Department of Computing

# Distributed Motion Coordination for Mobile Wireless Sensor Networks using Vision

Justin Lee

This thesis is presented for the Degree of
Master of Science
of
Curtin University of Technology

March 2003

# Abstract

*Mobile wireless sensor networks (MWSNs)* will enable information systems to gather detailed information about the environment on an unprecedented scale. These self-organising, distributed networks of sensors, processors and actuators that are capable of movement have a broad range of potential applications, including military reconnaissance, surveillance, planetary exploration and geophysical mapping.

In many of the foreseen applications a certain geometric pattern will be required for the task. Hence, algorithms for *maintaining* the geometric pattern of an MWSN are investigated. In many tasks such as land mine detection, a group of nodes arranged in a line must provide continuous coverage between each end of the formation. Thus, we present algorithms for maintaining the geometric pattern of a group of nodes arranged in a line.

An MWSN may also need to *form* a geometric pattern without assistance from the user. In military reconnaissance, for example, the nodes will be dropped onto the battlefield from a plane and land at random positions. The nodes will be expected to arrange themselves into a predetermined formation in order to perform a specific task. Thus, we present algorithms for forming a circle and regular polygon from a given set of random positions.

The algorithms are distributed and use no communication between the nodes to minimise energy consumption. Unlike past studies of geometric problems where algorithms are either tested in simulations where each node has global knowledge of all the other nodes or implemented on a small number of robots, the robustness of our algorithms has been studied with simulations that model the sensor system in detail. The nodes locate their neighbours using simulated vision where a ray-tracer is used to generate images of a model of the scene that would be captured by each node's cameras. The simulations demonstrate that the algorithms are robust against random errors in the sensors and actuators. Even though the nodes had incomplete knowledge of the positions of other nodes due to occlusion, they were still able to perform the assigned

tasks.

# Acknowledgements

I would like to thank a number of people who provided valuable assistance with this Master's project.

I thank my supervisors Prof. Svetha Venkatesh and Dr. Mohan Kumar for their academic advice and patience. I appreciate the tremendous effort they put into revising this thesis.

I thank Prof. Geoff West for his technical advice in the development of the image processing algorithm.

I thank my mother for her encouragement and financial support.

Lastly, I thank a number of past and present research students for their help on a variety of matters: Russell Keith-Magee, Leon Blackwell, Stuart Campbell, Ezra Tassone, Sebastian Luhr and Alexander MacLennan.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Mobile Wireless Sensor Networks

Conventional sensor systems use one of two approaches to a sensing problem [38]. One approach is to use large complex sensors positioned far from the phenomena being observed. The targets are separated from the environmental noise with complex signal processing algorithms. The other approach is to use several sensors with a carefully engineered placement and communications topology. As the sensors have no processing ability of their own, time series of the sensed phenomena are transmitted to a central processing node which performs computations and data fusion. There is another approach, however, that is starting to receive more research attention:

- use a large number of spatially *distributed* nodes (sensor units equipped with processing and communications hardware) *embedded* within or in close proximity to the phenomena being observed [26]; and
- use *cooperation* among the nodes to *autonomously* perform the sensing task.

Such a network is called a *wireless sensor network (WSN)*. Why use a distributed network of sensors rather than a single sophisticated sensor? While there are the obvious advantages of fault-tolerance and flexibility, solid justification for this approach can also be found in detection and estimation theory [54]. Formally, a sensing problem is stated as follows: which hypothesis $h_i$ is true given the set of observables $\{X_j\}$? A hypothesis is chosen according to estimates of a set of parameters $\{f_k\}$, called the *feature set*. We choose $h_i$ such that $P[h_i|\{f_k\}] > P[h_j|\{f_k\}]$ for all $j \neq i$. The reliability of the decision is improved by increasing the number of independent observations or the signal to noise ratio (SNR). The uncertainty of the decision increases with the number

of hypotheses that have to be considered. Using an embedded, distributed network of sensors allows the user to monitor the phenomena at close range, thereby increasing the SNR. Furthermore, a dense distribution of sensors with a limited detection range is less likely to detect multiple targets using a single sensor. Thus there are less targets to be distinguished and therefore less hypotheses that must be considered.

There is a field within WSNs that is highly ambitious and has only just begun to emerge. It draws upon research from a broad range of areas in science and engineering, including cooperative mobile robotics [12], microelectromechanical systems (MEMS) [61] and even biology [68]. This area of research, called *mobile wireless sensor networks (MWSNs)*, presents many interesting research challenges.

MWSNs will enable information systems to gather detailed information about the environment on an unprecedented scale. They have a vast range of potential applications, including

**Defence:** They can carry out reconnaissance and search for land mines, preventing human casualties.

**Security:** They can provide *mobile* surveillance, unlike present security systems.

**Exploration:** They can gather sensor data of unexplored environments, including those on other planets.

**Mapping:** They can perform scientific mapping tasks such as geophysical mapping.

## 1.2 Objectives

A major challenge in the development of MWSNs is coordinating the motion of the nodes to achieve the desired objective. In the envisioned applications, this is a non-trivial problem for a number reasons:

- Nodes are prone to failure or damage, especially in military applications.
- The user might be unable to manually configure the network, because (a) the operating environment is difficult to access or dangerous, as in military applications; or (b) the network must be deployed on short notice.
- The sensors and actuators are subject to error.

- Wireless communication requires a lot of power while the nodes have a limited supply of energy, restricting the amount of information that can be shared between the nodes.

For these reasons, the motion coordination algorithms should be *robust*, *self-organising* and *distributed*. The aim of this thesis is to explore motion coordination algorithms that meet these requirements.

Where do we begin in our investigation of motion coordination algorithms for MWSNs? What are the fundamental motion coordination problems that will be encountered in typical MWSN applications? Firstly, we note that in many of the foreseen applications the given task must be carried out with the nodes arranged in a certain geometric pattern which is subjected to disturbances. In land mine detection, for example, an array of nodes arranged in a line must sweep an area while maintaining a certain separation between neighbouring nodes so that every point between each end of the array lies within the detection range of a sensor. Besides imperfections in the sensors and actuators, the MWSN must recover from the loss of nodes when explosions are triggered. Hence, the first problem that is investigated in this thesis is *maintaining* the geometric pattern of an MWSN.

Secondly, we note that in many applications where a certain geometric pattern is required, the MWSN must be deployed without assistance from the user. In military applications such as reconnaissance, for example, the nodes will be dropped from a plane behind enemy lines, landing at random positions. As explained by Balch and Arkin [10], the formation used has a major impact on performance in such tasks. Hence, the second problem that is investigated in this thesis is *forming* the geometric pattern of an MWSN.

## 1.3 Approach

The proposed algorithms were tested through simulation. The velocity of each node was set according to the algorithm being tested and the positions of the nodes were computed for each interval of simulation time. The nodes used vision to locate their neighbours. A ray-tracer was used to construct a simple model of each scene and generate images from the perspective of each camera on each of the nodes.

The simulations included random errors to test the robustness of the algorithms. Random errors were introduced into the camera images, orientation of the cameras

and velocity of the nodes. The impact of each type of error on the performance of an algorithm was tested by increasing the level of error over a series of simulations while the levels of the other types of error were set to zero.

## 1.4 Significance

Little research has been done on the problem of forming a geometric pattern with multiple mobile robots beginning from a random set of positions. Sugihara and Suzuki [66] and Défago and Konagaya [18] propose algorithms for forming a circle, but their algorithms require the robots to have global knowledge of all other robots. Using our algorithm this task can be performed with only partial knowledge of the other robots and this has been demonstrated with simulations.

The approach taken in this thesis enables us to gain insight into the issues that would arise in real implementations of motion coordination algorithms. There are challenges that must be considered in the design of motion coordination algorithms if vision is used to locate other nodes, such as

**Occlusion.** A node can obstruct the view of other nodes, preventing them from being located.

**Image noise.** The impact of image noise on a node's knowledge of the locations of other nodes must be taken into account. Image noise can increase the error in the computed location of a node or prevent nodes from being recognised, for example.

The following approaches are typically used to test algorithms in research on motion coordination of multi-robot systems:

- Simulation where the robots have perfectly accurate knowledge of the positions of other robots and perfect control of motion, as in [10], [13], [14], [17], [21], [27], [28], [33], [67], [69] and [71]. Sugihara and Suzuki [66] include sensor and control errors in their simulations, although it is assumed that each node has global knowledge of all other nodes.

- Implementation on a small number (typically two to four) of robots, as in [10], [17], [27], [33] and [67].

Our study is unique in that the impact of some of the errors in the sensors and actuators are investigated in simulations that model the sensor system in detail.

## 1.5 Outline of Thesis

A review of related work is given in Chapter 2. The review is divided into two parts: (i) research on WSNs; and (ii) cooperative mobile robotics. The survey of research on WSNs covers

- Major projects.
- Network architecture, including protocols designed specifically for WSNs.
- Node architecture.

The second part covers research on cooperative mobile robotics contributing to the development of MWSNs. In this part we discuss research on

- Existing algorithms for geometric problems. The algorithms developed by Sugihara and Suzuki are described in detail.
- Cooperative searching.

In Chapter 3 the problem of maintaining the geometric pattern of an MWSN is investigated. Three algorithms are presented:

- a *greedy* algorithm for maintaining the geometric pattern of a line formation in which neighbouring nodes are separated by a desired distance, Algorithm CS/G;
- a *state-based* algorithm for maintaining the geometric pattern of a line formation in which neighbouring nodes are separated by a desired distance, Algorithm CS/SB; and
- an algorithm for maintaining the geometric pattern of a line formation in which all neighbouring nodes are separated by the same distance and the length of the line remains fixed, Algorithm CL.

The vision algorithm that is used to calculate the relative positions of other nodes is described. The effect on performance of the setting of certain parameters for Algorithm CS/SB is investigated. The algorithms are simulated using a variety of starting configurations and error settings to test their robustness.

In Chapter 4 the problem of forming a geometric pattern with an MWSN is investigated. Four algorithms are presented:

- an algorithm for forming a circle centred at a landmark, Algorithm C/L;
- an algorithm for forming a regular polygon centred at a landmark, Algorithm P/$n$/L;
- an algorithm for forming a circle without a specified final location, Algorithm C; and
- an algorithm for forming a regular polygon without a specified final location, Algorithm P/$n$.

We demonstrate that Algorithm CS/G and Algorithm CL which were presented in the previous chapter can also be used to form a line from a random starting configuration. The robustness of the algorithms is tested using a similar approach to that in Chapter 3.

Our final conclusions for this thesis and suggestions for future work are discussed in Chapter 5.

# Chapter 2

# Review of Related Work

## 2.1 Introduction

The field of *mobile wireless sensor networks (MWSNs)* is relatively new, so literature relating specifically to this area is scarce. The following areas make a valuable contribution to research on MWSNs:

1. Wireless sensor networks
2. Cooperative mobile robotics

*Wireless sensor networks (WSNs)* provide the main foundation for research on MWSNs, as MWSNs are essentially an extension of WSNs. Research on WSNs has focussed mainly on using stationary nodes, so research on MWSNs builds on WSNs by exploring robust, efficient and adaptive methods of using node mobility.

*Cooperative mobile robotics* addresses the highly complex problems in motion coordination and cooperative behaviour which are introduced when the nodes have mobility. When a node becomes inoperative, the network has to intelligently maintain the spatial pattern of the nodes as well as the communications network. The nodes must also be able to cooperatively form a given geometric configuration from random positions without assistance from the user after deployment. Creating a mobile network with these capabilities poses an enormous challenge in artificial intelligence.

WSNs and cooperative mobile robotics are discussed next.

## 2.2 Wireless Sensor Networks

Franklin *et al.* [32] investigated the problem of land mine detection from a new perspective. According to Franklin *et al.*, past approaches to this problem focussed on novel sensor technology rather than the system as a whole. They argue that we should seek *system-level* solutions, where the functional requirements, sensor technologies, models of sensors, method of sensor application and platforms from which the sensors are applied are all considered at once. Furthermore, they state that current sensor technologies provide enough information to solve the detection problem.

The same view should be taken with any sensing problem. Much can be gained from exploring new strategies for combining a number of sensors to work cooperatively on a given task. Regardless of what sensor technology is used, the reliability of a sensing system can be improved by combining observations obtained from different perspectives through a process called *data fusion*. Research directed at finding novel sensor technologies only benefits the applications in which those technologies can be applied, whereas cooperative sensing strategies tend to be independent of the sensor technology employed and thus can be used in a wide range of sensing tasks. The advantages of *distributed* sensor networks are described by Agre and Clare [5], Estrin *et al.* [24] and Clare *et al.* [15]:

- Broad coverage of the environment. The area of coverage of a number of sensors can be combined to overcome limits in the coverage offered by single sensor systems.
- Fault tolerance through redundancy and distribution of resources.
- The area of coverage can be shaped according to the application's geography, overcoming obstructions in the line of sight of sensors.
- Distributing coverage with multiple sensors allows phenomena to be monitored in close proximity, improving the SNR (signal to noise ratio).
- The ability to combine data from a variety of spatial perspectives and types of sensors, thereby increasing the SNR.
- Routing of radio signals around obstructions that would otherwise block transmission.
- Scalability through distributed processing.

For these reasons, we explore the potential of WSNs. A WSN is a distributed, wireless network connecting sensors, actuators and processors.

WSNs will essentially act as a robust, adaptive and flexible interface between information systems and the physical world. As discussed by Akyildiz *et al.* [7], WSNs will have a broad range of applications in many areas including

**Defence:** Their rapid deployment, self-organisation and fault tolerance make them ideal for defence applications. They can play an integral role in military C4ISRT (command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting).

**Environmental monitoring:** WSNs can be used to monitor ecosystems and the composition of the atmosphere, water and soil to measure the levels of pollutants. They can also be used to detect dangers such as forest fires and floods.

**Health:** Physiological data from patients can be collected with WSNs. They can be used to locate patients and doctors within a hospital.

**Enterprise:** WSNs can be used for factory automation, process control and tracking inventory.

### 2.2.1 Current Research Projects

Akyildiz *et al.* [7] provide a comprehensive and recent survey of research in this emerging field. Other reviews have been written by Rentala *et al.* [60] and Qi *et al.* [58], although [58] only covers network structure, data processing paradigms, sensor fusion algorithms and sensor deployment.

Current research projects can be divided into two categories: *design*, where protocols and algorithms are proposed and tested through simulation, on hardware testbeds or devices developed elsewhere; and *implementation*, where working devices are constructed. With reference to the network protocol stack, the design projects concentrate on the higher layers and the implementation projects concentrate on the lower layers. The major projects are described below.

#### Design

**SCADDS (Scalable Coordination Architectures for Deeply Distributed Systems):** The aim of the SCADDS project [1, 11, 23–25, 38, 75] is to develop *scalable* architectures for deeply distributed and dynamic systems. A fundamental principle of

this project is that global objectives should be achieved with *localised* algorithms, where processing is distributed among a number of nodes and communication is limited to the immediate neighbourhood of the node. Also, routing should be *data-centric*, meaning that the flow of data is directed by naming the *attributes* of the desired information rather than the source. A notable contribution of this project is *directed diffusion* [38], a data dissemination paradigm.

**SPIN (Sensor Protocols for Information via Negotiation):** A family of adaptive protocols for disseminating data in a WSN with a limited energy supply [36, 40]. The SPIN family is designed to overcome the shortcomings of classic flooding protocols, namely *data implosion*, *data overlap* and *resource blindness*. The SPIN family uses data-centric routing, like directed diffusion.

**LEACH (Low Energy Adaptive Clustering Hierarchy):** LEACH [35] is a self-organising, adaptive cluster formation protocol designed to prolong the life of a WSN by randomly rotating the cluster heads which consume more energy than nodes lower in the network hierarchy.

**SINA (Sensor Information Networking Architecture):** Middleware providing an abstract interface to a WSN where the sensor nodes are modelled as distributed objects [62].

**$\mu$AMPS (Micro-adaptive Multi-domain Power-aware Sensors):** The $\mu$AMPS wireless sensor node [63] was developed to demonstrate that energy consumption should be minimised by using a physical layer-driven approach to the design of protocols and algorithms. The authors state that in order to meet the system lifetime goals of wireless sensor networks, considering the parameters of the underlying hardware is critical.

Directed diffusion, SPIN, LEACH and SINA are discussed in Section 2.2.3.

#### Implementation

**WINS (Wireless Integrated Network Sensors):** At the University of California and the Rockwell Science Center some of the earliest work on WSNs was carried out developing the *wireless integrated network sensor* [2, 6, 9, 15, 52–55]. A WINS consists of a densely distributed network of sensors, controls and processors that the user can access via a conventional network such as the Internet. An

architecture for achieving self-organisation with WINS is described in [15]. A detailed description of a prototype sensor node that will serve as a development platform for future research on WINS is given in [6]. No recently published literature on the WINS project has been found.

**Smart Dust:** The most ambitious of the projects surveyed, the researchers of Smart Dust [39, 50, 70] are endeavouring to create minute nodes—termed "motes"—the size of a cubic millimetre. An innovative feature of Smart Dust is the use of optical communication, which has a number of advantages over RF communication. Optical communication consumes less energy, uses less complex circuitry and the transmitter and receiver occupy less space. Also, passive transmission can be used, where light is provided by an external source and is reflected with a device known as a *corner-cube retroreflector*, or CCR. A CCR consists of three mutually orthogonal mirrors that reflect light back in a direction parallel to the incident ray.

**PicoRadio:** The objective of the PicoRadio project [3, 59] is to develop ultra-low power, low-cost, small, lightweight sensor nodes called *PicoNodes*. The PicoNode is expected to have the following characteristics:

- Power dissipation less than 100 $\mu$W.
- Cost substantially less than \$1.
- Size less than 1 $cm^3$.
- Weight less than 100 g.

Energy scavenging is given fair consideration, as the ultimate goal of the project is to create self-powered PicoNodes. Reasonable attention is given to all layers of the protocol stack in the design of the PicoNode, and algorithms and protocols that minimise energy consumption have been developed [76, 77].

### 2.2.2 Design Challenges

Research into WSNs is greatly motivated by their applications in defence, so they have greater requirements of robustness and autonomy than systems with civilian applications. The foreseen applications of WSNs present the following design challenges [5, 7, 24]:

**Fault tolerance:** Fault tolerance will be essential in applications where nodes are likely to become inoperative, as in battlefield surveillance, or reliability is critical, as in flight navigation. The network should autonomously adapt to node failures. The quality of the information gathered should degrade gracefully as nodes become inoperative.

**Self-organisation:** A key design goal is to minimise the time and labour involved in the deployment of WSNs. In applications such as battlefield surveillance, where they are likely to be dropped from a plane and land at random locations, the ability to self-organise will be essential. The nodes should be able to rapidly establish a communications network after deployment without assistance from the user. Human installation could be dangerous in the intended military and law enforcement applications.

**Power consumption:** The constraint of power consumption has received the most attention in literature. The energy available to each node will be quite limited. The physical size of the nodes will limit the size of the batteries and devices used to scavenge energy from the environment.

**Scalability:** WSN deployments are expected to number in the thousands, possibly millions. The design should allow the size and density of the coverage to be increased as required by the application.

**Production cost:** Since WSNs will be deployed in large numbers, the cost of producing a single node must be minimal to justify their use. The overall cost of the network should be less than that if conventional sensors are used.

**Operating environment:** They will operate within harsh physical and radio environments. Since the nodes may have to operate inside a tornado or at the bottom of an ocean, for example, sturdy casing will be needed in many applications. On the battlefield the WSN could be subject to electronic jamming.

Correal discusses the design challenges from a layered architectural perspective in [16]. An overview of the issues associated with device technology, software, the physical layer, medium access control, networking, the transport layer and applications is given in this paper.

### 2.2.3 Network Architecture

The network goes through a bootup phase when it is first deployed [15]. During this phase, the network establishes the communications infrastructure without assistance from the user. Long-term circuits are established which reduce the transport latency of messages during normal operation.

The protocols should be designed so that uniqueness is determined at network bootup [53]. The nodes should not need an embedded serial number to resolve conflicts. It would be preferable to use a node's location as its address, for example.

Time division multiple access (TDMA) is suitable for communication during normal operation [53]. The use of TDMA as a basic communications framework has a number of advantages over random medium access control methods:

- Savings in power. During time slots not assigned to them, nodes can turn their transceivers off, saving power. Also, for a time slot that a transceiver is scheduled to receive a signal from a neighbouring node, the transceiver can be left on just long enough to determine if that node is going to use the slot to make a transmission. If a signal is received during a time slot, the node can switch off the transceiver for the remainder of the time slot once the end of the message has been detected.

- Network time synchronisation, which is needed at the application level for processing phenomena.

- Deterministic message transport latency.

The frequency band of each communications link is chosen at random. The short range of transmissions in WSNs enables the spatial reuse of frequencies. Thus there is a low probability that signals for different links sent during the same time slot will be in conflict with each other.

Because of signal attenuation, it is more efficient in terms of power to relay messages over a series of nodes [55]. Also, multi-hop communications allows messages to be routed around wavefront obstructions in the environment. Finally, multi-hop routing minimises the probability of message interception [53].

Hierarchical, distributed information processing will help conserve energy and achieve scalability, as discussed in [51]. Network traffic should be minimised by performing as much processing as possible at the node where the data originated. A node will first

process the gathered raw data locally before communicating with neighbours to come to a decision. These initial decisions may have relatively high false alarm probabilities. Then, if necessary, more information gathered from a number of nodes may be aggregated at a nearby processing site to perform data fusion. Such decisions involving collaborative processing would have lower false alarm probabilities. The decision and its probability are summarised by a few bits and carried to the user through a chain of nodes. As the message is passed through the chain, nodes can aggregate information into summary reports to reduce the size of the transmission. The user can request the original data if desired to make a more reliable decision or view extra detail about an event.

CDMA (code division multiple access) can be used to reduce the management of conflicting transmissions [53]. Due to the spatial reuse of frequencies, bandwidth is relatively abundant, making CDMA suitable. CDMA is also useful for security reasons. CDMA provides resistance to jamming and reduces the probability of detection in covert applications.

Physical layer wake-up techniques offer a couple of benefits in WSNs [53]. A node can "wake up" another to receive a message by sending a preamble, which is detected in a low-power reception mode. They also enable the recovery of network synchronisation.

The architecture of a WSN will be discussed within the framework of the layered model used by Akyildiz *et al.* [7], shown in Figure 2.1. The layers of the model are

**Application layer:** Set of high-level protocols that provide services to user software.

**Transport layer:** Provides end-to-end communication services to the application layer.

**Network layer:** Responsible for routing data supplied by the transport layer.

**Data link layer:** Responsible for the reliable transfer of data across a physical communications link.

**Physical layer:** Responsible for the transmission of the signal across the physical medium.

Additionally, there is the *power management plane*, *mobility management plane* and *task management plane* which span all the layers above, as illustrated. These planes handle the conservation of power, networking of mobile nodes and the distribution of tasks among the nodes.

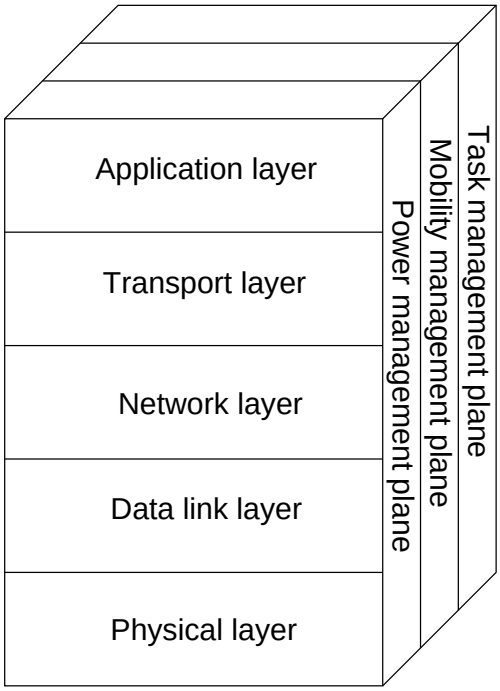


Figure 2.1: WSN architecture reference model.

### Application Layer

Despite the vast range of potential applications for WSNs, there has been little research at the application layer of the protocol stack [7]. A notable development in this area is SINA [62], introduced earlier, which has a programmable interface.

The SINA paradigm features

- hierarchical clustering, enabling scalability;
- attribute-based naming, where queries identify the *attributes* of the requested data rather than specific nodes; and
- location awareness.

These design choices reflect general trends in the development of WSNs [24, 25, 35, 36, 38, 40, 51].

Communication and configuration primitives are provided by the *sensor execution environment* (SEE). Three data gathering operations are implemented in the SEE:

**Sampling Operation:** Nodes choose whether to respond to a query based on a probability $p$.

**Self-Orchestrated Operation:** Nodes delay their response by a randomly chosen length of time. The delay is calculated from the formula $KH(h^2 - (2h-1)r)$, where $h$ is the number of hops from the sink, $r$ is a random number such that $0 < r \leq 1$, $H$ is a constant set to the estimated delay per hop and $K$ is a constant chosen to compensate for processing and queueing delays.

**Diffused Computation Operation:** Nodes only communicate with their immediate neighbours and data is aggregated as it is delivered to the sink.

These operations are designed to prevent the *response implosion problem*, a collision resulting from a large number of responses.

Within the abstract representation of the network provided by SINA, each node is conceived as a *datasheet*, and the datasheets contain *cells* which hold the attributes of the associated node. The datasheets of a network together form an *associative spreadsheet*, where the cells are referenced with attribute-based names.

The programming interface lies in a separate layer above the SEE. Applications are written in a procedural scripting language called SQTL (Sensor Query and Tasking Language). SQTL handles events using the *upon* construct, which supports three keywords defining the type of event:

**Receive:** Reception of a message.

**Every:** Periodic time-out of a timer.

**Expire:** Expiration of a timer.

### Transport Layer

Transport layer protocols will have to be developed to meet the unique requirements of WSNs [7]. The special factors that must be considered are:

- The use of attribute-based naming, unlike the methods of global addressing used in protocols such as TCP.
- Limited power.
- Limited memory.

Akyildiz *et al.* [7] also state that a scheme similar to TCP splitting may be required. Such a scheme involves terminating connection-oriented streams from the Internet or a

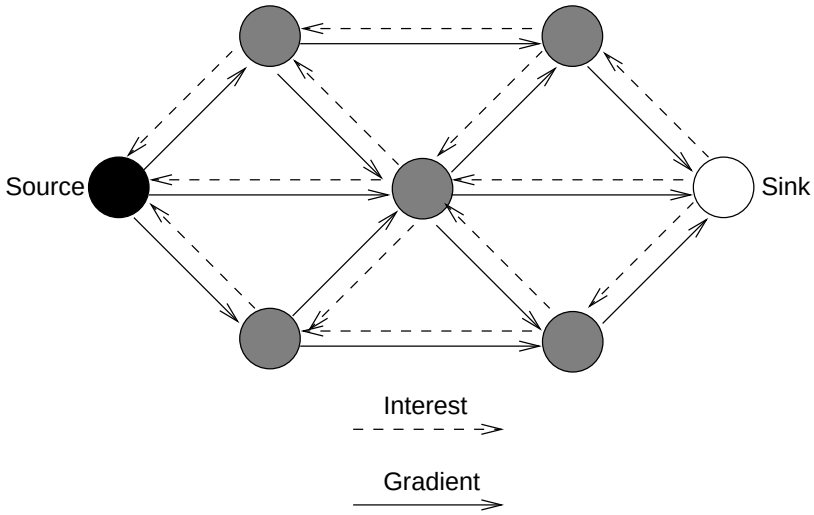


Figure 2.2: Interests and gradients.

satellite at the sink and converting them to a connectionless protocol like UDP inside the WSN.

### Network Layer

**Directed Diffusion** Directed diffusion is a data-centric, localised data dissemination paradigm [38]. To make a query, the *attributes* of the requested data, rather than the source, are named. A query may be issued in the form of attribute-value pairs, such as (type = four-legged animal, interval = 20 ms, duration = 10 s, rect = [-100, 200, 200, 400]) to check for four-legged animals every 20 ms for a period of 10 s within the rectangle bounded by the lines $x = -100$, $y = 200$, $x = 200$ and $y = 400$. Such descriptions of requested data are called *interests* and are distributed through the network using multi-hop communication, as shown in Figure 2.2. Each node stores the received interests in its *interest cache*. Associated with each interest are *gradients*, which indicate the neighbour from which the interest was received and the *value* of the interest. The *value* could represent the requested data rate, for example, but the meaning is application-specific.

When a node has data, like (type = four-legged animal, instance = elephant, location = [125, 220], intensity = 0.6, confidence = 0.85, timestamp = 01:20:40), for example, the interest cache is checked for a matching interest. If a matching interest entry exists, the data is sent to the nodes specified by the gradients within the interest entry, otherwise it is silently dropped. The data may be locally cached, or operations

that transform the data, such as aggregation, may be performed.

**SPIN (Sensor Protocols for Information via Negotiation)** The SPIN family [36, 40] is designed to overcome the following problems found in *classic flooding*:

**Implosion:** A node receives the same data from a number of neighbours.

**Overlap:** A node receives transmissions with a common component from sensor nodes with overlapping coverage.

**Resource blindness:** The nodes do not adapt according to the availability of energy.

To address these problems, the SPIN family features *negotiation* and *resource-adaptation*. *Negotiation* is accomplished with data that *describe* information, called *meta-data*. Each node handles *resource-adaptation* with a *resource manager*.

The SPIN family consists of four protocols:

**SPIN-PP:** Protocol designed for networks that use *point-to-point* transmission media.

**SPIN-EC:** A version of SPIN-PP designed for *energy conservation*.

**SPIN-BC:** Protocol designed for networks that use *broadcast* transmission media.

**SPIN-RL:** A *reliable* version of SPIN-BC designed for lossy transmission media.

The SPIN-PP protocol is as follows:

1. When a node has data to send, it transmits an ADV message which contains a description of the data this node is making available to neighbouring nodes.
2. Neighbouring nodes that do not have part or all of the data described in the ADV message reply with an REQ message, which contains a description of what data was advertised that the node does not have.
3. The advertising node sends the requested data to each neighbouring node in a DATA message.
4. The neighbouring nodes that received new data repeat this procedure.

SPIN-EC is the same as SPIN-PP, except a node will not communicate if there is not sufficient energy to complete the three-stage handshake.

SPIN-BC is a three-stage handshake protocol with an ADV, REQ and DATA message like SPIN-PP, only with a couple of differences. First, it is assumed that nodes communicate via a broadcast medium where a transmission will be received by all neighbouring nodes within a certain range. Secondly, neighbouring nodes do not reply immediately to ADV messages. When an ADV message is received and the node does not have part or all of the data advertised, it waits for a randomly chosen length of time. During this period the node listens for REQ messages sent by other nodes in response to the ADV message. If the node hears an REQ message with meta-data that includes all the data it is about to request, it cancels the transmission of its own REQ message, otherwise a request is sent at the end of the waiting period. The node that advertised the data is specified in the header of the REQ message. Thus, only one REQ message is sent for any of the data transmitted by the advertising node.

The SPIN-RL protocol is SPIN-BC with a couple of modifications. Nodes record the ADV messages sent by neighbouring nodes along with the senders of these messages, and if data for a request is not received within a certain period of time, the node re-requests the data. Secondly, the frequency at which data can be resent is limited. A node will wait for a certain period of time after a DATA message is sent before replying to any requests for that data to be resent.

**Sequential Assignment Routing (SAR)** SAR [64] is a flat routing protocol that provides protection from failures by constructing multiple paths from sensor nodes to the sink. Communication trees are constructed which begin from a root that is one hop from the sink and grow outward. The trees are formed by creating branches to nodes which are a greater number of hops from the sink while avoiding those with low energy or quality of service. The final result is that most nodes belong to more than one tree and thus have a number of routes to the sink to choose from.

For a given node, two parameters are associated with each path back to the sink:

- Energy resources measured by the number of packets that can be delivered through the path without energy depletion, assuming the source node has exclusive use of the path.
- Additive quality of service metric where a higher value means lower quality of service.

The node selects a path taking into account the parameters above and the priority of the packet. A weighted quality of service metric is calculated as the product of a weight

coefficient based on the priority of the packet and the additive quality of service metric. The goal of the SAR algorithm is to minimise the average weighted quality of service metric over the lifetime of the network.

**LEACH (Low Energy Adaptive Clustering Hierarchy)** The purpose of the LEACH protocol [35] is to establish a hierarchical network topology where the *cluster-heads*, the nodes responsible for collecting data from nodes within their cluster and transmitting it to a base station, are periodically rotated to prevent them from expending all their energy before other nodes.

LEACH is organised into *rounds*, which consist of a *set-up phase*, where the clusters are formed, followed by the *steady-state phase*, where data transmission takes place. To reduce the overhead associated with the formation of clusters, the steady-state phase is longer than the set-up phase.

The set-up phase begins with nodes independently electing themselves at random. Each node generates a random number between zero and one and compares it to a threshold given by

$$T(n) = \begin{cases} \frac{P}{1-P[r \bmod \frac{1}{P}]} & \text{if } n \in G \\ 0 & \text{otherwise} \end{cases}$$

where $P$ is the desired fraction of nodes to become cluster heads, $r$ is the round number, $n$ is the node number and $G$ is the set of nodes that have not elected themselves as cluster-heads in the last $\frac{1}{P}$ rounds. If the random number is less than $T(n)$, then the node becomes a cluster head. Each cluster-head then broadcasts an advertisement informing the other nodes that they have chosen to be a cluster-head for the current round. The other nodes choose the cluster to join from the received signal strength of the advertisements broadcasted by the cluster-heads and transmit a message back to the chosen cluster-head to inform it that the node has joined the cluster. Having been informed of all the nodes in their cluster, the cluster-heads set up a TDMA schedule and inform each member of the time slot on which it can transmit data.

The network then enters the steady-state phase and the nodes start sending sensor data to the cluster-heads. The cluster-heads aggregate the data received and transmit it to the base station.

**Cooperative Signal Processing** If a number of nodes detect the same event, it would be wasteful of power for all of them to send this information back to the user.

Pottie [53] describes a protocol to determine which node should inform the user of an event. When an event is detected, nodes wait an amount of time proportional to the SNR before broadcasting a message to inhibit other nodes from communicating. Thus the node with the most reliable information will be the first to broadcast the inhibition message, and it is this node that will convey information about the event to the user.

When a number of nodes need to collaborate in processing an event because the SNR of each individual node is not high enough to make a reliable decision, the following protocol described in [53] can be used to limit the number of nodes involved in the collaborative decision. If a node does not have an SNR above a certain threshold, it broadcasts an invitation to neighbouring nodes requesting more data. The nodes will wait an amount of time proportional to their SNR before responding. The node with the highest SNR will be the first to respond and share its data. If after their data is fused the SNR is above the required threshold, the first node will broadcast a message to inhibit further communication, otherwise this process is repeated until the SNR is acceptably high or no further responses are received indicating that the desired SNR could not be achieved.

There are two types of cooperative signal processing [64]:

**Noncoherent:** Sensor data is preprocessed at the node producing a small set of parameters, which are forwarded to a central node (CN) for further processing. Energy efficiency is achieved through *algorithmic efficiency*.

**Coherent:** Sensor data is timestamped after minimal preprocessing and then forwarded to a CN for more intensive processing. Energy efficiency is achieved through *path optimality*.

The SWE (Single Winner Election) algorithm forms a network for noncoherent processing and the MWE (Multi-Winner Election) algorithm forms a network for coherent processing [64].

The SWE algorithm elects a CN while a minimum-hop spanning tree is formed. It has three phases:

**Phase I:** The target is detected and data is collected and preprocessed. Nodes then decide whether to participate in the cooperative processing. The results from preprocessing, such as the SNR, could be used to make this decision.

**Phase II:** Nodes participating in the cooperative processing announce their involvement to neighbouring nodes.

**Phase III:** The CN is elected using criteria such as energy reserves, computational ability and SNR. Nodes announce themselves as potential CNs by broadcasting *Elect* messages which contain information about the node, including the values of the election criteria and routing information. Based on the election criteria, the nodes then compare themselves with the nodes identified in the Elect messages received. Information about the winner of the comparison is recorded in the node's registry and forwarded to neighbouring nodes in a further exchange of Elect messages. The election procedure is repeated until a minimum-hop spanning tree is formed with the overall winner as the root. By including a variable delay that favours nodes more likely to win before broadcasting Elect messages, the number of messages exchanged can be reduced.

The MWE algorithm performs two tasks:

1. Election of $n$ source nodes (SNs) that will generate sensor data. In coherent processing the number of SNs is limited because they must transfer larger volumes of data to the CN.

2. Election of a CN based on the *total energy* required to transfer data from the SNs to the CN.

The MWE algorithm is essentially an extension of the SWE algorithm. As before Elect messages are exchanged to compare candidate nodes, except each node records the $n$ best candidates rather than the single best candidate. The minimum-energy path to each SN is determined by piggybacking the link energy costs on Elect messages. After the $n$ SNs are elected the total energy required to send data to each node from the elected SNs can be computed. The CN is then elected using the SWE algorithm with the CN elected according to the total energy cost of uploading data from the SNs.

### Data Link Layer

**S-MAC (Sensor-MAC)** The authors of S-MAC [75] identify the following inefficiencies in energy consumption found in conventional wireless MAC protocols such as IEEE 802.11:

**Idle listening:** The receiver is left on while the channel is idle. Many measurements have found that idle listening consumes 50-100% of the total energy consumed by reception.

**Collisions:** Packets that are corrupted by other transmissions made at the same time must be discarded and retransmitted.

**Overhearing:** Energy is wasted if the receiver is left on to listen to transmissions sent to other nodes.

**Control packets:** Energy is consumed in transmissions that do not contain any user data.

S-MAC is a self-organising, scalable MAC protocol for WSNs designed to reduce the energy wasted by the factors above. The protocol is based on a scheme that uses a combination of contention and scheduling.

To reduce idle listening, nodes periodically sleep. During sleep mode the receiver is switched off. The listen and sleep schedule is broadcasted to neighbouring nodes to synchronise schedules, when possible, and inform them when data can be received. Nodes wait until the receiver is listening before transmitting. Synchronised neighbouring nodes are called *virtual clusters*, however they are quite different to the clusters formed with TDMA-based schemes such as LEACH for the following reasons:

- Neighbouring nodes must *contend* for the medium, whereas with TDMA-based schemes each node within the same cluster is assigned a separate time slot for transmission.
- S-MAC forms a *flat* topology, whereas TDMA-based schemes form a *hierarchical* topology.
- TDMA-based schemes use FDMA (frequency division multiple access) or CDMA (code division multiple access) to avoid interference between neighbouring clusters.

Collisions and overhearing are avoided using a procedure similar to that of IEEE 802.11. Neighbouring nodes contend for the medium using RTS (request to send) and CTS (clear to send) packets:

1. The listen period is divided into two parts. The first is for receiving SYNC (synchronise) packets which are used to synchronise schedules, and the second is for receiving requests to send data. Each part is divided into a number of time slots and one is selected at random. Transmission will begin at the selected time slot.

2. *Virtual* and physical carrier sense are performed before transmission during the listen period. *Virtual carrier sense* prevents collisions and overhearing by indicating the length of each transmission with a duration field. If a transmission intended for another node is received, the node sleeps until this transmission has finished. Physical carrier sense is performed by monitoring the channel for transmissions.

3. Transmission begins by first sending an RTS packet. The receiver informs the sender that it is ready to receive data by sending a CTS packet. Data is transmitted in DATA packets, and the receiver responds to each one with an ACK (acknowledge) packet. If the sender fails to receive an ACK packet, the current packet is retransmitted. The ACK packet not only ensures reliability, but prevents the *hidden terminal problem.* The ACK packet informs neighbours of the receiver that the medium is presently being used. If a node is within transmission range of the receiver but not the sender, it will assume that the medium is clear for transmission to the receiver unless ACK packets are broadcasted.

*Message passing* reduces control overhead and achieves fair application-level latency. A *message* is defined as a collection of meaningful, interrelated units of data. A message is divided into fragments and transmitted in a burst. The medium is reserved for the entire message by setting the duration field of each packet to the time needed to transmit the remaining packets of the message. The RTS and CTS packets are only sent once in the transmission of a message, but an ACK packet is sent after each data fragment so that only the corrupted or lost packet has to be be retransmitted when an error occurs.

