SELF-ORGANISING AN INDOOR LOCATION SYSTEM USING A PAINTABLE AMORPHOUS COMPUTER

By
John David Revill
B.Sc.(Hons)

A thesis submitted for the degree of
Doctor of Philosophy

Faculty of Engineering, Science and Mathematics,
School of Electronics and Computer Science,
University of Southampton,
United Kingdom.

June 2007

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Self-Organising an Indoor Location System using a Paintable Amorphous Computer

by John David Revill

This thesis investigates new methods for self-organising a precisely defined pattern of intertwined number sequences which may be used in the rapid deployment of a passive indoor positioning system's infrastructure.

A future hypothetical scenario is used where computing particles are suspended in paint and covered over a ceiling. A spatial pattern is then formed over the covered ceiling. Any small portion of the spatial pattern may be decoded, by a simple camera equipped device, to provide a unique location to support location-aware pervasive computing applications.

Such a pattern is established from the interactions of many thousands of locally connected computing particles that are disseminated randomly and densely over a surface, such as a ceiling. Each particle has initially no knowledge of its location or network topology and shares no synchronous clock or memory with any other particle.

The challenge addressed within this thesis is how such a network of computing particles that begin in such an initial state of disarray and ignorance can, without outside intervention or expensive equipment, collaborate to create a relative coordinate system. It shows how the coordinate system can be created to be coherent, even in the face of obstacles, and closely represent the actual shape of the networked surface itself. The precision errors incurred during the propagation of the coordinate system are identified and the distributed algorithms used to avoid this error are explained and demonstrated through simulation.

A new perimeter detection algorithm is proposed that discovers network edges and other obstacles without the use of any existing location knowledge. A new distributed localisation algorithm is demonstrated to propagate a relative coordinate system throughout the network and remain free of the error introduced by the network perimeter that is normally seen in non-convex networks. This localisation algorithm operates without prior configuration or calibration, allowing the coordinate system to be deployed without expert manual intervention or on networks that are otherwise inaccessible.

The painted ceiling's spatial pattern, when based on the proposed localisation algorithm, is discussed in the context of an indoor positioning system.

Contents

List of Figures

List of Tables

DECLARATION OF AUTHORSHIP

I, John David Revill, declare that the thesis entitled

Self Organising an Indoor Location System
using a Paintable Amorphous Computer

and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research.

I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- none of this work has been published before submission.

Signed:

Date:

Acknowledgements

Thanks go to David De Roure for his supervision, far reaching insight and for his endurance of a long running part-time student. Thank you also to Hugh Glaser for his feedback and for sparking the idea of ceiling painted gunk.

Many thanks to Danius Michaelides for all his insightful feedback and time spent running the thesis through a much needed sanity check.

I would also like to thank both IBM and the UK EPSRC for their funding support of this work.

My most sincere and heartfelt gratitude must go to Rachel Quinn, who has endured my relentless writing worries, whilst simultaneously offering unwavering support and optimism.

Finally, thank goodness, it is now all over.

Chapter 1

Introduction

Computing devices have been continually decreasing in cost and size over the years. Past perceptions of computers have changed from the large mainframes that serve many people, to the current personal computer serving just one person. Already it is the case that people own several relatively small computing devices, and even carry them around with themselves. A future is envisioned in which these devices become so small and abundant that they could be embedded and available within the environment. Computing devices in this future are likely to be such a common and accepted technology in society that they no longer become perceived as computers, but as a pervasive and imperceptible infrastructure supporting the people they surround. Pervasive computing strives toward a natural interaction between humans and such computing environments. The technology to support this interaction is already widely available in the form of handheld and wearable computers, ad hoc wireless communication links, and location sensing systems.

In order for the computational infrastructure to assist a person with their tasks it must reason correctly about that person's current context. One large defining factor of a person's context is their location. A well known example of applications with location awareness assisting people are the GPS-equipped navigational aids that are used in cars. These devices are well suited to outdoor use, but GPS can not be used to support location awareness indoors due to the poor signal received from the satellite. A range of different technologies appeared to cover this area, each with their own characteristics, yet no one single technology is suitable for all applications. Often, the technologies can be very expensive and require careful calibration.

Recently there has been an effort to produce the smallest computing devices possible, with their approaches ranging from microelectromechanical systems, through to cellular based computing, through to nanotechnology. Current CPUs suffer from high costs for manufacturing as they must be free of any defects. One of the possible effects from these miniaturisation efforts is that they can become extremely cheap to manufacture under mass production if the requirement for no defects is relaxed and the testing of individual units is skipped.

Each miniaturisation approach typically has the challenge of providing the devices with a means of communication with other devices. A networked collection of these devices provides the potential for massively parallel execution. With such large numbers and small size, dense networks of these devices can ignore individual defective units and maintain the network's correct operation through the redundancy of the numbers involved. Rather than being a problem, it can be an accepted norm that some devices will be faulty.

At some point, devices are likely to become cheap and small enough that it is conceivable that they could be suspended within paint and used to cover an entire surface, such as a ceiling, forming a network of many thousands of nodes. This thesis leapfrogs the current research into overcoming hardware constraints of miniaturisation and looks at such a possible theoretical computational medium to provide an indoor positioning system. It looks at the scenario where an amorphous computer is painted onto a ceiling and individual nodes, as their last sacrificial action before running out of power, may choose to discharge some ink in their locality. The ink marks the node's location within the entire painted network. With the use of a simple camera phone or other portable computer, the marks on the ceiling can be identified and translated into a location that provides context for an indoor pervasive computing application.

This thesis looks at a class of spatial patterns that are used in digital pens and paper enriched with watermarks. It discusses how they can be formed on a painted amorphous ceiling and what characteristics the resulting positioning system has.

One of the appealing features of this approach to indoor positioning is the potential for its rapid deployment and the unimportance of any training or expert configuration. To retain this feature, the sole configuration is assumed to be just an initial commencement of the pattern formation after the surface is fully painted. Perhaps this jump-start could be signalled by using a probe device or placing a special bootstrap node at some location in the network, to communicate with some nodes in a single area. No manual configuration is assumed other than the signalling for the pattern formation process to begin.

As no nodes in the amorphous computer begin with any location knowledge or even any knowledge of the network's topology, forming the pattern requires that the nodes first discover their location. Existing solutions to forming coordinate systems amongst many nodes in a network have typically used specially placed and configured reference nodes to facilitate the propagation of coordinates throughout the network. Other solutions have formed coordinate systems without reference nodes and solely from local connectivity, but these perform poorly in the face of obstructions in the network. It is this self-discovery of node's locations, without any pre-configuration, and in the face of obstructions within the network, that forms the crux and contribution of the thesis.

With no hardware and no equivalent for looking at this phenomena on such a large scale, simulations are used as the best expedient to explore this hypothetical scenario of the future.

The main research contributions of this thesis are:

- A new distributed edge detection algorithm that requires no location knowledge
- Identification and avoidance of the hop-based ranging error introduced by the network perimeter and other obstacles within a dense concave network
- A distributed localisation algorithm that requires no configuration, predetermined reference points or location knowledge at all. Specifically, a coordinate system born of this localisation algorithm can retain a shape closer to the actual network layout even when significant obstacles are present in the network or if the network's layout is concave.
- An outline of a new spatial addressing code that may be used in an indoor positioning system. This new code is a cross pollination of existing spatial codes in order to make them more suitable for use on a pattern of lower precision as formed by an amorphous computer.

The remainder of this thesis is organised as follows:

Chapter 2 introduces the different areas of research that are drawn upon in this thesis and explains the primary concepts involved. Specifically, it looks at amorphous computing, approaches to localisation within sensor networks, and positioning systems in pervasive computing with a focus on indoor infrastructures and their techniques.

In chapters 3 and 3, the challenges for localisation in such networks is identified, and we present the approaches and algorithms proposed to address these challenges. The formation of a relative coordinate system, over the nodes in the network, utilising these algorithms is detailed.

Three existing spatial patterns that provide positioning information, used in digital pen and paper applications, are explained in detail in chapter 6. Particular attention is given to the surface area covered by these codes and to the form of the encoding marks, or glyphs. A new code is described that both covers a large surface area and is more suitable to a ceiling based pattern comprised of glyphs that are not formed with a great degree of precision. The operation and properties of a low cost indoor positioning system based on this new code is outlined in the latter section of this chapter.

The simulation experiments of the both algorithms proposed in chapters 3 and 4 are explored in chapter 5. The simulation results are further evaluated in the concluding chapter 7 together with a critical analysis. Directions and possibilities for future work are explored and the chapter concludes with a summary of the work in this thesis.

Chapter 2

Background

This chapter situates this research within its wider context, namely: pervasive computing, localisation in sensor networks, and amorphous computing. Related literature and developments are outlined and the main concepts used in this thesis are described.

2.1 Pervasive computing and indoor positioning systems

Pervasive computing, as an area of research, came into being in 1991 when Weiser introduced Ubiquitous Computing (Weiser 1991). He envisioned people interacting naturally with computationally rich environments that provide them with information when and where it is required. This vision sees the computing environment move on from the current realm of the immobile personal computer atop a desk, to being distributed throughout the environment. At the time, Weiser and his colleagues brought about the distributed computational environment in the form of a number of devices. These devices ranged in scale from small personal hand-held devices to large smart whiteboards (Weiser 1993). However, since then mobile hardware and wireless networking have improved significantly, allowing devices like PDAs and mobile phones to be locally connected and inter-operate in an ad hoc fashion, through the use of technologies such as Bluetooth and wireless ethernet. These mobile devices together with pervasive computing will permit people to casually roam from place to place whilst transparently or unconsciously drawing upon, and being assisted by, unseen computing devices distributed in the environment.

The interaction between human and computer fundamentally changes when the computing resources become distributed in this fashion. The traditional GUI model involving a keyboard, mouse and screen needs to shift towards a more natural interaction, more akin to the way humans interact with the everyday physical world. Interfaces that utilise the more natural forms of human communication, such as speech and handwriting, are now slowly appearing as alternatives to some of the more traditional GUI methods. At some point, we can expect people to stop perceiving these resources as computers at all. Indeed, Weiser (1991) writes that "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it".

Human computer interaction (HCI) forms a large area of research in pervasive computing. However, with the projected trend over time of increasing numbers of computers per person – from the past of one mainframe to many people, through to one personal computer, through to the current and future multitude of devices per person – there has been an increasing focus on reducing the need for explicit user action. Proactive computing (Tennenhouse 2000) and autonomic computing (Parashar & Hariri 2004) both have similar aims in this respect. A typical example is a home heating system which adjusts the temperature and energy use in a proactive manner beyond that of a simple thermostat's homeostasis. Proactive heating decisions can be based on knowledge gained through contextual sensors such as the occupant's schedule or known typical room usage (Want et al. 2003). The heater may switch itself on shortly before the occupier is expected back each day, but if they are scheduled to be away on holiday that day then the heater could remain off.

2.1.1 Context Awareness

Mobile pervasive computing applications need to perform effectively within changing environments and in their communication with other devices. To be effective and useful to the user, an application must discover and reason about its current proximate environment, or context. Dey & Abowd (2000) offer a definition of the context to be:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Dey & Abowd (2000), and earlier by Ryan et al. (1998), also put forward that four contexts are more important than the others: identity, activity, time and location. Any other contextual information is secondary due to its dependence on one or more of these principal contexts. This contextual information is critical for context-aware applications (Schilit et al. 1994, Hull et al. 1997*a*, Pascoe 1998) to reason about why and what the user is doing, in order to provide relevant and useful services to them.

With people's identity, further useful information can be sought, such as phone numbers, occupations, relationships to other people in the environment, and so on.

A person's activity needs to be interpreted in order to be useful. Additionally, the reasons why the person is carrying out the activity is also useful. This is particularly difficult to capture as it involves the persons internal cognition; nevertheless, sensing the physiological state of the person, such as galvanic skin response, heart rate and body temperature may give an indication as to their emotional state (Picard 1997). The activity may be interpreted by relying on combinations of other contextual information (called context fusion) such as location and time.

The date and time is useful when combined with a person's schedule of events and provides a valuable indicator of what a person is, or normally would be, doing during the day.

The passing of time, such as the duration a person stays in one location, is also useful to an application. For example, a context-aware museum may offer additional information about an exhibit to a person if they stand near to it for some time. Conversely, the museum may deduce that the person is uninterested in the exhibit if they spend very little time near it.

Location is arguably the context of chief importance; it is the most useful and disambiguating element of context awareness so we will elaborate further on the existing positioning systems in the following section 2.1.2. Location awareness consists of absolute or relative position information, orientation and/or a symbolic name such as 'Museum Exhibit 4'.

2.1.2 Positioning Techniques and Existing Location Systems

There are generally three different techniques for determining position: triangulation, scene analysis and proximity (Hightower & Borriello 2001). This section describes the better known existing systems employing them.

2.1.2.1 Positioning Techniques

Triangulation may be further partitioned into the more accurate terms of lateration and angulation. By measuring the distance of an object from three different known locations, the object's two-dimensional location may be calculated by trigonometry; this is known as lateration. On the other hand, angulation determines position using trigonometry after having measured the angles from two known locations.

Scene analysis identifies distinguishing physical landmarks within an image taken of the physical environment. As the locations of the landmarks are known, the relative location of the observer can be determined. Unfortunately, scene analysis often requires intensive computation to recognise landmarks and their orientation, which typically can limit or prohibit its use on small devices such as mobile phones.

The presence and proximity of objects can be used to determine location. Physical contact with a pressure pad, discovery of a person's Bluetooth device in the local area, and a person's login to a desktop computer are all examples of location sensing using proximity techniques.

2.1.2.2 Indoor Positioning Systems

Probably the best known location system is the Global Positioning System. GPS uses the time-of-flight, or propagation delay, of radio waves from a satellite to a receiver as a measurement of distance. This distance and those from other satellites are used in determining the receiver's three-dimensional position by lateration, with a typical accuracy on the planet's surface to within a few metres. Unfortunately, GPS is restricted to outdoor use only due to severe multi-path reflection and poor penetration of the signal indoors. This section introduces a number of different positioning systems that operate indoors.

There are a number of systems that use the time-of-flight of radio waves indoors, such as (Bahl & Padmanabhan 2000, Saha et al. 2003, Youssef et al. 2003). One such system from Microsoft Research is RADAR (Bahl & Padmanabhan 2000) which uses a combination of wireless network cards and base stations to locate an individual network card's two dimensional location by lateration. RADAR is also capable of positioning by scene analysis which uses a trained map of the signal strength over a regular lattice of locations in the coverage area. A central controller uses the signal strength readings from all the base stations in the area to find the closest corresponding signal strength measurement in the map to find the location. Like GPS, determining the orientation is not intrinsic to RADAR and may only be estimated from the vector of recent movements. The precision and accuracy tends to vary; RADAR is accurate to within 3 to 4.3 metres, 50% of the time. The theoretical fundamental limit of precision of such wireless ethernet based systems is thought to be approximately 3m (Elnahrawy et al. 2004).

Several existing systems use ultrasound time-of-flight lateration techniques (Ward et al. 1997, Priyantha et al. 2000, Randell & Muller 2001). Ward et al.'s (1997) Active Bats system extends earlier work in mobile robot positioning (Figueroa & Mahajan 1994, Doussis 1993) and uses a fixed lattice of ceiling mounted ultrasound receivers with known locations, placed 1.2m apart. These receivers are connected to a central controller which is responsible for calculating positions and tracking the mobile entities. Small mobile transmitters called Bats produce short ultrasonic pulses in response to a received RF communication from the controller. The ultrasound pulses are sensed by the ceiling mounted infrastructure and passed along the controller to calculate the Bat's three dimensional position by lateration. Orientation is obtained either by attaching two bats to one object and computing the relative positions of each, or to an extent by analysing the conical ultrasound pattern received from different receivers. The calculated position is accurate to within 3cm 95% of the time, with a possible update rate of 50hz (Addlesee et al. 2001).

Priyantha et al.'s (2000) Cricket system works in a similar way to Active Bats, but the method is reversed; the receivers and transmitters are transposed. Each ceiling mounted transmitter periodically emits a synchronised RF signal and ultrasound pulse. Whilst this requires the moving receiver to calculate the position from the ceiling mounted transmitters, the infrastructure is unaware of the identity and locations of any mobile entities. The mobile entity itself is responsible for calculating the three dimensional position from received ultrasound pulses, which ensures their location is kept private and does not require the infrastructure to be trusted. The orientation is calculated by using two receivers fixed on one mobile entity. The Cricket system has a location accuracy to within 10cm and an orientation accuracy to within 3 degrees (Priyantha 2005).

Randell & Muller (2001) describe a low cost system similar to Cricket with just four ceiling mounted transmitters achieving a precision between 10 and 25cm, updated several times per second. The Pinger is another low cost system that provides proximity information (Hull et al. 1997*b*). The Pinger is a transmitter that intermittently sends a short identifying tag encoded in an RF signal over a range of 3 metres.

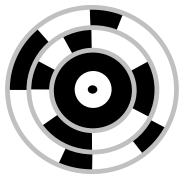
Active Badges (Want et al. 1992, Harter & Hopper 1994) are small tags worn by people that flood the room with unique infrared signals every 10 seconds. These infrared signals are detected by a sensor attached to the room, which will know the location of the tag and its associated owner because of its proximity. The accuracy granularity is tied to the area of the room itself.

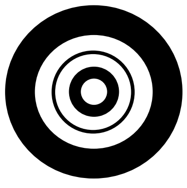
The EasyLiving vision system (Brumitt et al. 2000) uses stereo-vision 3D cameras in every room to analyse the scene and determine the locations of the objects within it by triangulation. The computational requirements are considerable though, and several cameras are required per room to work around visual obstructions.

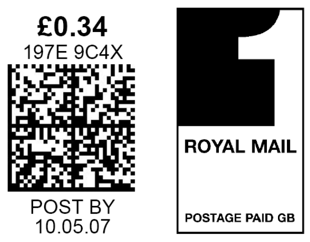
Visual means have been used in a number of other positioning systems, often for robot navigation. They employ the use of special markers placed on landmarks or entities in order to simplify the computation required in identifying them within a captured image. These special markers, or fiducials, have often come in the form of either circles (Cho & Neumann 1998, de Ipia et al. 2002) or squares. The properties of these fiducials are chosen so they yield position and orientation information whilst allowing for their simple, efficient and fast image processing.

The TRIP (Target Recognition using Image Processing) system (de Ipia et al. 2002) uses concentric circular rings as shown in figure 2.1(a). These fiducials can be captured by cheap cameras with reasonable resolutions of 640x480 and easily located from the captured image by detecting the inner ring. The inner ring also provides information for the attitude, or pose, of the fiducial with respect to the camera (Forsyth et al. 1991). The outer rings encode a ternary number ranging from 1–19,683. The precision of localising the fiducial's location is 1–6cm depending on the cameras distance and relative angle.

The BBC created a positioning system, called Free-D, for hand held studio cameras by placing circular fiducials over the studio ceiling (Jin et al. 1997). These fiducials, shown in figure 2.1(b), involved concentric rings of varying lengths, together forming a circular barcode. The fiducials used were retro-reflective and could be detected even in a noisy visual studio environment which included studio lights. The precision of this system was between 0.01°–0.06° degrees and within 1mm in a 50m^2 room.


(a) Circular barcode used in the TRIP system


(b) BBC's Free-D circular bardcode



(c) DataMatrix (ISO/IEC 16022:2006) barcode used by Royal Mail for postage stamps

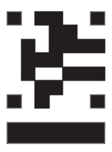

(d) CyberCode: A 2D barcode suitable for small CMOS/CCD camera based systems

Figure 2.1: Example 2D barcodes

The typical application of 2D barcodes is in product tagging or postage stamps, such as those used by Royal Mail (see figure 2.1(c)). Mobile phones equipped with cameras are becoming ubiquitous and it is attractive to use these camera phones to identify barcodes. Several commercial software systems have been developed specifically for use with mobile phones. Semacode created software to decode images of 2D barcodes based on the DataMatrix standard (see figure 2.1(c)). Semacode's barcodes encode a URL which may then be accessed on the mobile phone. QR (Quick Response code, ISO/IEC 18004:2000) is another square matrix based 2D barcode being used with increasing frequency to allow mobile phones to gather and use information from published content, such as adverts, coupons, business cards and logos. Shotcode is a 2D circular barcode and used in similar applications as the

QR code. Each of these mobile phone applications must all read the barcodes from a close distance due to their high density to size ratio. The CyberCode system, designed by Rekimoto & Ayatsuka (2000), involves highly distinctive barcodes to allow webcams and mobile phones to detect and read them from a greater distance away.

CyberCode's 2D barcodes act as markers to allow captured video images from a camera equipped mobile device to be augmented with annotated information related to the marker. Seen in figure 2.1(d), these fixed size 2D bar codes contain 24 or 48 bits of data and are printed on paper. They are used to tag physical objects in much the same way that RFID tags may be used. This tagging provides proximity information, but because the CyberCode barcode is of a fixed size, they also allow the position and pose of the barcode to be determined relative to the camera by analysing the size and distortion of the captured image.

