Investigation and Implementation of Biologically Inspired Flocking Behaviour in Swarm Robotics

Helmi Fraser

H00152077

An Honours report submitted for the degree of:

BEng in Robotics, Autonomous and Interactive Systems

Heriot-Watt University

Supervisor: Dr. Matthew W. Dunnigan

Declaration

I, Helmi Fraser

State that this work submitted for assessment is my own and expressed in my own words. Any uses made within it of works of other authors in any form (eg. ideas, figures, text, tables) are properly acknowledged at their point of use. A list of the references employed is included.

Signature:

Helmi Fraser

Acknowledgements

I would like to thank my supervisor, Dr Matthew Dunnigan, for providing me with project related support and advice. I would also like to thank Mohamed, my library friend, for lending me his industry experience and advising me on valuable software engineering practices.
Last but certainly not least, I would like to thank my loving girlfriend, Victoria, for keeping me motivated and focused and without whom my time as an undergraduate student would not have been the same.

Synopsis

This project investigates the behaviour and mechanics of natural swarm systems such as social insects, schools of fish and flocks of birds, and applies this to a group of robots. Purely through robot-robot interactions and without any direction from a supervisor, this swarm of robots is capable of quick self-organisation in order to aggregate and move as a cohesive unit.

Imitating nature in this manner is useful. Owing to its decentralised nature, a swarm system possesses a few desirable traits such as adaptation, fault tolerance and scalability. These properties can be useful if applied to an artificial system, leading to more robust engineering designs.

Contents

Chapter 1

Introduction

Nature has shown that it is often beneficial for animals to live and travel with groups of similar animals. In predatory social animals, they are often called packs and in prey animals the nomenclature varies from animal to animal, from flocks to herds, to schools. Indeed, humans are not exempt from this, with early humans coming together to form sizeable groups.

It is thought that the reason for this is that it is evolutionarily advantageous. For example, predators hunting in a group allow the hunting of far bigger prey animals than themselves, which provides an easier source of sustenance at the cost of being forced to share meals. [1] Conversely, other animals - like birds, fish, sheep and so on - group together in an effort to avoid such predations. These animals have developed antipredator adaptations through evolution because there is safety in numbers. Acting as a unit can provide protection and decrease the likelihood of an attack or fatality.

As can be seen, the concept of *flocking* and group behaviour is highly beneficial within natural systems. If this can be understood fully and leveraged for artificial systems, it can bring with it many of the benefits it has shown in nature. This is especially suited for implementation within robotics, specifically the field of *swarm robotics*.

Swarm robotics is a relatively new field of multi-robotics, in which the aim is to co-ordinate a large number of robots in a decentralized manner, similar to biological systems found in nature. The crux of a swarm system is feedback: a

swarm cannot operate without some sort of feedback to its constituent individual agents. Systems may contain direct or indirect methods of feedback, either agent-agent communication or agent-environment-agent communication. In addition to this, swarm systems promote *scalability*, by emphasizing a large number of agents.

Where this project is concerned, flocking is used to demonstrate the applicability of biologically inspired techniques when applied to a swarm of robots. Flocking behaviour and the techniques that are used in its operation can be especially useful when working with a group of robots.

1.1 Aim

This project aims to model, implement and explore the swarming behaviour exhibited in biological systems such as social insects, birds, schools of fish and bacteria. This will be achieved by developing a robot controller that will operate on every individual agent within a simulated robot swarm. The resulting global behaviour will then be compared against those found in natural systems.

1.2 Objectives

This project's objectives are:-

1. To understand the theory and concepts behind a swarm system, be they natural or artificial, in order to apply them within an engineering design.

2. To develop a robot controller which exhibits *flocking* behaviour, based on the techniques of self-organising swarm systems.

3. To evaluate the developed system under dynamic environmental conditions and stressors.

1.3 Relevance

From an engineering perspective, mimicking natural systems could provide better solutions to a multitude of problems across various industries. One way that is gaining major interest from academics and the industry alike is the application of swarm robotics.

In a world where there exists an increasing reliance on "smart" products and artificial intelligence, distributed computing and distributed robotics is becoming more and more commonplace. In effect, the world is a multi-agent system and there exists a need for artificial systems to be able to effectively operate in it.

Swarming theory can be applied to any problem that would benefit from having multiple agents solving it, such as search and rescue, subsea and space exploration and autonomous vehicles.

The potential use of the implementation of these algorithms in a commercial application is plentiful. For example, foraging and dispersal techniques can see major use in optimizing both disaster relief/rescue and agriculture. Flocking techniques can be applied to autonomous vehicles - which are predicted to significantly increase in market share in the coming years - in order to improve collision detection and efficiency. Of course, real life systems will be more nuanced and will need further development, but a proof of concept in the lab is vital to developing an engineer's understanding of these technologies further.

Chapter 2

Literature Review

In this chapter, an analysis of previous work completed in the field of swarm robotics and biologically inspired engineering is provided, as well as an outline of the theoretical concepts concerning the techniques applied during this project.

2.1 Biologically Inspired Computation and Engineering

Nature has provided examples of outwardly complex biological systems which are often efficient, fluid and resilient to partial breakdowns. Colonies of ants are able to forage for food and build large, complex structures. Fireflies are able to synchronize their flashing with one another. Flocks of birds and schools of fish exhibit fluid and efficient group movement. In the majority of cases, nature has achieved these despite utilizing very little to no communication between individual creatures and in the absence of a higher level director or supervisor. The animals react only to environmental stimuli, either the strength or type of pheromones detected in the environment in the case of ants (termed *stigmergy*) or the positions of other individuals in the case of fish and birds. For example, it has been shown that over time, the velocity of a flock of birds converges to a state where each individual has equal velocity. [2]

This is defined as *emergent behaviour*, the rise of previously unpredicted and

complex behaviour through the interaction of simple rules.

Researchers have attempted to replicate this in an artificial system via various means. For example, Bojinov et. al (2000) demonstrated how simple, local sensory rules can produce complex and useful structures in metamorphic robots. [3] Metamorphic, or modular, robots are a relatively new concept which can be described as "a robot that is composed of modules that are all identical". [4] In their paper, Bojinov et. al explored the use of purely *local* rules and interactions to develop control algorithms which enabled a Proteo robot to accomplish some otherwise difficult and complex tasks. Such tasks include forming three dimensional structures, locomotion and dynamic adaptation.

Previous work on this front has utilised an exact, a-priori definition of the configuration of the robot, calculated via algorithms that use heuristics. [5] These algorithms all require knowledge of the problem, the environment and the desired shape before hand, which in a real world setting is not always feasible. What makes the work of Bojinov et. al interesting is that their algorithm does not aim to produce a pre-defined shape, rather it aims to produce a shape with the desired *properties* needed to solve the problem at hand. In this way, the algorithm produced is robust to unknowns, noise or changes in the environment, in a similar manner to how animals can adapt to changes in their environment.

Taking advantage of the solutions to problems that nature has already solved and applying these in order to tackle similar problems in industry can be of great benefit to a engineer. While this effect can be seen in a wide range of fields, perhaps the biggest effect of biologically inspired engineering can be seen in the field of medical rehabilitation or human augmentation.

A particular area of interest to researchers and industry alike is the development of devices that can assist the lower body in performing work. Harvard University's Wyss Institute have developed several iterations of an assistive wearable robot - an exoskeleton - that can aid in load carrying, performance enhancing and even some medical applications. [6] Where their approach differs is the use of textiles in place of more rigid materials such as aluminium or carbon. In this way, they avoid the problem of having imperfectly

aligned rigid links interfering with the movement of biological joints and adding inertia to the legs. Inertia can significantly increased the metabolic cost of moving the legs, an 8% increase per kilogram at the legs versus a 1-2% increase per kilogram at the waist. [6] The Wyss Institute's earlier design relied on penumatic tubes which support the movement of the joints in the legs. Compared to more rigid designs, this provides a lightweight and low profile system with little effects on a user's kinematics. However, this comes at the cost of only being able to supply tensile forces and a significant reduction in maximum torques (1-10 times biological normal for rigid systems vs 0.1-1 times biological normal for soft systems). The most recent design operates on a similar principle, however it eschews pneumatics in favour of Bowden cables actuated by motors. This further reduces bulk, as compressed air is no longer needed and the cables are of a lower profile than pneumatic muscles.
As shown, applying biologically inspired techniques has to date been largely successful, with promising results from both academia and industry.

2.2 Artificial Intelligence and Artificial Life

One of the most expansive fields in computer science, artificial intelligence (AI) has decades of history and continues to be a hotly researched topic for academics and industry professionals alike. The field is defined as the study of intelligent agents, however where it differs from other attempts to understand intelligent agents (such as psychology or philosophy) is that AI seeks to construct them too. [7]
It is a common misconception of artificial intelligence that it is the search and development of purely "human-like" behaviour and thinking. Indeed, countless stories and movies portray this as fact. However, while the development and research into so called "strong AI" is prevalent and cutting edge, there are many domains within AI and a huge array of techniques that can be applied.
A notable technique in AI research is the artificial neural network (ANN). This strives to model the way a biological brain performs a given task, such as vision,

hearing and actuation. An artificial neural network is composed of multiple simple computation nodes (neurons), each passing an input (or a combination of inputs) through an activation function which determine the value of the output. It is through the interconnected nature of these neurons that give rise to the computational power of a neural network. [8]

ANNs have been used extensively since their conception, with applications ranging from function approximation to operating as a robot controller.

One of the major reasons for continued AI development is to act as a decision making tool that is many times faster than a human could ever achieve. With a rise in greater computational power coupled with a decrease in hardware size and cost, AI has become an increasingly ubiquitous tool that many companies and researches are utilising. For example, Amazon's development of transactional AI - which analyses the shopping habits of customers and uses this to make predictions on future purchases - has allowed the company to generate a substantial amount of profit. Another use of AI is in the operation of autonomous vehicles, which greatly benefits from a computer's fast reaction time and inability to become tired, bored or lose focus. Google, Tesla and a host of other technology and car manufacturers are investing significant capital into realising a safe autonomous system. For example, Tesla's Model S has all of the necessary hardware (such as sensors and actuators) built in for it to function as a fully autonomous vehicle, it just requires the correct software implementation. [9]

Differing from artificial intelligence, is the field of artificial *life* (AL, or ALife), which concerns itself with the study of systems relating to biological life and it's processes through computer "simulation, robotics, or biochemistry". [10] First coined in 1986 by theoretical biologist and one of the founders of the discipline Christopher Langton [11], the field differs from AI. Where AI seeks to study intelligence, AL seeks to study the fundamental processes of living systems.

There are three approaches to AL research: "soft" (software based approaches such as simulations), "hard" (hardware based approaches such as robotics) and "wet" (the application of biochemistry). Each approach has it's own pros and

cons, though many researchers are turning to simulation based studies to reduce time and the possibility of hardware issues.

Vehicles: Experiments in Synthetic Psychology by Valentino Braitenberg, a book acclaimed among roboticists and social scientists alike, describes a thought experiment in which simple vehicles (now known as Braitenberg vehicles) can "exhibit behaviors akin to aggression, love, foresight, and optimism". [12] In its most simple configuration, a vehicle has primitive sensors (often light sensors) connected directly to the wheels. The vehicle then does nothing more than move its wheels for a given sensor input. Therefore, depending on the wiring of the sensors to the actuators, the vehicles exhibit different behaviours, such as light following or light avoidance all with varying speeds and directions.

As computer simulation technology improved, Braitenberg's thought experiment was validated as true. In an environment with many sources of stimulus, the vehicles can produce complex behaviour, which despite its simple internals can appear as if under intelligent control.

2.3 Swarm Intelligence and Behaviour

In 1989, Gerardo Beni and Jin Wang coined the termed *swarm intelligence* in their paper *Swarm Intelligence in Cellular Robotic Systems* and Beni has authored several books on the subject of swarm robotics.

