Behaviour-Based Pattern Formation in a Swarm of Anonymous Robots

by

© Nicholi Shiell

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

September 12 2017

St. John's Newfoundland

tion.tex

Abstract

The ability to form patterns is useful to maximize the sensor coverage of a team of robots. Current pattern formation algorithms for multi-robot systems require the robots to be able to uniquely identify each other. This increases the sensory and computational requirements of the individual robots, and reduces the scalability, robustness, and flexibility of the pattern formation algorithm. The research presented in this thesis focuses on the development of a novel pattern formation algorithm called the Dynamic Neighbour Selection (DNS) algorithm. The DNS algorithm does not require robots to be uniquely identified to each other, thus improving the scalability, robustness, and flexibility of the technique. The algorithm was developed in simulation, and demonstrated on a team of vision-enabled Bupimo robots. The Bupimo robots were developed as part of the research reported in this thesis. They are a low-cost, vision enabled, mobile robotic platform intended for use in swarm robotics research and education. Experiments conducted using the DNS algorithm were performed using a computer simulation and in real world trials. The experiments conducted via simulation compared the performance of the DNS algorithm to an other similar algorithm when forming a number of patterns. The results of these experiments demonstrate that the DNS algorithm was able to assume the desired formation while the robots traversed a shorter distance when compared to the alternative algorithm. The real robot trials had three outcomes. First, they demonstrated the functionality of the Bupimo robots, secondly they were used to develop an effective robot-robot collision avoidance technique, and lastly they demonstrated the performance of the DNS algorithm on real robots.

Acknowledgements

First and foremost I would like to thank my supervisor Dr. Andrew Vardy for being my guide into the world of autonomous robotics, as well as for his time and patience through the writing of this thesis. I would also like to thank the members of the Bio-inspired Robotics Lab at Memorial University, especially Dr. Todd Wareham for our many interesting discussions. Lastly, I would like to thank Cheryl Hurley for introducing me to the many wonderful sights and sounds of Newfoundland.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter will introduce the terms used in the title of the thesis. First the principles of swarm robotics and its benefits will be described. This will be followed by a brief explanation of robot control architectures, with a focus on behaviour-based control. Next, pattern formation in mobile robotics will be reviewed. Finally, the contributions of this thesis, and an outline of the remainder of the thesis will be given.

1.1 Swarm Robotics

Swarm robotics is the application of swarm intelligence to the coordination and control of multiple, often simple[1], autonomous robots. This approach to autonomous robotics has been inspired by groups of social animals, such as ants. These types of animals are able to self-organize to accomplish complex tasks. To do this, they use cues from their local physical environment, and from nearby swarm mates [1]. This type of local sensing and communication is referred to as stigmergy [2]. The decentralized nature

[1] In this case simple means having limited sensory, control, and communication abilities.

of these social groups has many benefits when utilized for the control and coordination of groups of autonomous mobile robots. By using a decentralized control framework the developed system is easily scalable, flexible, and robust [3]. It is scalable in that a variable number of robots can be controlled in performing the desired task. The system will be flexible, in that the task can be performed correctly even in a dynamic environment. Finally, the system is robust because of unit redundancy, since the system is composed of many independent parts it can still function when some are disabled.

Examples of the types of problems studied by swarm robotics are aggregation, pattern formation, object clustering, collective exploration, or coordinated motion [4].

The characteristics of a swarm robotic system include [2] [4],

1. Autonomous agents
2. Local sensing and communication
3. Decentralized control
4. co-operation between agents

A more contentious component of swarm robotics is the simplicity of the robots used [2]. Early swarm robotic research was inspired by social insects such as ants, termites, and bees. At the time, the swarm robotic research community considered these insects as simple, or limited in sensory, communication, and locomotion abilities. As a consequence, the robots used in early swarm robotics research were limited in the same respects. Modern swarm robotics research does not always follow this

minimalist criteria and instead explores the use of highly capable robots in swarm robotic systems.

The development of the Dynamic Neighbour Selection (DNS) pattern formation algorithm follows the minimalism criteria for three reasons. Firstly, swarm robots are applicable to environments where some robots will be lost or disabled. To ensure the practicality of such a solution, each robot in the swarm must be financially expendable. Minimal robots are cheaper than more capable robots, and therefore more expendable. Secondly, although a swarm robotic system is robust to the loss of some individuals, its effectiveness is still reduced. Minimalist robots are likely to be more robust than complex ones for the simple reason that there are fewer parts (sensors, motors, and other actuators) which can malfunction. Therefore to keep as many robots operational for as long as possible a criteria of individual minimalism will be followed in the proposed work. Lastly, by restricting the sensory, computational, and communication abilities, the physical size of the robot can be greatly reduced. This means algorithms developed for minimalist swarms can more easily be transferred to robots at physical scales far below that of traditional robotics [5].

Though the DNS algorithm was developed with minimalism in mind, it will be tested on non minimalist robots, namely the Bupimo. The same process of developing a minimalist control algorithm, then testing it on a complex robot was used in [5]. The Bupimo robots developed as a part of this thesis are far from minimal: they possess extensive computational abilities, and a powerful omnidirectional camera. However, the DNS algorithm requires only a fraction of these capabilities. Namely, the ability to measure the bearing to near by swarm mates relative to a global reference direction (ie. North). By extracting only a limited amount of information from it's sensors,

the Bupimo is used as an analogue of a minimalist robot. This minimalism extends only to the robots sensing abilities. The Bupimo is still able to travel freely in any direction along the ground.

1.2 Behaviour-Based Robotic Control Architectures

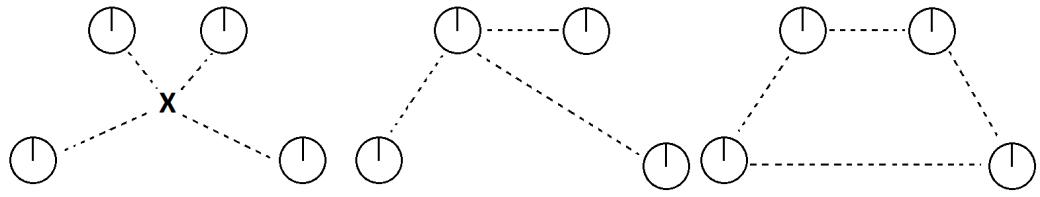
A control architecture defines how a robot interacts with the environment while in pursuit of its objectives. The environment acts on the robot by providing its sensors with input, and the robot interacts with the environment through its motors and actuators. How this transition from sensing to acting is carried out is dependant on the control architecture used. These various architectures exist on a spectrum with deliberative/hierarchical approaches at one end and behavioural/reactive approaches at the other. The main difference between these extremes is the extent to which the robot has an internal state. A deliberative robot will have a detailed internal state. This could be knowing its position in a map of its environment, and then using this knowledge to plan its path to some other part of the environment. A reactive robot has no, or at most a very primitive internal state. Deliberative architectures are well suited for structured and highly predictable environments, whereas behavioural approaches are better suited to dynamic and unknown environments [6]. Since swarm robotics is intended for use in dynamic real world environments, only behaviour-based control architectures will be discussed here. Specifically, behaviour-based control architectures can be thought of as a blueprint for how sensor data is used to coordinate behaviours. In the following chapters, an implementation of a control architecture will be referred to as a controller or more specifically a behaviour-based controller.

A robotic behaviour is a stimulus-response pair. Given a stimulus (sensor input) the behaviour calculates or determines an appropriate response (referred to here as a velocity command). What is deemed as an appropriate response depends on the task the behaviour is meant to accomplish. The encoding of a response can be either discrete or continuous. A discrete response has a finite number of values, and is generally determined by a logical statement (for example, IF at goal THEN stop). Continuous responses can assume an infinite number of values, and are generally calculated by a mathematical function dependent on sensor readings and the current state of the robot. Since the proposed work is intended to control a robot moving in a plane, the behaviour responses will take the form of two-dimensional velocity vectors.

A complete behaviour-based control architecture will invariably contain multiple behaviours. Each behaviour will generate a response, in the form of a velocity vector, to sensor stimuli. Eventually a situation will arise where response vectors oppose one another. In this case the behaviours are said to be conflicting. How this situation is handled depends on the behaviour-based control architectures used. These architectures primarily fall into one of two categories; competitive, or cooperative. A basic competitive approach selects a single behaviour's response. Examples of competitive approaches [6] are subsumption, state-based arbitration, and winner-takes-all. Cooperative methods use a superposition approach to build a response out of the multiple responses from the behaviours. Examples of cooperative approaches [6] include motor-schema, voting, and fuzzy-logic.

1.3 Pattern Formation in Teams of Autonomous Robots

The ability to self-organize into a pattern is useful for a swarm of robots. For example, in the case of cooperative sensing pattern formation can be used to maximize sensor coverage, or when travelling through an environment maintaining a formation prevents collisions between robots. For these reasons decentralized pattern formation is an active field of research [7–14]. Typically the task of pattern formation is broken into two parts. First the robots are assigned a location in the pattern by giving them a set of geometric constraints to maintain relative to a set of reference points. There are three basic reference types used; centre referenced, leader referenced, and neighbour referenced [15] (see Figure 1.1). The most common for decentralized pattern formation techniques is neighbour referenced. In this case, each robot is given a set of bearing or displacement (bearing and distance) constraints to unique neighbours. When all the robots have minimized the error in their constraints, the pattern is formed. Although commonly used, the requirement of unique neighbours significantly reduces the fault tolerance of the system by introducing a single point of failure (ie. the loss of a single robot). The work reported in [9] explores methods for dealing with the loss of a unique robot in such a scheme. The second part of a typical pattern formation algorithm is a set of control laws that cause the robots to move to their desired location within the pattern. This set could contain a single control law or multiple. The control laws typically make use of bearing, heading (direction of motion), and distance to neighbours as inputs. However, acquiring this information requires significant sensory capabilities.



(a) Centre Referenced (b) Leader Referenced (c) Neighbour Referenced

Figure 1.1: Examples of the three basic reference types. The circles represent robots, and the dashed lines represent geometric constraints. (a) Center referenced. Requires an agreed upon origin. (b) Leader referenced. Requires the selection of a leader which is visible to all other swarm mate. (c) Neighbour referenced is the most commonly used reference type in decentralized pattern formation.

The need for unique identities, and detailed state information about neighbours decreases the robustness, flexibility, and scalability of a pattern formation algorithm and increases the complexity of robots. The ability for a robot to uniquely identify another requires each robot to have a unique identifying marker and a sensor capable of reading it. The Dynamic Neighbour Selection (DNS) pattern formation algorithm developed as part of this thesis will differ from similar algorithms in two important ways; the definition of the desired formation, and the use of a control law which does not require high precision sensing, such as unique IDs, heading, or distance.

1.4 Contributions of the Thesis

There are two primary contributions made by the work reported in this thesis. The first contribution was the development of a decentralized behaviour-based pattern formation algorithm which uses a neighbour-referenced approach, and limited sensory

information about anonymous neighbours as input. This algorithm will be referred to as the Dynamic Neighbour Selection (DNS) algorithm. The second contribution was the development and testing of a swarm of low-cost vision enabled robots, called the Bupimos, from various off the shelf components (see Chapter 4 for details). Other examples of low-cost swarm and vision enabled robots are include [16] and [17] respectively.This work included the development of a PID controller which utilized feedback from motor encoders to control the velocity of the robot, interfacing between the main computer and a micro-controller to allow for communication of sensor data and motor commands, calibration of the omnidirectional camera to estimate distances, the integration of a digital compass to allow the robot to measure its heading in a global reference frame, and assisting in the development of control software. The Bupimo robots are intended for use in swarm robotics research and robotics education. The DNS algorithm was compared to a similar bearing-only algorithm [11] in simulation, and was then demonstrated in real-world trials using a swarm of Bupimo robots. The simulated experiments showed the DNS algorithm was able to achieve the desired formation with a lower average distance travelled per robot than the benchmark algorithm. The real world trials of the DNS algorithm had a number of interesting outcomes. First, they demonstrated the functionality of the Bupimo robots. Secondly an effective robot-robot collision avoidance technique was developed during the trails. Lastly the live trials demonstrated the performance of the DNS algorithm on real robots.