Colour circular fiducials and 2D square barcodes have been used by Coughlan et al. (2006) as a navigation aid for the visually impaired using mobile camera phones. The circular fiducials are recognised by the mobile from a distance and provide navigation hints towards some target. Once the target is approached the mobile reads a 2D barcode placed beside the circular fiducial.

Easily recognised markers are also used in positioning for digital pens equipped with a small CMOS camera (Jared et al. 2001, Wang et al. 2005, Pettersson 2001, Pettersson 2006). Pages of paper are covered with two dimensional codes that have a 'floating' property. Unlike individual barcodes, these two dimensional codes do not require the entire code in order to be decoded. An entire surface may be covered with a single code, but any small part of the surface (a floating window) may be decoded to discover that part's location within the whole surface. These codes are covered in detail in chapter 6 and are the basis for providing the ceiling's positioning information in the scenario used in this thesis.

The ceiling has been used before in camera based positioning by robots. Dellaert et al. (1999) show how robots may build up image maps of the ceiling's light levels. Successive samples of the current ceiling's light intensity directly above the robot are used to estimate the robot's global location within the map by using a Bayesian filtering method. Localisation is impeded by any changes to the environment's light intensity and it is unclear how the approach scales to large areas.

Floors, as are ceilings, are always available wherever a person might walk indoors and can be inconspicuous and unobtrusive. Location systems employing physical contact proximity techniques on floors include Orr & Abowd's (2000) Smart Floor and the Olivetti Research Lab's Active Floor (Addlesee 1997). Both these systems use pressure pads and attempt to identify the walker through their footstep signature. The system must be trained to recognise individuals' footstep signatures and good recognition rates of 91 to 93 percent are achieved; however, this recognition may not scale to large populations.

2.2 Sensor Networks

Wireless sensor networks consist of spatially distributed independent devices equipped with sensors monitoring the state of the environment in which they lie (Akyildiz et al. 2002, Pottie & Kaiser 2000, Romer & Mattern 2004). These sensor nodes usually cover a large area with large numbers, so they are generally small cheap computers with limited power, computation and memory capacity. The sensors monitor environmental state such as light, temperature, or pollutants and normally relay these readings via wireless links back to a more powerful base station for processing.

In its simplest form, the network topology is just a star with a single hop to the base-station. Larger installations involve multi-hop networks where nodes may be connected only to several other nodes nearby in the space to conserve energy whilst transmitting.

The applications for sensor networks vary vastly in their requirements and are wide ranging from glacial environmental monitoring (Martinez et al. 2006) through to sensors embedded in furniture parts that aid and instruct in its assembly (Antifakos et al. 2002).

The deployment may be accomplished through a simple random scattering of nodes, or a deliberate and carefully chosen placement. Typically deployment is a one time event, but it may also occur over several points in the networks lifetime. For example, more nodes may be added to to increase coverage and sensor readings in an area of particular interest. They can also vary from purely static node deployments to fully mobile networks. In mobile networks, the movement of nodes can occur naturally, such as due to the flow of water, or even intentionally as with mobile robots. The expected lifetime of the network can vary from just hours to years or more.

Nodes might not perform any processing of the sensor data themselves, and may instead simply route it back to other nodes. In other networks, nodes may process the data themselves. The nodes may even collaborate with other nearby nodes to gain a higher order understanding of spatial phenomena, such as the tracking, mapping and prediction of the spatial boundary line of a flow front of contaminants as they move over time.

2.3 Localisation

Supporting sensor networks during their lifetime and easing deployment through self-configuration are algorithms dealing with the routing of communications and the procurement of location information. Due to their inherent spatial nature, sensor data is normally interpreted along with a reference to the location of the sensor node source itself. Additionally, some routing algorithms such as geographic forwarding (Stojmenović 2002) or trajectory based forwarding (Niculescu & Nath 2003*b*) require location information in order to operate. This makes localisation a critical and fundamental prerequisite to many applications. Here we will focus on the localisation problem and introduce the algorithms and technologies behind obtaining physical coordinates for each node in sensor networks.

To function, all localisation schemes require some means of measuring distances, or ranges, between nodes. Based on the ranging measurements, an algorithm will then derive a single coordinate system in which all nodes know their coordinates. There are a variety of different approaches to these two problems, each with their own merits, issues and applicability to different deployment scenarios and applications.

2.3.1 Ranging

2.3.1.1 RSSI

In many networks, nodes use radio to communicate, which gives the potential to analyse the signal strength of incoming transmissions to determine the distance to the transmitter, based on the knowledge that radio waves dissipate in intensity relative to the square of the distance travelled from the source. This is called the received signal strength indication (RSSI). Unfortunately, significant measurement error is often encountered as it is very sensitive to occlusion and multi-path effects from obstacles that interfere with the radio waves.

2.3.1.2 TDOA

Timing the flight of some phenomenon travelling at constant speed between two points gives a distance measurement. Systems employing this timed distance of arrival (TDOA) commonly use ultrasound, or momentarily audible pulses. An RF pulse can be sent from a source to mark the beginning of the timing for the receiver. At the same time, or after a known delay, an ultrasound pulse is emitted. As sound propagates far slower than RF but still at a known speed, the distance from the source to receiver can be calculated from the time of arrival of the sound relative to the time of arrival of the RF pulse. This ranging method gives a good near centimetre accuracy (Priyantha et al. 2000, Randell & Muller 2001, Harter & Hopper 1994). However, it can only operate when there is a clear line of sight between the transmitter and receiver and it requires careful calibration of the microphone and speaker hardware on every device.

2.3.1.3 AoA

Some hardware is capable of measuring the angle of arrival of light, sound or radio. This angle of arrival (AoA) is angulation rather than lateration but can still be used in localisation algorithms and in the trigonometric calculation of distance given some known locations. Accuracy to within 3 to 5 degrees can be achieved (Priyantha et al. 2001) but the normal hardware used can be large and contribute greatly to the node's size.

2.3.1.4 Hop Counts

Not a ranging method as such, but in the absence of any specialised ranging hardware, the path of communication taken in a multi-hop network can be used as an approximation to the straight line path between two points. Given the hop count of a packet from source to destination and the expected average distance between forwarding nodes, an approximated distance is gained. This is the ranging method assumed in the algorithms in this thesis.

This method can suffer extreme error in the vicinity of obstacles, where the shortest path across intermediary nodes no longer resembles the straight line between the source and destination. This thesis proposes a new algorithm, in section 3.2, capable of detecting the network boundary around obstacles which allows this extreme error to be avoided.

The density of nodes within communications range can also affect this distance estimate. This, and some other factors relating to hop based ranging, are explored in detail in section 4.1.1.

2.3.2 Anchors

The coordinate system resulting from localisation will either be absolute or relative. Absolute systems will be aligned and referenced with respect to some reference coordinate system, such as longitude and latitude in the case of GPS. Relative coordinate systems are meaningful only to the relative positions of the nodes within the network itself, unless some known transformation between the network's relative coordinate system and some external coordinate system is known so a subsequent alignment to it can be made. Such a transformation will involve scaling, rotation, translation and possibly a reflection to an arbitrary degree, as the network's coordinate system was formed entirely independently from the external coordinate system.

Localisation approaches can be placed into two categories, either anchor based, or anchor free. In anchor based approaches, some nodes, called anchors, have prior knowledge of their physical absolute coordinates within the coordinate system. These special reference nodes can greatly help other nodes with estimates of their own coordinates. In anchor free approaches, no nodes have predetermined coordinates before the localisation begins and the resulting coordinate system is relative.

In anchor based approaches the anchor's prior knowledge of their coordinates is gained either through some specialised hardware, such as GPS, or through a manual configuration of their coordinates. The placement and the number of these anchors can also have significant effect on the resulting localised coordinate system. In some applications the anchors are randomly spread out in the network whilst in others, they are placed manually in strategic locations. Any specialised hardware causes the anchor nodes to become relatively expensive, and in some situations such specialised hardware may not even operate. For example, GPS does not work indoors or underwater, and is both costly and sizeable. Manual configuration also has the drawbacks that it becomes infeasible for larger numbers of anchors, and may not even be possible in applications where the deployment of the sensor nodes is uncontrollable or inaccessible.

2.3.3 Algorithms

Bulusu et al. (2000) proposed a distributed algorithm that requires no ranging hardware, but does require that localising nodes be within a single hop of anchors. There must be sufficient numbers of anchors placed in suitable places to cover every localising node in the network. A node simply calculates its coordinates to be the centroid of its neighbouring anchor's known coordinates.

Doherty et al. (2001) express the localisation problem in terms of geometric proximity constraints. The measured ranges from anchor nodes form a convex geometric area in which it is reasoned that the node must lie inside. For example, a hop count of 5 from an anchor creates a constraint on the unknown node's location to be within a circle of radius 5 from that anchor's location. Angle of arrival data can be used to constrain location within triangular regions. Every localising node gathers a list of geometric constraints on their location and passes the list back to a single centralised base station. The localisation problem is then solved by calculating the unknown node's coordinates to be the centroid of the intersecting region between all of the geometric constraints. Once solved, the coordinates are then communicated back from the base station to their respective localising nodes.

The APIT method of He et al. (2005) is similar to Doherty et al., but executes in a distributed fashion. APIT uses the established locations of anchors to create triangular regions between neighbouring anchor nodes. A localising node then tests whether it lies within these triangles through a simple trigonometric test based on its RSSI measurements. A centre of gravity calculation is then performed on the intersecting area of the overlapping triangles in which a node resides.

Niculescu & Nath created a family of distributed algorithms named APS – the Ad hoc Positioning System (Niculescu & Nath 2001, Niculescu & Nath 2003*a*). One of three separate methods of propagating ranging measurements may be used in this system, called DV-Hop, DV-Distance and Euclidean. DV-Hop distributes hop count distances to anchors by exchanging distance vector tables with neighbouring nodes. DV-Distance is also exchanged by distance vector exchange but replaces the estimated distance of each hop by the actual distance of each hop according to TDOA or RSSI measurements. After a node obtains 3 or more distance measurements from anchors, it calculates its coordinates by multilateration (a process described in detail in chapter 4). The Euclidean method propagates each node's own self measured distance to the anchor. By also using the measured distances to at least two neighbouring nodes in addition to neighbour's distances to the anchor, a node may calculate its location relative to the anchor's coordinates using trigonometry on all the geometric information presented.

Nagpal et al. (2003) used a gradient (see section 2.5.1) hop count based approach very similar to the distance vector exchange of DV-Hop. It uses both an offline inter-hop distance estimate (Takagi & Kleinrock 1984) based on the node density and a diffusion style smoothing of hop counts to further augment the coarse distance measurement of hop counts. An estimate accuracy within 20% of the radio range is achieved in a simulation environment.

Niculescu & Nath's (2001) algorithm allows estimates of nodes that are not within immediate range of anchors by using a vector exchange over multiple hops, reducing the requirements on anchor density. Another approach allowing partial anchor coverage comes from Savvides et al. (2001) who describe both an iterative multilateration and a collaborative multilateration. When a node has TDOA or RSSI measurements to three or more anchors it may estimate its coordinates through multilateration and subsequently becomes a new anchor itself. This new anchor may then provide the opportunity for other unknown nodes to estimate their location after obtaining their distance from the new anchor and at least two other existing anchors. This process is named iterative multilateration as new anchors become available. Collaborative multilateration takes this a step further and allows two or more neighbouring nodes to share their distances to nearby anchors in order to estimate their own location. For example, this allows neighbouring unknown nodes that both separately have insufficient measurements, but enough measurements when combined together to simultaneously solve their positions.

Savarese et al. (2002) separates the localisation into an initial stage and a refinement stage. In the initial stage the first estimate is calculated as normal with multilateration. These estimates are further refined in the next stage by taking range measurements from neighbouring nodes localised in the initial phase. Savvides et al. (2002) later provided a refinement stage to their algorithm using Kalman filters.

Multidimensional scaling (MDS) is a data analysis technique used to visualise highly dimensional data in a two-dimensional map and has received much recent attention when applied to localisation. Shang et al. used MDS to create MDS-MAP which requires no anchors to localise (Shang et al. 2003, Shang, Ruml, Zhang & Fromherz 2004). Each node determines its neighbours and sends this list to a central base station where the shortest path between all nodes in the network is then calculated. MDS is applied to the shortest path information which produces a two dimensional map of the nodes. After an optional translation to align the map with an absolute coordinate system, the nodes' coordinates in the map are sent back to the individual nodes themselves. The centralised processing requirement of MDS-MAP increases sharply as the network increases in size due to the complexity of the determining the shortest paths and of performing MDS.

Ji & Zha (2004) used MDS in a distributed fashion to create local maps of nearby nodes. These local maps are then transformed to align with each other and merged together into one coordinate system. This method of aligning local maps together has some drawbacks. Without sufficient overlap and nodes in common between two neighbouring local maps, an alignment may not be possible, or may even become misaligned resulting in two flipped coordinate systems being merged together.

2.3.4 Centralised and Distributed

Whilst a greater accuracy can be achieved with a global network knowledge, and individual node computational requirements are lowered with the centralised approaches, there are some issues. Using a base station requires a computer powerful enough to tackle the global network localisation problem which makes it more costly and probably markedly larger than a normal node. The communication of measurement data to and from the base station on multi-hop networks will cause communications congestion and greater power drain on nodes surrounding it. Due to its centralised nature, there will be an upper bound on the scaling of the network. This may be alleviated somewhat by using employing some hierarchical organisation of multiple base stations but further costs are introduced and inter-base station communication is still required. Furthermore, it is not always possible to even deploy a base station in some scenarios. Distributed algorithms, on the other hand, compromise some accuracy in the solution of the coordinate estimates through the loss of global network knowledge. Nevertheless, they avoid the scaling issues of centralised algorithms so are much more amenable to larger networks.

2.3.5 Line of Sight and Concave Networks

Many approaches require an uninterrupted line-of-sight between the localising nodes and the anchor nodes. In the algorithms mentioned above, there is generally a great sensitivity to network layouts with concave perimeters and large obstructions.

With localisation schemes involving pre-established anchors, a correction can be made for small anisotropic error effects by accounting for the discrepancies between the measured distance and the actual known distance between neighbouring anchors. In schemes where pre-established anchors are not present, this correction can not be made. The performance of MDS and other anchor-less based approaches is severely degraded in concave network layouts.

2.3.6 Comparison of Localisation Schemes

Table 2.1 shows a comparison between the different localisation schemes introduced in this section. The proposed localisation algorithm described in this thesis is a distributed algorithm producing a relative coordinate system through multilateration of hop based ranges.

Table 2.1: Comparison of different approaches to localisation in sensor networks

Approach	**Ranging**	**Anchor Coverage**	**Coordinate System**	**Algorithm**	**Solution by**
Bulusu et al.	None	Full ($\geq$ 2 seen by each node)	Absolute	Distributed	Centroid of anchors
Doherty et al.	Any	Full ($\geq$ 2 seen by each node)	Absolute	Centralised	Centroid of intersection area
He et al. (APIT)	Proximity & RSSI	Partial	Absolute	Distributed	Centroid of intersection area
Niculescu et al. (APS DV-Hop)	Hop count	Partial	Absolute	Distributed	Multilateration
Niculescu et al. (APS DV-Distance)	TDOA/RSSI	Partial	Absolute	Distributed	Multilateration
Niculescu et al. (APS Euclidean)	TDOA/RSSI	Partial	Absolute	Distributed	Multilateration
Nagpal et al.	Hop count	Partial	Absolute	Distributed	Multilateration
Savvides et al.	TDOA/RSSI	Partial	Absolute	Distributed	(Collaborative) Multilateration
Savarese et al.	TDOA/RSSI	Partial	Absolute	Distributed	Multilateration
Shang et al. (MDS-MAP)	RSSI/1 Hop	None	Relative	Centralised	MDS
Ji et al.	RSSI/1 Hop	None	Relative	Distributed	MDS

There are similarities of the proposed algorithm with APS DV-Hop and Nagpal et al.'s systems in that it uses the multilateration of hop-based measurements to established anchors. In the proposed scheme, however, the anchors are not initially present and instead new anchors are elected as the coordinates spread throughout the network. The spreading of the coordinate system and new anchors in the proposed scheme bears some resemblance to the introduction of new anchors in the iterative multilateration of Savvides et al. (2001).

As with the anchor-less based schemes of Shang et al. and Ji et al., a relative coordinate system is the result of having no initially configured anchors. However, unlike these other anchor-less systems, the proposed scheme can operate well in concave networks and does not require the intensive computation of MDS.

2.4 Small Devices in Large Numbers

The size and cost of wireless communications and computation has been steadily decreasing with advances in microelectromechanical systems (MEMS). MEMS integrate mechanical sensors and actuators with silicon based integrated circuits. The Smart Dust project (Warneke et al. 2001) was an effort to combine sensors, computing and communications systems into a cubic millimetre device. Whilst the size being strived toward for these devices, or motes, is in the cubic millimetre scale of a grain of sand or a particle of dust, the Smart Dust project attempts to discover the limitations of microfabrication technology itself.

MICA motes (Hill & Culler 2002) are currently commercially available at a size of 2.5×0.64cm with 128kb of memory, a CPU running at 4Mhz, and an onboard transceiver. On the commercial horizon, HP Labs have announced a $2 - 4\text{mm}^2$ device, called the Memory Spot, which can store from 256Kbit to 4Mbit of data and is capable of radio communication (HP 2006).

Recent research has the size of fully operational motes reduced to 5mm^2 (Warneke 2003) and future research in nanotechnologies could conceivably see these current size limitations of combining computation, communication and power, reduced down to far below this current scale. For example, computation and memory have already been demonstrated with carbon nanotubes (Fuhrer et al. 2000, Rueckes et al. 2000) and sensing has been shown with nano-wire sensors (Kong et al. 2000). In other areas such as natural computing, molecular and DNA based computation is possible in an unprecedented small size and large number (Paun et al. 2006, Livstone et al. 2006).

Cheap mass production at such a scale brings the capacity to create massively distributed sensor networks with numbers of nodes scaling to many thousands and conceivably to millions and beyond. It also brings several new problems. The sheer number of devices involved, makes any centralised interactions impractical or nearly impossible. The devices are small; as such, they may be computationally weak and have limited battery life and memory. As the devices are manufactured on such a large scale, it is infeasible to test each and every device, so some may be inoperable or unreliable.

2.5 Amorphous computing and sensor networks

Abelson et al. (2000) introduced Amorphous computing as an effort to create high-level programming language abstractions and constructs capable of engineering useful controlled behaviour in ultra-scale sensor networks, without the need to address directly their underlying complexities such as node failure and networking.

These ultra-scale networks are large spatially distributed systems comprised of innumerable, homogenous, and potentially unreliable nodes. The amorphous computer and network concepts used in this thesis are based on the following assumptions, which share a number of similarities to those in the lists given in (page 2, Beal 2005) and (page 3, Beal & Bachrach 2006):

- There are large numbers of nodes, the scale of which is unknown.
- Nodes are stationary and spatially dispersed over a two dimensional surface according to a random Poisson distribution. This irregular distribution is sufficiently dense enough that nodes remain connected within a radius, r, of each other. The density remains roughly uniform throughout the network.
- Each node can communicate only with nearby neighbouring nodes within a radius, r, in their local spatial neighbourhood by means of an unreliable broadcast communication. There is no routing infrastructure.
- Nodes are identical and are all installed with the same program. There is no individual identity; however, it is assumed that each node is equipped with a random number generator.
- Each node is autonomous; there is no global clock synchronising each node but each node's clock is assumed to be running at approximately the same speed.
- Nodes do not initially know their position or orientation. There can be no global knowledge of the network topology due to its arbitrary scale and initially there is not even knowledge of the local interconnection topology. Any knowledge of the topology gained will be through interactions from nodes in the local neighbourhood.
- All of these nodes are very cheap but have limited power, memory and processing available.

Due to the sheer scale of the network and the localised nature of interactions between nodes, any algorithms operating over nodes on the network must be fully decentralised. A number of powerful decentralised techniques for programming these networks and bringing about a global structure and coordination from just local interactions have been investigated in the amorphous computing effort (Beal &

Bachrach 2006, Beal 2005, Beal & Sussman 2005, Nagpal 2003, Butera 2002). The algorithms proposed in chapters 3 and 4 draw heavily on the primitives described by Butera's (2002) Paintable Computing and some of the concepts show a similarity to those used in the recent Amorphous Medium Language (Beal & Bachrach 2006, Beal 2005). The primary primitives and methods are described in the following subsections.

