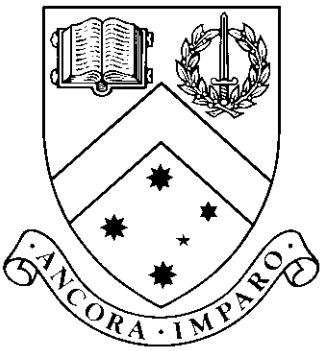
Autonomous Recharging of Swarm Robots

by

Jonathan Mullins



Thesis

Submitted by Jonathan Mullins

in partial fulfillment of the Requirements for the Degree of

Bachelor of Software Engineering with Honours (2770)

Supervisor: Dr. Bernd Meyer

Clayton School of Information Technology
Monash University

June, 2011

Contents

List of Tables

List of Figures

Autonomous Recharging of Swarm Robots

Jonathan Mullins
jlmul3@student.monash.edu
Monash University, 2011

Supervisor: Dr. Bernd Meyer
bernd.meyer@monash.edu

Abstract

The continuous operation of battery operated mobile robots requires that they have access to a reliable power source, and can access it when required. The modified e-puck robot (an educational swarm robot) that is the focus of this research, is equipped with a pickup coil, enabling its battery to be recharged wirelessly by driving onto an inductive charging platform. This goal of this research was to provide a means for the e-puck to navigate to its charging platform when required, such that it can be operated for long periods of time autonomously.

This research investigated nature inspired control and navigation strategies, and adapted the foraging strategy of E. coli bacteria into an effective audio based navigation solution for the e-puck to navigate to its charging platform. In simulation, the strategy was extended to swarms of robots, using a process found in nature called diffusion limited aggregation (DLA) for path formation. We conclude that both the bacterial search strategy and the DLA based collective navigation strategy are useful not only for our specific problem of robot recharging, but in many applications of Swarm Robot systems where navigation to some point of interest in an unknown environment is a required outcome.

Autonomous Recharging of Swarm Robots

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Jonathan Mullins
June 16, 2011

Acknowledgments

Much appreciation to Bernd, for being a constant source of insight, motivation and encouragement as I journeyed into the realm of robotics, particularly when every solution opened up another two problems, and when the real-world seemed just too damn analog. To my partner Christine, for your contribution of moral and financial support for the last few years, and for your regular contributions of lateral thinking genius, I am forever in your debt.

...

Jonathan Mullins

Monash University
June 2011

Chapter 1

Introduction

Swarm Robotics is the application of distributed control systems to large groups of mobile robots, taking inspiration mainly from the observation of social insects such as ants, termites and wasps (Şahin, 2005). Such insects exhibit group intelligence in tasks such as foraging, nest building, and transport, yet individuals use only local interactions to make decisions, without knowledge of the global state of the swarm. Applying swarm behaviour to multi-robot systems is appealing to researchers, as complex system level behaviour can be achieved using relatively simple robots and without centralised co-ordination mechanisms. Many applications of the approach have been proposed (Miner, 2007), including environment mapping, search and rescue, mining, and space exploration, although as Swarm Robotics is a relatively new field, many applications have yet to be realised. These applications can benefit from robust and scalable properties of robot swarms, particularly when the system is mission critical, and/or the size and characteristics of the target environment are variable or unknown.

The autonomous operation of mobile robots is a desirable property for applications where minimal human intervention is required, for example where the mission is of an extended length (remote sensing), or is in an environment not accessible to humans (space exploration). A key requirement for the ongoing operation of mobile robots is the access to a reliable power source. Solutions to this may be in the form of some onboard generator such as solar panels, however this project aims to investigate the autonomous recharging of robots at one or more charging stations located throughout the target environment. The physical robot that is the focus of this research is the *e-puck*, an educational swarm robot developed by EPFL (*E-puck Education Robot*, 2010). Off the shelf, its battery must be removed in order to be recharged, however a modification to the robot by Chen and Tong (2009) means that it can be recharged wirelessly and in situ using an inductive charging platform. The strategy for localisation and navigation to the charging platform had not yet been addressed, and formed the key outcome for this research.

The requirement of locating and navigating to an object or area of interest is one found in most applications of mobile robots, methods of achieving which vary depending on robot capabilities, the nature of the target, and the robots environment. An approach to navigation in a substantially complex environment might involve a robot knowing its own position, the position of the target, then calculation of a suitable path between them based on some known map of the environment (either preprogrammed or learned). Such an approach, however efficient it may be, is dependent on spatial information, and does not always allow for navigation in unknown or dynamic environments. Furthermore, reliance on this information may increase the required complexity of the robot (in terms of sensory, memory and processing capability), which is not a desirable property of Swarm Robot agents (Şahin, 2005). For this reason our research was interested in navigation strategies independent of pre-learned spatial information, and with minimal memory and processing

requirements for the individual robot agent. Although such search strategies may not be as efficient as more complex ones, they have the benefit of potentially being implemented on robots using only simple analog sensors and actuators (without a CPU), further aiding miniaturisation.

In Chapter 2, an overview of Swarm Robotics is presented, including its motivations and applications. A review of the relevant literature on nature-inspired navigation strategies and naturally occurring structures is presented, which provides an insight into some of the techniques and processes that were considered appropriate for the navigation problem that this research addresses. Chapter 3 describes how the foraging behaviour of E. coli bacteria was successfully implemented on the e-puck robot as a means of navigating through an artificial gradient (using audio as a medium) towards the charging platform. In Chapter 4, a description of how this single robot approach was extended to a swarm of robots is given, utilising the naturally occurring process of Diffusion Limited Aggregation as the foundation for a collective navigation strategy. This thesis presents these new approaches to individual and collective robot navigation, which proved through experimentation to be an effective solution for not just robot recharging, but many other robot navigation problems with similar constraints.

Chapter 2

Literature Review

2.1 Motivations and Strategies for Swarm Robot Systems

The emergent behaviour exhibited by swarm robot systems is the key motivation for research in the field. Utilising small and simple individual robots, they are capable of collectively completing tasks that would require a much more sophisticated and expensive centralised robotic system to achieve. Furthermore, the properties of swarm robot systems provide advantages over centralised systems in a number of areas described below.

Interestingly, the system level properties of robot swarms are very different to that of the individual robots. As is the case with insect swarms, individual agents are vulnerable, incapable (relative to the system-wide goal), and determine their behaviour through local interactions only. The system level properties, however, can be described as:

- Robust — the system is fault-tolerant, as the failure of some of the agents impacts efficiency and not overall system success.
- Scalable — the control algorithms rely only on local interactions among swarm agents, so scaling is much easier than centralised control schemes.
- Behaviourally complex — complex behaviour patterns emerge as a result of the large number of independent and probabilistic decisions being made at the individual level.

Robustness and scalability are desirable properties of a mobile robot system, particularly in applications involving exploration or dynamic environments. Behavioural complexity, on the other hand, can be a significant challenge in the development of swarm robot control algorithms, as there is a low level of predictability compared to centralised schemes, and proving algorithm correctness can be difficult. Simulation is usually required to demonstrate correct system behaviour, in contrast to centralised control schemes where it can be shown through formal proof (Şahin et al., 2008).

Self-organisation is intrinsic to the control strategies of swarm robot systems. A self-organised system is defined by Collier and Taylor (2004) as a system that exhibits the following features:

- The system is composed of units which respond to local stimuli.
- The units act together to achieve a division of labor.
- The overall system adapts to achieve a goal or goals more efficiently.

Dressler (2008) identified a generalised set of techniques that enable the desired behavior of self-organised systems:

- Interactions — The communication of information among system components, using either the environment or direct messaging.

- Positive and negative feedback — Individual system components respond to positive feedback with amplification, negative feedback provides a means of bounding the reaction.

- Probabilistic techniques — Used in all self-organising systems for determining local behaviours, or parameter settings for other deterministic algorithms.

The approach to control and navigation in swarm robot systems is what distinguishes it from conventional mobile robotics. In keeping with the concepts of Swarm Robotics described earlier, control algorithms are required to be simple, scalable and decentralised. The algorithms are required to autonomously position and operate the robotic agents throughout the environment according to the system objectives without global knowledge of the swarm configuration. Various nature inspired control and navigation strategies are described in detail in the following sections.

2.2 Control and Navigation in Nature Inspired Systems

2.2.1 Robot Navigation

The autonomous recharging problem that this research is addressing is principally a navigation problem. In fact, navigation is of critical importance to mobile robot systems in any application, as they must be positioned correctly to achieve any useful work. In a centralised navigation system, global mapping information of the target environment is used to aid navigation. A key motivation of swarm robot systems, however, is the tolerance of unknown environment characteristics, and therefore swarm robots cannot depend on a global map for navigation. As a result they must implement clever distributed protocols to navigate throughout the environment. Before addressing techniques for achieving this, it is useful to describe some common navigational tasks required of swarm robots:

- Dispersion — the uniform distribution of agents throughout the environment.

- Collective search — co-operative localisation of objects in an environment.

- Collective homing — an extension to collective search where agents converge on the target area.

- Path formation — arranging agents to form a path between two or more locations.

- Path following — the traversal of a path towards a goal (the path could consist of fellow swarm agents or some other environmental markers).

Dispersion, path formation and path following involve the structural organisation of the swarm, various techniques for which will be detailed later. The implementation of path *following* on mobile robots is not a matter directly related to collective navigation, in that it is conducted by a single agent. This paper will not attempt to review the multitude of approaches, as they are specific to the sensory capabilities of the target system. A type of collective navigation that is easily mappable from natural swarms to robotic swarms is distributed gradient search, described below.

Distributed Gradient Search

Examples of collective search are found throughout nature. Fish swimming in schools, for example, are able to collectively navigate to dense food sources by responding only to environmental stimuli and the behaviour of nearest neighbours. This type of environment can be modelled as a gradient on three dimensional space, with the fish performing a distributed gradient search in order to navigate towards food sources.

(Bachmayer and Leonard, 2002), (Ogren et al., 2004), and (Burian et al., 1996) show that distributed gradient search can be applied to applications of mobile robots where the descent (or ascent) of a sensory gradient is required. Each agent is able to measure the sensory gradient as it moves along its path, and compares its own gradient and bearing with that of the surrounding agents, in doing so constructing a gradient map of its local area. The agent then constructs a motion potential based on the gradient information, including an attraction / repulsion force between agents to maintain both overall swarm structure and adequate agent spacing. Once the potential is calculated for an agent, it is applied as a velocity and bearing parameter to improve its navigation toward the gradient minimum. An alternative to this approach is to have agents construct a gradient map using only sensory data at its location compared to data at neighbouring agents, and does not require the communication of bearing information along with sensor measurements.

(Kantor et al., 2006) showed a direct application of distributed gradient search to search and rescue, where teams of robotic agents search for areas affected by high temperature (i.e. a building fire) or noxious substances. It was successfully demonstrated in a burning building, with the robots entering the building, finding the source of the fire, and relaying visual and sensory data to a control centre before safely exiting again.

2.2.2 Swarm-based Optimisation

Swarm algorithms inspired by the behaviour of insects and bacteria have been applied to optimisation problems, particularly in problems requiring high computational complexity to solve using conventional analysis. Optimisation techniques based on swarming can be applied to collective navigation, as both use distributed techniques to find points in Euclidean space. Bacteria climb nutrient gradients and avoid noxious substances using *chemotaxes*, motions affected by chemical attractants and repellants. Ant colonies mark up the environment with pheromones in order to communicate favourable paths between food sources and the nest. The study of these behaviours has been applied to optimisation problems where gradient levels are unknown throughout the domain, and may be useful for certain robot navigation problems, in that the target can be represented as an optimum point in three dimensional space. Two prominent examples of swarm optimisation that could be applied to robot navigation are described below.

Bacterial Optimisation using E. Coli Behaviour

(Passino, 2002) applied the foraging behaviour of E. coli bacteria to optimisation problems. E. coli are able to move in two ways, tumble and run. During a tumble, the bacterium rotates on the spot to change its direction almost randomly. During a run the bacterium moves forward in a relatively straight line, the length of the run determined by the nutrient gradient (an increase in nutrient strength will result in a longer run). E. coli alternate between these two movements for the duration of their lifetime. The increased run lengths when climbing nutrient gradient result in an eventual convergence towards nutrient sources. Attractants released by bacterium dependent on nutrient concentration provide a swarming mechanism that helps to improve the performance of the bacteria as a whole. Healthy bacteria (in higher nutrient concentrations) are likely to reproduce, whilst unhealthy bacteria (in lower nutrient concentrations, or noxious substances) are

likely to die, further improving swarm convergence on the maximum. This component of the behaviour, however, is not directly mappable to swarms of robots.

The algorithm also implements an elimination / dispersion event (the removal of some bacteria and replacement at a random location) to prevent convergence on local minima. Implementation of elimination / dispersion in robot swarms could be achieved by randomly moving a robot to another location in the environment. The algorithm deliberately ignores some characteristics of the real-world behaviour, such as the consumption of nutrient by the bacteria. An application of the algorithm to swarm robot navigation was also presented, treating obstacles as noxious substances and the navigation target as the nutrient maximum. The simplicity of the algorithm makes it suitable for machines where computational power is limited, such as miniaturised robots.

Ant Colony Optimisation

Ant colony optimisation (ACO) is an optimisation technique based on the observation of path formation techniques exhibited in ant colonies, who lay pheromone trails in the environment to communicate path information between food sources and the nest. Stigmergy is the name given to this method of communication, where modification of the environment is used rather than agent-to-agent signalling, and its concepts could be applied to path formation in collective navigation systems. Whilst modification of the environment is not necessarily a desirable property for a swarm robot system (and replicating chemical secretion and detection is likely difficult), it is believed that stigmergy could be implemented using some of the robot agents to mark up the environment.

ACO has been applied to discrete optimisation problems that are N-P hard, such as the travelling salesman problem (TSP) and dynamic shortest path. Real world applications include telecommunication network routing (discussed further in §2.2.3), scheduling, and goods distribution, all examples of path formation problems and relevant to collective navigation. Dorigo et al. (2006) illustrate the minor differences between three popular ACO algorithms when applied to TSP, however to further illustrate the concept we will describe the Ant System (AS) implementation, first proposed by Dorigo et al. (1996).

AS is an iterative algorithm that solves the TSP problem using virtual pheromone level maintained at each edge on the graph, initially set to zero. After every iteration, pheromone levels for the edges are updated by all ants that traversed the graph and produced a possible solution. The strength of the pheromone applied to edges in a single ants path is relative to the quality of the solution compared to that of the other ants (a shorter path results in a stronger pheromone trail). Also, all pheromone levels are reduced by a constant evaporation factor at each iteration. When traversing the graph, each ant selects an unvisited node to move to next probabilistically, based on the length of the edge (shorter is better) and the strength of the pheromone level on the edge. Eventually the algorithm converges on a good (if not best) solution with all ants choosing the same path at every repeated iteration.

2.2.3 Self Organisation in Signal Routing

The application of nature inspired self-organisation to signal routing in wireless ad-hoc networks has proved to be more suitable than traditional routing protocols that were initially developed for wired networks, as ad hoc networks operate without any pre-determined network infrastructure or configuration (Dressler, 2008). Self-organisation techniques can be applied to maintain efficient network operation dynamically, and without global state information. Particularly relevant to path formation in collective navigation is its application to signal routing in networks, where routing tables are maintained dynamically between nodes to ensure efficient network operation.

Probably the most celebrated application of nature inspired self-organisation to signal routing is AntNet, first proposed in (Caro and Dorigo, 1998). AntNet is an adaptive and distributed routing algorithm that takes inspiration from real ants' behaviour in finding shortest paths. Agents build routing tables and determine network status information by using indirect communication while exploring the network, resulting in substantially better network performance than routing algorithms that preceded it. The algorithms characteristics can be summarised as follows:

- Mobile agents are periodically launched towards randomly selected destination nodes.
- Each agent moves step by step towards the destination looking for the shortest path, selecting nodes probabilistically using routing table information at the node and agent-private information.
- While moving, the agent collects information on time cost and congestion status between nodes.
- Once arrived at the destination, the agent returns to its source along the same path, modifying the routing tables of each node as a function of the goodness of the path taken.
- Once returned to the source, agents die.

Experiments comparing AntNet to six routing schemes popular at the time showed it out-performed competitors in almost all scenarios. It has since formed the basis for other routing algorithms (Baran and Sosa, 2000) (Barán, 2001) and has been implemented for *ns-2*, a networking simulation environment (*The Network Simulator - ns-2*, 2010). Applying the algorithm to robot swarms could be achieved by having some of the agents act as fixed nodes throughout the environment, with other mobile robots using them as beacons to navigate to and from different areas. Information on minimum cost paths could be maintained at the beacons by the mobile members of the swarm.

