoustic communication, the potential to be overwhelmed by flow and adding a power and charge system to the agents and the network. Though the changes can be considered simple or rudimentary, they open the door for intelligent response. Overwhelming flow provided the opportunity to model SLAM algorithms, and continuous autonomous inspection will not occur until a solution to the power issues is found. The simulation can now be passed forward with the capacity to explore other ideas in these fields.

- Ultrasonic and Acoustic Communication: The thesis went on the analyse the effects of communication ranges in pipeline networks, comparing both normalised and realistic methods. Where typically a larger communication range was beneficial to the inspection process, the unrealistic ranges were highlighted by the implementation of Acoustic and Ultrasonic signalling. This was necessary to prove realistic methods of communication were still sufficient to govern an efficient swarm behaviour. The results made it apparent that the behaviours were still able to inspect efficiently by sharing their TSLI memories on the rare occasions they communicated.
- SLAM Implementation: A topology base SLAM algorithm was presented to relocalise agents that had become lost. The strength of the algorithm lay in how little information was required for it to localise quickly — the only real landmarks in underground pipework are the junctions. The algorithm solves a real world issue and improved the efficiency of the swarm by returning to inspection behaviours quickly. Additionally, a novel SLAM algorithm was presented that utilises collated flow cycle data and the surrounding flow to further narrow down the potential pipe location, in an attempt to localise quicker. Again, the strength of the proposed approach lies in it's simplicity — once sufficient flow data has been collected, and the models created, this would be as simple as checking the expected flows at the current time and cross referencing this with the strength of the flow on the sensor.

- Flow Intelligence: The disruptive nature of dynamic flow causes every single behaviour becoming less efficient, regardless of communication range or prior proficiency. To counteract this, the thesis presented a novel flow intelligence method that revolved around modelling each individual pipe in a network as a flow pattern. The theory surrounding this relied heavily on Gaussian Process Regression models to prove that the data could collate quickly into a useable and accurate model. Fortunately due to the flow data required for the simulation it was possible to reconstruct a simple flow intelligence that tended towards a mature, 'perfect' data set quicker than a GPR might. It was hoped that providing a behaviour with near perfect flow information would allow the swarm to navigate tricky flows and keep a low TBI value. Though the Greedy Walk was unable to use the flow models to improve the inspection process, the method illustrated how the information could generated quickly and inform the behaviours.
- Multi-Objective Genetic Algorithm: Given the charge required for movement and sensors, the power capacity of autonomous inspection robots is limited. In order to find the correct balance between continuous inspection and losing charge, a Multi-Objective Genetic Algorithm was presented to help the behaviours make decisions that would optimise the TSLI they gained, while conserving power. The algorithm is designed to work in tandem with the flow data collected to accurately approximate the energy expended in a path plan. By finding a Pareto optimal solution between the charge expenditure and the TSLI gain, an equilibrium could be gained that improves the efficiency with which the agents traverse the network, delaying the need to charge. Unfortunately, the processing power and simulation time required to find an optimal solution compounded the COVID-19 pandemic and forced the work into Future Research.

6.2. **Conclusion.** To conclude, this thesis presented many ideas and solutions designed to improve the effectiveness of path planning algorithms in autonomous robotic swarm inspection. Though the research vision of finding a behaviour capable of navigating all the difficulties in pipeline networks has not been realised, progress has been made in creating solutions that are applicable to elements of water pipeline inspection. The presentation of an effective SLAM algorithm based solely on counting the number of junctions now allows robots to localise quickly. Additionally, the use of Gaussian Processes to model the flow of individual pipes provides the opportunity for intelligent navigation of a network in dynamic flow. The path planning algorithms presented in this thesis provided a good basis with which to explore solutions and adaptations to the simulation will enable further research into these areas.

Additionally, it is hoped that the Future Work presented is able to provide applications in autonomous swarm inspection. The Multi-Objective Genetic Algorithm provides a solid foundation from which the power constraints and feasibility can be explored. Similarly, it is hoped that the Ad Hoc behaviour and other, forward planning systems are be explored as these are the behaviours most likely to make use of the flow models and other data picked up in the inspection process.

Ultimately, though the adaptability of the stigmergy swarm behaviour provides a thorough and even inspection process, it is hoped that this thesis has illustrated that path planning algorithms are also viable inspection processes, and their capacity to integrate with other intelligent systems might elevate their efficiency further.