Swarm intelligence is defined as the collective behavior of self-organizing and decentralized natural or artificial systems, arising through the interactions between simple agents. [13] Beni states that the term was applied due to the similarities between his simulated cellular robots and social insects, notably: "decentralised control, lack of synchronicity, simple and (quasi) identical members". Another important aspect of swarm systems is the concept of self-organization, which was encapsulated by Bonabeau et al (1999): "self-organisation is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components". [14] What this means is that the rules which govern the

interactions within a self-organising system are executed using purely local information and without knowledge of any global/system wide stimulus. This means that any pattern that arises at a global level is an emergent property of the system, as opposed to something dictated by a higher level supervisor.
For example, a single ant operating by itself does not exhibit behaviour more complex than simply wandering around the immediate environment, lost. However, an *ant colony*, operating as one system are capable of complex feats that no individual ant is capable of performing, despite the lack of an overall co-ordinator. [15] A colony can expand or contract its territory, which will effect the patrol routes of guard ants, which in turn effect the routes of the foragers and so on. In this manner, an ant colony is said to be largely self-organising and the patterns emergent.
For this reason, ants are extremely interesting to researchers. Ants communicate via a process called *stigmergy*, in which they detect the pheromones left behind by other ants and based on this, decide what they should next do. This process can be replicated within an artificial system and has given rise to optimisation techniques based on environmental stimuli, such as Ant Colony Optimisation.
As can be seen, perhaps the most defining characteristic of a swarm system is emergent behavior.
One of the most prominent examples of emergent behavior in the field is the development of the artificial life program *Boids*, developed by Craig Reynolds in 1986. This provides a realistic simulation of the flocking behavior of birds through the interaction of agents present in the system.
At its most basic level Reynolds, in his paper *Flocks, herds and schools: A distributed behavioral model*, defined the rules followed by all individual agents as [16]:

1. *Separation*: move to avoid crowding local flockmates

2. *Alignment*: steer towards the average heading of local flockmates

3. *Cohesion*: steer to move toward the average position of local flockmates

This concept of locality means that agents do not react to the system as a whole, but only to agents within a certain radius - a neighborhood. This could model a creatures limited perception of its surroundings or its section of the environment in which to draw stimuli from.

Originally developed as a means to provide natural movements of various animals in computer graphics, the *Boids* model has now seen extensive use in a multitude of applications from games development to military vehicles, task optimization and swarm robotics. It is this set of rules that the robot controller of this project will be based upon.

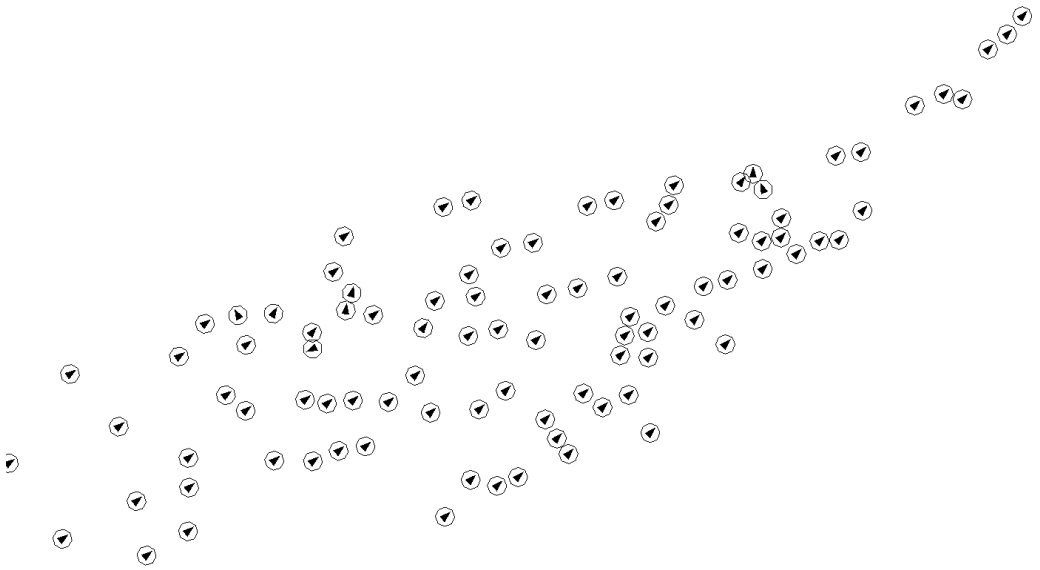


Figure 2.1: A demonstration of boids

Chapter 3

Methodology

Within this chapter, an overview of the tools utilised during the course of the project is provided.
After a thorough investigation during the early stages of the project, it was decided that the project shall progress as a simulation only model. This was done for a number of reasons:-

1. A simulation based approach negated the possibility of hardware failure, errors or major discrepancies between units. This is especially relevant for this project, where a swarm can be comprised of a large number of units, as the focus can lie with fixing software issues instead of both hardware and software.

2. A simulation based approach is highly flexible, effective and time efficient. It enables rapid testing of various experimental parameters.

3. A simulation based approach is not restricted to available lab space/experimental environments. It is possible to alter the swarm's environment at will, for example if a larger arena or various objects are needed.

4. A simulation based approach is cost efficient. There are no monetary restraints with regard to adding more robots to the swarm, or costs for repairs on broken robots/replacement parts.

Moreover, as the utilised software package directly simulates real robotic platforms, there will be no need for a major overhaul of the codebase for it to function on a real robot.

3.1 Hardware

Though the project is conducted entirely in simulation, the agents are based on a real robotic platform, the e-puck (or E-puck, epuck, Epuck). For this reason, an understanding of the capabilities and applications of the platform is essential.

3.1.1 E-puck robotic platform

Developed by Mondada et al. at the Autonomous Systems Lab of École Polytechnique Fédérale de Lausanne (EPFL), the e-puck (see Fig 3.1) is an open source, two wheeled, differential robot measuring roughly 7cm in diameter. [17] The e-puck was originally developed to allow students to address a wide range of engineering fields.

The platform features a wide range of sensing capabilities as standard (see Table A.1), with the additional option to expand through the use of top and bottom extension slots. (Fig.3.3, 3.4) These physical extensions can range from extra sensors, to additional controllers for more computational capabilities (such as an Arduino board).

The flexibility of the platform, coupled with low purchasing costs (owing to the open source nature of the hardware and the software) has thus led to swift adoption by robotics researchers worldwide.

After thorough research of other suitable robotic platforms (see 3.1.2), the e-puck was chosen as the most desireable base unit to create a swarm. This was due to several factors: -

1. The e-puck is flexible. Extensibility allows extra modules to be added if needed.
2. The e-puck's basic features already have the capabilities required of a swarm agent.
3. Its relative ease of use and good documentation will enable a quicker development cycle compared to other robots on the market.

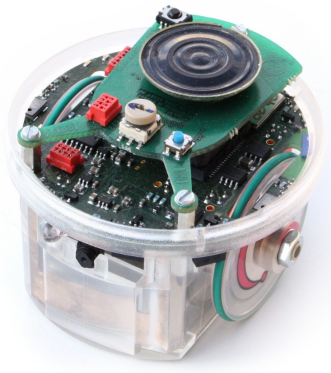


Figure 3.1: e-puck mobile robot

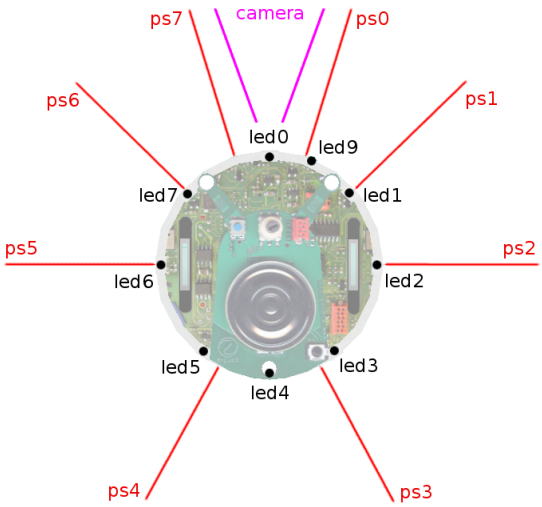


Figure 3.2: e-puck sensor positions, where ps0 - ps7 denote the positions of the IR sensors

The eight circumferential IR sensors that the robot possesses are crucial in order to exhibit swarm behaviour. These sensors are what allows the robot to "see" its immediate environment and without them the robot will be unable to react to its surroundings.

In addition to this, these sensors allow the use of an IR communication protocol alongside their normal function as proximity or ambient light sensors. In this way, the robots are capable of limited, short range communication with one another.

The e-puck also possesses various wired and wireless communications channels, used primarily to communicate with a host computer. Of note is the Bluetooth radio link, which implements a remote control protocol called the "BTcom protocol". This enables the robot to not only be driven wirelessly, but provides full remote access to the robot's features such as the camera's feed, LED states and sensor data. [17]

Another noteworthy application is the potential use of one robot's body LEDs signal to other robots its current state. For example, Christensen et al. demonstrated a decentralised system of robots that over time will synchronise their flashing with one another. [18]

The e-puck's small size, flexibility and relative low cost are things which would benefit a hardware based research approach.

As can be seen, the array of useful sensors and actuators that are available to the e-puck platform lends itself well to a swarm system application.

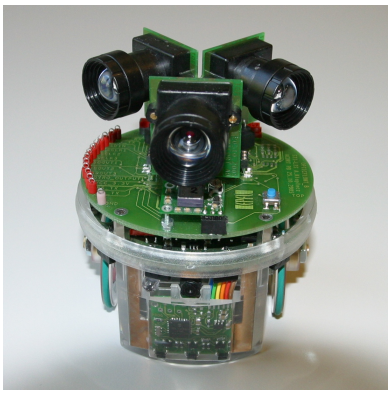


Figure 3.3: Flycam vision turret.

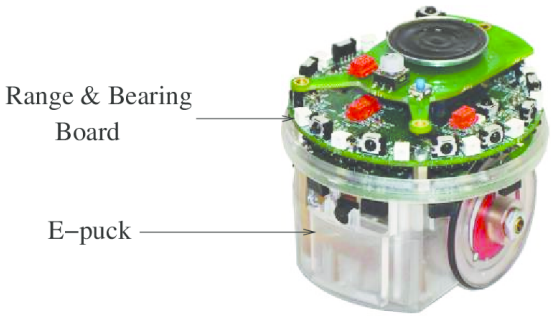


Figure 3.4: Range and bearing turret

3.1.2 Alternatives

The e-puck platform is not alone in the market of mobile, relatively small robots.

Another potential platform for the project is the Kilobot (Fig. 3.5) Developed by Hardvard University's Self-Organizing Systems Research Group and winner of the "African Robotics Network $10 Robot Challenge" [19], the Kilobot is a simple and low cost small robot. Envisaged to enable the testing of control algorithms on large swarms accessible to researchers, the Kilobot was designed to prioritise cost and ease of use. [20] To that end, each unit consists of only $14 (around £11) worth of parts, requires 5 minutes of assembly time and allows a single operator to control a large group with ease. The control is achieved using an overhead infra-red controller connected to a control station (a computer). This allows a user to broadcast a command or program and have it received by all robots in a swarm within a fixed amount of time, regardless of the number of robots.

Though extremely promising, this platform was not chosen simply for the fact that there was no simulated equivalent available. Also, though cheaper than the vast majority of platforms, the attractive price is only valid if bought or created in bulk.

K-Team's Khepera IV is also an attractive alternative (Fig. 3.6). The basic feature set of the Khepera IV far outstrips that of the e-puck, with the former housing a WiFi chipset, 4 IR ground sensors as standard, 5 sonar sensors, a run time of 5 hours and an extension ecosystem. In addition to this, the robot embeds a complete Linux operating system, allowing "almost any existing library" to be easily ported. [21]

This was not chosen as though it is technically superior, the extra functionality does not offer any substantial gains over the e-puck for this application. It is not open source, so not as widespread as the e-puck. In addition to this, the Khepera IV is significantly more expensive than the e-puck, at 2650 CHF (around £2140) for a single unit.

Bot'n Roll's ONE A unit was also considered. Aimed more toward beginners

rather than researchers, the ONE A is a differential wheeled robot with an LCD screen mounted on an Arduino programmable board. It comes with two forward facing IR sensors used for distance sensing. [22] This was not chosen as though it provided a few external connections for additional sensors, the extra sensors did not come as standard and has to be soldered manually. At 199 EUR, it also provided very little value for money compared with other platforms, and its simplicity is liable to be a hinderance.