2.2.4 Environment Coverage

The capability of a robotic swarm to cover an area in uniform density is quite useful, particularly in mobile sensing applications such as environment monitoring. Consider an application to a natural or man-made disaster zone, where a remote operations centre requires sensory data for an area that is considered too dangerous for humans to assess in person. In such a scenario the physical characteristics of the environment could be substantially different to the available mapping information, as a result rendering centralised coverage control impossible. Applying distributed coverage control to this application allows sensors to disperse through the environment uniformly, using distributed routing algorithms described in §2.2.3 to relay information back to the control centre. Two approaches to environment coverage that take inspiration from nature are *potential fields* and *diffusion limited aggregation* (DLA). The use of potential fields provides uniform distribution throughout the environment, whereas DLA results in a randomly grown rooted tree. DLA will be discussed further in §2.3.

Potential Fields

The concept of using potential fields for the problem of coverage uses virtual forces to repel agents from one another and from obstacles. It is based on physics theory related to forces including gravity and electromagnetism, where motion results from the interaction between groups of particles or objects. Howard et al. (2002) applied potential fields to

mobile robot coverage using a virtual electrostatic potential between agents. The repulsive forces of the virtual potential is translated into motion that spreads the agents throughout the environment uniformly. Whilst the approach has provable *static equilibrium* (where all agents eventually become stationary), is relatively simple to implement, and completely distributed and scalable, it does not guarantee the formation of structures that are efficient for path planning and navigation. Further research is needed to demonstrate that this approach is useful for not only environment coverage but navigation and path formation tasks.

2.3 Other Self-Organised Spanning Structure Formation in Nature

Spanning structures are ubiquitous in nature (plant roots, trees, ant trails, snowflakes etc.), and when used by living organisms are typically concerned with covering some area at minimum cost. These structures are useful to distributed navigation problems when efficient area coverage and path formation is required. This section will discuss other natural spanning structure formations that could have applications to control and navigation in distributed systems.

2.3.1 DLA

DLA is a process first discussed in Witten and Meakin (1983), where particles moving about randomly aggregate to form Brownian trees (see Figure 2.1), a type of fractal structure found in nature (e.g. crystal growth, dust coalescence and snowflakes). To clarify the meaning of DLA, diffusion refers to the movement of particles due to temperature fluctuations, limited refers to the increase in size by one particle at a time, and aggregation refers to the collection of particles connected together. Simulation of DLA is typically achieved with the following algorithm:

- A seed particle is added at the origin of a lattice.
- A second particle is added at a random location far from the origin. The particle random walks until it visits a location adjacent to the seed.
- Particles are added continuously in this manner to the lattice, randomly walking until joined to the cluster.
- If a particle random walks outside the bounds of the lattice, it is removed and a new particle added again at a random location.

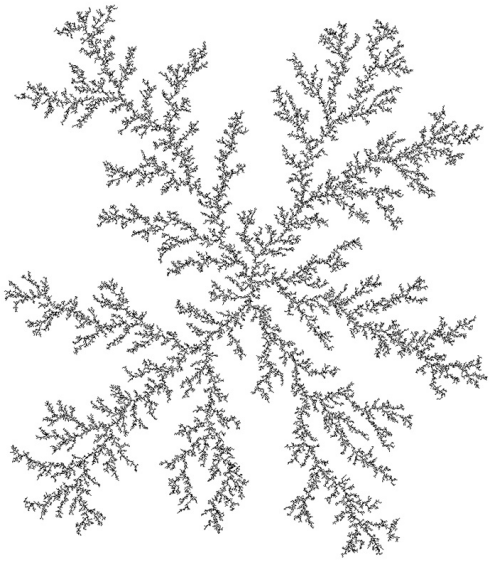


Figure 2.1: A computer generated DLA structure (From (Bourke, 2004))

Beal et al. (2009) showed that DLA could be applied to environment coverage, using a modified DLA algorithm called *random coalescence* which has two key differences to DLA:

- All robots exist in the environment from the start, rather than being added one at a time
- A maximum neighbour constant β is introduced, such that a robot will not consider itself connected if the number of adjacent neighbours is greater than β. In standard DLA, β can be considered infinite.

The algorithm proved only to converge quickly within a small range of parameters, and the authors suggest that a more intelligent strategy for the random walk component of algorithm would be required to improve performance. Nature-inspired behaviour discussed earlier in this paper such as E. coli foraging or potential fields could extend this algorithm to yield more acceptable performance. Furthermore, the approach was only considered for the problem of dispersion, and it is possible that the rooted tree structure created by the DLA approach could be useful for problems of navigation, particularly homing.

2.3.2 Shortest Path Formation

MCST

Minimum cost spanning trees are useful in swarm robot systems for a number of reasons. In a system where robot agents are dispersed through the environment, for example environment sensing or mapping, the construction of an MCST could be useful for communication back to a fixed node, or the navigation of one or more robots to another point in the environment. Determining an MCST on a graph is a well known problem that can be solved in polynomial time, however in distributed systems such as robot swarms a different approach is needed.

Gallager et al. (1983) proposed a distributed algorithm for constructing minimum spanning trees, aggregating fragments of nodes together repeatedly until all nodes are connected. Initially, for n robots there are n fragments at level 0 each with one robot. Each fragment then finds its minimum weight outgoing edge, and attempts to connect to another fragment with it. If two fragments have the same level L, and the same minimum weight outgoing edge, they combine to form a new fragment at level $L + 1$. If fragment F is at level L and fragment F' is at level $L' > L$, fragment F is absorbed by fragment F' and the level remains L'. The algorithm was shown to construct a MCST without chance of deadlock occurring.

Steiner Tree

The Steiner Tree problem is an extension to the MCST, where additional vertices and edges may be added to the graph to further reduce the total cost of the tree. This is clearly useful in path formation using mobile robots, as during the path formation agents could intelligently select positions to move to when positioning themselves in the structure. The problem is considered N-P hard to solve optimally, however nature-inspired algorithms can produce good solutions with better efficiency.

The Physarum polycephalum plasmodium is a type of slime mould that arranges its structure to efficiently cover nutrient sources when feeding. Experiments have shown that the organism can configure itself over the shortest possible route between two food sources in a maze (Nakagaki et al., 2001). Its behaviour can also be applied to the Steiner tree problem, as demonstrated by the algorithm in Nakagaki et al. (2008). Basically, the algorithm selects a source node in the graph at random, and probabilistically selects

a sink node in the graph with respect to its Euclidian distance from the source. The Physarum behaviour is applied between these nodes for a short time interval, and then the algorithm repeated until convergence. Simulation showed that the algorithm performed similarly to the natural organism, producing networks with lengths only slightly greater than optimal. Further research is needed to show that this behaviour could be implemented as a distributed algorithm for robot swarm path formation.

Trunk Trails

Trunk trails are a tree-like structure used by some species of ants to maintain a path from a foraging area back to the nest (Hölldobler and Möglich, 1980). The trail starts in the vicinity of the nest as a thick pathway, moving away from the nest and splitting first into branches and then into twigs. Ants will typically not stray far from the pathway until reaching the twig level, where they search for food and return it via the trail to the nest. When a rich food source is found, ants lay stronger recruitment pheromones to encourage other ants to take that path. A rich enough source results in a path at the twig level becoming a new branch, leading to variances in the overall structure over time.

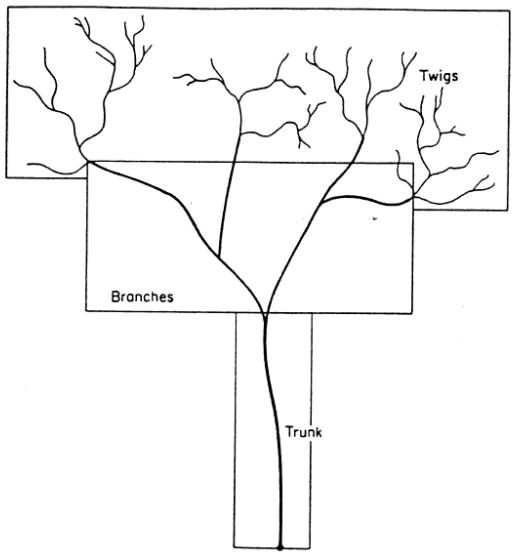


Figure 2.2: The trunk trail structure of the Pheidole milicida ant, found in southwestern U.S. deserts (From (Hölldobler and Möglich, 1980))

(Schweitzer et al., 1997) modelled this behaviour in computer simulation, where *active random walkers* move about the environment looking for food sources. The walkers choose random directions to move to next, biased toward the current direction of travel and also pheromone levels in surrounding locations. The simulations showed that similar structures to trunk trails were formed, with dense paths leading from the nest and breaking off into branches toward food sources. Whilst the simulations used virtual pheromones laid in the environment, it is possible that when applied to mobile robots, some of the agents could maintain path information whilst others perform actual work.

The trunk trail structure may have applications to path formation in robot swarms where the location of some pre-determined "nest" must be maintained whilst investigating the environment. The nest could be, for example, a recharging area that must be visited periodically to maintain battery power. For applications where higher densities of robots are required in areas of interest, this approach could ensure efficient use of the available robot agents whilst still maintaining a path back to the power source.

2.4 Conclusion

Applying distributed navigation and control algorithms to mobile robots can result in robust, adaptable, and scalable systems. Swarm Robotics applies nature-inspired techniques to such systems, exploiting probabilistic local interactions among individual agents that result in complex and capable system behaviour. This chapter has provided an insight into some of the nature-inspired strategies that have been applied to navigation, optimisation and communication problems, and also described some naturally occurring structures that may be useful to path formation tasks. The remainder of this thesis will describe how we applied two of these natural processes to the research outcome of autonomous robot recharging. In Chapter 3, an adaption of the E. coli foraging behaviour described in §2.2.2 is demonstrated through real-world experimentation to be an effective object localisation strategy for individual robots. In Chapter 4, a collective navigation strategy is provided using the DLA process described in §2.3.1 for path formation.

Chapter 3

Object Localisation

3.1 Introduction

The foraging behaviour exhibited by E-coli bacteria discussed in §2.2.2 is an example of how navigation to some place of interest (in the case of E-coli, a food source) can be achieved by iteratively improving the position of an agent using a simple and repetitive decision making process. The bacteria perform a gradient search on a nutrient medium using two types of movement, tumble and run. A navigation algorithm for a mobile robot based on this behaviour is presented in this section, as well as improvements that were made to it to make use of available sensors on the e-puck robot to improve its efficiency. Being modelled on the behaviour of bacterium, the algorithm naturally satisfies the constraint of being implementable with relatively little processing and memory requirements.

Development of the strategy required that a number of preliminary goals be achieved in order to have the required technical infrastructure, including the construction of a gradient source, and the extension of a robotics framework for experimentation. These preliminary tasks consumed a significant amount of project time, evident in the timeline provided in Appendix A. The method in which these goals were achieved will be discussed in §3.3 and §3.4, however many details of the technical challenges and achievements during this phase have been omitted, as they are not directly relevant to the research outcomes. A description of the navigation strategy itself will then be provided, and finally a formal analysis. First, the process by which a suitable gradient medium was selected will be described.

3.2 Gradient Generation and Detection

Selection of a suitable medium for use as a gradient source was primarily constrained by the sensory capabilities of the e-puck robot. The available sensors on the e-puck that can be used effectively for gradient detection are a low resolution video camera on the front of the robot, and three microphones located on the top of the robot. It is possible that the video camera could have been used to detect some light source, however the use of audio was considered more appropriate for a number of reasons:

- Sound waves have better reflective properties off most surfaces than light, making detection possible without line of sight.

- Using audio frequencies outside the human hearing range results in no environmental disturbance to humans in range of the beacon, whereas the video camera is responsive only to visible light.

- Audio and RF are conceptually similar, so adapting the method to the RF medium is conceivable.
- High ambient light conditions would probably render the video camera ineffective, whereas high ambient noise can be countered by analysing specific frequencies from the microphones if necessary.
- The e-puck's video camera is front facing, allowing detection of light in one direction only. The top mounted microphones allow for audio detection from all directions relatively evenly.

As illustrated in Figure 3.1, none of the e-puck's microphones are situated dead centre. To determine the variance of each microphone's response with respect to the orientation of the robot, a speaker generating white noise was placed near the robot, with volume measurements taken from each microphone individually at different orientations. The results (Table 3.1) showed that each microphone exhibited a significant variance, particularly microphones 0 and 1 which are located at the very edges of the robot. Hence for all experiments an average of the three microphones was used, minimising the bias introduced by robot orientation.

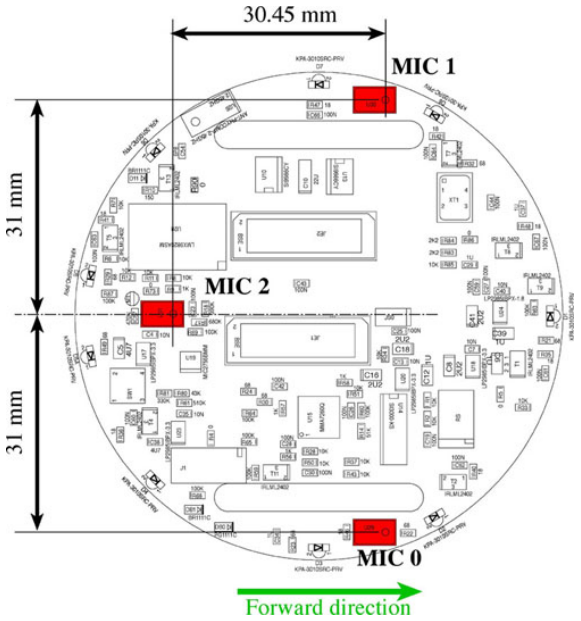


Figure 3.1: Microphone placement on the e-puck (*E-puck Education Robot*, 2010)

An assessment of the microphone response over distance was then performed using the same white noise source, to affirm that a meaningful gradient could be established using the relatively insensitive e-puck microphones. Volume at each orientation was measured at distances incrementing by 20cm, with the response curve (Figure 3.2) deemed to be useful up to at least 1.5m, albeit with a number of local maximums resulting from room reflections, and a small amount of rotational bias that under certain conditions could result in the incorrect assessment of gradient.

3.3 Omni-directional Sound Source

A typical speaker (as found in computer speakers or a hi-fi system) is a directional sound source by design. Preliminary investigations using a small speaker as an audio source in a basic rectangular environment demonstrated that a usable gradient (as measured by the

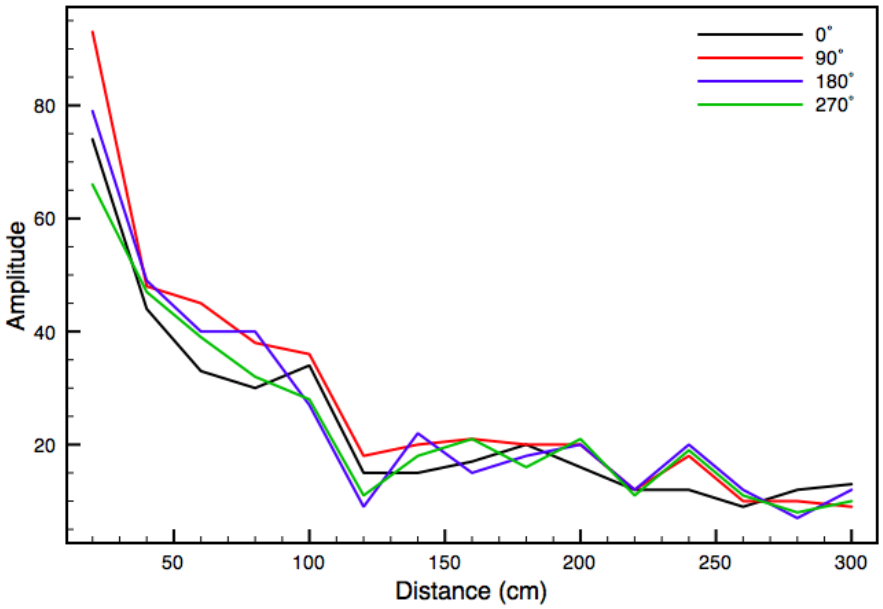


Figure 3.2: Microphone response to 3 meters

e-puck robot) was only present in an arc extending directly outwards from the speaker, as illustrated in Figure 3.3. Additionally, the steep edge at the boundary of the area in direct range of the speaker was problematic when using the navigation strategies described later. To generate a more even gradient in the environment, an omni-directional sound source was constructed using some 70mm plastic tubing and a small speaker. The speaker was mounted inside the tube facing downwards, with the tube elevated from the ground by roughly 20mm with small timber legs. As pictured in the Figure 3.4, the device allows the sound waves generated by the speaker to propagate in all directions relatively evenly.