The work presented in this thesis led to the publication of two peer-reviewed articles:

1. A Bearing-Only Pattern Formation Algorithm for Swarm Robotics. *International Conference on Swarm Intelligence*, 2016. [18]

2. BuPiGo: An Open and Extensible Platform for Visually-Guided Swarm Robots. *proceedings of the 9th EAI International Conference*, 2016. [19]

The material presented in [19] introduces the Bupimo (previously named the Bupigo) robot and corresponds roughly to Chapter 4 of this thesis. The latter paper [18] discusses the development and testing of the DNS algorithm in simulation. This material is covered in Chapter 3 of this thesis.

1.5 Thesis Outline

The layout of the remainder of this thesis will be as follows. Chapter 2 will contain two literature reviews. The first reviews work done on minimalist robotics, and the second reviews pattern formation algorithms in swarm robotics. Chapter 3 will describe the DNS pattern formation algorithm, its formation definition, and the behaviours used and the behaviour-based controller used to coordinate them. The DNS algorithm will also be compared in a computer simulation to a similar algorithm found in the literature. Chapter 4 will describe the development of the Bupimo, the robot's sensors, and the software used to control it. Chapter 5 will contain the results obtained from implementing the DNS algorithm on a swarm of Bupimo robots in a series of real world trials. Finally, Chapter 6 will summarize the major results and conclusions of the thesis, and discuss some possible avenues for future research.

Chapter 2

Literature Review

This chapter contains two literature reviews. First the literature pertaining to minimalist robotics will be reviewed. It will discuss the types of problems which have been studied, what is meant by minimalism, and the motivation for its study. Following this a selection of swarm robotic pattern formation algorithms found in the literature will be reviewed and compared to the DNS algorithm.

2.1 Minimalist Robotics

A wide range of canonical problems in mobile robotics have been investigated from a minimalist robotics point of view. These include aggregation [5, 20–22], flocking [21], localization [23], foraging [22], and coordinate motion [24]. These works differ in their approach to minimalism, some have restricted both sensory and motor control of the robot, while others have focused only on sensor minimalism. Another important difference is the motivations for investigating minimalism. This section will compare the differences between the cited works as well as comparing them to the work presented

in this thesis.

The work reported in [5, 21, 22, 24] have explored minimalism in both the sensors and motion control of the robots. In these studies the movements of the robots are restricted to a finite set of motor states. For instance in [21], the robots were only capable of three motions: move forward, rotate left, and rotate right. This type of limited motion control is referred to as quantized control [21], computation free robotics [5,22], or look-up tables [24]. In all of these works a finite set of sensor states has been mapped on to a finite set of motor states. In these works the relationship between sensor states and motor states can be written as an if-then-else blocks. The works reported in [20, 23] did not restrict the motion control of the robots. In these papers the robots had continuous, as opposed to discrete, motion control. The work presented in this thesis will follow a hybrid approach. A finite set of sensor states will be mapped to a finite set of behaviours. However, each behaviour uses a continuous motion control strategy.

The sensors available to the robots differed notably across the works cited. The works in [20, 21] investigates various omnidirectional sensors. In [20] robots are able to measure the local bearing to their neighbours. In [21] the field of view of the robots sensors are broken into discrete regions. Depending on the region in which neighbours are detected, different motion responses are made. The works in [5, 22] have used a single line of sight sensor directed forward in the robots frame. In [5] the sensor returns a binary value depending on the presence of a robot or free space. In [21] a trinary sensor is used which is capable of detecting free space, a robot, or an object (used for clustering). The ranges of the sensors used also differ between the various works. Sensors with infinite range were used in [5,21,22], whereas [20,21,24] investigated the

effects of limited sensor ranges. The work in [23] differed significantly from the others by using mainly internal (angular and linear odometers, and a compass) rather than external sensors. The only external sensor used was a contact sensor at the front of the robot with effectively zero range.

The work presented in this thesis follows [21] by using an omnidirectional sensor which has its field of view divided into discrete segments. The actions of a robot depend on which segments are observed to contain a swarm mate. The range of the camera was infinite in the simulations, however the real world trials have finite range. The use of discrete regions means only course bearing and distance measurements are required by the control algorithm.

The motivation for investigating minimalism in mobile robotics differ between works. The authors of [20, 21, 23] state the goal of their work to be an exploration of the minimal requirements needed to achieve useful behaviours in robots. The work reported in [5, 22] both specifically mention using minimalism for two reasons. First to ease the transfer of the control algorithm from simulation to reality. Secondly, they both discuss how the use of minimalism will facilitate the transfer of control algorithms to nano-robotics which are theorized to have extremely limited computation, sensory, and communication abilities. The work presented in this thesis uses minimalism to both investigate the minimal requirements for pattern formation, as well as for the practical concerns of easing transfer of the control algorithm from simulation to the physical world.

2.2 Decentralized Pattern Formation Techniques

There are many decentralized pattern formation techniques reported in the literature. A selection of the most recent techniques [7–14] will be compared here. The major differences between the various techniques include: the use of unique robot identities or anonymous robots, the information given to the robots, and the sensor data used as inputs to each control algorithm. The similarities and differences between the algorithms will be compared. From the reviewed techniques one which is shown to be most similar to the DNS algorithm will be selected as a benchmark algorithm against which it will be compared.

The use of unique robot IDs has a significant impact on the robustness of the pattern formation algorithm. The algorithms developed in [7], [9], and [11] require each robot to have a unique ID which is detectable by other robots in the swarm. In these algorithms each robot uses distance and/or bearing information to a predefined subset of unique neighbours as inputs. This introduces multiple points of failure into the algorithm, namely if any one of the robots is disabled the swarm will fail to converge to the intended formation. The work in [9] shows how a dynamic subset of neighbours can be used to overcome this limitation, however communication between all robots is required. The algorithms in [8, 10, 13, 14] use data only from visible neighbours as inputs and therefore are more fault tolerant.

The information given to each robot before the start of the pattern formation process differs between algorithms. As previously stated the robots in [7, 9, 11] are given a subset of neighbours from which bearings and/or distances are measured. Other kinds of prior information include agreement on environmental cues. The

algorithms reported in [7] and [11] use a common reference direction (i.e North) in order to localize rotation. Finally the work reported in [14] requires all robots to share a common origin and coordinate system.

The largest differences between algorithms can be seen in the sensor data used as inputs for the control algorithm. All the algorithms assume the robots are able to identify their swarm mates in the environment using some suitable sensor (a camera for instance). The techniques developed in [7–9, 11, 12] use bearing only information. The algorithms reported in [7] and [11] use a global bearing, whereas [12], [8] and [9] use local bearings. The technique developed in [8] guarantees convergence for triangular formations while avoiding robot-robot collisions. This work is extended in [7] to include any parallel rigid formation. The work discussed in [9] explores the scalability of bearing only formations and the use of dynamic interaction graphs. The work in [12] requires vision based sensing and uses image processing techniques in order to provide an estimate of the heading, speed, and time-to-collision with swarm mates. The algorithms in [13] and [14] require the bearing and distance to swarm mates. However, [13] requires only local distance, whereas [14] requires the position of swarm mates in a global reference frame.

In order for a formation control technique to be incorporated into a cooperative behaviour-based control architecture it must converge to its intended formation when combined with other velocity commands. Only the works reported in [8, 9, 11, 12] demonstrate convergence to the intended formation even in the presence of additional velocity commands. The addition of obstacle avoidance to the pattern formation controller in [12] and [11] was accomplished by directly adding a velocity vector from a properly formulated behaviour to the formation control vector. The work performed

in [11] also demonstrated convergence in the presence of a common translation and/or rotation command transmitted to each robot.

The works reported in [7] and [11] stand out from the others by incorporating a simple compass into the robot's sensor suite. The addition of such a sensor does not significantly increase the complexity of the robot. The compass is low power, and does not require any intensive computations to use. The compass allows robots to have a common rotational reference frame, and for a global orientation of the formation to be defined. Due to the limitations in formations available in [7] (triangles only) the technique for decentralized pattern formation developed in [11] will be used as a benchmark when evaluating the effectiveness of the DNS algorithm. The pattern formation algorithm developed in [11] will be referred to as the Static Neighbour Selection (SNS) algorithm.

Chapter 3

The Dynamic Neighbour Selection Algorithm

This chapter will have the following structure. First the methods used to define formations in the Dynamic Neighbour Selection and Static Neighbour Selection (SNS) algorithms will be explained. The range of formations, scalability, flexibility, and robustness of the two formation definitions will then be compared. Next the implementation of the DNS algorithm as a set of behaviours and their incorporation into a behaviour-based controller will be described. Finally, the results from a series of numerical simulations comparing the performance of the DNS and SNS algorithms will be discussed. Note the content in this chapter has been adapted from work previously reported in [18].

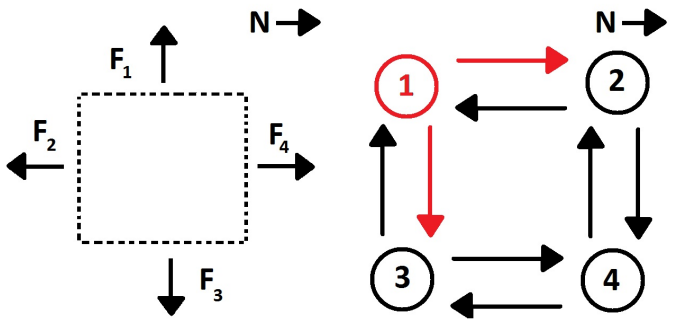
3.1 Formation Definitions

3.1.1 Dynamic Neighbour Selection

The DNS algorithm defines the desired formation by specifying a set of unit vectors, $\left\{\vec{F_i}\right\}$, perpendicular to the formation's edges. Note, for the remainder of the thesis perpendicular refers to a 90 degree rotation counter clockwise (CCW). Figure 3.1(a) illustrates how a square formation is defined by the DNS algorithm. The swarm of robots is divided into teams and each team assigned one vector from the set. During the operation of the algorithm the robots do not need to identify the individual or team ID of another robot.

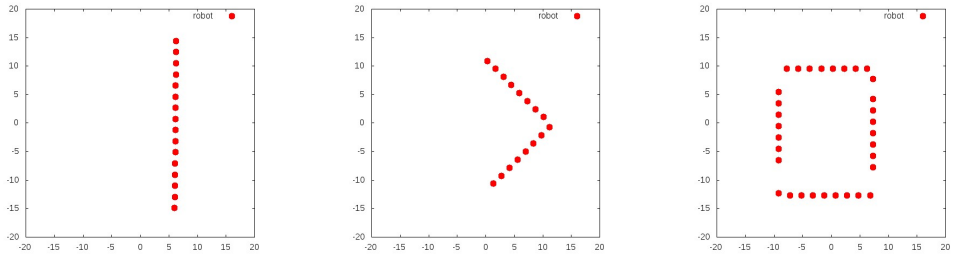
3.1.2 Static Neighbour Selection

The SNS algorithm [11] defines the desired formation by specifying bearing constraints between a robot and a subset of its swarm mates. Each robot must be uniquely identifiable in order for the algorithm to converge to the desired formation. Figure 3.1(b) illustrates how these constraints are defined by the SNS algorithm using the example of 4 robots forming a square. Each robot has a unique identification number (1 to 4), and a set of constraints (target ID and bearing). The bearing constraint is used to construct a unit vector $\vec{f}$ which the algorithm uses to calculate a desired velocity vector.



(a) DNS (b) SNS

Figure 3.1: Defining a square formation using the SNS and DNS algorithms. Each arrow in the figures represents a constraint or edge normal required by the formation definition. All bearings are measured counter clockwise from North as indicated in the top right corner each figure. In (a) the formation definition is given by only 4 values ($\pi/2$, π, $-\pi/2$,0) regardless of the number of robots. In (b) robot 1 has bearing constraints 0 and $-\pi/2$ with robots 2 and 3 respectively. Similar constraints exist for the remaining 3 robots. This means 16 values are needed to define the formation.



(a) Line (b) Wedge (c) Square

Figure 3.2: Examples of the formations studied in simulation. (a) The line maximizes the cumulative field of view (FoV) of sensors perpendicular to the formation. (b) The wedge formation has similar FoV benefits as the line with the added benefit of increased visual contact between robots. (c) The square formation could be useful for perimeter keeping and containment.