**SMACS (Self-Organising Medium Access Control for Sensor Networks)** SMACS [64, 65] is a decentralised, scalable, self-organising protocol for setting up the communications links of a WSN. It is based on the assumption that while the nodes have a limited supply of energy, the available bandwidth exceeds network requirements. A channel is assigned as soon as a neighbour is discovered, thus neighbour discovery and channel assignment are performed simultaneously. This enables the communications links to be established concurrently throughout the network.

The protocol is based on a modified form of TDMA. Nodes communicate using a repeating schedule called a *superframe*, which has a fixed duration $T_{\text{frame}}$. The superframe is divided into a BOOTUP period and a TDMA period. Link formation is performed during the BOOTUP period with a four way handshake, and the TDMA

period is divided into time slots for transmitting and receiving data. The incidence of collisions between adjacent links is reduced by selecting the frequency at random from a large pool. A *flat* topology is formed, as opposed to clusters organised in a hierarchy.

The following procedure is used during the BOOTUP period for establishing links with neighbouring nodes. After deployment, the nodes wake at random times and listen on a fixed frequency for invitations to connect to other nodes. These invitations are called TYPE1 messages. The node that transmits the TYPE1 message is called the *inviter*, and the node that receives it is called the *invitee*. The nodes listen for TYPE1 messages for a random length of time, and if none are received they begin broadcasting their own TYPE1 messages. The invitee responds to the TYPE1 message with a TYPE2 message which includes the number of connected neighbours. The invitee waits for a random length of time before transmitting the TYPE2 message to reduce the likelihood of collisions with responses from other invitees responding to the same invitation. The inviter then chooses the invitee that it will establish a link with based on a criterion such as the arrival time of the response, received signal level or number of neighbours already connected. The inviter broadcasts a TYPE3 message to announce which invitee has been chosen. The chosen invitee then responds with a TYPE4 message. Finally, the inviter and invitee exchange a pair of test messages and update their schedules to include the new link.

The information included in the body of the TYPE3 message and TYPE4 message depends on whether the inviter and invitee have already established links with other neighbours. If the inviter has links with other neighbours but not the invitee, the time slots for communication between the inviter and invitee during the TDMA period are chosen by the inviter; the chosen time slots are transmitted in the body of the TYPE3 message. Otherwise, the time slots are chosen by the invitee and are transmitted in the body of the TYPE4 message. If both the inviter and invitee already have links with other neighbours, the inviter will transmit its schedule and the time at which the next superframe will begin in the body of the TYPE3 message. In this case the time slots are chosen according to the schedule and time offsets of both the inviter and invitee.

Two nodes form a *sub-net*, which is defined as a subset of nodes that form a connected graph and have coinciding superframe epochs. The sub-nets grow as unconnected neighbours are added. When two sub-nets attempt to connect, they need not align their superframes; unassigned time slots from the different schedules that overlap are used, otherwise the nodes search for other neighbours to connect to.

**EAR (Eavesdrop and Register)** EAR [64] is an MAC protocol for connecting mobile nodes in a WSN that consists mainly of stationary nodes. It serves as an extension of the MAC protocol handling connections between the stationary nodes. Since relatively few stationary nodes need to connect to a mobile node at any given time, the mobile nodes are given most of the responsibility for managing their connections. Energy is conserved by minimising the specialised control signals transmitted by the stationary node for establishing and terminating connections with mobile nodes. Energy would be wasted if the stationary nodes broadcast invitations specifically for mobile nodes because few of them would be received. Hence, the mobile nodes "eavesdrop" on the control signals of the stationary MAC protocol.

The EAR protocol is as follows. First, the mobile nodes listen for BI (broadcast invite) messages broadcasted by the stationary nodes. The BI messages are invitations for both stationary and mobile nodes to make a connection with the stationary node; they are part of the stationary MAC protocol and reused by the EAR protocol. Information such as the received SNR, node ID and transmitted power is extracted from the signal. The mobile node then decides whether to request a connection from the stationary node according to the potential connection quality and the mobile node's connection status. If the mobile node decides to request a connection, an MI (mobile invite) message is transmitted to the stationary node. The stationary node then determines whether a connection is possible, which depends on the TDMA schedules of the mobile node and stationary node. If the stationary node accepts the request, it responds with an MR (mobile response) message; otherwise a decline is sent. The mobile node terminates the connection when the SNR falls below a predetermined threshold by transmitting an MD (mobile disconnect) message. There is no acknowledgement of the MD message. Note that the MR message is the only message transmitted by the stationary node that is specific to the EAR protocol.

Each mobile node maintains a registry of stationary nodes within its neighbourhood. It holds information for forming, maintaining and terminating connections. Information about every stationary node that is discovered is stored in the registry until it becomes full. Once the registry is full, information about stationary nodes is entered through contention based on a simple criterion such as channel quality.

Each stationary node also maintains a registry. Mobile nodes are added and removed as they establish and terminate connections.

#### Physical Layer

The nodes can communicate via radio, infrared waves, light, acoustic waves or wires [55], although radio is the only medium that is considered in just about all the literature surveyed. Smart Dust, which was discussed in Section 2.2.1, is one project where an alternative transmission medium is used. Smart Dust uses optical communication, which has the advantage that energy for transmission can be provided by an external source through reflection.

Generally, the minimum power required to transmit a radio signal over a distance $d$ is proportional to $d^n$, where $n$ can range from two to four [7]. The value of $n$ is closer to four for low-lying antennas due to partial cancellation by ground-reflected rays. The sharp decay of the signal with distance can be exploited by the designer [54]. "Shadowing", that is wavefront obstruction and confinement, can be overcome with multi-hop communication by routing the signal around obstacles. Also, frequencies can be spatially reused.

The ISM (industrial, scientific and medical) bands are available for transmission without licence [7]. These bands are specified in the International Table of Frequency Allocations contained in Article S5 of the Radio Regulations (Volume 1).

Shih *et al.* [63] investigate the impact of the modulation scheme used on power consumption. An analysis is performed to compare the power consumption of binary and M-ary modulation. It shows that M-ary modulation consumes more energy when the total energy consumption of the radio is dominated by the energy consumed during startup. Although M-ary modulation reduces the transmit on-time, it requires more complex circuitry which increases the overall power consumption.

### 2.2.4 Node Architecture

A WSN node can consist of sensors, processors and actuators. They may be powered by batteries or mains, or obtain their energy from the environment via solar cells, MEMS resonators or piezoelectrics [53].

Figure 2.3 shows the architecture of a WINS node. The WINS project was discussed briefly in Section 2.2.1. A varied level of node alertness enables the node to conserve power yet at the same time make reliable decisions. Normally, the node will operate at a low level of alertness which consumes minimal energy. If an event appears to be significant, then a higher level of node alertness is invoked and further processing is done. Detection of events is more reliable at a higher level of node alertness, although more

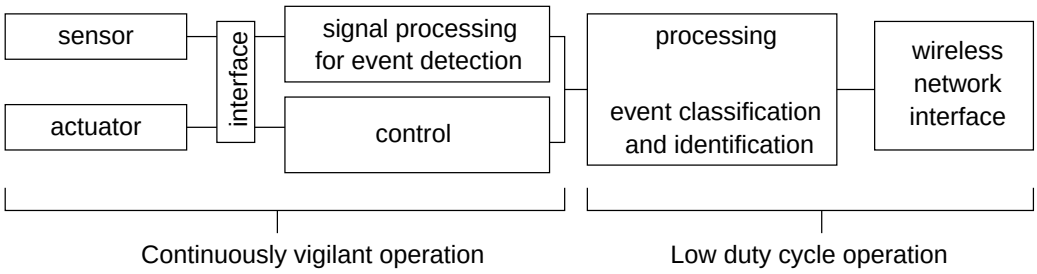


Figure 2.3: Architecture of a WINS node.

energy is consumed. The more frequently invoked functions will be implemented on specialised circuits thereby conserving power, while the more complicated but less frequently needed functions will be executed on general purpose processors. The hardware of the WINS node is discussed in detail by Asada *et al.* [9].

A compact, event-driven operating system called *TinyOS* has been developed by Hill *et al.* [37]. TinyOS only occupies 178 bytes of memory. The design goals of TinyOS are

**Efficient modularity:** The components should be integrated so that there is little overhead when commands and events are passed between them.

**Concurrency-intensive:** The operating system must be able to handle many streams of data at once.

## 2.3 Cooperative Mobile Robotics

A number of surveys of current research within the field of cooperative mobile robotics have been written by Cao *et al.* [12], Parker [49], Dudek *et al.* [22] and Défago [19]. The survey by Cao *et al.* is the most comprehensive, but the survey by Parker is more recent. Dudek *et al.* propose a taxonomy which is intended to provide a theoretical framework for designing multi-agent systems. The survey by Défago is brief; the main aim of this paper is to advocate the development of theoretical models which will provide a foundation for defining the most important problems in cooperative robotics.

In the survey by Cao *et al.* they outline existing work, the fundamental challenges and the direction of research in this field. Five areas of research within cooperative mobile robotics are identified:

**Group Architecture:** The infrastructure upon which the collective behaviours are implemented.

**Resource Conflict:** Mechanisms that allow multiple robots to share their environment, manipulable objects or communications media.

**The Origin of Cooperation:** Achieving cooperation using biological and game-theoretic models, rather than explicitly designing the behaviour into the system.

**Learning:** Methods of making multiple-robot systems discover the optimal control parameter values on their own.

**Geometric Problems:** The spatial problems inherent in mobile robots interacting with each other and the physical world.

In this thesis *geometric problems* will be explored. MWSNs raise many challenging geometric problems because often the positioning of the sensor nodes is critical to the task being carried out by the network.

### 2.3.1 Geometric Problems

There are two types of geometric problems:

- Maintenance of a geometric pattern.
- Formation of a geometric pattern.

In the former problem the geometric pattern has been established but requires maintenance because of some local perturbation. The geometric pattern of a team of robots might be perturbed by robot failures, imperfections in actuators, imperfections in sensors or disturbances from the environment, hence algorithms are needed to reposition the robots if a certain geometric pattern is required to perform the task. In the latter problem no geometric pattern exists and the desired pattern has to be formed from randomly positioned robots. It might not be possible to control the initial placement of the robots, so if a certain geometric pattern is necessary to carry out the given task an algorithm is needed to establish the pattern from a given set of random positions.

The two problems of maintaining a geometric pattern and forming a geometric pattern are similar, thus the distinction between them may not always be clear. A particular problem can lie anywhere on a continuum that ranges from pattern maintenance to pattern formation.

### Maintenance of a Geometric Pattern

An introduction to the basic concepts underlying the maintenance of a formation of autonomous ground vehicles is presented by Moscovitz and DeClaris [47]. In their paper they discuss the challenges that arise from maintaining a formation when the group leader makes a sharp turn or an obstacle is encountered. They also suggest methods of measuring how well the formation is maintained by a single vehicle or the group as a whole.

Earlier Wang [69] developed navigation strategies based on feedback control for a fleet of autonomous robots moving in formation. Wang explored navigation strategies that used both nearest-neighbour and multi-neighbour tracking. A sufficient condition for the asymptotic stability of a desired formation that uses nearest-neighbour tracking was derived. No collision-avoidance strategies were investigated in this work.

Determining the optimal balance between local and global control is a key problem in maintaining the geometric pattern of a robot formation. Parker [48] investigates this problem by testing various control strategies with different combinations of local and global control. Parker concludes from her study that local information should be used to ground global knowledge in the current situation.

Chen and Luh [13, 14] investigate the problem of coordinating a group of robots to transport an object. In their papers they discuss the issues involved in maintaining a certain geometric pattern while moving towards a goal location. They look specifically at making a right-turn with a rectangular formation of robots.

Tan and Lewis [67] develop the concept of a *virtual structure* where the formation is modelled as a rigid body in motion. The virtual structure, which represents the desired relative positions of the robots, is aligned as closely as possible with the current positions of the robots. The robots then adjust their velocity to match their positions with the points of the virtual structure. Effectively, these two steps of the algorithm occur simultaneously. Thus the geometric pattern of the formation is maintained with a *bi-directional* flow of control, unlike typical control systems. The control algorithm was tested by simulation and then implemented on three mobile robots.

A distributed control strategy developed by Yamaguchi [71] enables a formation to adapt according to the positions of landmarks. The paper focusses on using a cordon of mobile robots to block a trespasser entering between two landmarks. Yamaguchi introduces the idea of *formation vectors* which govern the behaviour of the formation. The control strategy assumes that each robot can sense the position and formation

vector of neighbouring robots and the positions of nearby landmarks. Simulations show that the control strategy is theoretically sound.

Fierro *et al.* [27, 28], Das *et al.* [17] and Desai *et al.* [21] develop a general framework for controlling formations of nonholonomic robots. Their work has two major components: (a) maintenance of a desired formation and (b) change of formation. A formation is maintained using two forms of control:

**Separation-Bearing Control:** The robot maintains a specified distance and bearing from another robot.

**Separation-Separation Control:** The robot maintains specified distances from two other robots.

**Separation Distance-To-Obstacle Control:** The robot maintains specified distances from another robot and the boundary of an obstacle.

The positional dependencies between neighbouring robots are described via a *control graph*. On the *control graph* each robot is shown as a vertex and the positional dependencies are represented by directed edges, which are labelled $l - \psi$ or $l - l$ depending on the type of control. Strategies for switching formations are investigated to enable a formation to adapt to imposed constraints, such as obstacles. Desai *et al.* [20] develop an algorithm for enumerating the valid control graphs for a given number of vertices. An algorithm for determining the transitions necessary to switch from one formation to another is also developed in [20]. The control framework is demonstrated with simulations and tests on robots.

Balch and Arkin [10] implement reactive behaviours for formation control in simulations, on mobile robots and on DARPA's Unmanned Ground Vehicles. In their experiments they evaluate the performance of four types of formations, shown in Figure 2.4, and three methods for determining a robot's relative position within a formation. The following methods for formation position determination were investigated:

**Unit-centre-referencing:** The *unit-centre*, also called the *centroid*, is the point $(x_{\text{av}}, y_{\text{av}})$, where $x_{\text{av}}$ is the average $x$-coordinate for all robots in the formation and $y_{\text{av}}$ is the average $y$-coordinate for all robots in the formation. Each robot determines its relative position from the unit-centre.

**Leader-referencing:** A robot is designated as the leader of the formation. The robots calculate their relative position from the leader. The leader is not involved in

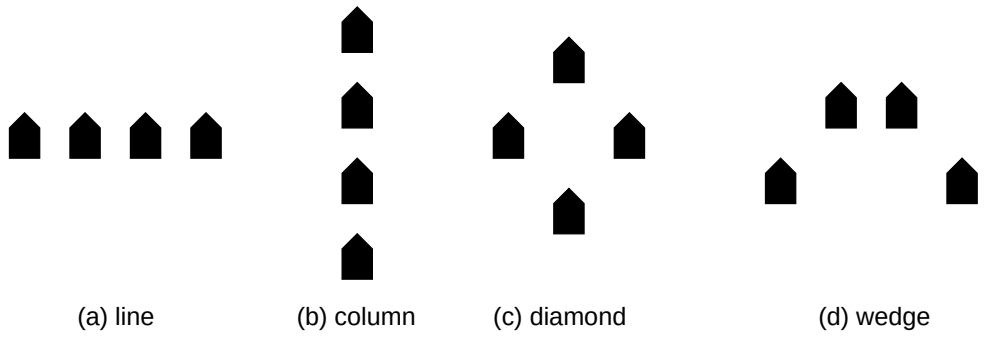


Figure 2.4: Formations investigated by Balch and Arkin in their experiments.

maintaining the formation.

**Neighbour-referencing:** Each robot calculates its relative position from a neighbouring robot.

They demonstrated 90° turns and obstacle avoidance with line, column, diamond and wedge formations using unit-centre-, leader- and neighbour-referencing. The following conclusions were made:

- For 90° turns, diamond formations performed the best with unit-centre-referencing, while wedge and line formations performed the best with leader-referencing.
- For obstacle avoidance, column formations performed the best with both unit-centre and leader-referencing.
- Unit-centre-referenced formations performed better than leader-referenced formations in most cases.

### Formation of a Geometric Pattern

Sugihara and Suzuki [66] develop distributed algorithms for forming geometric patterns with mobile robots. They present the following algorithms:

**Algorithm CIRCLE.** Let $D$ be the desired diameter of the circle and $\delta$ be a small constant greater than zero. For a given robot $R$, let $R'$ be the farthest robot and $R''$ be the nearest robot. Let $d$ be the distance between $R$ and $R'$. We consider three cases:

**Case 1.** If $d > D$, then $R$ moves toward $R'$.

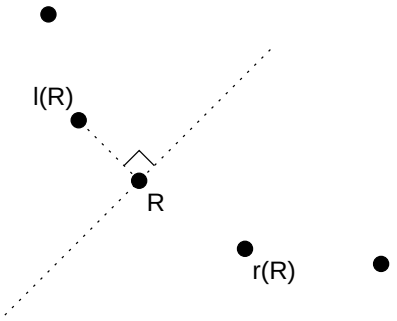


Figure 2.5: The left and right neighbours $l(R)$ and $r(R)$ of a robot $R$.

**Case 2.** If $d < D - \delta$, then $R$ moves away from $R'$.

**Case 3.** If $D - \delta \leq d \leq D$, then $R$ moves away from $R''$

Simulations show that sometimes a *Reuleaux's triangle* results from Algorithm CIRCLE. Chen and Luh [13, 14] modify the algorithm to incorporate collision avoidance.

**Algorithm CONTRACTION.** The following algorithm can be used to form an $n$-sided polygon when $n \geq 3$ or distribute the robots uniformly along a line segment formed with Algorithm FILLPOLYGON. The *left* neighbour of a given robot $R$, denoted by $l(R)$, is defined as the closest robot to $R$. The *right* neighbour of $R$, $r(R)$, is defined as the closest robot to $R$ on the opposite side of a line perpendicular to the line segment from $R$ to $l(R)$, as shown in Figure 2.5. If a polygon is being formed, the robots first execute Algorithm CIRCLE until each robot can clearly recognise its left and right neighbours, and then the user selects $n$ robots and shifts them to the positions of the vertices of the desired polygon. The other robots execute Algorithm CONTRACTION, where a robot $R$ continuously monitors $l(R)$ and $r(R)$ and moves to the midpoint of the line segment between $l(R)$ and $r(R)$, as shown in Figure 2.6.

**Algorithm FILLCIRCLE.** This algorithm distributes the robots uniformly within an approximation of a circle. Using the same definitions of $R$, $R'$, $R''$, $d$ and $D$ as for Algorithm CIRCLE, the algorithm is as follows:

**Case 1.** If $d > D$, then $R$ moves toward $R'$.

**Case 2.** If $d \leq D$, then $R$ moves away from $R''$.

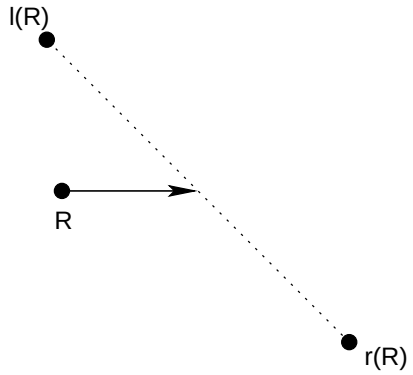


Figure 2.6: Algorithm CONTRACTION.

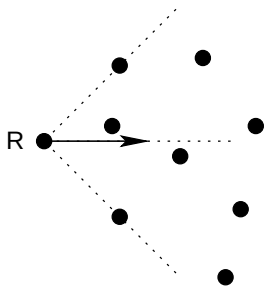


Figure 2.7: Algorithm FILLPOLYGON, Case 1.

As with Algorithm CIRCLE, a *Reuleaux's triangle* may result. Also, "bubbles"—convex regions with no robots—sometimes appear within the filled circle.

**Algorithm FILLPOLYGON.** Algorithm FILLPOLYGON distributes the robots uniformly within an $n$-sided convex polygon. $n$ robots are moved to the desired positions of the vertices while the other robots execute Algorithm FILLPOLYGON, which is as follows:

**Case 1.** If robot $R$ lies at the apex of a wedge that subtends an angle less than $\pi$, then $R$ moves into the wedge along the bisector of the apex, as illustrated in Figures 2.7 and 2.8.

**Case 2.** Otherwise, $R$ moves away from the nearest robot $R''$.

As with Algorithm FILLCIRCLE, "bubbles" can appear within the polygon. A line segment can be formed by moving two robots to the desired endpoints and executing Algorithm FILLPOLYGON on the other robots.

**Algorithm FOLLOW.** Algorithm FOLLOW divides the robots into $m$ groups. As

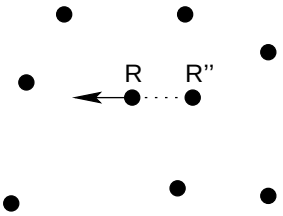


Figure 2.8: Algorithm FILLPOLYGON, Case 2.

with Algorithm CONTRACTION, the robots first execute Algorithm CIRCLE until each robot can clearly recognise its left and right neighbours. A leader for each of the $m$ groups is then selected. To make the groups approximately equal in size the leaders should be chosen so that they are distributed reasonably evenly around the circle. The chosen leaders are moved away from each other while the other robots execute Algorithm FOLLOW. For a robot $R$, let $S$ be the first robot of $l(R)$ and $r(R)$ to move, $a$ be the distance between $R$ and $S$ and $A$ be the initial distance between $R$ and $S$. In Algorithm FOLLOW, each robot $R$ monitors the value of $a$ and if it becomes greater than $A$ by say 20%, then $R$ moves towards $S$.

An important problem is the determination of a robot's relative position within a large group of identical robots. The robots may need to spatially differentiate themselves to form a geometric pattern or assume specialised functions that depend on their position within the formation. Liang and Beni [45] explore this problem using biological inspiration from the process of *cell determination* which occurs during the development of an embryo. They develop an asynchronous method of achieving spatial differentiation in a two-dimensional array of identical robots based on the propagation of boundary patterns through local interaction.

Yamaguchi uses feedback control laws to achieve a desired formation [74] and enclose a target using holonomic [73] or nonholonomic [72] mobile robots. Yamaguchi's approach is decentralised, well grounded mathematically and includes collision avoidance, however it is limited to formations that are already organised into some form of open chain.

Flocchini *et al.* [29–31], Prencipe [56] and Prencipe and Gervasi [57] study the computational issues of pattern formation. A model of time and motion is proposed which the authors believe more realistically models a system of robots. They also investigate how the agreement between the robots' local coordinate systems affects the solvability of a pattern formation problem. The following theorem summarises their

findings:

1. With a common coordinate system of (that is, the robots agree on) both $x$ and $y$ directions and orientations, the robots can form an arbitrary given pattern [31]. This is equivalent to assuming that the robots are equipped with a compass.

2. With agreement on only one axis direction and orientation, the pattern formation problem is unsolvable when $n$ is even, while it can be solved if $n$ is odd [31].

3. With agreement on only one axis direction and orientation, an even number of robots can form only symmetric patterns that have at least one axis of symmetry not passing through any vertex of the pattern [30].

4. With no agreement at all, the robots cannot form an arbitrary given pattern [31].

Défago and Konagaya [18] propose a distributed algorithm for forming the smallest enclosing circle from a given set of initial locations. The motion planning of each robot is based on the relationship between its *Voronoi cell* and the smallest enclosing circle of the beginning configuration. The drawback of this algorithm is that the nodes must have global knowledge of all the robot locations to compute the smallest enclosing circle.

### Maintenance and Formation of a Geometric Pattern

Fredslund and Matarić [33] develop a simple algorithm for establishing and maintaining a formation where each robot, except the leader of the formation, follows a neighbouring robot, as in the approach of Fierro *et al.* [27, 28]. Unlike the approach of Fierro *et al.*, the algorithm is limited to formations with a single chain of positional dependencies. Also, under the assumption that each robot's field of view is limited to $\pm 90°$ from the front, a robot cannot follow another that is behind it. The leader of the formation is called the *conductor*, and the robot that a neighbouring robot chooses to follow is called the *friend* of that robot. Each robot except the conductor maintains a certain distance and bearing from its friend, like separation-bearing control in the approach of Fierro *et al.* If one end of the chain of friends is at the front of the formation, the formation is *noncentred*; otherwise, it is *centred*. A centred and noncentred formation are shown in Figure 2.9. Each robot is assigned a unique numerical ID, which is used to determine the appropriate friend. If a robot has an ID lower than that of the conductor, it chooses a friend with an ID greater than its own. Similarly, if a robot has an ID

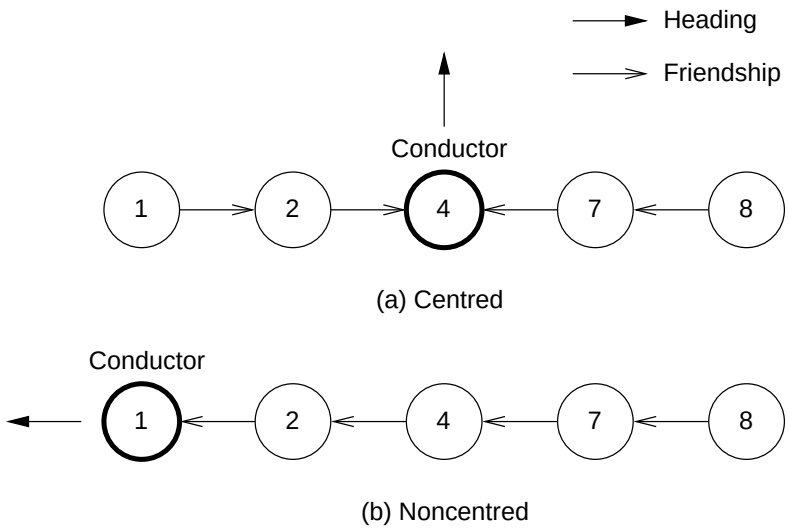


Figure 2.9: Algorithm of Fredslund and Matarić which uses "friendships" to form and maintain a geometric pattern.

greater than that of the conductor, it chooses a friend with an ID lower than its own. For noncentred formations the conductor has the lowest ID, thus the robots will always choose friends with a lower ID for these formations. The soundness of the algorithm was demonstrated with simulations and experiments using four robots.

### 2.3.2 Cooperative Searching

A key task of MWSNs will be *cooperative searching*. An MWSN can be used to search a large area in parallel so that the search can be performed in a fraction of the time it would take with one robot. This is another area where cooperative mobile robotics can make a contribution to the field of MWSNs.

The problem of coordinating the motion of a number of robots to sweep a given area is explored by Kurabayashi *et al.* [8, 41–43]. Kurabayashi *et al.* [42, 43] propose an off-line algorithm for dividing an area to be swept amongst multiple mobile robots. First a path for sweeping the area with a single robot is planned, and then this path is divided among the robots so that the time taken by the last robot to finish its sweep is minimised. The algorithm is extended to include the relocation of obstacles [8, 41].

The on-line algorithm of Min and Yin [46] does not require the location of obstacles to be known in advance, unlike the approach of Kurabayashi *et al.* The area to be searched is divided into square grid cells. The distance from the centre of each grid cell to its corners is equal to the sweeping radius of each robot, thus a grid cell will be searched once a robot passes through the centre of the grid cell. The robots cooperate

through a decentralised, market-like structure where the robots negotiate the task of sweeping each grid cell between themselves.

Goldsmith *et al.* [34] present a decentralised, behaviour based strategy for carrying out a collective search with multiple mobile robots. Each robot has a certain "social status" determined by the intensity of its sensor readings and seeks to improve its status by exploring regions that will improve the robot's present sensor readings. Robots play either an *alpha* role or *beta* role, and change roles according to how present sensor readings compare to neighbouring robots' sensor readings. Robots in the alpha role are adventurous and will venture out of a local minimum, whereas robots in the beta role are more conservative and tend to remain stationary until alpha agents discover regions that will provide a definite increase in the robot's status. The alpha agents seek a trajectory that leads to the global minimum and the beta agents follow the average location of the alpha agents.

In the next chapter, the problem of maintaining the geometric pattern of an MWSN will be explored.

# Chapter 3

# Maintaining the Geometric Pattern of an MWSN

## 3.1 Introduction

A survey of literature forming a basic foundation for the study of MWSNs was given in the previous chapter. In this chapter we explore the first of the two general problems presented in this thesis: *maintaining* the geometric pattern of the nodes in an MWSN.

A fundamental problem in MWSNs is maintaining a certain geometric pattern in a marching array of mobile nodes which search for objects in the environment. When searching for objects such as land mines, it is critical that the entire area swept by the array is covered by the nodes' sensors, thus a certain separation should be maintained between neighbouring nodes. When a node becomes inoperative after being destroyed by a land mine, for example, the neighbouring nodes must move in to fill the gap in the sensors' coverage.

In this chapter we investigate fault-tolerant algorithms for maintaining the spatial pattern of nodes arranged in a line, as illustrated in Figures 3.1 and 3.2. The nodes are advancing forward and one node is destroyed, leaving a "hole" in the coverage of the sensors. The nodes are also misaligned due to differences in the speed of the nodes. Two strategies for maintaining sensor coverage are explored:

1. Maintaining a constant separation between neighbouring nodes (Figure 3.1).
2. Maintaining an equal separation between neighbouring nodes whilst keeping the length of the array fixed (Figure 3.2).

In both cases we adjust the motion of the nodes to align them in a straight line.

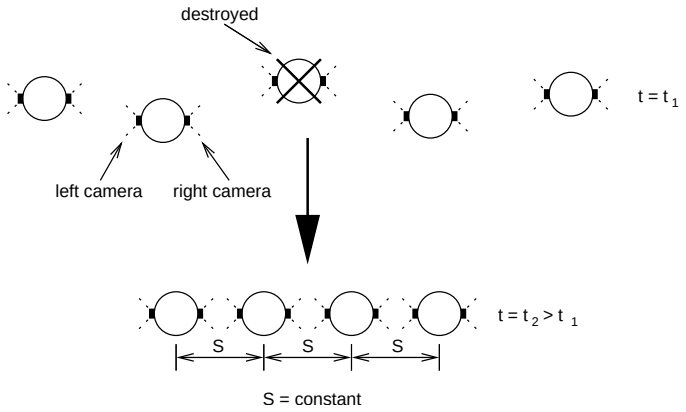


Figure 3.1: Maintaining a constant separation between neighbouring nodes.

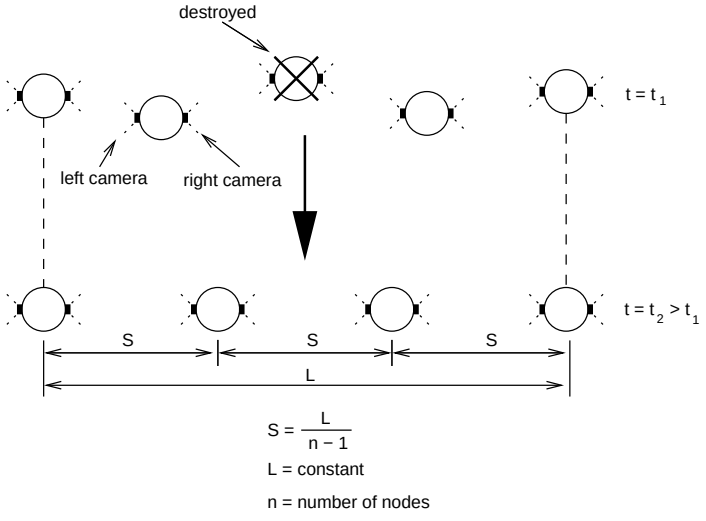


Figure 3.2: Maintaining an equal separation between neighbouring nodes when the length of the array is kept constant.

Each node obtains local information from cameras on the left and right and a compass. We wish to see what can be achieved without wireless communication, so the nodes have no global knowledge.

The organisation of this chapter is as follows. Section 3.2 discusses the vision and motion planning algorithms for implementing strategies 1 and 2 above. The performance of the algorithms is evaluated by simulation studies in Section 3.3, and conclusions are given in Section 3.4.

## 3.2 Algorithms

In the discussion that follows, we assume that the array runs parallel to the $x$-axis in the Cartesian plane.

Three algorithms have been devised:

**Algorithm CS/G:** A *greedy* algorithm for maintaining a constant separation between neighbouring nodes.

**Algorithm CS/SB:** A *state-based* algorithm for maintaining a constant separation between neighbouring nodes.

**Algorithm CL:** An algorithm for maintaining an equal separation between neighbouring nodes when the length of the array is kept constant.

The algorithms only differ in how the $x$-velocity of a node is adjusted to achieve the desired separation from neighbouring nodes, so most of the discussion that follows applies to all three algorithms.

Each algorithm is comprised of two components:

**Vision:** To determine the relative positions of other nodes using images captured by the cameras.

**Motion Planning:** To determine the appropriate adjustment in the node's velocity to maintain the desired formation.

Each of these components is described in detail below.

### 3.2.1 Vision

A number of assumptions have been made about the environment:

1. The ground is an infinitely large, perfectly flat plane.
2. The nodes are cylindrical in shape and always upright.
3. The cameras are located on opposite sides of the node midway between the top and bottom.
4. No other objects are present.

As a result of the above conditions, finding the nodes in the images is simply a matter of searching for objects approximately rectangular in shape centred on the horizon.

The image was repeatedly segmented using a recursive algorithm. As illustrated in Figure 3.3, regions of interest are identified and segmented using histograms of (a) the number of feature pixels counted in each column, and (b) the number of feature pixels counted in each row of the segment. A pixel is counted as a feature pixel when the following condition is true:

$$(r - R)^2 + (g - G)^2 + (b - B)^2 < D^2$$

where $R$, $G$ and $B$ are the known RGB values for the colour of the sensor nodes, $r$, $g$ and $b$ are the RGB values of the pixel being examined and $D$ is some chosen constant.

The algorithm consists of two stages:

1. Isolating a region containing a node or cluster of nodes.
2. Recursively segmenting the region along the edges where one node occludes another.

The algorithm will now be described in detail. In the discussion that follows, the word "cluster" will be used to refer to one or more nodes.

### Isolation of Node Clusters

A histogram of the feature pixels counted in each column of the image is formed. If, for a range of columns $c_{\min}, \ldots, c_{\max}$

1. $n_c(j) \geq T_c$ for $j = c_{\min}, \ldots, c_{\max}$, where $n_c(j)$ is the number of feature pixels counted in column $j$ and $T_c$ is the minimum number of feature pixels that may be allowed in any column of a segment; and
2. $c_{\max} - c_{\min} + 1 \geq T_w$, where $T_w$ is the minimum allowed width of a segment;

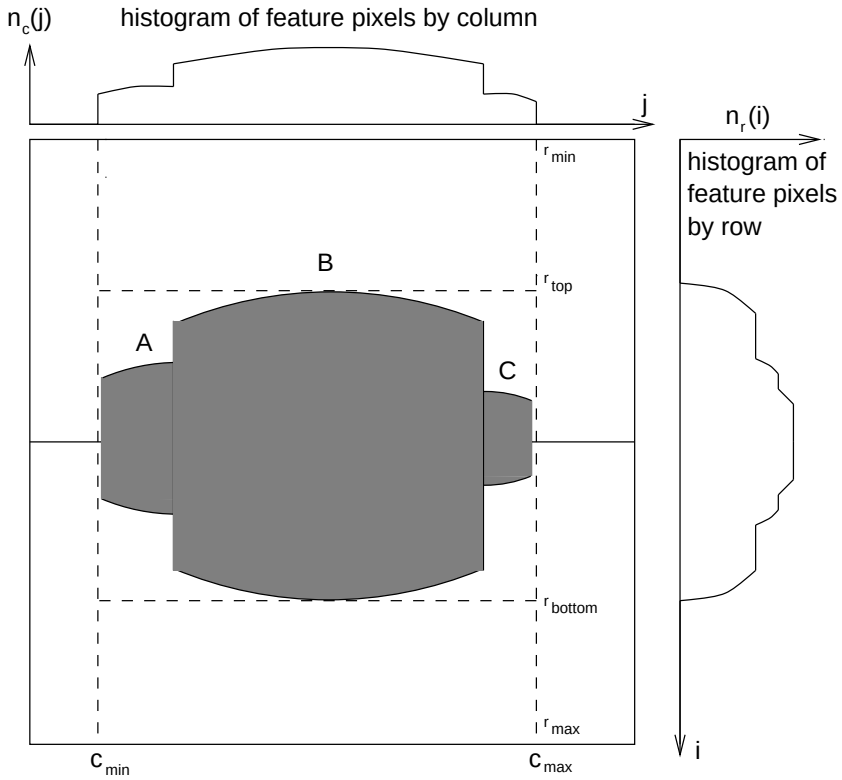


Figure 3.3: Isolating the nodes in an image using histograms of the number of feature pixels counted in the columns and rows.

then the segment formed by columns $c_{\min}, \ldots, c_{\max}$ is processed in the next stage of the algorithm. $T_{\mathrm{c}}$ and $T_{\mathrm{w}}$ are chosen to exclude small regions of "noise".

### Recursive Segmentation Along Occluding Edges

Given a segment of the image bounded by columns $c_{\min}$ and $c_{\max}$ and rows $r_{\min}$ and $r_{\max}$, the rows above and below the node cluster are removed from the segment using a histogram of the feature pixels counted along each row, as shown in Figure 3.3. The criteria for identifying the range of rows to be isolated, $r_{\mathrm{top}}, \ldots, r_{\mathrm{bottom}}$, is similar to that used for the initial identification of the column range in the first stage:

1. $n_{\mathrm{r}}(i) \geq T_{\mathrm{r}}$ for $i = r_{\mathrm{top}}, \ldots, r_{\mathrm{bottom}}$, where $n_{\mathrm{r}}(i)$ is the number of feature pixels counted in row $j$ between columns $c_{\min}$ and $c_{\max}$ and $T_{\mathrm{r}}$ is the minimum number of feature pixels that may be allowed in any row of a segment; and

2. $r_{\mathrm{bottom}} - r_{\mathrm{top}} + 1 \geq T_{\mathrm{h}}$, where $T_{\mathrm{h}}$ is the minimum allowed height of a segment.

It is possible for more than one range of rows to satisfy this criteria, due to noise, so the longest range is always chosen.

As illustrated in Figure 3.4, the distance $d$ (along an axis perpendicular to the image plane) of the closest node in the cluster can then be calculated using the height of the node $h$ (known *a priori*), the distance of the image plane from the viewpoint $d'$ and the height of the segment projected onto the image plane $h'$:

$$d = \frac{hd'}{h'}$$

Once the distance of the closest node in the cluster has been calculated, we then determine whether the cluster consists of more than one node. The criterion for determining whether the cluster consists of more than one node is as follows. Let $R_{\max}$ be the actual radius of a node (known *a priori*) plus some error margin and *assume* that the cluster is a single node with an unknown radius $r_{\mathrm{node}}$. We test this assumption by calculating $r_{\mathrm{node}}$ from $d$ and the distance of the cluster's projected edges from the centre of the image plane. If $r_{\mathrm{node}} > R_{\max}$ our assumption must be false and the cluster consists of more than one node.