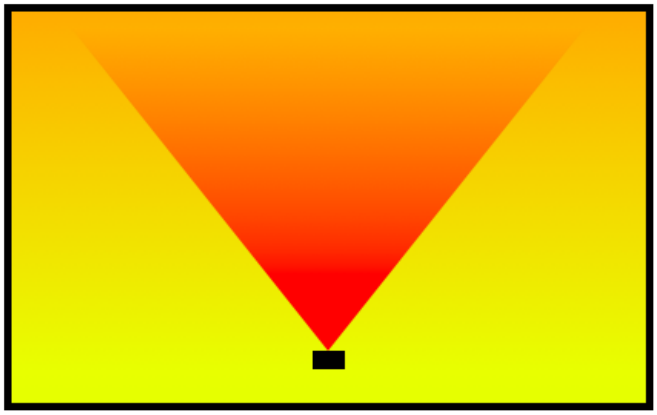


Figure 3.3: Audio gradient created by a directional speaker in rectangular environment

3.4 The Aseba Framework

Development on the e-puck robot is typically done using the supplied C libraries in conjunction with an appropriate compiler for the e-puck's DSPIC-30f microcontroller. Once compiled, a user program can be loaded onto the robot over bluetooth, or through cable connection from an MPLAB ICD3 debugger (*Microchip Development Tools*, 2011). The

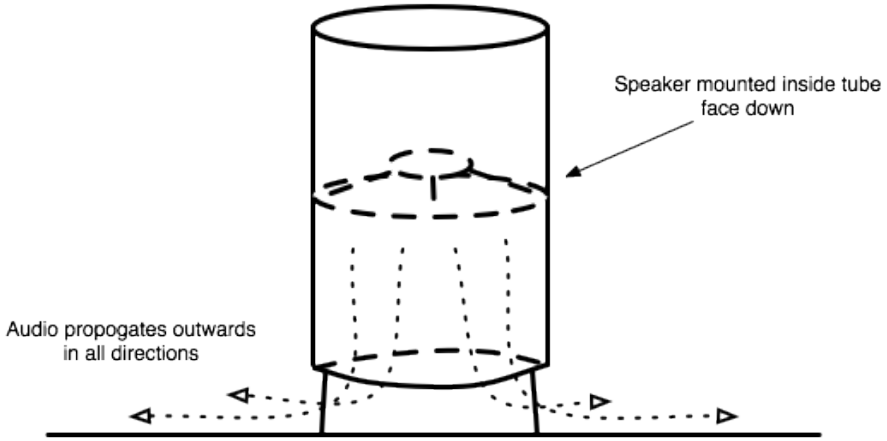


Figure 3.4: Omni-directional speaker using plastic tubing

ORIENTATION	MIC0	MIC1	MIC2	AVG
0°	72	94	106	90.7
90°	66	145	75	95.3
180°	130	86	85	100.3
270°	120	55	81	85.3
VARIANCE	**801**	**1045.5**	**136.2**	**30.8**

Table 3.1: Amplitude variance from orientation

various sensors and actuators on the robot can be configured and controlled to perform in the desired way, however there is no means to monitor the sensors in real-time whilst a program is in execution. Development of algorithms that change the robots behaviour according to sensory data are thus difficult to develop and debug. Furthermore, modification of the program code requires recompilation, reloading and a microcontroller reboot every time.

The Aseba Framework is an "event-based architecture for distributed control of mobile robots" (Magnenat et al., 2011). At its core is a virtual machine that can be loaded onto both real-world and simulated robots, allowing control of the robot through an interactive IDE (Aseba Studio) that also provides real-time feedback of the robots sensory data over bluetooth connection. Sensors and actuators specific to the robot being targeted are made available to Aseba Studio through variables that can be read from and written to within the control scripts. The control scripts can be debugged, recompiled and reloaded onto the virtual machine without requiring a reboot of the robots microcontroller, making development far easier than the native method. Figure 3.5 shows the communication between Aseba Studio, the virtual machine on the robot and the robot hardware.

The Aseba developers provide a ready made implementation of the virtual machine for the e-puck robot that provides control and monitoring of all the robots sensors and actuators, with the exception of the microphone and speakers. The decision was made to extend the Aseba virtual machine to support the audio capabilities of the robot (as opposed to developing the control algorithms natively in C), as having real-time access to the robots sensory data during algorithm development was predicted to be of great importance. Furthermore, the simulation component of Aseba (Aseba Playground), which was to be used for the second phase of development, includes a model of the e-puck (albeit

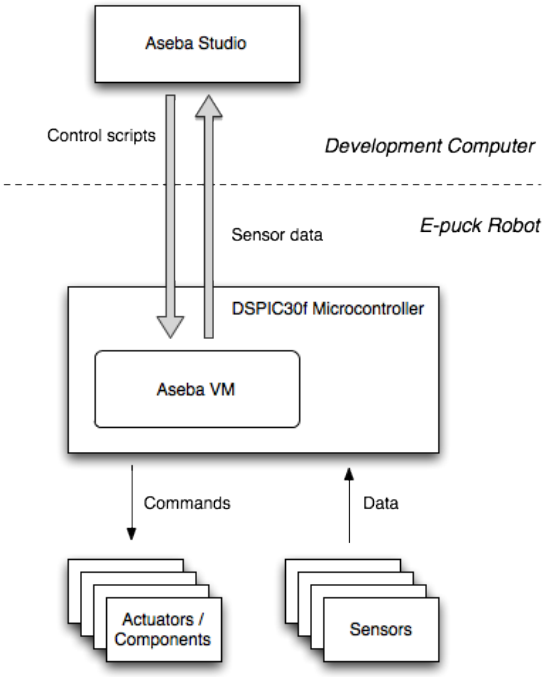


Figure 3.5: Communication flow between Aseba Studio and E-puck Robot

without audio support), so control scripts developed for the real robot could be loaded into a simulated e-puck without modification.

3.4.1 Extending the Framework

Native functions are a concept in the Aseba framework where frequently used or processor intensive functions can be implemented natively on the target robot, and called within an Aseba script as required. Rather than implementing the processing of raw audio data within the Aseba control scripts, a native function was implemented so that from within a control script, a function *get_volume()* could be called that returns the average amplitude from the most recent samples taken by the three microphones. When called, the virtual machine on the e-puck executes natively the equivalent of the following pseudo-code, calculating the total amplitude of the most recent sample from each microphone, and returning the average:

Algorithm 3.4.1: GETVOLUME()

```
global micBuffer[3]

totalAmp ← 0

// Wait 1 second to avoid residual motor noise
WAIT(1000)

// Determine amplitude for each mic buffer and add to total
for i ← 0 to 2
do { high ← −∞
     low ← ∞
     for j ← 0 to LENGTH(micBuffer[i])
     do { if micBuffer[i][j] > high
            then high ← micBuffer[i][j]
          if micBuffer[i][j] < low
            then low ← micBuffer[i][j]
     totalAmp ← totalAmp + (high − low)

// Return the average
return totalAmp/3
```

Adapting the E-coli foraging behaviour to a robot control algorithm also required random number generation for choosing rotation amounts probabilistically, and for the ability to rotate the robot by a given number of degrees, so three further native functions were implemented:

- **get_rand(mod)** returns a random integer modulo some integer *mod*
- **get_norm(mean, std)** returns a normally distributed random number with a mean of *mean* and standard deviation of *std*, generated using the Box-Muller method (Box and Muller, 1958)
- **rotate_degrees(deg)** where *deg* is an integer specifying the rotation amount in degrees, clockwise rotation for positive values, counter-clockwise for negative.

A variable *cm* was also defined in the virtual machine that is asynchronously updated by the robot to represent the distance travelled in centimetres. It can be reset at will within a control script, and was used as a convenient way to determine the distance travelled at any point in time of algorithm execution.

3.5 E-coli Inspired Gradient Search

With a development environment now established, an adaption of the E. coli foraging behaviour for the e-puck robot was developed. As discussed briefly in §2.2.2, the behaviour (a type of chemotaxis) centers around two states, tumble and run. During a tumble, the bacterium is rotating on the spot for a small amount of time (≈4sec), thereby picking a new random direction to start moving, slightly biased toward the current direction of travel. During a run phase, the bacterium is moving in a relatively straight line for an amount of time, the length of which increases when the bacterium detects that it is moving toward a more favourable nutrient source, or decreases if moving toward a noxious substance. When

in some neutral substance (neither nutrient nor toxin), the bacterium will continuously alternate between tumble and run states, essentially performing a random walk until either nutrient or noxious substance is encountered. Figure 3.6c illustrates the movement resulting from the chemotactic process, where the stronger nutrient solution is indicated by darker shading.

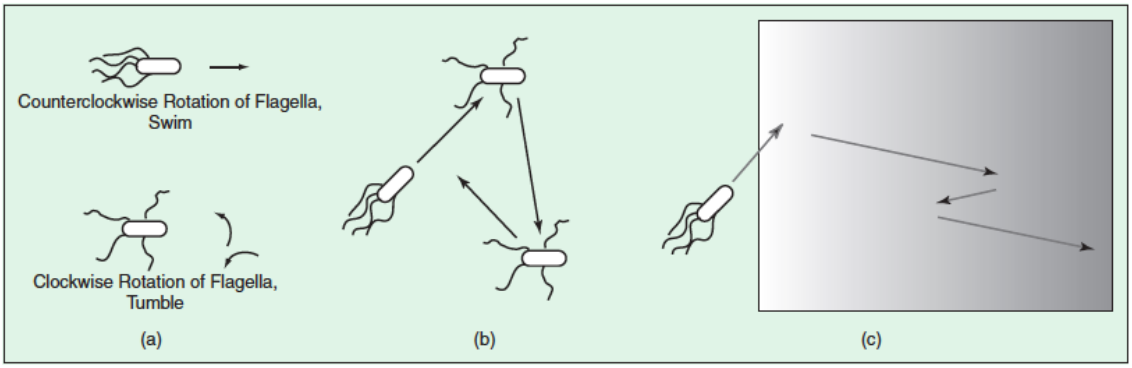


Figure 3.6: Chemotactic behaviour of E. coli (Passino, 2002)

Adapting this behaviour to a navigation strategy for the e-puck required slightly altering the way run lengths are modulated and rotations are performed. Whereas the bacterium has the ability to constantly monitor the nutrient concentration during a run, the e-puck must stop before measuring the audio level to avoid polluting the sample with motor noise. Thus to fully mimic the E. coli behaviour, the e-puck would have to continuously stop the motors, measure the volume, and start the motors again during the run phase, altering the run length accordingly. Instead, we proposed an algorithm where the run length is determined by the improvement (or otherwise) achieved by the *previous* run, not the improvement being made by the current one. For example, if run n results in an increase in volume ($\Delta v_n > 0$), then the length of run $n+1$ will be increased. Of course this would result in situations where a run yields an improvement, the robot then tumbles (rotates) to a new direction ϕ, then performs a run of increased length which may not be improving (or even worse, deteriorating) the robots position in the gradient field. To overcome this, if a run yields an improvement, the following tumble is heavily biased toward the current direction using a normal distribution around the robots current bearing, otherwise the tumble amount is uniformly random between -180 and 180 degrees. This makes it more likely that the increased length l of run $n+1$ will still be improving the robots position. If we specify a minimum and maximum run length of L_{min} and L_{max} respectively, and the volume level at the target as V_{target}, then we can say that:

$$l_n = \begin{cases} X : \mathcal{N}(0, 60), & \text{if } \Delta v_{n-1} > 0 \vee n = 0 \\ Y : (-180, 180), & \text{otherwise} \end{cases} \tag{3.1}$$

and

$$\phi_n = \begin{cases} \frac{\Delta v_{n-1}(L_{max} - L_{min})}{V_{target}} + L_{min}, & \text{if } \Delta v_{n-1} > 0 \vee n = 0 \\ L_{min}, & \text{otherwise} \end{cases} \tag{3.2}$$

Note that in equation 3.1 we chose a standard deviation of 60 to ensure 99.7% of values will fall within ±180 degrees.

Another consideration was how to deal with collisions. A common strategy for mobile robots is to apply a collision avoidance mechanism adapted from the theoretical vehicles

described in Braitenberg (1984), where an agent's motion is linked directly to some sensory feedback mechanism (for example multiple proximity sensors). A typical application of the method for a two-wheeled robot such as the e-puck is to use measurements from the proximity sensors around the robots turret, setting each wheel speed independently by summing the sensor value / coefficient products for each sensor. The robot moves forward in a straight until one or more of the sensors detects an object, at which time the robot slows the opposing wheel proportional to how close the object is. The result is a graceful deflection away from the obstacle. Considering our target was a detectable object, however, simply incorporating this behaviour into the algorithm would result in the robot never reaching the target. Instead a more simplistic approach was applied, where we simply stop short the current run when an obstacle is detected. The volume measurement occurs, and if we have not reached the target (i.e. $v < V_{target}$), the tumble phase is performed until the robot is unobstructed. This also allowed for a more stringent requirement for declaring that the target has been found, the requirement that there must be some obstacle directly in front of the robot. This helped to avoid false positives (where the robot measures $v > V_{target}$ even though it is not quite at the target). With some threshold P for the infrared proximity sensors on the robot, the exit condition for the algorithm (when the target has been found) is defined as:

$$v > V_{target} \wedge (prox_l > P \vee prox_r > P) \tag{3.3}$$

where $prox_l$ and $prox_r$ are the front-left and front-right infrared proximity sensors respectively.

First we define a simple algorithm for moving forward some distance. The S parameter represents a maximum wheel speed, and the global variables $lSpeed$ and $rSpeed$ are used to set the left and right wheel speeds of the robot:

Algorithm 3.5.1: DRIVE($length$)

---

global $cm, prox_l, prox_r, lSpeed, rSpeed$

// Reset odometer
$cm \leftarrow 0$

// Move forward
while $cm < length \wedge prox_l < P \wedge prox_r < P$ (1)
do $\begin{cases} lSpeed \leftarrow S \\ rSpeed \leftarrow S \end{cases}$

// Stop
$lSpeed \leftarrow 0$
$rSpeed \leftarrow 0$

The condition at (1) checks whether the the drive length has been achieved or a collision is immanent, and subsequently stops the robot.

Now we define the search algorithm in full:

Algorithm 3.5.2: BACTERIALSEARCH1()

global $prox_l, prox_r, cm, lSpeed, rSpeed$

$v \leftarrow 0$
$lastVol \leftarrow 0$
$runLength \leftarrow 0$

while true
do
- $v \leftarrow$ GETVOLUME()
- // **Check if we've reached target**
- **if** $v > V_{target} \wedge (prox_l > P \vee prox_r > P)$ (1)
  - **then exit**
- // **Check for improvement**
- **if** $v > lastVol$
  - **then**
    - ROTATEDEGREES(GETNORM(0, 60)) (2)
    - $runLength \leftarrow \frac{(v - lastVol)(L_{max} - L_{min})}{V_{target}} + L_{min}$
  - **else**
    - ROTATEDEGREES(GETRAND(360) − 180) (3)
    - $runLength = L_{min}$
- // **Remember volume and move**
- $lastVol \leftarrow v$
- DRIVE($runLength$)

Statement (1) checks if the exit condition is met, (2) and (3) generate a tumble amount following an improving or deteriorating run respectively. Note that in (3) we subtract 180 from our angle such that $-180 <= angle < 180$, to rotate the robot counter-clockwise for angles between 180 and 359.

Initial experiments with this algorithm only proved effective if the robot started the search within close range of the charging platform. It became apparent that when the robot was further than roughly 1.5m from the target, the gradient field was too noisy to be of use, and the robot would spend extended periods of time making little or no progress, or being regularly drawn toward small gradient peaks near the walls of the experiment area. In time the robot would eventually stumble upon the useful gradient area and complete the search, however it was considered too inefficient to be of practical use.

Drawing inspiration again from the E. coli chemotactic behaviour, the notion of having an area of neutrality was introduced, whereby instead of attempting to climb the gradient we just perform a random walk. In the case of our robot, the area outside of this useful gradient field (1.5m radius around the charging platform) needed to be treated as a neutral area rather than a usable gradient. A new constant V_{min} was defined to specify the volume level at the boundary of the useable gradient area, and then a variable $numLow$ to represent the number of consecutive volume measurements taken with $v < V_{min}$. The algorithm was then modified such that if $numLow$ is greater than some constant W, a pure random walk was initiated with length L_{max} until detection of $v > V_{min}$. The revised tumble and run calculations then became:

$$t_n = \begin{cases} X : (-180, 180), & \text{if } \Delta v_{n-1} <= 0 \vee n_{low} > W \wedge n > 0 \\ X : \mathcal{N}(0, 60), & \text{otherwise} \end{cases} \tag{3.4}$$

and

$$r_n = \begin{cases} L_{max}, & \text{if } n_{low} > W \\ \frac{\Delta v_{n-1}(L_{max} - L_{min})}{V_{target} - V_{min}} + L_{min}, & \text{if } \Delta v_{n-1} > 0 \vee n = 0 \\ L_{min}, & \text{otherwise} \end{cases} \tag{3.5}$$

The extended algorithm (see B.1.1) resulted in an observable improvement, and whilst local maximums were still present in the useable gradient area (on account of reflections and sample noise), the robot was no longer trying to climb small gradients outside this area that were largely meaningless.