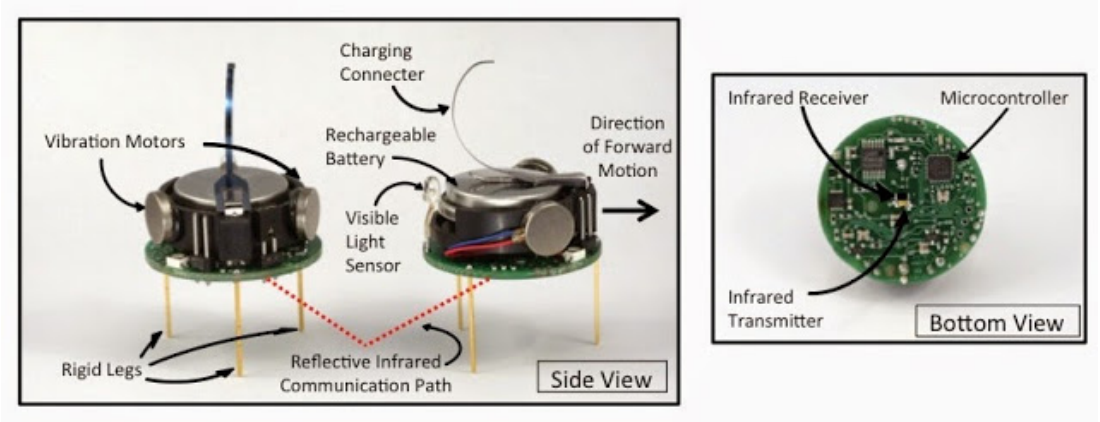


Figure 3.5: Kilobot

Figure 3.6: Khepera IV

3.2 Software

As the project is an entirely simulation based study, the utilised software tools are of paramount importance.

3.2.1 Webots simulator

Webots, developed by Cyberbotics, is a professional development and simulation environment for mobile robots. Cyberbotics is at the forefront of robot simulation, with Webots being used by over 1284 universities and research centres worldwide. [23]

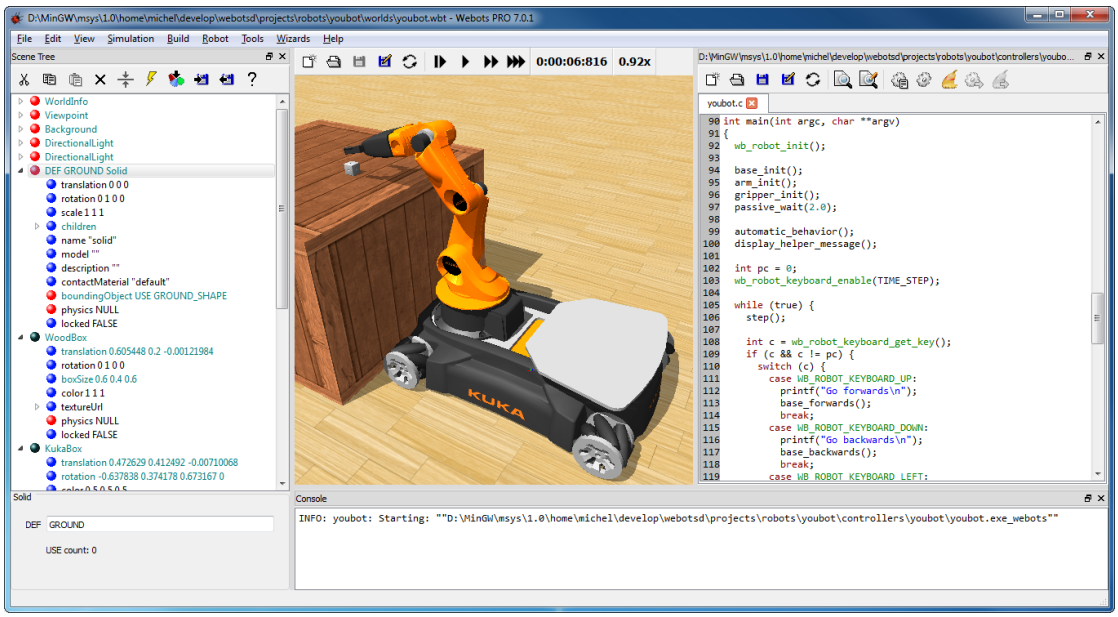


Figure 3.7: Webots simulator

Webots was chosen as the desired development environment as it allows quick creation of complex robotics setups, as well a streamlined way to set up a virtual operating environment. It features a built in integrated development environment for programming, should a user wish to use this. In addition to this, the simulator utilises the Open Dynamics Engine (ODE) in order to simulate rigid body dynamics and collision detection. This allows realistic physics to be accurately simulated in a virtual environment.
The simulator also allows cross compilation of code onto live robots, cutting

down development time if the user wishes to test on a live platform.
Another advantage afforded by Webots is that the distribution comes pre-packaged with a host of modifiable robot models, aimed at closely simulating any real life counterparts. Where this project is concerned, there already exists a virtual e-puck model. This e-puck is, for all intents and purposes, as functional as the hardware equivalent, and possesses all of the same sensors and actuators.
Conducting experiments in Webots is also streamlined. It features the ability to advance through a simulation step-by-step, at a time step defined by the user. This is useful for close analysis of robot behaviour. The virtual camera can be oriented in three dimensions and enables the taking of pictures and videos, also useful for experimental analysis and documentation.
There exists C, C++, MATLAB, Java and Python libraries for the functions of robot controller programs and Cyberbotics' documentation of this API aids in speeding up the development cycle.

Concepts

To fully understand Webots, some basic underlying principles have to be outlined. A simulation is defined as a *world*, containing *models* (which includes static objects and robots), and *controllers* are executed on these models if they are able to.
Each *world* is defined by a .wbt file, which contains all of the information about that simulation. Within this file, everything from the number and type of robots to the amount of ambient light present are defined.
A robot *model* is defined by a .proto file, which contains information regarding that type of robot. For example, this file could contain the positions of and number of sensors a particular robot has.
A robot *controller* is the program that executes on the robot itself. This is where the behaviour of the robot is defined and is written in a programming language chosen from those Webots supports. In the case of this project, C++ was chosen.

3.2.2 C++ programming language

Released as a commercial product in 1985, C++ was developed by Danish computer scientist Bjarne Stroustrup. Stroustrup developed the language with the aim of integrating object-oriented programming into the C language. This was viewed as an improvement to the C language, making the language easier to use for larger software development projects without sacrificing the low-level memory management, speed and portability that C is renowned for. [24]
C++ is in use in many different industries and has a very widespread adoption. It is currently positioned in third place on the TIOBE Programming Community index, which is a measure of the popularity of programming languages worldwide. [25]
It was decided that C++ was to be used for this project for the following reasons:-

1. The language is fast. C++ performs calculations relatively quickly in comparison with other languages.

2. C++ allows a high degree of user code optimisation, which could improve performance especially in resource limited applications.

3. C++ is object-oriented. This aids in the software design process by enabling a higher level of abstraction.

4. C++ is well documented. With over 30 years of history and frequent revisions, most errors and issues have been encountered and solved in the past.

5. It is suitable for embedded and robotics applications. It is currently in widespread use within the robotics industry, with C++ being one of the main supported languages of the ROS (Robot Operating System) framework.

Chapter 4

Implementation

This chapter details the design process of the swarm system, and the approach used when tackling the problem of multi-agent interactions.
Firstly, an outline of the problem representation is shown and then the structure and reasoning behind the software implementation.

4.1 Design

As shown in section 3.1, the standard configuration of the e-puck has all of the required capabilities to implement swarming behaviour.

4.1.1 Simulation Setup

Firstly, a new project was created in Webots.
This defined the virtual operating environment and the number of robots in the world, as well as the world parameters. Shown in Figures 4.1 and 4.2 are the parameters used for the simulation, all other parameters were kept as the default.
The simulation consisted of a bounded, rectangular arena of size 3 metres by 3 metres. Within this arena, there are two wooden boxes (one of size 0.2m x 0.2m x 0.2m and the other of size 0.5m x 0.5m x 0.5m) and a hollow cylinder (of height 0.3m, radius 0.2m and thickness 0.1m). In the setup shown, only 9

e-pucks are utilised, placed in random locations around the arena, see Figure 4.3.

Basic time step	32ms
Frames per second	60
North direction	-10m
Objects	3
e-pucks	9

Figure 4.1: World parameters

Length (x)	3 metres
Breadth (y)	3 metres
Wall thickness	0.01 metres
Wall height	0.1 metres

Figure 4.2: Arena parameters

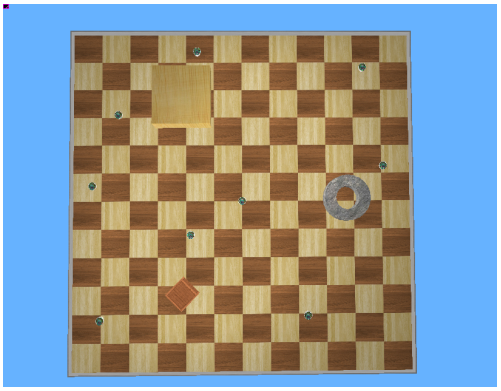


Figure 4.3: Arena

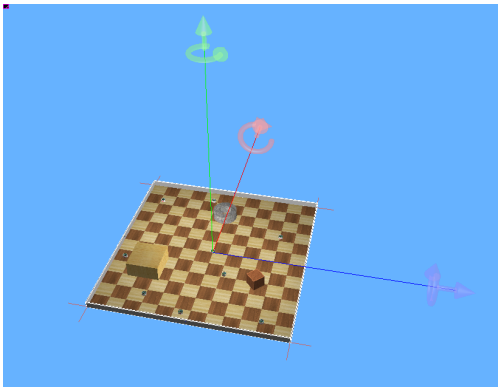


Figure 4.4: Arrows denote the positive world axes directions

Figure 4.4 shows the positive directions of the x, y and z axes. Red denotes positive x, green denotes positive y, blue denotes positive z. Hence the north direction has been set to (0, 0, -10).

The arena was setup in the manner shown in Figure 4.3 after careful consideration of evaluation parameters:-

1. Obstacles of various shapes and sizes tests the ability of the swarm and its agents to avoid obstacles.

2. The positioning of the large box and cylinder creates narrow corridors against the arena walls and this tests the swarm's ability to navigate confined spaces.

3. Objects will block line of sight (and therefore sensing) to other robots and this tests the ability of the swarm to form splinter groups.

4. Open spaces will test the ability of these splinter groups to amalgamate and form a singular flock.

Furthermore, this project differs from other demonstrations of swarm behaviour due to the fact that there is no overall destination for the flock to head towards (such as a user-defined point). Any resulting shapes or patterns formed by the swarm are entirely emergent.

4.1.2 e-puck Model Modifications

Though the model provided by Webots was sufficient for this project's intended aplication, some modifications were made to simplify the simulation and aid in the collecting of experimental data.
As stated in subsection 3.1.1, the e-puck is capable of robot-robot communication utilising the IR sensors as receivers and emitters. However, the implemented sensors (Fig. 3.2) are purely proximity and light sensors and does not detect IR light from other robots, only the reflection of its own signal. Thus, the simulated e-pucks are incapable of using exactly the same method of IR communication as real e-pucks.
This problem can be overcome through the use of Webots' supplied "Emitter" and "Receiver" nodes. These are additional modules that can be added to a model and simulate a method of communication. In this case the desired method of communication is through the medium of infra-red, though radio is possible.
One solution was to add an emitter and receiver at the positions of each proximity sensor. The emitter and receiver can then be limited to match the

range and aperture of the IR sensors. This would closely emulate the way a real e-puck would use the sensors to communicate. However, this would be computationally expensive to execute, as the simulator would have to process at worst eight incoming/outgoing messages per e-puck and over a large swarm could cause issues with memory and performance.
A better solution was implemented. Instead of eight emitter/receiver pairs, only one is utilised, placed on top of the centre of the robot. However, this emitter/receiver has a wide aperture of 360°. In this manner, the simulated robot can communicate with any neighbouring flockmates in a similar way to real e-puck, while significantly reducing the computational overhead of the simulator.
A necessary modification was the addition of a compass module. This simulated a real e-puck's "Range and Bearing turret" (Fig. 3.4), which enabled the e-puck to obtain a localised bearing.
Another modification made was the addition of a GPS unit to each e-puck. This is not utilised during the course of exhibiting flocking behaviour, rather it is added as an experimental tool to collect data on the trajectories of every e-puck in the world.

4.2 Code Implementation

The approach taken to implementing the desired swarming behaviour can be divided into two processes: abstracting the higher level desired behaviour, then implementing the required lower level dependencies.
A full code listing can be found in Appendix B.

4.2.1 Libraries and dependencies

In order to keep system complexity as low as possible, the use of any external C++ libraries were minimised. In this project, only the default Webots function library and the C++ standard library were utilised. This also aids in keeping program compatibility across different computers and operating systems.

The use of the *libircom* library for infra-red communication using the proximity sensors was investigated at the early stages of this project, though this was eschewed due to the problems outlined in section 4.1.2.