$r_{\mathrm{node}}$ is determined as follows. Consider a two-dimensional plane that intersects the viewpoint and is parallel to the ground, as illustrated in Figure 3.5. Let

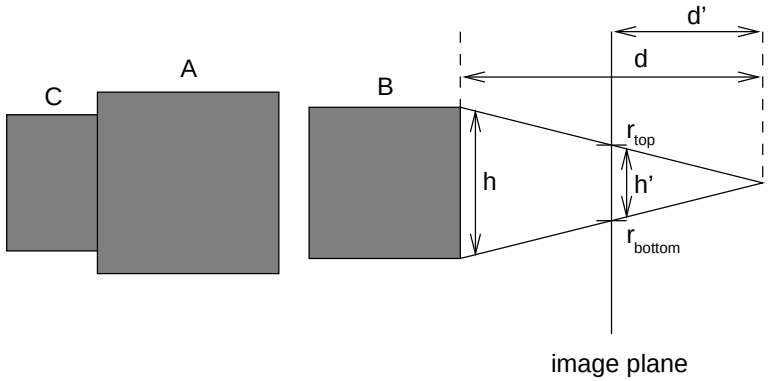


Figure 3.4: The view plane's distance $d'$, projected height of the cluster $h'$ and the actual height of the cluster $h$, which are used to calculate the cluster's distance $d$ along an axis perpendicular to the image plane.

| | | |
|---|---|---|
| $\mathbf{A}$ | = | vector beginning at the viewpoint and finishing at the projected left edge of the cluster; |
| $\mathbf{B}$ | = | vector beginning at the viewpoint and finishing at the projected right edge of the cluster; |
| $\mathbf{A}'$ | = | vector $\mathbf{A}$ rotated 90° clockwise; |
| $\mathbf{B}'$ | = | vector $\mathbf{B}$ rotated 90° anticlockwise; |
| $k_1$ | = | unknown scalar; and |
| $k_2$ | = | unknown scalar; |

Note that we denote a vector with components $x$ and $y$ by $\langle x, y\rangle$ and the magnitude of a vector $\mathbf{V}$ by $\|\mathbf{V}\|$. Also note that the vector obtained by rotating $\langle x, y\rangle$ 90° clockwise is $\langle y, -x\rangle$, and the vector obtained by rotating $\langle x, y\rangle$ 90° anticlockwise is $\langle -y, x\rangle$. As illustrated in Figure 3.5, we wish to find $r_{\text{node}}$ such that a circle with this radius is tangential to the line containing the vector $\mathbf{A}$, the line containing vector $\mathbf{B}$ and the line $x = d$. Thus we find $r_{\text{node}}$, $k_1$ and $k_2$ such that

$$k_1\mathbf{A} + r_{\text{node}}\frac{\mathbf{A}'}{\|\mathbf{A}'\|} = k_2\mathbf{B} + r_{\text{node}}\frac{\mathbf{B}'}{\|\mathbf{B}'\|}.$$

Let $a$ and $b$ be the $y$-components of vectors $\mathbf{A}$ and $\mathbf{B}$ respectively. Then

$$k_1\langle d', a\rangle + r_{\text{node}}\frac{\langle a, -d'\rangle}{\|\langle a, -d'\rangle\|} = k_2\langle d', b\rangle + r_{\text{node}}\frac{\langle -b, d'\rangle}{\|\langle -b, d'\rangle\|}.$$

If we separate the $x$- and $y$-components of the vector equation above, we get the following two equations:

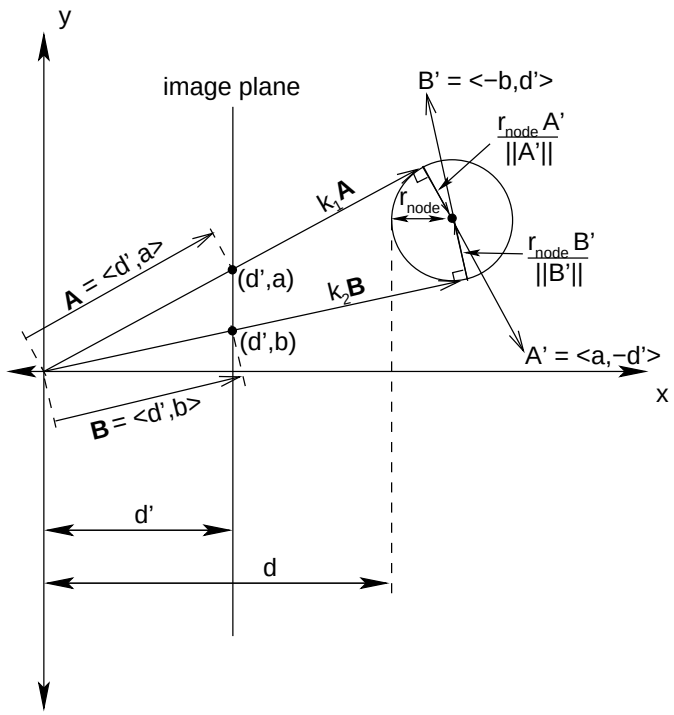


Figure 3.5: Vectors used to calculate the radius of a node projected onto the view plane.

$$k_1 d' + \frac{r_{\text{node}} a}{\|\langle a, -d' \rangle\|} = k_2 d' - \frac{r_{\text{node}} b}{\|\langle -b, d' \rangle\|} \tag{3.1}$$

$$k_1 a - \frac{r_{\text{node}} d'}{\|\langle a, -d' \rangle\|} = k_2 b + \frac{r_{\text{node}} d'}{\|\langle -b, d' \rangle\|} \tag{3.2}$$

As illustrated in Figure 3.5, the distance of the centre of the cluster from the viewpoint is $d + r_{\text{node}}$. Thus we equate the left hand side of Equation 3.1 to $d + r_{\text{node}}$ to obtain a third equation:

$$k_1 d' + \frac{r_{\text{node}} a}{\|\langle a, -d' \rangle\|} = d + r_{\text{node}} \tag{3.3}$$

Solving Equation 3.3 for $k_1$, we get

$$k_1 = \frac{1}{d'} \left[ d + r_{\text{node}} - \frac{r_{\text{node}} a}{\|\langle a, -d' \rangle\|} \right]. \tag{3.4}$$

Likewise, we equate the right hand side of Equation 3.1 with $d + r_{\text{node}}$ and solve for $k_2$:

$$k_2 = \frac{1}{d'} \left[ d + r_{\text{node}} + \frac{r_{\text{node}} b}{\|\langle -b, d' \rangle\|} \right] \tag{3.5}$$

Finally, we substitute Equations 3.4 and 3.5 into 3.2 and solve for $r_{\text{node}}$.

$$\frac{a}{d'}\left[d + r_{\text{node}} - \frac{r_{\text{node}}a}{\|\langle a, -d'\rangle\|}\right] - \frac{r_{\text{node}}d'}{\|\langle a, -d'\rangle\|} = \frac{b}{d'}\left[d + r_{\text{node}} + \frac{r_{\text{node}}b}{\|\langle -b, d'\rangle\|}\right] + \frac{r_{\text{node}}d'}{\|\langle -b, d'\rangle\|}$$

$$\begin{aligned}\frac{ad}{d'} - \frac{bd}{d'} &= \frac{r_{\text{node}}a^2}{d'\,\|\langle a, -d'\rangle\|} + \frac{r_{\text{node}}b^2}{d'\,\|\langle -b, d'\rangle\|} + \frac{r_{\text{node}}d'}{\|\langle a, -d'\rangle\|} + \frac{r_{\text{node}}d'}{\|\langle -b, d'\rangle\|} + \frac{r_{\text{node}}b}{d'} - \frac{r_{\text{node}}a}{d'} \\ r_{\text{node}} &= \frac{d\,(a-b)}{\frac{a^2}{\|\langle a,-d'\rangle\|} + \frac{b^2}{\|\langle -b,d'\rangle\|} + \frac{d'^2}{\|\langle a,-d'\rangle\|} + \frac{d'^2}{\|\langle -b,d'\rangle\|} + b - a}\end{aligned}$$

We then consider two cases:

**Case 1 ($r_{\text{node}} > R_{\text{max}}$):** When $r_{\text{node}} > R_{\text{max}}$ the segment contains more than one node, so the segment is divided into two and the routine calls itself for each of these two segments. The segment is divided vertically where the sharpest change occurs in $n_{\text{c}}(j)$. That is, we compute the gradient $g(j)$ for each column $j$ (where $g$ is defined) of $n_{\text{c}}(j)$ and find the column $c_{\text{x}}$ where $|g|$ is at a maximum. If the maximum of $|g|$ happens to be zero, then the segment is split half-way between $c_{\text{min}}$ and $c_{\text{max}}$. As illustrated in Figure 3.6, the sub-segment on the left will be bounded by columns $c_{\text{min}}$, $c_{\text{x}}$ and rows $r_{\text{top}}$, $r_{\text{bottom}}$, and the sub-segment on the right will be bounded by columns $c_{\text{x}} + 1$, $c_{\text{max}}$ and rows $r_{\text{top}}$, $r_{\text{bottom}}$. The recursive algorithm is then applied to these two sub-segments.

**Case 2 ($r_{\text{node}} \leq R_{\text{max}}$):** In this case the cluster contains a single node (which may be occluded). The node's displacement along an axis perpendicular to the image plane has already been calculated, so all that remains to be determined is its displacement along an axis parallel to the image plane, denoted by $l$. As shown in Figure 3.7, $l$ is determined using the value of $d$ calculated earlier and the projected lateral displacement $l'$:

$$l = \frac{dl'}{d'}$$

### 3.2.2 Motion Planning

The $x$- and $y$-components of the velocity are treated separately in the motion planning. The $x$-component depends purely on the relative $x$-coordinates of neighbouring nodes, and likewise the $y$-component depends purely on the relative $y$-coordinates of

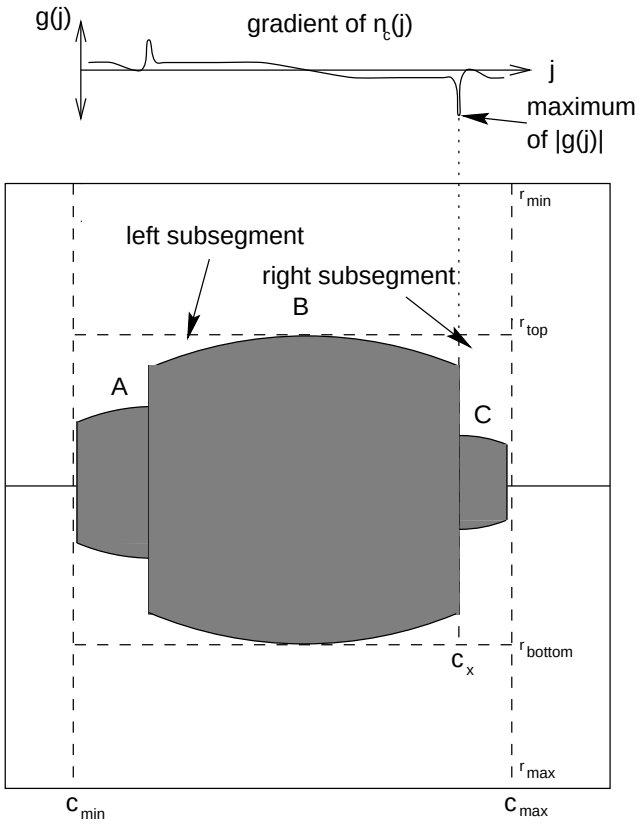


Figure 3.6: Dividing the segment along the column $c_x$ where the peak of $|g(j)|$ occurs. The recursive algorithm is then applied to these two sub-segments.

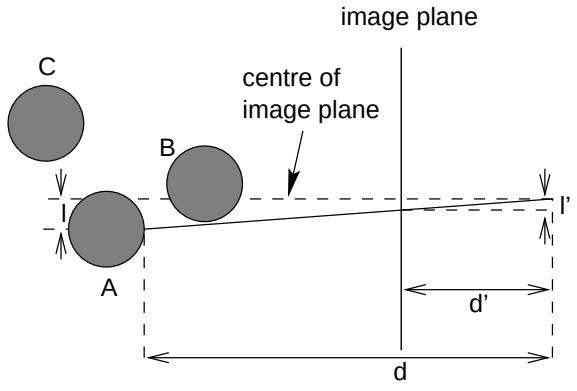


Figure 3.7: Projected lateral displacement of a node from the centre of the image plane $l'$, which is used to calculate the actual lateral displacement $l$.

neighbouring nodes.

The $x$-velocity and $y$-velocity are comprised of a constant component, which is the same for all nodes, and when necessary a certain adjustment to change the relative position of the node. To simplify discussion we will assume that the constant component is zero when introducing the concepts behind the algorithms. Note that the constant component does not affect the relative positions of the nodes and therefore does not affect decisions made with the algorithm.

### $x$-velocity

**Algorithm CS/G** The approach of this algorithm is for the nodes at each end to maintain a separation of $S$ from their neighbour while the other nodes maintain an equal separation from their neighbour on each side. When the system reaches a state of equilibrium, all neighbouring nodes are separated by $S$.

The algorithm is as follows. If another node is in-line with the node, that is, has the same $x$-coordinate, then move left or right, choosing the direction at random. Otherwise, if there is a neighbour on the left as well as on the right, move to the mid-point between them. If there is only a neighbour on one side, that is the node is at one end of the array, then maintain a constant distance $S$ from the neighbouring node. Note that for the problem presented in this chapter it is not necessary to handle the case when the node is in-line with another since we only wish to handle limited deviations from their desired positions, but this case needs to be handled when forming a line from a random set of starting positions in the next chapter.

Let

| | | |
|---|---|---|
| neighbour_in-line | = | Boolean variable that is TRUE when another node has the same $x$-coordinate and FALSE otherwise; |
| $I$ | = | random variable that assumes a value of 0 or 1 with equal probability; |
| left_neighbour | = | Boolean variable that is TRUE when a neighbour is detected on the left and FALSE otherwise; |
| right_neighbour | = | Boolean variable that is TRUE when a neighbour is detected on the right and FALSE otherwise; |
| $d_{\text{left}}$ | = | $x$-distance of the left neighbour; |
| $d_{\text{right}}$ | = | $x$-distance of the right neighbour; |
| $V_x$ | = | constant component of the $x$-velocity; |
| $\Delta v_x$ | = | constant value by which the $x$-velocity is increased or decreased, when necessary; and |
| $v_x$ | = | set $x$-velocity. |

Algorithm 1 shows the algorithm for setting the $x$-velocity in Algorithm CS/G.

**Algorithm 1** Setting the $x$-velocity in Algorithm CS/G.

**if** (neighbour_in-line) **then**
  $v_x \leftarrow V_x + (1 - 2I)\Delta v_x$
**else if** (left_neighbour **and** right_neighbour **and** $d_{\text{left}} < d_{\text{right}}$)
 **or** (left_neighbour **and not** right_neighbour **and** $d_{\text{left}} < S$)
 **or** (**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} > S$) **then**
  $v_x \leftarrow V_x + \Delta v_x$
**else if** (left_neighbour **and** right_neighbour **and** $d_{\text{left}} > d_{\text{right}}$)
 **or** (left_neighbour **and not** right_neighbour **and** $d_{\text{left}} > S$)
 **or** (**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} < S$) **then**
  $v_x \leftarrow V_x - \Delta v_x$
**else**
  $v_x \leftarrow V_x$
**end if**

**Algorithm CS/SB** Algorithm CS/SB was designed to use as little movement as necessary to eliminate openings in an array. Basically a node moves to decrease the separation on one side only when the separation on the opposite side is less than $S$ plus an error margin, while the nodes at each end of the array maintain a separation of $S$ as with Algorithm CS/G. To accomplish this a variable is needed to store the velocity in the $x$-direction that was previously set. Since the chosen velocity depends on the value of this variable, this algorithm is not greedy, unlike Algorithm CS/G.

In the discussion that follows:

- The *slack* is the separation between two neighbouring nodes minus the desired separation $S$.
- An *opening* is defined as a separation between two neighbouring nodes greater than $S_{\text{open}} = S + \Delta S_{\text{open}}$, where $\Delta S_{\text{open}}$ is some constant chosen to detect openings.
- A *break* is defined as a separation between two neighbouring nodes greater than $S_{\text{break}} = S_{\text{open}} + \Delta S_{\text{break}}$, where $\Delta S_{\text{break}}$ is some constant chosen to detect slack collisions.

As will become clear later, $\Delta S_{\text{open}}$ and $\Delta S_{\text{break}}$ are constants chosen according to the accuracy of distance measurements and the precision of motion control.

In understanding the algorithm, it is helpful to consider the "flow" of slack through the array as well as the movement of nodes. When a node moves to the right, for

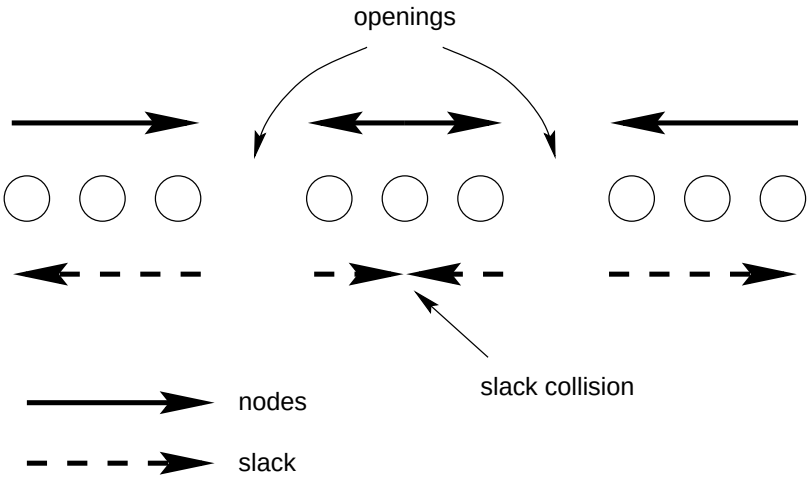


Figure 3.8: The "flow" of nodes and slack when the state-based algorithm is used. When slack from two different openings "collide", the nodes stop moving.

example, it also transfers slack from the gap on the right to the gap on the left. Thus slack and nodes move in opposite directions, analogous to the flow of positive and negative charges in electric current. The basic idea of the state-based algorithm is to transfer slack from an opening to the ends of the array.

As illustrated in Figure 3.8, global knowledge improves efficiency, otherwise when more than one opening exists, nodes will move in opposing directions and create new openings in the array. A more efficient strategy is to eliminate the openings that are closest to ends of the array first, followed by those closer to the middle. Put another way, we want to prevent "collisions" between slack flowing from different openings.

The state-based algorithm saves movement because nodes remain stationary when slack from two different openings collide. Thus there will be little movement in the nodes lying between the two openings in Figure 3.8. It can be seen that even though no wireless communication is used, global knowledge of the presence of openings is communicated through movement.

The state-based algorithm is as follows. When an opening exists on the right but not the left, move to the right until the separation on the right is $S$ or a break occurs on the left, which means a slack collision has occurred. The case is similar when an opening exists on the left but not the right. The case when there is a neighbour on only one side is handled as it is for Algorithm CS/G. Algorithm 2 shows how the $x$-velocity is set in Algorithm CS/SB.

The constants $\Delta S_{\text{open}}$ and $\Delta S_{\text{break}}$ must be chosen for robustness and stability. If neighbouring nodes disagree on whether an opening exists between them because of

**Algorithm 2** Setting the $x$-velocity in Algorithm CS/SB.

**if** {left_neighbour **and** right_neighbour
  **and** [(state = stationary **and** $d_{\text{left}} < S_{\text{open}}$ **and** $d_{\text{right}} > S_{\text{open}}$)
   **or** (state = moving_right **and** $d_{\text{left}} < S_{\text{break}}$ **and** $d_{\text{right}} > S$)]}
 **or** {left_neighbour **and not** right_neighbour **and** $d_{\text{left}} < S$}
 **or** {**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} > S$} **then**
  $v_x \leftarrow V_x + \Delta v_x$
  state $\leftarrow$ moving_right
**else if** {left_neighbour **and** right_neighbour
  **and** [(state = stationary **and** $d_{\text{left}} > S_{\text{open}}$ **and** $d_{\text{right}} < S_{\text{open}}$)
   **or** (state = moving_left **and** $d_{\text{left}} > S$ **and** $d_{\text{right}} < S_{\text{break}}$)]}
 **or** {left_neighbour **and not** right_neighbour **and** $d_{\text{left}} > S$}
 **or** {**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} < S$} **then**
  $v_x \leftarrow V_x - \Delta v_x$
  state $\leftarrow$ moving_left
**else**
  $v_x \leftarrow V_x$
  state $\leftarrow$ stationary
**end if**

error in the measurement of the separation, then the flow of slack might be blocked.

**Algorithm CL** As shown in Algorithm 3, the algorithm for maintaining an equal separation between neighbouring nodes whilst keeping the length of the array constant, Algorithm CL, is similar to the greedy algorithm for maintaining a constant separation between neighbouring nodes, Algorithm CS/G. The only difference is that the nodes at each end have no movement in the $x$-direction, as these nodes define the length of the array.

**Algorithm 3** Setting the $x$-velocity in Algorithm CL.

**if** (neighbour_in-line) **then**
  $v_x \leftarrow V_x + (1 - 2I)\Delta v_x$
**else if** (left_neighbour **and** right_neighbour **and** $d_{\text{left}} < d_{\text{right}}$) **then**
  $v_x \leftarrow V_x + \Delta v_x$
**else if** (left_neighbour **and** right_neighbour **and** $d_{\text{left}} > d_{\text{right}}$) **then**
  $v_x \leftarrow V_x - \Delta v_x$
**else**
  $v_x \leftarrow V_x$
**end if**

### $y$-velocity

Determining the appropriate $y$-velocity to align a node with its neighbours is basically a matter of determining the average relative $y$-displacement of neighbouring nodes, as

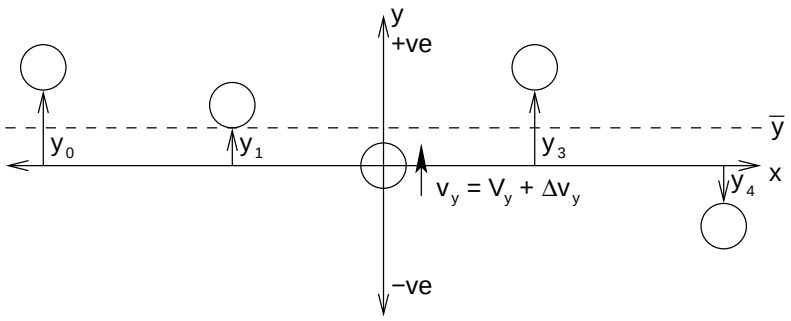


Figure 3.9: Using the average relative $y$-displacement of neighbouring nodes to align the nodes along the $y$-axis.

illustrated in Figure 3.9. In the example shown in this figure the average is greater than zero, so the $y$-velocity of the node at the origin is increased by $\Delta v_y$ to align itself with its neighbours. The $y$-displacement is defined to be positive in one direction and negative in the other. The $y$-velocity is chosen according to the sign of the average $y$-displacement.

Let

$y_i$ = relative $y$-displacement of neighbour $i$;
$n$ = number of known neighbours;
$V_y$ = constant component of the $y$-velocity;
$\Delta v_y$ = a constant value by which the $y$-velocity is increased or decreased, when necessary; and
$v_y$ = set $y$-velocity.

**Algorithm 4** Setting the $y$-velocity in Algorithms CS/G, CS/SB and CL.

$\bar{y} \leftarrow \frac{1}{n} \sum_i y_i$
**if** $\bar{y} < 0$ **then**
  $v_y \leftarrow V_y - \Delta v_y$
**else if** $\bar{y} > 0$ **then**
  $v_y \leftarrow V_y + \Delta v_y$
**else**
  $v_y \leftarrow V_y$
**end if**

Note that the algorithm does not depend on each node having accurate knowledge of the $y$-displacement of every other node. In fact, they only need to know the $y$-displacement of their left and right neighbours. However, the more global knowledge that is used in the decision the more efficient and robust the performance.

## 3.3 Experiments

### 3.3.1 Simulation Setup

The algorithms were tested by simulation. The paths of the nodes were computed in discrete steps according to the algorithm being tested. To simulate vision, the *POV-Ray* ray-tracer [4] was used to produce images of the scene as viewed through the side cameras of each node.

To test the robustness of the algorithms, a random error with a Gaussian distribution was added to the following:

**Image pixels:** Error was added to the red, green and blue values of each pixel.

**Orientation of cameras:** A second source of error in the node's local knowledge is the compass reading, which is needed to determine the orientation of the cameras. The impact of error in the compass reading can be investigated by either fixing the compass reading and randomly varying the orientation of the cameras, or *vice versa*. In these experiments the former approach was used, as illustrated in Figure 3.10. The orientation of the cameras was rotated by a random angle while it is assumed in the vision system that they were pointing parallel to the $x$-axis.

**Velocity:** Without any error or constant velocity component, a node may move in one of eight directions depending on the $x$-adjustment, which may be $-\Delta v_x$, 0 or $+\Delta v_x$, and the $y$-adjustment, which may be $-\Delta v_y$, 0 or $+\Delta v_y$. Thus the velocity is expressed as a 2-D vector and an error is added to both the $x$- and $y$-components. If the error on its own is treated as a vector, the magnitude of the error vector has a *Rayleigh distribution* [44] and the angle is uniformly distributed.

The simulations were carried out under the following conditions:

- The nodes are cylindrical with a height of 0.3 distance units and radius of 0.15 distance units. The cameras are positioned on the circumference of the cylinder mid-way between the top and bottom of the node.
- The "snapshot" taken by each camera is 200 pixels wide and 200 pixels high. The image plane is located 1 distance unit in front of the camera and is 4 distance units wide and 4 distance units high. The colour of each pixel is represented by RGB values ranging from 0 to 255. The chosen threshold value for deciding whether a pixel's colour is close enough to belong to a node is 50.

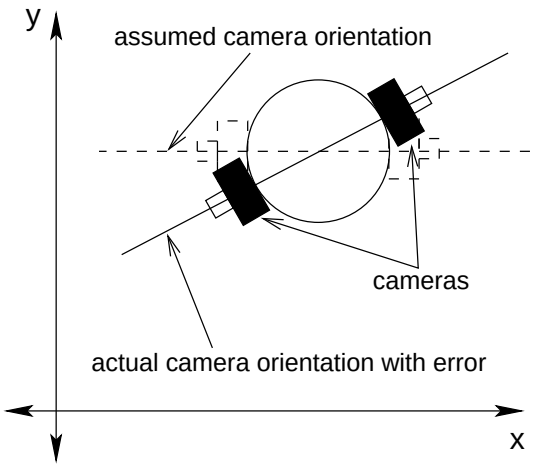


Figure 3.10: Error introduced into the orientation of the cameras.

- For Algorithms CS/G and CS/SB, the desired separation between neighbouring nodes is $S = 1$.
- The magnitude of the $x$-adjustment for eliminating openings is $\Delta v_x = 0.05$ distance units/time unit. The magnitude of the $y$-adjustment for aligning the nodes is $\Delta v_y = 0.05$ distance units/time unit. For interpreting the results it is desirable that subsequent plots of the nodes' locations don't overlap, so a constant component is included in the velocity of the nodes. Each node moves in the positive $y$-direction at 0.2 distance units/time unit, excluding any adjustment to maintain the geometric pattern.
- Each simulation lasted for a duration of 100 time units.

Figures 3.11 to 3.15 show the initial locations of the nodes for all simulations. Configuration 3A was used for investigating the effect of parameters $S_{\text{open}}$ and $S_{\text{break}}$ on the performance of Algorithm CS/SB and the robustness of the algorithms in Section 3.3.3. To demonstrate how Algorithm CS/G adjusts to the addition of nodes, a node was added at $t = 50$, position (0,5) using Configuration 3B. At this time the nodes were expected to be settled at locations close to their desired positions and the centroid is expected to be near the point (0,5) since the constant component of the $y$-velocity is 0.2 distance units/time unit. Algorithm CS/G's handling of configurations with staggered $y$-coordinates was explored with Configuration 3C. A node was removed at $t = 50$ using Configuration 3D to demonstrate how Algorithm CS/SB adjusts to the removal of nodes. Algorithm CL's handling of nodes beginning in an angular formation was

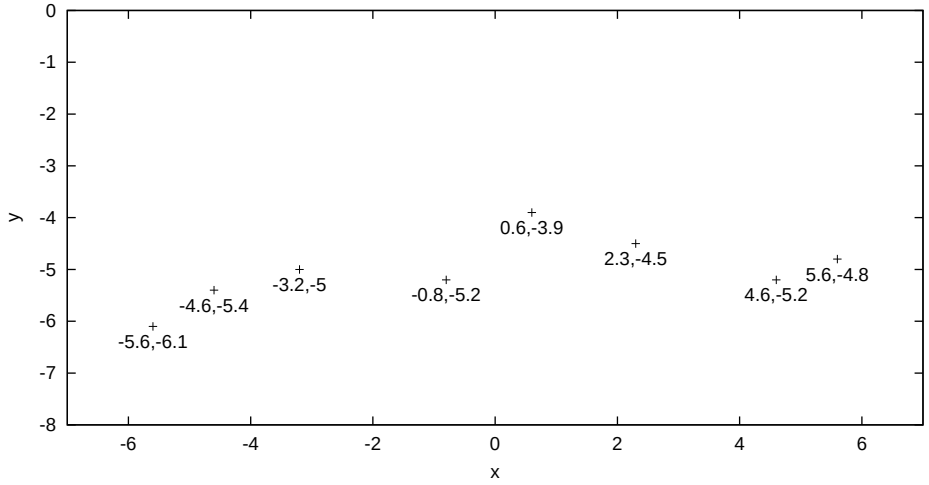


Figure 3.11: Configuration 3A.

investigated with Configuration 3E.

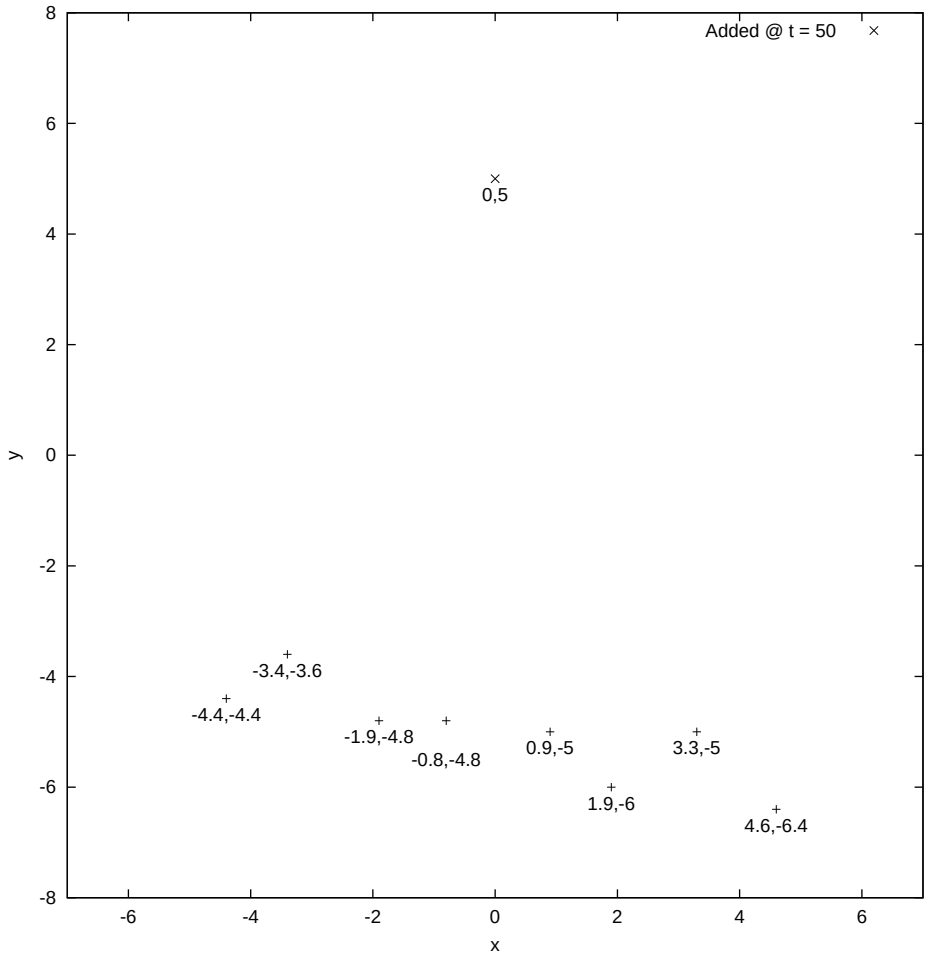


Figure 3.12: Configuration 3B.

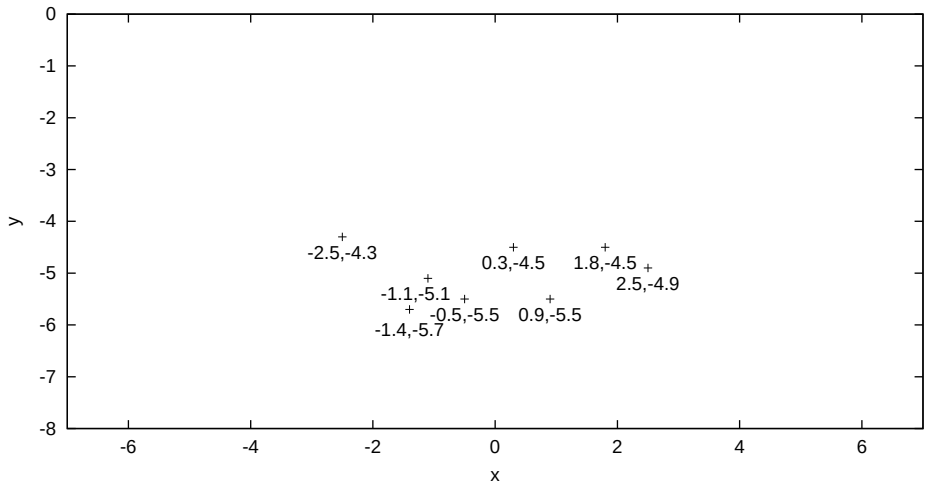


Figure 3.13: Configuration 3C.

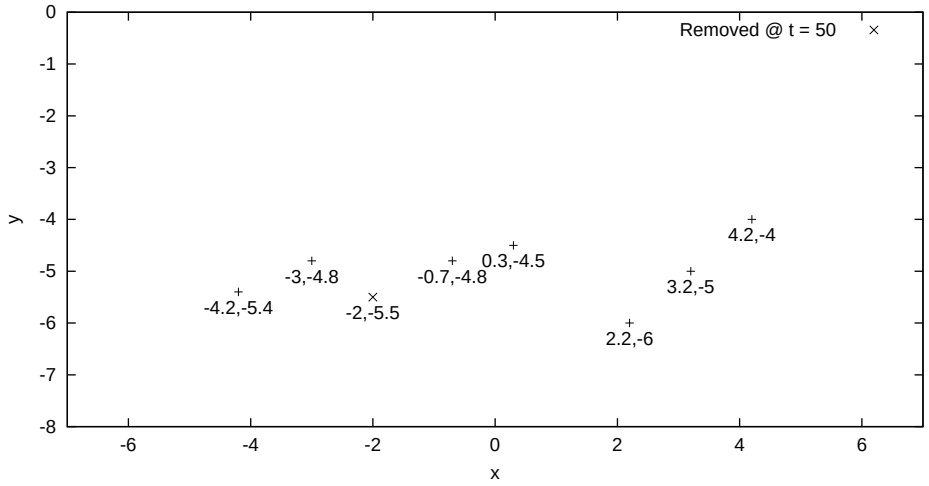


Figure 3.14: Configuration 3D.

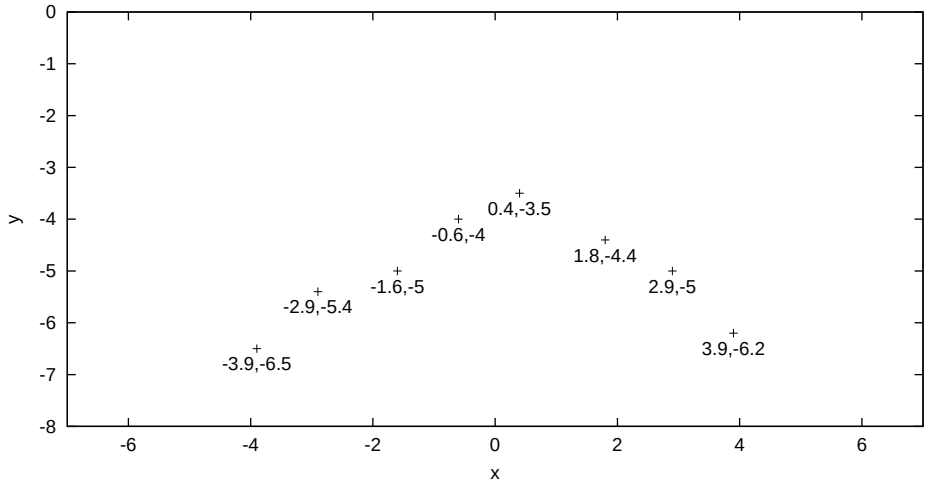


Figure 3.15: Configuration 3E.

### 3.3.2 Performance Analysis

The following performance measures were defined to evaluate the performance of the algorithms:

**Error ($e(t)$):** The deviation of the nodes from the desired geometric pattern.

**Rate of convergence ($r(t)$):** How quickly the desired geometric pattern is being restored.

**Efficiency ($q(t)$):** How efficiently the motion of the nodes is being coordinated.

**Settlement time ($t_s$):** The time taken for the array to become stable, ignoring small movements caused by sensor and actuator imperfections.

These measures are defined below.

#### Error

The *error* $e(t)$ is defined as the average distance of each node $i$, $i = 0, \ldots, N-1$, where $N$ is the number of nodes, from its corresponding *desired position* $(\hat{x}_i, \hat{y}_i)$ at a given time $t$. The *desired positions* are defined as the set of positions which are separated by $S$, have the same $y$-coordinate $\hat{y}$ and minimise the sum of the square of the distance $e_i$ of each point from the corresponding node's current position $(x_i, y_i)$. We use $S = \frac{L}{N-1}$ for Algorithm CL, where $L$ is the length of the array. Thus we wish to minimise

$$F_e = \sum_{i=0}^{N-1} e_i^2 = \sum_{i=0}^{N-1} \left[ (\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2 \right].$$

Note that $\hat{x}_1 = \hat{x}_0 + S$, $\hat{x}_2 = \hat{x}_1 + S$, and so on, since the desired positions are separated by $S$. Thus $\hat{x}_i = \hat{x}_0 + Si$ and

$$F_e = \sum_{i=0}^{N-1} (\hat{x}_0 + Si - x_i)^2 + \sum_{i=0}^{N-1} (\hat{y} - y_i)^2.$$

First, we determine the value of $\hat{x}_0$ that will minimise $F_e$ by taking the partial derivative with respect to $\hat{x}_0$:

$$\frac{\partial F_e}{\partial \hat{x}_0} = 2 \sum_{i=0}^{N-1} (\hat{x}_0 + Si - x_i)$$

$$\begin{aligned} &= 2\left[N\hat{x}_0 + S\sum_{i=0}^{N-1} i - \sum_{i=0}^{N-1} x_i\right] \\ &= 2\left[N\hat{x}_0 + S\frac{N(N-1)}{2} - \sum_{i=0}^{N-1} x_i\right] \\ &= 2N\left[\hat{x}_0 + S\frac{(N-1)}{2} - \frac{1}{N}\sum_{i=0}^{N-1} x_i\right]. \end{aligned}$$

Equating $\frac{\partial F_e}{\partial \hat{x}_0}$ to zero and solving for $\hat{x}_0$,

$$\begin{aligned} \hat{x}_0 &= \frac{1}{N}\sum_{i=0}^{N-1} x_i - S\frac{(N-1)}{2} \\ &= \frac{1}{N}\sum_{i=0}^{N-1} x_i - \frac{L}{2}. \end{aligned}$$

Note that $\frac{\partial^2 F_e}{\partial \hat{x}_0^2} = 2N > 0$, hence the solution above minimises $F_e$. Thus the $x$-coordinate of the desired position of node $i$ is given by

$$\hat{x}_i = \frac{1}{N}\sum_{i=0}^{N-1} x_i - \frac{L}{2} + Si.$$

Next, we take the partial derivative of $F_e$ with respect to $\hat{y}$:

$$\begin{aligned} \frac{\partial F_e}{\partial \hat{y}} &= 2\sum_{i=0}^{N-1} (\hat{y} - y_i) \\ &= 2\left[N\hat{y} - \sum_{i=0}^{N-1} y_i\right] \\ &= 2N\left[\hat{y} - \frac{1}{N}\sum_{i=0}^{N-1} y_i\right]. \end{aligned}$$

Equating $\frac{\partial F_e}{\partial \hat{y}}$ to zero and solving for $\hat{y}$,

$$\hat{y} = \frac{1}{N}\sum_{i=0}^{N-1} y_i.$$

$\frac{\partial^2 F_e}{\partial \hat{y}^2} = 2N$, so the above solution for $\hat{y}$ minimises $F_e$.

#### Rate of Convergence

The rate of convergence $r(t)$ is defined as

$$r(t) = \frac{e(0) - e(t)}{N}.$$

#### Efficiency

The *efficiency* of the network was measured by computing the average of the displacement divided by the distance travelled for each node. Thus the *efficiency* $q(t)$ is defined as

$$q(t) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{|\mathbf{p}_i(t) - \mathbf{p}_i(0)|}{\ell_i(t)}$$

where $\mathbf{p}_i(t)$ is the position of node $i$ at time $t$ and $\ell_i(t)$ is the total distance travelled by node $i$ since $t = 0$.