3.6 Target Acquisition using Proximity Sensors

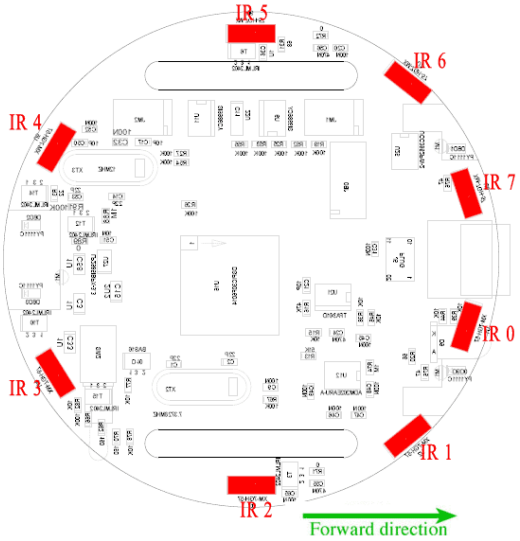


Figure 3.7: Infrared proximity sensor locations on the e-puck robot (*E-puck Education Robot*, 2010)

In an attempt to improve the algorithms performance, we sought to exploit the infrared proximity sensors on the robot to aid in acquiring the target when in close range of it. Initial observations of the algorithm in execution had shown that while the robot was acquiring the target successfully, there were frequent occurrences of a "near miss", where the robot drives within a few centimetres of the target without stopping. In the algorithm just described, the exit condition requires that one of the front two proximity sensors detects an object. The e-puck has another six proximity sensors around its body (Figure 3.7) that were thus far unused. A reverse collision avoidance method was implemented, in that we have the robot steer *towards* the sensor with the highest value, rather than away from it. This strategy was used when the robot was within a certain range of the target, deemed to be when the most recent detected volume was above some level V_{hot}. The algorithm is described by the following pseudo-code, and is called by the main algorithm to perform a drive when $v > V_{hot}$:

Algorithm 3.6.1: DRIVEHOT($length$)

```
global cm, lSpeed, rSpeed, ir[8]                                   (1)

cm ← 0

// Loop until length reached or collision immanent
while cm < length ∧ prox_l < P ∧ prox_r < P
do { hottestSensor
     hottestSensorVal ← 0

     // Find the hottest IR sensor
     for i ← 0 to 7                                                (2)
       do { if ir[i] > 0 ∧ ir[i] > hottestSensorVal
            then { hottestSensor ← i
                   hottestSensorVal ← ir[i]

     if hottestSensorVal > 0                                       (3)
       then set wheel speeds to steer toward sensor

       else { lSpeed ← S                                           (4)
              rSpeed ← S

// Drive complete
lSpeed ← 0
rSpeed ← 0
```

At (2) the algorithm is determining the highest sensor value and setting the *hottestSensor* variable (if any). If there is an object in range, (3) reduces the speed of one wheel to steer the robot towards the object. If no object is in range, (4) sets both wheels to maximum speed to continue moving in a straight line. Note that we now are accessing all the IR sensors on the robot (1), rather than just the front left and right. To implement the the new behaviour into the bacterial search, the only change required was to call the *driveHot*() algorithm if the last measured volume was above V_{hot} (see B.1.2 for full implementation).

Figure 3.8 illustrates the difference between the algorithm with and without the proximity sensor assistance. In the run phase between positions 3 and 4, where the robot would have otherwise driven past the target, the right wheel speed is reduced as a result of the right IR sensors detecting the target, and subsequently results in a successful acquisition.

3.7 Staying Warm

The probabilistic nature of the algorithm means that target acquisition does not necessarily occur when the robot comes within relatively close range of the beacon. During the tumble phase, the new rotation is chosen at random (either uniformly or biased toward the current heading), and an unfortunate combination of tumbles can easily result in the robot wandering away from the target again. This behaviour was somewhat dealt with by the proximity assistance described in the previous section, however the limited range of the sensors meant that this prevented near misses only at very close range (< 10cm). An extension to the algorithm (B.1.3) was developed which utilised the concept of a *warm*

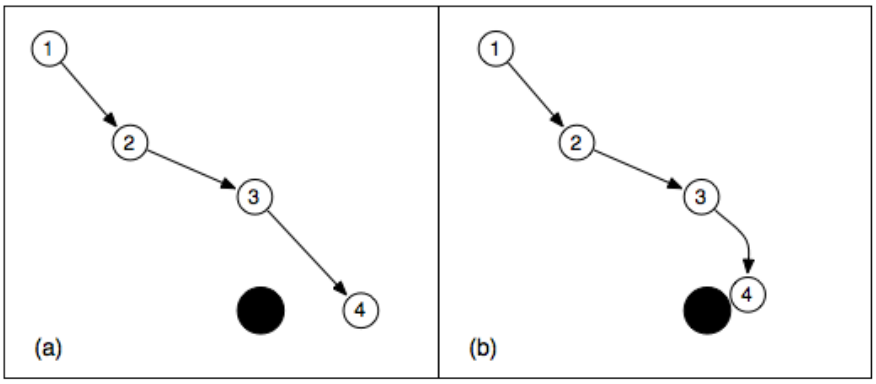


Figure 3.8: Bacterial algorithm without proximity assistance (a) and with proximity assistance (b)

area, defined as the area immediately surrounding the beacon where the detected volume v is above some threshold V_{warm}, such that $V_{warm} < V_{hot} < V_{target}$. Once entering the warm area, if the robot wanders outside it, it performs a 180° turn and returns to the previous position, ensuring that the progress made up to this point is not lost. Note that V_{warm} must be sufficiently high such that for the whole environment, no area other than that directly surrounding the beacon has $v > V_{warm}$. This is to ensure the robot does not become trapped in some local maximum elsewhere in the environment.

This small modification to the algorithm (described in B.1.3) results in the robot returning to its previous position after detecting $v < v_{warm}$. The *runLength* variable is left untouched so as to run the same distance as the run that moved the robot out of the warm area. Figure 3.9 demonstrates a scenario where implementing the warm area strategy results in a quicker acquisition of the target. The shaded area around the beacon indicates the area where $v > V_{warm}$. In (b), after leaving the warm area and stopping at position 4, the robot performs a 180 degree turn and returns to position 3 to resume the search.

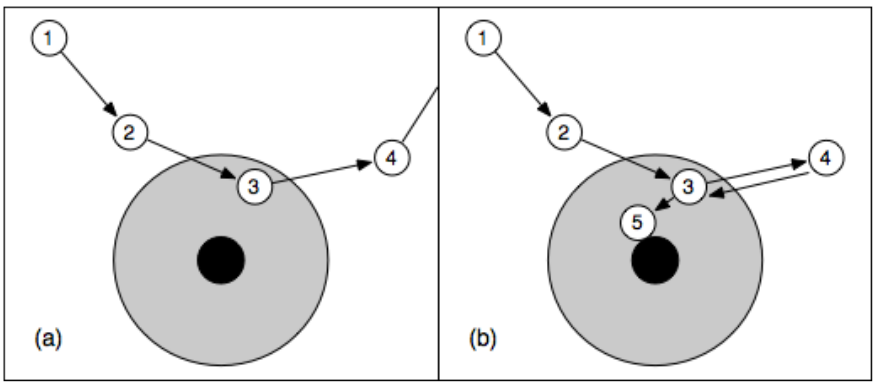


Figure 3.9: Bacterial algorithm without the warm area strategy (a), and with the warm area strategy (b)

3.8 Experiments

Once development of the bacterial search algorithm was completed through iterations of design, experimentation and informal assessment, some more formal experiments were

designed to determine if the approach was actually a useful object localisation strategy. Not having any other object localisation approach implemented on the robot to compare ours against, a simple experiment was designed comparing our baseline algorithm (B.1.1) against a pure random walk. Subsequent experiments were carried out to calculate the improvement (if any) that the extensions to the algorithm (B.1.3) had on performance. A description of the experiment method and results follows.

3.8.1 Method

A rectangular environment 2400mm x 3600mm was constructed from MDF sheets for the experiments, as the small amount of wheel clearance on the e-puck results in it becoming stuck on all but the smoothest of surfaces. Walls were fixed to the edge of the environment (roughly 50mm high) to contain the robot, and the audio beacon placed midway down the long edge of the environment, 300mm from the wall. The audio beacon was connected to a white noise source and operated at constant volume for all experiments. Reasonable values for constants such as V_{target}, P, L_{min}, L_{max} etc. were chosen, and the starting position and orientation of the robot was the same for each experiment. The algorithms were implemented on the robot and modified such that after the robot successfully navigated to the target, it performed a random walk of a fixed number of steps to a new location in the environment to start again. Measurements were taken of the time taken to reach the target from the start of each attempt (excluding random walk time between attempts), and each algorithm was given 150 minutes of running time.

3.8.2 Results

Table 3.2 summarises the results of the experiments, illustrating an improvement in average acquisition times both from the control experiment (random walk) to the baseline algorithm *bacterialSearch*2 (B.1.1), and also from the baseline algorithm to the algorithm improved with proximity sensors and warm area behaviour (B.1.3). This improvement in average acquisition times, whilst promising, is not substantial evidence to claim that *bacterialSearch*2 is definitely better than the random walk, nor that *bacterialSearch*4 is better than *bacterialSearch*2, as the sample sizes are small, particularly for the control. To check statistical significance, a Student's t-test was performed on the sets of individual acquisition times for each algorithm. A t-test result gives the probability that two sample groups are from populations with the same mean. Results for the t-tests are given in Table 3.3.

Algorithm	**No. acquisitions**	**Avg. acquisition time**
Control	14	8m 56s
bacterialSearch2	37	3m 05s
bacterialSearch4	47	2m 29s

Table 3.2: e-puck experiments summary of results

Control vs bacterialSearch2	0.012
Control vs bacterialSearch4	0.003
bacterialSearch2 vs bacterialSearch4	0.278

Table 3.3: e-puck experiments t-test results

The t-test results indicate that the performance improvement of both *bacterialSearch*2 and the extended *bacterialSearch*4 compared to the control experiment is statistically significant, in that the probability that the samples come from populations with the same mean is roughly 1%. The data does not, however, prove with any certainty that there is a difference between *bacterialSearch*2 and *bacterialSearch*4, as there is a 28% chance that the samples come from populations with the same mean. It is likely that if data was collected for the two algorithms with lower variance, either by collecting much more data or by starting each search attempt from the same location, the t-test result would show a significant difference. Using less formal reasoning, it can be argued that correctly implementing the proximity assistance and warm area behaviour will certainly not result in a deterioration in performance, as even if the extensions provide no benefit, the underlying algorithm remains the same.

3.9 Future Work

The bacterial algorithm developed for the e-puck proved to be a viable gradient-based navigation method, extending the foraging behaviour of E. coli bacteria to utilise some extra sensory information and memory available to the e-puck robot. There are, however, some outstanding practical issues that remain to be addressed in order for it to operate autonomously for extended periods of time. A brief description of these issues and suggested resolutions follows.

The microcontroller on the e-puck has no access to the voltage level of the battery, meaning that it has no means on predicting when a recharge is required. A workaround for this could involve keeping track of the operating time of the robot, and after some predetermined time beginning a search for the charging platform. This approach could be successful, however the varying cycle times of different batteries would be problematic, as would the fact that the charge is not drained linearly over time, but rather proportionally to the amount of sensor and actuator activity. A more ideal solution would be to fit a voltage sensor to the main circuit of the robot and provide this information to the robots microcontroller, such that the decision to head for the charging platform could be made using the actual voltage level, rather than the predicted level. Whether or not this extension to the robot is possible given its circuit design has not been investigated.

The charging platform that provides the electromagnetic energy to charge the e-puck requires that the robot be centred over the top of it for optimal energy transfer. The omni-directional speaker described in §3.3 is intended to sit directly above the platform in order to guide the e-puck to correct location, however the placement of the speaker at this point means that the e-puck can never move to the optimal position over the platform. A more suitable arrangement would be to have the audio beacon located high enough over the platform that the robot could drive directly underneath it and into the optimal area.

The audio beacon used for the experiments emitted a white noise source as a gradient for the e-puck to climb. If this approach were to be used in some real-world application of swarm robots in the vicinity of people, it is likely that having a constant source of noise would be unwelcome. We initially hoped to use a sound source higher than the audible human hearing range, however doing so would require having each of the microphones on the e-puck sample at a rate of at least 50KHz to sample a tone at 25KHz, which is above the human hearing range. The e-pucks A/D converter operates at 100KHz, so sampling all three microphones gives a maximum sample rate of 33KHz, insufficient for the task. It would be possible to sample from two microphones at 50KHz each, however doing so would require modification of the lower level microcontroller libraries, and was not attempted as part of this research.

3.10 Contribution

This component of the research has demonstrated through physical experimentation that the foraging behaviour of E. coli bacteria can be applied to real-world navigation problems successfully, using some gradient source and the capability to measure it. To our knowledge this has not been demonstrated in the real-world before, and certainly not using audio as a gradient source. The approach could be beneficial to any application of mobile robots where the individual agents have only basic sensory capabilities and limited memory. Through extension of the algorithm to use the proximity sensors on the robot, we have also demonstrated that the strategy is easily enhanced using sensory information that is available within a smaller range of the target. The minimal sensory and memory requirements of the base algorithm could be useful in applications of mobile robots where miniaturisation is required, and whilst implemented as a solution for our specific problem of robot recharging, could be applied to any number of scenarios where object localisation is a required outcome.

In terms of the specific problem for which this project was undertaken (recharging of the e-puck robot), this research has provided a solution to the navigation component of the task, in that the e-puck can now navigate to the charging platform from within the sensory limitations of its microphones. Whilst there are some outstanding technical issues that need addressing to enable fully autonomous operation of the robot (described in §3.9), a signification portion of the solution has been designed and implemented on the e-puck robot successfully. Once fully autonomous operation is achieved, the robot will no longer require manual intervention for recharging, increasing the ease of conducting experimental research over extended periods of time.

Chapter 4

Collective Navigation

4.1 Introduction

The E. coli inspired algorithm described in the previous chapter proved through experimentation to be a viable object localisation strategy within the context of which we applied it. Considering, however, that the range of useful gradient in our experiments was roughly 1.5 meters from the charging platform, the strategy becomes ineffective once we extend our robots environment to a larger area, as the robot resorts to random walks once outside the gradient threshold. This chapter will describe how the search strategy was extended to be effective in a much larger environment, through the introduction of more robot agents and swarm behaviour. Implementing a collective navigation strategy based on the DLA process discussed in §2.3.1, we demonstrate that the swarm can organise itself into a useful structure that can assist a robot in navigating to the charging platform. Experiments were conducted using the Aseba Playground robotics simulator, after extending it for audio support, as discussed in the next section. Finally, an assessment of the collective navigation strategy is made with respect to the robot work time that is sacrificed by the swarm in order to assist an individual robot returning to the charging platform.

4.2 Structure Formation

Collective navigation strategies for swarms of robots using the behaviour of social organisms, insects or animals, have been proposed by Schmickl and Crailsheim (2007), Schweitzer et al. (1997), Passino (2002), Passino (2000) and many others. The navigation problem at the centre of this research is essentially a homing problem, where at some point in time we require that our mobile agent visits a predefined location in the environment for maintenance purposes, in our case for replenishment of their power source. It is easy to draw a parallel between our requirement to move an agent to some useful location and the requirement of an ant to return to its nest after foraging for food. As discussed in §2.3.2, some species of ant perform this task through the formation of a tree-like structure rooted at the nest. Whilst the ants achieve this through the laying of pheromones in their environment, and we are interested in strategies that do not require this for reasons already described, the formation of some useful structure is nonetheless a plausible way to aid our robot agent in this task.

As a means to work around the constraint of our agents not altering the environment, we propose an alternative, whereby the robot agents themselves act as the environmental markers when required, and return to their primary work task (whatever that may be) after assisting in the collective navigation task. In §2.3.1 we proposed that the naturally occurring process of DLA may be useful for constructing pathways to some point of interest, as it is a rooted tree structure. In the following sections a method of constructing

a DLA tree rooted at the charging platform is given, using only the audio capabilities available to the e-puck robot. In simulation we demonstrate that once the swarm has constructed the DLA tree rooted at the charging platform, the agent requiring a recharge can navigate there relatively easily, using the bacterial search algorithm presented in the previous chapter with only minor modification.

4.3 Extending Aseba Playground

Given that this research had only a single physical robot available for experimentation, a simulation environment was required for the swarm related component. In §3.4, the extension of the Aseba virtual machine for the physical e-puck robot was described. A similar extension was required for the virtual e-puck (provided by the Aseba Framework) to support audio processing in the Aseba Playground simulator, as well as the extension of the simulator environment itself.

The *swis2d* sound plugin for the Webots robot simulator was developed by Cyberbotics in conjunction with Swarm-Intelligent Systems Group at the EPFL, Switzerland (Cyberbotics, n.d.). It provides audio simulation in two-dimensional environments through the association of virtual microphones and speakers to objects. The plugin was ported to Aseba Playground such that any abstract robot object could be equipped with a speaker, microphone, or both. During simulation, a robot equipped with a speaker object can emit audio samples, which are detectable by any microphone object in the simulation within a definable range. To calculate the detected audio samples at a given microphone, the swis2d plugin calculates any sound paths that exist between the speaker and the microphone (including reflected audio from walls and obstacles, the reflection amount definable within the plugin configuration), and adjusts the transmitted samples to account for time delay and amplitude drop over the path length. Whilst the plugin does not consider complex frequency dependent room acoustics, it provided a semi-realistic model of audio properties that were adequate for the research.