6.3. **Future Work.** Complications in the research project have left many areas for considerations of Future Work:

- Simulations should be run with the Ad Hoc behaviour introduced in Section 3.6. The algorithm was primarily designed to circumvent the communication issues in the network whilst remaining autonomous. Additionally, it's path planning functionality in small areas made it the preferred candidate to make use of the Gaussian flow intelligence capabilities.
- It is recommended that the Multi-Objective Genetic Algorithm in Section 5 should also be run, as it is possible to track the efficiency of the solution as the genetic algorithm progresses. The MOGA was expected to bring the work together, using path planning algorithms capable of adapting to both flow and power intelligently. The Pareto optimal weighting will undoubtedly provide the most efficient solution to the power stipulations.
- Additionally, Chapter 4.9 details an extension of the SLAM algorithm that uses gathered data to aid in the localisation process. Though this addendum was inspired by the availability of the Gaussian Process data, any appropriate model of the flow data is likely to improve the speed at which the algorithm localises. The additional variable will help distinguish between otherwise identical pipes and the check can be completed in the same manner as the number of junction links.
- Similarly, Chapter 4.8 demonstrates the need for a behaviour that can actively work with flow to ensure an even inspection process. Given the vast availability of flow data and the speed with which it can be collated and analysed, the opportunity to implement intelligent systems is difficult to ignore. It is reasonable to assume the behaviour will tend to having perfect knowledge of the network with the goal of circumventing the effects of the dynamic flow, providing a much more efficient inspection process.

References

[1] Ahr, D., Reinelt, G., (2006) '*A tabu search algorithm for the min-max k-Chinese Postman Problem*', Computers and Operations Research 33, 3403 - 3422

[2] Aleman, R. E., Zhang, X., Hill, R. R., (2008). '*An adaptive memory algorithm for the split delivery vehicle routing problem*'. Springer Science and Business Media – Heuristics. 16, 441 – 473.

[3] T. Bailey and H. Durrant-Whyte, (2006) '*Simultaneous localization and mapping (SLAM): part II*,' Robotics and Automation Magazine, 13(3), 108-117

[4] Bonabeau, E., M. Dorigo, G. Theraulaz, '*Swarm Intelligence: From Natural to Artificial Systems*', New York, NY: Oxford University Press, 1999.

[5] Breivoll EU. (2023). *Breivoll Diagnostics - Pipeline Inspection.* Available at https://breivoll.eu/diagnostics/pipeline-inspection/ (Accessed: January 2023).

[6] Bondy, J.A. and Murty, U.S.R., 1976. *Graph theory with applications* (Vol. 290). London: Macmillan.

[7] Boyd, S. and Mattingley, J., 2007. Branch and bound methods. Notes for EE364b, Stanford University, 2006, p.07.

[8] Quinonero-Candela, J., (2005), '*A Unifying View of Sparse Approximate Gaussian Process Regression*', Journal of Machine Learning Research, 6, 1939-1959

[9] Chakraborty, A. and Kar, A.K., 2017. *Swarm intelligence: A review of algorithms.* Nature-inspired computing and optimization, pp.475-494.

[10] Couceiro, M.S., 2017. *An overview of swarm robotics for search and rescue applications.* Artificial Intelligence: Concepts, Methodologies, Tools, and Applications, pp.1522-1561.

[11] Cui, Z. and Gao, X., 2012. *Theory and applications of swarm intelligence.* Neural Computing and Applications, 21(2), pp.205-206.

[12] Dantzig, G.B. and Ramser, J.H., 1959. *The truck dispatching problem.* Management science, 6(1), pp.80-91.

[13] Davidson, R., 2002. *An Introduction to Pipeline Pigging*, Pigging Products and Services Association, 9.

[14] Deng, Y., Chen, Y., Zhang, Y. and Mahadevan, S., 2012. *Fuzzy Dijkstra algorithm for shortest path problem under uncertain environment.* Applied Soft Computing, 12(3), pp.1231-1237.

[15] Discover Water. (March 2022). *Discover Water - Leaking Pipes.* Available at https://www.discoverwater.co.uk/leaking-pipes (Accessed: January 2023).

[16] Dorigo, M., Bonabeau, E. and Theraulaz, G., 2000. *Ant algorithms and stigmergy.* Future generation computer systems, 16(8), pp.851-871.

[17] Dorigo, M., Gambardella, L.M., (1997), '*Ant Colonies for the Travelling Salesman Problem*' Biosystems, 43(2), 73-81,ISSN 0303-2647

[18] H. Dubreil, Z. Altman, V. Diascorn, J.-M. Picard, and M. Clerc. *Particle swarm optimization of fuzzy logic controller for high quality rrm auto-tuning of umts networks*. In Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st, pages 1865 – 1869 Vol. 3, 2005.