3.1.3 Comparison of Formation Definitions

3.1.3.1 Variety of Formations.

The SNS algorithm is able to form any parallel rigid formation [11], including shapes with internal structure. Based on the formations tested in simulation, the DNS algorithm is limited to a line, a wedge, and a square. Although limited, the formations available to the DNS algorithm are useful in certain tasks, such as collective sensing. Example formations are shown in Figure 3.2.

Neither algorithm controls the scale of the formation. This is a result of having only bearing information and coarse distance estimates to define the formation. However, with the inclusion of an obstacle avoidance behaviour, which treats other robots as obstacles, a minimum scale is maintained.

3.1.3.2 Scalability, Flexibility, and Robustness.

The advantages of a swarm robotic system as identified by [3] are scalability, flexibility, and robustness. The DNS and SNS algorithms will be evaluated based on these attributes.

Scalability, as defined by [3], in a swarm robotic system means the same control algorithm can be used regardless of swarm size. Both DNS and SNS are decentralized algorithms, and so both scale in this sense. However, both algorithms were developed with human interaction in mind. In order for a human operator to manipulate the formation new information must be transmitted to the swarm (i.e sets of formation normals, or target IDs and bearing values for DNS and SNS respectively). In order for communication with the swarm to be scalable, it must be independent of group size [9]. Figure 3.1 shows how the information require to define a formation in the SNS algorithm depends linearly on the group size. Therefore, communication with the swarm is not scalable when using the SNS algorithm. Formation definition for the DNS algorithm is independent of group size, and therefore the information require to communicate with the swarm, to change formation, remains constant as the size increases. However, communication would scale linearly with the complexity of the formation (ie. number of edges).

Flexibility in a swarm robotic system is the ability to handle changes in group size [3]. In this event, the SNS algorithm requires changes to the bearing constraints of neighbours of a lost or added swarm member. The DNS algorithm requires no changes to the information stored by the swarm when members are added or removed.

A source of robustness in swarm robotic systems comes from unit redundancy

[3]. That is any member of the swarm can take on the role of another member of the swarm (assuming the robots are homogeneous). The DNS algorithm maintains unit redundancy by not requiring neighbours to be uniquely identifiable. The SNS algorithm requires robots to be uniquely identifiable and so lacks unit redundancy.

3.2 Control Laws

3.2.1 DNS Behaviour Based Controller

The Dynamic Neighbour Selection (DNS) algorithm is implemented via a competitive behaviour based controller [6]. The algorithm is composed of five basic behaviours;

- Avoid Collision
- Forward
- Back
- Alter Course
- Stop

Each of these behaviours responds to sensor stimuli with a velocity vector. The behaviours are coordinated by a competitive type architecture where a single behaviour is active at a time. Details of the stimulus and responses of the beahviours, as well as the method by which behaviour selection is achieved will be discussed in the following sections.

Behaviour selection is achieved by dividing the field of view (FoV) of the omni-directional camera into four discrete regions (see Figure 3.3). This is similar to the

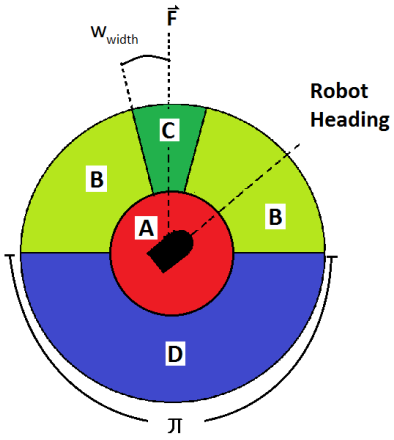


Figure 3.3: The sensing regions are defined with respect to the formation normal, $\vec{F}$, and not the orientation of the robot. The width of region C is controlled by the ω_{width} parameter, and the radius of region A is defined by the r_{avoid} parameter.

method of behaviour selection used in [21]. The FoV is divided into an avoidance region labelled A (the inner red circle), and three manoeuvring regions labelled B (light green), C (dark green), and D (blue). The division of the FoV into these discrete regions relies more on accurate bearing measurements than accurate distance measurements since the robot only needs to distinguish distances to other robots as either "far away" (outside region A) or "too close" (inside region A). This is in keeping with the constraint of minimalist robotics since bearing can be measured by a less complex sensor than distance[1]. Furthermore the calculations of the behaviour

[1]For example, bearing measurements could be made using a reflected outgoing signal (ex. infrared light or ultrasound) to detect objects. If a sensor pointing in the direction ϕ with a FoV of $\pm\delta$ detects a reflected signal, then an object is somewhere in its FoV. If such a sensor detects a reflected signal then the bearing to the object has been measured, namely $[\phi - \delta, \phi + \delta]$. Distance on the other hand needs to take into account the magnitude of the reflected signal which depends on the distance to the object as well as the reflectivity of the object's surface.

State	Behaviour
1,3,5,7,9,11,13,15	Avoid Collision
4,6,12,14	Alter Course
8	Backwards
2,10	Forward
0	Stop

Table 3.1: Robot sensor state and corresponding behaviour.

responses require only bearing measurements as inputs. When a swarm mate is detected in a region, that region is considered active, otherwise the region is inactive. The combination of activated regions, referred to as a sensor state, determines which behaviour is selected. The encoding of the robots sensor state as a 4-bit binary number is similar to the encoding of the robots sensor state in [5] which used a 2-bit sensor state. Since there are four regions, each with the possibility of being active or inactive, this leads to 16 sensor states. Each region is assigned a value (A = 1, B = 2, C = 4, D = 8). The sensor state is determined by summing the values of the activated regions. Table 3.1 lists each sensor state and its corresponding behaviour. The details of each of the behaviours are given in the following paragraphs.

Avoid Collision Behaviour. The input for the collision avoidance behaviour is a vector, $\vec{r}$, in the direction of the detected obstacle. If there is no obstacle within region A, then the response is $\vec{v}_{obst} = \vec{0}$. The behaviour response in the presences of an obstacle, $\vec{v}_{obst}$ is given by Equation 3.1,

$$\vec{v}_{obst} = R(\gamma_{avoid})\vec{r} \tag{3.1}$$

where $\vec{r}$ is the bearing to the neighbour in the avoidance zone, R is the rotation matrix, and γ_{avoid} is a scalar with value. For the simulated results γ_{avoid} was set to 180^o. However, this value was changed during the live trials for reasons discussed in Section 5.2.

Alter Course Behaviour. This behaviour causes a robot to move perpendicular to its formation normal until its FoV region C is deactivated. This behaviour causes robots in the same group to spread out along the edge defined by the group's formation normal. Unlike other behaviour responses, which are functions of the bearing to another robot, the alter course behaviour has no such dependence. This behaviour is an example of a discrete behaviour (see Section 1.2). The behaviour response, $\vec{v}_{alt}$, is given by Equation 3.2,

$$\vec{v}_{alt} = \vec{F^{\perp}} \tag{3.2}$$

where $\vec{F}$ is the formation normal assigned to the robot, and $\perp$ means a counter clockwise (CCW) rotation of 90 degrees. The choice to always rotate CCW means that robots travelling towards each other will always turn in opposite directions. Turning clockwise would have the same effect.

Forward Behaviour. This behaviour causes the robot to move along the formation normal defined by $\vec{F}$. Its speed decreases as the robot gets closer in line with the forward most robot. Forward in this case means along the formation normal. The sensor input for the Forward behaviour is the set of unit vectors, $\{\vec{r_i}\}$, encoding the

bearings to all visible swarm mates. The response vector, $\vec{v}_{fwd}$, is given by Equation 3.3,

$$\vec{v}_{fwd} = \max(\vec{r}_i \cdot \vec{F})\vec{F} \tag{3.3}$$

where $\vec{r_i}$ is the bearing to the i^{th} robot. The function $max(\cdot)$ is equal to the largest dot product between $\vec{F}$ and a member of the set of bearing vectors $\vec{r}_i$.

Backward Behaviour. This behaviour causes the robot to travel anti-parallel to the formation normal. The Backward behaviour uses the same input as the Forward behaviour, $\{\vec{r}_i\}$. The response vector, $\vec{v}_{bwd}$, is given by Equation 3.4,

$$\vec{v}_{bwd} = \min(\vec{r}_i \cdot \vec{F})\vec{F} \tag{3.4}$$

where $\vec{r_i}$ is the bearing to the i^{th} robot. The $min(\cdot)$ function is the opposite of $max(\cdot)$, and causes the robot to move backward along the formation normal, $\vec{F}$.

Stop Behaviour. As the name suggests this behaviour brings the robot to a halt. Similar to the Alter Course behaviour the Stop behaviour takes no input. The response vector, $\vec{v}_{stp}$, is predictably the zero vector, $\vec{0}$.

3.2.2 SNS Behaviour Based Controller

The Static Neighbour Selection algorithm is implemented through a cooperative behaviour controller as described in [11]. The controller is comprised of one copy of the tangential behaviour for each neighbour assigned to the robot (see Section 3.1.2 for an explanation of neighbour assignment) and a collision avoidance behaviour. As in the case of the DNS algorithm, each behaviour responds to stimuli with a target

velocity vector. The cooperative approach to behaviour selection sums the resulting behaviour velocity vectors into a single velocity command,

$$\vec{v_{cmd}} = \sum_i \vec{v_i} \tag{3.5}$$

Where $\vec{v_{cmd}}$ is the velocity command, and $\vec{v_i}$ is the i^{th} behaviour's velocity vector with magnitude bounded between [0,1]. Details of the tangential behaviour are given below. The collision avoidance behaviour is the same as the one used by the DNS algorithm.

Tangential Behaviour. The input for the tangential behaviour is a unit vector, $\vec{r}$, in the direction of the target robot specified by the bearing constraint. (Section 3.1.2). The behaviour response, $\vec{v_{tan}}$ is given by Equation 3.6,

$$\vec{v_{tan}} = (\vec{r} \cdot \vec{f^{\perp}})\vec{r^{\perp}} \tag{3.6}$$

where $\vec{f^{\perp}}$ is perpendicular to the target bearing, $\vec{f}$, associated with the target robot, and $\vec{r^{\perp}}$ is the vector perpendicular to input $\vec{r}$. This behaviour causes the robot to travel along a circular arc centred on the target robot until the target bearing is achieved. The diagram shown in Figure 3.4 illustrates the tangential control law.

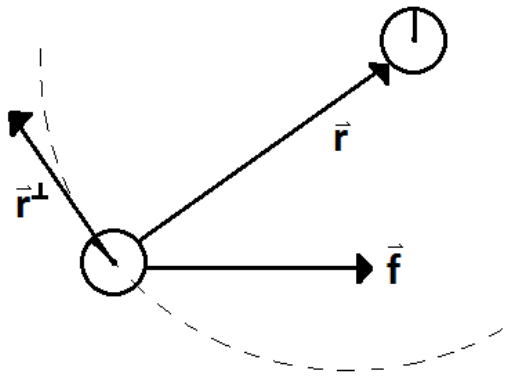


Figure 3.4: Graphical description of the tangential control law used by the SNS pattern formation algorithm. Where $\vec{f}$ is the bearing constraint vector, $\vec{r}$ is the bearing vector to the target robot, and $\vec{r}^{\perp}$ is perpendicular to $\vec{r}$.

3.3 Algorithm Evaluation

A single integrator simulation was used to evaluate the performance of both algorithms (available for download from GitHub[2]). The performance of the algorithms was evaluated based on the mean integrated path length, α, of all robots in the swarm (see Equation 3.7). This metric was chosen because the distance travelled by the robot has a large effect on the robots' power consumption. It can generally be agreed that lower power consumption is a useful attribute for a mobile robot, since its on-board power supplies are limited.

$$\alpha = \frac{\sum_i d_i}{N} \tag{3.7}$$

[2]https://github.com/nicholishiell/DiskSimulation

Where d_i is the integrated path length of the i^{th} robot, and N is the total number of robots in the swarm. The integrated path length of each robot is defined by Equation 3.8.

$$d_i = \sum_j v_i(j)\Delta t \tag{3.8}$$

Where d_i is the integrated path length of the i^{th} robot, the sum is over all time steps j, the velocity of the i^{th} robot at time step j is $v_i(j)$, and Δt is the time step used in the simulation.