4.2.2 Swarming Algorithm

The algorithm can be split up into three main processes: receive data from the environment and other agents (proximity sensors and messages), process the received data and decide on the next course of action, execute the next course of action.

Figure 4.5 shows a highly abstracted procedure of the robot controller. Subsequent flow charts will detail the lower level processes within each abstracted process.

Overall procedure

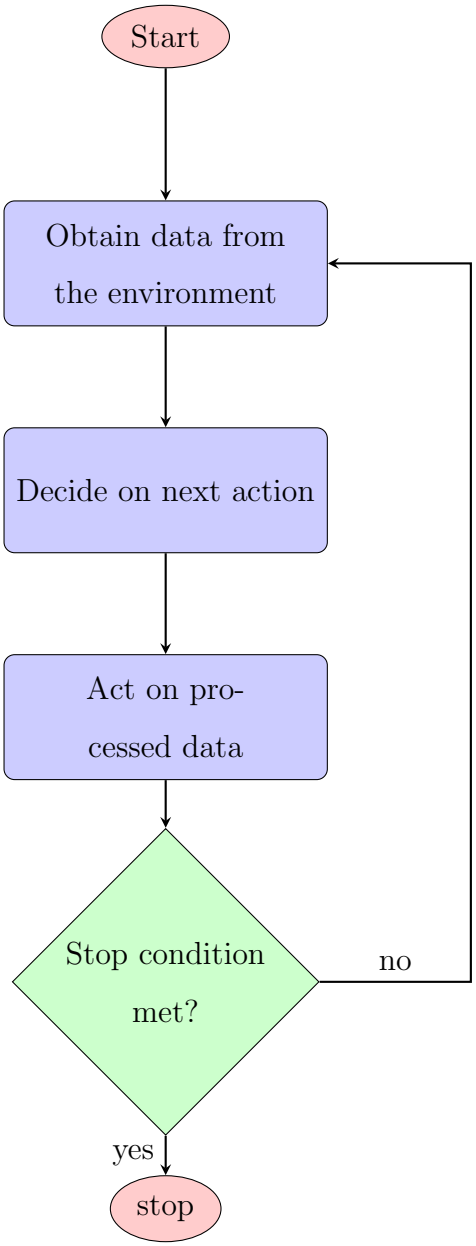


Figure 4.5: Simplified algorithm procedure

Obtaining data

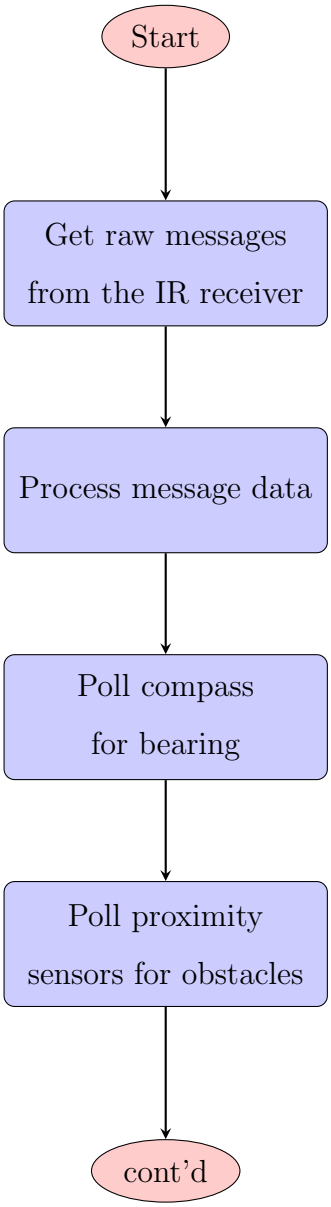


Figure 4.6: First process

Listing 4.1: Message protocol

```
1  void Swarm::getReceiverData() {
2      Receiver *copy = (Receiver *)malloc(sizeof(Receiver));
3      myData.orientationString = {};
4      robots = roundNum((receiver->getQueueLength() + 1) / 2);
5      for (int i = 0; i < 100; i++) {
6          signalStrength[i] = 0;
7      }
8      for (int k = 0; k < receiver->getQueueLength(); k++) {
9          data = (char *)receiver->getData();
10         signalStrength[k] = (double)receiver->getSignalStrength
               ↪ ();
11         for (int n = 0; n < 3; n++) {
12             emitterDirection[k][n] =
13                                     receiver->getEmitterDirection()[
                                           ↪ n];
14         }
15         memcpy(copy, data, sizeof(Receiver));
16         myData.orientationString[k] = (char *)copy;
17         receiver->nextPacket();
18     }
19 }
```

Listing 4.1 shows the procedure used to convert a raw message from another robot, into a string type. This function also processes a number of global variables: the number of detected robots in the neighbourhood, the signal strength (and thus distance) of the message signal and the direction of the message signal. Having a euclidean distance and an angle allows the robot to obtain a vector to the message origin.

Listing 4.2: Message processing

```
1  void Swarm::processReceiverData(std::array<std::string,
       ↪ ARRAY_SIZE> data) {
2      myData.received = false;
3      for (unsigned int i = 0; i < ARRAY_SIZE; i++) {
4          myData.orientationDouble[i] = 0;
5      }
6      try {
7          std::regex re("[*0-9*.*0-9*]+");
8          for (int i = 0; i < robots; i++) {
9              std::sregex_iterator next(data[i].begin(), data[i].
                   ↪ end(), re);
10             std::sregex_iterator end;
11             while (next != end) {
12                 std::smatch match = *next;
13                 std::string match1 = match.str();
14                 myData.orientationDouble[i] = atoi(match1.c_str())
                       ↪ ;
15                 if (myData.orientationDouble[i] >= 352) {
16                 myData.orientationDouble[i] = 0;
17                 }
18                 next++;
19                 if (next == end) {
20                     myData.received = true;
21                 }
22             }
23         }
24     } catch (std::regex_error &e) {}
25 }
```

Listing 4.2 details the method by which a received message of type string is processed. This was achieved by iterating through an array containing the messages from every robot and using a regular expression to parse these strings.

The regular expression, in this case, searches for a decimal number with trailing digits. If a match is detected, the number is converted from a string type into a double type and saved in a global variable (line 14).

Listing 4.3: Getting orientation data

```
1 void Swarm::saveCompassValues() {
2     const double *compassVal = compass->getValues();
3     for (int i = 0; i < 3; i++) {
4         currentOrientation[i] = compassVal[i];
5     }
6 }
```

Listing 4.3 shows how the robot can access the data from the compass module in order to obtain its orientation in the world. The data returned from the compass module is a three dimensional vector toward the north point within the simulation.

Listing 4.4: Getting proximity information

```
1 void Swarm::distanceCheck() {
2     for (int i = 0; i < 8; i++) {
3         ps_values[i] = checkDistanceSensor(i);
4     }
5     right_obstacle = (ps_values[0] > PS_THRESHOLD) || (
      ↪ ps_values[1] > PS_THRESHOLD) || (ps_values[2] >
      ↪ PS_THRESHOLD);
6     left_obstacle = (ps_values[5] > PS_THRESHOLD) || (ps_values
      ↪ [6] > PS_THRESHOLD) || (ps_values[7] > PS_THRESHOLD);
7     front_obstacle = (ps_values[0] > PS_THRESHOLD) & (ps_values
      ↪ [7] > PS_THRESHOLD);
8     back_obstacle = (ps_values[3] > PS_THRESHOLD) & (ps_values
      ↪ [4] > PS_THRESHOLD);
9 }
```

Listing 4.4 demonstrates how the robot can access the data from its circumferential proximity sensors. In this case, the sensors are discretized into

ranges, depending on the position of the sensor on the robot. If the value of a sensor within that range is above a certain user defined threshold, a boolean value is set.

Decision and actuation

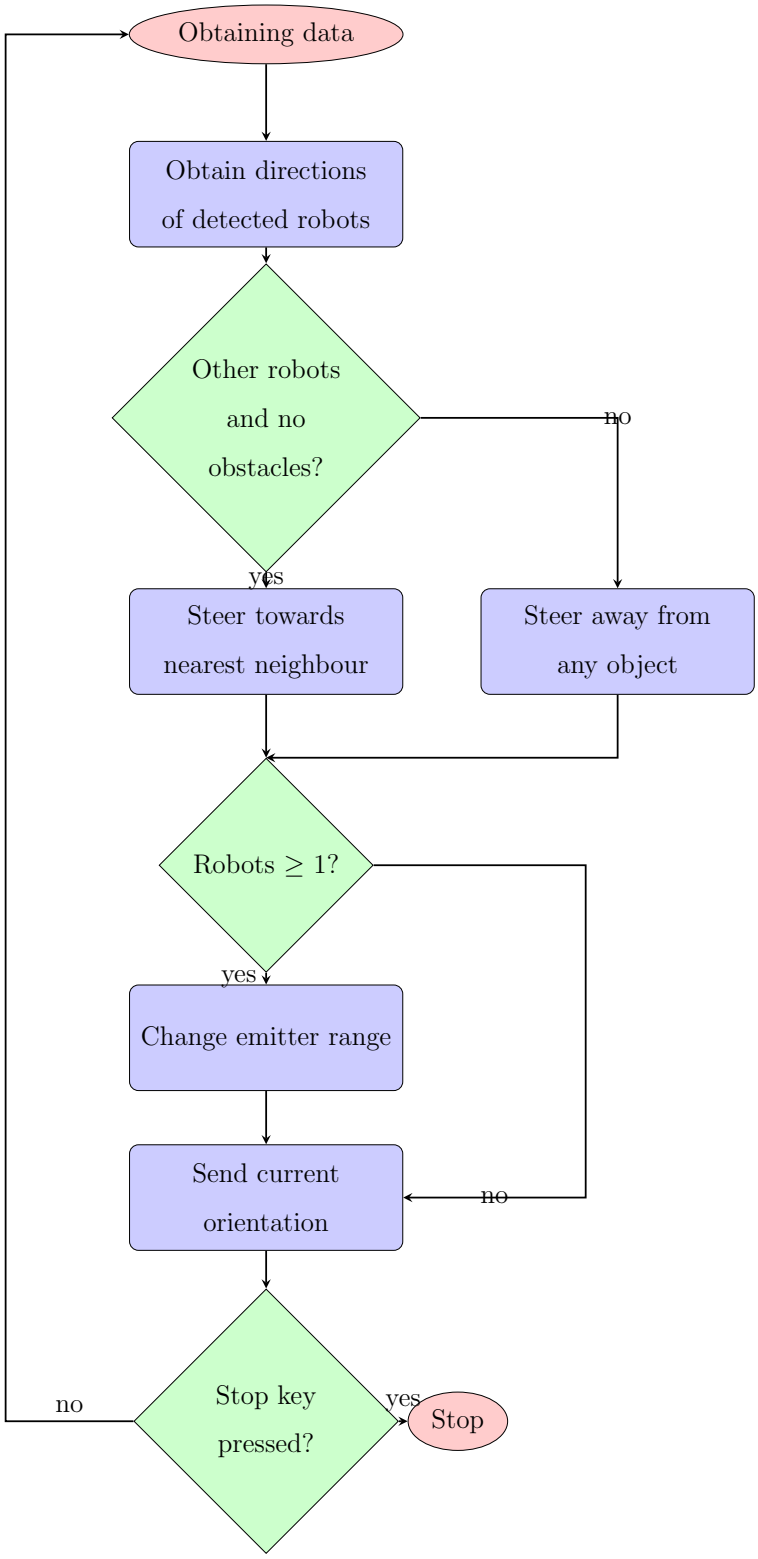


Figure 4.7: Second and third process

Listing 4.5: Calculating target vector

```
1  int index = std::distance(signalStrength, std::max_element(
       ↪ signalStrength, signalStrength + sizeof(signalStrength)
       ↪ / sizeof(double)));
2
3  std::array<double, 3> direction = {0, 0, 0};
4
5  if (robots > ALIGN_THRESHOLD) {
6      for (int k = 0; k < 3; k++) {
7          direction[k] = emitterDirection[index + 1][k];
8      }
9  } else if (robots == 1) {
10     for (int k = 0; k < 3; k++) {
11         direction[k] = emitterDirection[index][k];
12     }
13 }
14
15 double nearestNeighbour = computeVectorAngle(direction);
```

Listing 4.5 demonstrates the method by which the robot can determine the direction of the neighbour it should steer toward. This is achieved by returning the vector of its second nearest neighbour in the case that the number of robots detected is greater than a pre-defined value, or the nearest neighbour in the case that only one robot is detected.