The key difference when comparing the sample data collected from the real e-puck robot compared to the simulation was the absence of a noise floor, in that a simulated microphone in an environment without speakers literally hears nothing, which is not the case in the real world. However, in the bacterial search algorithm described in the previous chapter, the parameter V_{low} was used to differentiate between the noise floor and meaningful audio, so this difference is somewhat trivial.

4.4 Implementing DLA Aggregation

Before considering how a robot could navigate a DLA structure formed by robot agents, the task of constructing the structure was addressed. As described in §2.3.1, the process of DLA involves the aggregation of randomly moving particles to one starting particle, which results in a growth of a snowflake-like structure, the initial particle being at the centre. To adapt this process to mobile robots, we simply need a starting growth point, a random walk capability for the robots, and some means for the robot agents to determine that they have aggregated (i.e. to stop random walking and become part of the DLA structure). The bacterial search algorithm described in the previous chapter already implements a random walk, and our fixed point to start the DLA growth was obviously the charging platform, so an implementation of the particle aggregation was the only extension required to achieve the DLA formation.

To implement aggregation, we required that an agent be able to determine when it is within some range of the DLA structure. At the start of the structure formation, we have only one particle in the structure, the charging platform, which is emitting an audio

tone. Again, our bacterial algorithm already has the capability to detect and measure the volume of the tone, and as the volume is relative to the distance between the sender and receiver, it was an obvious choice to use a volume level as the threshold for our virtual aggregation. Of course, the DLA process does not simply involve agents aggregating to the starting particle, it requires that any connected component of the structure become a valid point for a new agent to connect to, so as our robot agents connect to the structure, they too must begin emitting an audio tone for other agents to be able to detect and connect to. With the introduction of some volume threshold V_{dla} for aggregation, and some L as the random walk length, an implementation of the DLA process was given as:

Algorithm 4.4.1: DLAFORMATION1()

```
global out                                              (1)

v ← GETVOLUME()

// Loop until DLA found
while v < V_dla
  do { ROTATEDEGREES(GETRAND(360) − 180)
     { DRIVE(L)
     { v ← GETVOLUME()

// Stop here and transmit DLA signal
out ← true
```

Note that at (1) we access a new global variable *out* which, when set to true, causes the robot to emit a tone through its speaker.

This implementation does not specify what sound the connected nodes in the DLA structure emit, or even what type of sound is being used, it could be a tone on a certain frequency, or it could be white noise. Whilst in simulation this approach resulted in the formation of a DLA structure, it failed to address two requirements that are needed to make the structure useful. The first is that there is no simple way to signal deconstruction the DLA structure when it is no longer required. Even if the root node (the charging platform) were to disable its tone, its child nodes (the swarm agents) would remain with $v > V_{dla}$ as a result of *their* children's tone output. The second issue is that the structure is not yet useful for our homing problem, as a robot traversing it has no indication of where the root is, and could quite easily make a wrong turn at a branch and begin heading outwards towards leaf nodes.

A new iteration of the algorithm was developed, utilising frequency shifting to solve both these problems simultaneously. First, we specify that the root node in the structure (the charging platform) emits a tone at some frequency F_{root}. As agents random walk looking for the structure, they no longer simply measure the total volume in the environment, but rather measure the frequency of the loudest tone they can hear, and the volume at that frequency only. When a new agent aggregates onto the structure at the root, it transmits a tone of $F_{root} + 1$. When an agent aggregates onto that node, it transmits $F_{root} + 2$, and so on. The result is that we are now building a directed acyclic graph (DAG) from the root, which is much more appropriate for our homing problem, illustrated in Figure 4.1. Furthermore, the fact that each node is now listening only to the volume level of its parent node, disconnecting the structure is a simple matter of disabling the tone at the root node. The disconnection can then propagate through the rest of the structure.

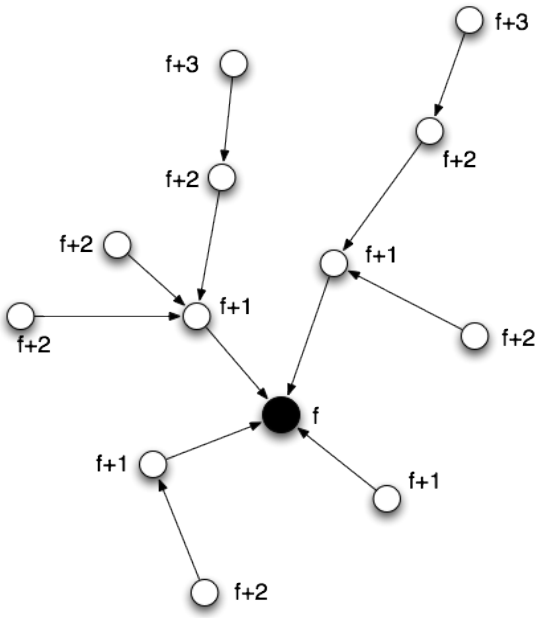


Figure 4.1: Robots connected in a DLA structure rooted at the charging platform

To implement this, some new global variables were made available to the simulated e-pucks' control scripts. $inFreq$ provides the frequency of the loudest tone the agent can hear, and $inVol$ provides its volume (not the total volume over the whole audio spectrum as before). We retain the *out* variable which is set to true to enable the agent's speaker, however we can also set the output frequency for the agent with $outFreq$. The updated algorithm is given in B.2.2.

4.5 Navigating the DLA Structure

With a process for constructing and deconstructing the DLA structure established, we then considered how the bacterial search could be adapted to take an agent from some point in the environment to the root of the structure. The modification required was trivial, in that the only significant change made was to have the agent calculate the gradient based off the volume of the *lowest* frequency DLA tone detectable, rather than measuring the total volume at the microphone. As the agent performs a gradient search toward a node in the structure, it eventually comes within range of the node's parent, and subsequently begins the gradient search on the parent. This process is repeated until the agent reaches the charging platform, the root node in the structure. The exit condition for the bacterial search now requires that not only $v > V_{target}$, but also that the current input frequency is f, the frequency being emitted by the charging platform. Once the charging platform detects that a robot has boarded, it disables its audio output, which disconnects all robots in the structure who can then return to performing their primary task. A description of the algorithm is given in B.2.1.

4.6 Sending the Alert

At this point we had developed a mechanism to create a DLA structure, and to have a robot agent navigate through it to the charging platform, however now some method was required to initiate the process. It was believed that, ideally, a robot should be able to

activate the swarm to create the DLA structure when it detects that it requires recharging. To achieve this, another frequency F_{alert} was specified (with $F_{alert} < F_{root}$), transmitted by an agent that requires a recharge. Upon detection of the signal, other swarm agents retransmit the same signal such that it propagates through the swarm, and at the same time they begin the DLA formation process. Once aggregated onto the DLA structure, an agent no longer transmits F_{alert}, they simply transmit their appropriate DLA tone. When the agent requiring a recharge arrives at the charging platform, it disables its F_{alert} output, the charging platform disables its output, and the other swarm agents can return to their primary task.

For an agent to begin alerting the swarm that it needs a recharge, a condition that its battery level b is below some threshold B_{low} is introduced. Once its battery level is above another threshold B_{max}, the robot returns to its primary task. The following algorithm describes the process as a whole:

Algorithm 4.6.1: SWARMAGENT1()

```
global inFreq, inVol, outFreq, out, b

while true
do {
    // Check if recharge required
    if b < B_low                                   (1)
      then {
        outFreq ← F_alert
        out ← true
        DLABACTERIALSEARCH1()
        out ← false
        while b < B_max                            (2)
          do WAIT()
    }
    // Check if DLA formation required
    if inFreq = F_alert                            (3)
      then {
        outFreq ← F_alert
        out ← true
        DLAFORMATION2()
    }
    // Otherwise, do primary task
    DOSOMEWORK()                                   (4)
```

Swarm agents are performing some arbitrary work task (4), until one of the two outer conditions (1) or (3) holds true. At (1) the agent checks if it should be initiating a recharge, and if so activates the swarm and begins the bacterial search towards the charging platform. Once at the platform, the agent waits for its battery to be replenished (2). A robot that hears the recharge tone (3) begins the DLA formation process, and returns to work when the process is complete.

4.7 Synchronising the Swarm State

A significant challenge with Swarm Robot systems is that message delivery is not guaranteed to every member of the swarm, depending on their current configuration and physical location in the environment. If the swarm becomes detached, in that there is a subset of agents that are not in communication range of another subset, this can be problematic. In some applications it is inconsequential, for example if all agents exist in one state and

implement only one single behaviour. In algorithm 4.6.1, however, we have implemented three states, let's call them *lowbatt*, *dla* and *worker*. Clearly the *lowbatt* state is activated and deactivated by an agents own battery level, and is not a state that requires any synchronisation amongst agents. The *dla* state, however, becomes active as the result of a message, and requires that the an agent in that state becomes part of a DLA structure for some time, before being disconnected and returned to the *worker* state. There is an inherent problem with this arrangement, which is best illustrated using a scenario with five swarm agents, a_{1-5}:

- All agents are in the *worker* state
- a_1 enters *lowbatt* state, and sends an alert which propagates to rest of the swarm, placing them in the *dla* state
- a_{2-4} form a DLA structure
- a_1 navigates through the DLA structure, begins recharging, disconnecting a_{2-4} and placing them in the *worker* state again
- a_5 never connected to the DLA structure, and is still in the *dla* state
- a_{2-4} come in range of a_5, sending them back into the *dla* state, even though the initial *lowbatt* agent has already been serviced

This scenario is not only plausible, but was observed in simulation to be very common when dealing with agents in a suitably large environment. The randomness of the agents dispersion results in many disconnected subsets of agents appearing in different locations at different times, the resulting loop of *dla* requests completely incapacitating the swarm. Even if we introduce another state in our protocol, where a *dla* agent becomes an *allclear* agent after servicing the DLA request, who's state takes precedence if one agent is transmitting *dla* and another is transmitting *allclear*? The *dla* agent may still be transmitting the alert that the *allclear* agent has already serviced, or it may actually be transmitting a new alert for some other agent that has entered the *lowbatt* state since then.

With no immediately apparent solution to this problem without the use of some form of synchronisation, the messaging protocol between agents was extended to depend on a global clock t within the range T_{min} to T_{max}. Earlier we introduced a frequency F_{alert}, used to relay the DLA request through the swarm. Rather than have agents communicate temporal information using a data connection, instead the protocol was modified such that instead of having a constant F_{alert} frequency, we select it from a frequency range F_{amin} to F_{amax} such that:

$$f_{alert} = \frac{t(F_{amax} - F_{amin})}{T_{max} - T_{min}} + F_{amin} \tag{4.1}$$

and similarly, an "all clear" frequency such that:

$$f_{allclear} = \frac{t(F_{cmax} - F_{cmin})}{T_{max} - T_{min}} + F_{cmin} \tag{4.2}$$

with the frequency ranges for *alert* and *allclear* being of equal size, and with all frequency ranges (including the DLA tones described in §4.4) being necessarily disjoint. Note that in (4.1), the value of t is the time an agent enters the *lowbatt* state, and in (4.2) the value of t is the time that the agent was disconnected from the DLA structure. Agents that are simply relaying these tones through the swarm do so without shifting the frequency.

With this newly specified protocol, it was now possible for the agents to determine the ordering of *alert* and *allclear* events based on the frequency at which the messages are being transmitted on. Each agent maintains a variable $freqAlert$ and $fregClear$, and determines its state based on which variable is higher within its range. Agents update their state using the following algorithm:

Algorithm 4.7.1: SYNCHRONISESTATE($freqAlert, freqClear$)

```
// The frequencies detected by the microphone
global inFreqAlert, inFreqClear

// Check for updated messages
if inFreqAlert > freqAlert
  then freqAlert ← inFreqAlert
if inFreqClear > freqClear
  then freqClear ← inFreqClear
```

The updated *dlaFormation* and *swarmAgent* algorithms are given in B.2.3 and B.2.4 respectively. The agent's state is regularly synchronised as part of the main control loop, and also when an agent is still random walking, trying to connect to the DLA structure. The implementation of the revised algorithms in simulation resulted in the swarm being able to maintain state information without entering an infinite loop of DLA formations. A pictorial example of the algorithm at work in the Aseba Playground simulator is provided in Appendix D.

4.8 Experiments

Following the development of this working algorithm through the same process used for the real e-puck robot of iterating design, experimentation and informal assessment, some analysis was performed to determine the efficiency of the collective navigation strategy. Rather than attempt to define any formal metrics for analysing collective navigation strategies in general, we instead attempted to gather some meaningful statistics for the DLA based navigation strategy, particularly with respect to the total amount of agent time committed to returning a single agent to the charging platform. These statistics are useful in determining some performance requirements for the strategy to be more effective than having each robot fend for itself, and help with the identification of further requirements for the feasibility of the algorithm in a real-world scenario. This section will describe the design, execution and results of the experiments, which were conducted on the Aseba Playground simulator whose extension we described in §4.3. An analysis of the results is provided, including a discussion of what applications may benefit from the approach, and how it may be improved.

4.8.1 Method

Some simple statistics were identified as necessary for providing any meaningful analysis of the strategy. Assuming some environment with n agents, and some agent entering the *lowbatt* state from a given position, we need to know the:

- Average time taken for a to reach the charging platform

- Average amount of total agent time spent assisting a in reaching the charging platform (i.e. the sum of all time spent by helper agents)

Whilst it would be interesting to compare these statistics under different environment configurations, different swarm densities and different starting locations within the environment, the time available for conducting these experiments limited us to one configuration, a simulated square environment 10m x 10m, with 60 virtual e-puck robots. The charging platform and agent in the *lowbatt* state were placed on opposing sides of the environment with 30cm clearance from the wall, with the other agents distributed randomly throughout the environment. As a control experiment, the helper agents were removed for a series of passes, such that we could ascertain some statistics on the average time for an agent to reach the charging platform without swarm assistance. Both experiment configurations use implementations of the same *swarmAgent*2 algorithm (B.2.4).

4.8.2 Results

Predictably, the time taken for an agent to locate the charging platform in such a large environment *without* the aid of other helper agents was on average very large, and also with large variance as the agent is simply random walking through the environment until coming in range of the charging platform. Table 4.1 contains the results of the control experiments, where $t_{lowbatt}$ is the time the search started (ie. the agent went into the *lowbatt* state), and $t_{charging}$ is when the charging platform was located.[1] The control experiments resulted in an average of 1hr 53m for the robot to navigate to the charging platform.

Table 4.2 contains the results of the experiments conducted with the help of the other swarm agents for navigating to the charging platform. $t_{charging} - t_{dla}$ indicates the time difference between when the agent first made contact with a DLA connected agent and when it reached the target, whereas $t_{charging} - t_{lowbatt}$ is the total search time from when the agent first sent the low battery alert to when it reached the target. Hence the difference between $t_{charging} - t_{lowbatt}$ and $t_{charging} - t_{dla}$ is indicative of the time it took the swarm to propogate the alert and build the DLA structure to an adequate size for the *lowbatt* agent to detect it. The average total search time of 18m 20sec is substantially faster than the control experiment, however it has come at a significant cost. The total work time committed by other swarm agents $t_{downtime}$ to aiding the robot in distress is on average 13hr 36m, a huge sacrifice in terms of the time lost for the swarm's primary objective.[2]

Experiment	$t_{charging} - t_{lowbatt}$
20110606_173547	03:33:54
20110606_183323	01:16:00
20110606_184157	00:31:12
20110606_184925	04:24:59
20110606_191928	00:32:20
20110606_192957	02:32:42
20110606_194618	00:24:13
Average	**01:53:37**

Table 4.1: Control Experiment Results (hh:mm:ss)

[1] The time unit used for the measurement of simulation experiments was *not* taken from a clock calibrated to real world e-puck speeds, and is strictly for comparison within the simulation experiment results

[2] $t_{downtime}$ is the sum of each agent's time commitment to the DLA process, not the time difference between when the first agent was recruited and the last agent was released

Experiment	$t_{downtime}$	$t_{charging} - t_{dla}$	$t_{charging} - t_{lowbatt}$
20110606_145800	13:44:32	00:14:53	00:22:04
20110606_151420	21:30:56	00:26:27	00:31:07
20110606_152824	10:16:16	00:11:32	00:16:53
20110606_154050	21:13:50	00:12:48	00:25:36
20110606_155407	14:53:52	00:11:44	00:19:59
20110606_160556	09:21:51	00:10:28	00:14:16
20110606_161435	10:58:01	00:07:25	00:13:08
20110606_163237	11:10:19	00:08:18	00:14:30
20110606_164053	14:30:45	00:08:47	00:17:59
20110606_165307	08:54:51	00:06:07	00:11:26
20110606_170049	04:27:59	00:05:00	00:08:18
20110606_172015	22:14:24	00:18:51	00:24:51
Average	**13:36:28**	**00:11:51**	**00:18:20**

Table 4.2: Collective Navigation Results - DLA Assisted Search (hh:mm:ss)

4.8.3 Discussion

Even with the very small amount of data collected (particularly for the control experiment), a t-test on the total search time for the agent in distress indicates with with 95% probability that an improvement has been gained using the collective navigation strategy. As already mentioned, however, the cost of this improvement is substantial, with an average of over 13 hours work time lost by the swarm in the process. This loss may be reduced through tweaking of the swarm density and DLA aggregation threshold, however it is safe to assume that when using a collective navigation strategy to guide a single agent to some location, the time cost is almost always going to outweigh the gain in search time for the single agent. This of course does not render the strategy useless, and it is easy to generalise a number of scenarios where this cost may be acceptable:

- The successful navigation of the agent to the target location is a primary objective of the swarm (as opposed to a secondary objective such as battery recharging). In such a scenario, the time cost to the swarm would not be considered a cost at all, but rather a contribution to the objective.