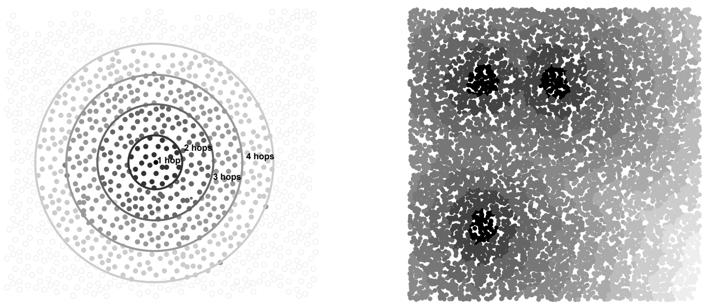
2.5.1 Gradients

The principal primitive used in much of the amorphous research is the gradient. A metaphor of biological pheromones, or chemical gradients explains this primitive well. A source emits a signal, such as a pheromone, which dissipates into the local area with an intensity that decreases with distance from the source. Anything that can sense the gradient can act usefully depending on its intensity. For example, escherichia coli bacteria exhibit chemotaxis when they move toward higher concentrations of attractants, such as nutrients, or away from repellents, such as poisons (Bren & Eisenbach 2000). Foraging ants show chemotactic interactions when they tend toward greater intensities of trail pheromones left by other ants, which through the method of stigmergy will collectively form the shortest path to a food source (Theraulaz et al. 2003). Mosquitos are attracted towards and along downwind trails of carbon dioxide exhaled from potential hosts, whilst blackflies and tsetse flies are primarily attracted to the contrast between colours or light intensity.

In networking terms, a simple gradient is emulated by a node broadcasting a gradient message with a set intensity to all of its connected neighbouring nodes. Each node receiving the gradient message will store it in the node's local state, decrement the gradient message's intensity and forward it on to its neighbouring nodes, via another broadcast. This process repeats until the gradient message has zero intensity. Any node that receives a gradient message that is already stored in its local state will only store, decrement and forward it if its intensity is greater than the currently stored gradient message. In a sufficiently dense, reasonably uniform arrangement of nodes, a gradient results in concentric circles of nodes around the source node, with decreasing intensities as the hop count increases (see 2.2(a)). A gradient with multiple sources is also possible, resulting in contours of intensities over the network space (see 2.2(b)).

Multiple gradients can be handled easily by assigning a unique identifier to distinguish them from each other. Each node can be ascribed a unique identifier by generating a large random number.[1] This node ID can put in the gradient message to identify the gradient source and a separate gradient ID can be put in the message to distinguish between multiple gradients from the same source node.

[1]Technically this random number cannot be guaranteed to be unique, but with a sufficiently large number and a properly performing random number generator, it is highly probable



(a) An example gradient emitted from a single source node in the centre

(b) 3 sources for one gradient

Figure 2.2: Example gradients

2.5.2 Shared State in Local Neighbourhood

It is useful to allow nodes in the neighbourhood to share some state between each other. The idea of nodes containing HomePages was developed by Butera (2002), similar to the concept of blackboard systems used in artificial intelligence for communication between agents. Each node stores a variable sized read/write mapping of keys and their values, called a HomePage. Each node exports this table by broadcasting the HomePage to its immediate neighbours. Neighbours store read only versions of all received HomePages exported from neighbours. Whenever a change to a node's HomePage is made, the node retransmits the HomePage to its neighbours. To handle the unreliable nature of the broadcast communication, HomePages can be periodically exported, after a short random time has elapsed, even if no change is made. Some housekeeping is needed to remove neighbouring pages after a time, in cases where a neighbouring node unexpectedly dies. This method of shared state effectively sets up mirroring pairs throughout the neighbourhood and abstracts the communication between nodes and their processes.

Sharing state in this manner makes it easy to write programs to disseminate information over spaces beyond the local neighbourhood. Gradients can be implemented easily using HomePages by searching through all neighbouring pages for gradients with the highest intensity and writing these gradients, with one lower intensity, to the HomePage (unless it is already present). This search for gradients can occur whenever neighbouring HomePage updates are received. If an existing gradient on a node other than the source node is no longer seen in any neighbouring HomePages, the gradient should be declared dead and removed from the HomePage. Entire gradients can be removed by removing a gradient entry in the source node.

If there are periodic updates of the HomePage broadcast to neighbours, gradients automatically adapt to any changes in the topology such as a single node failure or even the loss of an area of nodes. Gradients continually maintained in this way are called Active Gradients (Clement & Nagpal 2003). A variant of this is the Active Morphogen (Nagpal 2003) where the gradient intensity stored on nodes decays over time unless the source is continually emitting, similar to the evaporation of pheromones.

2.5.3 Regions

It is useful to have the notion of spatial regions of which nodes are members. Regions delineate areas of nodes from each other and allow for groups of nodes to specialise, much in the way that many cells in an organism form distinct specialised organs. This eases the coordination of a larger number of nodes than those in the immediate local neighbourhood.

A connected region is formed from the evaluation of a boolean membership predicate. Such a predicate could be attached to a gradient propagated from some source node. Nodes that satisfy the predicate become members of the region. As an example, figure 2.3 shows the region formed by evaluating a simple predicate ensuring that all the members must be within a certain range from the sources of a gradient.

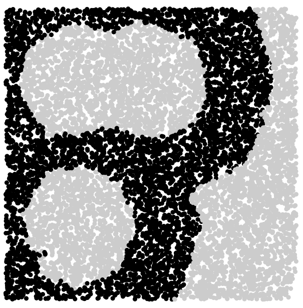


Figure 2.3: Region formed between 12 and 24 hops from the multiple gradient sources of figure 2.2(b)

One or more gradients may also form regions. As an example of using multiple gradients to form regions, tube formation (Abelson et al. 2000) uses two gradients propagating over the same space from two different nodes to create a thin tube-like region of nodes between the two sources. In tube formation, one source node propagates a gradient, g_1. This gradient is received by a second source node which then propagates a second gradient, g_2, with the same initial intensity. The second node also stores the hop count, $dist = hops(g_1)$, of g_1 into g_2 to be carried as information. As g_2 spreads out, intermediary nodes perform a small calculation to see if they reside on the tube region. Any node that has $|dist - intensity(g_1) + intensity(g_2)|$ less than or equal to a desired tube thickness is considered to form part of the tube. A useful biological analog here is the distinction of different areas according to the interaction between two or more chemical gradients.

The dissemination of information throughout an existing region is achievable by a gossip-based communication (Beal & Gilbert 2004). This gossip floods the data throughout the region by having member nodes assume neighbouring members' shared gossip data into their local state. In a similar fashion to gossip, the propagation of gradients, and their attached information, can be constrained to within a specific region or combination of regions.

Through node failure or node mobility, regions may become fragmented into two or more separated and disconnected areas. The separate areas are cloned fragments of the same region. Unless reconnected, each fragment will run region specific code independently of other fragments, possibly leading to their state and behaviour diverging. The reconnection of such fragmented regions has been explored in the development of persistent nodes (Beal 2003). Briefly, a persistent node is a region encompassed by another region, called the umbra. If the umbra regions of two fragmented persistent nodes ever come into contact with each other, one of the two fragments is removed and overwritten with the other.

2.5.4 Forming Spatial Structures over Nodes

The formation of topological patterns on the nodes over the network space was demonstrated by Coore (1999) by using gradients. Using the biological metaphors of pheromones and chemotaxis, the pattern topology is grown by attracting and repelling it from certain concentrations of pheromones that are excreted by nodes forming part of the pattern. This is in much the same way as a botanical tropism, whereby plant growth tends towards or away from environmental effects such as light or gravity. The target global topological pattern is specified in the Growing Point Language (GPL) and compiled down to a single program that executes on each individual node. The language is powerful enough to describe and construct any planar graph. A tropism can be easily programmed on nodes with the use of gradients and HomePages by finding the neighbours with the lowest gradient hop count and choosing those neighbours as the next nodes to grow on.

In (Nagpal 2001) geometric shapes are formed on the nodes over the network space by applying successive folds according to axioms used in origami, the art of paper-folding. The global shapes are described in Origami Shape Language that, like GPL, compiles into the single program that executes on each node. The approach relies on starting with some of the nodes differing in their state, specifically that nodes know whether they are on the edge of the used network space or not and have a polarity. The edges of the folds are formed mainly from two methods, by tropisms and by looking to the hop counts of two gradients relative to the distance between them. The second method is a variation on the tube formation described above in section 2.5.3, where instead a roughly perpendicular tube is formed that bisects the straight line between the two gradient sources.

Assembly of arbitrary geometric shapes over the network is achieved in (Kondacs 2003) by packing a sufficient number of covering discs of varying sizes over the space. Reference points on the discs are obtained through triangulating the hop counts and distance from the neighbouring discs. New discs are grown at elected reference points to continue the formation of the shape.

Note that the approaches in the last three paragraphs each aim to create a structured shape to occupy a space within the network, whereas in this thesis one of our aims is to use geometrical structures to discover the geometry and shape of the network space itself.

The recent Amorphous Medium Language (Beal 2005, Beal & Bachrach 2006) allows a more general spatial programming (as does Paintable Computing (Butera 2002)) and description of the intended global behaviour. It abstracts the idea of the amorphous network to a continuous medium rather than as a space approximated by many discretised points or nodes.

The infrastructure and some applications of Paintable Computing, where computing particles are suspended in paint and covered over a surface, was explored in (Butera 2002). This idea of surfaces covered in a computational medium by painting it is also the method of deployment assumed in this thesis for creating the indoor location system in section 6.

Chapter 3

Perimeter Detection

This chapter proposes and details a distributed algorithm for detecting perimeters of a network, by finding circular paths around nodes. Particular focus is given to networks without a strictly convex perimeter, or those containing large areas with no connectivity. Networks with large disconnected areas effectively have more than one perimeter, where all the inner perimeters are concave in nature – these networks are referred to in this text as concave (regardless of whether their outer perimeter is convex).

One of the largest sources of localisation error, in networks with concave perimeters, is caused by the network perimeter perverting the hop-based ranging measurement. The nature and consequences of this error is examined in depth, together with other sources of multilateration error, in the following chapter under section 4.1.1.

This new proposed algorithm is evaluated by simulations in chapter 5.

3.1 Edge Detection within Sensor Networks

In order to avoid the distorted distance measurements arising from edges, we must first be able to determine the perimeter of the network itself. Using the perimeter knowledge, anchor gradients may be augmented to include a count of the edge nodes that it has travelled through. This edge node count is effectively a metric for the potential error introduced from the edges of the network, and it can be used to discriminate which anchors to use in multilateration.

Within this section we shall briefly look to the current approach for detecting the perimeter of the network. In the following section, we propose a new approach that is more applicable in the context of localisation.

3.1.1 Current Distributed Approaches to Edge Detection

A distinction should be made between discovering the boundary of phenomena within the network and discovering the physical perimeter of the deployed network itself. Boundary estimation delineates between different regions of the network, based on sensor readings of phenomena such as temperature gradients (Zhang et al. 2004) or contamination levels (Ailamaki et al. 2003); whereas perimeter detection discovers nodes deemed to be at or near the edge of the network's domain, beyond which no connectivity exists and no communicable nodes reside.

Three decentralised approaches for boundary estimation are explored by Chintalapudi & Govindan (11 May 2003), all of which collect decisions on region memberships from neighbouring nodes. The first approach is statistical and a boundary node is detected if the number of its neighbours inside or outside of the region lie within some acceptable threshold. The second approach is inspired by high pass filtering from the image processing literature used for detecting edges. A weighted average of the neighbours' region membership is used to mitigate the effects of an arbitrary placement of sensors. The last algorithm is classifier-based and motivated by the pattern recognition literature.

In (Nowak & Mitra 2003), the network domain is partitioned into a square grid. The cells within this grid containing nodes from different regions (i.e. cells which are ambiguous with respect to which region they belong to) are subsequently refined into four smaller partitions. The smaller partitions are progressively refined in this manner along the boundary until no cell's region membership remains ambiguous. Once this refinement is complete, an approximation of the boundary is obtained from the sides of cells which border cells from another region.

All these boundary estimation algorithms expect a distribution of sensors overlapping the regions and so can not be applied to detecting the network perimeter as connectivity is lost beyond the boundary. Even in (Wood et al. 2003) where a jammed, and therefore disconnected, region of the network is mapped, a prior knowledge is required of the nodes involved in the jammed area. Nevertheless, the metrics used in evaluating boundary estimation algorithms (Chintalapudi & Govindan 11 May 2003) are useful as a basis for evaluating the perimeter detection algorithms, which will be introduced and used in chapter 5.

Boundary estimation of phenomena has within the last few years been given a great deal more attention than perimeter detection of the network. In fact, at the time of writing, there appears to be just one paper describing an algorithm for discovering the network perimeter in a sensor network (Martincic & Schwiebert 2004, Martincic & Schwiebert 2006). This distributed algorithm runs on each node and collects its local 1-hop and 2-hop neighbour's location and connectivity information. This collected information allows a node to perform a brute force search for any possible enclosing cycle around the node. If such an enclosing cycle can not be found, the node is considered to be an edge node.

The algorithm executes on each node in the network. Each node performs the following stages:

- Collect the locations of neighbours within 1 and 2 hops.
- Identify all of the node's 1-hop and 2-hop neighbours' neighbours. After this stage, there is full knowledge of the connectivity within 2 hops of the node.
- Using a list of the neighbours sorted according to their relative angle to the node, a brute force search is made for a valid enclosing cycle. This cycle must satisfy the following criteria:
    1. All nodes in the cycle are either 1-hop neighbours or 2-hop neighbours.
    2. When 2-hop neighbours are in the cycle, they are both preceded and followed in the cycle by 1-hop neighbours. There may not be two or more consecutive 2-hop neighbours in the cycle.
    3. Each node in the cycle is a neighbour of both the preceding and following node.
    4. The cycle is complete, the starting neighbour node is connected to the ending neighbour node.
    5. The node lies in the interior of the closed polygon formed from the locations of the nodes in the cycle.

This approach works well with a sufficiently dense distribution of sensors and produces a relatively thin perimeter of edge nodes. Such a thin perimeter is not a problem as the thickness can be increased by applying a form of 'peer pressure' - non-edge nodes may be pressured into becoming edge nodes if the number of their neighbours already marked as edge nodes falls within a certain threshold.

Unfortunately this algorithm is unusable for a couple of reasons. Firstly, when the distribution of nodes is not very dense, the algorithm shows a sensitivity to local areas with few neighbours and incorrectly considers nodes located in these regions as part of the network's perimeter[1]. Such a sensitivity will introduce considerable noise into our augmented edge-counting gradients and cause difficulty in selecting appropriate anchors for multilateration. The second reason is more grave and fundamental: the algorithm assumes all node's locations are known, but at the stage that we wish to ascertain the perimeter, the nodes have no known location. An algorithm that discovers edge nodes without any location knowledge is needed and a reduced sensitivity to local disconnected areas is desirable. Such an algorithm is proposed below in 3.2.

[1]As shown in chapter 5, the algorithm is very sensitive when the average node degree, or number of average nodes in a node's neighbourhood, is lower than 30

3.2 Perimeter Detection Without Location Knowledge

This section describes in detail the proposed edge detection algorithm. Unlike the boundary of phenomena estimation algorithms, this algorithm does not rely on any node sensor readings. The phenomenon in question, the network perimeter, is detected solely through connectivity information shared by nodes in the local neighbourhood. It fully distributed like Martincic & Schwiebert's (2006) algorithm, but does not require any existing location knowledge of the nodes at all.

In amorphous computing, the creation of spatial structures over the network using gradients has been demonstrated before (e.g. Coore 1999, Nagpal 2001). Behind this algorithm is the reasoning that nodes next to the network perimeter, attempting to form a circular region from a simple gradient, will instead result in forming a pie like region due to the presence of the network edge.

The algorithm operates by attempting to form a topological structure from a simple gradient, that of a ring on the outermost part of the gradient's extent. It is then possible to analyse this ring to check if it has the simple topological property that a cycle around the ring is present. In the case of a normal circular gradient, a full ring is formed, and a cycle around the ring can be found; however, where the gradient is next to a network edge, the ring around the pie shaped region will be broken and disconnected. Where broken rings exist, the algorithm attempts to reconnect the full ring topology by rebuilding a new path between the broken ends of the ring. It is the nodes on this rebuilt path that are flagged as edge nodes and form part of the detected network perimeter.

If enough rings sufficiently cover the network, the entire network perimeter will be detected by the breaks in all the rings.

The algorithm is split into a number of stages, covered in detail from section 3.2.4 through to 3.2.10. Stages 1 to 2 involve the formation of the rings over the network. Stages 3 to 5 serve to break the symmetry of the ring and to establish a sense of direction within it, so a search for a cycle may be performed. Stage 6 detects whether there are any breaks in the ring and locates nodes closest to them. Stage 7 prepares the way for a good path to be found between the broken ends of the ring. Stage 8 actually creates the path and reconnects the ring, flagging each passed node as an edge node.

It should be noted that the pseudocode listings throughout this chapter only serve to illustrate the algorithm and some detail and complexity has been omitted for conciseness. Whilst the listings are not exhaustive, they are comprehensive enough to produce an implementation. Please refer to appendix A for a full Java implementation of this algorithm.

Before looking in detail at the different stages, a description of the mechanisms used for handling gradients and timeouts is given in section 3.2.1.

3.2.1 General Computational Model of a Node

Here we describe the gradient forwarding mechanism used, and define the functions required to understand the pseudocode in the following sections. Also, in its current form, the code for the different stages of the algorithm is executed on nodes after a period of time elapses - the mechanisms for scheduling and handling these timeouts are described. Finally, the different gradients and timeouts used in the proposed edge detection algorithm are listed.

A single node only executes code response to two events: a previously scheduled timeout, and receiving a neighbours' message. All code is assumed to be executing in a serial fashion. When an event is handled, it is handled in its entirety before handling of any other event may occur.

In the pseudocode, a node can read and write local state with the *myState* object. This object contains one well known property, *nodeId*, which is the node's randomly chosen identifier. In some places[2], the pseudocode may reference a *hostState* object, which is an alias for the *myState* object residing on the node that the code is executing on.

3.2.1.1 Timeouts

A node can schedule a timeout using the SCHEDULETIMEOUT($TIMEOUT\_NAME$) function. For the sake of simplicity, in the pseudocode, the $TIMEOUT\_NAME$ argument is just a mnemonic name - the point in time it corresponds to would be calculated based on the convergence times of the different stages (refer to section 3.2.11 for these calculations).

When a timeout event occurs, the SCHEDULETIMEOUT($TIMEOUT\_NAME$) function will be called to execute code to handle the timeout.

3.2.1.2 Communications

The communications capability of a node is assumed to include broadcasting of small messages to nearby neighbours lying within in a roughly circular radius. It is assumed that the communications link between neighbours is bi-directional - a node should not be considered a neighbour if the link is only unidirectional.

This communication link is unreliable: message loss, collisions, and other faults are possible.

3.2.1.3 Homepages

A node maintains a list of key-value pairs in a read-write table called a HomePage. A node will periodically broadcast this HomePage to its neighbouring nodes, regardless of whether the HomePage has changed or not[3]. Each node also stores the

[2]Namely, in the *Constaint* predicate (see section 3.2.1.4)

[3]It should be noted that the communication's mechanism used in the simulations of the algorithms (as provided in appendix A) differs slightly from this description. For simplicity, there are

latest received, or imported, HomePage from each neighbour, as a read-only table. This mechanism allows the exchange of state between neighbouring nodes.

If no HomePage is received from a neighbour after a reasonable period of time, that imported HomePage is deleted and the node is no longer considered to be a neighbour. Likewise, if a new node is placed within communications range, or an old node has become available after a period of failure, it will be considered a new neighbour by nodes receiving its HomePage.

A neighbouring node, at certain points in time, may not have the resources available to receive or process incoming messages. Also, a collision, or some other communications fault, can cause a broadcast message containing the HomePage to be lost. It is a node's periodic broadcast of the HomePage to neighbours that can accommodate for the unreliable nature of the communications. If the sent HomePage was not received, and the fault is not chronic, it is likely that the message will be received by one of the subsequent periodic HomePage broadcasts. After broadcasting a HomePage, a node will schedule a timeout for the next broadcast. This timeout will occur after a random time, *period*, has elapsed, where $MinPeriod \leq period \leq MaxPeriod$. The range between $MinPeriod$ and $MaxPeriod$ is chosen to best avoid repeated collisions. In the face of common communication errors, a constant called $MaxMsgDelay$ is chosen to denote the longest reasonable delay to expect before a HomePage update is received by a neighbour. The $MaxMsgDelay$ constant is useful in deciding upon how long the different stages of the algorithm should take (e.g. see section 3.2.11). This constant may be different for different communications mediums and operating environments, but as an example, $MaxMsgDelay$ could be set to $(MaxPeriod*3)$ to allow for recovering from up to two consecutive communication failures.