Listing 4.6: Action for next step

```
1 if ((robots >= ALIGN_THRESHOLD) && (!left_obstacle) && (!
    ↪ right_obstacle) && signalStrength[index] < 10) {
2     setLEDs(0);
3     adjust(nearestNeighbour);
4 } else {
5     setLEDs(1);
6     objectDetection(1.0);
7 }
```

Listing 4.6 is a crucial step in the algorithm. This process defines the behaviour that the robot will exhibit at the next time step. If a certain condition is met (there are sufficient robots in the neighbourhood and no obstacles detected and the nearest neighbour is far way), the robot steers towards the direction calculated in Listing 4.5. Else, the robot will exhibit obstacle avoidance. The parameter for *objectDetection* (line 6) is simply a speed multiplier i.e. exhibit obstacle at 1.0*normal speed. In each case, the robot either switches its circumferential LEDs on or off and this aids the user in determining the state of the robot during simulation.

Listing 4.7: Adjust heading

```
1  void Swarm::adjust(double angle) {
2     double left_speed = WHEEL_SPEED;
3     double right_speed = WHEEL_SPEED;
4
5     if (((angle > (360 - ALIGN_ERROR)) & (angle < 360)) | ((
        ↪ angle > 0) & (angle < ALIGN_ERROR))) {
6         right_speed = left_speed = WHEEL_SPEED;
7     } else if ((angle > 270) & (angle < (360 - ALIGN_ERROR))) {
8         left_speed = (1 + (ALIGN_ERROR + angle - 360) / (90 -
            ↪ ALIGN_ERROR)) * WHEEL_SPEED;
9         right_speed = WHEEL_SPEED;
10    } else if ((angle >= 180) & (angle < 270)) {
```

```
11          left_speed = ((angle - 270) / 90) * WHEEL_SPEED;
12          right_speed = WHEEL_SPEED;
13      } else if ((angle >= 90) & (angle < 180)) {
14          right_speed = ((90 - angle) / (90)) * WHEEL_SPEED;
15          left_speed = WHEEL_SPEED;
16      } else if ((angle >= ALIGN_ERROR) & (angle < 90)) {
17          right_speed = (1 - (angle - ALIGN_ERROR) / (90 -
                ↪ ALIGN_ERROR)) * WHEEL_SPEED;
18          left_speed = WHEEL_SPEED;
19      }
20
21      if (left_speed > 1000) {
22          left_speed = 1000;
23      }
24
25      if (right_speed > 1000) {
26          right_speed = 1000;
27      }
28
29      move((int)left_speed, (int)right_speed);
30  }
```

Listing 4.7 details the method by which the robot adjusts its heading to steer toward a specified angle.

Given an angle to steer toward, this function adjust the wheel speeds dynamically dependent on the magnitude of the error to a "zero" heading. For example, adjusting to a heading of 180°will result in a more pronounced speed difference versus adjusting to a heading infront of the robot.

Shown in Figure 4.8 are the sectors of the angle discretizations. It was necessary to apply an error value, *ALIGN_ERROR*, when determining a "zero" heading to prevent unwanted oscillations. The function determines which of these sectors the desired heading is located in and applies one of several velocity equations:-

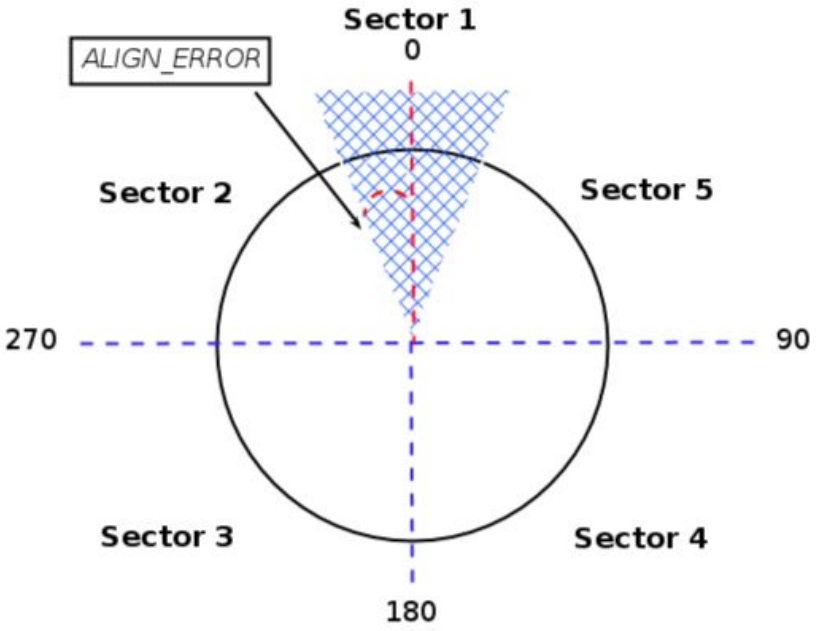


Figure 4.8: Sector discretizations, where 0° is straight ahead

1. Sector 1:

   left wheel speed = BASE_WHEEL_SPEED

   right wheel speed = BASE_WHEEL_SPEED

2. Sector 2:

   left wheel speed =
   $$1 + \frac{ANGLE\_ERROR + angle - 360}{90 - ANGLE\_ERROR} \times BASE\_WHEEL\_SPEED$$
   right wheel speed = BASE_WHEEL_SPEED

3. Sector 3:

   $$\textit{left wheel speed} = \frac{angle - 270}{90} \times BASE\_WHEEL\_SPEED$$
   right wheel speed = BASE_WHEEL_SPEED

4. Sector 4:

   left wheel speed = BASE_WHEEL_SPEED
   $$\textit{right wheel speed} = \frac{90 - angle}{90} \times BASE\_WHEEL\_SPEED$$

5. Sector 5:

   left wheel speed = BASE_WHEEL_SPEED

$$right\ wheel\ speed = 1 - \frac{angle - ANGLE\_ERROR}{90 - ANGLE\_ERROR} \times BASE\_WHEEL\_SPEED$$

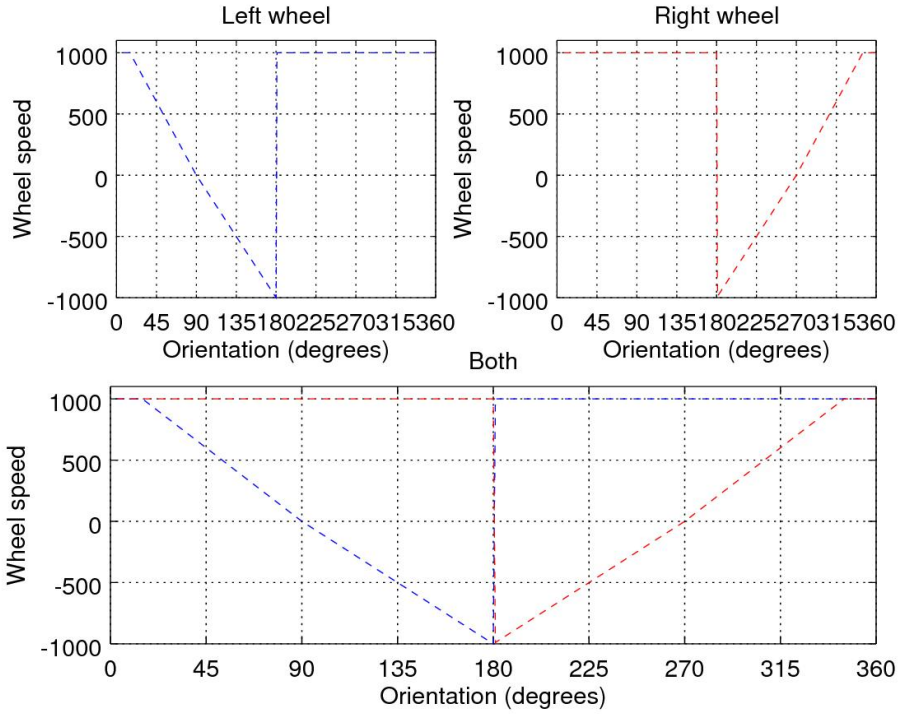


Figure 4.9: Wheel speed control

In this manner, the velocity varies with the distance to zero.

Listing 4.8: Object avoidance

```
1   void Swarm::objectDetection(double speedAdjust) {
2     distanceCheck();
3
4     double left_speed = speedAdjust * WHEEL_SPEED;
5     double right_speed = speedAdjust * WHEEL_SPEED;
6
7     if (!left_obstacle & right_obstacle) {
8       left_speed -= speedAdjust * WHEEL_SPEED;
9       right_speed += speedAdjust * WHEEL_SPEED;
10    } else if (!right_obstacle & left_obstacle) {
11      left_speed += speedAdjust * WHEEL_SPEED;
12      right_speed -= speedAdjust * WHEEL_SPEED;
```

```
13      } else if (right_obstacle & left_obstacle) {
14        left_speed = -speedAdjust * WHEEL_SPEED;
15        right_speed = -speedAdjust * WHEEL_SPEED;
16      }
17
18      if (left_speed > 1000) {
19        left_speed = 1000;
20      }
21
22      if (right_speed > 1000) {
23        right_speed = 1000;
24      }
25      move((int)left_speed, (int)right_speed);
26    }
```

Listing 4.8 details the method by which the robot can perform basic obstacle avoidance. This is achieved by first polling the distance sensors (line 2), then simply checking where there exists an obstacle. If one is detected, the wheel speeds are adjusted appropriately.
Lines 18 to 24 perform velocity clipping, this is needed as the maximum allowable velocity is *1000*, as the controller can crash if this is exceeded.

Listing 4.9: Emitter range manipulation

```
1   if (robots >= 1) {
2       emitter->setRange(RANGE * (robotsDouble + 1));
3   }
```

Listing 4.9 shows how the robot's emitter range is modified. The range varies linearly with the number of robots detected + 1.
This adds a "gravity" to a group. The larger a group is, the more influence it exerts on the environment.

Listing 4.10: Broadcast current orientation

```
1  void Swarm::sendCurrentOrientation() {
2    std::string message = std::to_string(
       ↪ getCurrentOrientation());
3    sendPacket(message);
4  }
```

This method enables the robot to broadcast its current orientation to its neighbours.

Chapter 5

Evaluation

Within this section, the evaluation process of this project will be outline, as well as the resulting experimental results.
Evaluating this swarm system successfully poses several questions:-

1. Can the system be described as exhibiting flocking behaviour in line with the definition of a swarm system?
2. How quickly does the system form a state of flocking?
3. Does the swarm have the ability to form splinter groups?
4. Can these splinter groups amalgamate to form one flock?
5. How well does the system deal with failure?
6. Is the swarm system scalable?

Answering these questions will provide a thorough evaluation of the system's success. To that end, an experiment will be conducted for each question, with the aim of gathering data to make conclusions.

5.1 Testing

Experiment 1: Can the system be described as exhibiting flocking behaviour in line with the definition of a swarm system?

With regard to robotics, Van Dyke Parunak and Brueckner defined swarming as "useful self-organization of multiple entities through local interactions". [26] Following this definition, an assessment of the system can be made.
The ability of the system to self-organise and move as a group can be tested by executing the program and observing the simulation. The cohesiveness of the swarm can be validated by capturing both the GPS data and the number of detected robots in the swarm and comparing these.
The swarm may be evaluated further by comparing the behaviour of the system to the behaviour exhibited by natural systems and observing the similarities and differences.

Experiment 2: How quickly does the system form a state of flocking?

This can be measured by recording the current heading of the robot and the number of neighbours each robot detects, at every simulation time step. Once this data has been gathered, an analysis can be made.

Experiment 3: Does the swarm have the ability to form splinter groups?

This question can be answered by executing a simulation where some members of the swarm are segregated from the others and observing the behaviour over time. Also, positional and neighbourhood data can be gathered to further validate the conclusion.

Experiment 3: Can these splinter groups amalgamate to form one flock?

This question and the last can be tested during the same experiment. Once separate groups have been formed, the segregation can be removed and the

behaviour of the two groups analysed.

Experiment 4: How well does the system deal with failure?

Failure within a swarm can be defined in a number of ways. The main point of failure, however, are the agents themselves. In this case it can be simulated by powering down robots during execution of the program. This will test to see if the system can adapt to changing circumstances.

Experiment 5: Is the swarm system scalable?

One of the defining features of a robot swarm is the system's scalability. This can be easily tested through the addition of more agents to the swarm and observing the system's behaviour. GPS and neighbourhood data can also be gathered and these can be compared with different swarm sizes for analyses.