- The average search time *without* a collective navigation strategy is unacceptable, for example if the agents in our simulation had a battery life of one hour, using the individual search strategy would result in most agents never making it to the charging platform at all.

In terms of the scenario for which this strategy was developed, the case of each agent needing periodic maintenance (battery charging), it is possible that with some extension the strategy can become cost neutral, and even beneficial. Our experiments assume that only one agent requires assistance in navigating to the charging platform, however in reality (with the e-puck robot having a battery life of roughly 2 hours), it is likely that in most cases there will be more than one agent requiring recharging when dealing with a swarm size of 60. A break-even point can be established by determining the number of agents that would need to simultaneously take advantage of the DLA structure at the same time to offset the cost to the swarm. Given that in our experiments the average search time gain when using the collective navigation strategy is 01:35:37, and our average time cost to the swarm is 13:36:28, we would require that 9 agents (or 15%) of the swarm

use the same DLA structure to achieve a benefit. Whether or not this threshold of %15 scales with swarm size is unknown, however it is likely that a number of factors including swarm density and environment configuration would alter this threshold.

An obvious extension to our strategy would be to ensure that a threshold of $.15n$ agents are in the *lowbatt* state before beginning the DLA formation. Rather than an agent entering the *lowbatt* state and immediately triggering the formation of the DLA structure by the swarm, have each swarm agent maintain a list of agents known to be in the *lowbatt* state, and share this information throughout the swarm. When an agent detects that at least $.15n$ agents require a recharge, it begins the DLA process. This would result in an increase in total work time available to the swarm, thus resulting in a preferable strategy to the "every man for himself" method.

4.9 Future Work

A number of outstanding issues remain with the DLA based collective navigation strategy just described. Whilst the experiments have provided a proof of concept for the approach, we did not test it in different environmental configurations (ie multiple rooms or a maze-like structure), or with varying swarm densities. More experimentation is needed to determine if there is a optimal way of selecting parameters such as swarm density and DLA connection threshold. Also, a number of improvements could be made to the strategy. This section will identify some of these improvements, as well as potential solutions for them.

Synchronisation of the swarm state currently depends on a global clock. This is not something that can always be achieved easily (depending on the robot hardware, it may not even be possible), and is an added layer of complexity to the strategy that would ideally be removed. An implementation of a logical clock to manage causality would be more desirable, as synchronisation would occur through the message passing protocol itself, rather than through comparison with a common time source. Lamport's Clocks are an example of a logical clock (Lamport, 1978), whereby each agent maintains a counter that they increment when sending or receiving a message, and include their counter value in any messages they send. On receipt of a message, they compare their own counter with the counter value received, and set theirs to the greater of the two. It creates partial ordering of events, in that a 'happened before' relationship can be established between some events in the system. This is not immediately useful to our collective navigation strategy, as the swarm must be able to determine the happened before relationship between *any* request or release for DLA formation, not just some events. Whilst a logical clock cannot provide a full ordering of events (as we cannot possibly determine the causal ordering of two independent events occurring in two disconnected subsets of agents), it is possible our strategy can be modified to resolve an unknown ordering in some other way that retains reasonable performance.

We described an extension to the strategy of requiring some minimum number of agents in the *lowbatt* state to trigger the construction of the DLA formation. Our strategy currently requires that the charging platform disable its beacon when it detects a robot has boarded it, however this would no longer be a viable solution to releasing the swarm back to its primary work task, as the charging platform would need to know how many agents it is now expecting. Furthermore, if we attempt to extend the strategy to multiple charging platforms, this becomes a more complex problem. The management of when DLA connected agents are released back to work probably requires a significant change to allow the strategy to work beyond one charging platform and one agent in search mode.

4.10 Contribution

This research has demonstrated in substantially realistic simulation that a collective navigation strategy can be implemented using the DLA process at the swarm level, and the bacterial navigation strategy described in §3 at the agent level, using audio as both a means of communication and a gradient medium. Whilst collective navigation strategies modelled on biology have been proposed before in (Schmickl and Crailsheim, 2007), (Schweitzer et al., 1997), (Passino, 2002), (Nakagaki et al., 2001), (Edelstein-Keshet, 1994) and many others, using the DLA process to form a navigable structure for mobile agents has not been implemented in simulation before.

Applying the strategy to the problem of recharging mobile robots, we demonstrated that the approach is a feasible solution for enabling individual agents to operate well beyond the sensory range of their charging platform, and still be able to return to it in reasonable time with assistance from the swarm. The limited experimental data suggests that with suitable selection of parameters for a given environment, the approach can be cost effective if the DLA structure is utilised by multiple agents at once. Also, with the agents not needing geographical information on the placement of charging platforms, the platforms could be relocated as required with minimal impact to the swarm.

More generally, the collective navigation strategy we developed could be useful in any application where the navigation of one or more agents to some point of interest is an objective of the system, and must be done so without prior knowledge of the point of interest's location, or even the environment they are situated in. Consider an application such as search and rescue after a natural disaster, where known mapping information could be inaccurate at best, useless at worst. Many agents could be dispersed into an environment to locate a survivor. Upon locating the survivor, the swarm is required to converge on the location and provide some form of assistance (transport or otherwise). With only minor adjustment to our strategy, it could be adapted to act as a homing mechanism, where leaf nodes continuously begin traversal towards the root until all agents are at the required location.

In keeping with some of the fundamental concepts of Swarm Robotics, the navigation strategy is decentralised, robust, and results in the swarm exhibiting an intelligence far greater than that of the individual agent. Applications for the strategy include unknown or dynamic environments, where paths between places of interest cannot be formed using global knowledge of the environment, or there can be no assumption that previously learned information on an area is still useful. It is likely that although the strategy was developed for a particular problem of recharging, it has applications extending far beyond robot maintenance.

Chapter 5

Conclusion

This thesis is the culmination of a research project aimed not only at solving a real-world localisation problem, but also at exploring new ways of adapting naturally occurring processes to Swarm Robot systems. In Chapter 2, a literature review of work relevant to this project was given, including the motivations for distributed control of robot systems, and some of the navigation strategies used by social insects and animals to achieve complex system level behaviour through localised agent interactions. We identified the foraging strategy of E. coli bacteria as one of particular interest to this project, as it seemed directly mappable to the task of having the e-puck robot navigate to its charging platform, should we create an artificial gradient peaking at the platform. The literature review also investigated naturally occurring structures, as they were considered to be of potential use in extending the navigation strategy to swarms of robots.

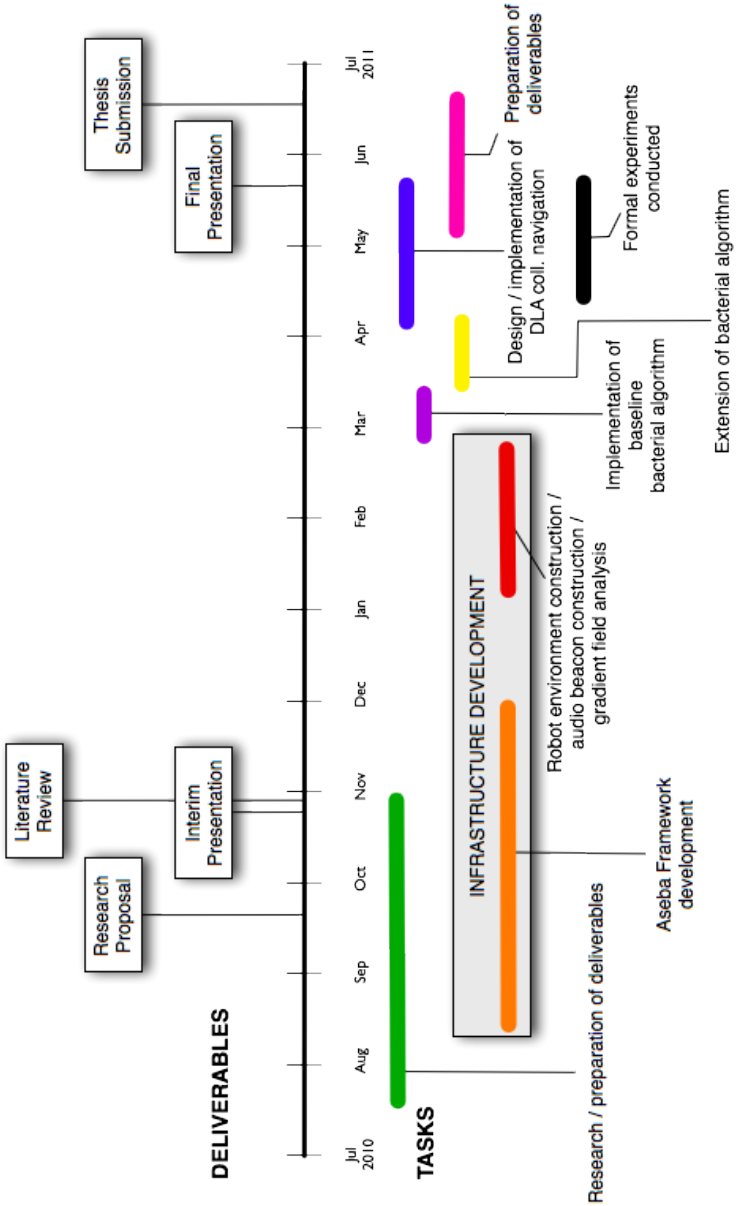
The principal outcome of the research, to devise a means for the e-puck robot to autonomously navigate to its recharging platform, was realised through the implementation of an audio-based gradient search modelled on the behaviour of the E. coli bacteria, as detailed in Chapter 3. The bacterial search proved through experimentation to be not only effective, but extensible using more refined localisation techniques once within closer range of the target. With some further work on the technical aspects of implementation, the strategy will provide a robust solution for autonomous recharging of the robot, which will aid future experimentation by providing continuous unaided operation. As well as providing a solution for the e-puck recharging problem, the research has provided evidence through experimentation that the bacterial foraging strategy is a viable solution to object localisation problems where limited memory and sensory capabilities are available. Minimising sensory and memory requirements for the mobile agents can also produce payoffs in terms of miniaturisation and cost to build, both of which are significant technical constraints in real-world applications of Swarm Robot systems.

The naturally occurring process of diffusion limited aggregation was demonstrated in Chapter 4 to be useful as a means of path formation for a collective navigation strategy. In simulation, we demonstrated that a swarm of robots can form a tree structure using DLA, resulting in a series of paths that all lead to some point of interest. Using only audio signals, the strategy enables an individual agent to alert the swarm that it requires assistance, causing the swarm to construct the DLA structure. The initiating agent then traverses the structure to its root using the bacterial algorithm described in Chapter 3, and subsequently the swarm are released back to their primary objective. An extension to the strategy was described whereby it could not only provide a better search time for the individual agent, but reduce the overall search time spent by the swarm as a whole. The approach, developed for the specific task of having an agent return to a charging platform, could be applied in any application that requires one or more agents to converge on a point of interest in an environment.

The research presented in this thesis has provided a contribution not only to the e-puck recharging problem that was the initial focus of the work, but has also developed a collective navigation strategy that we believe has applications well beyond the context of recharging robots. We have presented implementations of nature-inspired search strategies using a real-world robot, and a simulated swarm. The bacterial search algorithm is a reliable search strategy that could be adapted to other gradient mediums, whilst the DLA based collective navigation strategy was demonstrated in simulation to be a viable means of path formation and homing, and with further refinement could become an effective navigation approach for many applications of Swarm Robot systems.

Appendix A

Project Timeline



Appendix B

Algorithms

B.1 Bacterial Search

Algorithm B.1.1: BACTERIALSEARCH2()

global $prox_l, prox_r, cm, lSpeed, rSpeed$

$v \leftarrow 0$
$lastVol \leftarrow 0$
$runLength \leftarrow 0$
$numLow \leftarrow 0$ (1)

while true
do
- $v \leftarrow$ GETVOLUME()
- **// Check if we've reached the target**
- **if** $v > V_{target} \wedge (prox_l > P \vee prox_r > P)$
  - **then exit**
- **// Check if we're outside usable area**
- **if** $v < V_{min}$ (2)
  - **then** $numLow \leftarrow numLow + 1$
  - **else** $numLow \leftarrow 0$
- **if** $numLow > W$ (3)
  - **then**
    - ROTATEDEGREES(GETRAND(360) − 180)
    - $runLength = L_{max}$
  - **else**
    - **// Check for improvement**
    - **if** $v > lastVol$
      - **then**
        - ROTATEDEGREES(GETNORM(0, 60))
        - $runLength \leftarrow \frac{(v - lastVol)(L_{max} - L_{min})}{V_{target} - V_{min}} + L_{min}$
      - **else**
        - ROTATEDEGREES(GETRAND(360) − 180)
        - $runLength = L_{min}$
- **// Remember volume and move**
- $lastVol \leftarrow v$
- DRIVE($runLength$)

The variable *numLow* is defined at (1), and incremented at (2) if we're outside the usable gradient area, or reset to zero if we're not. At (3) the tumble and run parameters are set for a random walk if the number of iterations outside of the useable gradient exceeds W.

Algorithm B.1.2: BACTERIALSEARCH3()

global $prox_l, prox_r, cm, lSpeed, rSpeed$

$v \leftarrow 0$
$lastVol \leftarrow 0$
$runLength \leftarrow 0$
$numLow \leftarrow 0$

while true
do
- $v \leftarrow$ GETVOLUME()
- // **Check if we've reached the target**
- **if** $v > V_{target} \wedge (prox_l > P \vee prox_r > P)$
  - **then exit**
- // **Check if we're outside usable area**
- **if** $v < V_{min}$
  - **then** $numLow \leftarrow numLow + 1$
  - **else** $numLow \leftarrow 0$
- **if** $numLow > W$
  - **then** ROTATEDEGREES(GETRAND(360) − 180); $runLength = L_{max}$
  - **else**
    - // **Check for improvement**
    - **if** $v > lastVol$
      - **then** ROTATEDEGREES(GETNORM(0, 60)); $runLength \leftarrow \frac{(v - lastVol)(L_{max} - L_{min})}{V_{target} - V_{min}} + L_{min}$
      - **else** ROTATEDEGREES(GETRAND(360) − 180); $runLength = L_{min}$
- $lastVol \leftarrow v$
- // **Check if we should activate IR sensors for drive**
- **if** $v > V_{hot}$
  - **then** DRIVEHOT($runLength, P, S$)
  - **else** DRIVE($runLength, P, S$)

Algorithm B.1.3: BACTERIALSEARCH4($V_{target}, V_{hot}, V_{warm}, V_{min}, L_{min}, L_{max}, P, S, L$)

global $prox_l, prox_r, cm, lSpeed, rSpeed$

$v \leftarrow 0$
$lastVol \leftarrow 0$
$runLength \leftarrow 0$
$numLow \leftarrow 0$

while true
do
- $v \leftarrow$ GETVOLUME()
- **// Check if we've reached the target**
- **if** $v > V_{target} \wedge (prox_l > P \vee prox_r > P)$
  - **then exit**
- **// Check if we're outside usable area**
- **if** $v < V_{min}$
  - **then** $numLow \leftarrow numLow + 1$
  - **else** $numLow \leftarrow 0$
- **if** $numLow > W$
  - **then**
    - ROTATEDEGREES(GETRAND(360) − 180)
    - $runLength = L_{max}$
- **// Check if we've moved outside the warm area**
- **if** $lastVol >= V_{warm} \wedge v < V_{warm}$ (1)
  - **then** ROTATEDEGREES(180)
  - **else**
    - **// Check for improvement**
    - **if** $v > lastVol$
      - **then**
        - ROTATEDEGREES(GETNORM(0, 60))
        - $runLength \leftarrow \frac{(v - lastVol)(L_{max} - L_{min})}{V_{target} - V_{min}} + L_{min}$
      - **else**
        - ROTATEDEGREES(GETRAND(360) − 180)
        - $runLength \leftarrow L_{min}$
- $lastVol \leftarrow v$
- **// Check if we should activate IR sensors for drive**
- **if** $v > V_{hot}$
  - **then** DRIVEHOT($runLength, P, S$)
  - **else** DRIVE($runLength, P, S$)

B.2 Collective Navigation

Algorithm B.2.1: DLABACTERIALSEARCH1()

```
global inFreq, inVol, prox_l, prox_r, cm, lSpeed, rSpeed                    (1)

lastVol ← 0
runLength ← 0
numLow ← 0

while true
do { // Check if we've reached the target
     if inVol > V_target ∧ inFreq = F ∧ (prox_l > P ∨ prox_r > P)            (2)
       then exit

     // Check if we're outside usable area
     if inVol < V_min
       then numLow ← numLow + 1
       else numLow ← 0
     if numLow > W
       then { ROTATEDEGREES(GETRAND(360) − 180)
              runLength ← L_max

       else { // Check if we've moved outside the warm area
              if lastVol >= V_warm ∧ inVol < V_warm
                then ROTATEDEGREES(180)
                else { // Check for improvement
                       if inVol > lastVol
                         then { ROTATEDEGREES(GETNORM(0, 60))
                                runLength ← ((inVol − lastVol)(L_max − L_min)) / (V_target − V_min) + L_min

                         else { ROTATEDEGREES(GETRAND(360) − 180)
                                runLength ← L_min
     lastVol ← inVol

     // Check if we should activate IR sensors for drive
     if inVol > V_hot
       then DRIVEHOT(runLength)
       else DRIVE(runLength)
```

The global variables $inFreq$ and $inVol$ (1) are used instead of the $getVolume$ method used previously. At (2) the exit condition now has a check to ensure the charging platform frequency F has been localised.