Two functions related to HomePages are referred to in the pseudocode:

- putInMyHomePage(key, value) - Places a new mapping in the node's HomePage. This will be exported to all neighbours in the usual fashion when the node next broadcasts its HomePage.
- existsInNeighboursHomePage(id, key) - Checks for the presence of the key in the latest imported HomePage from the neighbour identified by id

3.2.1.4 Gradients

This subsection will describe how gradients are started, how they propagate, and will introduce the functions and events used in the pseudocode.

no communication faults in the simulations, and there are no node failures. This means there is no need for periodically re-sending HomePages if they have not been modified - something that allows a quicker execution of the simulations.

A gradient is started by placing details of it in the HomePage. A gradient normally has several standard details associated with it, which are briefly enumerated in the following list:

- *GRADIENT_TYPE* - A name for the gradient. This designates which gradient handler functions are used during propagation.
- *sourceID* - This is the ID of the node which started the gradient. Where gradients have multiple sources, this is not used.
- *gradientID* - An ID identifying the gradient. This distinguishes different gradients, of the same *GRADIENT_TYPE*, from each other in the HomePage.
- *Constraint* - This is a predicate that is evaluated whenever deciding if a gradient should be allowed on to a particular node. It serves to direct and restrict the propagation of the gradient[4]
- *Hops* - This is the number of hops that the gradient has travelled from the source. In the source node's HomePage, this will be 0.
- *isDecaying* - A flag indicating the gradient was stopped by the originating source node and is currently in the process of being removed from participating nodes' HomePages.

In the pseudocode, there are two functions for starting a gradient:

- MAKEGRADIENT(*GRADIENT_TYPE*) - This just creates and returns a new gradient object (with a hop count 0) of the given type. It does not place it in the HomePage. This new object would then be populated in the pseudocode with gradient specific details.
- STARTGRADIENT(*gradientObject*) - The passed gradient is placed in the node's HomePage, effectively allowing it to begin propagating when the HomePage is next broadcast to neighbours.

Gradients propagate throughout the network whenever a node exports its HomePage. All nodes, when receiving an exported HomePage from a neighbour, will look for any gradient details in the HomePage updates. When a gradient is found in this update, a gradient object is reconstructed from the details and this gradient object has its *Hops* value incremented by one. The gradient's *Constraint* predicate will then be evaluated, which possibly can draw on local node state. Only if the *Constraint* evaluates to *true*, is the gradient considered for propagating on to the receiving node.

[4]For brevity, the pseudocode treats this predicate as mobile code passed from node to node and evaluated at run-time. In the Java simulations however, every node contains the code for all gradient types' predicates at compile time. The Java simulations also do not store and propagate the predicates via the HomePage.

If a gradient from a neighbour's HomePage satisfies the *Constraint* predicate, the local node will check it's own HomePage to see if the same gradient has previously propagated on to this node. If the gradient is not already present in the node's local HomePage, then it is simply placed in the HomePage. If the gradient is already present in the node's local HomePage, the gradient forwarding mechanism will call the function

$$\textsc{isBetterThan}(\mathit{GRADIENT\_TYPE},\ \mathit{newGradientObject},\ \mathit{existingGradientObject})$$

to decide whether the existing gradient is overwritten with the new gradient. If the isBetterThan function returns false, no further action is taken, and the gradient is effectively ignored; however, if the isBetterThan function returns true, the existing gradient is overwritten with the new incoming gradient.

Once a gradient has satisfied both the *Constraint* predicate and the isBetterThan function, it is written in the node's HomePage, thus allowing it to be exported to neighbouring nodes as per the normal HomePage updates. Immediately after writing the gradient to the node's local HomePage, the

$$\textsc{gradientDetected}(\mathit{GRADIENT\_TYPE},\ \mathit{newGradientObject})$$

function is called to allow the node to handle the new information.

The gradient forwarding mechanism allows a gradient to be stopped and eventually removed from all participating nodes' state by flagging the gradient as decaying, using the *isDecaying* gradient attribute. These decaying gradients are treated differently to normal gradients and will be ignored, during forwarding, by nodes that have not before received a gradient of the same type and ID. They are not ignored, during forwarding, by nodes that are host to a non-flagged gradient of the same type and ID, and the incoming decaying gradient causes the node's current non-flagged gradient to then be flagged as decaying. This continues until all of the nodes that ever received the gradient have it flagged as decaying. After some suitable time (in order to avoid any undecayed part of the gradient from spreading again) each node will remove the decayed gradient from their state.

The steps taken by the gradient forwarding mechanism, upon receiving a neighbours' HomePage broadcast, is summarised in Listing 3.2.1.

Algorithm 3.2.1: GRADIENTFORWARDINGMECHANISM(*neighbourHomePage*)

```
for each $gradient \in neighbourHomePage$
  do
    $gradient.Hops \leftarrow gradient.Hops + 1$
    $existingGradient \leftarrow$ GETGRADIENT($gradient.GRADIENT\_TYPE$)

    if (NOTNULL($existingGradient$) and $existingGradient.isDecaying$)
      then comment: Gradient is ignored as it is decaying
      else if ($gradient.isDecaying$ and ISNULL($existingGradient$))
      then comment: Gradient is ignored as it is decaying
      else if ($gradient.isDecaying$ and NOTNULL($existingGradient$)
      then
        comment: Flag the gradient as decaying, if it is not already
        $existingGradient.isDecaying \leftarrow true$

      else if EVAL($gradient.Constraint$) = $true$
      then
        $acceptGrad \leftarrow true$
        if NOTNULL($existingGradient$)
          then
            $gType \leftarrow gradient.GRADIENT\_TYPE$
            $acceptGrad \leftarrow$ ISBETTERTHAN($gType, gradient, existingGradient$)
        if $acceptGrad = true$
          then
            comment: Store the gradient in the node's own HomePage
            GRADIENTDETECTED($gradient.GRADIENT\_TYPE, gradient$)
          else comment: Gradient is ignored

      else comment: Gradient is ignored
```

Two further functions, for obtaining gradients from a node's HomePage, may be seen in the pseudocode:

- GETGRADIENTS($GRADIENT\_TYPE$) - Obtains a list of all gradients, of the specified $GRADIENT\_TYPE$ type, from the node's own HomePage.
- GETGRADIENT($GRADIENT\_TYPE$), GRADIENTID - Obtains the single gradient object, with the specified $GRADIENT\_TYPE$ and ID, from the node's own HomePage. This will be null if the gradient is not present in the HomePage.

3.2.1.5 Failures

The periodic updates of the HomePage mechanism act as a buffer against communication failures, and allow for gradients to adapt to changes in the network's connectivity; nevertheless, the simulations of the algorithms in this thesis do not cover node failures, communication failures, node additions, or node mobility. The description of the algorithms will contain only passing reference to possible handling of communication and node failures. Further discussion of chronic failures and the ramifications of other topological changes for the algorithms is deferred to chapter 7. Comprehensive consideration of all cases of communication and node failures is an area for future research.

3.2.2 Gradients and Timeouts Involved in the Algorithm

Table 3.1 shows the different timeouts involved in the perimeter detection algorithm. The timeouts in this table are listed in the order at which they are scheduled. It can be seen that a node's timeouts are chained, one after the other, with each timeout's handler being responsible for scheduling the following timeout in the list. Not every node will follow the complete chain of timeouts, especially when no perimeter is detected nearby.

Table 3.1: Timeouts and gradients involved in the perimeter detection algorithm

TIMEOUT_NAME	Timeout Handler May Schedule Timeout	Timeout Handler May Start Gradient of Type	Stages Gradient is Referenced in
DETECT_NEIGHBOURS	*ELECT_CELL_LEADER*	*NEIGHBOUR*	1
ELECT_CELL_LEADER	*ELECT_MEM_LEADER*	*CELL*	2, 6, 7
ELECT_MEM_LEADER	*ELECT_SAT_NODES*	*MEM_LEADER*	3, 4, 5, 6
ELECT_SAT_NODES	*MEM_PERIAMBULATE*	*SAT_NODE*	4, 5
MEM_PERIAMBULATE	*DETECT_ENDPOINT*	*PERIAMBULATE*	5, 6
DETECT_ENDPOINT	*ELECT_MIDPOINT*	*ENDPOINT*	6, 7
ELECT_MIDPOINT	*SPREAD_POINTS*	*MIDPOINT*	7
SPREAD_POINTS	*MEMBRANE_REGROWTH*	*CELLWIDE_MIDPOINT* *CELLWIDE_ENDPOINT*	7, 8 7, 8
MEMBRANE_REGROWTH			

During the handling of a timeout, it is possible the node may choose to start propagating a new gradient. Table 3.1 shows which timeouts may start which gradients. The stages of the algorithm, in which the gradient can be referenced by name, are listed in the last column. During the algorithms' execution, there is an opportunity to free up memory by allowing nodes to remove any gradients which are no longer going to be referenced by the current or future stages.

3.2.3 Starting the Algorithm

There may be a few ways in which the initial stage 1, of neighbourhood detection, is triggered. It may be possible to have all nodes programmed to begin at a predestined point in time. At this time, all nodes would exchange messages, containing their node ID, with their neighbours.

Another simple trigger for stage 1 may be to place a single special starter node, or to use a special starter device, anywhere in the network, whose sole purpose is to flood a gradient throughout the whole network. Any node may begin stage 1, after receiving this initial flooding gradient and waiting a short reasonable delay for nearby nodes to also receive the gradient. During the flood of this initial gradient, all neighbouring nodes should be aware of each other's node ID.

3.2.4 Stage 1: Neighbourhood Detection and Local Leader Selection

The purpose of this stage is two-fold. Firstly, all neighbouring nodes should become aware of each other and *their node IDs*. Secondly, a number of nodes at a *reasonably regular spacing*throughout the entire network become group leaders.

There can be several ways to select local leaders, with varying degrees of sophistication; however, the way described here just uses a very simple competition, akin to a local lottery, to allow nodes to self-select themselves as one of possibly several leaders within the local neighbourhood. This selection competition involves each node choosing a randomly chosen number within the local neighbourhood. If a node realises it has chosen the highest number out of all the nodes in the neighbourhood, it self-selects itself as a group leader.

As an example, figure 3.1 shows some of the self-elected group leaders in local neighbourhoods across the network. Each group is the collection of nodes in the neighbourhood of the group leader node. It is possible for these groups to overlap each other slightly, but a group leader node itself will not belong to another group.

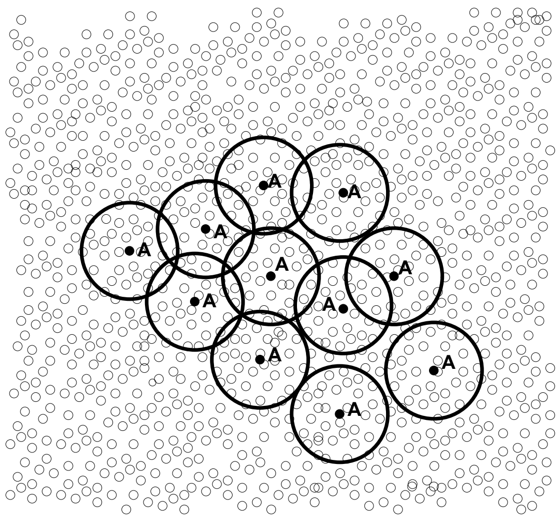


Figure 3.1: Nodes are self-selected as leaders (labelled A) within their local neighbourhoods (shown by the circles).

The pseudocode in listings 3.2.2, 3.2.3 and 3.2.4 show the algorithm from each individual node's perspective during this first step.

Algorithm 3.2.2: $\text{HANDLETIMEOUT}(DETECT\_NEIGHBOURS)$

```
comment: This node has received the initial starting timeout
comment: Make a NEIGHBOUR gradient message
$g \leftarrow \text{MAKEGRADIENT}(NEIGHBOUR)$
comment: Make this NEIGHBOUR gradient distinct to other gradients of the type NEIGHBOUR
$g.GradientID \leftarrow myState.nodeID$
comment: Ensure gradient is constrained to only 1 hop. Each receiving node
comment: only accepts the gradient if this constraint is satisfied
$g.Constraint \leftarrow (g.Hops \leq 1)$
comment: Add our Node ID as an attribute of the gradient
$g.sourceID \leftarrow myState.nodeID$
comment: Start the gradient from this node over 1 hop
$\text{STARTGRADIENT}(g)$
comment: Check later if this node is the elected leader
$\text{SCHEDULETIMEOUT}(ELECT\_CELL\_LEADER)$
```

In listing 3.2.2, each node in the network creates and sends a single gradient of type $NEIGHBOUR$ that is allowed to propagate over just one hop. The $GradientID$ attribute has a special handling by the gradient forwarding framework and separates the gradient from other different gradients with the same type. If the $GradientID$ were the same for two gradients started from different nodes, they would be considered as multiple sources for the same single gradient. $Hops$ is incremented each time the gradient is forwarded. The $Constraint$ attribute is evaluated by the gradient forwarding mechanism on a new node to determine if that node is allowed to participate in the gradient's storage and forwarding.

Algorithm 3.2.3: $\text{GRADIENTDETECTED}(NEIGHBOUR, gradient)$

```
comment: This node has received a new or updated NEIGHBOUR gradient
$\text{ADDTOLIST}(myState.neighbourList, gradient.sourceID)$
```

Listing 3.2.3 shows a node's reception of a $NEIGHBOUR$ gradient that has satisfied the gradient propagation constraint. In this case, the gradient's constraint is simply that its hop count is not allowed to increase beyond 1. After all gradients have finished propagating, every single node has a list of each neighbouring node's ID. The beginning of listing 3.2.4, executed at a time after the gradients are expected to have finished propagating, shows how the highest ID from the list is chosen as the newly elected local group leader.

Algorithm 3.2.4: handleTimeout($ELECT\_CELL\_LEADER$)

```
scheduleTimeout($ELECT\_MEM\_LEADER$)
$leaderID \leftarrow myState.nodeID$
for each $neighbourID \in myState.neighbourList$
  do { if $neighbourID > leaderID$
         then $leaderID \leftarrow neighbourID$
if $myState.nodeID = leaderID$
  then { comment: Send CELL gradient from this leader over $c$ hops
         $g \leftarrow$ makeGradient($CELL$)
         $g.Constraint \leftarrow (g.Hops \leq c)$
         $g.GradientID \leftarrow myState.nodeID$
         $g.sourceID \leftarrow myState.nodeID$
         startGradient($g$)
```

Algorithm 3.2.5: isBetterThan($CELL, newGrad, existingGrad$)

```
comment: This is the default overwrite check
if $existingGrad.Hops = 0$ or $newGrad.Hops \geq existingGrad.Hops$
  then return ($false$)
  else return ($true$)
```

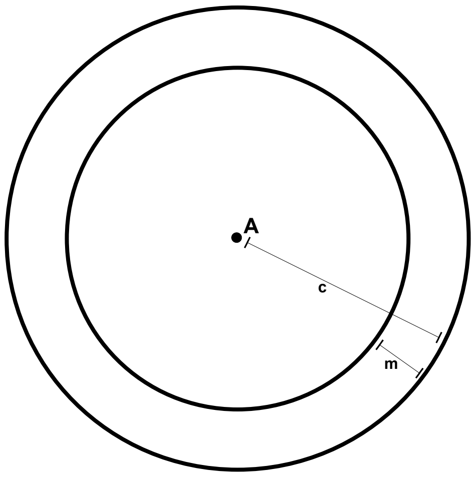
The gradient forwarding mechanism checks whether an incoming gradient is deemed better than an existing gradient of the same type and *GradientID*. If an incoming gradient is better, the existing gradient is overwritten. By default, this check only allows existing non-source gradients to be overwritten if the new gradient has a lower hop count, as shown in listing 3.2.5.

3.2.5 Stage 2: Cell and Membrane Region Formation

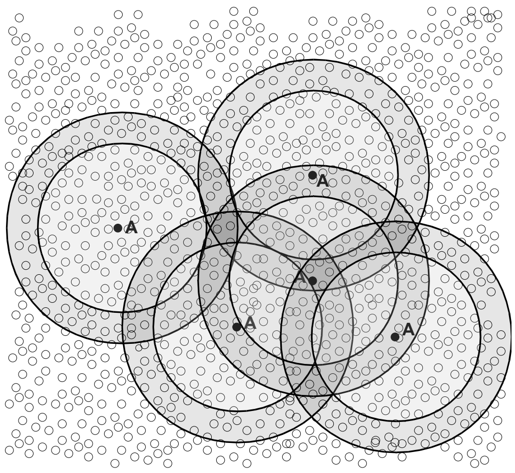
Once these group leaders have been elected, they propagate a gradient for c hops to form many roughly circular groupings, or cells (see listing 3.2.4 and figure 3.2). The nodes lying on the outermost points of these cells, between c and $(c - m)$ hops away consider themselves to form part of the membrane of the cell. The membrane width, m, is chosen to be large enough to ensure a membrane normally contains nodes that are fully connected throughout the membrane.[5] The cell and membrane form two related regions, one overlaid on top of the other. Every member of the membrane region is also a member of the parent cell region.

As shown in figure 3.2(b), all of the different elected leaders emit gradients which cover and overlap other nearby elected leaders and their gradients. After this stage is complete, each node in the network will be a member of several independent cell regions and potentially several membrane regions.

[5]In networks with an average neighbourhood of 15 nodes, values of m at 3 and c around 7 work well. The value of m effectively serves as a crude control of the sensitivity of the algorithm to local disconnected areas.



(a) A single fully formed cell of radius c hops with a membrane of width m hops. The cell leader is shown as A.



(b) Several cell and membrane regions overlapping each other over the network.

Figure 3.2: Gradients from elected leaders form the cell and membrane regions

Algorithm 3.2.6: GRADIENTDETECTED($CELL, gradient$)

```
ADDTOLIST($myState.cellList, gradient.sourceID$)
if $gradient.Hops > (c - m)$
  then ADDTOLIST($myState.membraneList, gradient.sourceID$)

  else REMOVEFROMLIST($myState.membraneList, gradient.sourceID$)
comment: Normal gradient propagation continues...
```

Listing 3.2.6 shows how a node constructs a list of membranes that it currently resides on. It is worth noting that, due to intermittent communications failure and multipath effects, a node may prematurely believe it resides on the membrane. If the node believes it is on a membrane, but receives an updated gradient with a shorter hop count, it will reconsider whether it is still within the cell's membrane region. The duration of stage 2 (see the convergence times in section 3.2.11) should be sufficiently long enough to account for intermittent communications failure and multipath effects.

3.2.6 *Stage 3 and 4: Election of Membrane Leader and Satellite Nodes*

After the formation of the cell membrane, a search for a complete cycle around the group leader along this membrane is performed. If a cycle is found then no further action is taken. If a cycle is not found then it is reasoned that the perimeter of the network, or at least one large locally disconnected area, has perforated some segment of the membrane.

The search for a simple cycle around the membrane involves a number of steps spanning stages 3 to 5. Stages 3 and 4 serve to break the symmetry of the membrane to provide a sense of direction within it. Stage 5 can then use the directions to divide the membrane into three segments and discover if any segment is perforated.

In stage 3 and 4, a membrane leader is elected, by simply designating as leader a node within the membrane that chooses the highest random number. This membrane leader then propagates a gradient amongst all the nodes on the membrane. Nodes that are s hops away from the membrane leader hold a contest to elect a satellite node.[6] In properly formed unperforated membranes these elections result in two satellite nodes that may be used as reference points to give direction (in addition to just a hop count distance) within the membrane relative to the membrane leader. Figure 3.3 shows an example cell with its elected membrane leader and satellite nodes.

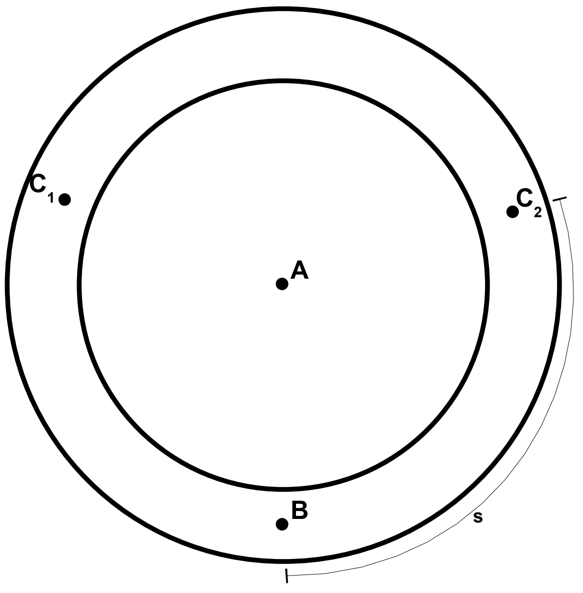


Figure 3.3: A single unperforated cell formed from a cell leader labelled A. B is the elected membrane leader. C_1 and C_2 are the two elected satellites residing s hops from the membrane leader.