[19] Eberhart, R.C and Xiaohui, H., *Human tremor analysis using particle swarm optimization*. In Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, 1999.

[20] Eiselt, H. A., Gendreau, M., Laporte, G., (1995). *Arc Routing Problems, Part I: The Chinese Postman Problem*. Operations Research. 43(2), 231 – 242.

[21] Echologics. (2023). *Echologics - LeakFinder*. Available at https://www.echologics.com/products/leakfinderst/ (Accessed: January 2023).

[22] Fletcher, R., Chandrasekaran, M., *SmartBall: A New Approach in Pipeline Leak Detection*, American Society of Mechanical Engineers, IPC2008-64064, 117-133.

[23] Gasparetto, A., Boscariol, P., Lanzutti, A. and Vidoni, R., 2015. *Path planning and trajectory planning algorithms: A general overview*. Motion and operation planning of robotic systems, pp.3-27.

[24] Hydromax. (2006). *Set Inspection: Side Scanning Evaluation Technology*. Available at http://www.hydromaxusa.com/SSET.htm (Accessed June 2021).

[25] Hydrosave. (2023). *Hydrosave Pipeline Camera Inspections*. Available at https://www.hydrosave.co.uk/services/pipeline-condition-inspections/pipeline-camera-inspections/ (Accessed: January 2023).

[26] Hydrosave. (2023). *Hydrosave Sampling Services*. Available at https://www.hydrosave.co.uk/services/pipeline-condition-inspections/sampling-services/ (Accessed: January 2023).

[27] Iszmir Nazmi, I., et al., *Development of In-Pipe Inspection Robot: A Review*, IEEE Conference on Sustainable Utilization and Development in Engineering and Technology, 2012.

[28] De Jaegere, N., Defraeye, M., Van Nieuwenhuyse, I. (2023). *The Vehicle Routing Problem: State of the Art Classification and Review*. Faculty of Economics and Business, KU Leuven, Belgium

[29] Karaboga, D., Gorkemli, B., Ozturk, C. and Karaboga, N., 2014. *A comprehensive survey: artificial bee colony (ABC) algorithm and applications*. Artificial Intelligence Review, 42(1), pp.21-57.

[30] Konak, A. , Coit, D.W. , Smith, A. E (2006), '*Multi-Objective optimization using genetic algorithms: A tutorial*', Reliability Engineering and System Safety, 91, 922-1007.

[31] Lanning, D.R., Harrell, G.K. and Wang, J., 2014, March. *Dijkstra's algorithm and Google maps*. In Proceedings of the 2014 ACM Southeast Regional Conference (pp. 1-3).

[32] Li, X., Yu, W., Lin, X. and Iyengar, S.S., 2011. *On optimizing autonomous pipeline inspection.* IEEE Transactions on Robotics, 28(1), pp.223-233.

[33] Liu, Z., Kleiner, Y., *State of the art review of inspection technologies for condition assessment of water pipes*, Elsevier, Measurement, vol. 46, Issue 1, January 2013.

[34] Mandal, S.K., Chan, F.T. and Tiwari, M.K., 2012. *Leak detection of pipeline: An integrated approach of rough set theory and artificial bee colony trained SVM.* Expert Systems with Applications, 39(3), pp.3071-3080.

[35] Marlow, D., et al., *Condition Assessment Strategies and Protocols for Water and Wastewater Utility Assets*, Tech. Rep., Water Environment Research Foundation, 2007.

[36] Maaref, M. and Kassas, Z.M., 2020, April. *Optimal GPS integrity-constrained path planning for ground vehicles.* In 2020 IEEE/ION Position, Location and Navigation Symposium (PLANS) (pp. 655-660). IEEE.

[37] Molyneux, R. , Parrott, C. , Horoshenkov, K. (2019), '*An Application of Path Planning Algorithms for Autonomous Inspection of Buried Pipes with Swarm Robots*', World Academy of Science, Engineering and Technology, Open Science Index 153, International Journal of Mechanical and Materials Engineering, 13(9), 555 - 563.

[38] Nestleroth, J.B., 2004. *Implementing current in-line inspection technologies on crawler systems.* Technology Status Report, http://www. netl. doe. gov, pp.1-14.

[39] Noto, M. and Sato, H., 2000, October. *A method for the shortest path search by extended Dijkstra algorithm.* In Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0 (Vol. 3, pp. 2316-2320). IEEE.

[40] Nuchter, A., Surmann, H., Lingemann, K., Hertzberg, J. and Thrun, S., 2004, April. 6D SLAM with an application in autonomous mine mapping. In IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 (Vol. 2, pp. 1998-2003). IEEE.