Algorithm B.2.2: DLAFORMATION2()

```
global inFreq, inVol, outFreq, out

// Loop until DLA found
while inVol < V_dla
  do { ROTATEDEGREES(GETRAND(360) − 180)
     { DRIVE(L)

// Stop here and transmit DLA signal
outFreq ← inFreq + 1                                  (1)
out ← true

// Wait until DLA signal lost, then continue
while inVol >= V_dla                                  (2)
  do WAIT()

out ← false
```

At (1) the output frequency is set to the detected DLA frequency plus one, and at (2) we are simply waiting for our parent to disable its tone, signalling that we are finished with the DLA process.

Algorithm B.2.3: DLAFORMATION3($freqAlert, freqClear$)

```
global inDlaFreq, inDlaVol, outFreq, out

// Loop until DLA found
while inVol < V_dla
      { // Check for updated state messages
      { SYNCHRONISESTATE(freqAlert, freqClear)
      {
  do  { if freqAlert − F_amin ≤ freqClear − F_cmin
      {   then exit
      { ROTATEDEGREES(GETRAND(360) − 180)
      { DRIVE(L)

// Stop here and transmit DLA signal
outFreq ← inFreq + 1
out ← true

// Wait until DLA signal lost, then continue
while inVol >= V_dla
  do WAIT()

out ← false
```

Algorithm B.2.4: SWARMAGENT2()

```
global outFreq, out, b

// Variables for retaining state
freqAlert ← F_amin
freqClear ← F_cmin

while true
do { // Update state information
     SYNCHRONISESTATE(freqAlert, freqClear)

     // Check for low battery
     if b < B_low
       then { outFreq ← F_alert
              out ← true
              DLABACTERIALSEARCH1()
              out ← false
              while b < B_max
                do WAIT()

     // Check for DLA formation required
     if freqAlert − F_amin > freqClear − F_cmin
       then { outFreq ← freqAlert
              out ← true
              DLAFORMATION3(freqAlert, freqClear)

     // Check if we should update a nearby robot with allclear
     if inFreqAlert
       then { outFreq = freqClear
              out ← true

     // Just do some work...
     DOSOMEWORK()
```

Appendix C

e-puck Experiment Results

Table C.1: Acquisition times for e-puck experiments (h:mm:ss)

Random walk	**bacterialSearch2**	**bacterialSearch4**
0:10:09	0:01:19	0:00:32
0:06:08	0:00:29	0:01:05
0:04:14	0:04:48	0:06:17
0:14:04	0:02:09	0:00:12
0:09:39	0:05:11	0:01:08
0:18:08	0:02:47	0:04:02
0:09:52	0:00:19	0:01:44
0:23:59	0:01:26	0:01:58
0:05:08	0:02:37	0:04:23
0:05:19	0:01:59	0:09:37
0:00:33	0:01:41	0:00:56
0:03:09	0:04:04	0:02:39
0:01:36	0:03:43	0:01:18
0:13:09	0:00:10	0:00:18
	0:03:23	0:02:49
	0:00:16	0:02:25
	0:01:22	0:00:45
	0:02:58	0:03:56
	0:01:22	0:04:33
	0:06:24	0:02:15
	0:01:14	0:03:03
	0:04:12	0:02:35
	0:01:42	0:00:02
	0:01:40	0:02:26
	0:01:33	0:03:39
	0:00:26	0:01:17
	0:04:35	0:05:26
	0:01:18	0:03:21
	0:03:06	0:02:32
	0:02:44	0:01:51
	0:04:20	0:02:17
	0:02:20	0:00:30
	0:02:51	0:01:51
	0:10:03	0:01:26
	0:05:32	0:04:59
	0:03:24	0:00:26
	0:15:01	0:01:51
		0:06:10
		0:02:11
		0:00:39
		0:00:00
		0:02:46
		0:03:03
		0:03:04
		0:01:59
		0:02:11
		0:02:52

Appendix D

Simulation Screenshots

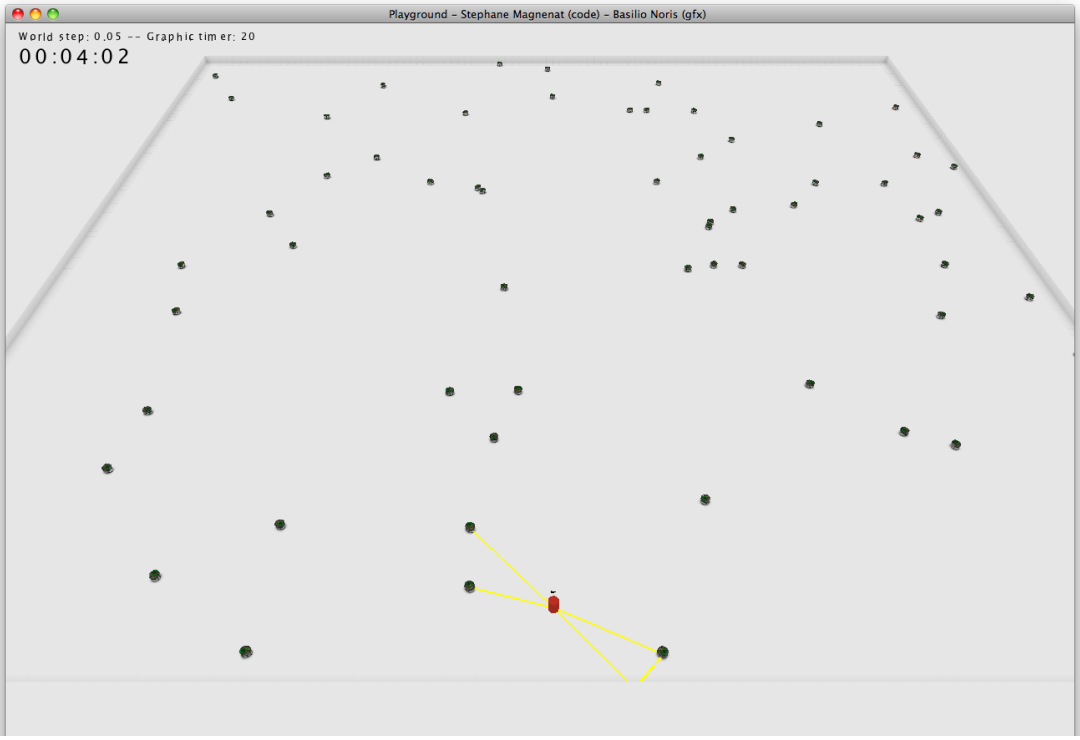


Figure D.1: Swarm robots performing some arbitrary task. The red cylinder represents the charging platform beacon, yellow lines indicate sound paths between microphones / speakers

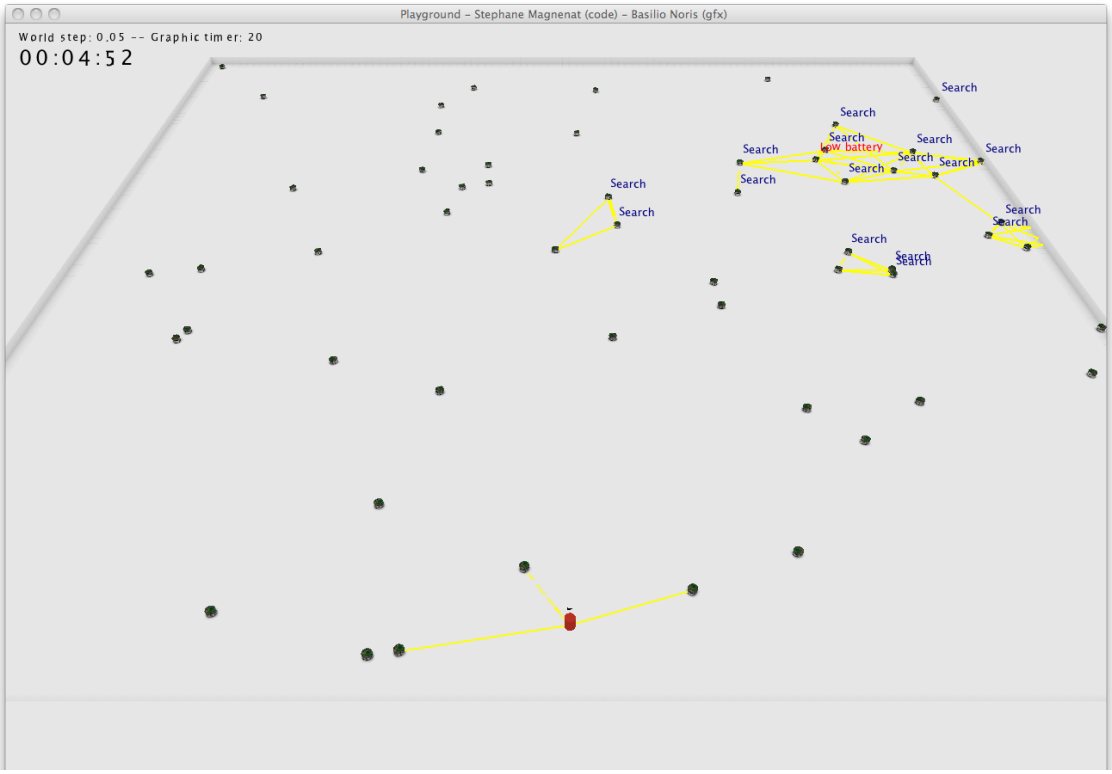


Figure D.2: Robot enters *lowbatt* state, sending alert to nearby agents who relay the alert through the swarm

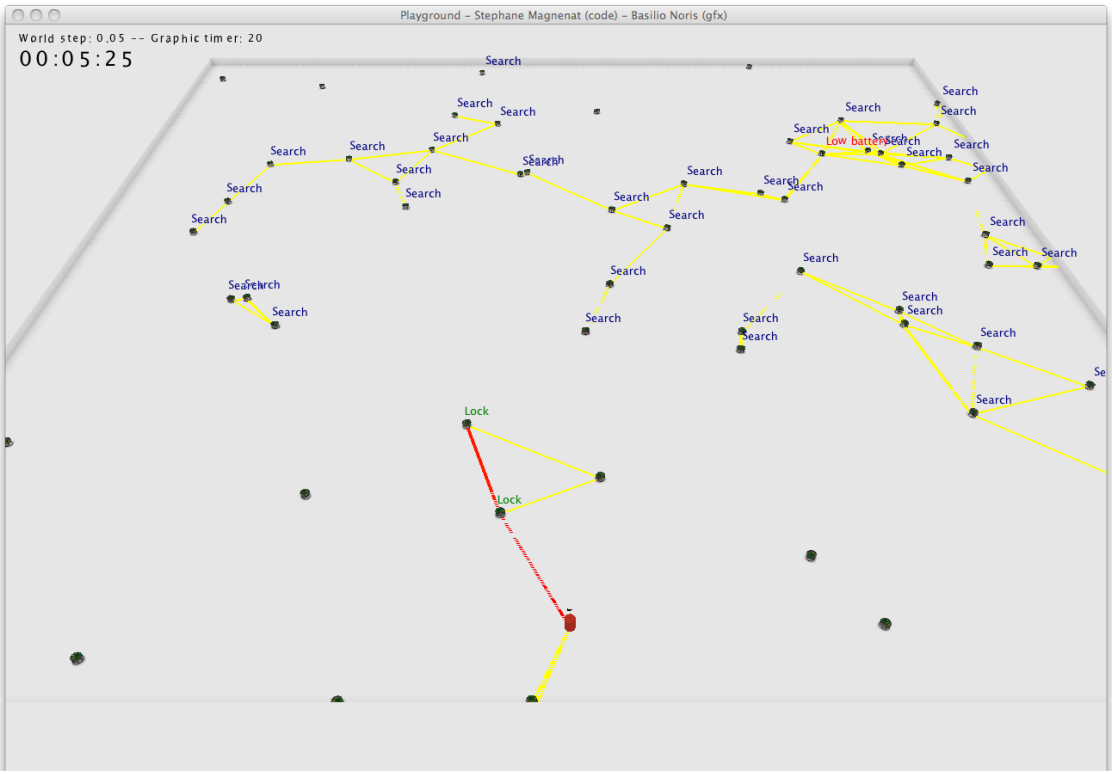


Figure D.3: Agents near the charging platform beacon (centre bottom) begin construction of DLA structure (indicated by red audio paths)

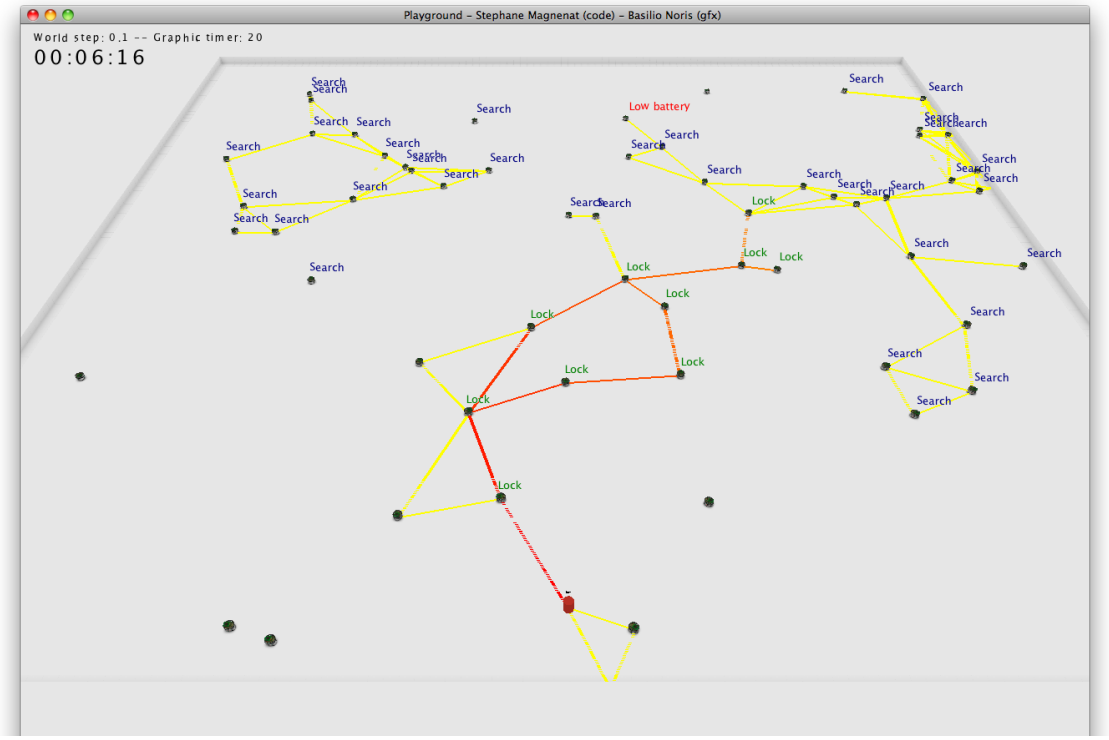


Figure D.4: DLA structure continues to grow through the environment

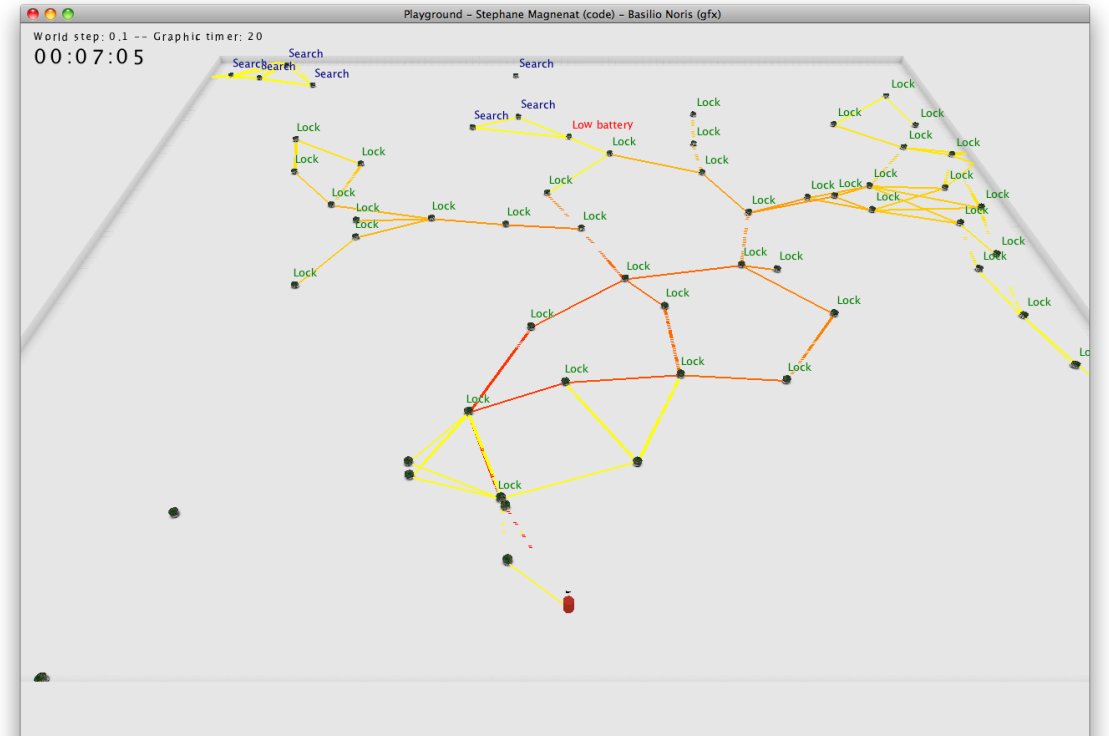


Figure D.5: Initiating robot in the *lowbatt* state makes first contact with a DLA connected agent