#### Settlement Time

To assist the comparison of Algorithm CS/SB with different values of $S_{\text{open}}$ and $S_{\text{break}}$ the above performance measures were evaluated at either $t = 100$, or the *settlement time* $t_{\text{s}}$, if it exists. The *settlement time* is defined as the smallest value of $t_{\text{s}}$ satisfying $|e(t_{\text{s}}) - e(t)| \leq \varepsilon$ for $t \geq t_{\text{s}}$ with $t_{\text{s}} \leq 100 - T$, where $T$ is the chosen minimum period over which the condition must be satisfied before $t_{\text{s}}$ can be defined for the simulation. Since the condition for $t_{\text{s}}$ is easy to satisfy just before the end of the simulation, when it can only be applied to a short period, $t_{\text{s}}$ is restricted to values less than or equal to $100 - T$. For these experiments, $\varepsilon = 0.15$ and $T = 30$.

### 3.3.3 Results

#### Simulation Traces

Traces of the path of the nodes from simulations are shown in Figures 3.16 to 3.22. The array begins at the bottom of the plot at time $t = 0$ and finishes at the top at time $t = 100$. The positions are shown for every five time units. For a given "snapshot" of the positions the nodes are joined by a solid line, while the path of an individual node is shown with a dashed line of the same thickness. The desired positions for each "snapshot" are indicated by crosses joined with a thin dashed line. No random error was included in these simulations. For Algorithm CS/SB, $S_{\text{open}} = 1.25$ and $S_{\text{break}} = 1.50$, because of all the values tested these parameter settings resulted in the most efficient

motion planning, as explained next.

**Algorithm CS/G** The nodes did not finish as close to their desired positions as expected using Configurations 3B and 3C. With Configuration 3B the error dropped from 0.952 at $t = 0$ to 0.043 at $t = 49$, jumped to 0.461 at $t = 50$ when the extra node was included and finished at 0.250. The final error was 0.219 for Configuration 3C. As the traces clearly show in Figures 3.17 and 3.18, the error in the $x$-coordinate of each individual node is accumulated by the nodes at the ends of the array. While random errors were not included in these simulations, the finite resolution of the cameras and the discrete time modelling of the system were still a source of error.

**Algorithm CS/SB** Note how the motion planning of Algorithm CS/SB is more rigid than that of Algorithm CS/G. The paths of nodes A, B and C, labelled in Figures 3.16 and 3.19, differ noticeably for Algorithms CS/G and CS/SB. With Algorithm CS/G nodes A and C initially move away from B, and then move towards B at $t = 12$, whereas with Algorithm CS/SB nodes A and C do not move in the $x$-direction until $t = 29$ and $t = 23$, respectively. When Algorithm CS/SB is used, movement in nodes A and C is triggered when the opening shared by the more distant neighbour is eliminated. Thus the motion planning of Algorithm CS/SB was more efficient, as expected.

**Algorithm CL** For Configuration 3E, nodes D and E began close to their desired positions and there was little change in the relative positions of these nodes, demonstrating the efficiency of the motion coordination along the $y$-axis. Also, the other nodes adjusted their relative $y$-velocity so that they became aligned with nodes D and E.

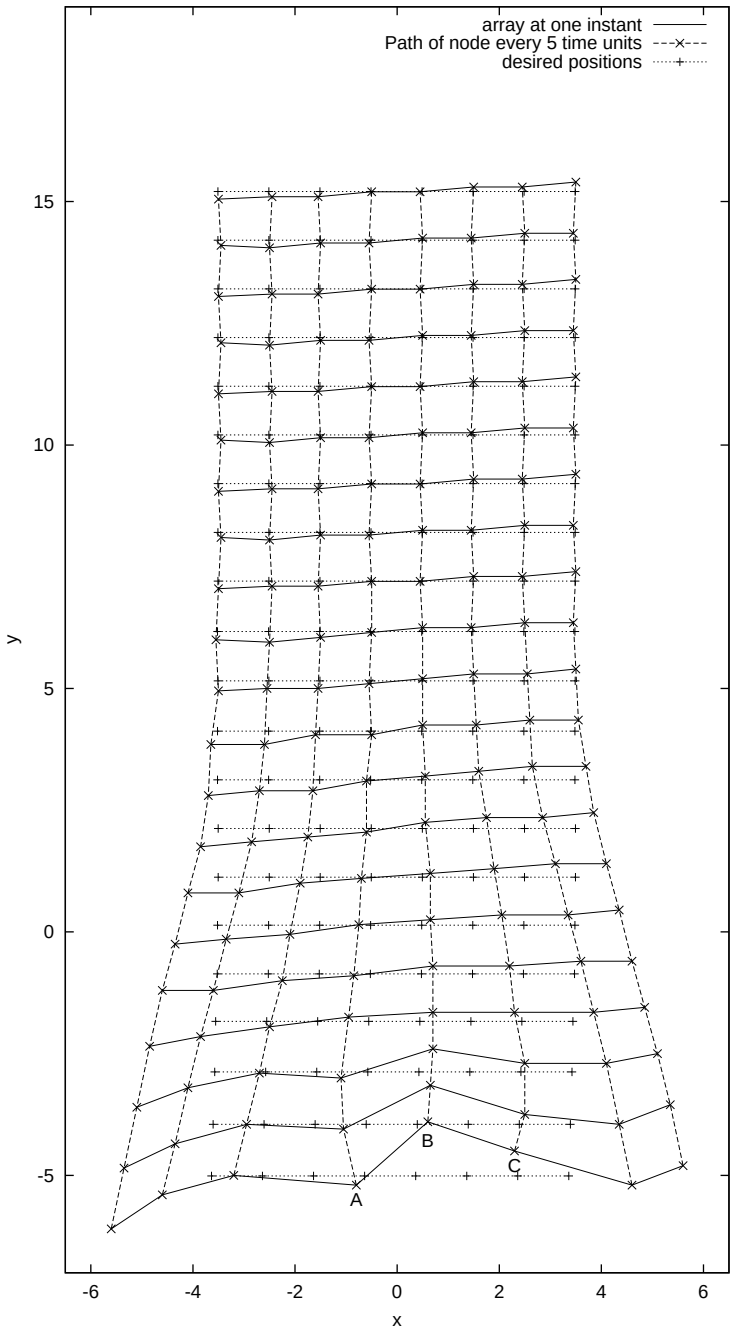


Figure 3.16: Simulation trace of Algorithm CS/G with Configuration 3A.

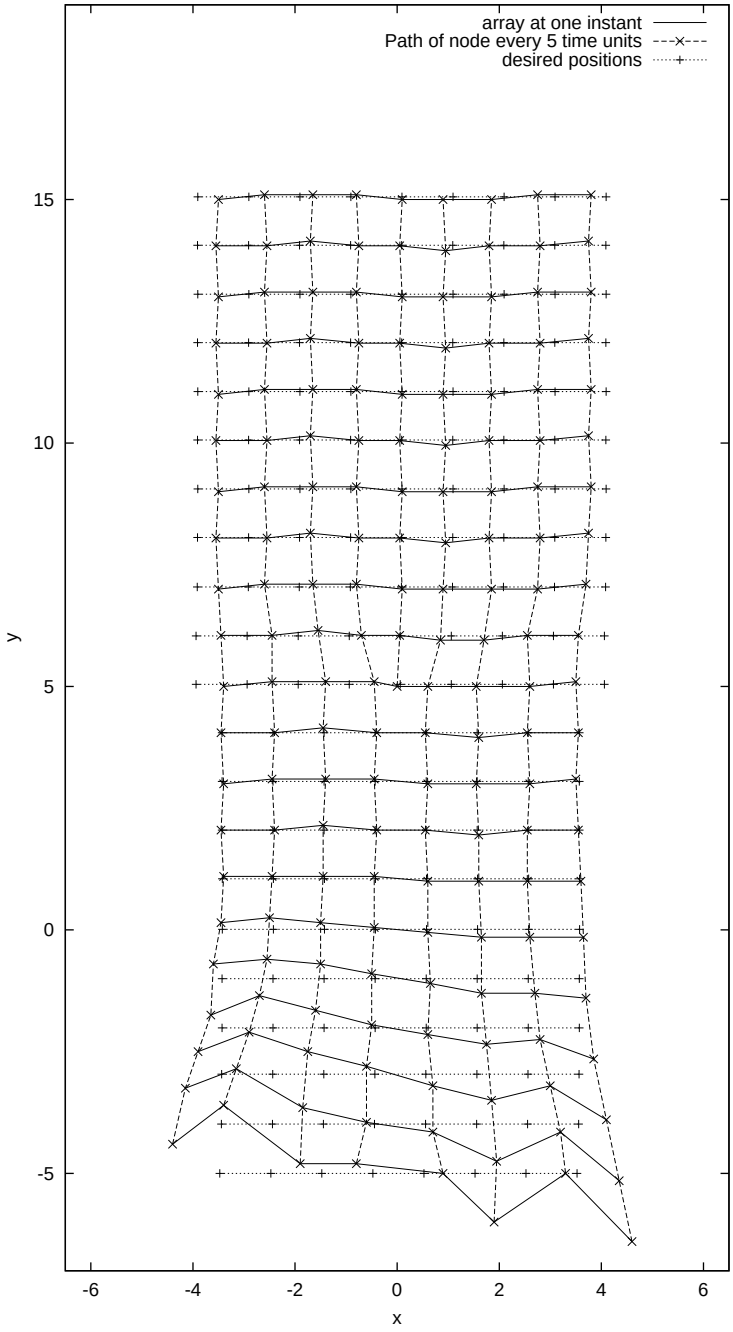


Figure 3.17: Simulation trace of Algorithm CS/G with Configuration 3B.

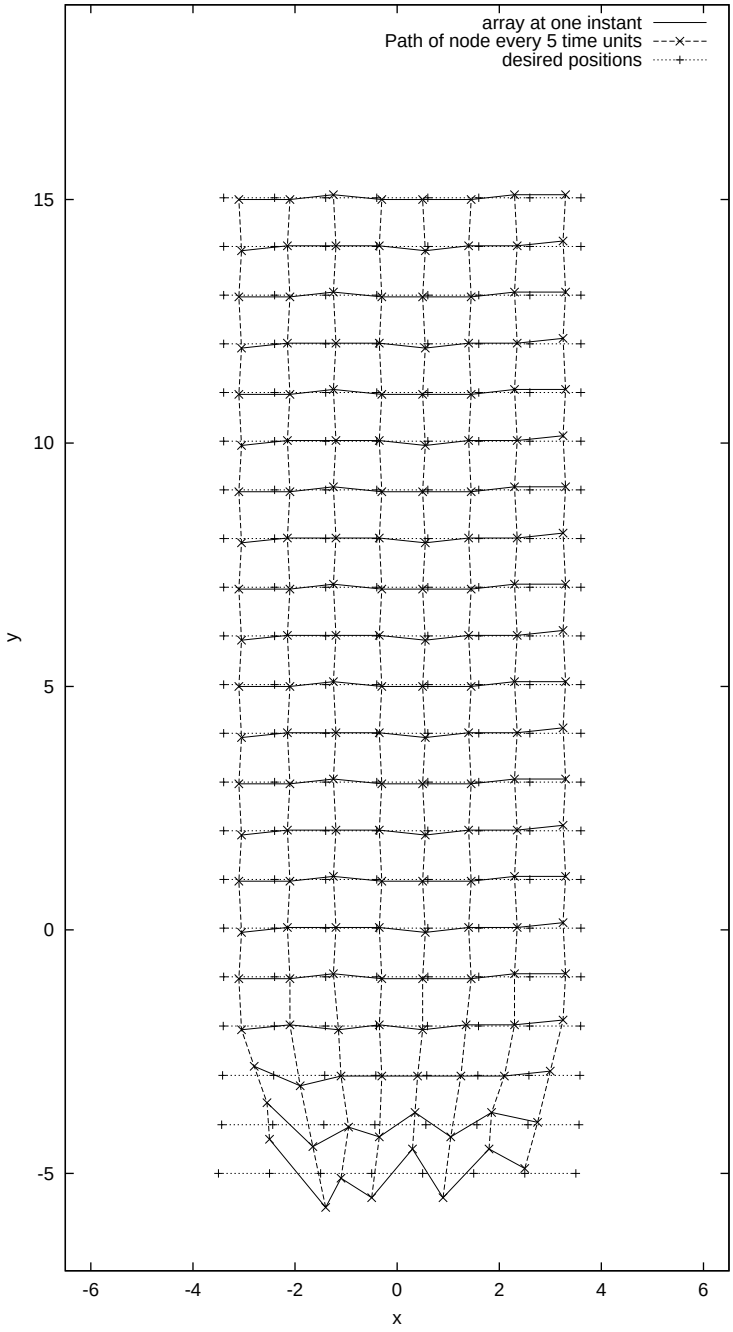


Figure 3.18: Simulation trace of Algorithm CS/G with Configuration 3C.

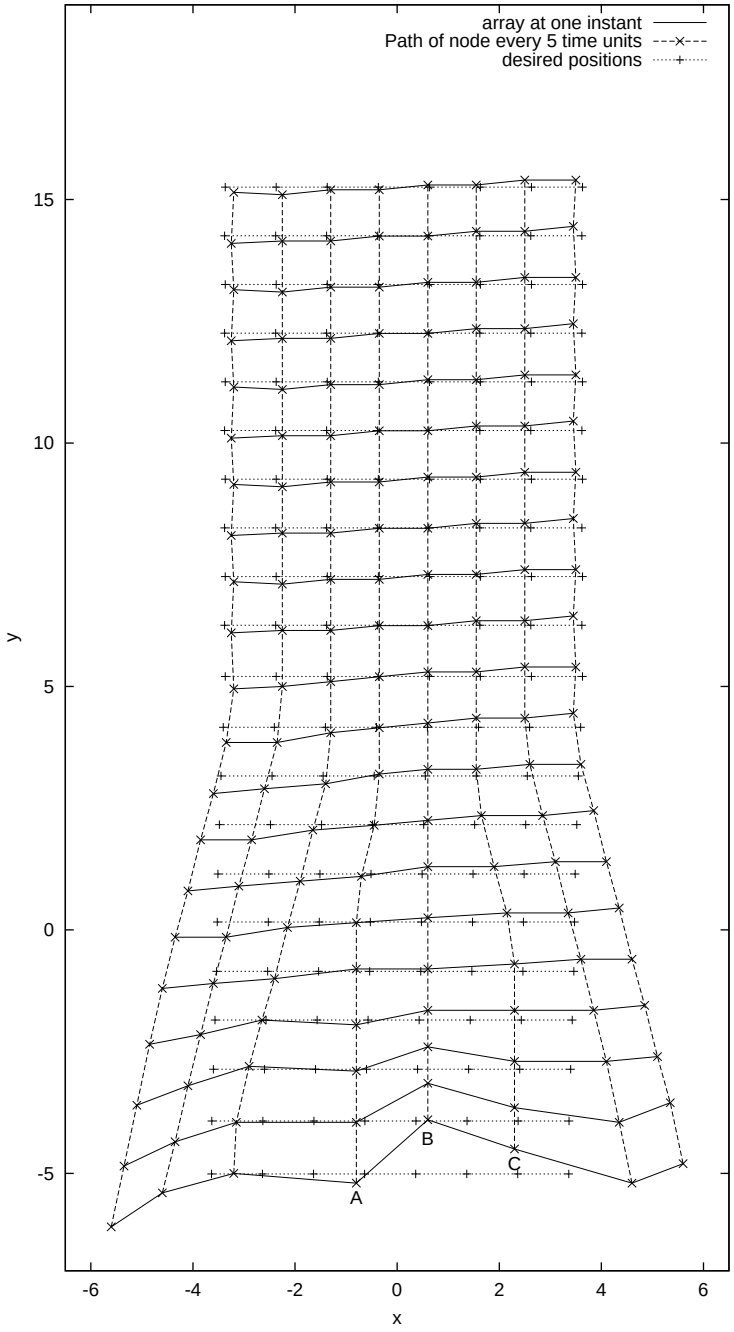


Figure 3.19: Simulation trace of Algorithm CS/SB with Configuration 3A.

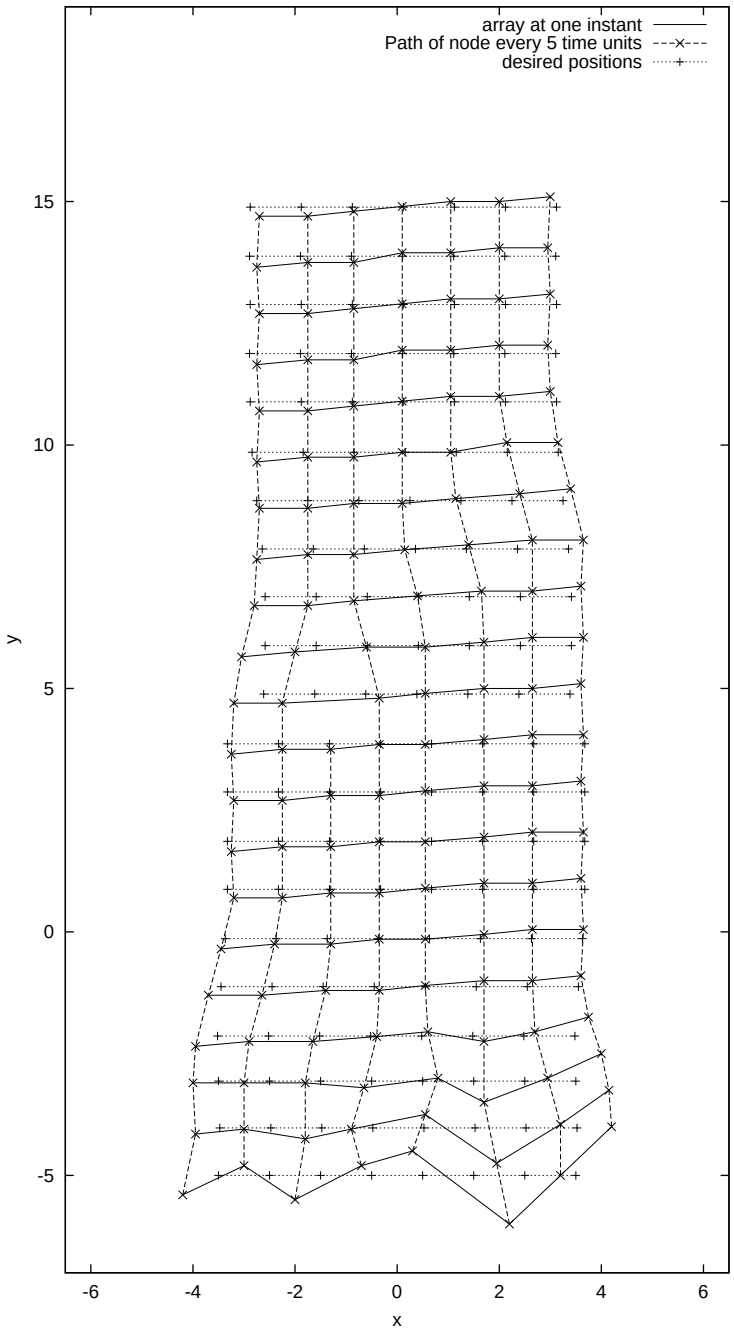


Figure 3.20: Simulation trace of Algorithm CS/SB with Configuration 3D.

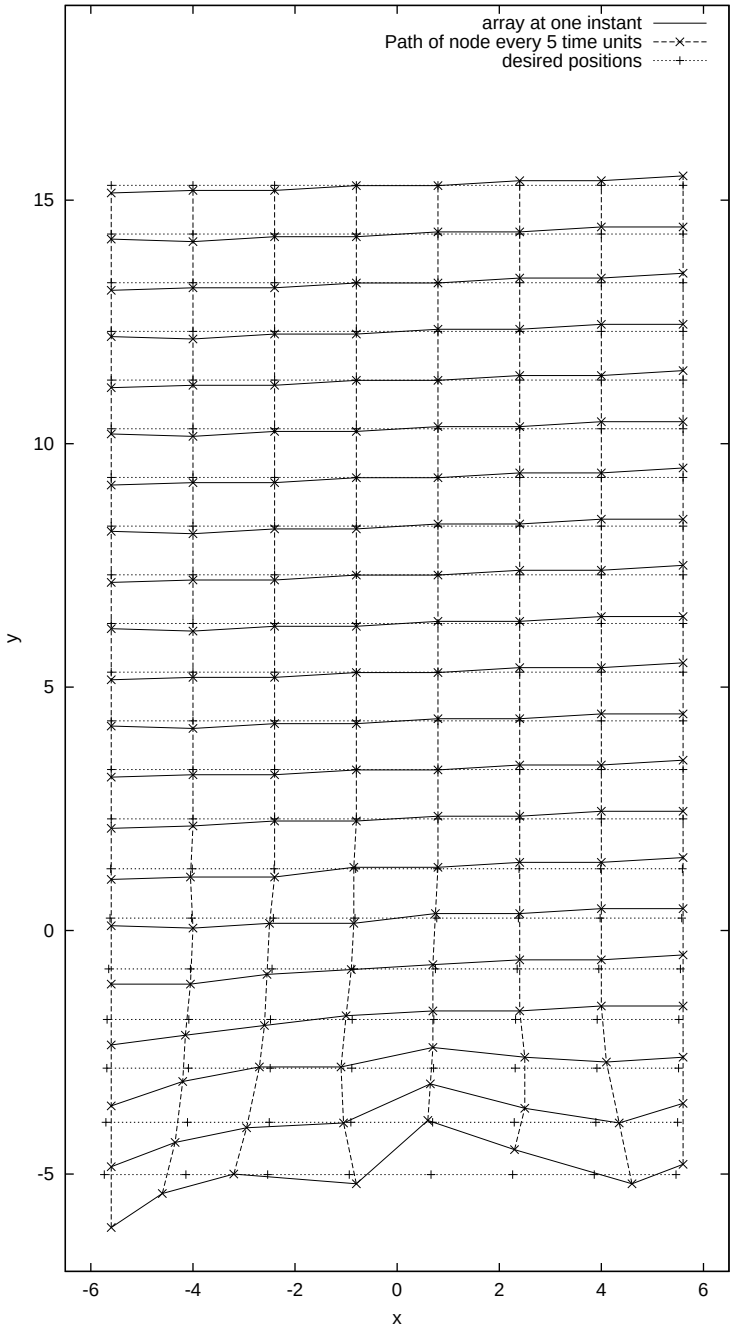


Figure 3.21: Simulation trace of Algorithm CL with Configuration 3A.

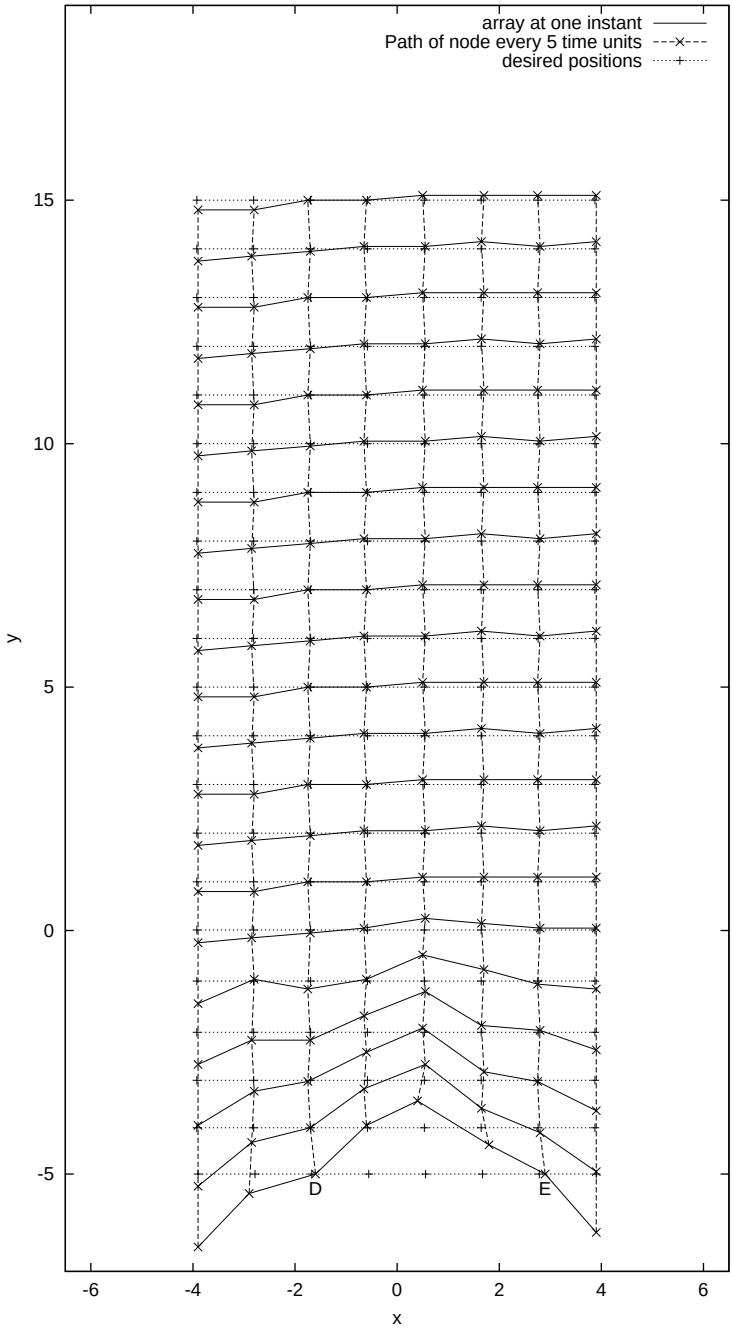


Figure 3.22: Simulation trace of Algorithm CL with Configuration 3E.

**Effect of Values for $S_{\text{open}}$ and $S_{\text{break}}$ on the Performance of Algorithm CS/SB**

To investigate how $S_{\text{open}}$ and $S_{\text{break}}$ affect the performance of Algorithm CS/SB, a series of simulations were executed with different values of $S_{\text{open}}$ and $S_{\text{break}}$ using Configuration 3A. $S_{\text{open}}$ was first set to $S = 1$ and then increased to 2.25 in increments of 0.25. For each value of $S_{\text{open}}$, $S_{\text{break}}$ was first set to $S_{\text{open}}$ and then increased to 2.75 in increments of 0.25. This sequence of simulations was carried out under perfect conditions where the error parameters are set to zero, and then repeated under imperfect conditions, using $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

**Performance Under Perfect Conditions** Figures 3.23, 3.25, 3.27, 3.29, 3.31 and 3.33 show plots of the error for each value of $S_{\text{open}}$ and $S_{\text{break}}$ tested. The efficiency is plotted for each simulation in Figures 3.24, 3.26, 3.28, 3.30, 3.32 and 3.34. The settlement times are given in Table 3.1. The cases where the settlement time was not reached before the end of the simulation are marked with a dash. Tables 3.2, 3.3 and 3.4 show the error, rate of convergence and efficiency calculated at either settlement time for the cases where the settlement time was reached or $t = 100$ for the cases where the settlement time was not reached. The entries in which the performance measure is calculated at $t = 100$ are indicated with an asterisk.

The settlement time was not reached for $S_{\text{open}} = S_{\text{break}} = S = 1$, showing that even under perfect conditions $S_{\text{break}}$ should be greater than $S$ to avoid a major loss in performance. $\Delta S_{\text{open}}$ also should not be too high, as the rate of convergence and efficiency greatly dropped for $S_{\text{open}} \geq 2$.

The algorithm performed best with $S_{\text{open}} = 1.25$. For this value of $S_{\text{open}}$ the setting of $S_{\text{break}}$ did not affect performance. While $S_{\text{open}} = S_{\text{break}} = 1.5$ produced a similar rate of convergence and efficiency, the error at settlement time was noticeably higher for these parameter settings.

As $S_{\text{open}}$ was increased above 1.25 the rate of convergence and efficiency for $S_{\text{break}} = S_{\text{open}}$ remained the same or decreased. Likewise, for a given value of $S_{\text{open}}$, the rate of convergence and efficiency remained the same or decreased as $S_{\text{break}}$ was increased. Therefore, there is no evidence that using a non-zero value of $\Delta S_{\text{break}}$ improves the rate of convergence or efficiency under perfect conditions except when $S_{\text{open}} = S$. Note that for $S_{\text{open}} = 2$ and $S_{\text{open}} = 2.25$ the final error actually decreased when $S_{\text{break}}$ was increased to 2.50 and then 2.75, and the plots of $e(t)$ for these settings show that the error would probably continue to decrease unlike the plots for lower values of $S_{\text{break}}$.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | - | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 1.25 | | 36 | 36 | 36 | 36 | 36 | 36 | 36 |
| 1.50 | | | 35 | 36 | 53 | 53 | 51 | 51 |
| 1.75 | | | | 51 | 51 | 51 | 69 | 69 |
| 2.00 | | | | | 50 | 50 | - | - |
| 2.25 | | | | | | 50 | - | - |

Table 3.1: Settlement time.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.425* | 0.291 | 0.291 | 0.291 | 0.291 | 0.291 | 0.291 | 0.291 |
| 1.25 | | 0.279 | 0.279 | 0.279 | 0.279 | 0.279 | 0.279 | 0.279 |
| 1.50 | | | 0.334 | 0.326 | 0.305 | 0.305 | 0.323 | 0.323 |
| 1.75 | | | | 0.278 | 0.272 | 0.272 | 0.433 | 0.433 |
| 2.00 | | | | | 0.628 | 0.628 | 0.348* | 0.348* |
| 2.25 | | | | | | 0.628 | 0.348* | 0.348* |

Table 3.2: Error at settlement time or $t = 100$ when the settlement time was not defined.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.012* | 0.033 | 0.033 | 0.033 | 0.033 | 0.033 | 0.033 | 0.033 |
| 1.25 | | 0.037 | 0.037 | 0.037 | 0.037 | 0.037 | 0.037 | 0.037 |
| 1.50 | | | 0.036 | 0.035 | 0.024 | 0.024 | 0.025 | 0.025 |
| 1.75 | | | | 0.026 | 0.026 | 0.026 | 0.017 | 0.017 |
| 2.00 | | | | | 0.019 | 0.019 | 0.012* | 0.012* |
| 2.25 | | | | | | 0.019 | 0.012* | 0.012* |

Table 3.3: Rate of convergence at settlement time or $t = 100$ when the settlement time was not defined.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.207* | 0.524 | 0.524 | 0.524 | 0.524 | 0.524 | 0.524 | 0.524 |
| 1.25 | | 0.587 | 0.587 | 0.587 | 0.587 | 0.587 | 0.587 | 0.587 |
| 1.50 | | | 0.586 | 0.571 | 0.469 | 0.469 | 0.410 | 0.410 |
| 1.75 | | | | 0.463 | 0.449 | 0.449 | 0.304 | 0.304 |
| 2.00 | | | | | 0.340 | 0.340 | 0.267* | 0.267* |
| 2.25 | | | | | | 0.340 | 0.267* | 0.267* |

Table 3.4: Efficiency at settlement time or $t = 100$ when the settlement time was not defined.

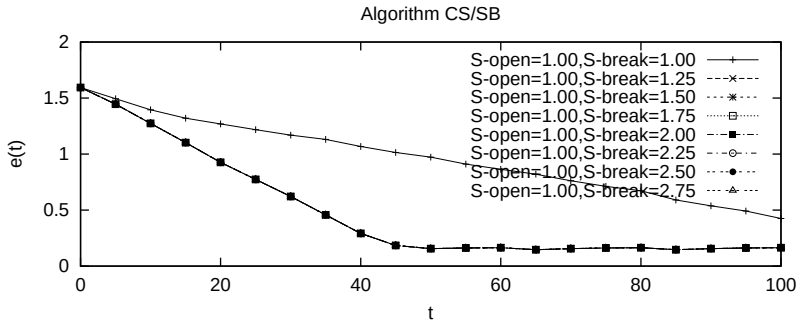


Figure 3.23: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

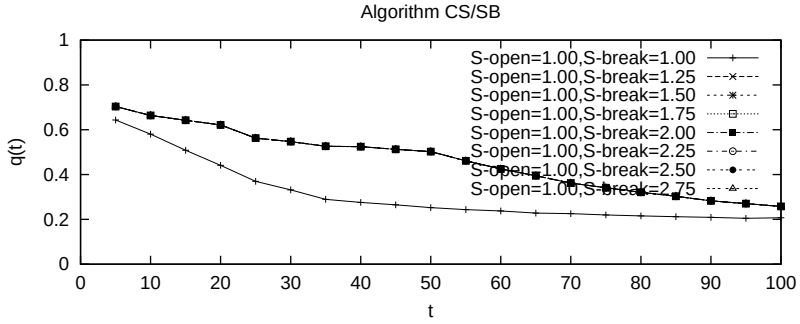


Figure 3.24: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

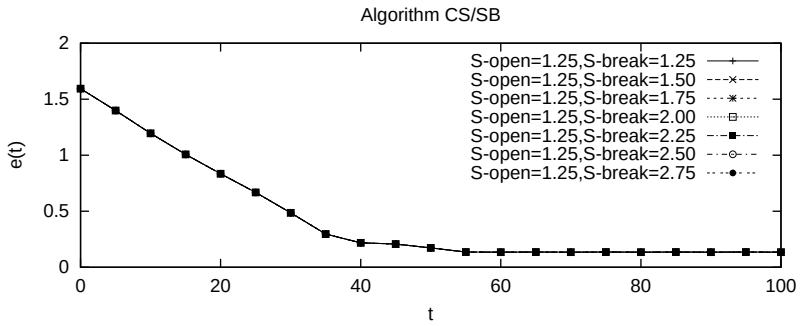


Figure 3.25: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

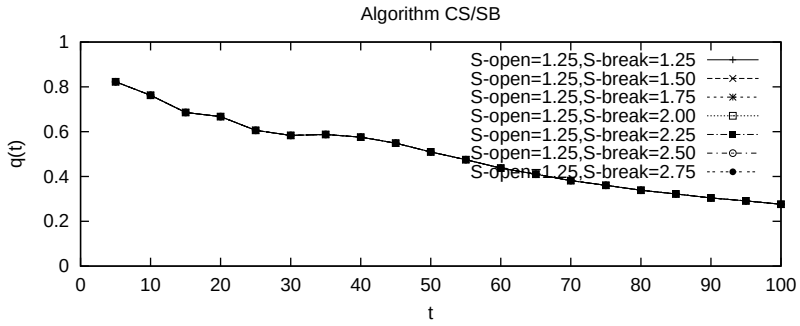


Figure 3.26: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

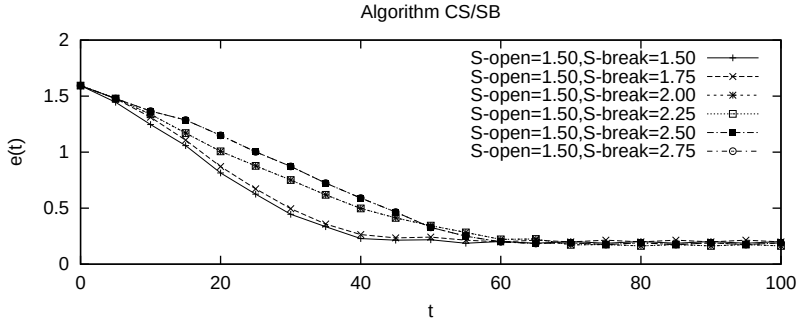


Figure 3.27: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.5$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

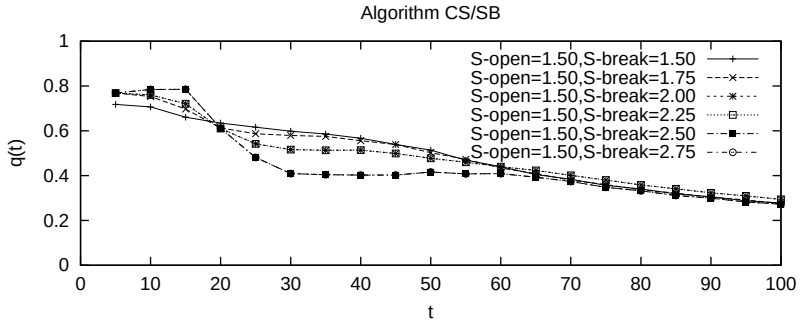


Figure 3.28: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.5$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

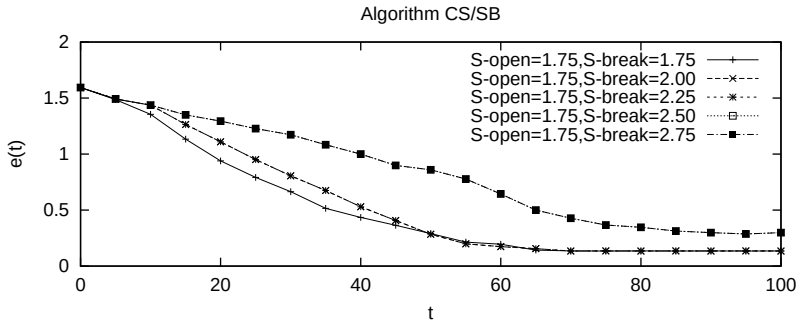


Figure 3.29: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.75$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

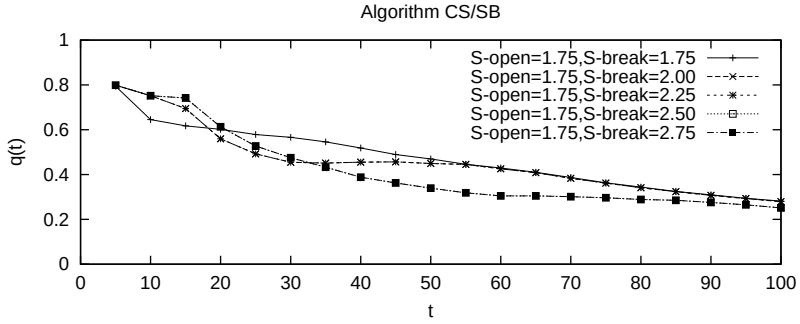


Figure 3.30: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.75$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

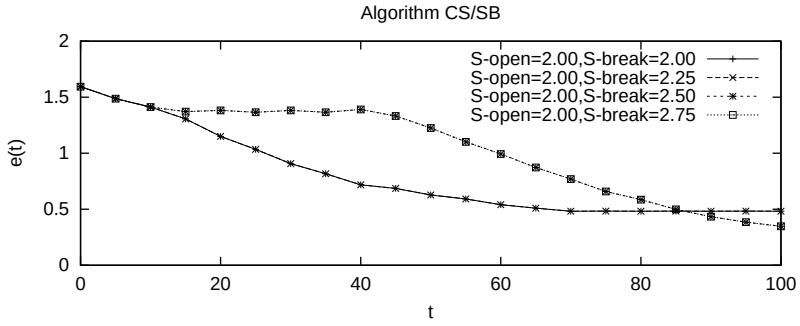


Figure 3.31: Error over time for Algorithm CS/SB with $S_{\text{open}} = 2$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

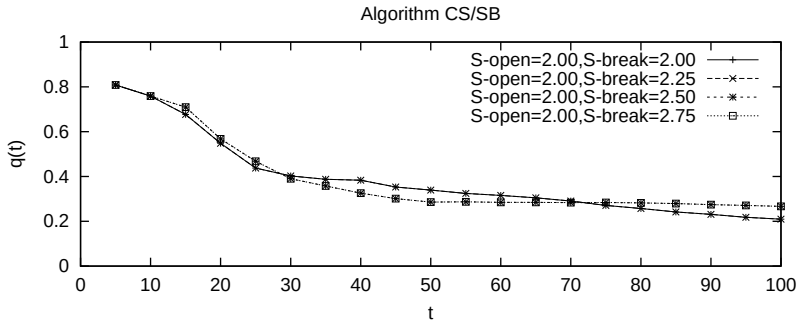


Figure 3.32: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 2$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

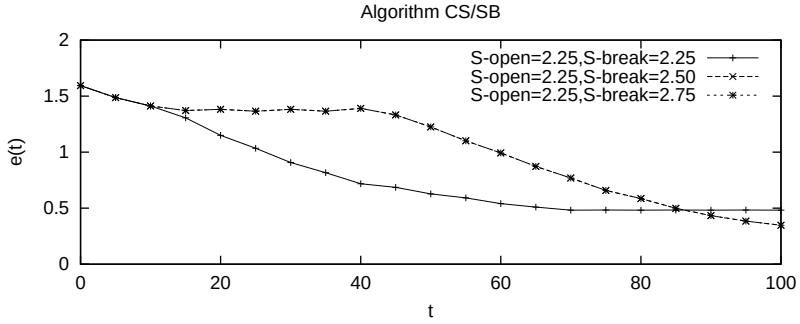


Figure 3.33: Error over time for Algorithm CS/SB with $S_{\text{open}} = 2.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

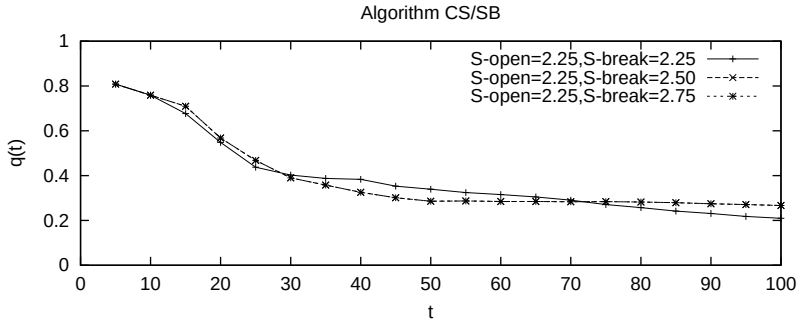


Figure 3.34: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 2.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0°$ and $\sigma_{E_{V_x,V_y}} = 0$.