The robots were modelled as holonomic circles with radius r_{robot} and equal masses of m_{robot}. Collisions between robots were approximated using two dimensional kinematics. The simulation cycle was as follows; robots were updated with sensor data (global bearing to all neighbours), then a velocity response for each robot was calculated, lastly robot positions were updated and collisions handled. This loop was repeated until a maximum number of time steps, t_{max}, was reached. Table 3.2 summarizes the parameters used by the simulation. It was assumed the robots were always visible to each other regardless of range or line of sight. This assumption is beneficial to both algorithms since it means a robot is never left with no sensor inputs. This is specifically advantageous to the SNS algorithm since robots require sensor data about specific neighbours. The simulation was initialized by randomly distributing the robots in a circle of radius r_{deploy}. Figure 3.5 and Figure 3.6 shows some screen shots taken from the simulation.

Parameter	Value	Description
Δt	0.25	Length of time step
t_{max}	10000	Max number of time steps
r_{deploy}	50	Radius of initial deployment
ω_{alt}	25^o	Angular width region C
r_{avpid}	20	Radius of region A
r_{robot}	5.0	Robot radius
m_{robot}	1.0	Robot mass

Table 3.2: Summary of parameters used during the evaluation simulations.

3.3.1 Simulation Results

The performance of both algorithms was evaluated when constructing line, wedge, and square formations with various group sizes (8, 16, 20, and 32 robots). Each set of evaluation parameters (algorithm, formation, group size) was simulated 100 times and the values of the performance metric recorded for each run. The average metric values and standard deviation over all runs were calculated, and the results shown graphically in Figures 3.7, 3.8, and 3.9.

The standard deviation of α values associated with the SNS algorithm are relatively large compared to the DNS algorithm. This shows there was a strong variation in average integrated path lengths between runs with the same formation and group size. The only difference between these runs was in the initial positions of the robots, which were uniformly distributed in a circle of radius r_{deploy}. This indicates that the SNS algorithm depends more strongly on initial deployment than the DNS algorithm.

The performance of the algorithms, as measured by α, diverge quickly for group sizes larger than 8. The SNS algorithm shows a stronger dependence on group size than the DNS algorithm, and this trend can be seen in all formations tested. Therefore, the performance of the DNS algorithm scales better with group size than the SNS algorithm.

Most important among the results is the ability of the DNS algorithm to successfully converge to the desired formations. This shows that a limited set of formations is achievable without the use of robots with unique identities, and with a limited amount of *a priori* information given to the robots. Although the DNS algorithm has improved upon some aspects of decentralized pattern formation algorithms, it still has limitations particularly those common to bearing-only algorithms. The relative lengths of polygon segments are not controlled, and the density of robots along a segment of the formation is not uniform.

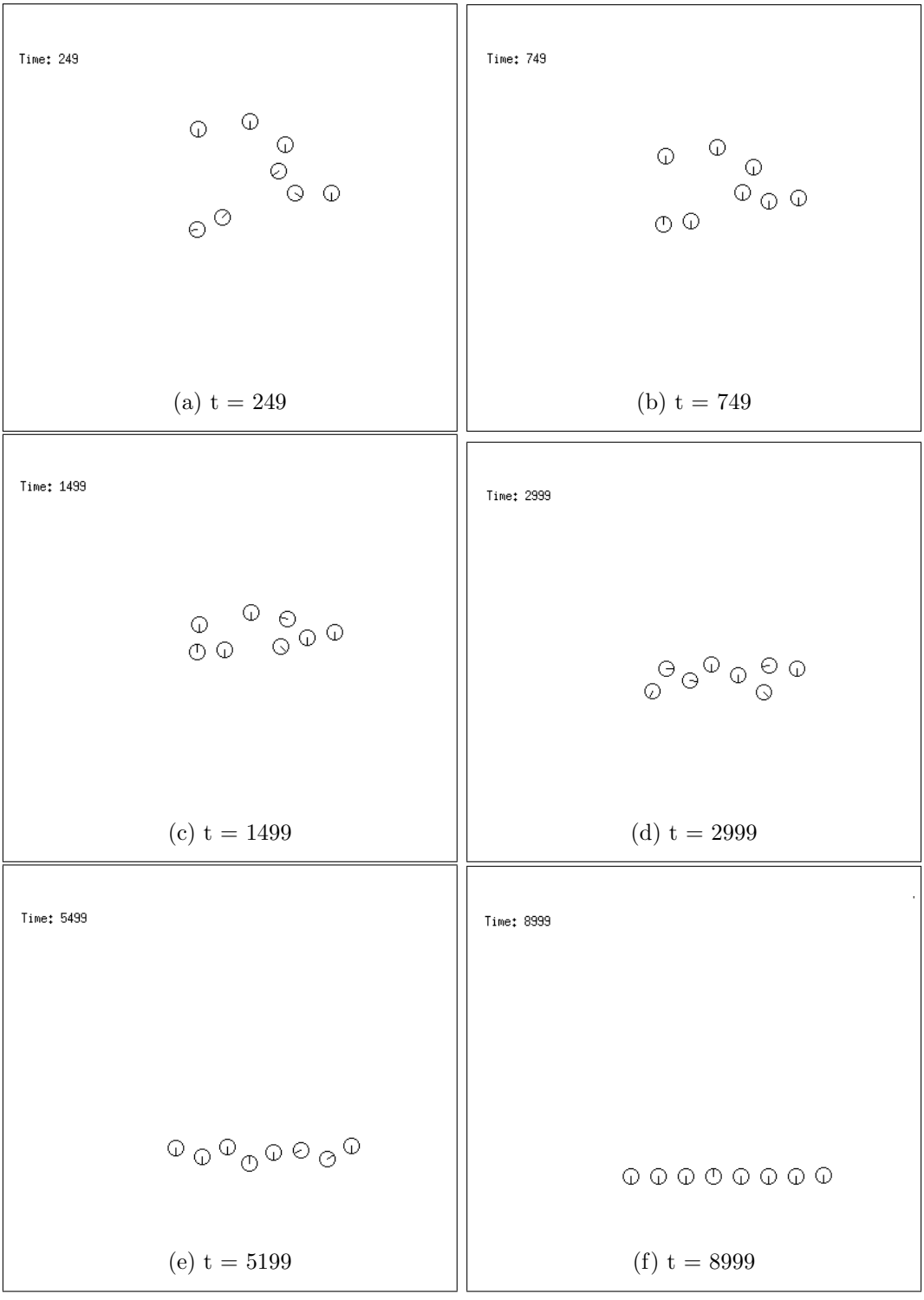


Figure 3.5: Screen captures from the simulation used to evaluate the performance of the DNS and SNS algorithms. In this example 8 robots are simulated using the DNS algorithm to form a line.

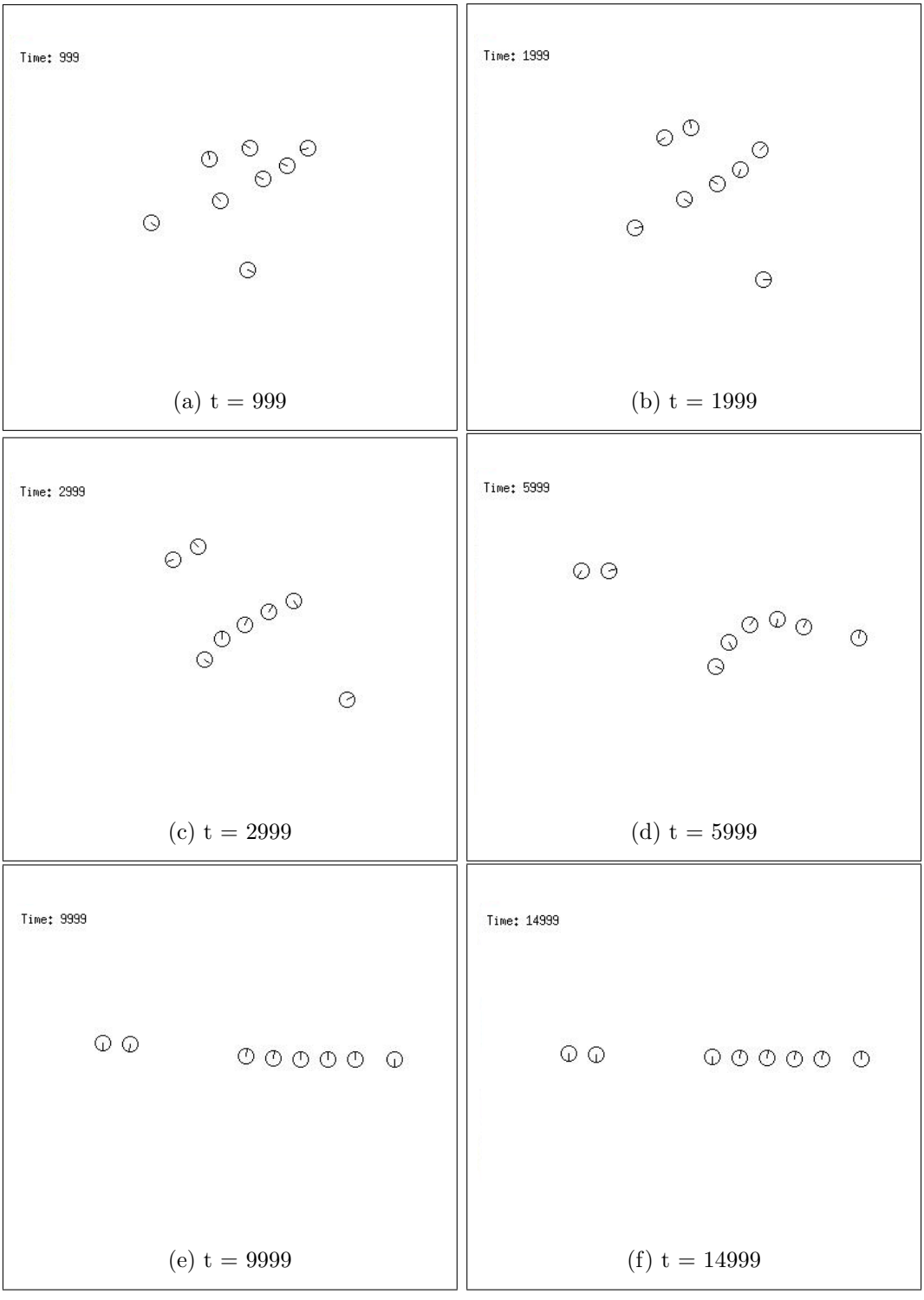


Figure 3.6: Screen captures from the simulation used to evaluate the performance of the DNS and SNS algorithms. In this example 8 robots are simulated using the SNS algorithm to form a line.

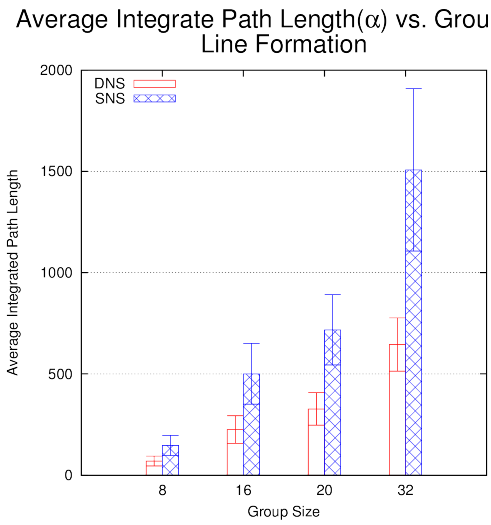


Figure 3.7: Line Formation results of mean integrated path length averaged over 100 trials.

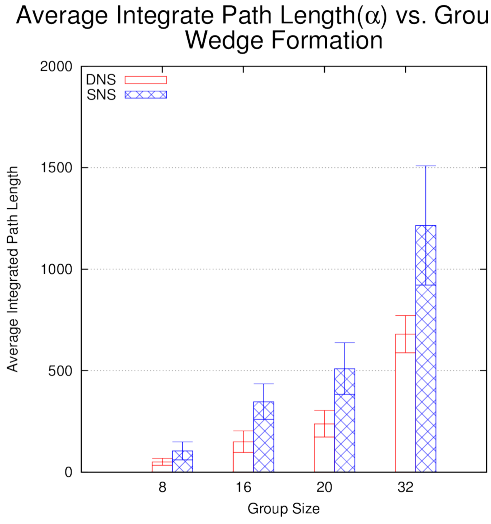


Figure 3.8: Wedge Formation results of mean integrated path length averaged over 100 trials.