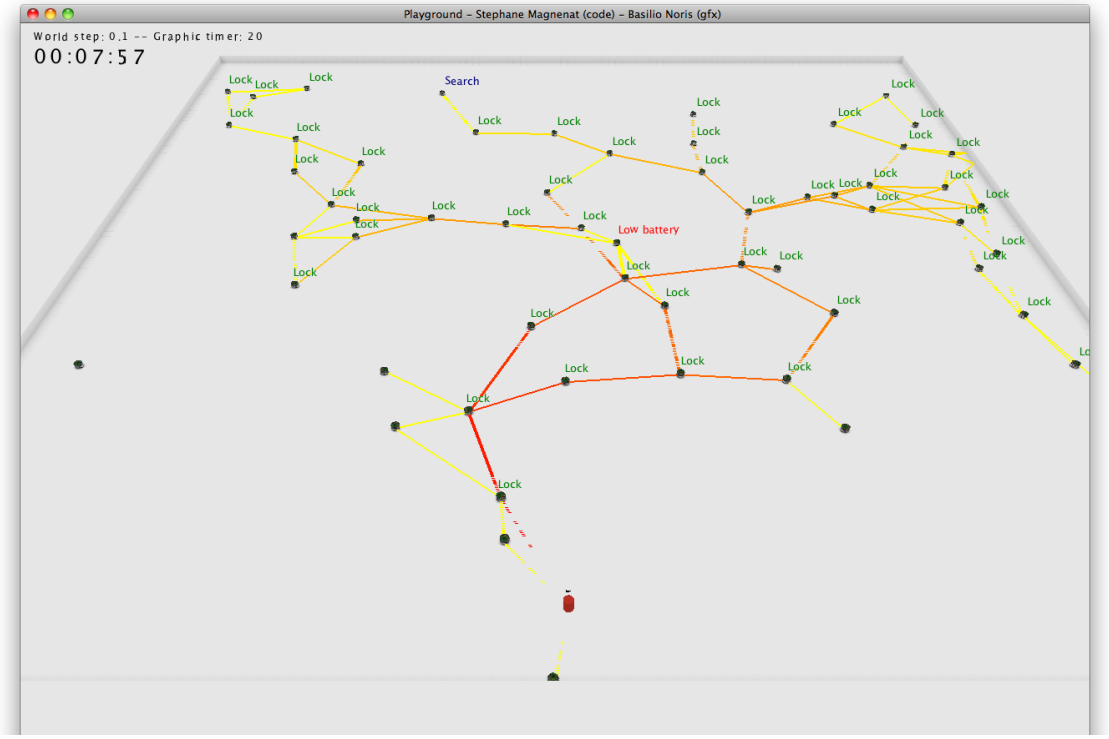


Figure D.6: *lowbatt* robot traversing the DLA structure toward the lowest frequency

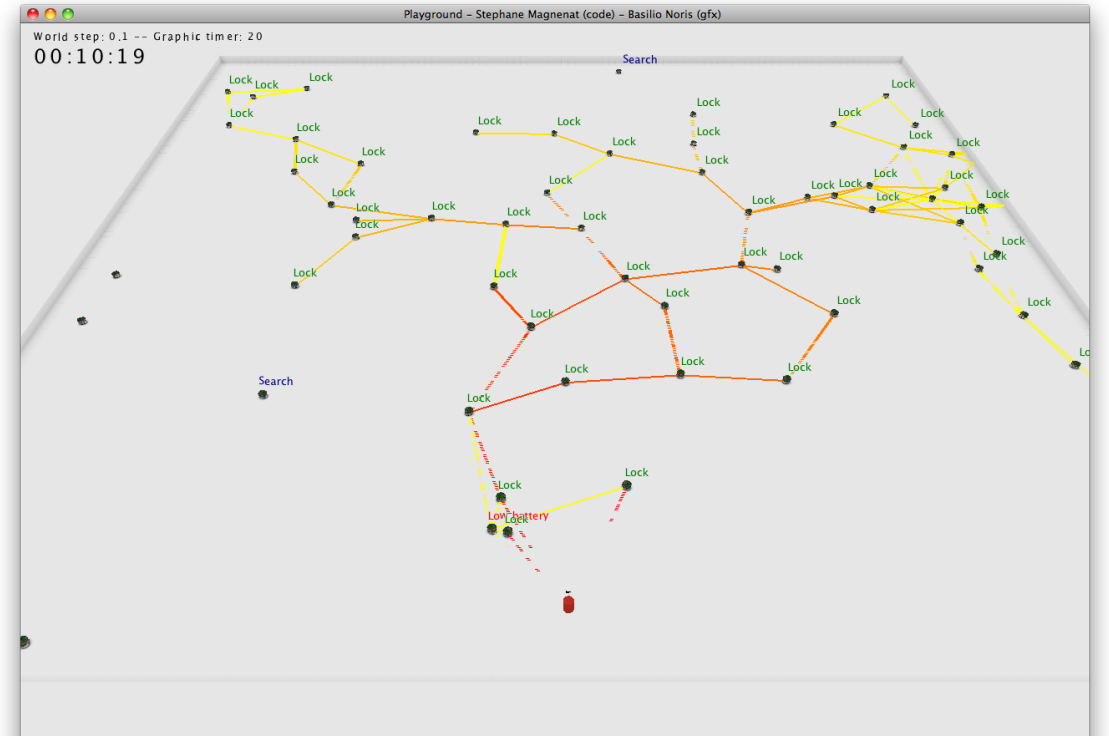


Figure D.7: *lowbatt* robot has almost arrived at the charging platform

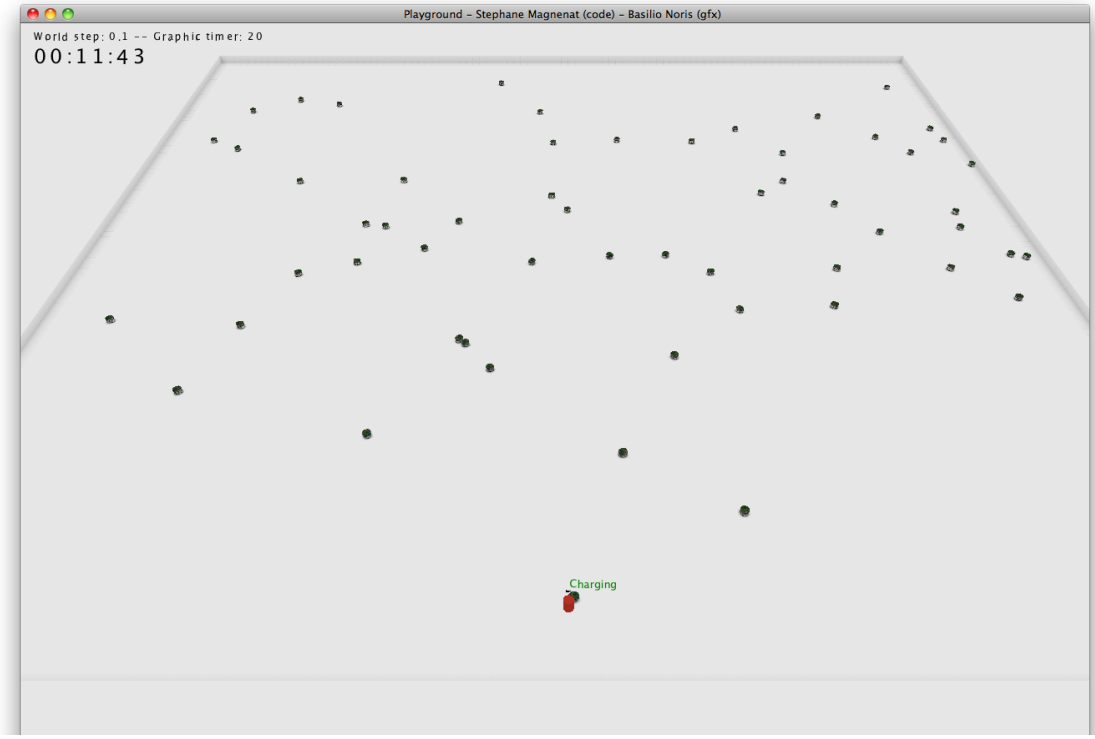


Figure D.8: Robot has begun recharging; audio beacon is disabled, releasing the swarm back to work

References

Bachmayer, R. and Leonard, N. (2002). Vehicle networks for gradient descent in a sampled environment, *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on* **1**: 112 – 117 vol.1.

Barán, B. (2001). Improved antnet routing, *SIGCOMM Comput. Commun. Rev.* **31**(2 supplement): 42–48.

Baran, B. and Sosa, R. (2000). A new approach for antnet routing, *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on* pp. 303 –308.

Beal, J., Correll, N., Urbina, L. and Bachrach, J. (2009). Behavior modes for randomized robotic coverage, *Robot Communication and Coordination, 2009. ROBOCOMM '09. Second International Conference on*, pp. 1 –6.

Bourke, P. (2004). Dla - diffusion limited aggregation. [Accessed May 26, 2010].
URL: *http://paulbourke.net/fractals/dla/*

Box, G. E. P. and Muller, M. E. (1958). A note on the generation of random normal deviates, *Annals of Mathematical Statistics* **29**(2): 610–611.

Braitenberg, V. (1984). *Vehicles: Experiments in synthetic psychology.*, MIT Press.

Burian, E., Yoerger, D., Bradley, A. and Singh, H. (1996). Gradient search with autonomous underwater vehicles using scalar measurements, *Autonomous Underwater Vehicle Technology, 1996. AUV '96., Proceedings of the 1996 Symposium on* pp. 86 –98.

Caro, G. D. and Dorigo, M. (1998). Antnet: Distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research* .

Chen, L. and Tong, W. (2009). A contactless charging platform for swarm robots, *Technical report*, University of Auckland.

Collier, T. and Taylor, C. (2004). Self-organization in sensor networks, *Journal of Parallel and Distributed Computing* **64**(7): 866–873.

Cyberbotics (n.d.). Webots user guide. [Accessed Mar 22, 2010].
URL: *http://www.cyberbotics.com/cdrom/common/doc/webots/guide/section6.6.html*

Dorigo, M., Birattari, M. and Stutzle, T. (2006). Ant colony optimization, *Computational Intelligence Magazine, IEEE* **1**(4): 28 –39.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents, *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* **26**(1): 29 –41.

Dressler, F. (2008). A study of self-organization mechanisms in ad hoc and sensor networks, *Computer Communications* **31**(13): 3018–3029.

Edelstein-Keshet, L. (1994). Simple models for trail-following behaviour; trunk trails versus individual foragers, *Journal of Mathematical Biology* **32**: 303–328. 10.1007/BF00160163.

E-puck Education Robot (2010). [Accessed Sep. 21, 2010].
URL: *http://www.e-puck.org/*

Gallager, R. G., Humblet, P. A. and Spira, P. M. (1983). A distributed algorithm for minimum-weight spanning trees, *ACM Trans. Program. Lang. Syst.* **5**(1): 66–77.

Hölldobler, B. and Möglich, M. (1980). The foraging system of pheidole militicida (hymenoptera: Formicidae), *Insectes Sociaux* **27**: 237–264. 10.1007/BF02223667.

Howard, A., Mataric, M. J. and Sukhatme, G. S. (2002). Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem, *6th International Symposium on Distributed Autonomous Robotics Systems*, pp. 299–308.

Kantor, G., Singh, S., Peterson, R., Rus, D., Das, A., Kumar, V., Pereira, G. and Spletzer, J. (2006). Distributed search and rescue with robot and sensor teams, *in* S. Yuta, H. Asama, E. Prassler, T. Tsubouchi and S. Thrun (eds), *Field and Service Robotics*, Vol. 24 of *Springer Tracts in Advanced Robotics*, Springer Berlin / Heidelberg, pp. 529–538.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system, *Commun. ACM* **21**(7): 558–565.

Magnenat, S., Rétornaz, P., Bonani, M., Longchamp, V. and Mondada, F. (2011). ASEBA: A Modular Architecture for Event-Based Control of Complex Robots, *IEEE/ASME Transactions on Mechatronics* **16**(2): 321–329.

Microchip Development Tools (2011). [Accessed May 26, 2010].
URL: *http://www.microchip.com/*

Miner, D. (2007). Swarm robotics algorithms: A survey.

Nakagaki, T., Tero, A., Kobayashi, R., Onishi, I. and Miyaji, T. (2008). Computational ability of cells based on cell dynamics and adaptability, *New Generation Computing* **27**: 57–81. 10.1007/s00354-008-0054-8.

Nakagaki, T., Yamada, H. and !'gota TÛth (2001). Path finding by tube morphogenesis in an amoeboid organism, *Biophysical Chemistry* **92**(1-2): 47 – 52.

Ogren, P., Fiorelli, E. and Leonard, N. (2004). Cooperative control of mobile sensor networks:adaptive gradient climbing in a distributed environment, *Automatic Control, IEEE Transactions on* **49**(8): 1292 – 1302.

Passino, K. (2000). Distributed optimization and control using only a germ of intelligence, *Intelligent Control, 2000. Proceedings of the 2000 IEEE International Symposium on* pp. P5 –13.

Passino, K. (2002). Biomimicry of bacterial foraging for distributed optimization and control, *Control Systems Magazine, IEEE* **22**(3): 52 –67.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application, *in* E. Sahin and W. M. Spears (eds), *Swarm Robotics*, Vol. 3342 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 10–20.

Şahin, E., Girgin, S., Bayindir, L. and Turgut, A. E. (2008). Swarm robotics, *in* G. Rozenberg, T. Bäck, J. N. Kok, H. P. Spaink, A. E. Eiben, C. Blum and D. Merkle (eds), *Swarm Intelligence*, Natural Computing Series, Springer Berlin Heidelberg, pp. 87–100.

Schmickl, T. and Crailsheim, K. (2007). A navigation algorithm for swarm robotics inspired by slime mold aggregation, *in* E. Sahin, W. Spears and A. Winfield (eds), *Swarm Robotics*, Vol. 4433 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 1–13.

Schweitzer, F., Lao, K. and Family, F. (1997). Active random walkers simulate trunk trail formation by ants, *BioSystems* **41**(3): 153–166.

The Network Simulator - ns-2 (2010). [Accessed June 16, 2010].
URL: *http://nsnam.isi.edu/nsnam/index.php/*

Witten, T. A. and Meakin, P. (1983). Diffusion-limited aggregation at multiple growth sites, *Phys. Rev. B* **28**(10): 5632–5642.