**Performance Under Imperfect Conditions** Figures 3.35, 3.37, 3.39, 3.41, 3.43 and 3.45 show plots of the error for each value of $S_{\text{open}}$ and $S_{\text{break}}$ tested with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$. The efficiency is plotted for each simulation carried out under these conditions in Figures 3.36, 3.38, 3.40, 3.42, 3.44 and 3.46. The settlement times are given in Table 3.5. Tables 3.2, 3.3 and 3.4 show the error, rate of convergence and efficiency calculated at either settlement time for the cases where the settlement time was reached or $t = 100$ for the cases where the settlement time was not reached.

Unlike the simulations under perfect conditions, using $\Delta S_{\text{break}} = 0$ did not always result in the highest rate of convergence and efficiency for a given value of $S_{\text{open}}$. Under the imperfect conditions, the greatest rate of convergence and efficiency were achieved using $S_{\text{open}} = 1.25$ and $S_{\text{break}} = 1.50$. $S_{\text{open}} = 1.25$ and $S_{\text{break}} = 2.75$ produced the second highest rate of convergence and efficiency. For $S_{\text{open}} = 1$ and $S_{\text{open}} = 2$, the best performance was obtained with $S_{\text{break}} = 2.25$ and $S_{\text{break}} = 2.75$, respectively. For $S_{\text{open}} = 1.5$, the error and rate of convergence were highest using $S_{\text{break}} = 2.25$, and while the efficiency was highest with $S_{\text{break}} = S_{\text{open}} = 1.5$, the error was 0.250, which is significantly higher than that for $S_{\text{open}} = 1.25$ and $S_{\text{break}} = 1.5$, which was 0.204. Therefore, it can be concluded that $S_{\text{break}}$ should be greater than $S_{\text{open}}$ for robustness.

Since $S_{\text{open}} = 1.25$ and $S_{\text{break}} = 1.5$ produced the highest rate of convergence and efficiency and the error for these settings was 0.204, which is reasonably close to 0.179, the lowest of all the error values, these settings were used for all the simulations of Algorithm CS/SB that follow.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | - | 55 | 48 | 56 | 56 | 46 | 45 | 48 |
| 1.25 | | 42 | 40 | 44 | 50 | 44 | 61 | 41 |
| 1.50 | | | 47 | - | 52 | 42 | 67 | - |
| 1.75 | | | | 42 | 44 | 51 | 55 | 42 |
| 2.00 | | | | | 69 | 70 | - | 33 |
| 2.25 | | | | | | 46 | - | - |

Table 3.5: Settlement time with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.596* | 0.203 | 0.193 | 0.183 | 0.203 | 0.179 | 0.221 | 0.187 |
| 1.25 | | 0.248 | 0.204 | 0.213 | 0.238 | 0.192 | 0.210 | 0.234 |
| 1.50 | | | 0.250 | 0.218* | 0.430 | 0.216 | 0.344 | 0.133* |
| 1.75 | | | | 0.291 | 0.407 | 0.246 | 0.332 | 0.465 |
| 2.00 | | | | | 0.970 | 0.393 | 0.593* | 0.868 |
| 2.25 | | | | | | 0.580 | 0.626* | 0.960* |

Table 3.6: Error at settlement time or $t = 100$ when the settlement time was not defined with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.010* | 0.025 | 0.029 | 0.025 | 0.025 | 0.031 | 0.031 | 0.029 |
| 1.25 | | 0.032 | 0.035 | 0.031 | 0.027 | 0.032 | 0.023 | 0.033 |
| 1.50 | | | 0.029 | 0.014* | 0.022 | 0.033 | 0.019 | 0.015* |
| 1.75 | | | | 0.031 | 0.027 | 0.026 | 0.023 | 0.027 |
| 2.00 | | | | | 0.009 | 0.017 | 0.010* | 0.022 |
| 2.25 | | | | | | 0.022 | 0.010* | 0.006* |

Table 3.7: Rate of convergence at settlement time or $t = 100$ when the settlement time was not defined with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

| | $S_{\text{break}}$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $S_{\text{open}}$ | 1.00 | 1.25 | 1.50 | 1.75 | 2.00 | 2.25 | 2.50 | 2.75 |
| 1.00 | 0.163* | 0.369 | 0.436 | 0.372 | 0.366 | 0.457 | 0.446 | 0.428 |
| 1.25 | | 0.471 | 0.491 | 0.484 | 0.394 | 0.467 | 0.330 | 0.485 |
| 1.50 | | | 0.464 | 0.203* | 0.319 | 0.459 | 0.279 | 0.220* |
| 1.75 | | | | 0.453 | 0.412 | 0.407 | 0.353 | 0.393 |
| 2.00 | | | | | 0.177 | 0.273 | 0.216* | 0.361 |
| 2.25 | | | | | | 0.376 | 0.169* | 0.125* |

Table 3.8: Efficiency at settlement time or $t = 100$ when the settlement time was not defined with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

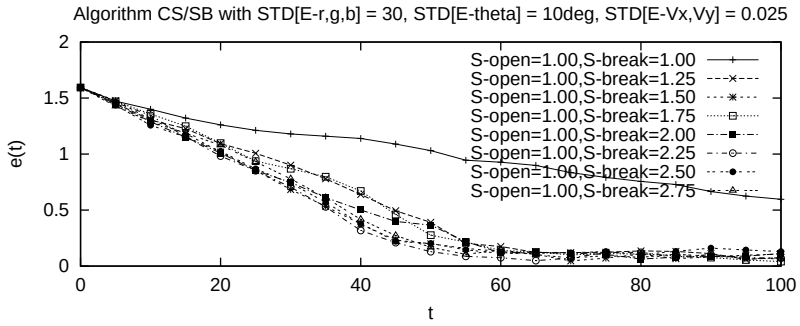


Figure 3.35: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

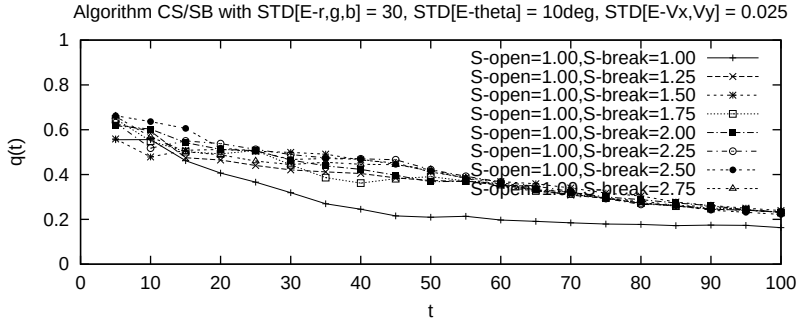


Figure 3.36: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

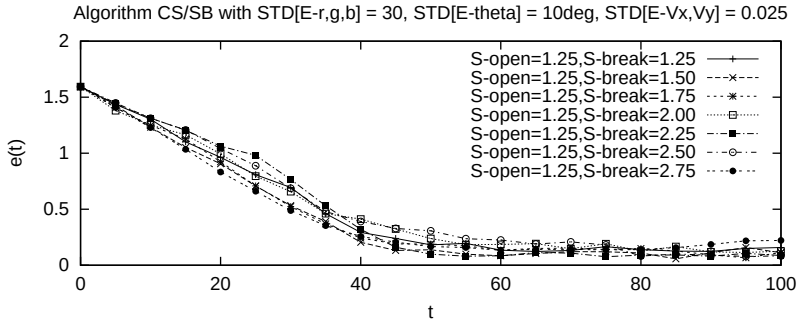


Figure 3.37: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

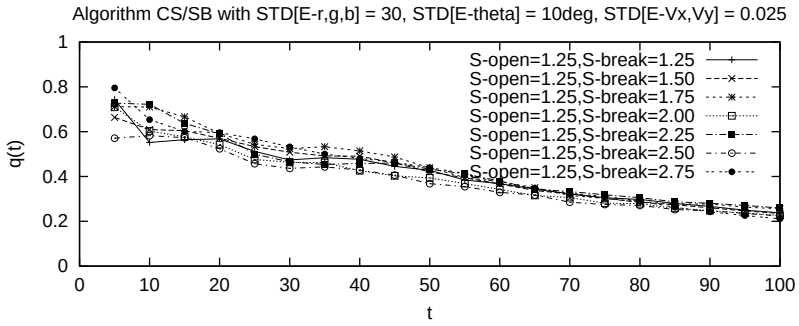


Figure 3.38: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

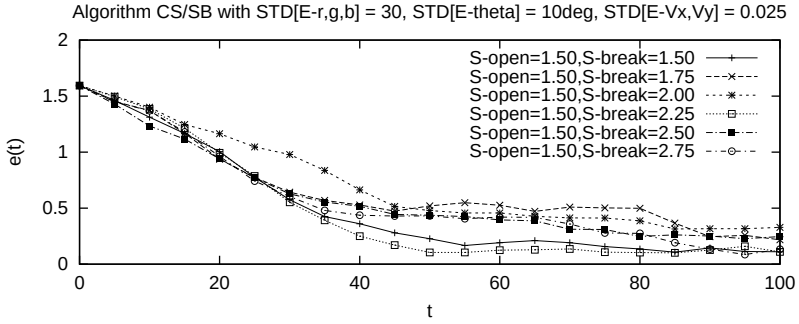


Figure 3.39: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.50$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

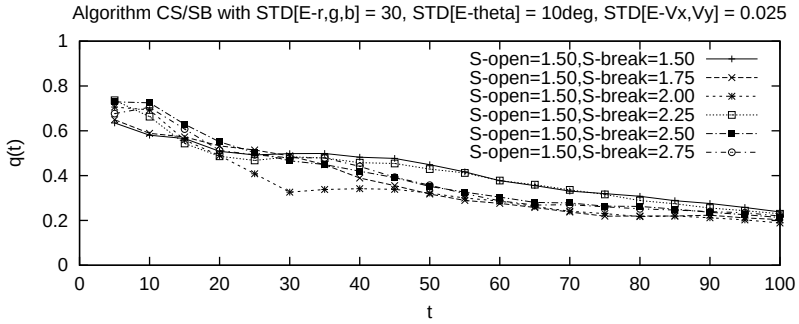


Figure 3.40: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.50$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

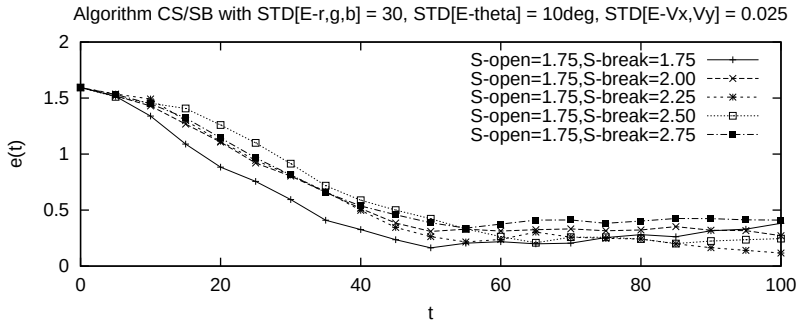


Figure 3.41: Error over time for Algorithm CS/SB with $S_{\text{open}} = 1.75$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

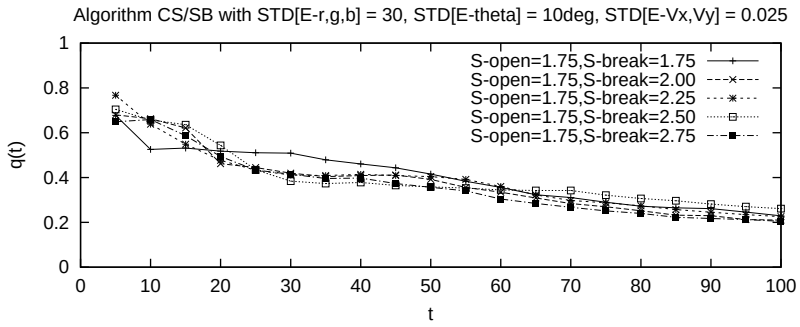


Figure 3.42: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 1.75$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

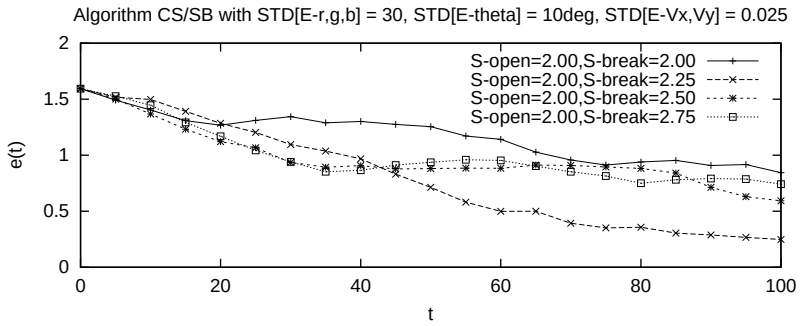


Figure 3.43: Error over time for Algorithm CS/SB with $S_{\text{open}} = 2.00$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

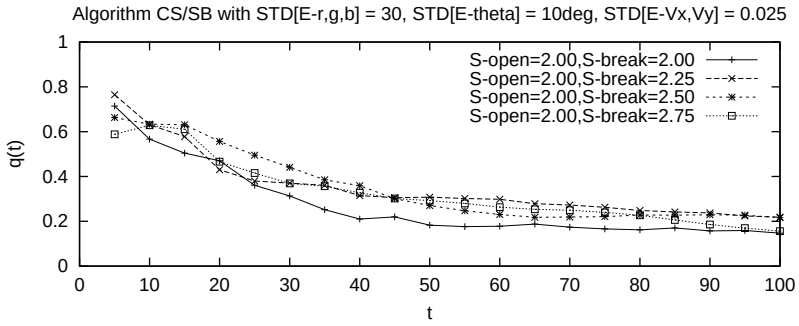


Figure 3.44: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 2.00$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

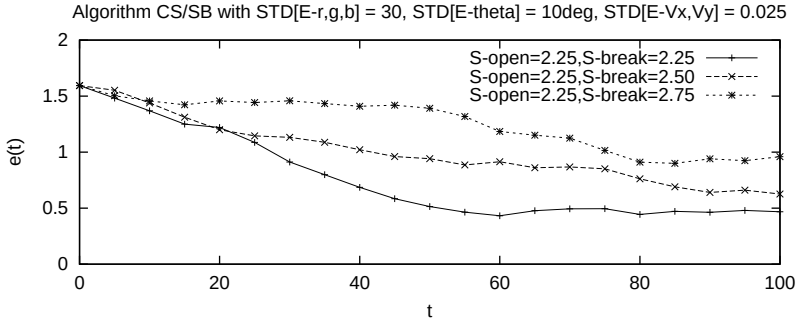


Figure 3.45: Error over time for Algorithm CS/SB with $S_{\text{open}} = 2.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

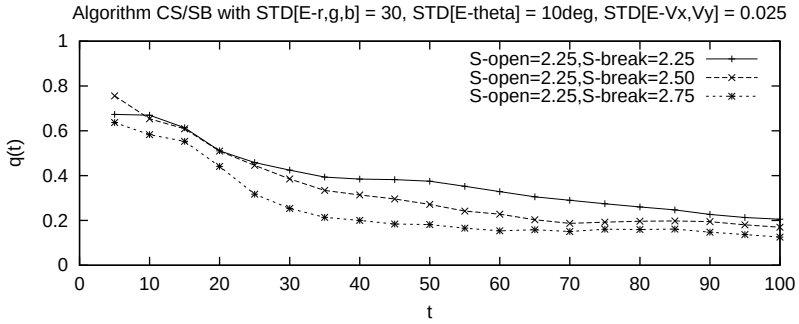


Figure 3.46: Efficiency over time for Algorithm CS/SB with $S_{\text{open}} = 2.25$, various values of $S_{\text{break}}$, $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

### Robustness Against Image Noise

To investigate the robustness of the algorithms against image noise, the algorithms were tested with $\sigma_{E_{r,g,b}} = 30$, 40 and 50 while $\sigma_{E_\theta}$ and $\sigma_{E_{V_x,V_y}}$ are zero using Configuration 3A. The error for these simulations is plotted in Figures 3.47, 3.49 and 3.51 and the efficiency in Figures 3.48, 3.50 and 3.52.

Algorithm CS/SB was slightly more efficient when $\sigma_{E_{r,g,b}} = 30$, however for $\sigma_{E_{r,g,b}} = 40$ the rate of convergence at $t = 100$ was 0.0153 with Algorithm CS/G, whereas for Algorithm CS/SB it was 0.0081 at $t = t_s = 35$. For $\sigma_{E_{r,g,b}} = 50$, both of these algorithms failed to perform as the error varied little from the initial value throughout these simulations.

### Robustness Against Error in Orientation of Cameras

To investigate their robustness against error in the orientation of the cameras, three simulations were executed for each algorithm with $\sigma_{E_\theta} = 30°$, 60° and 90° while $\sigma_{E_{r,g,b}}$ and $\sigma_{E_{V_x,V_y}}$ are zero using Configuration 3A. The error for each algorithm at each setting of $\sigma_{E_\theta}$ is plotted in Figures 3.53, 3.55 and 3.57. The efficiency for each algorithm at each setting of $\sigma_{E_\theta}$ is plotted in Figures 3.54, 3.56 and 3.58.

For $\sigma_{E_\theta} = 30°$, the error of Algorithm CS/SB remained below that for Algorithm CS/G from about $t = 40$ to about $t = 80$, and the efficiency was higher from about $t = 25$ onwards. For $\sigma_{E_\theta} = 60°$, Algorithm CS/SB generally performed better than Algorithm CS/G as the error and efficiency were higher throughout the simulation. The error for Algorithm CS/SB was a little higher than that of Algorithm CS/G from about $t = 10$ onwards for $\sigma_{E_\theta} = 90°$, but generally there was little difference in efficiency for this setting of $\sigma_{E_\theta}$.

### Robustness Against Error in Velocity

Each algorithm was tested with $\sigma_{E_{V_x,V_y}} = 0.05$, 0.1 and 0.15 while $\sigma_{E_{r,g,b}}$ and $\sigma_{E_\theta}$ are zero using Configuration 3A. The error of each algorithm for each value of $\sigma_{E_{V_x,V_y}}$ is plotted in Figures 3.59, 3.61 and 3.63, and the efficiency in Figures 3.60, 3.62 and 3.64.

Algorithm CS/G had a lower error than that of Algorithm CS/SB throughout the simulation for $\sigma_{E_{V_x,V_y}} = 0.05$, except from about $t = 40$ to about $t = 45$. For this value of $\sigma_{E_{V_x,V_y}}$ Algorithm CS/G had a higher efficiency until about $t = 35$, after which the efficiency of Algorithm CS/SB was higher. For $\sigma_{E_{V_x,V_y}} = 0.1$, the error of Algorithm CS/SB was significantly higher than that of Algorithm CS/G for the entire

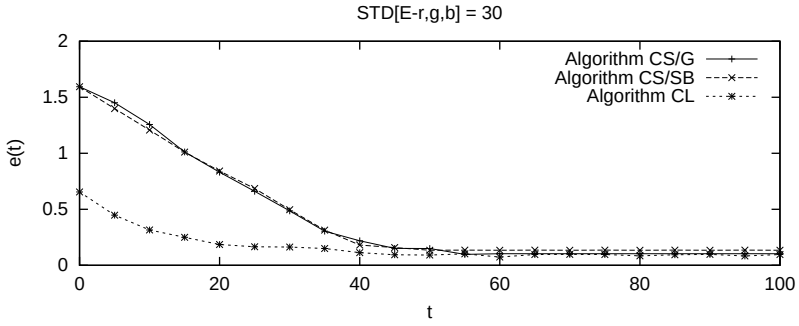


Figure 3.47: Error when $\sigma_{E_{r,g,b}} = 30$.

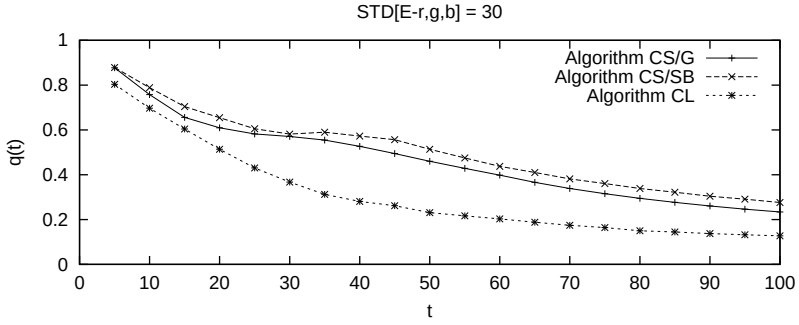


Figure 3.48: Efficiency when $\sigma_{E_{r,g,b}} = 30$.

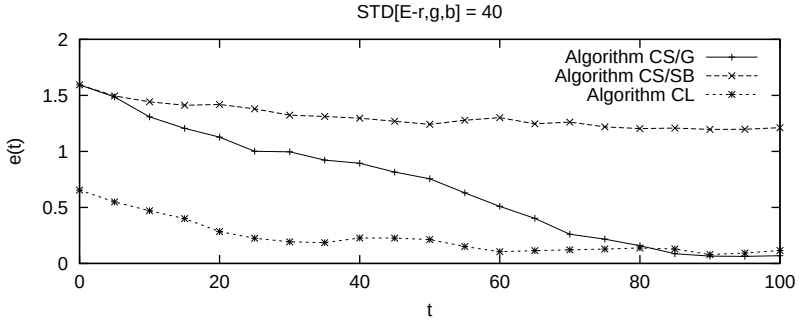


Figure 3.49: Error when $\sigma_{E_{r,g,b}} = 40$.

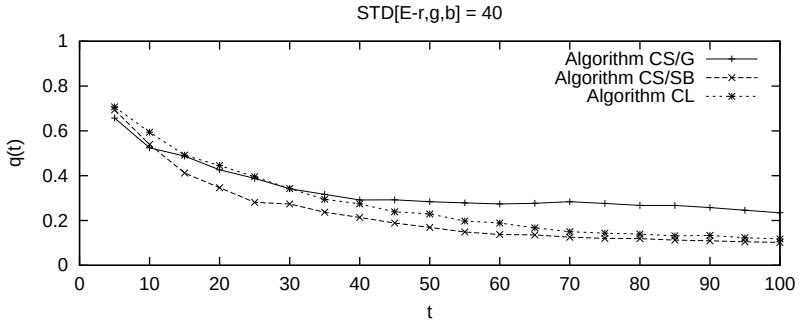


Figure 3.50: Efficiency when $\sigma_{E_{r,g,b}} = 40$.

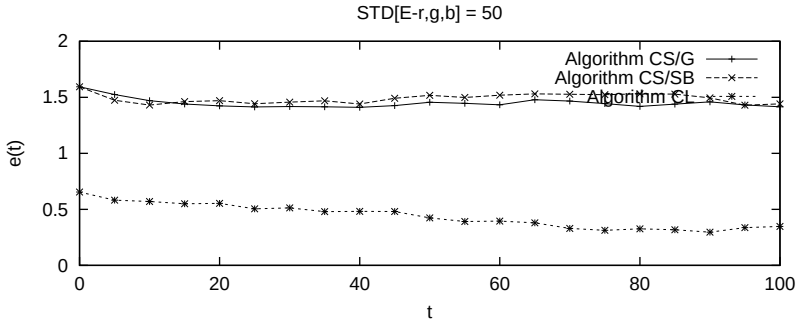


Figure 3.51: Error when $\sigma_{E_{r,g,b}} = 50$.

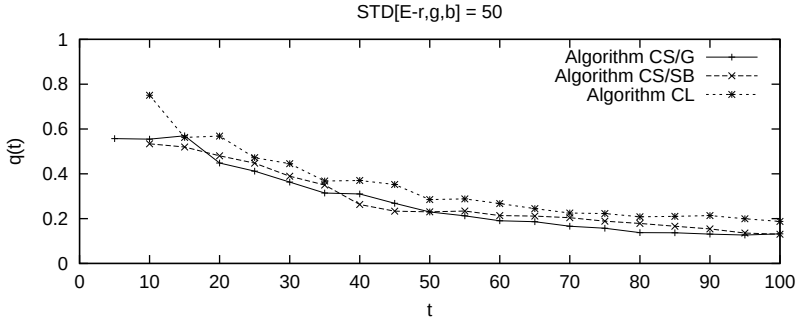


Figure 3.52: Efficiency when $\sigma_{E_{r,g,b}} = 50$.

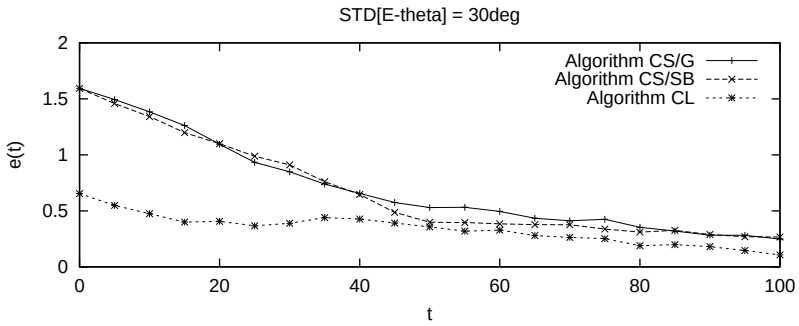


Figure 3.53: Error when $\sigma_{E_\theta} = 30°$

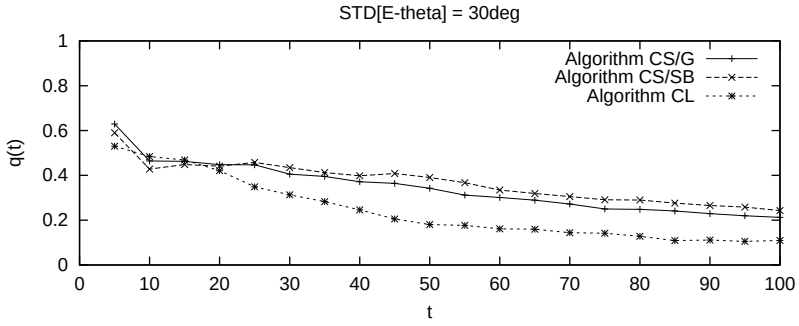


Figure 3.54: Efficiency when $\sigma_{E_\theta} = 30°$

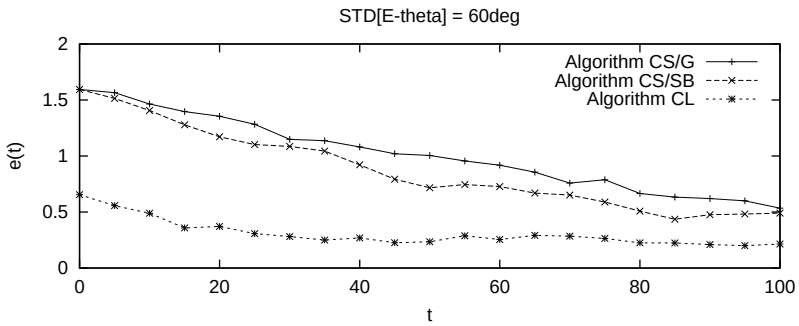


Figure 3.55: Error when $\sigma_{E_\theta} = 60°$.

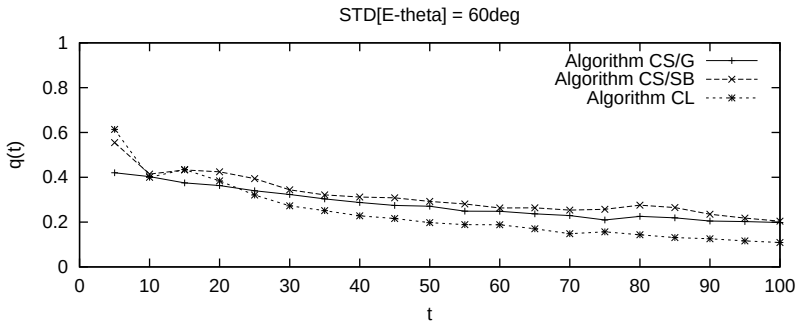


Figure 3.56: Efficiency when $\sigma_{E_\theta} = 60°$

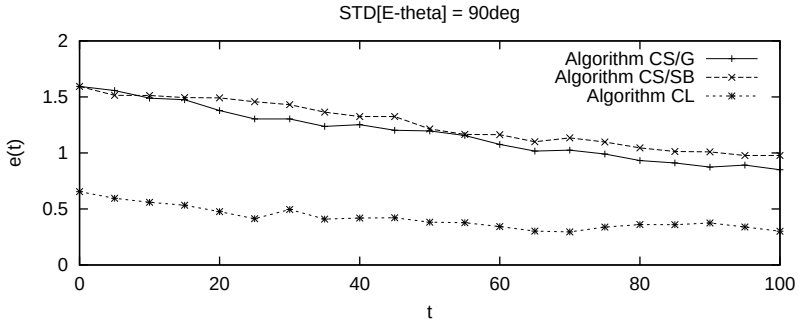


Figure 3.57: Error when $\sigma_{E_\theta} = 90°$.

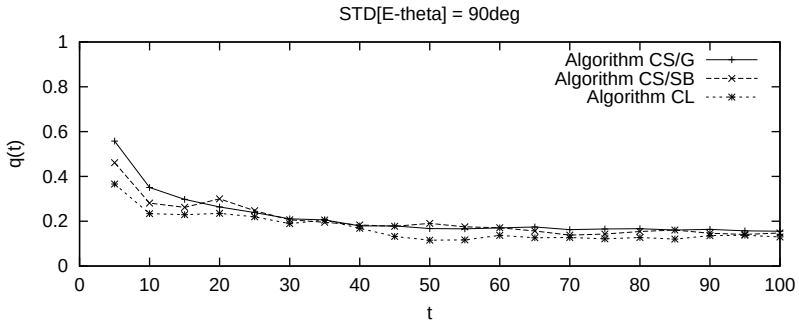


Figure 3.58: Efficiency when $\sigma_{E_\theta} = 90°$.

| Setting | $\sigma_{E_{r,g,b}}$ | $\sigma_{E_\theta}$ | $\sigma_{E_{V_x,V_y}}$ |
|---|---|---|---|
| A | 30 | 10° | 0.025 |
| B | 40 | 15° | 0.05 |
| C | 50 | 20° | 0.075 |

Table 3.9: Settings used to test the robustness of the algorithms when image noise, error in the orientation of the cameras and velocity error are present in the system.

simulation, although there was little noticeable difference in efficiency. For $\sigma_{E_{V_x,V_y}} = 0.15$, Algorithm CS/SB appeared to be more robust as the error was substantially lower than that of Algorithm CS/G from about $t = 25$ to about $t = 95$ . Also from about $t = 30$ onwards the efficiency of Algorithm CS/SB was higher.

For $\sigma_{E_{V_x,V_y}} = 0.075$, the nodes drifted from their desired positions with Algorithm CL but converged towards their desired positions when the other two algorithms were used. This can be explained from the fact that with Algorithm CL the inner nodes depend on the end nodes to position themselves correctly in the $x$-direction. In Algorithms CS/G and CS/SB, the end nodes use their neighbours as a local reference to position themselves in the $x$-direction, thus they tend to resist changes made to their relative positions by errors in the velocity. In Algorithm CL, the end nodes have no local reference, thus errors in their velocity permanently change their relative $x$-position and have an impact on the positions of all the nodes in the array.

### Robustness Against All Types of Error Combined

The performance of the algorithms when all three forms of error are present in the system was investigated. Each algorithm was tested under the settings shown in Table 3.9.

The error and efficiency for each algorithm under setting A are plotted in Figures 3.65 and 3.66, respectively. There is very little difference in the plots for Algorithm CS/G and Algorithm CS/SB with this setting. For setting B, the settlement time for Algorithm CS/G was 65, at which the rate of convergence was 0.0195, whereas Algorithm CS/SB had no defined settlement time and the rate of convergence at $t = 100$ was 0.0084, as illustrated in Figure 3.67. Under this setting Algorithm CS/G had a higher efficiency from about $t = 25$ onwards, as shown in Figure 3.68. For setting C the error of Algorithm CS/G was lower than that of Algorithm CS/SB, as shown in Figure 3.69, although there was little difference in efficiency, as the graph in Figure 3.70 shows.

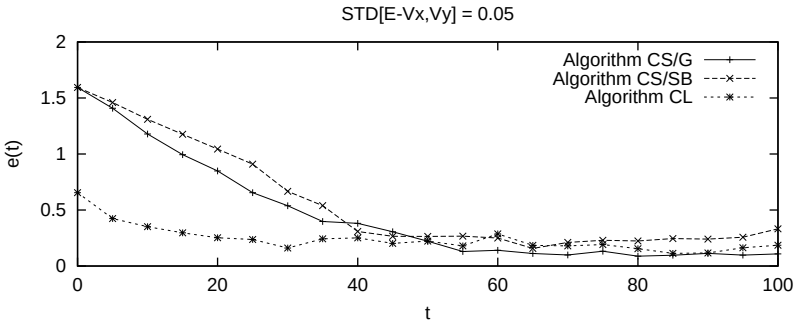


Figure 3.59: Error when $\sigma_{E_{V_x},V_y} = 0.05$.

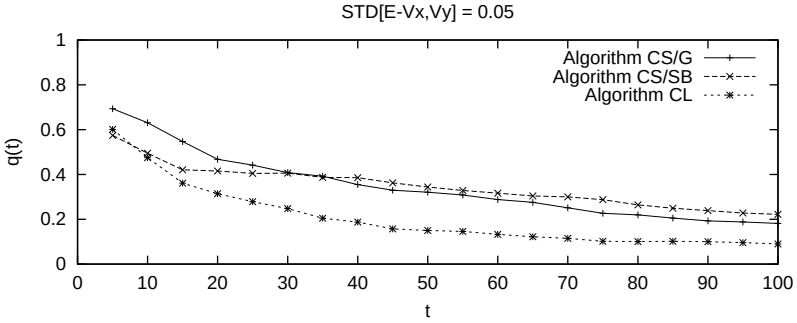


Figure 3.60: Efficiency when $\sigma_{E_{V_x},V_y} = 0.05$.

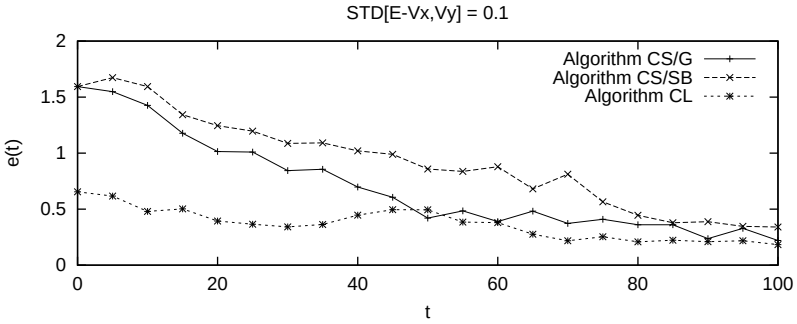


Figure 3.61: Error when $\sigma_{E_{V_x},V_y} = 0.1$.

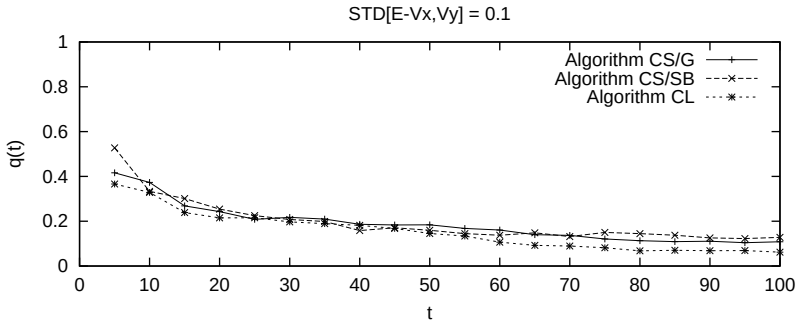


Figure 3.62: Efficiency when $\sigma_{E_{V_x,V_y}} = 0.1$.

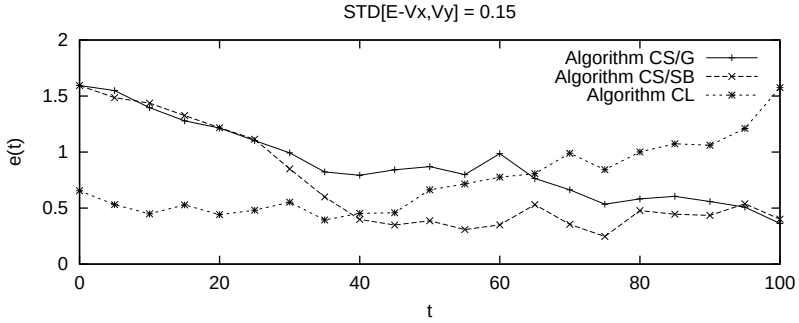


Figure 3.63: Error when $\sigma_{E_{V_x,V_y}} = 0.15$.

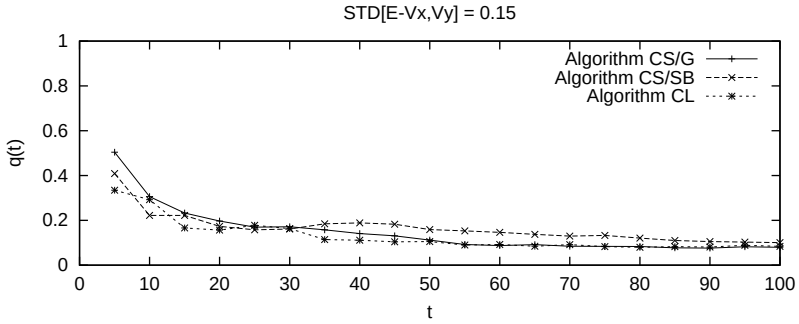


Figure 3.64: Efficiency when $\sigma_{E_{V_x,V_y}} = 0.15$.

With Algorithm CL, the nodes appeared to converge towards their desired positions under settings A and B, but drifted from their desired positions under setting C.

## 3.4 Conclusion

In this chapter we presented three algorithms for maintaining the geometric pattern of a marching array of mobile sensor nodes arranged in a line: a greedy algorithm for maintaining a constant separation between neighbouring nodes, Algorithm CS/G; a state-based algorithm for maintaining a constant separation between neighbouring nodes, Algorithm CS/SB; and a third one to keep the nodes equally spaced between the two end nodes of the array, Algorithm CL. The effect of $S_{\text{open}}$ and $S_{\text{break}}$ on the performance of Algorithm CS/SB was investigated, and it was demonstrated that $S_{\text{break}}$ should always be greater than the desired separation to obtain the best performance, and greater than $S_{\text{open}}$ for robustness. The effectiveness of the algorithms from various starting configurations was demonstrated. The robustness of the algorithms was investigated and the performance of Algorithms CS/G and CS/SB were compared under various conditions. For low levels of sensor error, that is image noise or error in the orientation of the cameras, Algorithm CS/SB was more efficient. Algorithm CS/G was notably more robust than Algorithm CS/SB against image noise. With the second strategy for maintaining sensor coverage, that is Algorithm CL, the nodes settled into a state of equilibrium more quickly, although this algorithm was less robust against velocity error. Since the final $x$-coordinates of the nodes depend purely on the $x$-coordinates of the nodes at each end, the performance of Algorithm CL is affected more by errors in the motion of the nodes at each end of the array.

This chapter has addressed a fundamental problem in the emerging field of mobile wireless sensor networks. Maintaining the relative spatial positioning of the sensors will be of major importance in many applications, including foraging, security and mine-sweeping. A more difficult problem is explored in the next chapter - *forming* a geometric pattern when the nodes begin at random positions.