5.2 Results and Analysis

Upon conducting the experiments outlined in the previous section, the results as well as a discussion of the findings are presented here.

5.2.1 Experiment 1

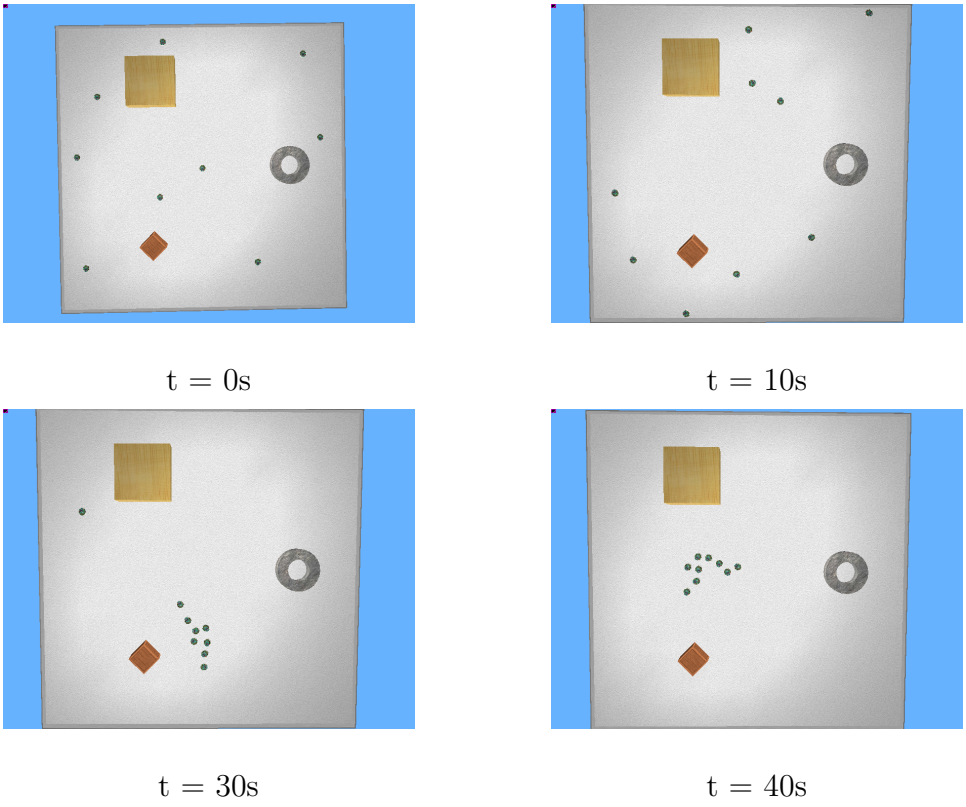


Figure 5.1: Swarm movement over time

Figure 5.1 shows the movement of the swarm over a 40 second period. Initially, the robots are separated. As can be seen, at $t = 10s$, each unit has moved relative to its starting position however do not yet exhibit group behaviour. Soon after this point though, the agents begin to coalesce into a cohesive group. By $t = 30s$, the majority of agents are within detection range and are moving together. By $t = 40s$, all of the agents have formed one group, which expand, contract and move around the environment together.

This can be shown graphically (Fig. 5.2, Fig. 5.3), through the collection of data directly from the simulated robots. As can be seen, as time increases, the

number of robots detected tends to the maximum in the arena.

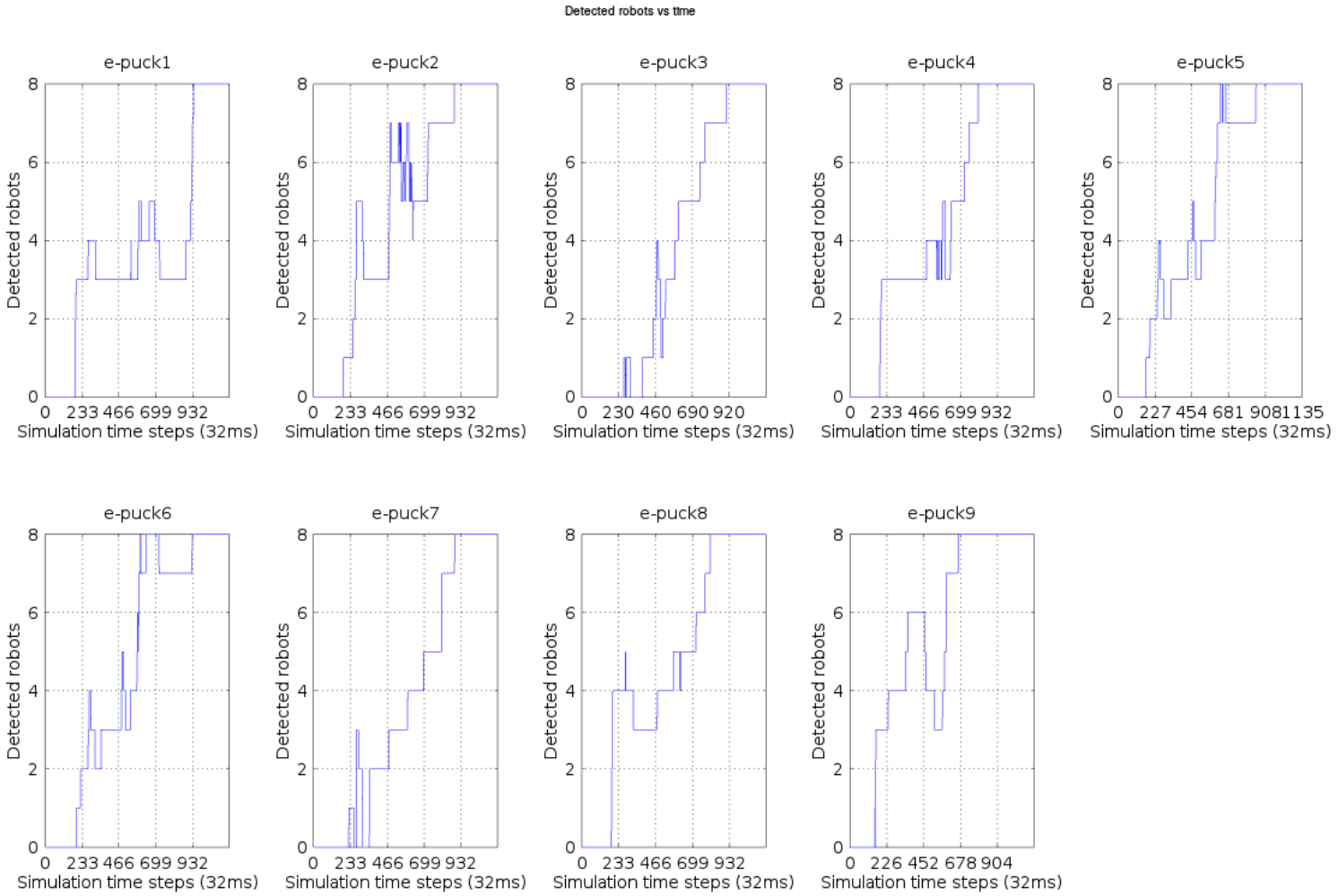


Figure 5.2: Number of robots detected vs time

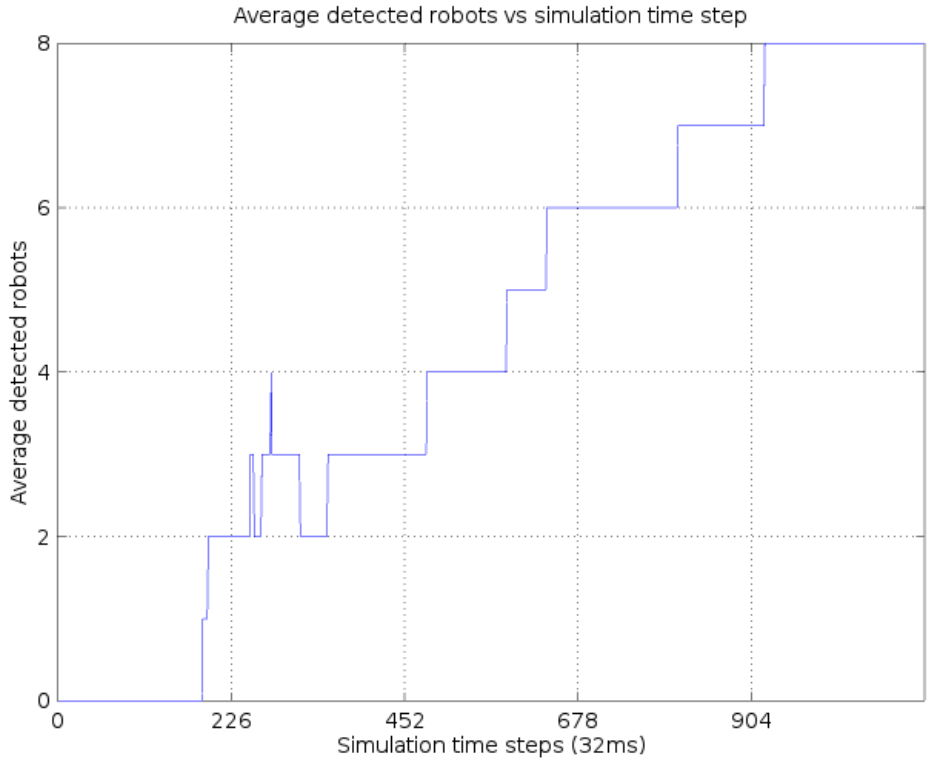


Figure 5.3: Average number of detected robots

This proves that given enough time, the swarm system will form into one cohesive flock. Due to the nature of the design of the system, there is no higher level supervisor (human or otherwise) co-ordinating the movements of the flock. Therefore, as the system moves to eventually form a group without external aid, the swarm can be said to exhibit self-organization. Furthermore, since each robot broadcasts information locally and does not require any form of positional data detailing the whole swarm, the system can be said to be only interacting locally.

Given the above analysis, this system satisfies Van Dyke Parunak and Brueckner's definition of a swarm, *"useful self-organization of multiple entities through local interactions"* [26].

Next, the behavioural similarities to natural systems can be examined.

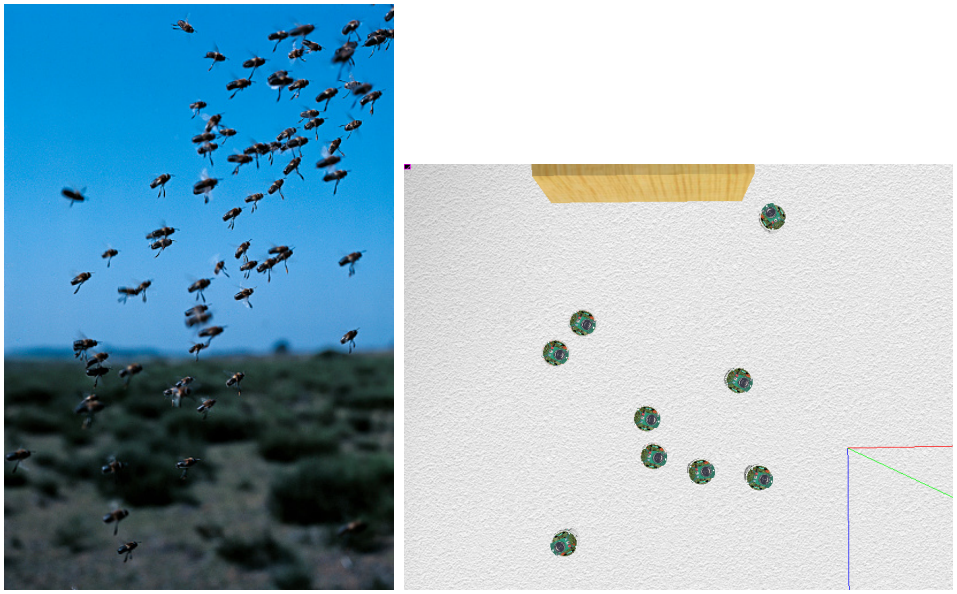


Figure 5.4: Swarm of bees vs a swarm of e-pucks

Owing to the e-puck swarm's rapid movements within the swarm boundary, the most similar biological equivalent would be a swarm of bees. As can be seen from Figure 5.4, both systems exhibit strikingly similar behaviour. Both are homogenous groups, forming a loose collective of units. The majority of agents of both groups are oriented to the same heading, with some outliers steering toward the group.

Through analysis of the above experiment, the question can be answered. The swarm system can be said to be exhibiting flocking and is an example of biologically inspired behaviour.