Except for the group leaders, the election processes used in this algorithm involve the propagation of one gradient from multiple competing sources constrained to within the area of competition. The area of competition is a temporary region determined by some predicate, such as ‘lies within a hops from b’ or ‘is in region c’. Once the gradient has propagated to cover the entire competition region, the election is complete. It is during the propagation of the gradient, that inferior competing nodes are eliminated from the election. A node with an existing competing gradient that receives a competing gradient from another source will determine which of the two sources is superior and allow it to be propagated whilst removing the inferior gradient from the local state. For convex disc shaped competing

[6] A good value for s is roughly a third of the circumference, in hops, of the membrane.

regions, this mechanism has a time complexity of linear proportion to the region's diameter and a space/communications complexity of logarithmic proportion to the region's diameter. If the diameter of the region is known roughly at the beginning of the election, a future timeout can be scheduled on all nodes to finish the election. When the convergence timeout occurs, all nodes in the competing region will have exactly one winning gradient.

Algorithm 3.2.7: HANDLETIMEOUT($ELECT\_MEM\_LEADER$)

```
SCHEDULETIMEOUT($ELECT\_SAT\_NODES$)
for each $cellID \in myState.membraneList$
  do
    $memLeaderGrad \leftarrow$ GETGRADIENT($MEM\_LEADER, cellID$)
    if $memLeaderGrad = NULL$ or $memLeaderGrad.sourceID < myState.nodeID$
      then
        $g \leftarrow$ MAKEGRADIENT($MEM\_LEADER$)
        $g.GradientID \leftarrow cellID$
        $g.Constraint \leftarrow (cellID \in hostState.membraneList)$
        $g.sourceID \leftarrow myState.nodeID$
        STARTGRADIENT($g$)
```

Listing 3.2.7 shows nodes on the membrane starting a gradient to eventually cover the whole membrane. The *GradientID* is identical for each node on the same membrane so the gradient has many multiple sources.

Algorithm 3.2.8: ISBETTERTHAN($MEM\_LEADER, newGrad, existingGrad$)

```
if ($newGrad.sourceID < existingGrad.sourceID$) or ($newGrad.Hops \geq existingGrad.Hops$)
  then return ($false$)
  else return ($true$)
```

For the election, losing competing membrane nodes are eliminated by allowing their source gradients to be overwritten with better competing node gradients, as shown in listing 3.2.8. After the gradient has finished propagating, only one source remains for each membrane gradient. Each node in the network will have one gradient from the leader of each membrane it belongs to.

Algorithm 3.2.9: HANDLETIMEOUT($ELECT\_SAT\_NODES$)

```
SCHEDULETIMEOUT($MEM\_PERIAMBULATE$)
for each $cellID \in myState.membraneList$
  do
    $memLeaderGrad \leftarrow$ GETGRADIENT($MEM\_LEADER, cellID$)
    if $memLeaderGrad.Hops = s$
      then
        $g \leftarrow$ MAKEGRADIENT($SAT\_NODE$)
        $g.GradientID \leftarrow cellID$
        $g.Constraint \leftarrow$
            ($cellID \in hostState.membraneList$)
            and
            ($g.Hops \leq m$)
        $g.sourceID \leftarrow myState.nodeID$
        STARTGRADIENT($g$)
```

Algorithm 3.2.10: IsBetterThan($ELECT\_SAT\_NODES, newGrad, existingGrad$)

```
if $(newGrad.sourceID < existingGrad.sourceID)$ or $(newGrad.Hops \geq existingGrad.Hops)$
  then return $(false)$
  else return $(true)$
```

The election of the satellite nodes, shown in listings 3.2.9 and 3.2.10, occurs in a similar way to the membrane leader election. The only differences are that the election begins on nodes lying s hops from the membrane leader, and compete in a region on the membrane of width m hops. The width competition region is chosen to be m, the width of the membrane, because all the nodes s hops away from the membrane leader are not necessarily connected neighbours.

3.2.6.1 Constraints on Membrane Width and Cell Size for Electing Satellites

One of the aims of electing the two satellites is to break the symmetry in the membrane. If a potential satellite node is ambiguous to which side of the membrane it lies, it is possible for that single potential satellite node to be elected as both C_1 and C_2 satellite nodes. In such a case, the symmetry is certainly not broken.

This can happen if the satellite election regions from both sides of the membrane overlap, and there is at least one potential satellite that is within m hops of two other potential satellites, but those two other potential satellites are not within m hops of each other. In other words, this situation is somewhat similar to a hidden node problem, where the hidden nodes have all agreed to elect the common visible node. This common visible node is ambiguous as to which side of the membrane it lies.

It is ok for the election regions from opposite sides of the membrane to overlap, but it is unacceptable for there to be any ambiguous potential satellite node. To ensure that the election regions are free of these ambiguous nodes, the arc length of the membrane's inner circumference, between any potential satellite nodes from opposite sides, must be greater than m hops. The membranes inner circumference is the path between nodes that lie $(c - m)$ hops from the cell leader.

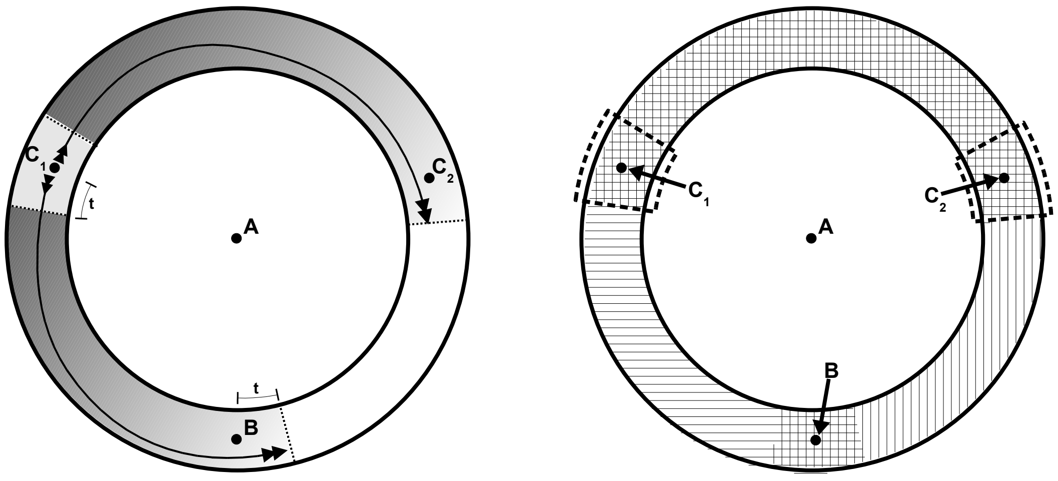
The choice of s clearly has an impact on the above mentioned minimum arc length. If s is too low, the arc length constraint will be violated, and it is possible an ambiguous satellite node could be elected near to the membrane leader. Likewise, if s is too high, an ambiguous satellite node could be elected on the far side of the membrane from the membrane leader.

In practice, these constraints require the inner circumference of the membrane to increase whenever the membrane width is increased. i.e. if the membrane width is increased to the point of violating the arc length constraint, the cell size must be sufficiently increased to satisfy the constraint.

3.2.7 Stage 5: Membrane Periambulation

Following the election of the two satellite nodes, gradients are propagated from both of these satellites within the membrane as shown in figure 3.4(a). After these gradients pass beyond an initial threshold, t, the direction from the satellite in which the gradient is propagating becomes evident by looking at the known hop count from the membrane leader. The gradients are then labelled as either propagating towards the membrane leader or towards the other satellite.

The small threshold t is used to avoid prematurely and incorrectly identifying the direction of the gradient. The gradients are restricted to propagate only to the neighbouring membrane leader or satellite node, plus a small threshold t to ensure a good coverage. A discussion on the constraints on the value of t is deferred until section 3.2.7.1.



(a) The shaded area shows the extent of the propagated gradient within the membrane from just one satellite C_1 and its subsequent discrimination, after the threshold t, into two different directions (towards the membrane leader and towards the other satellite).

(b) The membrane has roughly three segments after both satellite gradients have fully propagated. The horizontal and vertical hatching shows C_1 and C_2's gradients respectively. Within the dotted area, one of the two satellite gradients has an unknown direction, and one has a known direction.

Figure 3.4: A fully formed cell from leader A. B is the elected membrane leader. C_1 and C_2 are the two elected satellites.

Once the propagation of the two satellite gradients has completed, the membrane is roughly divided into three segments as shown in figure 3.4(b). One segment lies between the two satellite nodes where the membrane leader does not lie. The other two segments lie between a satellite and the membrane leader.

In an absence of any break in the membrane, all three of these segments will be complete and it can be considered that at least one simple cycle exists around the membrane. A break in one of these segments at this stage indicates that a cycle around the membrane can not be found, that the membrane is not fully formed and that there must be a nearby local disconnection in the network or network perimeter within vicinity of the cell.

It is worth noting that in membranes without any break, and after the satellite gradients have finished propagating, the nodes within the threshold t of satellites will contain two satellite gradients: one with an unknown direction, and one with a known direction. Without any break in the membrane, every node will be covered by at least one satellite gradient with a known direction (though the algorithm would not fail if any node had both satellite gradients with unknown directions).

Algorithm 3.2.11: HANDLETIMEOUT($MEM\_PERIAMBULATE$)

```
SCHEDULETIMEOUT($DETECT\_ENDPOINT$)
for each $cellID \in myState.membraneList$
do
    $satNodeGrad \leftarrow$ GETGRADIENT($SAT\_NODE, cellID$)
    if not ($satNodeGrad = NULL$) and ($satNodeGrad.sourceID = myState.nodeID$)
    then
        $g \leftarrow$ MAKEGRADIENT($PERIAMBULATE$)
        comment: Quotes are used to denote string literals.
        comment: The CONCAT function returns a concatenated string of its arguments
        comment: e.g. If b is 4, CONCAT("a",b) returns "a4"
        $g.GradientID \leftarrow$ CONCAT($cellID$, "_", $myState.nodeID$)
        $g.Constraint \leftarrow$
            [$lg \leftarrow hostState$.GETGRADIENT($MEM\_LEADER, cellID$)]
            ($cellID \in hostState.membraneList$)
            and
                ($g.direction = UNKNOWN$)
                or ($g.direction = TO\_LEADER$ and $g.Hops < (s + t)$)
                or ($g.direction = TO\_SAT$ and $lg.Hops > (s - t)$)
        $g.sourceID \leftarrow myState.nodeID$
        $g.cellID \leftarrow cellID$
        $g.direction \leftarrow UNKNOWN$
        STARTGRADIENT($g$)
```

Listing 3.2.11 shows the start of the two satellite gradients propagating around the membrane. The constraint on the gradient effectively changes as the gradient is propagating. The direction around the membrane in which the gradient is is propagating is realised after t hops. Listing 3.2.12 shows the modification of the gradient as the direction is realised – affecting the gradient's constraint and subsequent propagation in the process.

Algorithm 3.2.12: GRADIENTDETECTED($PERIAMBULATE, gradient$)

```
if $gradient.direction = UNKNOWN$ and $gradient.Hops > t$
then
    if ($hostState$.GETGRADIENT($MEM\_LEADER, gradient.cellID$)).$Hops < s$
      then $gradient.direction \leftarrow TO\_LEADER$
      else $gradient.direction \leftarrow TO\_SAT$
    comment: Normal propagation continues with this modified gradient
```

3.2.7.1 Constraints on the Periambulation Gradient Threshold

Here we provide the upper and lower bounds on the choice of t - the number of hops before $PERIAMBULATE$ gradients will attempt to discern their direction of propagation.

The threshold t must be strictly greater than the width of the membrane, m. This is to ensure that the gradient's direction is correctly identified. The upper bound on t is half the circumference of the membrane, as there is no reason for the gradients to propagate further (as they will have already covered the entire membrane).

In its current form, the algorithm requires the threshold to let the $PERIAMBULATE$ gradients determine their direction. The reason for these gradients to know their direction is purely to allow an optimisation. It allows the gradient to stop propagating after it has covered one membrane segment in each direction, avoiding the need to propagate fully over the last remaining segment.

In fact, with a few changes to the algorithm, it is possible to remove the need for the threshold t, and its constraints, entirely. This can be achieved by allowing the $PERIAMBULATE$ gradients to propagate throughout the entire membrane, and by changing the detection of any membrane breaks in listing 3.2.13[7].

3.2.8 Stage 6: Endpoint Detection

The presence of a break in one of the segments can be detected by a node looking to the gradients it has received from the satellites and membrane leader. If a node lies on the segment between the two satellites where the leader does not lie, then there is a break in this segment if the node has only received a gradient from one satellite node (as in figure 3.5). Similarly, if a node lies on the segment between a satellite and the membrane leader, but has never received any gradient from the satellite, then it can deduce that the satellite's election never even occurred and that a break in that segment must exist at some point beyond itself (further from the membrane leader). If no satellite gradients are seen by any nodes, then the membrane is so small in length that the election of either satellite did not occur. Some more example perforations of the membrane are shown in figure 3.10.

Once a node detects the presence of a break, it enters a contest with other nodes on the same segment to become the endpoint for that segment. The contest is won firstly by the node with the highest hop count from the membrane leader and in case of a tie, secondly by the distance from the cell leader (any remaining tie can be resolved by simply choosing the node with the highest chosen random number).

[7]Specifically, all nodes on a fully formed membrane will always have exactly 2 $PERIAMBULATE$ gradients after stage 5 has finished. The different parts of any break in the membrane can still be deduced without the $PERIAMBULATE$ gradients' concept of directions.

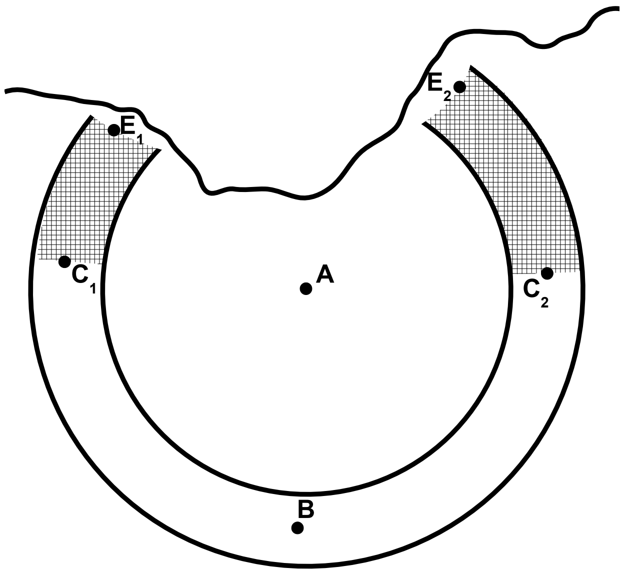


Figure 3.5: Nodes in the shaded segment will have received satellite gradients from only one satellite, so a break in the membrane is detected in this segment. All nodes in the two disjoint shaded regions hold a competition yielding the elected endpoints E_1 and E_2 next to the membrane break.

After this contest, the location of the breaks in the perforated membrane are known and two specific nodes have been nominated as being representative of the furthest frontier of the cell's membrane residing beside a void in the network's connectivity.

Algorithm 3.2.13: HANDLETIMEOUT($DETECT\_ENDPOINT$)

```
SCHEDULETIMEOUT($ELECT\_MIDPOINT$)
for each $cellID \in myState.membraneList$
  do
    $g \leftarrow$ MAKEGRADIENT($ENDPOINT$)
    $isBroken \leftarrow false$
    $memLeaderGrad \leftarrow$ GETGRADIENT($MEM\_LEADER, cellID$)
    $cellLeaderGrad \leftarrow$ GETGRADIENT($CELL, cellID$)
    $satGradList \leftarrow$ GETGRADIENTS($PERIAMBULATE, cellID$)
    if $|satGradList| = 0$
      then
        comment: Break detected in segment between membrane leader and satellite
        $isBroken \leftarrow true$
        if $memLeaderGrad.Hops < s$
          then $g.GradientID \leftarrow$ CONCAT($cellID$, "$NEAR$")            (i)
          else $g.GradientID \leftarrow$ CONCAT($cellID$, "$FAR$")             (ii)
      else if $|satGradList| = 1$
      then
        $satGrad \leftarrow$ GETGRADIENT($PERIAMBULATE, cellID$)
        if not ($satGrad.direction = TO\_LEADER$)
          then
            comment: Break detected between the two satellite nodes
            $isBroken \leftarrow true$                                     (iii)
            $g.GradientID \leftarrow$ CONCAT($cellID$, "_", $satGrad.sourceID$)
          else if ($satGrad.direction = TO\_LEADER$) and ($satGrad.Hops > s$)
          then
            comment: Break detected very close to membrane leader
            $isBroken \leftarrow true$                                     (iv)
            $g.GradientID \leftarrow$ CONCAT($cellID$, "_", $satGrad.sourceID$)
    if $isBroken = true$
      then
        $g.Constraint \leftarrow (cellID \in hostState.membraneList)$
        $g.sourceID \leftarrow myState.nodeID$
        $g.cellID \leftarrow cellID$
        $g.hopsFromMemLeader \leftarrow memLeaderGrad.Hops$
        $g.hopsFromCellLeader \leftarrow cellLeaderGrad.Hops$
        STARTGRADIENT($g$)
```

Algorithm 3.2.14: ISBETTERTHAN($ENDPOINT, newGrad, existingGrad$)

```
if $newGrad.sourceID = existingGrad.sourceID$
  then
    if $newGrad.Hops < existingGrad.Hops$
      then return ($true$)
      else return ($false$)
if $newGrad.hopsFromMemLeader > existingGrad.hopsFromMemLeader$
  then return ($true$)
  else if $newGrad.hopsFromMemLeader = existingGrad.hopsFromMemLeader$
  then
    comment: Prefer endpoint nodes on the outermost part of the membrane
    if $newGrad.hopsFromCellLeader > existingGrad.hopsFromCellLeader$
      then return ($true$)
      else if $newGrad.hopsFromCellLeader = existingGrad.hopsFromCellLeader$
      then
        if $newGrad.sourceID > existingGrad.sourceID$
          then return ($true$)
return ($false$)
```

Listing 3.2.13 shows the different cases in which a node can detect that a break in the membrane is nearby. In this listing, the lines (i), (ii), (iii), and (iv) correspond to the situations shown in figures 3.6, 3.7, 3.8, and 3.9 respectively.

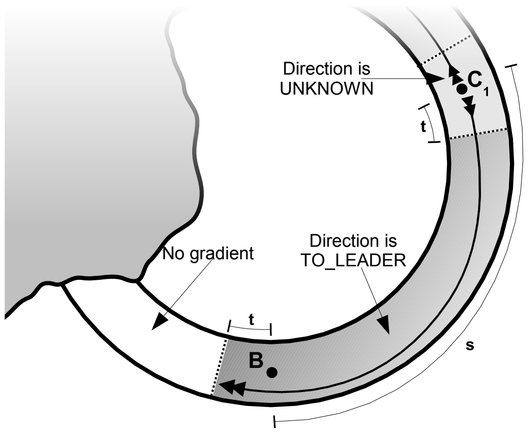


Figure 3.6: A break is present in the membrane where no satellite gradients have propagated. The break is near to the membrane leader, B. C_1 is the only elected satellite node.

Figure 3.6 shows a broken membrane in the segment between the membrane leader and a satellite. The break has prevented one of the satellites from having been elected, whilst the other satellite, C_1, has propagated its gradient to the fullest extent in the direction of the membrane leader. C_1's gradient does not propagate further than t hops past the membrane leader, which leaves those nodes in the broken segment, which are further than t hops from the leader, without any satellite gradient.

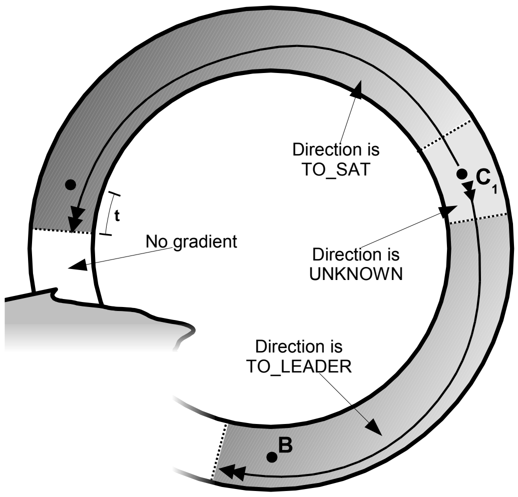
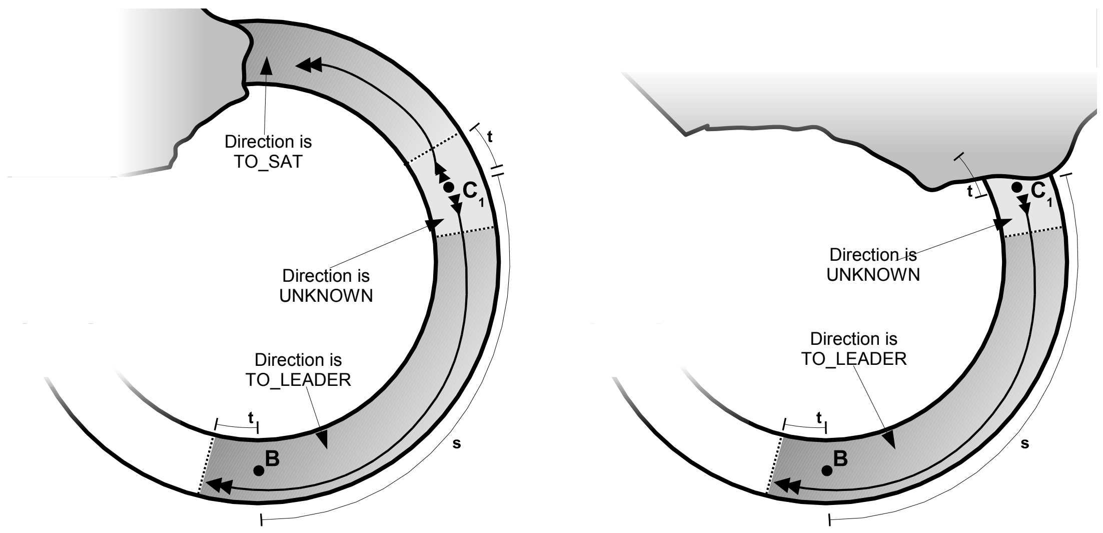


Figure 3.7: A break is present in the membrane where no satellite gradients have propagated. The break is far from the membrane leader. C_1 is the only elected satellite node.