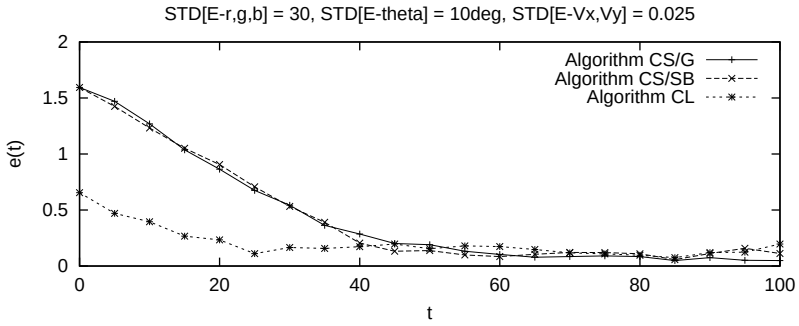


Figure 3.65: Error when $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

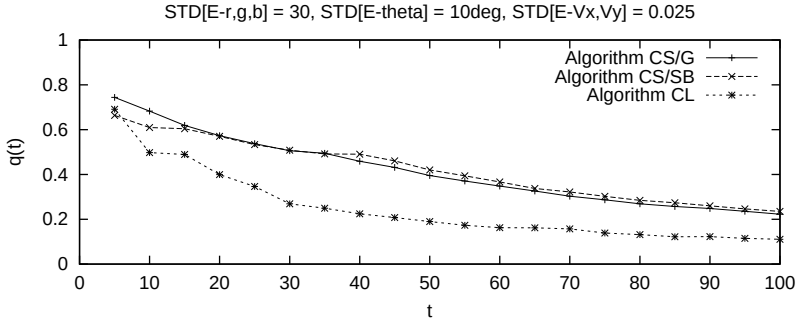


Figure 3.66: Efficiency when $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

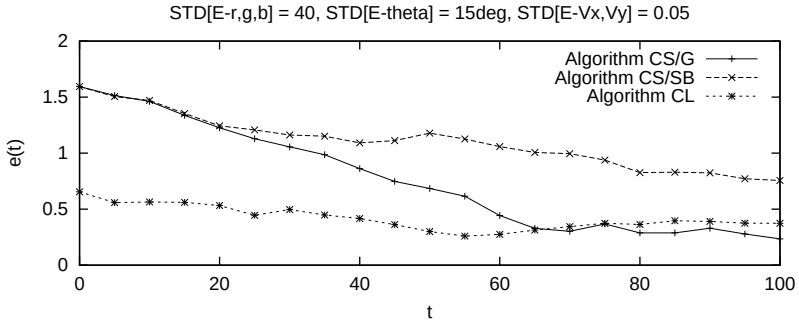


Figure 3.67: Error when $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

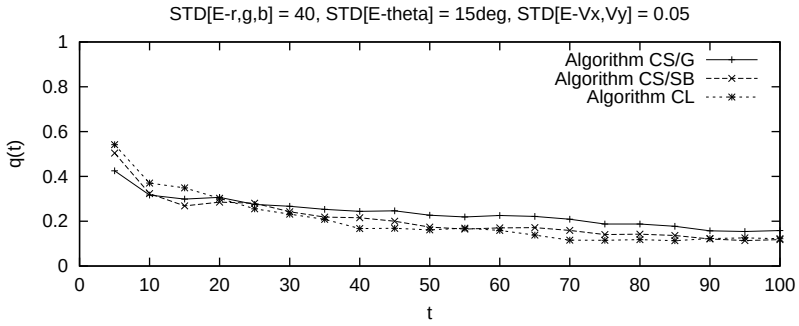


Figure 3.68: Efficiency when $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

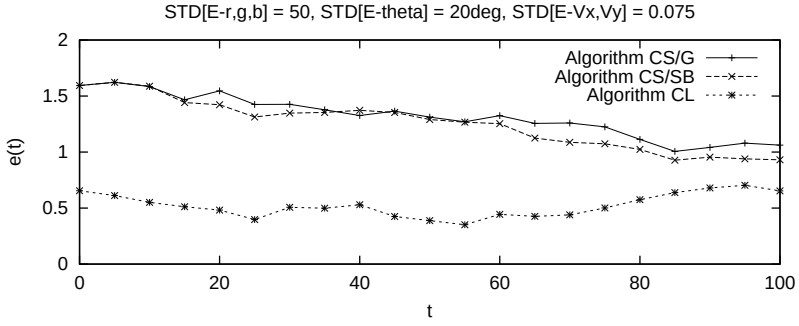


Figure 3.69: Error when $\sigma_{E_{r,g,b}} = 50$, $\sigma_{E_\theta} = 20^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.075$.

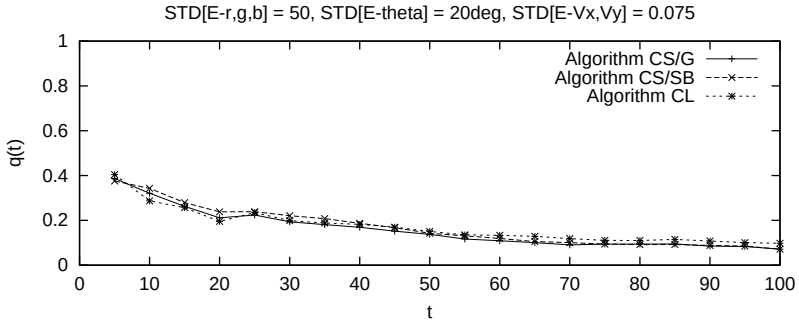


Figure 3.70: Efficiency when $\sigma_{E_{r,g,b}} = 50$, $\sigma_{E_\theta} = 20^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.075$.

# Chapter 4

# Forming the Geometric Pattern of an MWSN

## 4.1 Introduction

In the last chapter we explored the problem of maintaining the geometric pattern of an MWSN. We presented a number of algorithms that can be used to maintain a consistent pattern in a team of nodes arranged in a line. In this chapter we address a second fundamental challenge in coordinating the motion of an MWSN: the user will have no control over the locations of the nodes when the network is deployed in many of the foreseen applications of MWSNs, such as battlefield surveillance where the nodes will be dropped from a plane. In such applications the nodes will begin from random locations and therefore the network should be capable of distributed, fault-tolerant, spatial self-organisation. In this chapter we investigate the problem of *forming* a geometric pattern from any given set of initial node positions.

Six algorithms have been devised, two of which were discussed earlier:

**Algorithm CS/G:** Forms a line parallel with the $x$-axis with the nodes separated by a specified constant distance $S$.

**Algorithm CL:** Forms a line parallel with the $x$-axis, beginning at $x = x_{\min}$ and ending at $x = x_{\max}$, where $x_{\min}$ and $x_{\max}$ are the minimum and maximum $x$-coordinates, respectively. The nodes are equally spaced between the nodes at each end.

**Algorithm C/L:** Forms a circle with a given radius $R_c$ centred at a landmark.

**Algorithm P/$n$/L:** Forms a regular polygon with $n$ sides centred at a landmark. The distance from the centre to the closest point on the perimeter of the regular polygon is specified by the constant $R_{\min}$.

**Algorithm C:** Forms a circle with a given radius $R_c$ centred approximately at the centroid of the initial locations of the nodes.

**Algorithm P/$n$:** Forms a regular polygon with $n$ sides centred approximately at the centroid of the initial locations of the nodes. The distance from the centre to the closest point on the perimeter of the regular polygon is specified by the constant $R_{\min}$.

In the notation used to label the shape-formation algorithms, the algorithms that form a circle are denoted by C, and the algorithms that form a regular polygon with $n$ sides are denoted by P/$n$. If the label ends with L then the shape is centred at the landmark, otherwise the shape is centred approximately at the centroid of the nodes' locations.

Algorithms CS/G and CL were introduced in the previous chapter. We will show simulation traces demonstrating that these algorithms can also be used to form a stationary line with the nodes beginning from random positions when the combined viewing range of the cameras is extended to 360°.

Algorithm C/L is similar to Algorithm C, and Algorithm P/$n$/L is similar to Algorithm P/$n$. Algorithm C/L differs from Algorithm C in that no landmark is used to define the location of the latter, and likewise for Algorithms P/$n$/L and P/$n$.

The organisation of this chapter is as follows. The algorithms are discussed in Section 4.2. In Section 4.3 we demonstrate that Algorithms CS/G and CL can be used to form a line from a random initial configuration and then investigate the robustness of Algorithms C/L, P/$n$/L, C and P/$n$. Final conclusions for this chapter are given in Section 4.4.

## 4.2 Algorithms

The pattern-formation algorithms are comprised of the same components as the algorithms discussed in Chapter 3:

**Vision:** To determine the relative positions of neighbouring nodes and the landmark.

**Motion Planning:** To set the node's velocity to achieve the desired shape.

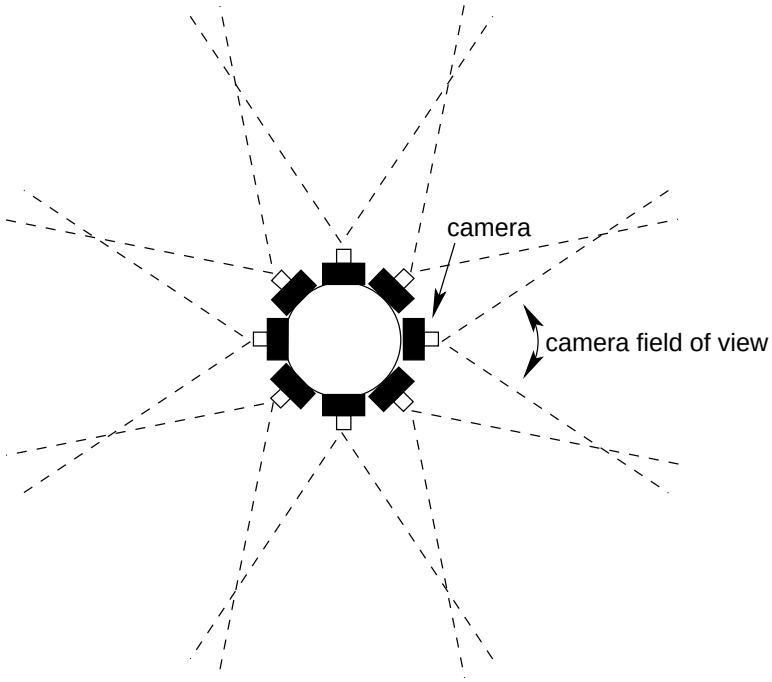


Figure 4.1: Using several cameras to achieve a 360° field of view.

### 4.2.1 Vision

In the following discussion, an *object* is either a node or a landmark.

The vision algorithm is based on the same assumptions that were used in the previous chapter, with the following exceptions:

- Each node is equipped with eight cameras positioned evenly on the circumference of the cylinder to achieve a 360° field of view, as illustrated in Figure 4.1. The views of adjacent cameras overlap to ensure coverage in all directions.
- A landmark is drawn as a cylinder with the same dimensions as a node, but with a different colour.

When processing an image from a particular camera, the coordinates of the object are first calculated with respect to the camera's local coordinate system. These coordinates are then converted to the local coordinate system of the node using the angle of the camera and its displacement from the centre of the node. Figure 4.2 illustrates the camera's, node's and global coordinate system.

The colour of the object that we wish to detect is passed as a parameter to the vision algorithm. Thus for Algorithms C/L and P/$n$/L the vision algorithm is executed twice for each image: one call to locate other nodes and a second to locate the landmark.

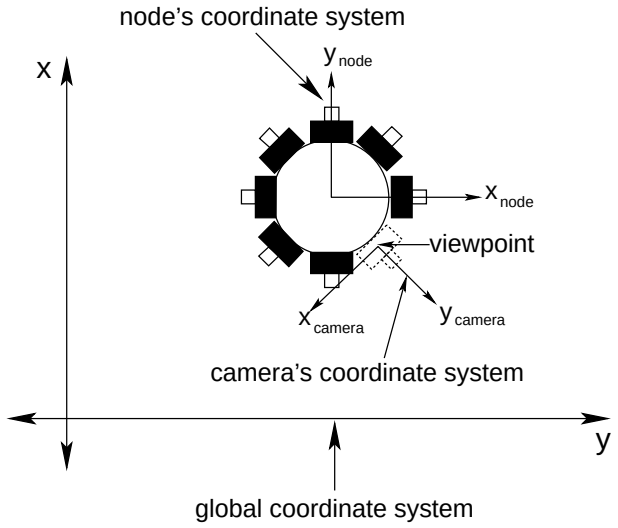


Figure 4.2: Coordinate systems of the camera and node and the global coordinate system.

If an object is located in a region where the views of two cameras overlap, then there will be two coordinates for the same object. Coordinates representing the same object are eliminated as follows: if two coordinates $(x_1, y_1)$ and $(x_2, y_2)$ are less than a distance of $2R$ apart, where $R$ is the radius of the landmark or node, then these coordinates are replaced by the centroid of the two.

### 4.2.2 Motion Planning

In the discussion that follows, for a given node $i$ with polar coordinates $(r_i, \theta_i)$, as illustrated in Figure 4.3:

- The *clockwise neighbour* is the node with the closest $\theta$-coordinate clockwise from $\theta_i$ and is labelled node $i - 1$.
- The *anticlockwise neighbour* is the node with the closest $\theta$-coordinate anticlockwise from $\theta_i$ and is labelled node $i + 1$.
- The *immediate neighbours* are the clockwise and anticlockwise neighbours.
- $\theta_{\text{cw}}$ is the angle subtended by node $i$ and its clockwise neighbour.
- $\theta_{\text{acw}}$ is the angle subtended by node $i$ and its anticlockwise neighbour.

The pattern-formation algorithms are analogous to Algorithms CS/G and CL in that the motion of each node consists of two components that are determined independently, and one of these components is responsible for achieving a symmetrical

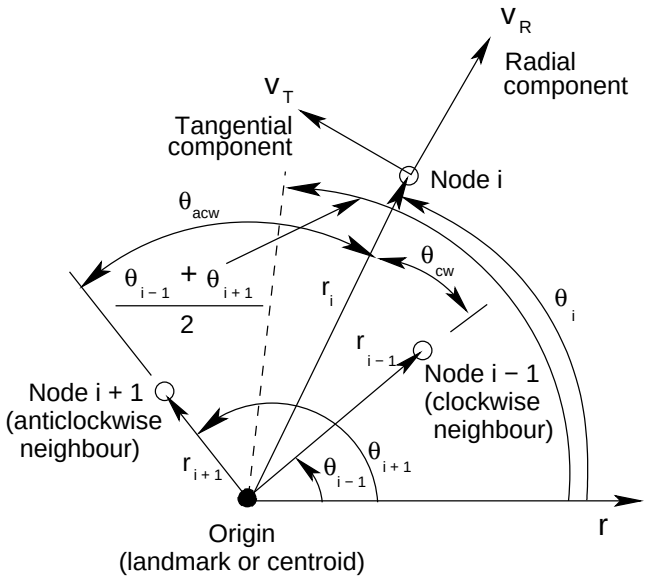


Figure 4.3: The *radial component*, $v_{\mathrm{R}}$, and *tangential component*, $v_{\mathrm{T}}$.

geometric relationship with the immediate neighbours. The motion of each node is considered in terms of the following two vector components, as illustrated in Figure 4.3, where the origin is either a landmark, as with Algorithms C/L and P/$n$/L, or the centroid of the nodes' current positions, as with Algorithms C and P/$n$:

**Radial component ($v_{\mathrm{R}}$):** A component *parallel* to a line from the origin to the node. $v_{\mathrm{R}}$ is set to move node $i$ towards the edge of the shape. If $\rho(\theta_i)$ is the distance of the edge of the shape from the origin at an angle of $\theta_i$, $v_{\mathrm{R}}$ is positive when $r_i < \rho(\theta_i)$, 0 when $r_i = \rho(\theta_i)$ and negative when $r_i > \rho(\theta_i)$.

**Tangential component ($v_{\mathrm{T}}$):** A component *perpendicular* to a line from the origin to the node. $v_{\mathrm{T}}$ is set to move node $i$ in the direction of the line with polar coordinates satisfying $\theta = \frac{\theta_{i-1}+\theta_{i+1}}{2}$. Eventually the nodes reach a state of equilibrium where the angle subtended by every pair of neighbouring nodes is $\frac{2\pi}{N}$ for a network with $N$ nodes.

For simplicity, positive and negative values of the same magnitude were used for setting $v_{\mathrm{R}}$ and $v_{\mathrm{T}}$. That is, $v_{\mathrm{R}} = \pm\Delta v_{\mathrm{R}}$ or 0 and $v_{\mathrm{T}} = \pm\Delta v_{\mathrm{T}}$ or 0, where $\Delta v_{\mathrm{R}}$ and $\Delta v_{\mathrm{T}}$ are constants chosen according to the capabilities of the actuators and vision processing system.

For Algorithms C/L and C, $\rho(\theta_i) = R_{\mathrm{c}}$. For Algorithms P/$n$/L and P/$n$, $\rho(\theta_i)$ depends on $\theta_i$ and is determined as follows. Assume that the desired regular polygon

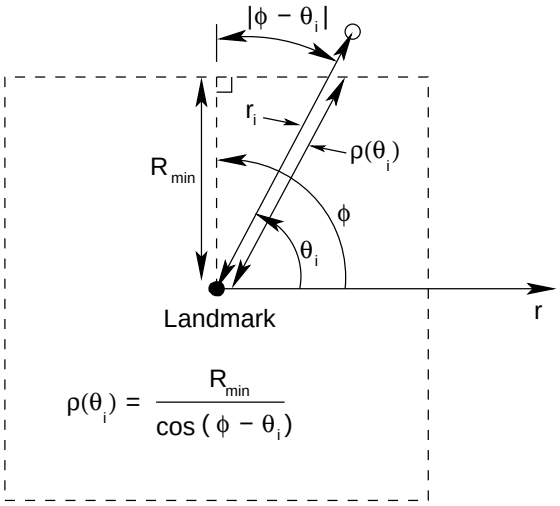


Figure 4.4: Calculating $\rho(\theta_i)$ when forming a regular polygon with Algorithms P/$n$/L and P/$n$.

is oriented so that one of the sides intersects the line $\theta = 0$ at 90°. First, we determine which side of the shape intersects a line drawn from the origin to the edge of the shape at an angle of $\theta_i$ with respect to the origin. We then determine the angle $\phi$ of a line from the origin which intersects that side of the shape at 90°. These two lines and the edge of the shape form a right triangle, where the vertex at the origin has an angle of $|\phi - \theta_i|$, as illustrated in Figure 4.4. Thus, using trigonometry, $\rho(\theta_i) = \frac{R_{\min}}{\cos(\phi - \theta_i)}$.

Once the values for $v_{\mathrm{R}}$ and $v_{\mathrm{T}}$ have been determined, we rotate the vector $< v_{\mathrm{R}}, v_{\mathrm{T}} >$ by $\theta_i$ to obtain $< v_x, v_y >$, the final velocity defined in the node's local coordinate system.

The shape-formation algorithms are formally stated below. $\rho(\theta_i)$ is calculated in Step 1 and is different for each algorithm. The velocity is determined in Step 2.

**Step 1.** For Algorithms C/L and P/$n$/L, the origin is set to the location of the landmark, and for Algorithms C and P/$n$, the origin is set to the centroid of the nodes' current locations.

Algorithms C/L and C begin with

$$\rho(\theta_i) \leftarrow R_{\mathrm{c}}$$

and Algorithms P/$n$/L and P/$n$ begin with

$$\begin{aligned} \Delta\theta &\leftarrow \frac{2\pi}{n} \\ \phi &\leftarrow \mathsf{round}\left(\frac{2\pi}{\Delta\theta}\right)\Delta\theta \\ \rho(\theta_i) &\leftarrow \frac{R_{\min}}{\cos(\phi-\theta_i)} \end{aligned}$$

where $\mathsf{round}(x)$ is $x$ rounded to the nearest integer.

**Step 2.** Step 2 is shown in Algorithm 5.

**Algorithm 5** Step 2 for Algorithms C/L, P/$n$/L, C and P/$n$.

**if** $r_i < \rho(\theta_i)$ **then**
  $v_{\mathrm{R}} \leftarrow \Delta v_{\mathrm{R}}$
**else if** $r_i > \rho(\theta_i)$ **then**
  $v_{\mathrm{R}} \leftarrow -\Delta v_{\mathrm{R}}$
**else**
  $v_{\mathrm{R}} \leftarrow 0$
**end if**
**if** $\theta_{\mathrm{acw}} > \theta_{\mathrm{cw}}$ **then**
  $v_{\mathrm{T}} \leftarrow \Delta v_{\mathrm{T}}$
**else if** $\theta_{\mathrm{acw}} < \theta_{\mathrm{cw}}$ **then**
  $v_{\mathrm{T}} \leftarrow -\Delta v_{\mathrm{T}}$
**else**
  $v_{\mathrm{T}} \leftarrow 0$
**end if**
$v_x \leftarrow v_{\mathrm{R}} \cos\theta_i - v_{\mathrm{T}} \sin\theta_i$
$v_y \leftarrow v_{\mathrm{R}} \sin\theta_i + v_{\mathrm{T}} \cos\theta_i$

## 4.3 Experiments

The pattern-formation algorithms were tested using a similar approach to the pattern-maintenance algorithms discussed in the previous chapter. The robustness of the algorithms were tested by running simulations with various levels of image noise, error in the orientation of the cameras and velocity error.

### 4.3.1 Simulation Setup

As in the previous chapter, the positions of the nodes were computed at discrete points in time. The ray-tracer produced a "snapshot" of the scene from each camera for a

given instant of time. The same conditions were used, with the following exceptions:

- The image plane is one unit wide and one unit high.
- Each image is 100 pixels wide and 100 pixels high.
- For Algorithms CS/G and CL, $V_x = V_y = 0$.

The constants used in motion planning had the following values:

- $\Delta v_{\mathrm{R}} = \Delta v_{\mathrm{T}} = 0.05$.
- $R_{\mathrm{c}} = 2$ when forming a circle and $R_{\mathrm{min}} = 2$ when forming a regular polygon.

The various initial configurations that were used are shown in Figures 4.5 to 4.9. Configuration 4A was used to test the robustness of each algorithm against image noise, error in the orientation of the cameras and velocity error, as in Chapter 3. Configuration 4B was used to investigate the performance of Algorithm C/L when the landmark is located outside the convex hull of the initial locations. Algorithm C was tested with Configuration 4C to investigate how well this algorithm handles an initial configuration consisting of two groups of nodes that are relatively far apart. Using Configuration 4D, Algorithm C was also used to form a circle with the nodes initially arranged in a line. Finally, Algorithm P/4 was tested with two nodes beginning far outside the main group using Configuration 4E.

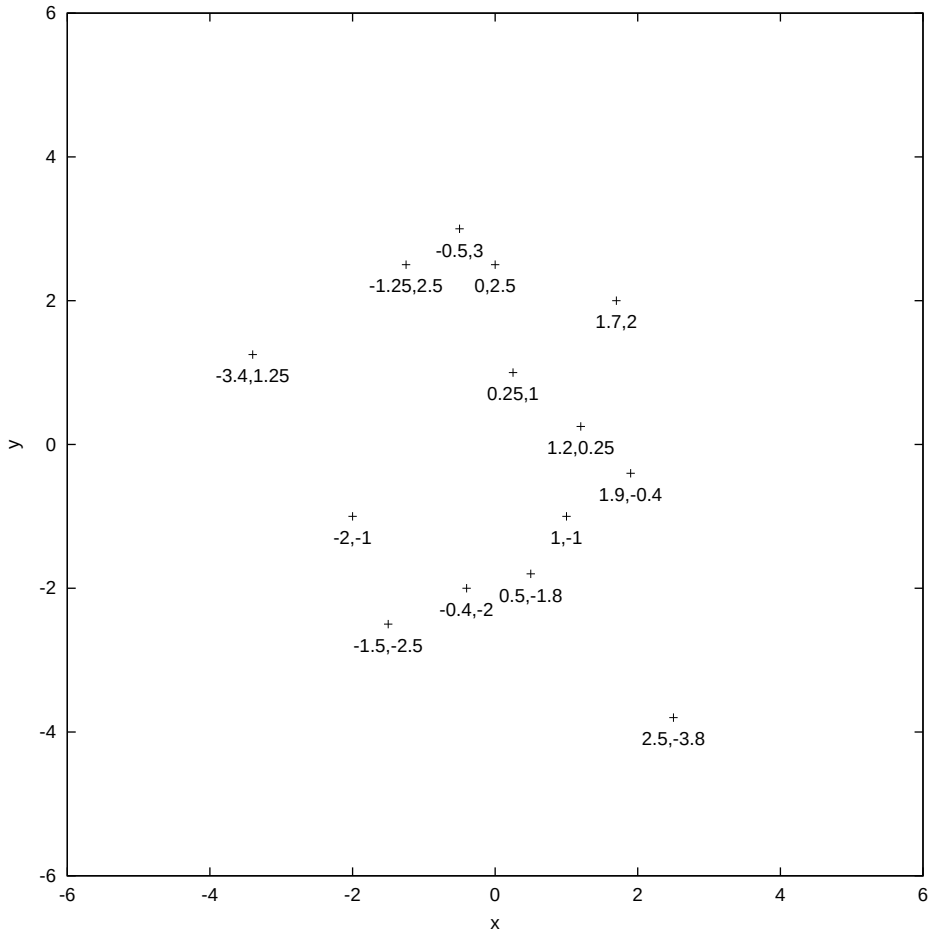


Figure 4.5: Configuration 4A.

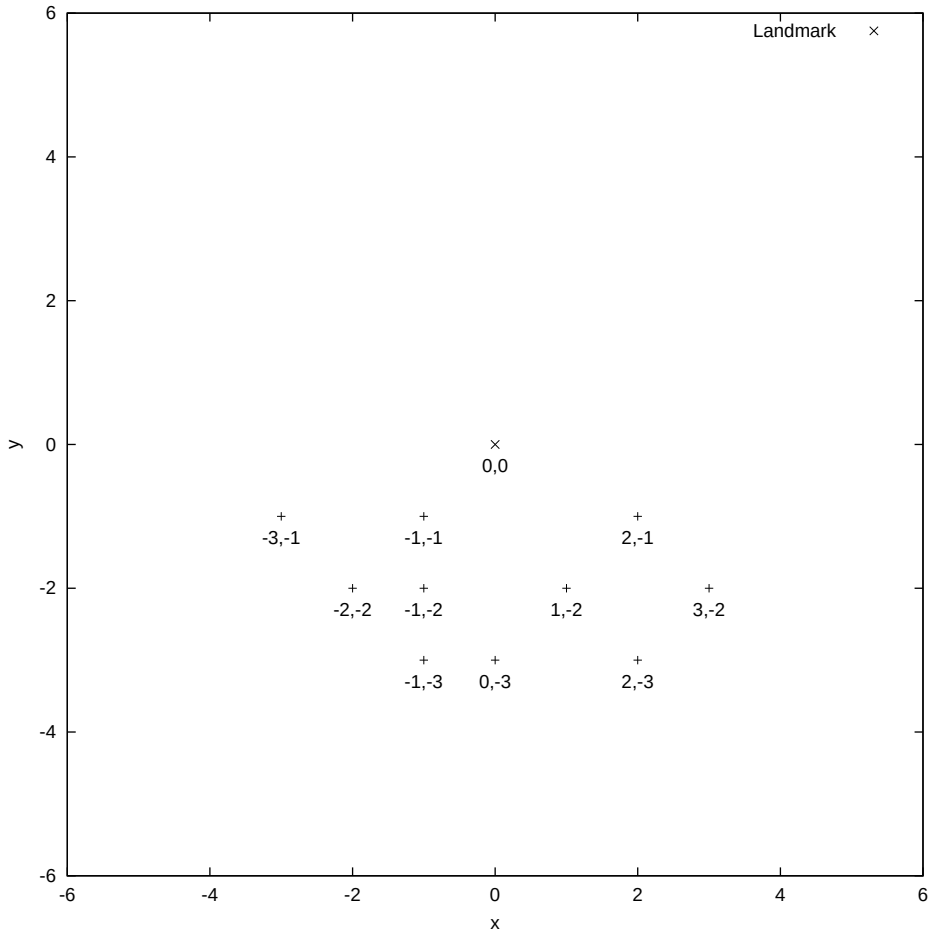


Figure 4.6: Configuration 4B.

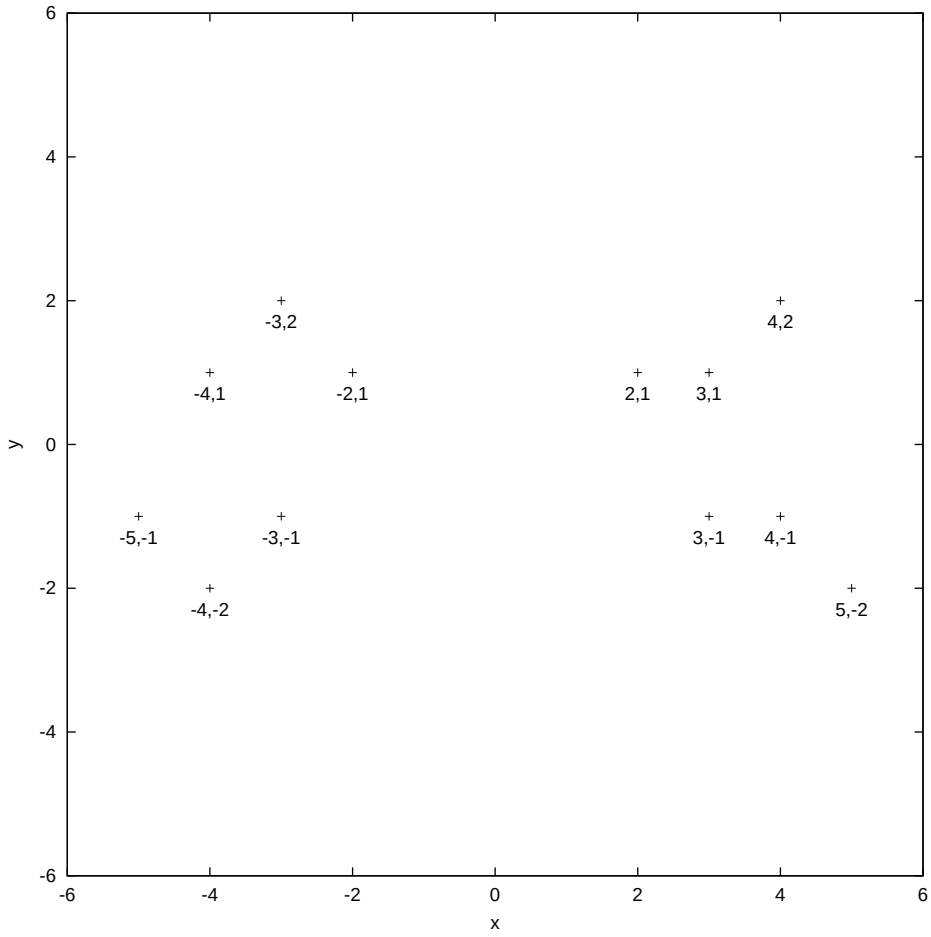


Figure 4.7: Configuration 4C.

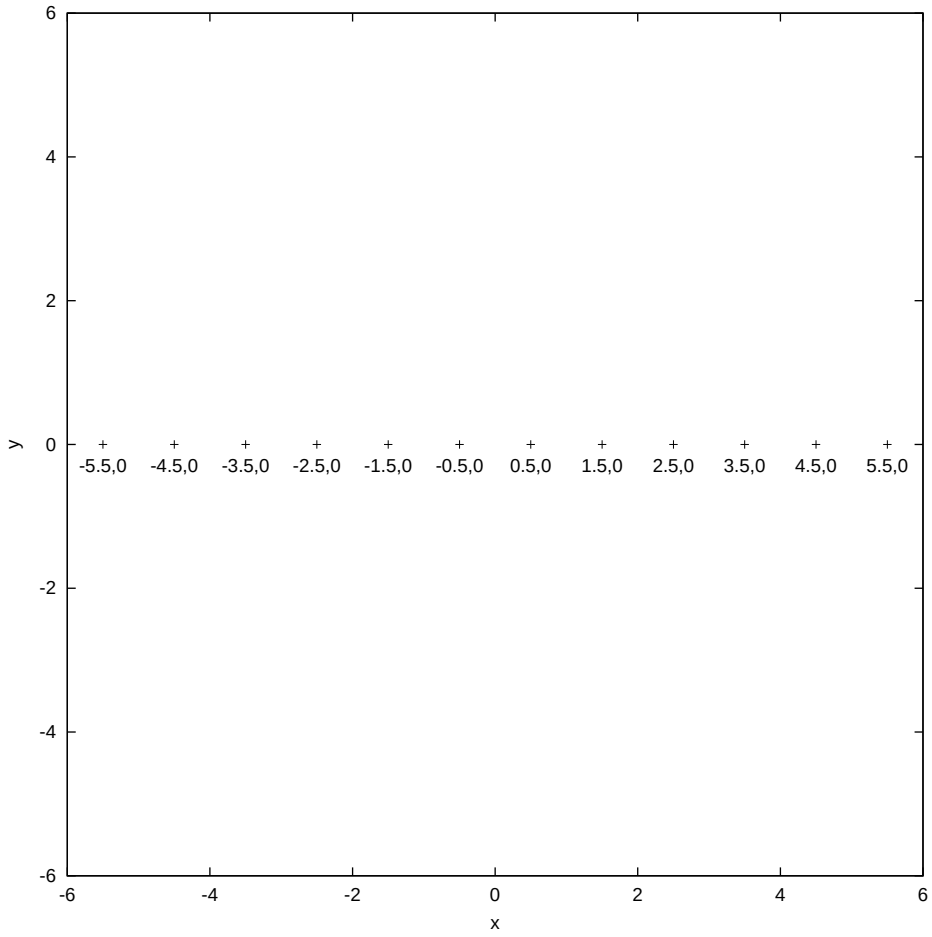


Figure 4.8: Configuration 4D.

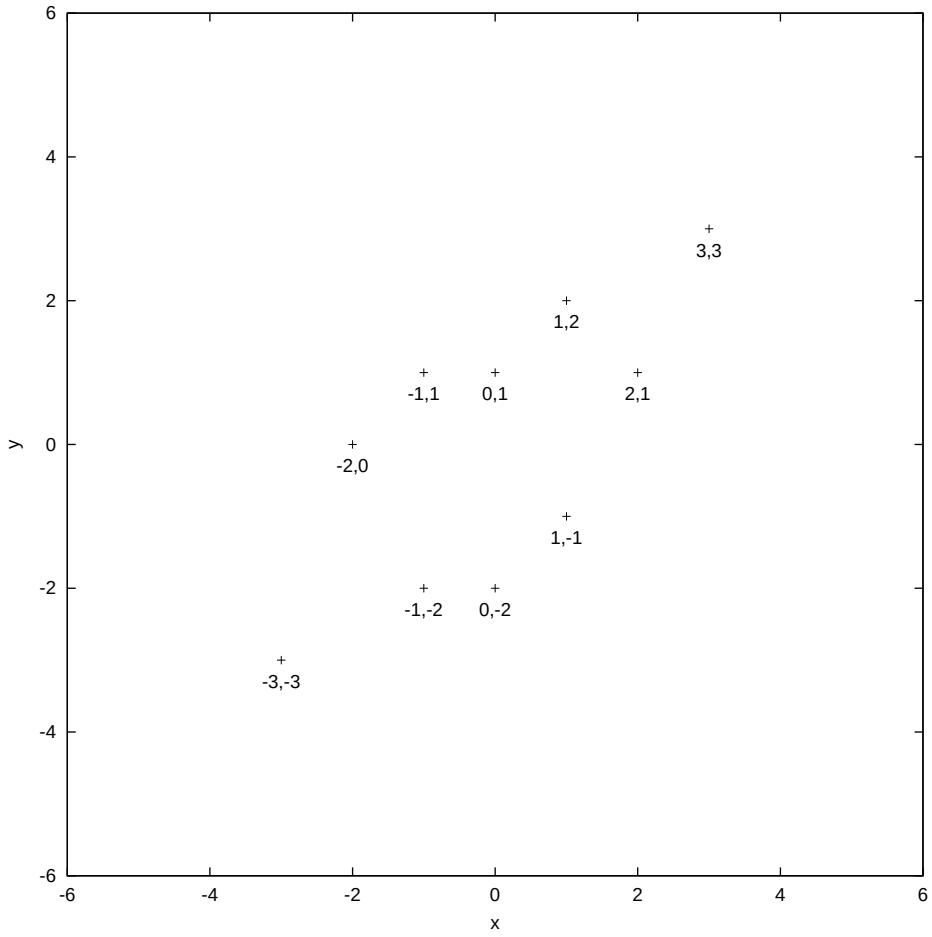


Figure 4.9: Configuration 4E.

### 4.3.2 Performance Analysis

The error $e(t)$ and efficiency $q(t)$, as defined in Chapter 3, were plotted over time. The desired positions are chosen to minimise the sum of the square of the angle between each node and its desired position with respect to the origin. Since it is important only that any given pair of neighbouring nodes subtend an angle of $\frac{2\pi}{N}$ with respect to the origin, the final angles of the nodes are rather arbitrary. Once the desired angle of a node has been determined, the desired distance of the node from the origin is calculated using Step 1 of the motion coordination algorithm for the shape that is desired.

Assume that $N$ nodes each with polar coordinates $(r_i, \theta_i)$, for $i$, $i = 0, \ldots, N-1$, are indexed so that $\theta_0 < \theta_1$, $\theta_1 < \theta_2$, and so on until $\theta_{N-2} < \theta_{N-1}$. Let $\hat{r}_i$ and $\hat{\theta}_i$ be the desired $r$- and $\theta$-coordinates of node $i$, respectively. Then

$$\begin{aligned} F_\theta &= \sum_{i=0}^{N-1} \left(\hat{\theta}_i - \theta_i\right)^2 \\ &= \sum_{i=0}^{N-1} \left(\hat{\theta}_0 + \frac{2\pi i}{N} - \theta_i\right)^2. \end{aligned}$$

We wish to find the value of $\hat{\theta}_0$ that minimises $F_\theta$, so we determine the partial derivative of $F_\theta$ with respect to $\hat{\theta}_0$ and equate it to zero:

$$\begin{aligned} \frac{\partial F_\theta}{\partial \hat{\theta}_0} &= 2 \sum_{i=0}^{N-1} \left(\hat{\theta}_0 + \frac{2\pi i}{N} - \theta_i\right) \\ &= 2 \left[ N\hat{\theta}_0 + \frac{2\pi}{N} \frac{N(N-1)}{2} - \sum_{i=0}^{N-1} \theta_i \right] \\ &= 2 \left[ N\hat{\theta}_0 + \pi(N-1) - \sum_{i=0}^{N-1} \theta_i \right] \end{aligned}$$

$$\frac{\partial F_\theta}{\partial \hat{\theta}_0} = 0 \Rightarrow \hat{\theta}_0 = \frac{\sum_{i=0}^{N-1} \theta_i - \pi(N-1)}{N}$$

Since $\frac{\partial^2 F_\theta}{\partial \hat{\theta}_0^2} = 2N > 0$ the above solution minimises $\hat{\theta}_0$. Thus, the desired $\theta$-coordinate of node $i$ is

$$\hat{\theta}_i = \frac{\sum_{i=0}^{N-1} \theta_i - \pi(N-1)}{N} + \frac{2\pi i}{N}$$

and

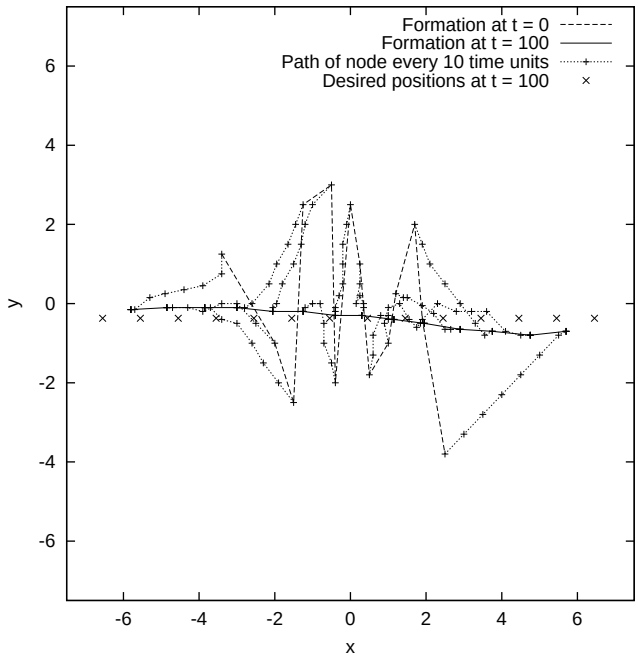


Figure 4.10: Simulation trace of Algorithm CS/G using Configuration 4A.

$$\hat{r}_i = \rho(\hat{\theta}_i).$$

### 4.3.3 Results

#### Simulation Traces

**Formation of a Line** Simulation traces of Algorithms CS/G and CL executed with a zero constant component in the velocity are shown in Figures 4.10 and 4.11, respectively. The progressive positions of the nodes after every ten time units are shown in the plot. The desired positions at the end of the simulation are also plotted.