5.2.2 Experiment 2

Run	Average time steps (32ms)
1	920
2	380
3	1459
4	1873
5	914
Average	1109 (35.488s)

Figure 5.5: Average time steps to aggregation

In order to answer this question, the average of multiple simulation runs were made. It can be seen from Fig. 5.5 that the average time to aggregation is 35.488 seconds. This is in line with Fig. 5.1. The wide range in average times can be explained by the randomness of movement that is inherrent in emergent systems like this one.

Though the time to aggregation is fast, this can be improved further if a higher level controller is directing the behaviour of every e-puck. However, this would not be a true, decentralised swarm system and would thus be more susceptible to errors and failure.

5.2.3 Experiment 3

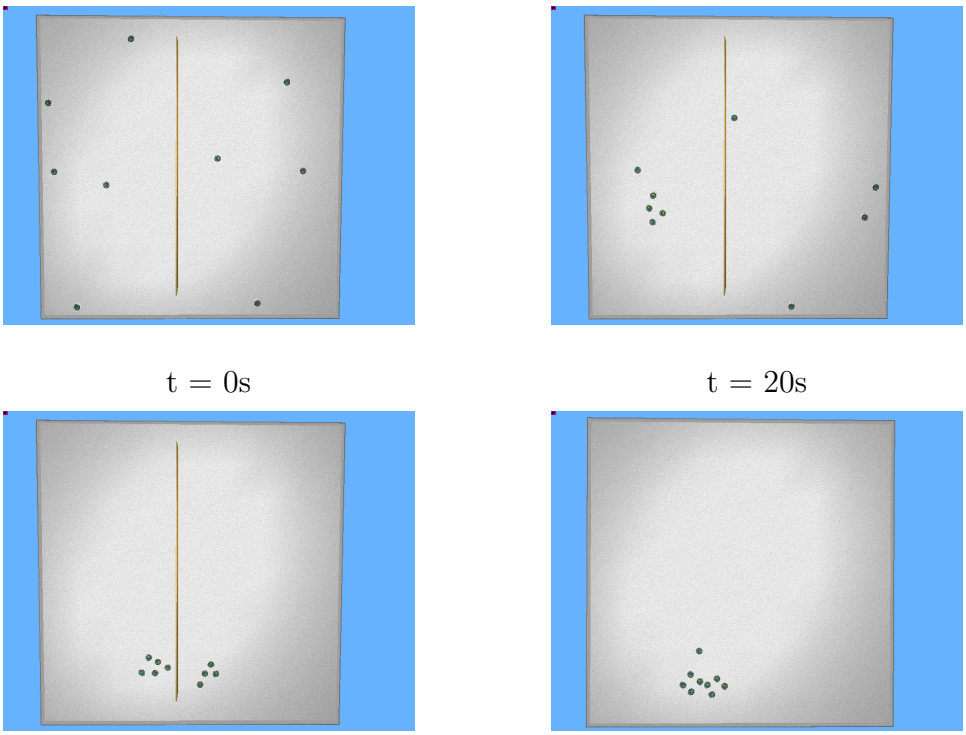


Figure 5.6: Splinter group test

Figure 5.6 shows the segregated system over a 70 second time period. As predicted, each group on either side of the barrier are quickly able to form independent formations, though the time taken to do this is increased due to a decrease in swarm density. Once the barrier is removed, the two groups quickly attract one another and amalgamate to form one flock.

This is due to the inherrent locality of the swarm's interactions. Each member of the swarm does not have information on the state of the entire system, thus it behaves according to what is presented to it nearby.

In addition to this, this experiment showed the ability of the swarm to operate in a different and changing environment. Again, this is due to the locality of the agent-agent interactions. This is a demonstration of how crucial the concept of locality is to a swarm's ability to cope with fault tolerance and a dynamic environment.

5.2.4 Experiment 4

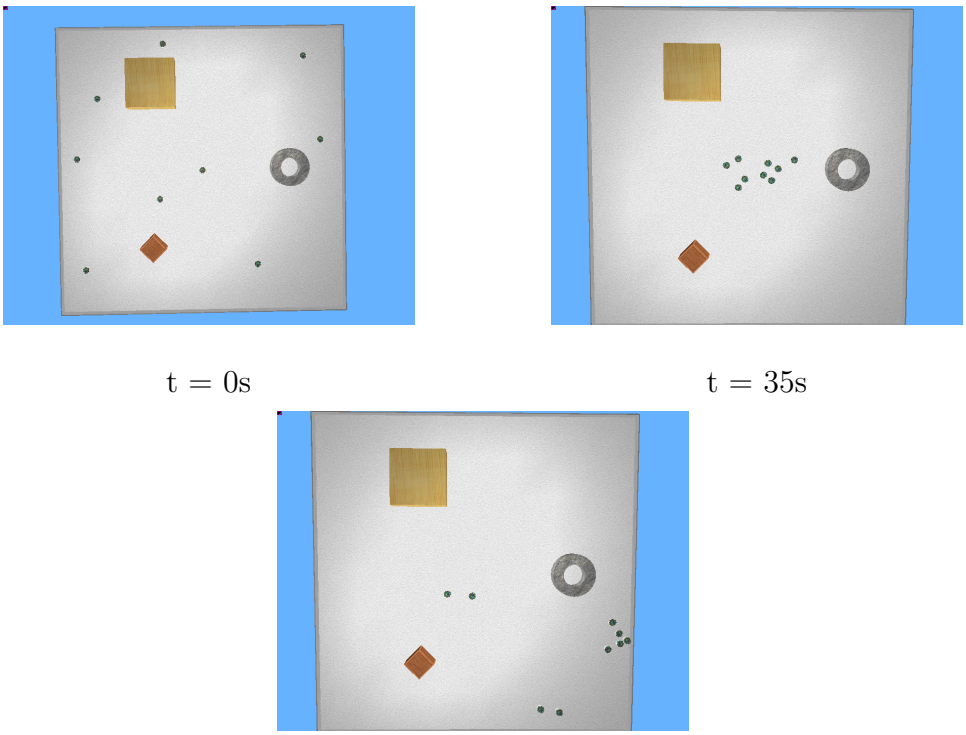


Figure 5.7: Failure test

The simulation was allowed to progress until a flock had amalgamated, in this case at $t = 35s$. After this instance, two agents were powered down (e-puck1 and e-puck4, in the center of the arena) to simulate a fatal error/fault. It can be seen in Fig. 5.7 that over time, the system ignores these two inoperable robots and continues exhibiting flocking behaviour.

This is because the functional agents do not receive any signals from the inoperable agents and will treat them as if they were another obstacle in the environment, something to be avoided.

This can be validated through Fig. 5.8. As can be seen, all robots bar e-puck1 and e-puck4 settle at detecting 6 robots, which is the total number of functional robots in the swarm.

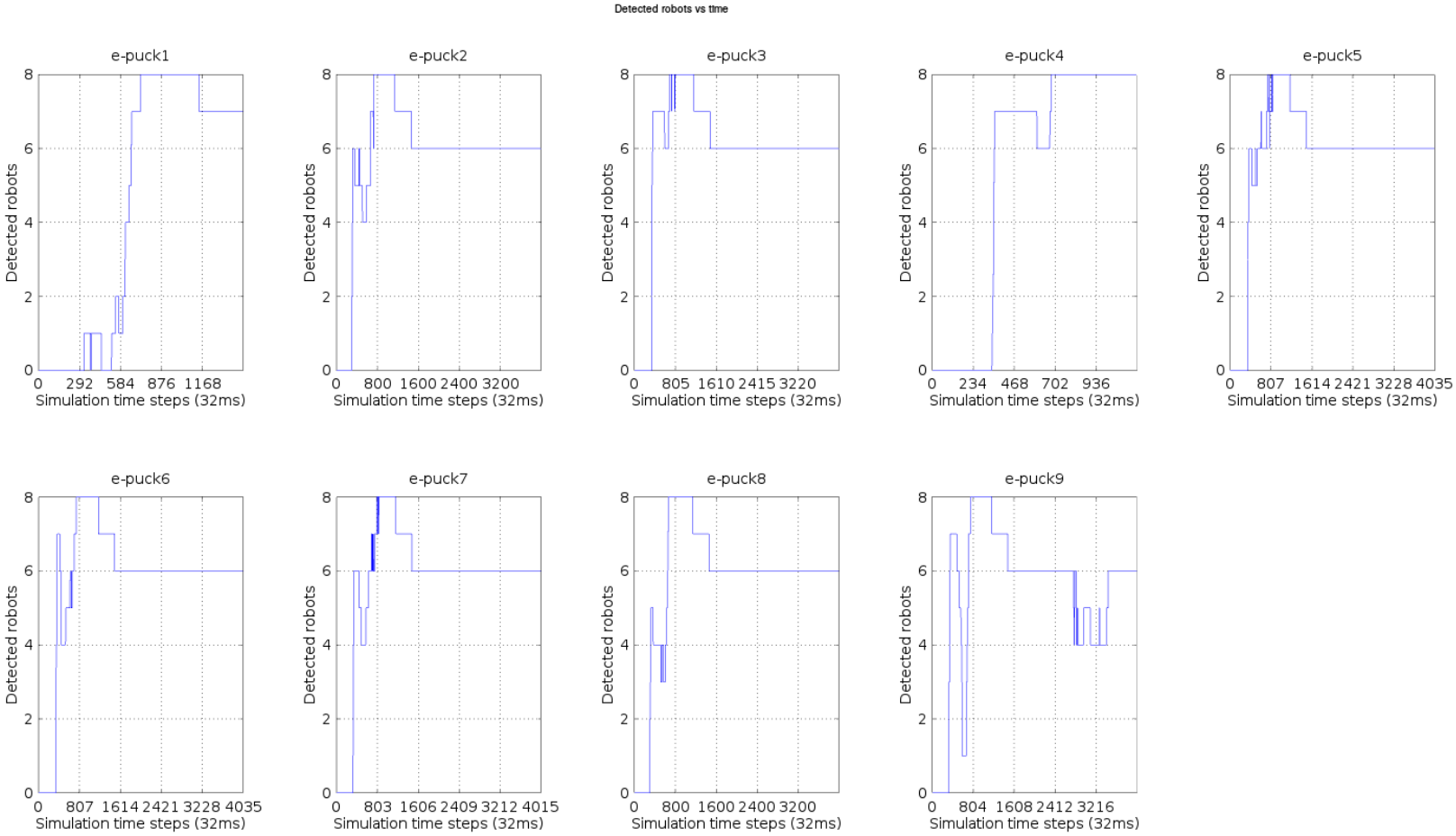
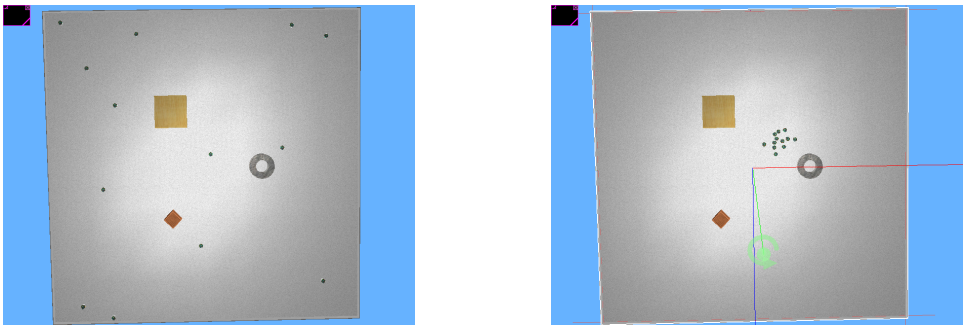


Figure 5.8: Number of robots detected vs time

5.2.5 Experiment 5



t = 0s t = 90s

Figure 5.9: Scalability test

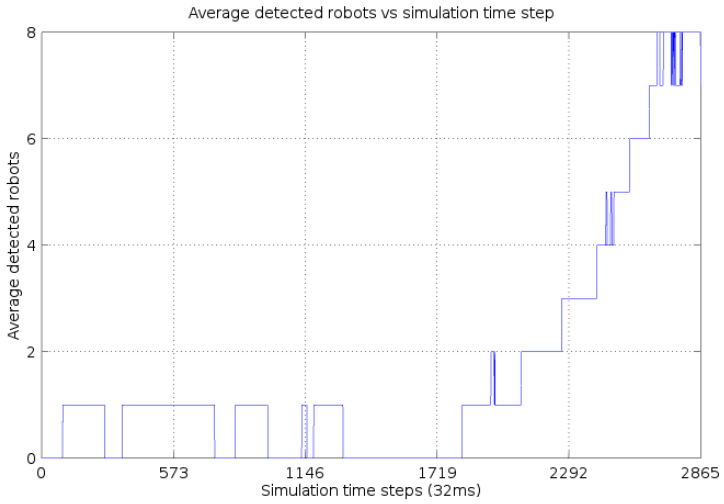


Figure 5.10: Average number of detected robots

Finally, one of the key features of a robot swarm system is its ability to scale in size and still exhibit similar behaviour.