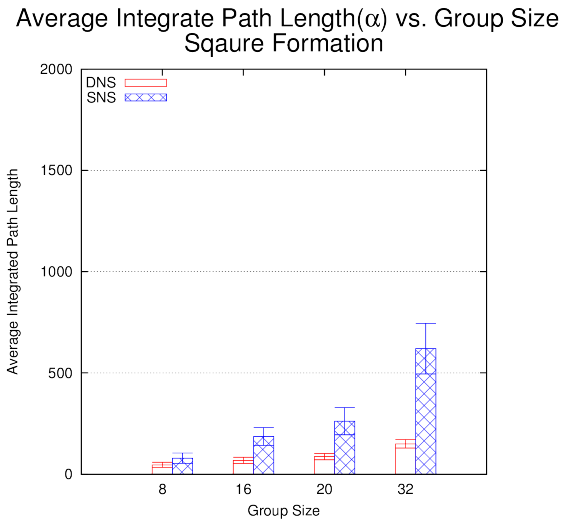


Figure 3.9: Square formation results of mean integrated path length averaged over 100 trials.

Chapter 4

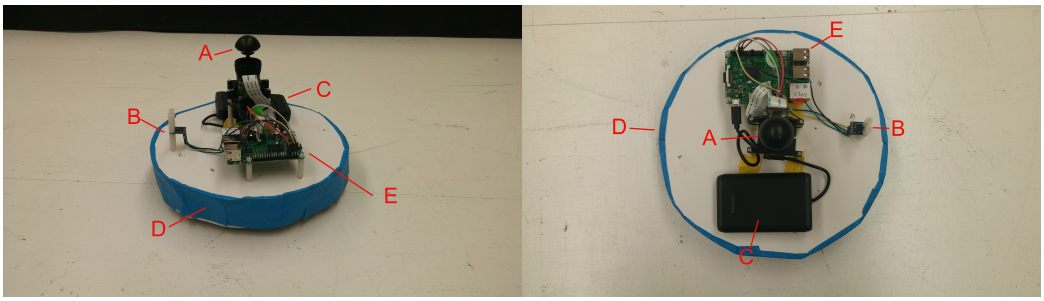
The Bupimo Swarm Robot

This chapter explains the hardware and software of the Bupimo robot. The hardware section includes descriptions of the robot chassis, on-board computers, and sensors. The techniques used to calibrate and analyze sensor data are also discussed. The software section describes how the Robot Operating System (ROS) [25] software framework was used to connect the various physical components of the Bupimo into a functioning mobile robot.

4.1 Bupimo Overview

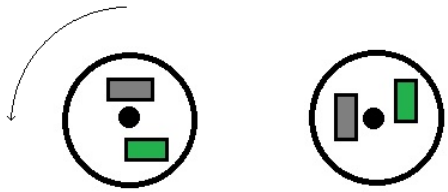
The Bupimo, shown in Figure 4.1, is a low-cost differential drive robot with omni-directional vision. To reduce cost it has been constructed largely from off the shelf components. The name Bupimo is derived from the primary components of the robot; "Bu" is for BubbleScope (see Figure 4.4), "pi" for the Raspberry Pi 3 single board computer (see Figure 4.7), and "mo" for Zumo a small (10cm x 10cm) robot chassis produced by Polulu Robotics (see Figure 4.3). The BubbleScope is a smart phone

attachment which uses a hyperbolic mirror to capture panoramic images. When coupled with a Raspberry Pi camera an effective low cost omnidirectional vision sensor is formed. The Raspberry Pi 3 provides high level computing power required to process image sensor data and execute the robot's control software. The Zumo provides the differential drive platform for the Bupimo, as well as a motor control board, wheel encoders, a serial port for communicating with the Raspberry Pi, and a microcontroller to handle low level computing. The base of the robot is covered in a distinct colour of blue which facilitates the identification of the Bupimo in images captured by an observer. The remainder of this chapter will give details on the hardware, sensor calibrations, and software used to fuse the separate components of the Bupimo into a useful mobile robotic system.

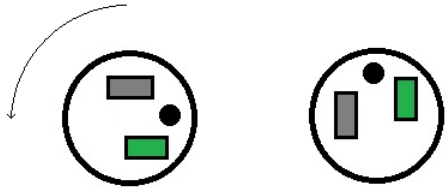


(a) Side view. (b) Top down view.

Figure 4.1: The major components of the Bupimo robot are labelled in both the side and top down views. (A) The omnidirectional camera is located directly above the robots center of rotation. This prevents the images observed by the camera from shifting while the robot rotates (See Figure 4.2). (B) 3 Axis Digital Compass. (C) Battery pack. (D) Coloured Skirt. (E) Raspberry Pi 3. The Zumo is not visible in these pictures.



(a) Centered camera



(b) Off center camera

Figure 4.2: These figures show the effects of positioning the camera at the robot's center of rotation versus off center. The camera is represented by the black circle, the black rectangle is the battery pack located at the front of the robot, and the green rectangle represents the Raspberry Pi located to the rear of the robot. In both images the robot is shown to be rotating to the left. (A) When the camera is off center it does not translate as the robot rotates. (B) When the camera is off center its position changes significantly as the robot rotates.

4.2 Hardware

4.2.1 Locomotion

The Zumo provides the ability for the Bupimo to move through the environment via a differential drive system. The Zumo includes two brushed DC motors with 100:1

ratio gear boxes, a dual H-Bridge motor driver, a power supply (4 AA batteries), two wheel encoders, serial communications, and an Arduino compatible 8-bit Atmel AVR microcontroller. The Zumo also has a number of other sensors (IMU, IR proximity sensors, battery power monitor, etc.) however these are not currently used by the Bupimo. The microcontroller was programmed using the Arduino IDE. It accepts tokenized motion control messages over its serial port, and uses a simple PID controller with feedback from the wheel encoders to match the robot's motion to the message received. The format of these messages are show in Equation 4.1.

$$v_l : \omega : t \tag{4.1}$$

Where v_l is the linear speed given in m/s, ω is the rotational speed given in $rads/s$, and t is the length of time, in seconds, the robot should maintain the given motion. If the time argument has a value of -1 the robot maintains the motion until a new message is received.

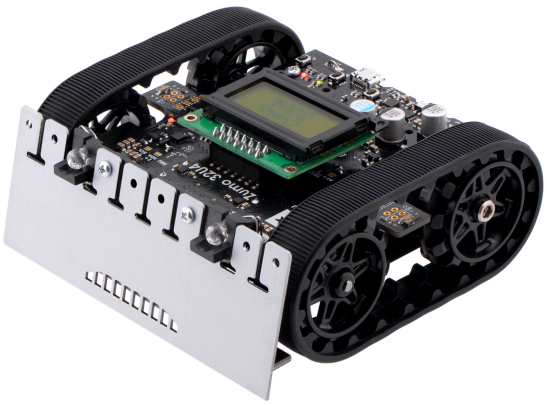


Figure 4.3: The Zumo 32u4 from Polulu Electronics. Note the plough at the front has been removed for use in the Bupimo robot.

The speed of the Zumo's motors are controlled by two integer values m_r (right motor) and m_l (left motor). These values are between -400 (full reverse) and 400 (full speed ahead). The value of m_r and m_l are determined by a PID controller which models the Zumo as a differential drive system. The controller calculates target left and right wheel speeds, $w_{l,t}$ and $w_{r,t}$ respectively, based on the linear and rotational speeds received over the serial port. This calculation is shown in Equation 4.2.

$$\begin{aligned} w_{r,t} &= (2v_l + \omega L)/2R \\ w_{l,t} &= (2v_l - \omega L)/2R \end{aligned} \tag{4.2}$$

where v_l and ω are the desired linear and rotational speeds, L is the base length of the robot ($8.5cm$), and R is the radius of the wheels ($1.9cm$).

The controller compares the target wheel speeds to the current wheel speeds to determine how to adjust the motor control values. The current wheel speeds are determined using the data from the wheel encoders, and Equation 4.3.

$$\begin{aligned} w_{r,c} &= 2\pi\frac{c_r}{C}\Delta T \\ w_{l,c} &= 2\pi\frac{c_l}{C}\Delta T \end{aligned} \tag{4.3}$$

where ΔT is the time between samples, c_r and c_l are the encoder counts measured since the last sample, and C is the number of counts per rotation (which is 1204.44 for the Zumo).

To determine the correct m_r and m_l values which will produce the target linear and rotational speeds the difference between the target wheel speeds, and the current wheel speeds is used. This difference is then multiplied by a constant k_p of 10.0. This calculation is shown in Equation 4.4.

$$m_r = m_r + K_p(w_{r,d} - w_{r,c})$$
$$m_l = m_l + K_p(w_{l,d} - w_{l,c}) \tag{4.4}$$

If the values for the motor control inputs exceed the bounds of [-400, 400] they are truncated at the max/min values.

4.2.2 Sensor

The primary external sensors of the Bupimo are a digital compasses and an omni-directional camera. This section will describe each sensor, its calibration procedure, and how it is used by the Bupimo.

4.2.2.1 Digital Compass

To measure its orientation in a global reference frame the Bupimo uses the HMC5883L [26] triple axis digital compass from Honeywell. The low cost, low power consumption IC has customizable field range (±0.88Ga to ±8.1Ga), output data rate (0.75Hz to 75Hz), and communicates with the Bupimo's main computer using the I2C serial communication protocol. The field range and output data rate used by the Bupimo are ±1.3Ga and 30Hz respectively. These settings are the default values recommended by the HMC5883L reference manual. The compass uses a 12-bit ADCs with low noise amplifiers to convert the analog sensor signals into signed integer values. The compass is oriented so its x-axis is aligned with the forward direction of the robot, and the z-axis points away from the floor.

The channels of the compass must be calibrated in order to provide accurate orientation sensing. Since the robot moves in 2 dimensions only the x and y channels

are calibrated. To calibrate the sensor the robot rotates about its mid point approximately three times. During these rotations the highest (b_{max}) and lowest (b_{min}) ADC readings measured by each axis is recorded. These values are used to scale the measurements of the local magnetic field using Equation 4.5.

$$b_{scaled} = 2\frac{b_{ADC} - b_{min}}{b_{max} - b_{min}} - 1 \tag{4.5}$$

Where b_{ADC} is the value read directly from the ADC, and b_{scaled} is the projection of the normalized local magnetic field vector onto either the x- or y-axis. The Bupimo's orientation can be calculated from the x and y channel data using Equation 4.6.

$$\theta = atan\left(\frac{b_{scaled,y}}{b_{scaled,x}}\right) \tag{4.6}$$

where θ is the orientation of the robot, and $b_{scaled,x}$ and $b_{scaled,y}$ are the scaled ADC values for the x and y axis' respectively.

The Zumo has a built in digital compass incorporated into its on-board IMU. However, the IMU is positioned in close proximity to the Zumo's motors. When engaged, the motors create significant magnetic interference, rendering the compass readings useless. For this reason an external compass was added and positioned away from the motors (see Figure 4.1).

4.2.2.2 Omnidirectional Camera

The omnidirectional vision sensor is the main sensor on the Bupimo used by the DNS pattern formation algorithm. It is used to measure the bearing to neighbouring robots, as well as determining a rough distance used for obstacle avoidance. This section will

briefly describe the construction of the camera, how the camera is calibrated, and the image analysis technique used to measure bearing and distance to neighbours.