The lines formed are not quite parallel with the $x$-axis because of occlusion. When the nodes are nearly aligned along the $y$-axis each node's immediate neighbours occlude all the other nodes in the array. If, for example, the left neighbour is slightly in front by a distance $\Delta y$ for each node, then the node at the left end of the array will be a distance $(N-1)\Delta y$ ahead of the node on the right end of the array.

**Formation of a Shape** Simulation traces of the shape-formation algorithms are shown in Figures 4.12 to 4.21. Note that the vertices of the final shape can appear

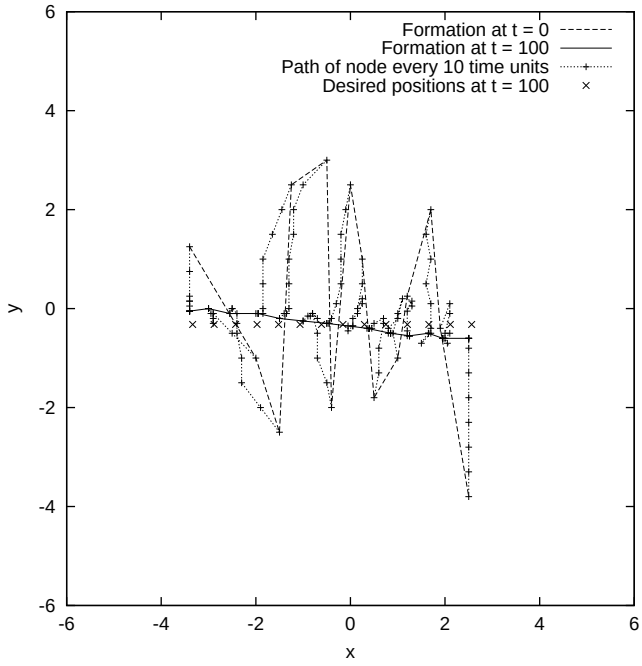


Figure 4.11: Simulation trace of Algorithm CL using Configuration 4A.

cut-off even if the nodes are at their desired positions because neighbouring nodes are always directly joined by a line and the set of desired positions does not necessarily include the vertices of the desired shape.

The results demonstrate that the shape-formation algorithms are robust against the initial random placement of the nodes. In all the simulations performed with $\sigma_{E_{r,g,b}} = \sigma_{E_\theta} = \sigma_{E_{V_x,V_y}} = 0$, the nodes were generally reasonably close to their desired positions.

Due to occlusion, the polygons were smaller than desired when the nodes used the centroid of their locations to position themselves. The immediate neighbours of a given node, except for neighbours belonging to other sides, occlude the other nodes forming the same side. Since all the nodes forming the same side except the immediate neighbours are not included in the calculation of the centroid, the centroid is calculated to be further away than what it really is. As a result, there is a tendency for node $i$ to finish with $r_i < \rho(\theta_i)$. Among all the simulations of the shape-formation algorithms with $\sigma_{E_{r,g,b}} = \sigma_{E_\theta} = \sigma_{E_{V_x,V_y}} = 0$, the final error was greatest when Algorithm P/3 was used because the triangle formed in this simulation had the longest sides and therefore was the most affected by occlusion.

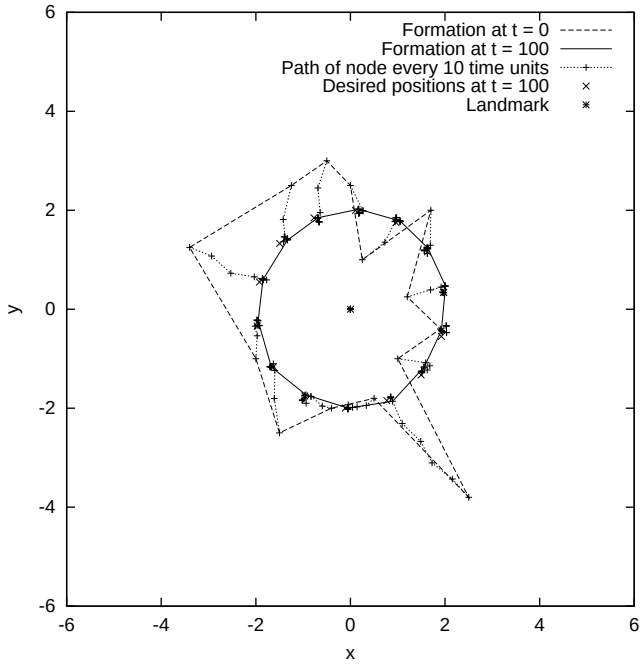


Figure 4.12: Simulation trace of Algorithm C/L using Configuration 4A.

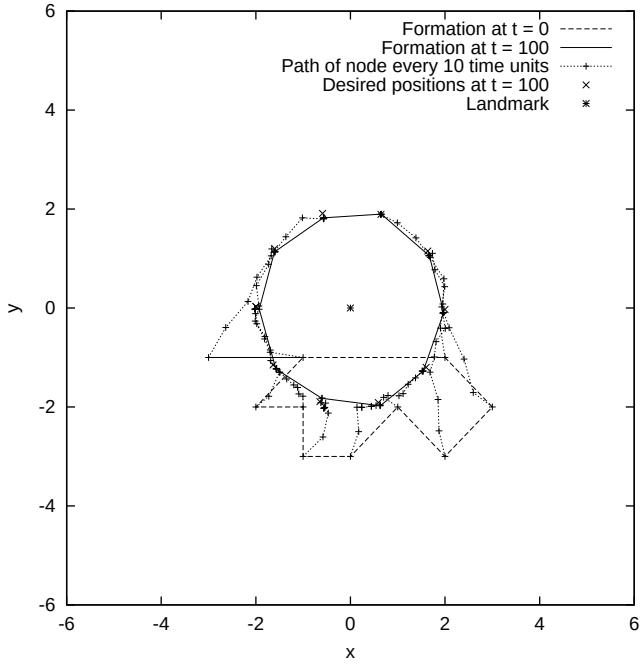


Figure 4.13: Simulation trace of Algorithm C/L using Configuration 4B.

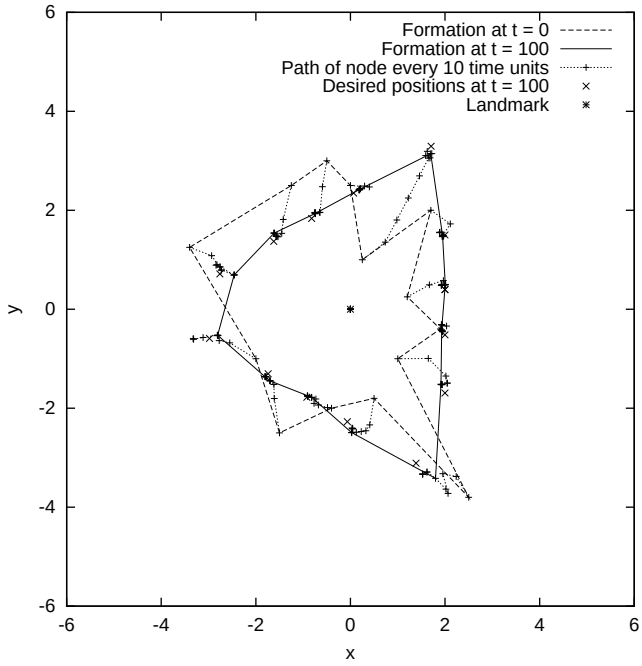


Figure 4.14: Simulation trace of Algorithm P/3/L using Configuration 4A.

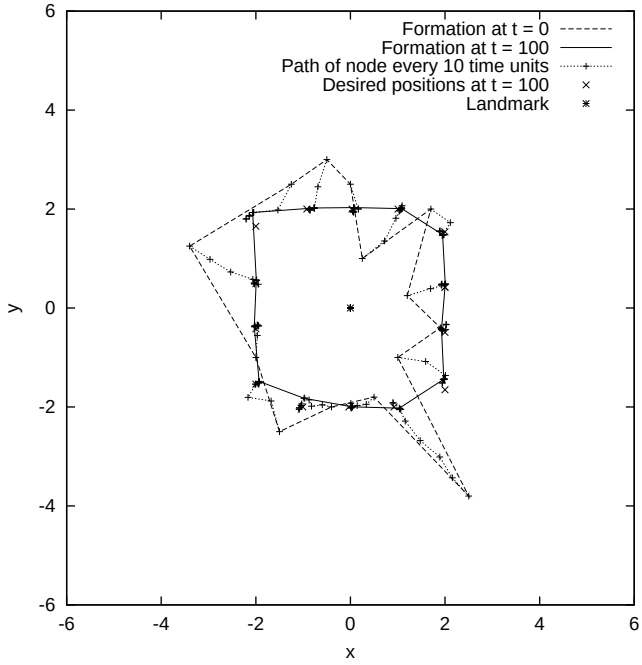


Figure 4.15: Simulation trace of Algorithm P/4/L using Configuration 4A.

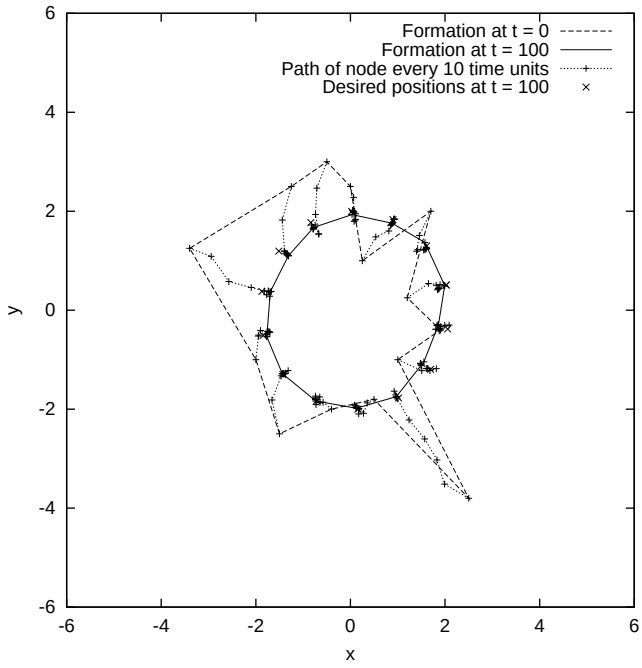


Figure 4.16: Simulation trace of Algorithm C using Configuration 4A.

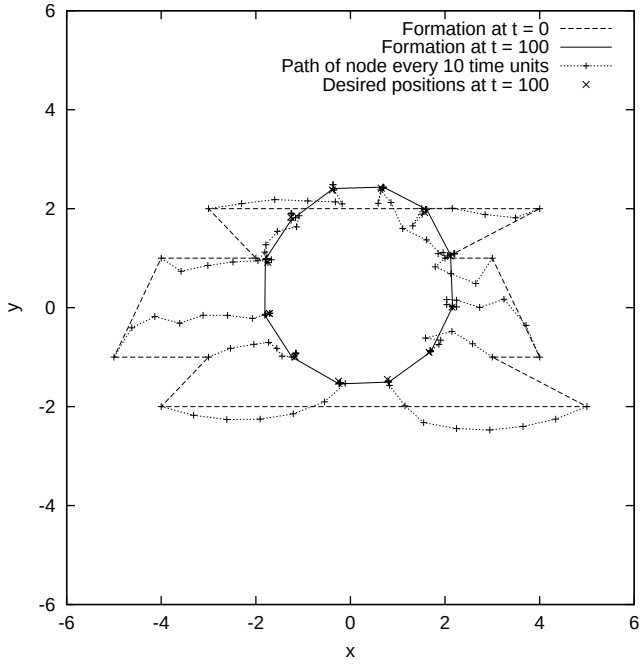


Figure 4.17: Simulation trace of Algorithm C using Configuration 4C.

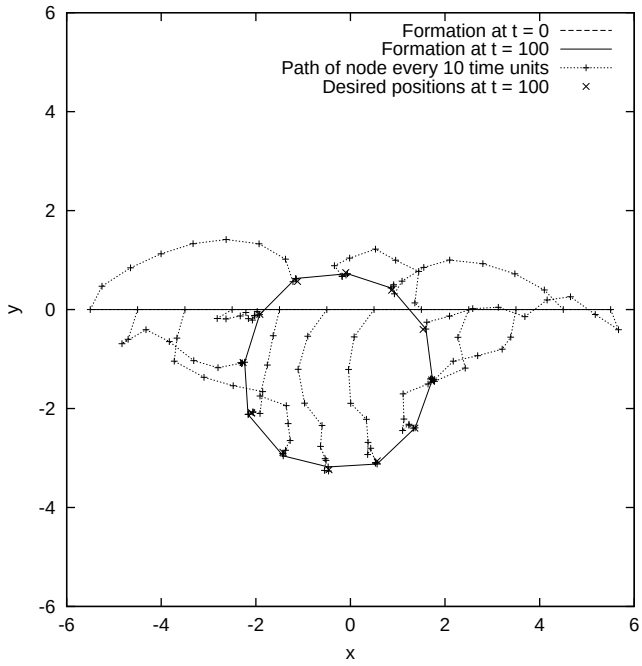


Figure 4.18: Simulation trace of Algorithm C using Configuration 4D.

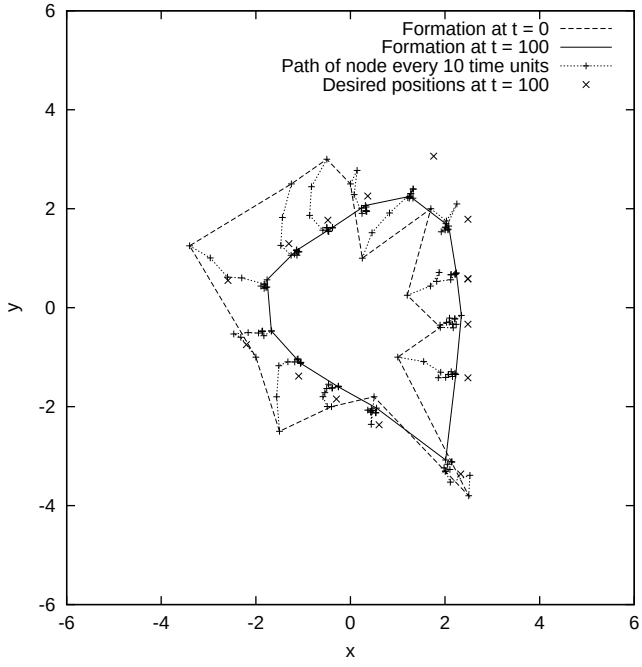


Figure 4.19: Simulation trace of Algorithm P/3 using Configuration 4A.

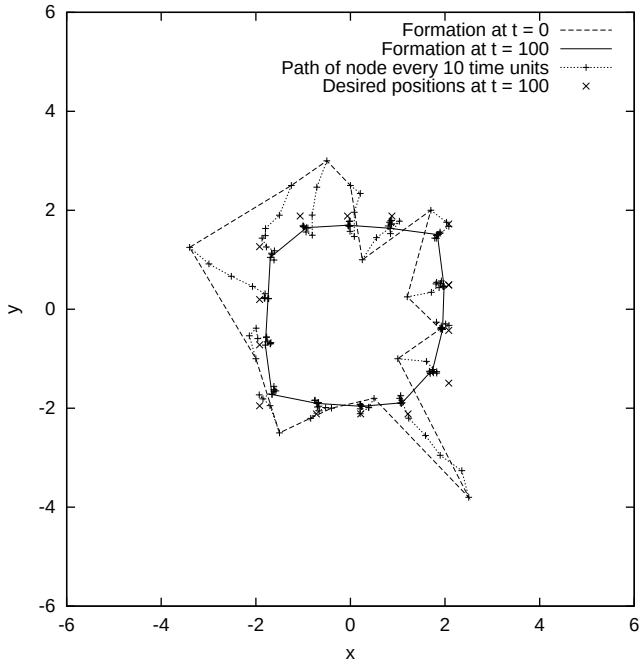


Figure 4.20: Simulation trace of Algorithm P/4 using Configuration 4A.

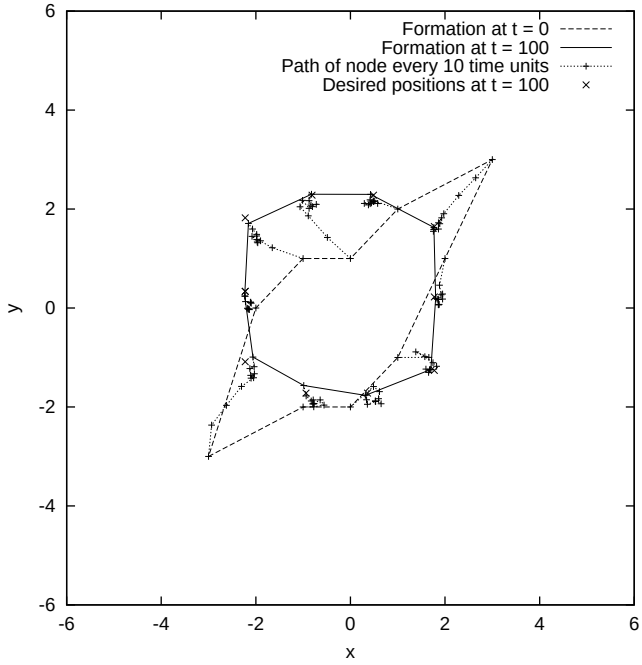


Figure 4.21: Simulation trace of Algorithm P/4 using Configuration 4E.

### Robustness

**Robustness Against Image Noise** The error over time for each pattern-formation algorithm using Configuration 4A with $\sigma_{E_{r,g,b}} = 30$, 40 and 50 is shown in Figures 4.22, 4.24 and 4.26, respectively. The efficiency over time for each of these simulations is shown in Figures 4.23, 4.25 and 4.27, respectively.

The performance of the landmark-based algorithms became significantly degraded when $\sigma_{E_{r,g,b}} = 50$. For this value of $\sigma_{E_{r,g,b}}$, Algorithms C/L and P/4/L achieved little reduction in the error, while Algorithm P/3/L initially dropped a little and then increased.

The error for the centroid-based algorithms was higher than that of the landmark-based algorithms because of occlusion, as explained in Section 4.3.3. For $\sigma_{E_{r,g,b}} = 40$ and 50, the error for the centroid-based algorithms initially decreased as the more distant nodes moved closer to the centroid and then rose at about $t = 15$ because each node $i$ continues to move closer to the centroid than the desired distance $\rho(\theta_i)$. The vision algorithm tends to exclude "noisy" segments of pixels in distant nodes and fragment distant nodes with "noisy" vertical segments, causing the calculated location of the centroid to be further away than the real location. Hence the final shape is smaller than desired.

The error varied noticeably with shape because the accuracy of the calculated relative position of a node or landmark decreases with distance and the average distance between neighbouring desired positions varied with the shape. The average distance between neighbouring desired positions for the circle was lower than that for the square, thus when the nodes were close to their desired positions they tended to locate their neighbours more accurately in the formation of the circle than in the formation of the square. Likewise, when the nodes were close to their desired positions they tended to locate their neighbours more accurately in the formation of the square than in the formation of the triangle.

The error was also affected by the average distance of the nodes from the landmark. The desired positions for the circle were generally closer to the landmark than those for the square, thus when the nodes were close to their desired positions the landmark was located more accurately in the formation of the circle than in the formation of the square. For the same reason, the nodes tended to locate the landmark more accurately when they were close to their desired positions in the formation of the square than in the formation of the triangle.

Note that the efficiency is meaningful only when the error is decreasing, since the efficiency for a node can be high after moving to a location that is further from the desired position if the path taken to that point is close to a straight line.

**Robustness Against Error in Orientation of Cameras** The error over time for each pattern-formation algorithm using Configuration 4A with $\sigma_{E_\theta} = 30°$, $60°$ and $90°$ is shown in Figures 4.28, 4.30 and 4.32, respectively. The efficiency over time for each of these simulations is shown in Figures 4.29, 4.31 and 4.33, respectively.

Generally the error varied with shape in the same manner as it did in the simulations with image noise. The error plots show that generally (i) Algorithm C/L performed better than Algorithm P/4/L; and (ii) Algorithm P/4/L performed better than Algorithm P/3/L. Likewise, generally Algorithm C performed better than Algorithm P/4 and Algorithm P/4 performed better than Algorithm P/3.

**Robustness Against Error in Velocity** The error over time for each shape-formation algorithm using Configuration 4A with $\sigma_{E_{Vx},V_y} = 0.05$, 0.1 and 0.15 is shown in Figures 4.34, 4.36 and 4.38, respectively. The efficiency over time for each of these simulations is shown in Figures 4.35, 4.37 and 4.39, respectively.

The variation of error with shape that occurred in the set of simulations with image noise and the set of simulations with error in the orientation of cameras also occurred with $\sigma_{E_{V_x},V_y} = 0.05$. Also, the plots of error for $\sigma_{E_{V_x},V_y} = 0.05$ show that the centroid-based algorithms are affected by velocity error more than the landmark-based algorithms.

The performance of the algorithms was noticeably more erratic over time than that when image noise or error in the orientation of cameras was included in the simulations. For example, at $t = 20$ Algorithm P/3 had the lowest error with $e(20) \approx 0.4$, but at $t = 80$ it had the highest error with $e(80) \approx 0.8$.

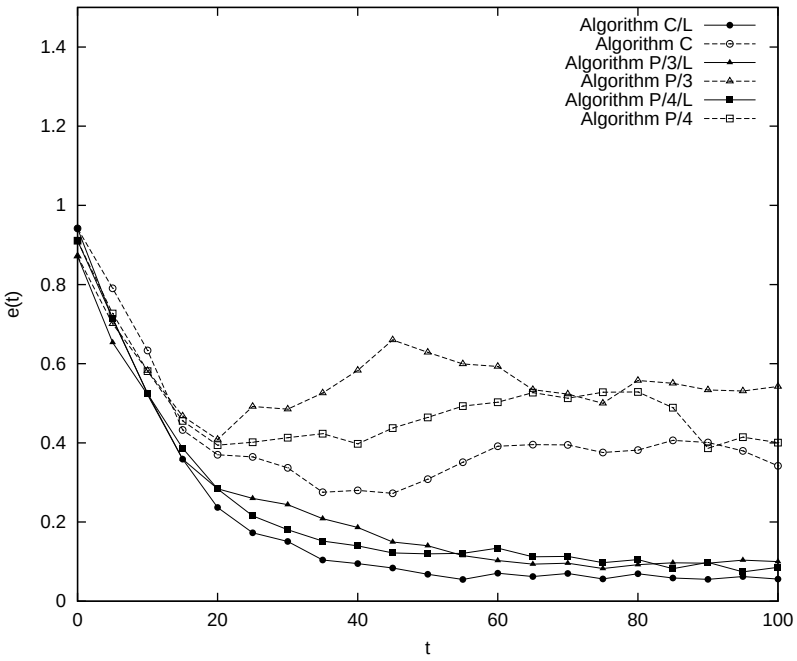


Figure 4.22: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 30$.

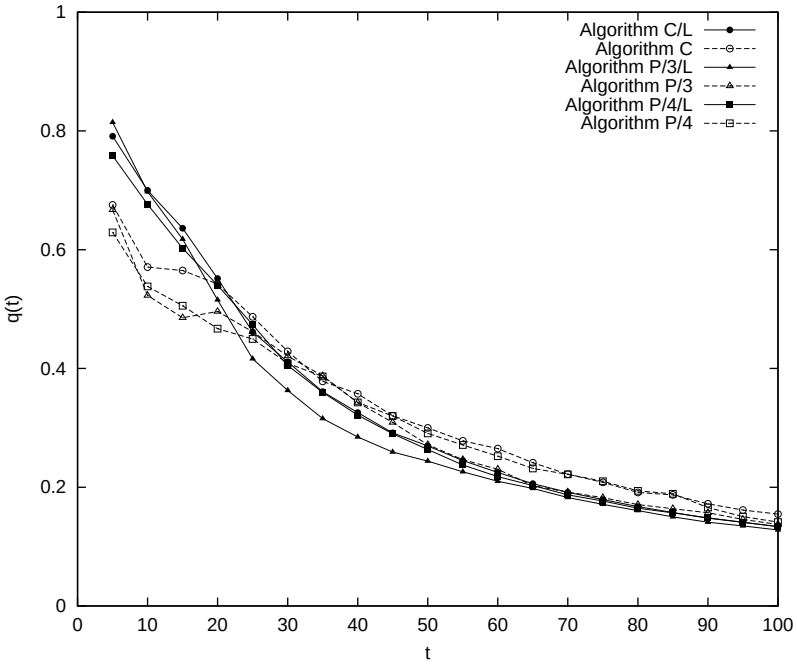


Figure 4.23: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 30$.

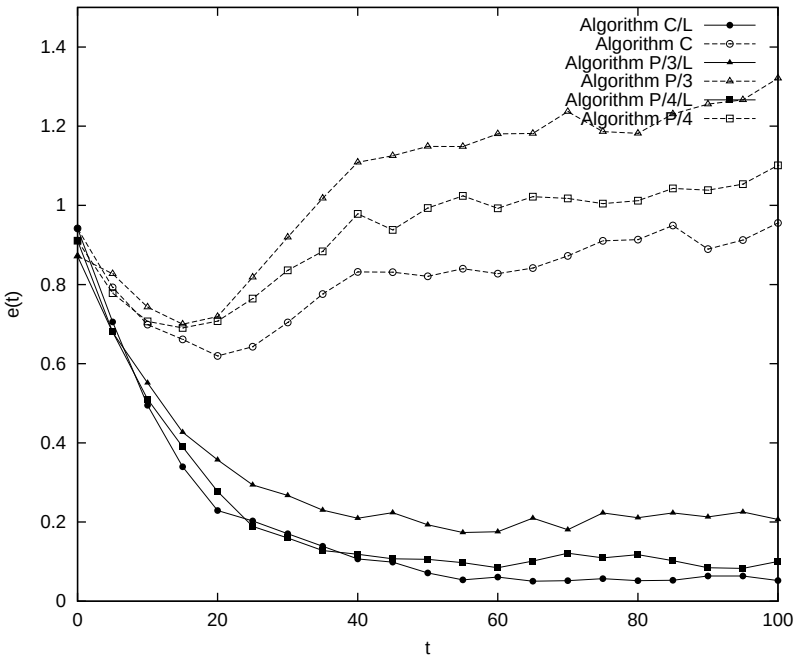


Figure 4.24: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 40$.

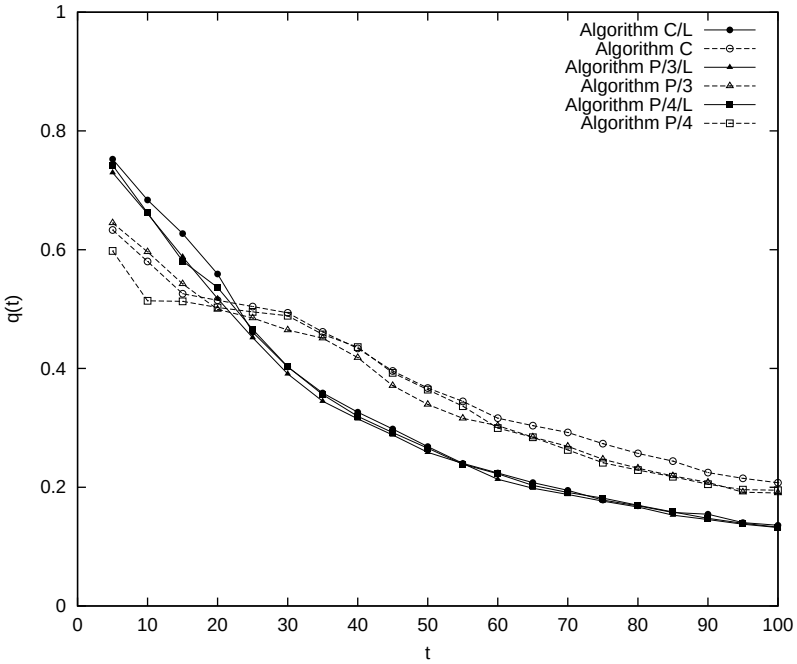


Figure 4.25: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 40$.

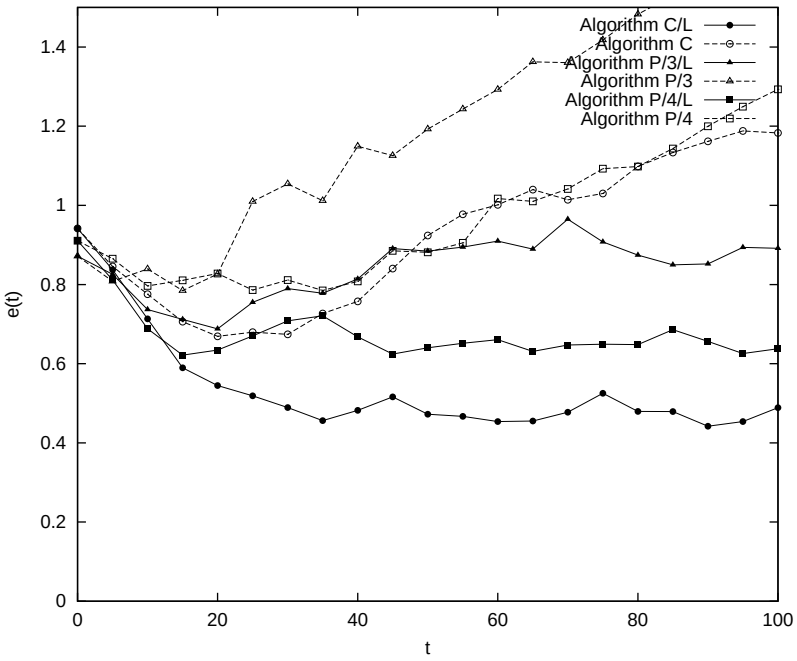


Figure 4.26: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 50$.

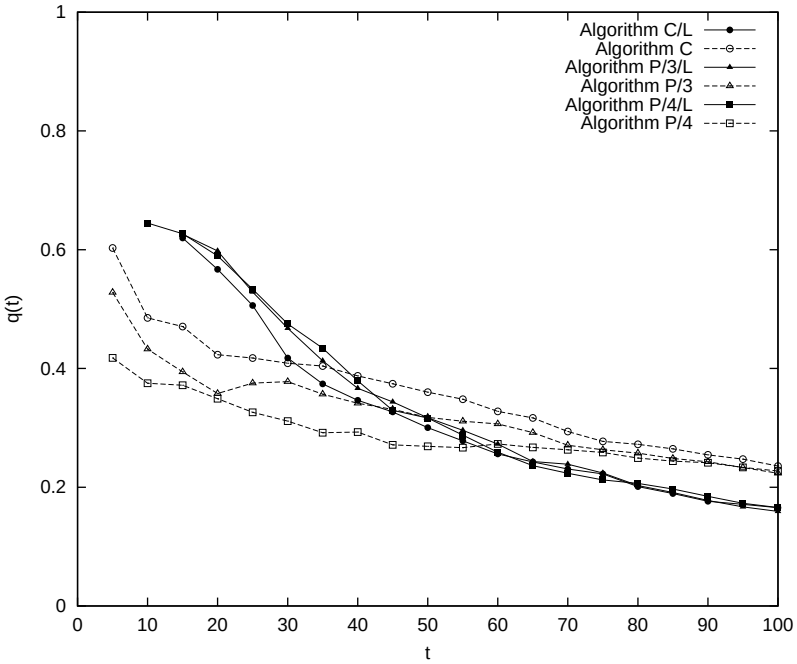


Figure 4.27: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 50$.

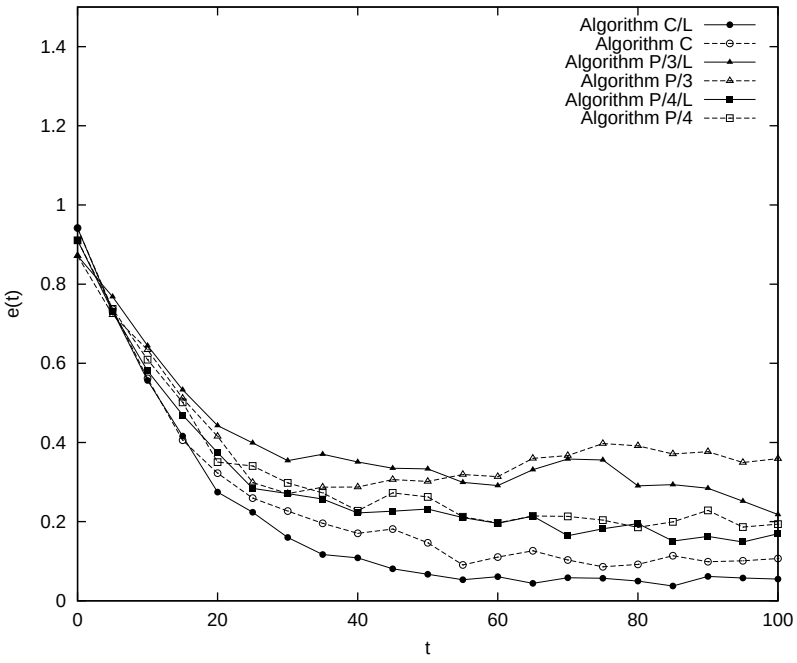


Figure 4.28: Error using Configuration 4A with $\sigma_{E_\theta} = 30°$.

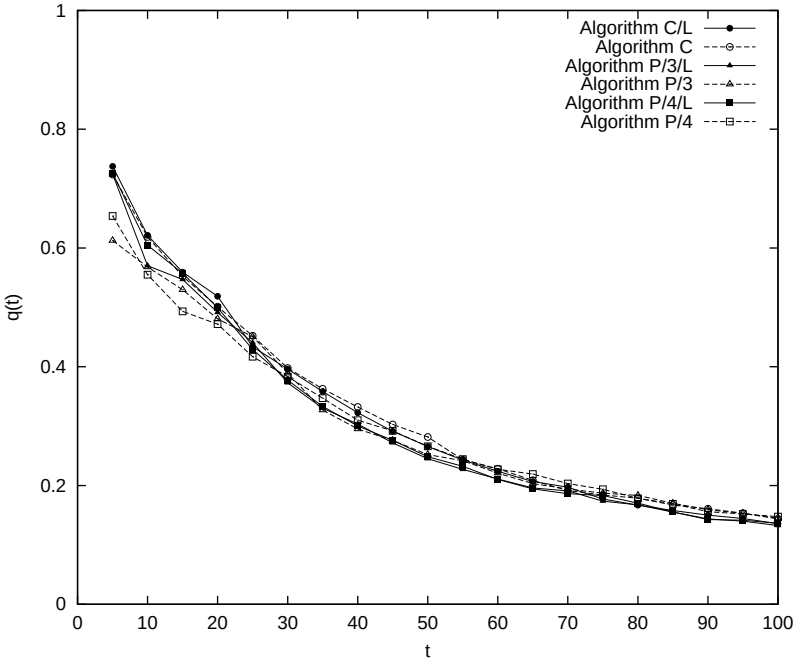


Figure 4.29: Efficiency using Configuration 4A with $\sigma_{E_\theta} = 30°$.

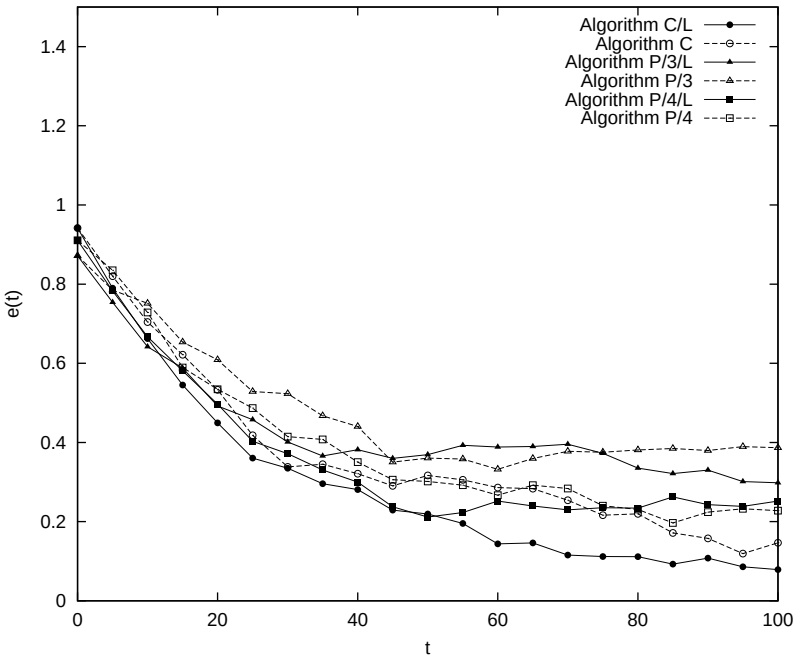


Figure 4.30: Error using Configuration 4A with $\sigma_{E_\theta} = 60°$.

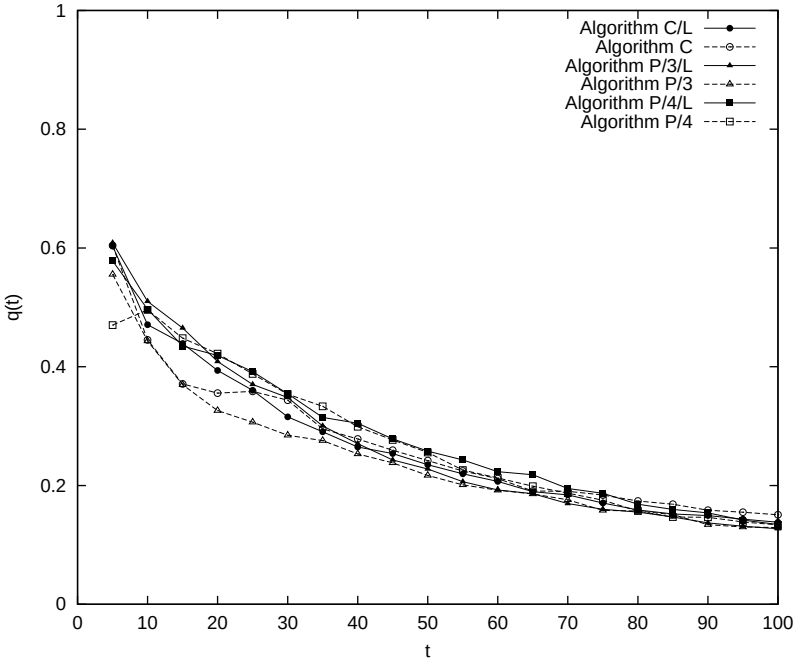


Figure 4.31: Efficiency using Configuration 4A with $\sigma_{E_\theta} = 60°$

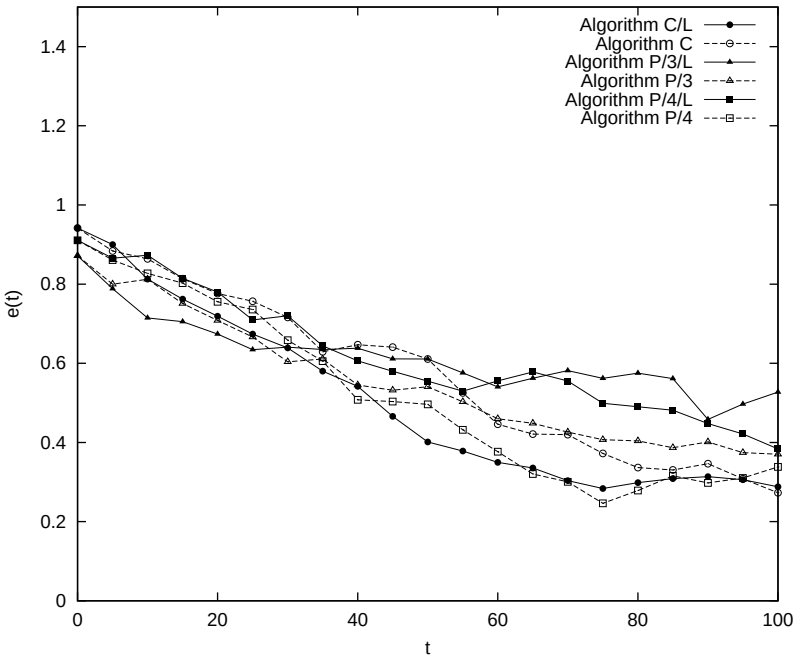


Figure 4.32: Error using Configuration 4A with $\sigma_{E_\theta} = 90°$.