Along side the swarm's ability to handle errors and faults, Experiment 4 showed how the swarm is capable of scaling *down* in size and still function properly (agent count decreased from 9 to 7). This experiment aimed to shed light on the swarm's ability to scale *up*.

In this case, another 3 agents were introduced to the environment (an increase of 33%) and the simulation was executed. Shown in Fig. 5.9 is a 90 second

period of time from the beginning of the simulation. By the end, the swarm had coalesced into a similar formation as shown in previous experiments.
Once again, this is due to the local nature of the agent-agent interactions. The individual agents do not care about the state of the agents it cannot see nor does it glean any information from a global source.
Adding extra agents to the system does increase the workload that is placed on the simulator, with the simulation running slower than previously. However, this does not effect the behaviour of the robot controller so is ignored.

Chapter 6

Conclusion

In conclusion, this project set out to study the swarming behaviour exhibited in some biological systems such as social insects, flocks of birds, schools of fish and implement them in an artifical system: a group of mobile robots. It has shown that the application of biologically inspired techniques to the field of swarm robotics can provide interesting and useful results.
In particular, the rise of emergent self-organisational behaviour within this system satisfies Van Dyke Parunak and Brueckner's definition of a swarm, *"useful self-organization of multiple entities through local interactions"* [26].
The original objectives of this project were:-

1. To understand the theory and concepts behind a swarm system, be they natural or artificial, in order to apply them within an engineering design.

2. To develop a robot controller which exhibits *flocking* behaviour, based on the techniques of self-organising swarm systems.

3. To evaluate the developed system under dynamic environmental conditions and stressors.

As shown in previous sections, all three objectives were achieved.
Objective 1 was fulfilled through in depth analysis of current state-of-the-art research within the fields of biologically inspired engineering and collective

robotics. This provided a basis for the fundamental knowledge needed to fulfill objective 2, to engineer a swarm system from first principles.

Objective 2 was fulfilled through the design and implementation of the swarming algorithm detailed in Section 4.2. One of the significant achievements of this work was the development of a single robot controller to exhibit system wide swarming. The principal biologically inspired techniques used within this controller were the concepts of locality and limited agent-agent communication. As discussed and experimentally proven in Section 5, this provides a system that is scalable, tolerant of faults and is capable of self-organisation. In addition to this, having one adaptable controller that can be distributed to as many robots as desired is far more time efficient than developing a bespoke controller for every robot in the swarm.

Objective 3 was achieved through experimental analysis, detailed in Section 5.2. The system was tested to measure its performance in a variety of environments and conditions. These environments and conditions are typical of those faced by swarms, such as a changing world, agent failure, obstacle avoidance and flock merging. The ability of the system to adequately cope with these stressors is a crucial proof of concept of the applicability of swarming techniques within artificial systems. It would be highly beneficial if these useful traits could be obtained for use in other applications, for example use within a collective of autonomous vehicles.

6.1 Problems encountered

During the course of the project, many technical problems were encountered. The main problem encountered was dealing with obstacle recognition. Animals or insects that exhibit flocking behaviour are able to distinguish between, for example, a wall and a member of its own species. The simulated e-puck has no way of doing this natively, the proximity sensors simply tell the e-puck how close an object is, and not *what* the object is. In contrast, a real e-puck is able to utilise its proximity sensors as a way of signalling its presence, by

broadcasting infra-red light. In order to overcome this problem, the use of an infra-red emitter was decided upon. The e-puck uses this emitter to broadcast its orientation, and in this way, provides a way to signal its presence to other e-pucks.

Leading on from this, the next problem was to provide a method that allows the e-puck to interpret an incoming message packet into actionable information. This problem was overcome through the messaging processing methods detailed in Listing 4.1 and 4.2.

Another issue that arose was determining a suitable method to gather data, especially neighbourhood data. One solution was to graphically measure the emitter ranges of every robot and hence determine how many other robots it detects. Doing this many times over in every simulation would be extremely time consuming and could lead to false results. Instead, it was decided to implement data gathering features within the robot controller itself. The robots automatically write any needed values (such as neighbourhood, orientation and GPS data) to files, for later analysis. Then, a MATLAB script was created that parses these files, extracts the data and plots them. In this manner, plenty of accurate data is gathered.

6.2 Limitations

The main limitation of this work is that the swarm stuggles to move in a similar heading for more than a few seconds, with its movements tending more toward attraction and aggregation. This is due to the fact that no overall goal is presented to the swarm, such as moving to a defined co-ordinate as a group.

Another limitation was the performance of the simulator itself. Adding more e-pucks to the simulation consumed more computational resources, which decreased the execution time and limited the ability to validate and test the scalability of the swarm.

It was found that the algorithm developed in this project, though inspired by Reynolds' rules [16], was not as effective at providing flocking-like behaviour,

though the behaviour exhibited is still similar to some biological systems.

6.3 Future work

This project serves as a case study into the effectiveness of applying biologically inspired techniques to engineering designs and is demonstrated successfully during the course of this project. Mimicking natural systems could provide a number of solutions to problems within industry.
This work can be extended on several fronts:-

1. Using evolutionary robotics techniques such as evolving a neural network (which would act as the robot controller) with a genetic algorithm could provide a more robust controller. This would be at the cost of time and complexity.

2. Implementing a method of user interaction with the swarm (such as following the user's mouse cursor) would increase the scope of experimental testing and could lead to more cohesive behaviour.

3. Implementing an ability for the simulation to read parameters contained within files would enable experiments to be run much easier.

4. Using a radio transmitter instead of infra-red would allow signals to travel through obstacles. This would allow the development of different behaviour.

Bibliography

[1] Guy Beauchamp. In *Social Predation: How group living benefits predators and prey*. Academic Press, San Diego, 2014. ISBN 978-0-12-407228-2. doi: http://dx.doi.org/10.1016/B978-0-12-407228-2.01001-5. URL `http://www.sciencedirect.com/science/article/pii/B9780124072282010015`.

[2] F. Cucker and S. Smale. Emergent behavior in flocks. *IEEE Transactions on Automatic Control*, 52(5):852–862, May 2007. ISSN 0018-9286. doi: 10.1109/TAC.2007.895842.

[3] H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2:1734–1741, 2000. ISSN 1050-4729. doi: 10.1109/ROBOT.2000.844846.

[4] M. Yim. *Locomotion with a Unit-Modular Reconfigurable Robot*. PhD thesis, Stanford University, 1995.

[5] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, Aug 1997. ISSN 1042-296X. doi: 10.1109/70.611311.

[6] A. T. Asbeck, S. M. M. De Rossi, I. Galiana, Y. Ding, and C. J. Walsh. Stronger, smarter, softer: Next-generation wearable robots. *IEEE Robotics Automation Magazine*, 21(4):22–33, Dec 2014. ISSN 1070-9932. doi: 10.1109/MRA.2014.2360283.

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

[8] S.S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009. ISBN 9780131471399. URL https://books.google.co.uk/books?id=K7P36lKzI_QC.

[9] Autopilot — tesla. https://www.tesla.com/en_GB/autopilot/?utm_campaign=GL_AP_101916&utm=&redirect=no.

[10] Dictionary. http://www.dictionary.com/browse/artificial--life.

[11] The MIT Press. *The MIT Encyclopedia of the Cognitive Sciences*. 2003. ISBN 978-0-262-73144-7.

[12] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Bradford Books. MIT Press, 1986. ISBN 9780262521123. URL https://books.google.co.uk/books?id=7KkUAT_q_sQC.

[13] Wang J. Beni, G. Swarm intelligence in cellular robotic systems. *Proceed. NATO Advanced Workshop on Robots and Biological Systems*, June 1989.

[14] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm intelligence : From natural to artificial systems. *Santa Fe Institute studies in the sciences of complexity*, 1999.

[15] Stanford university news service. http://news.stanford.edu/pr/93/931115Arc3062.html.

[16] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987. ISSN 0097-8930. doi: 10.1145/37402.37406. URL http://doi.acm.org/10.1145/37402.37406.

[17] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a

robot designed for education in engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, vol. 1, num. 1, p. 59-65*, 2009.

[18] Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo. From fireflies to fault-tolerant swarms of robots. *Trans. Evol. Comp*, 13(4): 754–766, August 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2009.2017516. URL http://dx.doi.org/10.1109/TEVC.2009.2017516.

[19] African Robotics Network. https://www.wired.com/2012/09/afron-winners/.

[20] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298, May 2012. doi: 10.1109/ICRA.2012.6224638.

[21] Khepera IV. https://www.k-team.com/mobile-robotics-products/khepera-iv/introduction.

[22] Bot'n Roll ONE A. http://www.botnroll.com/en/bot-n-roll-one-a/1135-bot-n-roll-one-a-assembled.html.

[23] Cyberbotics - About. https://www.cyberbotics.com/company.

[24] C++ history. http://www.cplusplus.com/info/history/.

[25] TIOBE index. https://www.tiobe.com/tiobe-index/.

[26] H. Van Dyke Parunak and Sven A. Brueckner. *Engineering Swarming Systems*, pages 341–376. Springer US, Boston, MA, 2004. ISBN 978-1-4020-8058-6. doi: 10.1007/1-4020-8058-1_21. URL http://dx.doi.org/10.1007/1-4020-8058-1_21.

Appendix A

e-puck Specifications

A.0.1 Electronic Systems

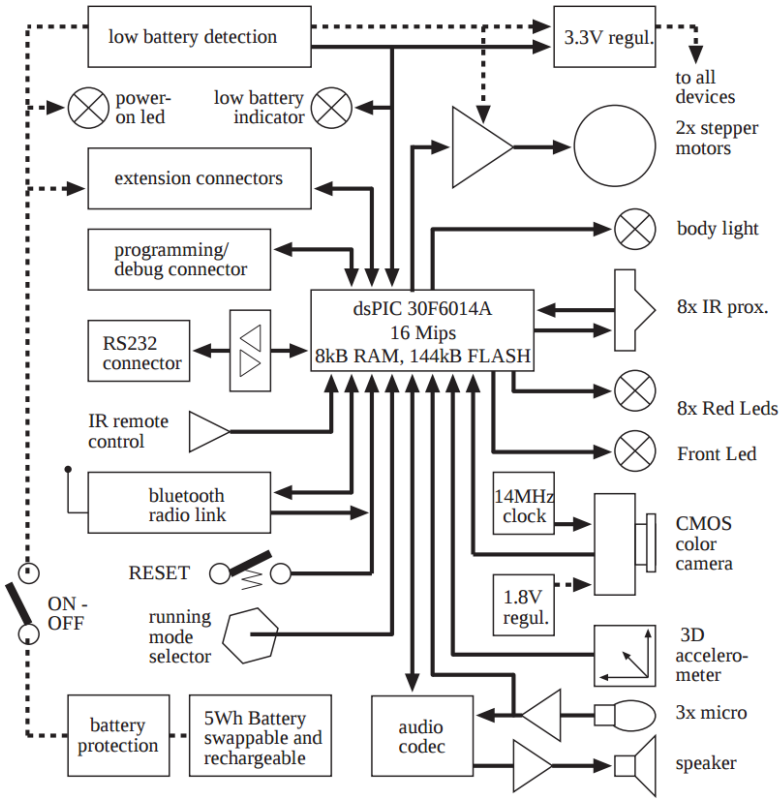


Figure A.1: e-puck electronics systems [17]

A.0.2 Technical Specifications

Diameter	7cm
Weight	200g
Top speed	13cm/s
Battery	2 hours
Processor	dsPIC 30 CPU @ 30 MHz (15 MIPS)
RAM	8KB
Memory	144KB Flash
Motors	Step
IR sensors	8 infra-red proximity and light sensors
Camera	Front facing, 640x480 resolution
LEDs	8 circumferential, 1 body, 1 front
Accelerometer	1, three dimensional
Microphones	3
Speaker	1

Table A.1: e-puck specifications

Appendix B

Code

Complete source code can be found at:
`https://github.com/helmifraser/Swarm`.