(a) BubbleScope (b) RPi Camera Module

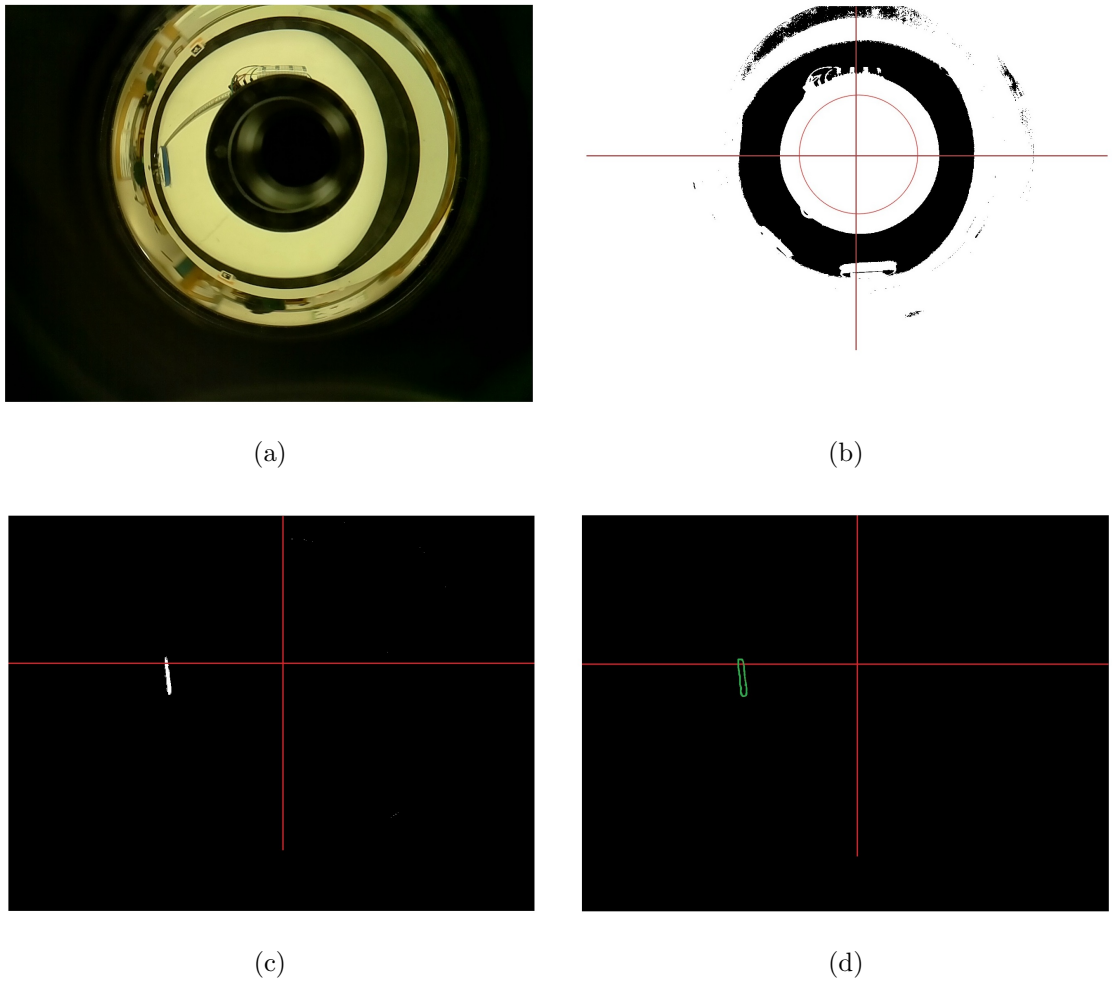
Figure 4.4: The BubbleScope (a) and the Raspberry Pi Camera module (b) are used to construct the Bupimo's omnidirectional vision sensor.

The Bupimo's omnidirectional vision sensor is built out of three basic components; the BubbleScope (Figure 4.4(a)), the mount (not pictured), and the Raspberry Pi Camera Module (Figure 4.4(b)). The BubbleScope uses a hyperbolic mirror to reflect light from all directions down to an angled mirror contained in the BubbleScopes body. The angled mirror reflects the light through a small aperture in the side of the BubbleScopes body, similar to a periscope. The image is then captured by the Raspberry Pi Camera attached to the BubbleScope's aperture. The mount supports, and attaches the Raspberry Pi camera to the aperture of the BubbleScope. An example of a typical image capture by the camera is show in Figure 4.5(a). The

region of interest (ROI) of the image is centred on the black circle created by the pillar supporting the hyperbolic mirror. The position of the center of mass of this black region (x_{roi}, y_{roi}) is calculated and used to translate from an image coordinate system with its origin in the lower left corner, to one with an origin at the center of the ROI (See Figure 4.5(b)).

The Bupimo detects fellow robots using a basic colour threshold filter. The base of each Bupimo is covered in a distinctive colour of blue, which is separated from the background using the colour threshold filter. This filtering technique uses a range of RGB values (r_{min}, r_{max}, g_{min}, g_{max}, b_{min}, b_{max}) to either accepted or reject pixels. Each pixel of the image is broken down into its RGB values. Pixels whose RGB values fall within the filter ranges are set to white, while pixels which fall outside the range are set to black. The exact values used to define the acceptable range are unique to each camera, due to differences in camera alignment and sensitivity. The effects of applying this filter can be seen in Figure 4.5(c). After the filter is applied, connected groups of white pixels are grouped together into so-called blobs (See Figure 4.5(d)). The center of mass (CoM) of each blob is then calculated which gives its position with respect to the image's coordinate frame. The CoM position (x_{blob}, y_{blob}) of each blob is transformed to the ROI frame using the values x_{roi}, y_{roi} described previously. The bearing to the CoM is calculated with respect to the axis shown in Figure 4.5(b). The forward direction in the robot's frame points to the top of the image. Finally, the local bearing is converted to a global bearing by subtracting the robots orientation as measured by the on-board compass.

This basic colour threshold filter is not very robust. It has a strong dependence on the ambient light in the environment, and is only applicable in a tightly controlled

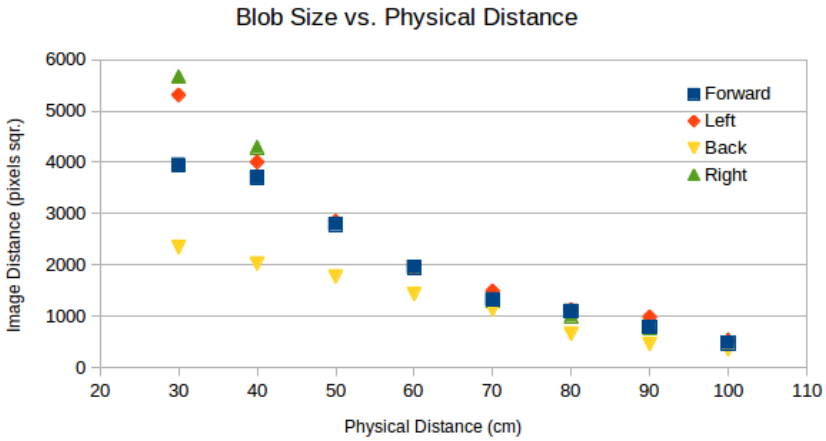


(a) (b)

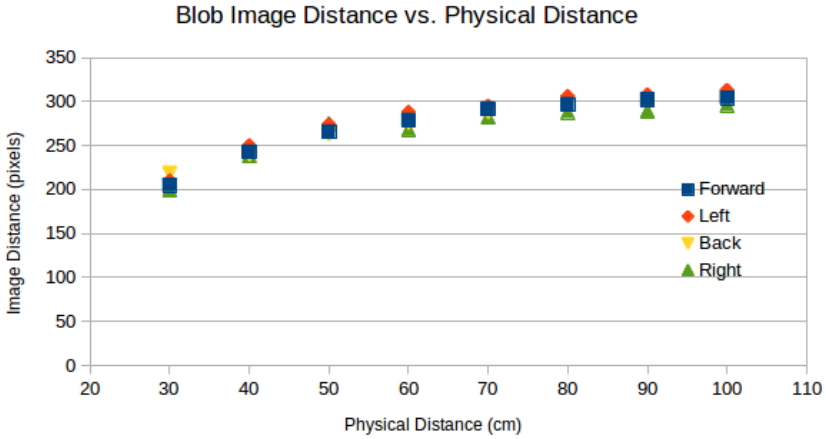
(c) (d)

Figure 4.5: (a) The raw image captured by the omnidirectional camera. (b) The red circle is fitted to the central black region of the raw image. This is used to calculate the centre of the image's ROI, which is used as the origin of the coordinate system. (c) The results of applying the colour threshold filter.(d) The blob formed after collecting spatially connected pixels from image (c).

setting. However, the DNS algorithm is intended for use with minimalist robots that may not be capable of image processing, or too small to contain a camera. The use of the Bupimo's camera is used as an analogy for a more basic bearing only sensor, and so a more robust technique was not investigated.



(a)



(b)

Figure 4.6: Estimates of distance to detected robot using (a) blob size (b) distance from blob to center of ROI.

Two methods were evaluated for estimating the distance to robots using images captured by the omnidirectional camera. The first method of distance estimation takes advantage of the circular shape of the Bupimo robot. A rotationally symmetric object will produce a constant sized blob in an observer's image regardless of its orientation. The size of the blob created by the filtering technique described above is directly proportional to the distance from the observer. Figure 4.6(a) shows the results of using blob size to estimate distance. Four different data sets were measured, where the observed robot was placed directly in front (Forward), 90 degrees to the left (Left), 90 degrees to the right (Right), and directly behind (Back). The data shows there are significant asymmetries in the camera which results in different blob sizes measured at the same distance. In addition to asymmetries in blob size measurements due to direction, the size can also be affected in other ways. For instance, if the observed robot is partially obscured its blob will appear smaller and thus be perceived to be further away then it actually is. The observed size of the blob could also be affected by the lighting in the environment which could change the amount of blue which passes through the colour threshold filter.

The second method for distance estimation exploits the geometry of the mirror used in the BubbleScope. The image created by the mirror causes objects which are further way to appear further from the center of the image. Figure 4.6(b) shows the results of measurements taken using this method. The distance from center method does not suffer from the same asymmetries as the blob size method. For this reason blob distance from the origin will be used to estimate distances to swarm mates during the live trials of the DNS algorithm.

4.2.3 Computing

The main computing power of the Bupimo robot is delivered by the Raspberry Pi 3 (referred to as the RPi) single board computer (See Figure 4.7). It runs the Raspbian Linux distribution, and has the Robot Operating System (ROS) [25] software framework installed. The RPi processes data from sensors, communicates with the Zumo, executes the control software for the robot, and provides wireless network access to the robot.

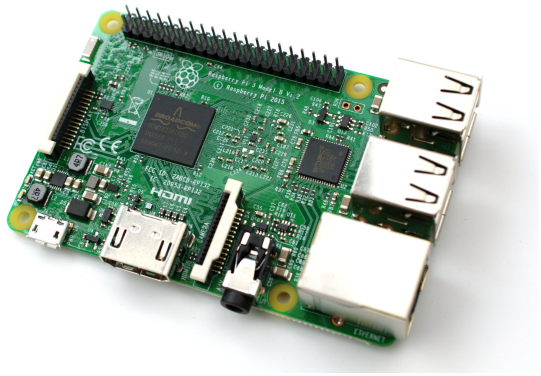


Figure 4.7: The Raspberry Pi 3 provides high level computing to the Bupimo.

4.3 Software

The Bupimo's control software was written using the Robot Operating System (ROS) [25] software framework. Though ROS is not a true operating system, it does provide many of the services of an operating system. Among these services are the handling of multiple processes, and facilitating message passing between them. In ROS, processes are referred to as nodes, and the communication between them is carried out using a publisher/subscriber pattern, otherwise known as a peer-to-peer network. In

this type of message passing pattern a single publisher node writes to a topic which one or more nodes subscriber to. Messages can be either a single typed field or a data structure containing many typed fields. Typically the control software for a robot is composed of a number of nodes each performing a particular task such as; communicating with a sensor, controlling motor speeds, or using sensor data to localize the robot. The layout of the control software can be viewed as a graph (see Figure 4.8) where the vertices are nodes (ovals) or topics (rectangles) and the edges represent message passing between nodes and topics (solid lines represent a node publishing to a topic, and a dashed line represents a node subscribing to a topic). The Bupimo's control software is composed of 5 nodes which communicate through 4 topics. The remainder of this section will describe the function of each node, and how it communicates with the rest of the control software.

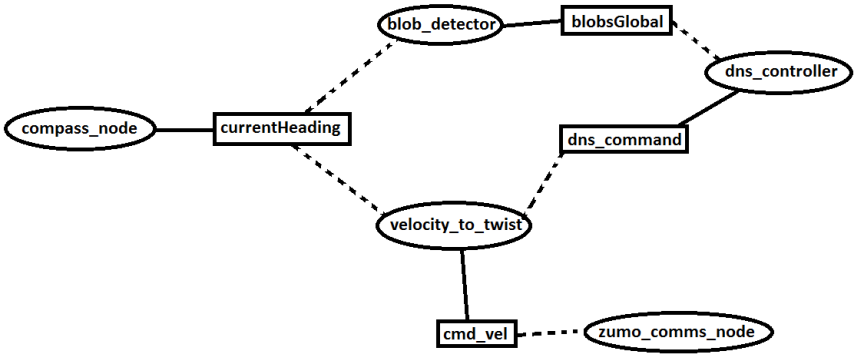


Figure 4.8: This figure illustrates the interaction of the various ROS nodes and topics which make up the Bupimo's control software. The nodes are represented as ovals, topics are rectangles, and in going and out going messages are represented by solid and dashed lines respectively.