Like in figure 3.6, figure 3.7 shows a broken membrane in the segment between the membrane leader and a satellite. Just as in figure 3.6, the break has prevented one satellite from being elected and there are also nodes in the broken segment which have no satellite gradient (beyond the reach of C_1's gradient). The difference between these two figures is in the hop count, from the membrane leader, of the nodes lying next to the break. In figure 3.7, the hop count of these nodes will be far greater than s, as the membrane leader's gradient has travelled two thirds, or more, of the circumference of the membrane.



(a) The break lies next to nodes which received only one satellite gradient with a direction of TO_SAT

(b) The break is very close to the satellite node and lies next to nodes which received only one satellite gradient with a direction of UNKNOWN.

Figure 3.8: A break is present in the membrane, on the segment between the two satellites.

Figure 3.8 shows two cases where a membrane break could occur in the segment between the two satellites. Whilst only one satellite is shown, it is quite possible for two satellites to have been elected. The nodes on the edge of the membrane break will all have received just one satellite gradient, originating from C_1. Regardless of whether the direction of the satellite gradient is TO_SAT (where $hops > t$, as in figure 3.8(a)) or UNKNOWN (where $hops \leq t$, as in figure 3.8(b)), all these nodes will detect the nearby break in the segment because of the absence of the other satellite's gradient.

The last possible position in which an edge of a membrane break may lie is shown in figure 3.9. The break lies within t hops of the membrane leader, where all nearby nodes have just one satellite gradient. The absence of one satellite gradient indicates a break on the side of the membrane leader where the missing satellite gradient would normally have been elected.

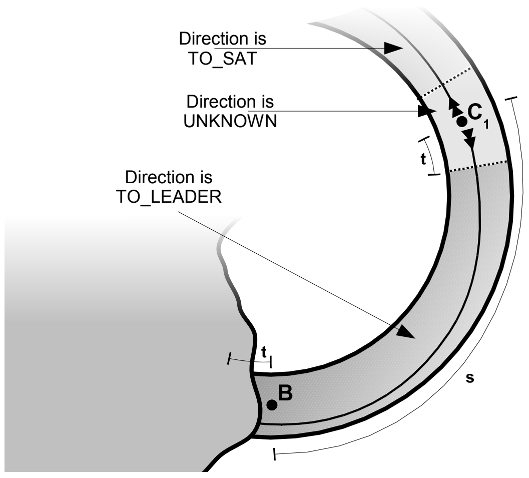
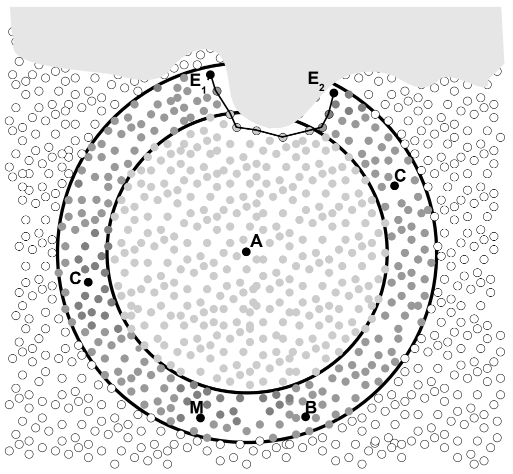


Figure 3.9: A break is present in the membrane, on the segment between the membrane leader and a satellite. This break is very close to the membrane leader.

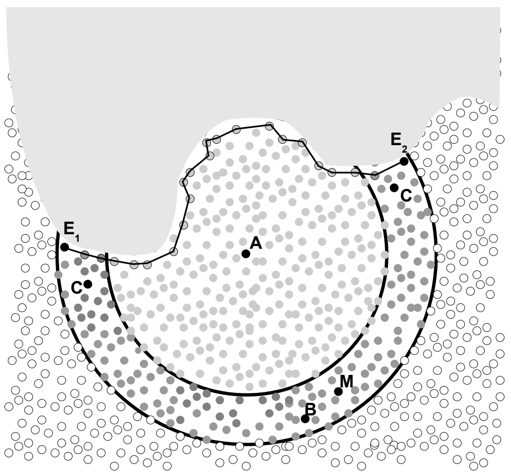
3.2.9 Stage 7: Endpoint and Midpoint Gradient Propagation

We have now discovered two of the nodes from one cell that reside on the perimeter of the network. It can be noted that with a sufficiently dense number of cells formed within the network, there will be enough instances of these endpoint nodes along the perimeter of the network that the problem of discovering the perimeter is already solved. Such a dense number of cells suffers a great communication cost so a different solution is desirable.

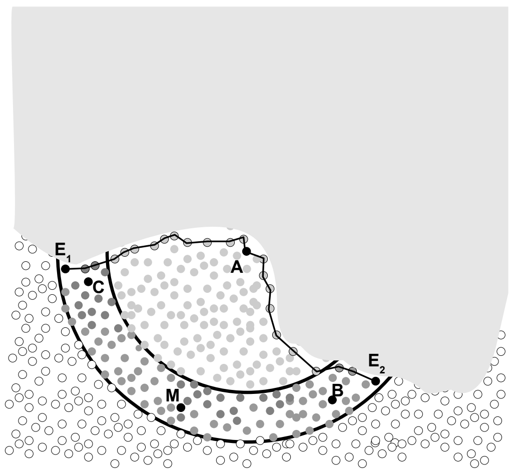
The shortest path along nodes within the cell starting from one endpoint node and ending with the other provides a reasonable approximation of the network perimeter. In many cases this approximation is good, yet the perimeter penetrates the cell to an unknown extent and it is possible that this approximation performs poorly when there is more than just a very shallow penetration by the perimeter. A better approximation is gained by attempting to 'regrow' the membrane on nodes between the two endpoints that lie as close as possible to the expected location of the membrane were it fully formed and unbroken. One way to achieve this is to exert a bias during the formation of the path to push it further towards the edge. This bias and regrowth is described below.



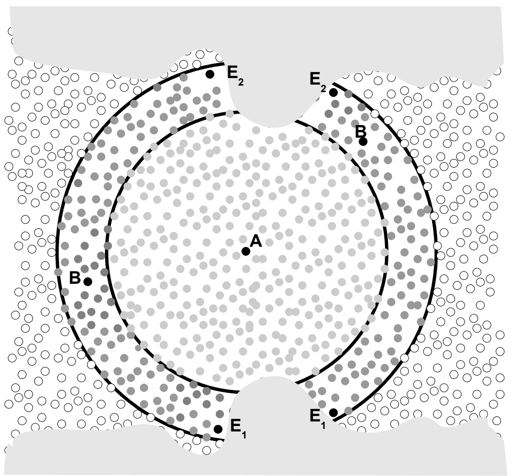
(a) Small perforation of one membrane segment between the two satellites



(b) Perforation of one membrane segment between the two satellites. The shortest path between the two endpoints would be suboptimal but the midpoint gradient guides the path toward the perimeter.



(c) Perforation spanning two membrane segments.



(d) Two separate perforations causing two membrane fragments. In this case the path between the endpoints of the same membrane would be erroneous as would be the use of a midpoint

Figure 3.10: Some examples of cells perforated by the network perimeter and their membrane regrowth path. A is the cell leader, B is the membrane leader, C is a satellite, E_1 and E_2 are the two membrane endpoints, and M is the membrane midpoint roughly halfway between E_1 and E_2.

Once elected, the endpoints each propagate a gradient throughout the membrane as shown in figure 3.11. In short membranes where no satellites have been elected, the contest for nodes to become an endpoint will result in just one endpoint. This particular case is not handled in this algorithm, but if wanted, with a little extra complexity it could be handled, by electing a second endpoint as the node with the highest hop count from the first.

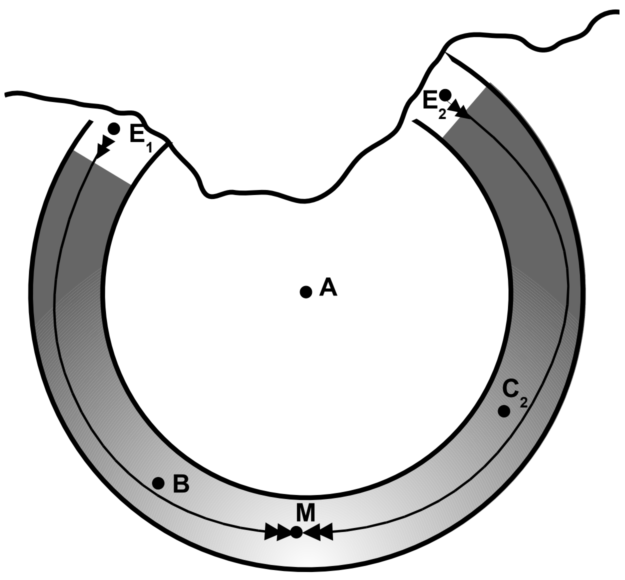


Figure 3.11: The two elected endpoints E_1 and E_2 send a gradient through the membrane to allow an election of a membrane midpoint node, M.

With hop counts from the two membrane endpoints, a single node that lies equally between them both is elected. In the case of a tie in the election, nodes further away from the cell leader are preferred. The elected node becomes the membrane midpoint (see figure 3.11). Before the regrowth of the membrane can begin, all nodes in the cell must know their distances from the endpoints and the midpoint. The midpoint node, and both endpoint nodes, propagate gradients throughout the cell, called CELLWIDE_MIDPOINT and CELLWIDE_ENDPOINT respectively. At the beginning of the membrane reparation, in the handling of the MEMBRANE_REGROWTH timeout, it is always the endpoint with the lower ID that decides to begin the membrane regrowth (see algorithm 3.2.18).

Algorithm 3.2.15: HANDLETIMEOUT($ELECT\_MIDPOINT$)

```
SCHEDULETIMEOUT($SPREAD\_POINTS$)
for each $cellID \in myState.membraneList$
  do
    $endpointGradList \leftarrow$ GETGRADIENTS($ENDPOINT, cellID$)
    if $|endpointGradList| = 1$
      then
        comment: No satellites have been elected so the symmetry is not broken.
        comment: This is a very short segment $< 2s$ hops long.
        comment: Either re-elect closer satellites, or do nothing and ignore this cell.
      else if $|endpointGradList| = 2$
      then
        $endpoint1Grad \leftarrow$ GETFIRST($endpointGradList$)
        $endpoint2Grad \leftarrow$ GETLAST($endpointGradList$)
        $hopDiff \leftarrow$ ABS($endpoint1Grad.Hops - endpoint2Grad.Hops$)
        if $hopDiff \leq 1$
          then
            $g \leftarrow$ MAKEGRADIENT($MIDPOINT$)
            $g.GradientID \leftarrow cellID$
            $g.Constraint \leftarrow$
              ($cellID \in hostState.membraneList$)
              and
              ($g.Hops \leq m$)
            $g.sourceID \leftarrow myState.nodeID$
            $g.hopDiff \leftarrow hopDiff$
            $g.hopsFromCellLeader \leftarrow$ GETGRADIENT($CELL, cellID$)$.Hops$
            STARTGRADIENT($g$)
```

Algorithm 3.2.16: ISBETTERTHAN($MIDPOINT, newGrad, existingGrad$)

```
if $newGrad.hopDiff < existingGrad.hopDiff$
  then return ($true$)
  else if $newGrad.hopDiff > existingGrad.hopDiff$
  then return ($false$)
comment: Prefer midpoint nodes on the outermost part of the membrane
if $newGrad.hopsFromCellLeader > existingGrad.hopsFromCellLeader$
  then return ($true$)
  else if $newGrad.hopsFromCellLeader < existingGrad.hopsFromCellLeader$
  then return ($false$)
if $newGrad.sourceID > existingGrad.sourceID$
  then return ($true$)
return ($false$)
```

Algorithm 3.2.17: handleTimeout($SPREAD\_POINTS$)

```
scheduleTimeout($MEMBRANE\_REGROWTH$)
for each $cellID \in myState.membraneList$
  do {
    $endpointGradList \leftarrow$ getGradients($ENDPOINT, cellID$)
    $endpoint1ID \leftarrow$ getFirst($endpointGradList$)$.sourceID$
    $endpoint2ID \leftarrow$ getLast($endpointGradList$)$.sourceID$
    if ($endpoint1Grad.sourceID = myState.nodeID$)
     or ($endpoint2Grad.sourceID = myState.nodeID$)
      then $gradientType \leftarrow CELLWIDE\_ENDPOINT$
    if (getGradient($MIDPOINT, cellID$)$.sourceID = myState.nodeID$)
      then $gradientType \leftarrow CELLWIDE\_MIDPOINT$
    $g \leftarrow$ makeGradient($gradientType$)
    $g.GradientID \leftarrow$ concat($cellID$, "_", $myState.nodeID$)
    $g.Constraint \leftarrow (cellID \in hostState.cellList)$
    $g.sourceID \leftarrow myState.nodeID$
    startGradient($g$)
  }
```

3.2.10 Stage 8: Membrane Regrowth

At the start of the regrowth, the endpoint holds a token. This token will be passed from node to node until the corresponding endpoint is reached. Each interim node visited by the token is designated as an edge node (some examples are shown in figure 3.10). By simulating a local shared memory amongst neighbours using the idea of a HomePage (Butera 2002) (see section 2.5.1), each neighbour's distances from the midpoint and target endpoint become known to the node currently holding the token. A simple greedy strategy applied locally lends itself well to finding a reasonable path between the two endpoints. The decision of which neighbour to pass the token to next is based on the following heuristics in order of influence:

1. Pass to an unvisited neighbour node furthest away from the midpoint
2. Pass to an unvisited neighbour node closest to the target endpoint
3. Pass back to the previously visited node if there are no unvisited neighbours

The idea of visiting nodes and leaving a mark for a possible later backtracking is also used by Butera (2002), where it is called breadcrumbs.

Note that it may be possible to avoid the endpoints' membrane-wide gradients and the need for a midpoint node altogether. If the arc length of the formed membrane is sufficiently large enough then the existing known hop counts from the cell leader can be used in place of the midpoint gradient's hop counts.

In some situations, such as a network with areas of some local loss of connectivity, the membrane of a cell may be perforated more than once leaving the cell with more than one membrane fragment (as shown in figure 3.10(d)). A node is alerted to this situation if there are gradients from more than one membrane midpoint for the cell. When this happens, the membrane must not be regrown between the endpoints of the same membrane as the network perimeter does not wholly separate the cell in two. Ideally, a path would be grown between the correct endpoints from different membranes fragments, but it is not immediately possible to choose which sibling endpoint to grow towards. This is a rare case and the complexity involved in handling it is probably better substituted by having other nearby cells detecting the disconnected areas.

Algorithm 3.2.18: HANDLETIMEOUT($MEMBRANE\_REGROWTH$)

```
for each $cellID \in myState.membraneList$
  do
    if $|$GETGRADIENTS$(CELLWIDE\_MIDPOINT, cellID)| > 1$
      then
        comment: Ignore this cell, it has multiple membranes. See figure 3.10(d).
        return
    $endpointGradList \leftarrow$ GETGRADIENTS$(CELLWIDE\_ENDPOINT, cellID)$
    $endpointGradList \leftarrow$ SORTLISTBYSOURCEID$(endpointGradList)$
    if GETFIRST$(endpointGradList).sourceID = myState.nodeID$
      then
        comment: This endpoint node begins the membrane regrowth...
        CHOOSENEXTMOVE$(cellID)$

procedure GETUNVISITEDNEIGHBOURS$(cellID)$
  $unvisitedList \leftarrow$ CREATELIST()
  for each $neighbourID \in myState.neighbourList$
    do
      if not EXISTSINNEIGHBOURSHOMEPAGE$(neighbourID,$ CONCAT$(cellID, \text{"}visited''))$
        then ADDTOLIST$(unvisitedList, neighbourID)$
  return $(unvisitedList)$

procedure CHOOSENEXTMOVE$(cellID)$
  $targetEndpointGrad \leftarrow$ GETLAST(GETGRADIENTS$(CELLWIDE\_ENDPOINT, cellID))$
  if $targetEndpointGrad.sourceID = myState.nodeID$
    then
      comment: Other endpoint reached, regrowth is complete
      return
  $unvisitedNodeList \leftarrow$ GETUNVISITEDNEIGHBOURS$(cellID)$
  if $|unvisitedNodeList| = 0$
    then BACKTRACK$(cellID)$
    else
      comment: Sort the unvisited nodes by looking to the gradients in their HomePages.
      comment: First sort by distance from the midpoint node. Furthest come first.
      comment: Second, sort by distance to the target endpoint node. Closer comes first.
      $sortedList \leftarrow$ SORTWITHHEURISTICS$(unvisitedNodeList)$
      MOVETOKEN(GETFIRST$(sortedList), cellID, false)$

procedure MOVETOKEN$(toNeighbourID, cellID, isBacktrack)$
  $g \leftarrow$ MAKEGRADIENT$(TOKEN\_MOVE)$
  $previousMsg \leftarrow$ GETGRADIENT$(TOKEN\_MOVE)$
  if NOTNULL$(previousMsg)$
    then
      comment: We have sent a message before.comment: Make sure this message is seen as an new m
      $g.msgNumber \leftarrow previousMsg.msgNumber + 1$

    else $g.msgNumber \leftarrow 0$
  $g.GradientID \leftarrow myState.nodeID$
  $g.sourceID \leftarrow myState.nodeID$
  $g.toNodeID \leftarrow toNeighbourID$
  $g.cellID \leftarrow cellID$
  $g.isBacktracking \leftarrow isBacktrack$
  $g.Constraint \leftarrow (g.toNodeID = hostState.nodeID)$
  STARTGRADIENT$(g)$

procedure BACKTRACK$(cellID)$
  MOVETOKEN$(myState.previousNode, cellID, true)$
```

Listings 3.2.18, 3.2.20 and 3.2.19 show a slight deviation from the normal use of the gradient forwarding mechanism, for the $TOKEN\_MOVE$ gradient. This gradient is only forwarded to one specific neighbour. The periodic updates of the HomePage will recover from simple communications failures, but it is possible that the message is never received due to a chronic communications failure or node failure. In its current form, the algorithm assumes this passing of the token from one node to another is reliable and the algorithm will fail if the gradient is never received by the target neighbour. A potential remedy is to introduce acknowledgements into the passing of the token, but this, and other failure considerations, is left to future research.

Algorithm 3.2.19: isBetterThan($TOKEN\_MOVE, newMsg, oldMsg$)

```
if $newMsg.msgNumber > oldMsg.msgNumber$
  then return ($true$)
  else return ($false$)
```

Algorithm 3.2.20: gradientDetected($TOKEN\_MOVE, msg$)

```
comment: Only flag nodes as edges if they are not part of the cell membrane
if not $msg.cellID \in myState.membraneList$
  then $myState.isEdgeNode = true$
if not $msg.isBacktracking$
  then $myState.previousNode \leftarrow msg.fromID$
putInMyHomePage(concat($cellID$, "$visited$"), $true$)
comment: Wait for our HomePage update to be sent to neighbours...
waitAShortTimeForUpdate()
chooseNextMove($msg.cellID$)
```

3.2.11 Convergence Times

This algorithm is fully distributed and concurrent. The worst case convergence time can be calculated at the start and is mainly dependent upon the cell radius. Table 3.2 explains the convergence times of each of the algorithm's stages.

3.3 Summary

This chapter has proposed and described in detail a new distributed algorithm which, solely through connectivity information, detects nodes lying next to the perimeter, boundary, or edge, of the network itself. One other distributed edge detection algorithm has been published before (Martincic & Schwiebert 2006), but operates only with knowledge of each node's location – though such knowledge is unavailable before localisation.