[41] Nur Afiqah Binti Haji, Y. et al., *Development and Adaptability of In-Pipe Inspection Robots*, IOSR Journal of Mechanical and Civil Engineering, vol. 11, Issue 4, Aug 2014.

[42] Okamoto Jr, J., Adamowski, J.C., Tsuzuki, M.S., Buiochi, F. and Camerini, C.S., 1999. *Autonomous system for oil pipelines inspection.* Mechatronics, 9(7), pp.731-743

[43] Osterhause, A., Mariak, F., (2005). '*On variants of the k-Chinese Postman Problem*'. Operations Research and Wirtschaftsinformatik, 30.

[44] Parrott, C., Dodd, T., Boxall, J.B. and Horoshenkov, K. (2020) 'Simulation of the Behaviour of Biologically-Inspired Swarm Robots for the Autonomous

Inspection of Buried Pipes' Tunnelling and Underground Space Technology. Vol. 101 doi.org/10.1016/j.tust.2020.103356

[45] Picacorp. (2023). *Pipeline Inspection and Condition Analysis Corporation - Water Main Inspection.* Available at https://www.picacorp.com/Services/Water-Main-Inspection (Accessed: January 2023).

[46] Rasmussen, C.E., Williams, C.K.I., (1996) '*Gaussian Processes for Regression*', Advances In Neural Information Processing Systems, (8), Chapter 2, Regression.

[47] Roh, S. G. et al., *Actively Steerable Inpipe Inspection Robots for Underground Urban Gas Pipelines*, IEEE International Conference on Robotics and Automation, 2001.

[48] Ryew, S. M. et al., *Inpipe Inspection Robot System with Active Steering Mechanism*, IEEE International Conference on Intelligent Robots and Systems, 2000.

[49] Sanchez-Torrubia, G., Torres-Blanc, C. and Gimenez-Martinez, V., 2008. *An eMathTeacher tool for active learning Fleury's algorithm.*

[50] Saravanan, R. and Sujatha, P., 2018, June. A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification. In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 945-949). IEEE.

[51] Sit, H., (2019). *Quick Start to Gaussian Process Regression*, Towards Data Science. Available at https://towardsdatascience.com/quick-start-to-gaussian-process-regression-36d838810319. (Accessed: January 2023)

[52] Schulz, E., Speekenbrink, M. and Krause, A., 2018. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. Journal of Mathematical Psychology, 85, pp.1-16.

[53] Taketomi, T., Uchiyama, H. and Ikeda, S., 2017. Visual SLAM algorithms: A survey from 2010 to 2016. IPSJ Transactions on Computer Vision and Applications, 9(1), pp.1-11.

[54] Tang, G., Tang, C., Claramunt, C., Hu, X. and Zhou, P., 2021. Geometric A-star algorithm: an improved A-star algorithm for AGV path planning in a port environment. IEEE Access, 9, pp.59196-59210.

[55] Theraulaz, G. and Bonabeau, E., 1999. *A brief history of stigmergy.* Artificial life, 5(2), pp.97-116.

[56] Thibleby, H., (2003). '*The directed Chinese Postman Problem*', Software - Practice and Experience, 33: 1081-1096

[57] Thrun, S. and Montemerlo, M., 2006. The graph SLAM algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research, 25(5-6), pp.403-429.

[58] Weisstein, Eric W. (2023). *Königsberg Bridge Problem.* MathWorld–A Wolfram Web Resource. Available at

https://mathworld.wolfram.com/KoenigsbergBridgeProblem.html (Accessed: January 2023)

[59] Worley, R. and Anderson, S. (2020) '*Topological Robot Localization in a Large-Scale Water Pipe Network*', Towards Autonomous Robotic Systems. Springer International Publishing, 77 - 89.

[60] R. Worley, Y. Yu and S. Anderson,(2020) '*Acoustic Echo-Localization for Pipe Inspection Robots*,' International Conference on Multisensor Fusion and Integration for Intelligent Systems 160-165

[61] Xu, Y., Wang, Z., Zheng, Q. and Han, Z., 2012, August. *The application of Dijkstra's algorithm in the intelligent fire evacuation system.* In 2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics (Vol. 1, pp. 3-6). IEEE.

[62] Yanming, G. et al., (2016), '*Deep Learning for Visual Understanding: A review*', Neurocomputing, 187, 27-48

[63] Yilin, Y., et al., (2019), '*A Survey on Deep Learning: Algorithms, Techniques and Applications*', ACM Computing Surveys, 51(5), 1-36

[64] Zhang, S., Dubljevic, S., *Pipeline crawler development for mapping gas pipeline topology*, American Control Conference, 2021.