The **dns_controller** node implements the behaviour based Dynamic Neighbour Selection pattern formation algorithm. It subscribes to the "blobsGlobal" topic which publishes the bearings and approximate distances (measured in the image frame) to all detected swarm mates. This data is used to determine the sensor state (see Section 3.2.1) and determine the appropriate behaviour response. The behaviour response is then published to the "dns_command" topic. In order to function the node requires three parameters to be set ω_{alt}, r_{avoid}, and $\vec{F}$. The first two parameters, ω_{alt}, and r_{avoid}, define the sizes of the C and D regions of the discretized FoV of the omnidirectional camera (see Section 3.2.1). The final parameter $\vec{F}$ defines the direction of the formation normal in the global reference frame (see Section 3.1.1).

The **velocity_to_twist** node translates the velocity commands (which are in the form of a heading and speed) from the "dns_command" topic into linear and rotational speeds. It subscribes to the "currentHeading" topic which gives the current orientation of the robot with respect to North. It then uses a simple PID controller (see Section 4.2.1) to determine the linear and rotation velocities required to achieve the target velocity. These values are then published to the "cmd_vel" topic.

The **zumo_comms_node** handles communication between the Raspberry Pi 3 and the Zumo microcontroller. This node subscribes to the "cmd_vel" topic and transmits any messages it receives to the Zumo using the message format described in Equation 4.1.

The **blob_detector** node interfaces with the Raspberry Pi Camera module attached to the BubbleScope, and implements the colour segmentation technique described in Section 4.2.2.2. This node subscribes to the "currentHeading" topic, in order to convert bearings to blobs from the local frame of the robot to the global

frame. The bearing, and approximate distances (measured in pixels from the center of the ROI) are then published to the "blobsGlobal" topic. The "blob_detector" node requires eight parameters to be set. The first two are the position of the center of the ROI of the image (x_{roi}, and y_{roi}). The other six parameters are min/max RGB values used for the colour threshold filter.

The **compass_node** interfaces with the HMC5883L triple axis digital compass. This node uses readings from the compass' x- and y-axis' to determine the current heading of the robot. The orientation of the robot, measured in degrees, is then published to the "currentHeading" topic. The node requires four parameters x_{min}, x_{max}, y_{min}, and y_{max} which are used to calibrate the compass readings. The calibration process used is described in Section 4.2.2.1.

Chapter 5

Real World Trials

This chapter describes the various tests that were conducted to demonstrate both the functioning of the Bupimo robots, as well as to evaluate, and calibrate the DNS algorithm. Two formation types were tested using the DNS algorithm, linear and wedge, and the results recorded using an overhead camera. The results of these tests are then discussed and useful lessons highlighted.

5.1 Experimental Setup

Four Bupimo robots were available for the live trials of the DNS algorithm. The trials were recorded using an overhead camera which captured a 640x480 resolution image every 0.25 seconds. In order to distinguish individuals, each robot had a coloured marker attached to its battery pack. The four colours were green, yellow, pink, and purple. The tests were conducted in a square arena with dimensions of approximately 2.5m. Due to the limited number of robots, only two formations types, linear and wedge, were evaluated.

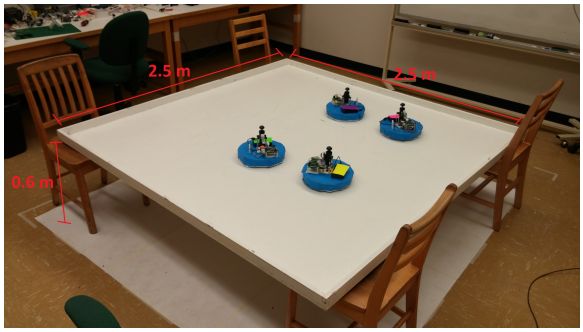


Figure 5.1: The Arena in which the live trials of the DNS algorithm were conducted. Each side of the approximately 2.5m. The arena is suspended approximately 60cm above the ground to avoid magnetic anomalies in the floor.

5.2 DNS Implementation on Bupimo Robots

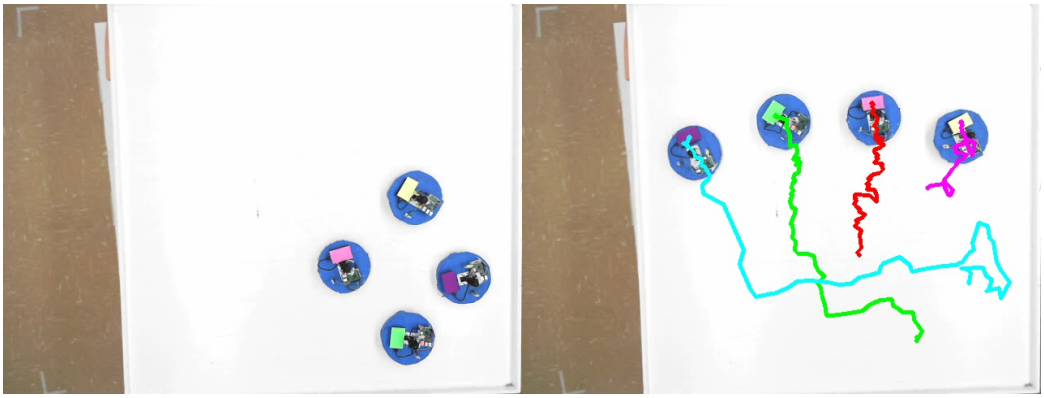
Before testing of the DNS algorithm could be conducted, four parameters (See section 3.2.1) needed to be set. These parameters are: the formation normal ($\vec{F_i}$), avoidance radius (r_{avoid}), angular width of region C (ω_{width}), and avoidance angle, (γ_{avoid}). The parameter values used during the testing are summarized in Table 5.1. The formation normal, defines the forward direction used by the DNS algorithm when determining the orientation and linear speed of the robot. For the linear formation all robots were given the same value of 0^o. For the wedge formation tests two robots (pink, and purple) were given formation normals of -45^o, and the remaining two (green and yellow) assigned the value 45^o. The avoidance radius defines the size of visual region A. This parameter was set to 240 pixels, which corresponds to a distance which is approximately 1.5 times the diameter of the Bupimo's body. The angular width of visual region C influences the density of robots along one edge of the formation. For both formations tested this parameter was set to 15^o. Lastly the avoidance angle,

Parameter	Value
$\vec{F_i}$	$[0^o]$, $[-45^o, 45^o]$
ω_{width}	15^o
r_{avoid}	240 [pixel]
γ_{avoid}	100^o, 115^o, 135^o, 180^o

Table 5.1: DNS live trial parameters

γ_{avoid}, used by the avoidance behaviour was varied throughout the trials between 100^o and 180^o. This range of values was used since it encompasses a wide range of angles without being redundant. If an angle of 90^o or less is used the robots will spiral into each other. If an angle greater than 180^o is used the behaviour will be the same as using the complementary angle, but in the opposite direction.

5.3 Linear Formation



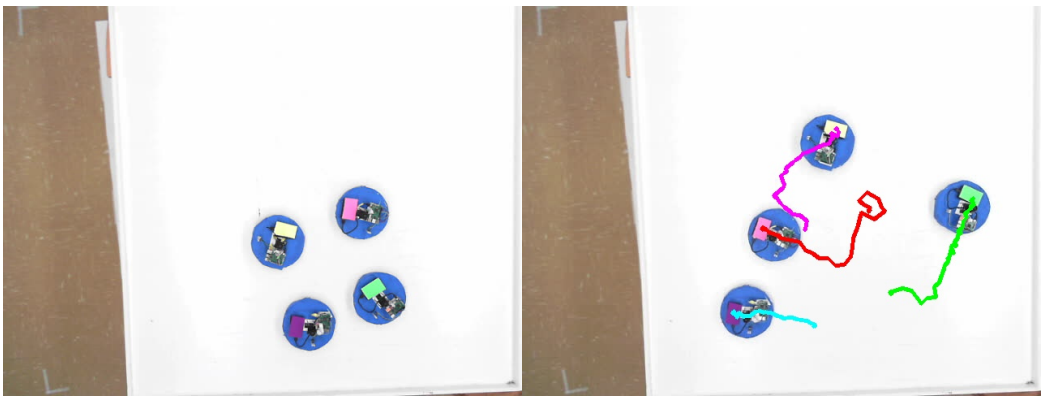
(a) Initial Positions. (b) Final Positions.

Figure 5.2: An example of the DNS algorithm forming a linear formation with 4 Bupimo robots.

The linear formation was evaluated first and the lessons learned during these tests were implemented in the wedge formation evaluations. Figure 5.2 illustrates a successful formation of a linear pattern. During these trials two major issues were encountered. The first issue encountered was the slow creeping forward of the line after the formation had been achieved. This was seen in some, but not all trials. Two possible sources of this issue are; misalignment of the omnidirectional camera resulting in a systematic rotation of the visual regions in the robots point of view, or random errors in the measurement of robots heading caused by noise in the digital compass. Regardless of the true cause of the issue, the creeping was observed to occur at very low speeds (less than 0.05 m/s). To fix this issue a minimum linear speed threshold of 0.075 was set. If a velocity command was given with a linear speed below this thresh-

old, the speed was reduce to 0. The second issue was divergence of the formation caused by magnetic anomalies in the floor of the laboratory. These anomalies were thought to be caused by structural steel beams in the floor. As a temporary solution to this problem the arena in which the tests were being conducted was raised off the floor by approximately 60 cm. This removed the affects of the magnetic anomalies, but is not a practical solution. A more practical solution to this issue is discussed in Section 6.1.

5.4 Wedge Definitions



(a) Initial Positions. (b) Final Positions.

Figure 5.3: An example of the DNS algorithm forming a wedge formation with 4 Bupimo robots.

Figure 5.3 illustrates a successful convergence of a wedge formation. However, during the evaluation of the wedge formation another issue with the DNS algorithm was encountered. Certain initial starting positions of the robots led to cyclic behaviours

in pairs of robots which resulted in divergence from the intended formation.

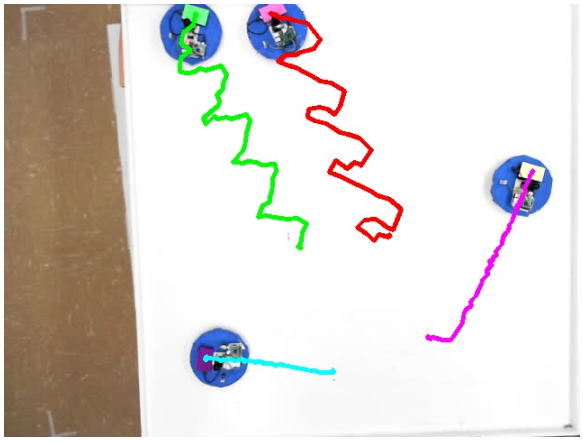
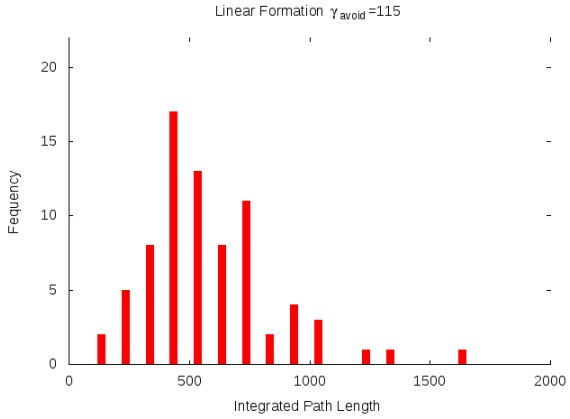


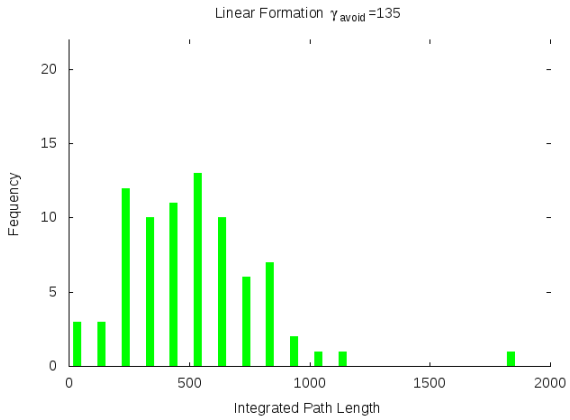
Figure 5.4: Example of the cyclic behaviour seen in wedge formations when using $\gamma_{avoid} = 180^o$. The red and green paths of the robots form a repeating pattern in which they enter and exit each others avoidance regions.