Table 3.2: Convergence times of stages in the edge detection algorithm. Refer to section 3.2.1.3 for a description of $MaxMsgDelay$

	Stage	Typical time to complete	Maximum time
1	Detection of neighbours and cell leader election	After one communication packet is received from each node in the neighbourhood	$MaxMsgDelay$
2	Formation of cell and membrane regions	Proportional to the radius of the cell region	$MaxMsgDelay * CellRadius$
3	Membrane leader election	Proportional to half the circumference of the membrane	$MaxMsgDelay * \frac{MembraneCircum.}{2}$
4	Election of satellite nodes	Proportional to the membrane width	$MaxMsgDelay * MembraneWidth$
5	Membrane periambulation	Proportional to arc length between both satellites	$MaxMsgDelay * MembraneSegLen$
6	Endpoint detection	Proportional to arc length between membrane leader and endpoint	$MaxMsgDelay*MembraneCircum.*2$
7	Endpoint and midpoint gradient propagation	Proportional to cell diameter	$MaxMsgDelay * CellRadius * 2$
8	Membrane regrowth	Proportional to arc length between both endpoints	$MaxMsgDelay * CellMembers$

The proposed distributed edge detection algorithm uses no location information. Instead, edge nodes are detected by first attempting to form geometric spatial ring structures over the network nodes and then analysing any topological abnormalities in the formed structures. Specifically, there ought to be a fully connected cycle around each formed ring structure in the absence of any obstacle. Where these rings are broken by obstacles, a path connecting nodes between the broken points on the ring is found (now giving a fully connected cycle around the ring). The nodes on this path (i.e. the nodes involved in regrowing the broken section of the ring) are all flagged as edge nodes.

The next chapter describes the use, and dissemination, of this perimeter knowledge for forming a coordinate system.

Chapter 4

Forming a Coordinate System

This chapter will look at applying existing approaches to node localisation in hop-based sensor networks to our scenario. Sources of accuracy error in these existing approaches are identified and particular focus is given to networks without a strictly convex perimeter, or those containing large areas with no connectivity.

In line with the effort to minimise and eliminate any need for deployment configuration in our system, a new method to create a relative coordinate system without any prior knowledge of relative or absolute node positions is proposed in section 4.2. This new distributed localisation algorithm spreads a relative coordinate system, without any pre-configured reference points, by iteratively electing new reference nodes based upon their perceived geometric dilution of precision (GDOP).

Unlike other hop-based localisation algorithms involving no pre-configured anchors, this new localisation scheme works well in concave networks. This is achieved by selectively avoiding the use of any bad ranging measurements in multilateration. Ranging measurements are deemed unusable if they are known, by use of the proposed perimeter detection algorithm, to be perverted by the network perimeter.

This new proposed localisation algorithm is evaluated by simulations in chapter 5.

4.1 Introduction

In our scenario, we are attempting to paint many computing particles onto a ceiling and then have them collaborate to determine their own relative positions on the ceiling. It is desirable for a localisation algorithm in this situation to be robust to obstructions such as lighting fixtures and non convex shapes such as the ceiling of a L-shaped corridor. Only recently has the sensor network research begun to tackle the problem of these convex network topologies, where previously it was typically assumed that the nodes were connected and spread over a simple square, or convex, area.

Typically, the currently published hop-based localisation algorithms assume that a number of reference nodes are already deployed through the network. These reference nodes, or anchor nodes, would have prior knowledge of their location within the network, gained through either special hardware or manual configuration.

One of the attractive aspects of the scenario is the ease with which the system can be deployed by simply painting it over a ceiling. To retain this ease of deployment, the localisation process needs to have as few, if any, configuration requirements as possible. The minimal action of simply prompting the localisation to begin is ideal. Also, due to their prohibitive cost, nodes capable of determining their position, using special hardware, are precluded from the scenario. This calls for a different means of introducing these known anchor positions in our scenario.

We adopt an approach whereby new anchor nodes are elected, from an initial starting point. As more anchors are elected, the coordinate system slowly spreads across the entire network until all nodes have enough anchors to estimate their location. This propagation of anchors is explained in section 4.2.

4.1.1 Sources of Error in DV-Hop based Localisation

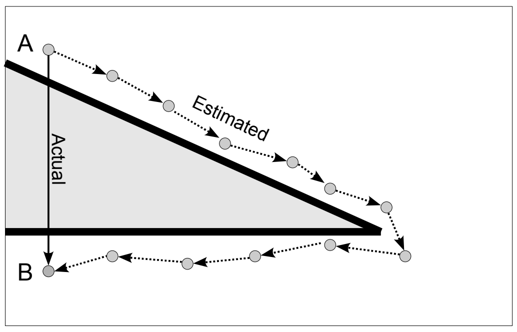
In the APS DV-Hop localisation scheme(Niculescu & Nath 2003*a*), nodes use the hop counts from pre-configured anchor nodes to estimate their location. This section will look to the error characteristics of this localisation scheme. DV-Hop is chosen because its basic operation is very similar to the new proposed localisation scheme, with the exceptions that it assumes pre-configured anchors.

Of particular note is the hop count's ranging error, present in convex networks, arising from the interference of network edges. It will be explained in later sections, how this error may be detected and subsequently avoided in localisation. The avoidance of this error is one of the main goals of the new proposed algorithms.

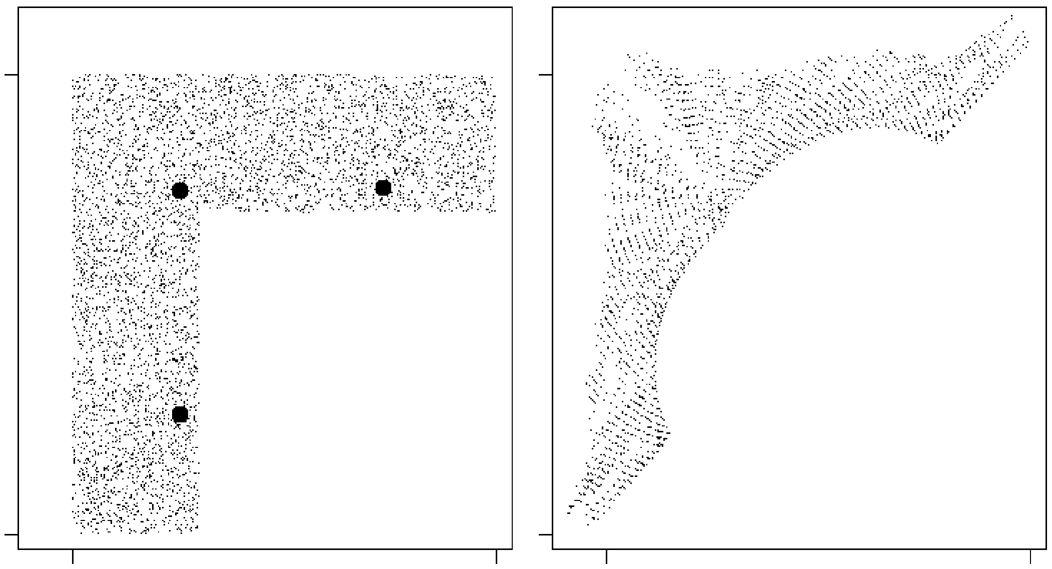
The error in DV-Hop based localisation algorithms in convex networks has been studied before (e.g. Nagpal et al. 2003, Savvides et al. 2003), but the sources of error introduced in concave networks has been given less attention. Current methods to avoid error in concave networks include simply using the closest three anchors (Cheng et al. 2005) or the closest four (Niculescu & Nath 2003*a*) during multilateration. These simple methods do not address the actual cause of the error – the network edges, described in section 4.1.1.1.

4.1.1.1 Edges and Local Failures in Connectivity

The distance measurements may vary with respect to the direction, that is, the network may have a concave perimeter or contain a disconnected region. Indeed, if the distance measurement is obtained through the hop count of a gradient that has travelled along and around an edge of the network, the actual Euclidian distance will be far shorter than this perceived shortest-path distance as shown in figure 4.1(a).



(a) Hop count distance measurement error due to network perimeter



(b) Simulation of multilateration using hop counts to three pre-configured anchors. The network perimeter disrupts the distance measurements of two of the anchors shown.

Figure 4.1: Perturbation of measurements from the network edge and other obstacles

The effect of this measurement error has the potential to distort the position estimation significantly. Figure 4.1(b) shows three anchors with pre-established coordinates. These anchors propagate the coordinates throughout the entire network, hop by hop. Hop counts from the anchor placed in the middle of the network are unaffected from this error. Hop counts from the other two anchors are unaffected for just over half of the nodes in the network, but all remaining nodes incorrectly perceive the anchors as being further away than their actual positions. These exaggerated distance measurements distort the coordinates calculated through multilateration.

Nodes that lie on the perimeter of the network itself have on average fewer anchors to use in multilateration and those anchors that are visible will also be from a limited range of directions. This affects the quality of the estimation for nodes lying close to the network perimeter.

4.1.1.2 Integral Precision of Hop Counts

The hop count of a gradient propagated from one node to another can be used as an estimation of the distance between those two nodes. In other words, the number of edges along the shortest path between two nodes can be used to estimate the Euclidian distance between them. The method of greedy geographic forwarding (Finn 1987), where each node forwards to its neighbour closest to the destination, is shown to provide paths with a low dilation (Bose et al. 2001). The paths from greedy geographic forwarding and MFR (Most Forward within Radius Takagi & Kleinrock 1984) are a good approximation for the shortest path. Nevertheless, the main source of range error in DV-Hop based localisation is due to this estimation and the assumption that each hop progresses the same fixed distance towards the destination.

Localisation under DV-Hop (Niculescu & Nath 2003*a*) takes the average hop counts from neighbouring anchors, and the known locations of those anchors, to calculate the average expected hop distance, d_{hop}, to be:

$$d_{hop} = \frac{\sum \sqrt{(x_i - x_j)^2 + (y_i - y_i)^2}}{\sum h_{ij}}$$

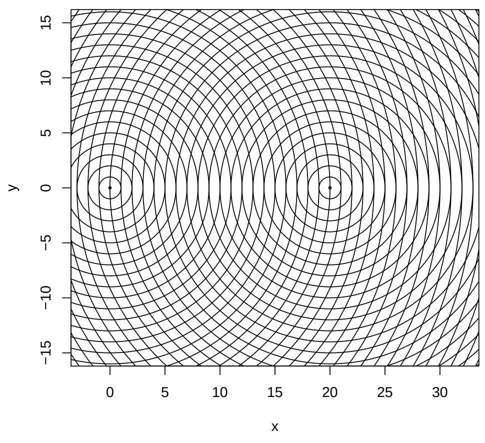
where anchors i and j have locations (x_i, y_i) and (x_j, y_j) respectively. h_{ij} is the number of hops between anchors i and j.

The distance covered by one hop can of course be anywhere up to the node's maximum communication limit, but in a uniform, densely distributed network d_{hop} is shown mathematically by Kleinrock & Silvester (1978) to be:

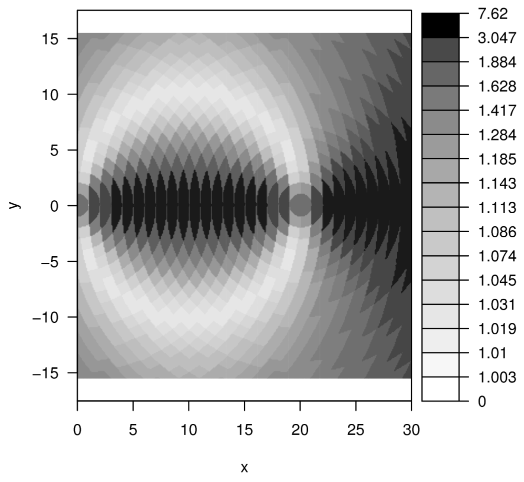
$$d_{hop} = r(1 + e^{-n_{local}} - \int_{-1}^{1} e^{-\frac{n_{local}}{\pi}(\arccos t - t\sqrt{1-t^2})} dt)$$

where n_{local} is the average number of nodes within a node's communication radius neighbourhood r. From the analysis in (Bachrach et al. 2004) it is shown that an average neighbourhood of 15 nodes is the critical number at, and above which, nodes are very likely to lie close to the straight line path and therefore have low error for distance measurements.

Figure 4.2(a) shows the possible hop counts from two nodes as concentric circles. When determining location from the hop counts from these two nodes known locations, an uncertainty arises as to where exactly the node lies within the area of intersection between the concentric circles. This area of uncertainty is calculated and shown in figure 4.2(b).



(a) Concentric circles show distance in hop counts from two nodes located at (0,0) and (20,0)



(b) Area of uncertainty (potential error) in determining position using hop counts from the same two nodes

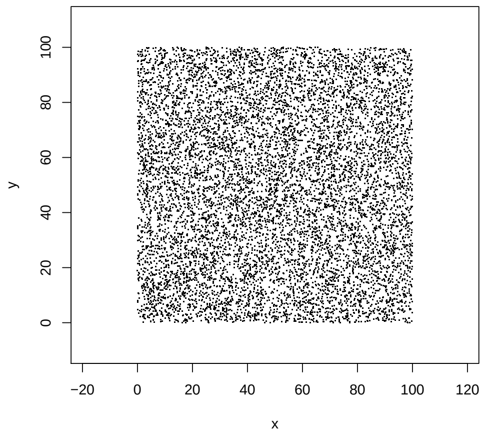
Figure 4.2: The integral nature of the hop counts and the resulting localisation uncertainty when used in lateration.

The influence of the integral hop counts can be seen in figure 4.3 where a simulation of 10000 nodes placed randomly in unique positions (4.3(a)), using two known anchors, produces only 2065 unique estimated positions through trilateration (4.3(b)). Given a higher resolution in distance measurements, a greater number of unique positions can be calculated and the areas of uncertainty become smaller, which results in better position estimations.

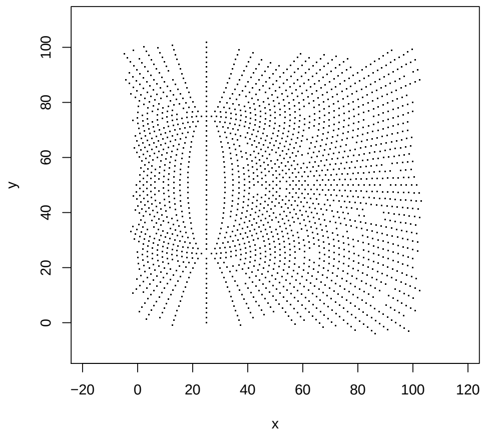
The resolution of the distance measurement through hop counts is clearly limited by their integral nature. Using just the hop count, the measurements will normally have error of at least half the expected hop distance. Described by Nagpal et al. (2003) is a method to improve the distance measurement of hop counts through taking a average of all the hop counts in the local neighbourhood. Nodes surrounded by neighbours with a hop count one higher than themselves are likely to have a real distance closer to their neighbours, greater than their own hop count suggests. This method lowers the measurement error (reduces the area of uncertainty shown in figure 4.2(b)) when there are more than 15 nodes in the neighbourhood. In densities below 15, it is counterproductive and the hop count provides a more reliable measurement.

4.1.1.3 Geometric Dilution of Precision

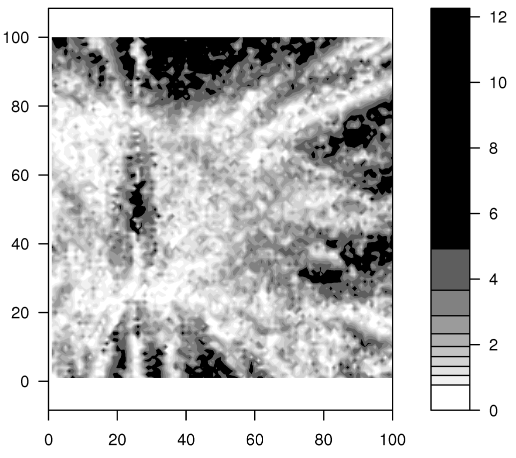
The figures 4.2(b) and 4.3(c) show that a node's localisation error varies due to its position relative to the two known anchors. This effect is known as the geometric dilution of precision (GDOP). In this case with two anchors, if a node lies in a position where the angle between itself and the two known anchors is very large or very small, the distance measurement error is amplified due to their bad relative geometric placement. The error is minimised in locations where the relative angle between the anchors is closest to 90° (Nagpal et al. 2003). Large error is seen where the angle is very large, such as those nodes placed close to the line between the two anchors; however, the greatest deterioration in precision is found in nodes where the angle is very small such as those placed beyond the anchors on the line passing through the two anchors.



(a) Actual positions of nodes. All 10000 positions are unique.



(b) Estimated positions of all nodes. Only 2065 estimated positions are unique.



(c) Error distribution. Average distance from node's actual locations to their estimated locations

Figure 4.3: Simulation of 10000 randomly (poisson) distributed nodes. Nodes were localised from the hops from two known anchors placed approximately 18 hops from each other at (25,25) and (25,75). For clarity, it was assumed that all nodes know on which side of the two anchors they lie.

There has not been an exhaustive exploration of the geometric effect on the error due to the placement of the anchors in sensor networks. It is known, however, that in general, the more anchors used in multilateration results in lower error (Savvides et al. 2003) because there are more constraints on the optimisation algorithm. A number of visible anchors from 8 to 10 is suggested in (He et al. 2005) for low error. As was seen in section 4.1.1.1, this mainly applies to convex networks.

Throughout the localisation literature where this geometric effect is acknowledged, the conclusion typically drawn is that a judicial manual placement of the anchors results in better localisation estimates than a purely random placement (Bulusu et al. 2001). It is normally suggested that the best placement of anchors is along the perimeter of the network (e.g. Doherty et al. 2001, Nagpal et al. 2003). This placement works well because most nodes reside within the convex hull of the anchors and therefore are more likely to avoid the degenerate case where the node's angle to the anchors is very small or zero (Savvides et al. 2003).

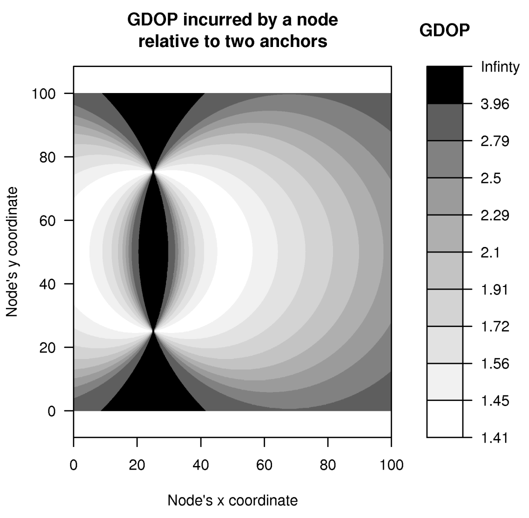
During localisation when using two anchor nodes i and j in trilateration, a simple metric to assess a localising node's vulnerability to error due to the anchor's geometric placement is $|\theta_{ij} - (45°)|$, where θ_{ij} is the angle between nodes i and j relative to the localising node. When a set of N anchors are used in multilateration, another metric can be used to determine the localising node's GDOP (Spirito 2001):

$$GDOP = \sqrt{\frac{|N|}{\sum_{i \in N} \sum_{j \in N, j > i} \sin {\theta_{ij}}^2}} \tag{4.1}$$

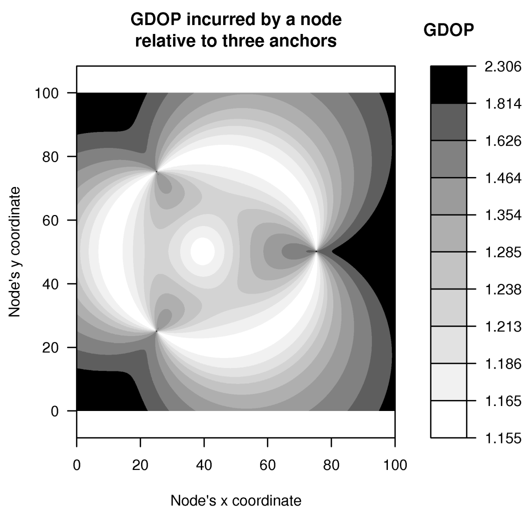
This GDOP metric is a measurement of the geometric conditioning itself, separate from the distance measurement errors, and assumes that all the anchor nodes used have perfect location information. This GDOP calculation can be used, and is used later in the proposed localisation algorithm in section 4.2, to measure the potential scale of the error involved in a node's location estimate.