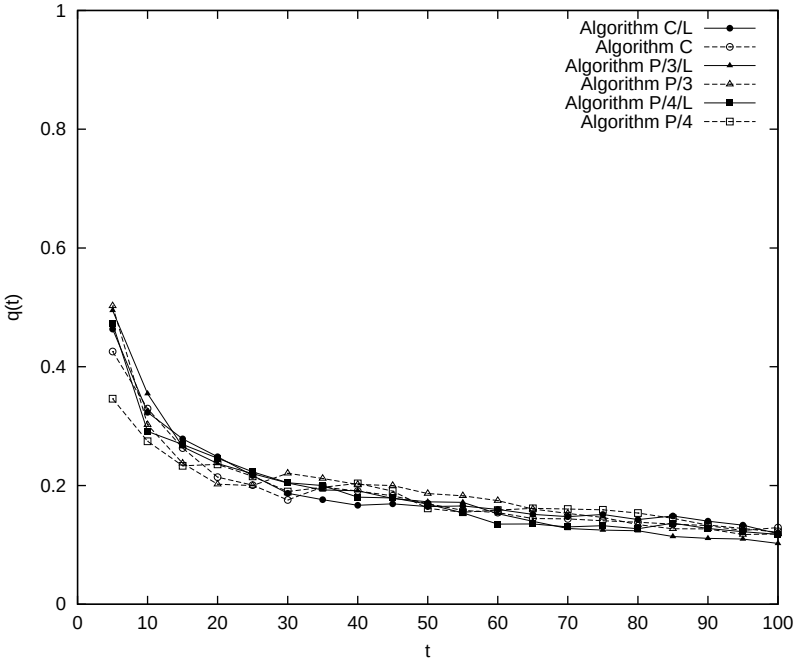


Figure 4.33: Efficiency using Configuration 4A with $\sigma_{E_\theta} = 90°$.

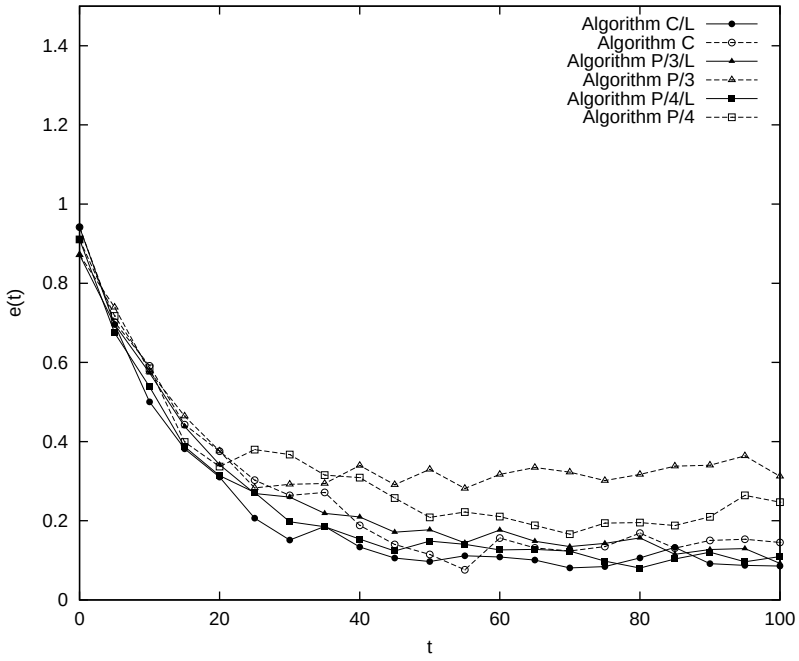


Figure 4.34: Error using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.05$.

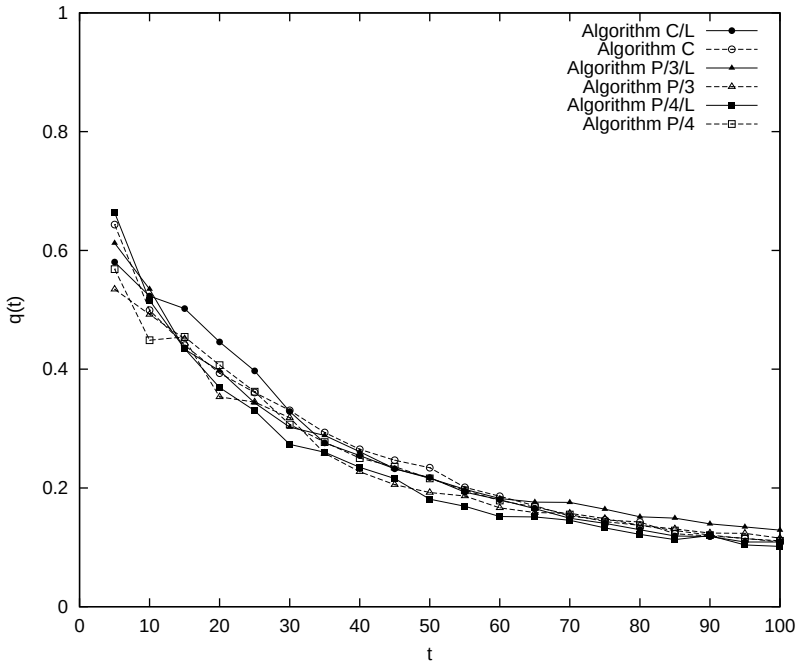


Figure 4.35: Efficiency using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.05$.

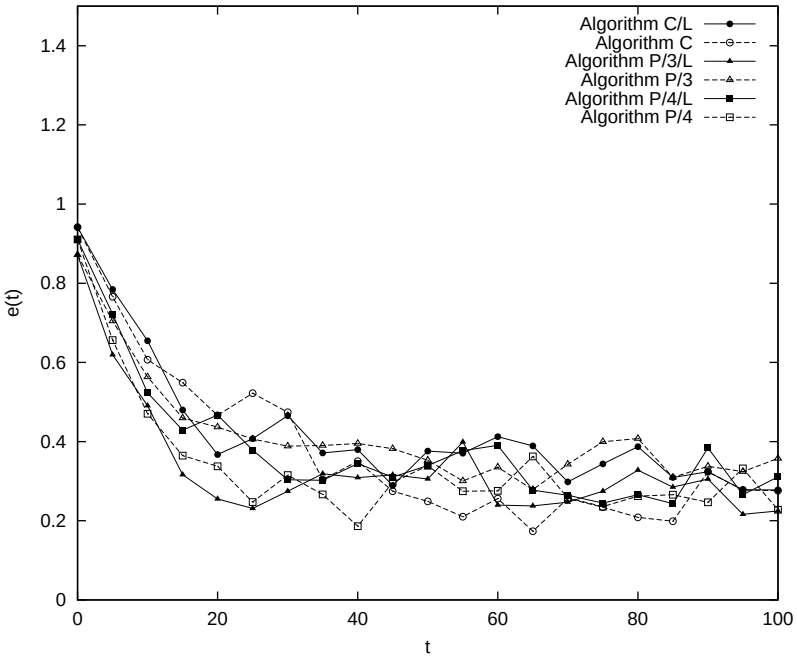


Figure 4.36: Error using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.1$.

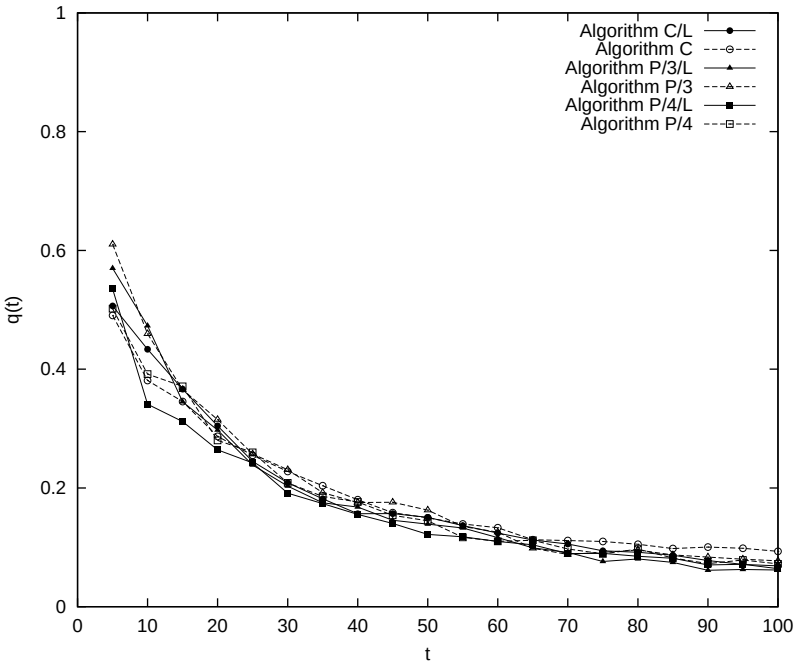


Figure 4.37: Efficiency using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.1$.

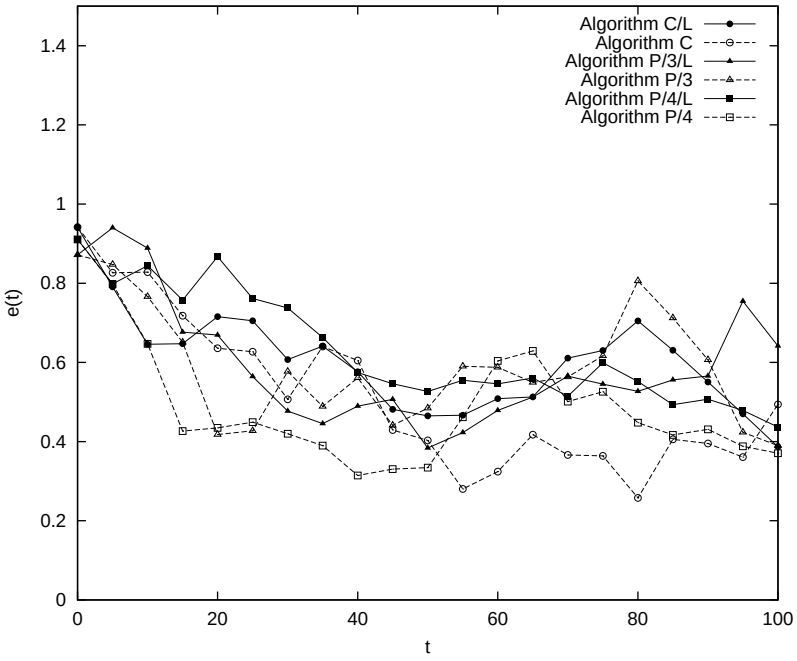


Figure 4.38: Error using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.15$.

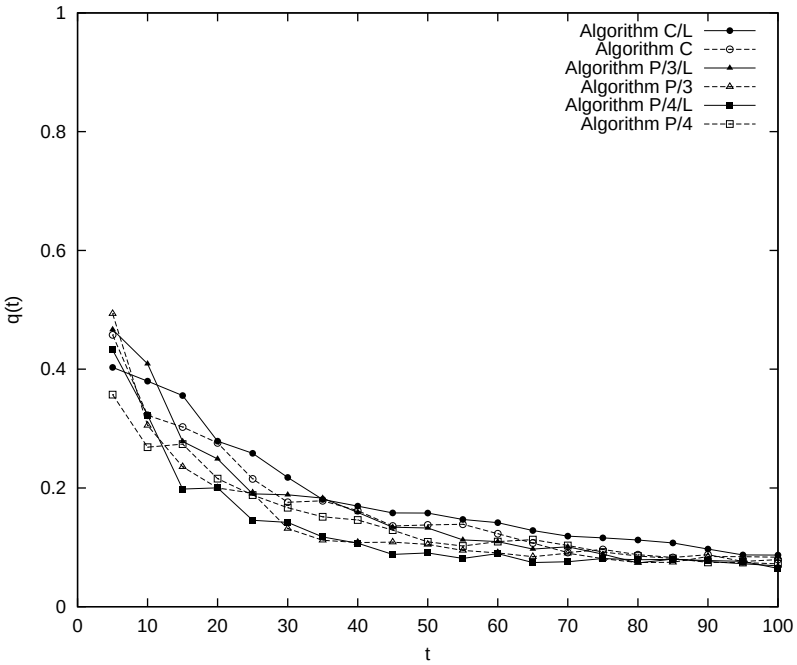


Figure 4.39: Efficiency using Configuration 4A with $\sigma_{E_{V_x,V_y}} = 0.15$.

**Robustness Against All Types of Error Combined** Table 4.1 gives the values of the error parameters for these simulations. Figures 4.40, 4.42 and 4.44 show the error for each algorithm using Configuration 4A with Settings A, B and C, respectively. Figures 4.41, 4.43 and 4.45 show the efficiency for each of these simulations.

The impact on error which the shape had with most of the other error settings was clearly evident for Settings A and B. With the landmark-based algorithms the nodes failed to approach their desired positions with Setting C, while with the centroid-based algorithms the nodes failed to approach their desired positions with Settings B and C.

## 4.4 Conclusion

In this chapter we presented algorithms for forming a line, circle and regular polygon with mobile nodes beginning from random positions. These algorithms are distributed and only use locally obtained sensor information, making them suitable for implementation in MWSNs, where communication is costly in terms of energy consumption.

We have demonstrated using simulations that these algorithms will perform with unreliable sensor information and imprecise control of motion. Occlusion is an issue that needs to be addressed when motion planning is based on the centroid of the node positions. When forming a regular polygon, a node will not be able to locate other nodes on the same side of the shape beyond its immediate neighbours. This results in the positions of some nodes being omitted in the calculation of the relative position of the centroid. There are two approaches to this problem:

- *Memorise* the last known position and velocity of nodes before they become occluded and include an estimate of the position in the calculation of the centroid while the node is hidden from view.

- Use *local knowledge*, such as the positions of immediate neighbours, to coordinate the motion of the nodes. This may be combined with *global knowledge*, such as the centroid of the nodes' positions, in which case the optimal balance between local knowledge and global knowledge for coordinating the motion of the nodes should be determined.

A conclusion to this thesis' work will be presented in the following chapter.

| Setting | $\sigma_{E_{r,g,b}}$ | $\sigma_{E_\theta}$ | $\sigma_{E_{V_x,V_y}}$ |
|---|---|---|---|
| A | 30 | 10° | 0.025 |
| B | 40 | 15° | 0.05 |
| C | 50 | 20° | 0.075 |

Table 4.1: Settings used to test the robustness of the algorithms when image noise, error in the orientation of the cameras and velocity error are present in the system.

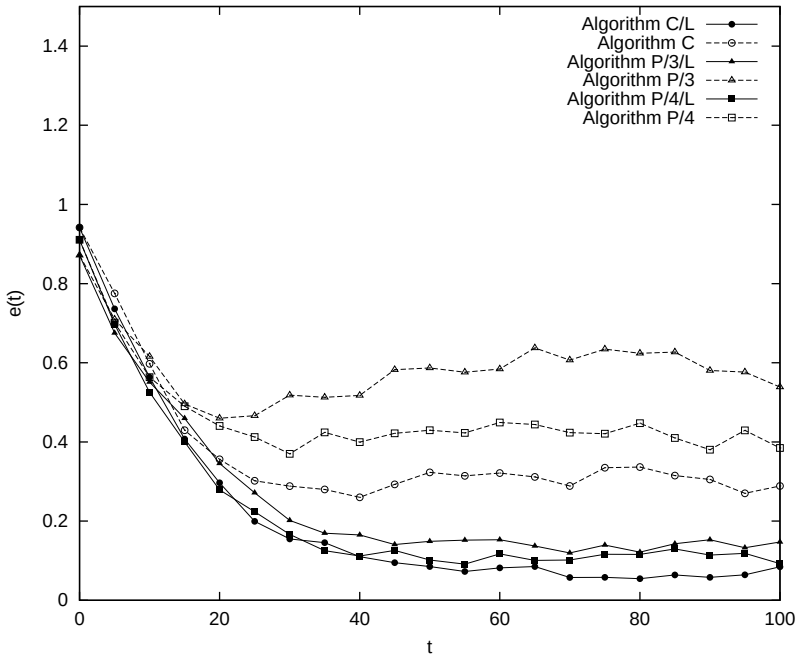


Figure 4.40: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

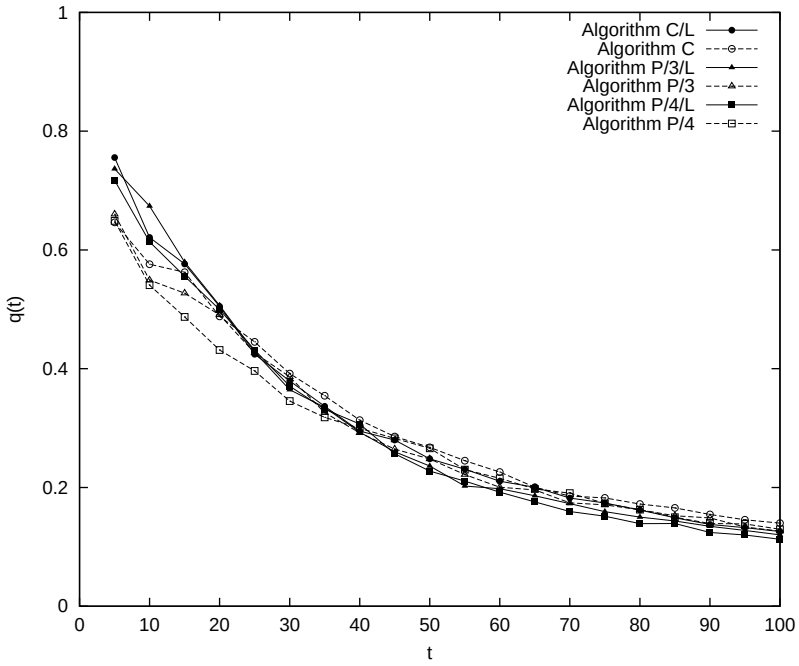


Figure 4.41: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 30$, $\sigma_{E_\theta} = 10°$ and $\sigma_{E_{V_x,V_y}} = 0.025$.

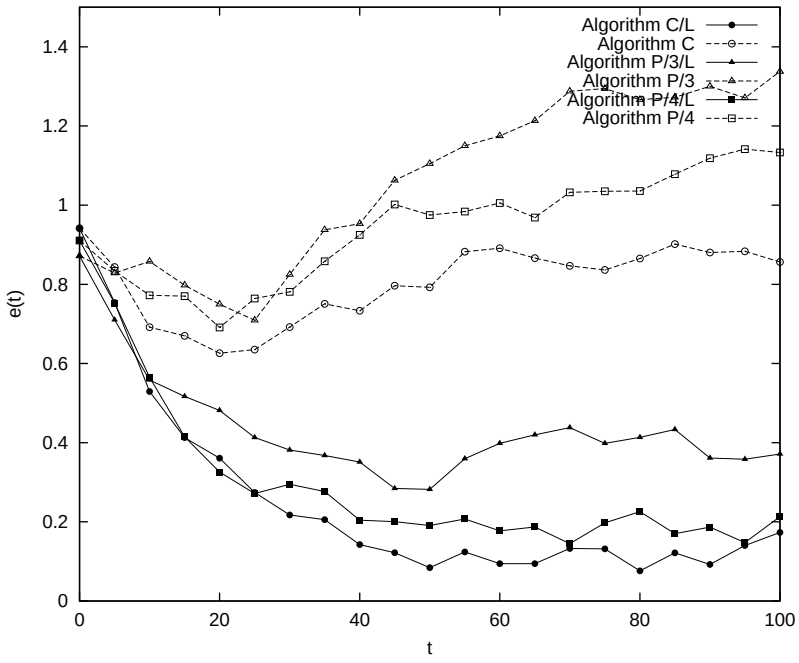


Figure 4.42: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15°$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

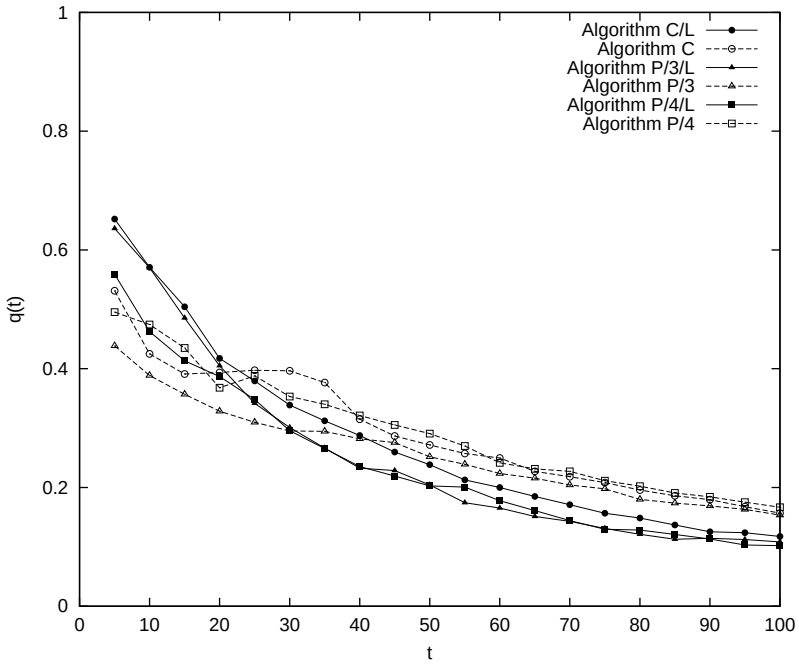


Figure 4.43: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15°$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

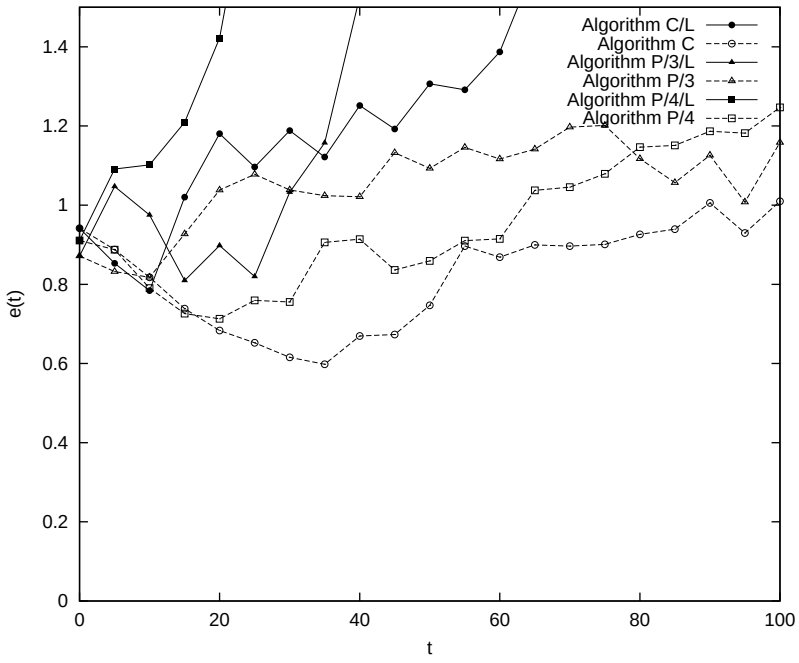


Figure 4.44: Error using Configuration 4A with $\sigma_{E_{r,g,b}} = 50$, $\sigma_{E_\theta} = 20°$ and $\sigma_{E_{V_x,V_y}} = 0.075$.

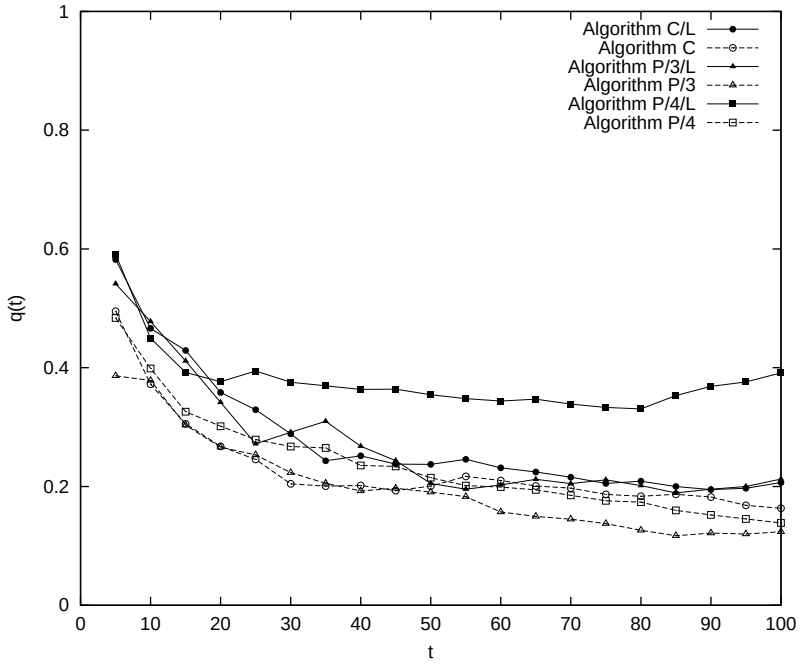


Figure 4.45: Efficiency using Configuration 4A with $\sigma_{E_{r,g,b}} = 50$, $\sigma_{E_\theta} = 20°$ and $\sigma_{E_{V_x,V_y}} = 0.075$.

# Chapter 5

# Conclusion

In Chapter 3, the problem of maintaining the geometric pattern of an MWSN was studied. Algorithms for the following geometric problems were presented:

1. Maintaining a constant separation between neighbouring nodes arranged in a line.
2. Maintaining an equal separation between neighbouring nodes arranged in a line while the length of the line remains constant.

The algorithms for both problems also align the nodes so that they all have the same $y$-coordinate, assuming that the desired line runs parallel to the $x$-axis in the Cartesian plane. No wireless communication is used, nor do the nodes need any form of identification to make them distinguishable. Two solutions for the first problem were presented and compared: a greedy algorithm, Algorithm CS/G, and a state-based algorithm, Algorithm CS/SB. Simulations demonstrated that Algorithm CS/G is more robust against image noise, while Algorithm CS/SB is more efficient for low levels of image noise and camera orientation error. The algorithms can be used in tasks where an area must be swept to search for objects or map a continuous series of data, as in geophysical mapping. They can also be used to maintain a cordon for tasks such as guarding an entrance.

In Chapter 4, the problem of forming a geometric pattern with an MWSN from a set of random positions was studied. It was shown that Algorithms CS/G and CL introduced in Chapter 3 can also be used to *form* a line from a given set of random positions. Algorithms for the following geometric problems were presented:

1. Formation of a circle centred at a landmark, Algorithm C/L.
2. Formation of an $n$-sided polygon centred at a landmark, Algorithm P/$n$/L.

3. Formation of a circle without any specified final location, Algorithm C.

4. Formation of an $n$-sided polygon without any specified final location, Algorithm P/$n$.

As with the algorithms presented in Chapter 3, no wireless communication is needed, nor must the nodes be distinguishable. Occlusion did affect the final positions of the nodes for Algorithm P/$n$, but generally the algorithms performed quite well. The results show that the centroid of the nodes' positions provides a robust global reference for coordinating their motion. Thus robust algorithms for forming more complex patterns based on the centroid can be developed. The pattern formation algorithms presented in this chapter have many potential defence and security applications where an MWSN must form a geometric pattern to carry out a reconnaissance or security task.

One of the main aims of this thesis was to demonstrate that a global objective can be achieved without communication using a *distributed* approach. Not only do distributed algorithms reduce energy consumption because less communication is needed between the nodes, but they are inherently more robust and scalable. For this reason distributed algorithms have been advocated throughout the literature. The algorithms that have been presented, which are robust yet simple, demonstrate the effectiveness of this approach.

## 5.1 Future Work

A couple of problems follow naturally from our study. One is to form an arbitrary shape defined by a relatively large landmark. The nodes begin from random positions outside the landmark and move to enclose the landmark so that each node is a specified distance from the perimeter of the landmark and neighbouring nodes are equally separated. An algorithm for this task would be useful in security applications where an MWSN guards a building, for example.

The second problem is for the nodes to form an arbitrary shape on their own, beginning from random positions. Some of the nodes which are either predetermined or self-elected move to the corners of the desired shape. The remaining nodes position themselves between corner nodes such that neighbouring nodes are equally separated.

A suggestion for solving the above problems and others that might be proposed is to base the motion of the nodes on some model of a mechanical system. For example, the nodes could be modelled as charges having a repulsive force between them with an

attractive force like that of a spring existing between nodes that should neighbour each other once the desired formation is established.

Another suggestion which could result in simple yet more effective algorithms is to *train* the nodes to form the desired pattern through demonstration rather than manually enter data describing the pattern, which could be quite unwieldly. The user positions the nodes at the desired locations and each node generates data characterising its relationship with other nodes using only its own local information. For example, each node could calculate its position and bearing from the centroid of only the *known* positions of other nodes. The nodes then broadcast this information to the other nodes so that all nodes have the geometric information for all the desired positions. This approach would overcome the problem of occlusion which was particularly evident for Algorithm P/3 because the correct distance from the centroid of only the *visible* nodes can be determined.

# Bibliography


[1] http://www.isi.edu/scadds/, August 2003.

[2] http://www.janet.ucla.edu/WINS/, August 2003.

[3] http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/, August 2003.

[4] *Persistence of Vision Ray-Tracer POV-Ray Version 3.1g User's Documentation*, May 1999.

[5] J. Agre and L. Clare. An integrated architecture for cooperative sensing networks. *Computer*, 33(5):106–108, May 2000.

[6] J. R. Agre, L. P. Clare, G. J. Pottie, and N. P. Romanov. Development Platform for Self-Organizing Wireless Sensor Networks. In *Unattended Ground Sensor Technologies and Applications*, volume 3713 of *Proceedings of SPIE*, pages 257–268, April 1999.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, March 2002.

[8] T. Arai, D. Kurabayashi, J. Ota, and S. Ichikawa. Motion planning for cooperative sweeping with relocating obstacles. In *Proceedings of the 1996 IEEE International Conference on Systems, Man and Cybernetics*, pages 1513–1518, 1996.

[9] G. Asada, I. Bhatti, T. H. Lin, S. Natkunanthanan, F. Newberg, R. Rofougaran, A. Sipos, S. Valoff, G. J. Pottie, and W. J. Kaiser. Wireless integrated network sensors (wins). In Vijay K. Varadan, editor, *Proceedings of SPIE*, volume 3673, pages 11–18, March 1999.

[10] T. Balch and R. C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[11] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable coordination for wireless sensor networks: Self-configuring localization systems. In *Proceedings of the 6th International Symposium on Communication Theory and Applications (ISCTA '01)*, Ambleside, Lake District, UK, July 2001.

[12] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, March 1997.

[13] Q. Chen and J. Y. S. Luh. Coordination and control of a group of small mobile robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2315–2320, 1994.

[14] Q. Chen and J. Y. S. Luh. Distributed motion coordination of multiple robots. In *Proceedings of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1493–1500, 1994.

[15] L. P. Clare, G. J. Pottie, and J. R. Agre. Self-organizing distributed sensor networks. In Edward M. Carapezza, David B. Law, and K. Terry Stalker, editors, *Unattended Ground Sensor Technologies and Applications*, volume 3713 of *Proceedings of SPIE*, pages 229–237, Orlando, FL, USA, April 1999.

[16] N. Correal and N. Patwari. Wireless sensor networks: Challenges and opportunities. In *Proceedings of the 11th MPRG/Virginia Tech Symposium on Wireless Personal Communications*, June 2001.

[17] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18(5):813–825, October 2002.

[18] X. Défago and A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proceedings of the 2nd ACM Annual Workshop on Principles of Mobile Computing (POMC'02)*, pages 97–104, Toulouse, France, October 2002.

[19] Xavier Défago. Distributed computing on the move: From mobile computing to cooperative robotics and nanorobotics. In *Proceedings of the ACM Workshop on Principles of Mobile Computing (POMC'01)*, pages 49–55, Newport, RI, USA, August 2001.

[20] J. P. Desai, V. Kumar, and J. P. Ostrowski. Control of changes in formation for a team of mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pages 1556–1561, May 1999.

[21] J. P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 2864–2869, May 1998.

[22] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4):375–397, December 1996.

[23] Jeremy Elson and Deborah Estrin. Random, ephemeral transaction identifiers in dynamic sensor networks. In *Proceedings of the 21st International Conference on Distributed Computing*, pages 459–468, Mesa, AZ, USA, April 2001.

[24] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 4, pages 2033–2036, Salt Lake City, UT, USA, May 2001.

[25] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, pages 263–270, Seattle, WA, USA, August 1999.

[26] Deborah Estrin, Ramesh Govindan, and John Heidemann. Embedding the Internet. *Communications of the ACM*, 43(5):38–41, May 2000.

[27] R. Fierro, A. K. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 157–162, Seoul, Korea, May 2001.

[28] R. Fierro, P. Song, A. Das, and V. Kumar. Cooperative control of robot formations. In R. Murphey and P. Pardalos, editors, *Series on Applied Optimization*, chapter 1. Kluwer Academic Press, March 2001.

[29] P. Flocchini, G.Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2000)*, pages 480–485, Dearborn, MI, USA, October 2000.

[30] P. Flocchini, G.Prencipe, N. Santoro, and P. Widmayer. Pattern formation by autonomous robots without chirality. In *Proceedings of the VIII International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001)*, pages 147–162, June 2001.

[31] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Proceedings of the 10th International Symposium on Algorithms and Computation (ISAAC 99)*, volume LNCS 1741, pages 93–102, December 1999.

[32] D. E. Franklin, A. B. Kahng, and M. A. Lewis. Distributed sensing and probing with multiple search agents: toward system-level landmine detection solutions. In *Detection Technologies for Mines and Minelike Targets*, volume 2496 of *Proceedings of SPIE*, pages 698–709, 1995.

[33] J. Fredslund and M. J. Matarić. A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, October 2002.

[34] S. Goldsmith, J. Feddema, and R. Robinett. Analysis of decentralized variable structure control for collective search by mobile robots. In *Sensor Fusion and Decentralized Control in Robotic Systems*, volume 3523 of *Proceedings of SPIE*, pages 40–47, November 1998.

[35] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, pages 1–10, Maui, Hawaii, USA, January 2000.

[36] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185, Seattle, WA, USA, August 1999.

[37] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[38] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000. ISBN 1-58113-197-6.

[39] J. Kahn, R. Katz, and K. Pister. Emerging challenges: Mobile networking for "smart dust". *Journal of Communications and Networks*, 2(3):188–196, September 2000.

[40] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 8 (2-3):169–185, March 2002. ISSN 1022-0038.

[41] D. Kurabayashi, J. Ota, T. Arai, S. Ichikawa, S. Koga, H. Asama, and I. Endo. Cooperative sweeping by multiple mobile robots with relocating portable obstacles. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1472–1477, 1996.

[42] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. An algorithm of dividing a work area to multiple mobile robots. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 286–291, 1995.

[43] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 1744–1749, 1996.

[44] A. Leon-Garcia. *Probability and Random Processes for Electrical Engineering.* Addison-Wesley Publishing Company, second edition, May 1994.

[45] P. Liang and G. Beni. Robotic morphogenesis. In *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, pages 2175–2180, 1995.

[46] T. W. Min and H. K. Yin. A decentralized approach for cooperative sweeping by multiple mobile robots. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 380–385, October 1998.

[47] Y. Moscovitz and N. DeClaris. Basic concepts and methods for keeping autonomous ground vehicle formations. In *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*, pages 44–49, September 1998.

[48] L. E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, pages 582–587, 1993.

[49] L. E. Parker. Current state of the art in distributed autonomous mobile robotics. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems*, volume 4, pages 3–12. Springer-Verlag, Tokyo, Japan, 2000.

[50] K. S. J. Pister. http://www-bsac.eecs.berkeley.edu/~pister/SmartDust/, August 2003.

[51] G. J. Pottie. Hierarchical information processing in distributed sensor networks. In *Proceedings of the 1998 IEEE International Symposium on Information Theory*, August 1998.

[52] G. J. Pottie. Wireless Sensor Networks. In *1998 Information Theory Workshop*, pages 139–140, June 1998.

[53] G. J. Pottie and L. P. Clare. Wireless Integrated Network Sensors: Towards Low Cost and Robust Self-Organizing Security Networks. In *Proceedings of the SPIE International Symposium on Enabling Technologies for Law Enforcement and Security*, volume 3577 of *Proceedings of SPIE*, pages 86–95, Boston, MA, USA, November 1998.

[54] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[55] G. J. Pottie, W. J. Kaiser, L. P. Clare, and H. O. Marcy. Wireless integrated network sensors, September 1998. UCLA Technical Report.

[56] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. In *Proceedings of the Fourth European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, pages 185–190, May 2001.

[57] G. Prencipe and V. Gervasi. On the intelligent behavior of stupid robots. In *Proceedings of the AI*IA Workshop on Robotics (GLR)*, September 2002.

[58] H. Qi, S. S. Iyengar, and K. Chakrabarty. Distributed sensor networks-a review of recent research. *Journal of the Franklin Institute*, 338(6):655–668, September 2001.

[59] J. M. Rabaey, M. J. Ammer, J. L. da Silva Jr., D. Patel, and S. Roundy. Picoradio supports ad hoc ultra-low power wireless networking. *Computer*, 33(7):42–48, July 2000.

[60] P. Rentala, R. Musunuri, S. Gandham, and U. Saxena. Survey on sensor networks.

[61] P. Rudakevych, H. Greiner, and B. Pletta. Micro unattended mobility system (mums). In Edward M. Carapezza, David B. Law, and K. Terry Stalker, editors, *Unattended Ground Sensor Technologies and Applications*, volume 3713 of *Proceedings of SPIE*, pages 142–159, Orlando, FL, USA, April 1999.

[62] C.-C. Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8(4):52–59, August 2001.

[63] Eugene Shih, Seong-Hwan Cho, Nathan Ickes, Rex Min, Amit Sinha, Alice Wang, and Anantha Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *Proceedings of the seventh annual international conference on Mobile computing and networking*, pages 272–286. ACM Press, July 2001. ISBN 1-58113-422-3.

[64] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, October 2000.

[65] Katayoun Sohrabi and Gregory J. Pottie. Performance of a novel self-organization protocol for wireless ad hoc sensor networks. In *Proceedings of the IEEE 50th Vehicular Technology Conference*, pages 1222–1226, September 1999.

[66] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems*, 13(3):127–139, March 1996.

[67] K.-H. Tan and M. A. Lewis. Virtual structures for high-precision cooperative mobile robotic control. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 132–139, November 1996.

[68] M. Teruya, Z. Nakao, Y. W. Chen, F. E. A. F. Ali, K. Matsuo, and T. Miyagi. A boid-like example of adapative complex systems. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 266–270, 1999.

[69] P. K. C. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[70] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, January 2001.

[71] H. Yamaguchi. Adaptive formation control for distributed autonomous mobile robot groups. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 2300–2305, April 1997.

[72] H. Yamaguchi. A cooperative hunting behavior by multiple nonholonomic mobile robots. In *Proceedings of the 1998 IEEE International Conference on Systems, Man and Cybernetics*, pages 3347–3352, 1998.

[73] H. Yamaguchi. A cooperative hunting behavior by mobile-robot troops. *The International Journal of Robotics Research*, 18(8):931–940, September 1999.

[74] H. Yamaguchi and J. W. Burdick. Asymptotic stabilization of multiple nonholonomic mobile robots forming group formations. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 3573–3580, May 1998.

[75] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, USA, June 2002.

[76] L. Zhong, J. Rabaey, C. Guo, and R. Shah. Data link layer design for wireless sensor networks. In *Proceedings of IEEE MILCOM 2001*, volume 1, pages 352–356, Washington DC, USA, October 2001.

[77] L. C. Zhong, R. Shah, C. Guo, and J. Rabaey. An ultra-low power and distributed access protocol for broadband wireless sensor networks. In *IEEE Broadband Wireless Summit*, Las Vegas, NV, USA, May 2001.