Figure 5.4 shows an example of these pathological cases. The issues occurred when two robots travelling at right angles enter each others avoidance regions. Once in each other's avoidance regions the robots switch to the avoidance behaviour and immediately travel directly away from each other. Once outside of the avoidance region, the robots turn back to their original heading, directly back at each other. This causes the cycle to repeat. A similar cyclic behaviour was seen while developing the DNS algorithm in simulation. It was thought that this type of behaviour was an unstable numerical effect that would be removed by the random noise of the real world. This however was not the case, although given enough time the robots might untangle themselves, the problem still decreases the efficiency of the DNS algorithm. To remove this issue a number of avoidance angles between 90^o and 180^o could be

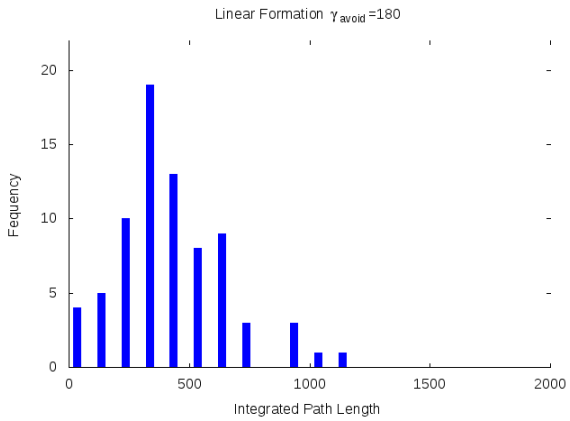
tested. However, both the extreme values (90^o and 180^o) lead to cyclic behaviours. An angle of 90^o would cause robots to endlessly circle each other and so it was not tested, and a value of 180^o produced the current problem. Values of 100^o, 115^o, and 135^o were tested. All three values removed the cyclic behaviour which had been observed previously. However, the lower values (100^o, and 115^o) often caused the robots to become physically entangled, and locked in an orbit around each other. This problem was more common with 100^o avoidance angle, than for the 115^o avoidance angle. An avoidance angle of 135^o was found to be useful at removing the cyclic behaviours while preventing the robots from becoming physically entangled. Linear formations using an avoidance angle of 135^o were also successful and had an efficiency comparable to using 180^o. Figure 5.5 compares the distances travelled by robots when using 115^o, 135^o, and 180^o avoidance angles. As the graphs show, the distances travelled are comparable in value.



(a) $\gamma_{avoid} = 115^o$



(b) $\gamma_{avoid} = 135^o$



(c) $\gamma_{avoid} = 180^o$

Figure 5.5: Distribution of integrated path lengths for robots executing the DNS algorithm with various γ_{avoid} values.

Chapter 6

Conclusions and Future Work

This chapter summarizes the results of the simulated and real world trials of the DNS algorithm, and highlights a number of future avenues of research.

6.1 Conclusions

The original goals of this thesis were twofold. First, was the development of a decentralized pattern formation algorithm which utilized limited sensory input, and anonymous members. The second was assisting in the development and testing of the Bupimo mobile robot, a low-cost vision enabled mobile robot for use in research and education. In addition to these contributions, the research presented in this thesis also lead to some insights into obstacle avoidance strategies for swarms of mobile robots.

The Dynamic Neighbour Selection (See Section 3) algorithm was developed and demonstrated in computer simulations and real world trials. Using limited sensory information the DNS algorithm was able to successfully drive a swarm of robots into

desired linear, wedge, and square formations. Specifically, the DNS algorithm functions without the need for robots to uniquely identify one another. The avoidance of unique identities increases the robustness, flexibility, and scalability of the algorithm. Furthermore, the DNS algorithm was compared to the SNS algorithm in computer simulation. These simulations demonstrated that the DNS algorithm was able to converged to the desired formation with a lower average distance traveller per robot than the SNS algorithm. However, the SNS algorithm was able to form a larger variety of formations.

The testing of the Bupimo mobile robot's functionality was carried out during the live robot trials of the DNS algorithm. During the trials both sensory and control capabilities of the robot were tested. Sensory and control signals were successfully communicated between the Raspberry Pi and the Zumo's microcontroller. This was demonstrated by the control software running on the Raspberry Pi being able to control the speed and orientation of the robot using feedback from the Zumo's motor encoders. The Bupimo was also able to identify fellow swarm mates using its omnidirectional camera, and coordinate these measurements with the on-board digital compass to give globally referenced bearings to neighbours. Additionally, two methods for distance estimation using images captured by the omnidirectional camera were evaluated. These methods used either blob size or distance from the origin as an estimate of distance. The latter measurement returned a more isotropic (the same in every direction) result and was used to successfully avoid collisions between robots during the trials.

Various obstacle avoidance behaviours were investigated throughout the development of the DNS algorithm, in both computer simulation, and during the live trials.

It was found that moving directly away ($\gamma_{avoid} = 180^o$) from other moving obstacles is not always the best strategy. This was seen in the wedge formation trails (Section 5.4). When two robots assigned to different edges meet they exhibit form a cyclic behaviour which causes the swarm to diverge from the desired formation. However, using a value for γ_{avoid} which is too low ($< 100^o$) results in other undesirable behaviours such as orbiting and robots becoming entangled in each other. An improved strategy is to move radially and tangentially at the same time (ie. $\gamma_{avoid} \in [115^o, 135^o]$).

6.2 Future Work

The work presented in this thesis could lead to a number of interesting avenues for future research. Improvements to the performance of the DNS algorithm and the overall functionality of the robot could be achieved by optimizing parameters. The DNS algorithm parameters which require further adjustment pertain to the definition of the sensing regions. Values for the radius of region A, and the angular width of region C were arrived at by trail and error. The optimal value of these parameters depends on the final dimensions of the Bupimo body. Since the final configuration of the Bupimo is yet to be decided, the optimization of these parameters will have to be postponed. Other robot parameters such as; the frequency of sensor reading, image resolution, communication speeds between ROS nodes and between the Raspberry Pi and Zumo's microcontroller where also arrived at using trail and error. The values chosen for these parameters were demonstrated to be adequate for the function of the robot, however a more rigorously chosen set of parameters could be more effective.

Finding a more practical solution to the problem of magnetic anomalies is an-

other interesting research direction. The interference of magnetic anomalies in the floor with the operation of the DNS algorithm was experienced during the live trials. One possible solution would be the use of a complementary filter. This filter could eliminating these effects by combining magnetometer measurements with readings from an on-board gyroscope. Changes in magnetic reading that are not associated with a rotation in the gyroscope would be ignored. This would allow the robot to distinguish between changes in bearing caused by magnetic anomalies or through rotation of the robot itself. The patterns of detected magnetic anomalies could be used as a simple map of the environment. This could lead to further research into the use of magnetic anomalies in the environment to solve the robot localization problem.

Another interesting avenue of research is the development of new control strategies for pattern formation through the use of evolutionary robotic techniques. This is similar to the research reported in [5, 21, 22, 24]. The current version of the DNS algorithm maps sixteen possible sensor states to five motor states. The sixteen sensor states are the result of the robot's field of view being divided into four binary sensing regions. The five motor states are represented by the behaviours which make up the DNS algorithm's behaviour-based controller. This means there are $5^{16} \approx 1.53x10^{11}$ possible DNS controllers. The specific instance of the DNS controller chosen in this research was arrived at by observing the behaviour of the robots in computer simulation, and further refined during live trials. Using a detailed computer simulation a large selection of possible controllers could be explored. Furthermore, the division of the camera's FoV into different numbers and sizes of sensing regions could be explored.

Bibliography

[1] Dario Floreano and Claudio Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[2] A. Sharkey. Swarm robotics and minimalism. *Connection Science*, 19(3):245–260, 2007.

[3] Alcherio Martinoli. Collective complexity out of individual simplicity. *Artificial Life*, 7(3):315–319, 2001.

[4] M. Brambilla et. al. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[5] Melvin Gauci, Jianing Chen, Wei Li, Tony J Dodd, and Roderich Groß. Self-organized aggregation without computation. *The International Journal of Robotics Research*, page 0278364914525244, 2014.

[6] R Arkin. *Behaviour-based robotics. Bradford Book*. MIT Press, 1998.

[7] A.N. Bishop, I. Shames, and Brian D.O. Anderson. Stabilization of rigid formations with direction-only constraints. In *Decision and Control and European*

Control Conference (CDC-ECC), 2011 50th IEEE Conference on, pages 746–752, Dec 2011.

[8] A.N. Bishop and M. Basiri. Bearing-only triangular formation control on the plane and the sphere. In *Control Automation (MED), 2010 18th Mediterranean Conference on*, pages 790–795, June 2010.

[9] A. Franchi and P. Giordano. Decentralized control of parallel rigid formations with direction constraints and bearing measurements. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5310–5317, Dec 2012.

[10] F. Morbidi, G.L. Mariottini, and D. Prattichizzo. Vision-based range estimation via immersion and invariance for robot formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 504–509, May 2008.

[11] E. Schoof, A. Chapman, and M. Mesbahi. Bearing-compass formation control: A human-swarm interaction perspective. In *American Control Conference (ACC), 2014*, pages 3881–3886, June 2014.

[12] N. et. al Moshtagh. Vision-based, distributed control laws for motion coordination of nonholonomic robots. *Robotics, IEEE Transactions on*, 25(4):851–860, Aug 2009.

[13] Eduardo Montijano, Dingjiang Zhou, Mac Schwager, and Carlos Sagues. Distributed formation control without a global reference frame. In *2014 American Control Conference*, pages 3862–3867. IEEE, 2014.

[14] Edward A Macdonald. Multi-robot assignment and formation control. *Georgia Institute of Technology*, 2011.

[15] Tucker Balch and Ronald C Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[16] James Hilder. The pi swarm: A low-cost platform for swarm robotics research and education. pages 151–162. Springer, 2014.

[17] C. Hart, E. J. Kreinar, D. Chrzanowski, K. A. Daltorio, and R. D. Quinn. A low-cost robot using omni-directional vision enables insect-like behaviors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5871–5878, May 2015.

[18] Nicholi Shiell and Andrew Vardy. A bearing-only pattern formation algorithm for swarm robotics. In *International Conference on Swarm Intelligence*, pages 3–14. Springer, 2016.

[19] A. Vardy and N. Shiell. Bupigo: An open and extensible platform for visually-guided swarm robots. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communication Technologies*, BICT'15, pages 315–319. AMC, 2016.

[20] Levi-Itzhak Bellaiche and Alfred Bruckstein. Continuous time gathering of agents with limited visibility and bearing-only sensing. *arXiv preprint arXiv:1510.09115*, 2015.

[21] Christoph Moeslinger, Thomas Schmickl, and Karl Crailsheim. A minimalist flocking algorithm for swarm robots. In *European Conference on Artificial Life*, pages 375–382. Springer, 2009.

[22] Matthew Johnson and Daniel Brown. Evolving and controlling perimeter, rendezvous, and foraging behaviors in a computation-free robot swarm. In *proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) on 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 311–314. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[23] Jason M O'Kane and Steven M LaValle. Localization with limited sensing. *IEEE Transactions on Robotics*, 23(4):704, 2007.

[24] Mark Beckerleg and Chan Zhang. Evolving individual and collective behaviours for the kilobot robot. In *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, pages 263–268. IEEE, 2016.

[25] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. Ros: an open-source robot operating system. In *ROS*, 2017.

[26] Mark A. Finlayson. *3-Axis Digital Compass IC HMC5883L Data Sheet*. Honeywell.