The GDOP values involved in some different geometric anchor placements is shown in figure 4.4. Figures 4.4(a), 4.4(b) and 4.4(c) show good geometric placements of anchors. It can be seen that as the number of anchors increase, the GDOP can only decrease. Figure 4.4(d) shows a bad placement of three anchors that are nearly lying on a line and will, as can be seen from the upper bound on the GDOP values, cause considerably large error for nodes lying in certain areas. The placement in 4.4(d) is far worse than the same number of anchors in 4.4(b) and only marginally better than 4.4(a) where one less anchor is involved. In all of the figures it can be seen that the GDOP tends to be very bad for nodes lying on the far outer side of anchors (that is, outside the convex hull around the anchors). To a lesser extent, a dip in the GDOP is seen on the inner side at points very close to the anchors.

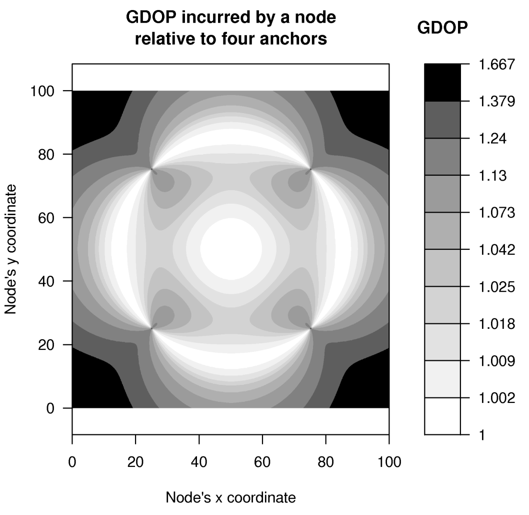
The proposed localisation algorithm in section 4.2 uses this GDOP calculation for deciding upon good locations for introducing new anchor nodes into the network.



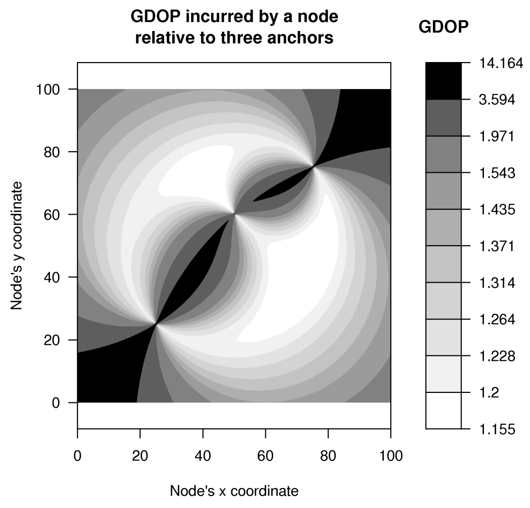
(a) Two anchors at (25,25) and (25,75)



(b) Three anchors at (25,25), (25,75) and (75,50)



(c) Four anchors at (25,25), (25,75), (75,25) and (75,75)



(d) Bad geometric placement of three near-collinear anchors at (25,25), (50,60) and (75,75)

Figure 4.4: GDOP distribution for nodes relative to different anchor placements

It is the guiding metric that protects the proposed algorithm's spreading coordinate system from choosing degenerate anchor placements.

4.1.1.4 Algorithmic

Further error can occur from the localisation algorithm itself due to effects such as the finite numerical precision or compromises made to reduce communication and computation costs. Multilateration is effectively a non-linear optimisation problem and the method used to solve it may not be entirely robust. The algorithmic error can be arbitrarily large.

This remainder of this section is primarily here, for sake of completeness, to describe the standard optimisation algorithms used in multilateration.

The estimator, or optimisation algorithm used in localisation operates on the system of quadratic distance measurement equations, whose solution give the estimated coordinates of the unknown node:

$$(x_i - x_j)^2 + (y_i - y_j)^2 = r_{ij}^2 \tag{4.2}$$

for $j = 1, \ldots, N$, where j is an anchor node, i is the unknown localising node and r_{ij} is the estimated range between i and j. The problem is to find some location for (x_i, y_i) that minimises the total squared error between the estimated/calculated Euclidean ranges, r_{ij}, and the observed ranges derived from the shortest path hop counts, $\hat{r}_{ij}$:

$$f(x, y) = \sum_{j=1}^{N} (r_{ij} - \hat{r}_{ij})^2$$

There are two established standard methods (Reichenbach et al. 2006) of linearising the system of N equations in equation 4.2 to facilitate an efficient computation: a Taylor series method and a linearisation tool.

An expansion of the first few terms in a Taylor series around an initial estimated point (neglecting the terms of second order and higher) is used to linearise the equations and obtain a refined estimate (Foy 1976). This is then repeated iteratively until the desired accuracy is achieved:

$$f(x_i + \Delta x, y_i + \Delta y) = f(x_i, y_i) + (\Delta x \frac{df}{dx} + \Delta y \frac{df}{dy})$$

The partial derivatives are:

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{N} (x_i - x_j)(1 - \frac{r_{ij}}{\hat{r}_{ij}}) \text{ and } \frac{\partial f}{\partial y_i} = \sum_{j=1}^{N} (y_i - y_j)(1 - \frac{r_{ij}}{\hat{r}_{ij}})$$

The iterative updates to the estimated coordinates are:

$$\Delta x_i = -\alpha \frac{\partial f}{\partial x_i} \text{ and } \Delta y_i = -\alpha \frac{\partial f}{\partial y_i}$$

where α is chosen between 0 and 1. This gradient descent method is used for multilateration in all the simulations in this thesis.

A least squares minimisation via this gradient descent method provides an accurate solution provided that a good initial estimate is used. If the initial estimate is poor, there may be a very slow convergence or the solution may converge to a local minima. There is also a sensitivity to anchor placements with bad geometric placements.

The linearisation tool subtracts one equation from the rest of the set of equations in (4.2). If we subtract the first equation (where $j = 1$) we obtain:

$$r_{ij}^2 - r_{i1}^2 = 2(x_1 - x_j)x_i + x_j^2 - x_1^2 + 2(y_1 - y_j)y_i + y_j^2 - y_c^2$$

for $j = 2, \ldots, N$. This set of equations can be written in a linear algebraic form as $\mathbf{Ax} = \mathbf{b}$ where

$$\mathbf{A} = \begin{pmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \\ \vdots & \vdots \\ x_N - x_1 & y_N - y_1 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{N1} \end{pmatrix}$$

This is the linear form which can be solved using the linear least squares method by:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{4.3}$$

This method requires no initial estimate and no mathematical differentiation whilst still remaining efficient. Unfortunately, different solutions are obtained depending on which equation is subtracted from the others. If the equation chosen to be subtracted contains a distance measurement with large error, the solution becomes inaccurate.

A compromise can be made which still arrives at an accurate solution and avoids the need to use a reliable initial estimate. This is possible by using the solution to equation (4.3) as the initial estimate when solving the non-linear system in equation (4.2) (Shang, Shi & Ahmed 2004).

4.1.1.5 Outliers

The ranging measurements that are very distant to all other measurements used in multilateration are called outliers. It is often the case that the optimisation algorithms used in multilateration are not robust and are sensitive to the presence of one or more of these outliers resulting a bad estimation. Some optimisation algorithms are more robust to outliers but tend to be more computationally expensive, which makes them undesirable for use on small resource constrained nodes. A more desirable approach is to use a set of measurements without outliers.

It can be possible to identify and filter out some outliers before multilateration. Some outlier rejection algorithms are specific to the ranging technology itself. For example, some interference effects seen in RF or ultrasound distance measurements may only with a low likelihood be seen in more than one sample. These outliers can be detected and avoided by looking at a history of repeated samples at different times.

Other outlier rejection algorithms perform by using a simple geometric check against each distance measurement to ensure that they are logically consistent (Kwon et al. 2005). Two nodes may each independently measure the distance to each other and ensure that the two measurements are similar and discarded otherwise. If three nodes have measurements to each other, a triangle inequality check can be made to ensure that any one measurement is less than the sum of the other two and also greater than the difference between the other two measurements. This approach has the drawback that it is not known which of the measurements is the faulty outlier.

This thesis shows a method for hop based networks to identify exactly which hop based distance measurements actually had no clear line-of-sight. These distance measurements are considered outliers affected by the network perimeter and filtered out from use in multilateration.

4.1.1.6 Further Errors in Hardware Implementation

Further errors can be incurred due to the particular hardware implementation used. For example, if communication was implemented using RF or ultrasound, the transmission coverage pattern of a transmitting node may be irregular and prone to various effects such as multipath propagation, interference and shadowing. It could also be the case that the hardware could allow for some optimisations, such as the use of Received Signal Strength Indication (RSSI) to provide a greater resolution to distance measurements than is possible with simple integral hop counts.

In this thesis we are not committing to any specific hardware implementation of such a multi-hop network, so the error characteristics of the hardware are an issue for separate research.

Within this section we shall briefly look to the current approach for detecting the perimeter of the network. In the following section, we propose a new approach that is more applicable in the context of localisation.

4.2 Propagation of Anchors

This section proposes a new localisation algorithm from simple node connectivity which involves no initial anchors, as with MDS-Map (Shang et al. 2003, Shang, Ruml, Zhang & Fromherz 2004). Unlike MDS-Map, it also operates in concave networks where the network perimeter causes significant ranging error. Anchor nodes are instead elected, within a single starting region, and continue to be elected on nearby nodes until every node in the entire connected network is suitably close to enough good anchor nodes for a reliable multilateration. This is in contrast to established anchor based localisation schemes where the anchor nodes are already distributed throughout the network and already know their location reference.

Anchor nodes are deemed to provide the most accurate reference location in that region of the network. Anchors cannot be placed manually in geometrically advantageous positions with near perfect location information, so we must instead use metrics to evaluate the suitability and accuracy of nodes as potential new anchors. This is of crucial importance as, unlike networks with pre-configured anchors each with similar levels of error distributed throughout the network, the propagation of new anchors according to our algorithm will inevitably accumulate error as the distance from the initial starting point increases and as the generation of new anchors increases. Each successive generation of new anchor has its own accuracy bounded by the accuracy of the parent anchors used in the multilateration. Successive anchors and will effectively inherit their parent's accumulated error in addition to their own error contribution.

A node's perceived location estimation accuracy is calculated by its GDOP in relation to the anchors used in multilateration. The anchors used in a node's multilateration are selected according to the known risk of error in the range measurements to the anchors. The risk of error can be judged by the node to be unacceptably high for range measurements from anchors that are known to reside in a location beyond a sizeable break in the network connectivity such as a corner of the network perimeter.

The proximity to the network perimeter that ranging measurements have encountered is made possible using the proposed algorithm introduced in section 3.2. Anchors disseminate their estimated location by means of propagating a gradient along a set number of hops. As the gradient message is passed from node to node, a count of the number of edge nodes passed can be updated in it. When a gradient spreads over the nodes in the network space, any forwarding edge node casts a 'shadow' of uncertainty to fall on the rest of the gradient beyond that edge node (see figure 4.5).

This simple gradient shadow mechanism allows for any node receiving the gradient to know the degree of risk of error in the distance measurement as based on the gradient's hop count. If a gradient is received that has passed no edge nodes, some confidence is allowed that the resulting range measurement will be closer to the shortest path between the two nodes than would a gradient having passed many edge nodes.

The only manual intervention assumed by this algorithm is the selection of a single node as an initial starting point for the formation of the coordinate system. Even this assumption may be relaxed by simultaneously running the algorithm on locally elected nodes throughout the network and then stitching all of the local

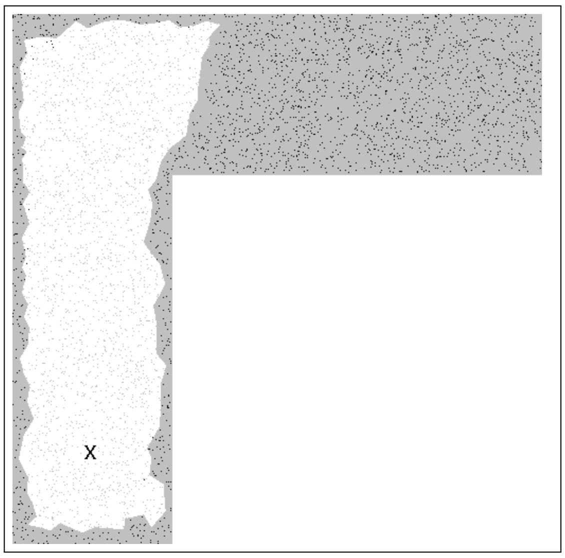


Figure 4.5: Gradients forwarded by known edge nodes cast a shadow of doubt over their hop based distance estimates. X marks the source of the example gradient.

coordinate systems together as the anchors are propagated, or more simply by marking the spreading coordinate systems with the initiating node's ID and only allowing the spread of the coordinate system with the highest ID when two systems begin to overlap. For clarity, the system described here assumes a single node has been selected manually as the starting point of the coordinate system to be formed. In a deployment scenario after the network has been placed over a surface, it is not unreasonable to imagine some external I/O device placed in one location solely for the purpose of beginning the edge detection and coordinate formation process. It is not assumed, however, that this starting node is chosen to be in any strategic location.

Pseudocode listings are included throughout this chapter to outline the algorithm and clarify the narrative. However, some detail and complexity has been omitted for conciseness. Please refer to appendix A for a full Java implementation of this algorithm.

Having one node as a starting point, three tokens are passed separately from node to node, according to some decision function, until they are approximately equidistant from each other. As a token is passed to a new node, a gradient is propagated from the new node and the previous gradient from the passing node is ceased. These shifting gradients allow the three token carrying nodes to determine their current distances from each other. If the hop counts are considered as spring forces, the placement problem is analogous to the resolution of these forces (as explored by Butera (2002)). The decision function is chosen to resolve the forces and choose the next node from the neighbours to be that which has the least potential energy $\sum_{i=1}^{3}(h_i - h_{ideal})^2$ (where i is the token ID, h_i is the neighbouring node's hop count to token i and h_{ideal} is the ideal target hop distance to the other tokens). After the resolution of these forces, the three tokens will have migrated into a roughly triangular configuration.

The node holding the first token, $i = 1$, becomes the origin of the new coordinate system with coordinates $(0, 0)$. The node with the second token defines one axis and has coordinates $(h_1, 0)$. The third token does not provide any coordinates as it is likely not to be in a favourable geometric position relative to the first two tokens (i.e. its relative angle to the first two tokens is probably not the closest to $90°$ possible from all of the nodes in the area). The third token is simply used to distinguish the positive and negative sides of the y axis. The nodes holding the first two tokens become the initial first two anchors and both propagate a gradient with their location.

Within the local area, a node is elected to become the third anchor. This is chosen to be the node that has the relative angle to the first two anchors closest to $90°$. This node then becomes the third initial anchor and its coordinates are calculated using simple trigonometry. The sign of the y axis is adjusted depending on which side of the first two anchors the node lies, given by its distance to the node holding the third token. With the three initial anchors, the initial phase is now complete and the coordinate system can be spread by introducing new anchors in favourable geometric locations.

When a node receives a new anchor gradient, it estimates its location if three or more good anchor gradients are available. Good anchors are defined as those whose gradients have not passed through any edge nodes. If a new estimate is gained, it is recorded in the anchor gradient before propagating it on to neighbouring nodes. This allows a good close initial location estimate to be used by subsequent nodes for the multilateration algorithm and avoids the local minima convergence problem. As a good initial estimate is available, the multilateration problem is solved by gradient descent (Nagpal et al. 2003), as described in section 4.1.1.4.

Algorithm 4.2.1: GRADIENTDETECTED($ANCHOR, gradient$)

```
if $myState.isEdgeNode = true$
  then $gradient.edgesPassed \leftarrow gradient.edgesPassed + 1$                    (i)
$unadulteratedAnchors \leftarrow$ GETFILTEREDANCHORS()
if $|unadulteratedAnchors| \geq 3$
  then
    $initialEstLoc \leftarrow$ GETAVGINITIALESTLOC($unadulteratedAnchors$)
    $myState.estimatedLoc \leftarrow$ MULTILATERATE($unadulteratedAnchors, initialEstLoc$)
    if ISACCEPTABLEFORNEWANCHOR($myState$)                                         (ii)
      then SCHEDULERANDOMTIMEOUT($ESTABLISH\_ELECTION\_REGION$)
    $gradient.initialEstLoc \leftarrow myState.estimatedLoc$                       (iii)

procedure GETFILTEREDANCHORS()
  $filtered \leftarrow NULL$
  for each $anchorGrad \in$ GETGRADIENTS($ANCHOR$)
    do
      if $anchorGrad.edgesPassed = 0$
        then ADDTOLIST($filtered, anchorGrad$)
  return ($filtered$)
```

Listing 4.2.1 shows the pseudocode, from an individual node's perspective, that deals with newly received anchor gradients. These received gradients are modified to contain the new estimated location (line (iii)) and possibly an incremented edge node count (line (i)) before forwarding them on to neighbours.

When a node has a new estimate of its location it will consider entering into a competition with other nearby nodes to become a new anchor (as on line (ii)). The node will only compete for becoming a new node if it is within a certain number of hops from the closest anchor. A minimum number of hops is enforced to ensure both that the density of anchors does not become too high and that a reasonable distance is covered in the propagation of the coordinate system. A maximum number of hops from the closest anchor is also enforced to ensure a minimum density of anchors. This is a simplistic anchor density control mechanism and can be improved upon, particularly in some situations which can benefit from a varying density of anchors such as very narrow areas of the network sandwiched between the network perimeters.

The election of a new anchor is split into three stages. In the first stage, the election region is formed. The second stage sees all the member nodes of the election region competing to become the new anchor. The third stage removes the election region. This careful staging of the election is to ensure that no two election regions overlap each other. As only one election can occur in one place, only one participating node, hopefully the best node, will become the new anchor in that area. The election region size can be controlled to be large enough for a sufficient number of nodes to be considered, and to be small enough to converge quickly and allow other neighbouring region competitions to occur simultaneously.

If the election regions overlapped, problems could arise, such as two new anchors elected next to each other violating the distance constraints placed on them. If overlapping regions were to be allowed, the overlapping elections must be synchronised with each other to avoid problems – leading to very slow convergence times and complicated code.

4.2.1 Election Stage 1: Establishing the Region

Nodes that are acceptable for becoming new anchors and wish to start a new competition region first schedule a timeout event to occur in the future. The time of this event is set to occur after the anchor gradient has fully propagated and all other acceptable nodes covered by the gradient have also scheduled their own competition timeouts. A large random delay is added to each scheduled time in order to fairly allow each acceptable node a chance at becoming the centre of the new election region.

Algorithm 4.2.2: HANDLETIMEOUT(*ESTABLISH_ELECTION_REGION*)

```
$g \leftarrow$ MAKEGRADIENT($ELECTION$)
$g.GradientID \leftarrow myState.nodeID$
$g.Constraint \leftarrow (g.Hops \leq ElectionRadius)$
$g.stage \leftarrow ESTABLISHING\_REGION$
$g.isPoisoned \leftarrow false$
STARTGRADIENT($g$)
SCHEDULETIMEOUT($ELECTION\_COMPETE$)
```

As shown in listing 4.2.2, when the competition timeout event is triggered, the node becomes the centre of a new potential election region and starts propagating an gradient that will form the region. This centre node schedules another timeout event to check if the region has been formed and to proceed to the next stage.

Algorithm 4.2.3: GRADIENTDETECTED($ELECTION, gradient$)

```
UNSCHEDULETIMEOUT($ESTABLISH\_ELECTION\_REGION$)
$establishingRegions \leftarrow NULL$
$competingGrad \leftarrow NULL$
for each $grad \in$ GETGRADIENTS($ELECTION$)
  do
    if $grad.stage = ESTABLISHING\_REGION$
      then ADDTOLIST($establishingRegions, grad$)
    else if $grad.stage = COMPETING$
      then $competingGrad \leftarrow grad$
$bestFormingGrad \leftarrow NULL$                                        (i)
$highestGradID \leftarrow -1$
for each $grad \in establishingRegions$
  do
    if $grad.GradientID > highestGradID$
      then
        if not $NULL = bestFormingGrad$
          then $bestFormingGrad.isPoisoned \leftarrow true$
        $bestFormingGrad \leftarrow grad$
        $highestGradID \leftarrow grad.GradientID$
if not $competingGrad = NULL$
  then
    if not $NULL = bestFormingGrad$
      then $bestFormingGrad.isPoisoned \leftarrow true$                  (ii)
    if $NULL = myState.gdopMetric$
      then
        comment: First time this node has seen the region competing..
        $myState.gdopMetric \leftarrow$ GETGDOP(GETFILTEREDANCHORS())
        if $myState.gdopMetric < competingGrad.winningMetric$            (iii)
          then
            $competingGrad.winningMetric \leftarrow myState.gdopMetric$
            $competingGrad.winningID \leftarrow myState.nodeID$
            SCHEDULETIMEOUT($ELECTION\_COMPLETE$)
```

Listing 4.2.3 shows the steps taken by each node that receives an election gradient (the corresponding propagation rule is given in listing 4.2.5). All receiving nodes suppress any other region forming by removing their existing scheduled timeout for establishing a new region. If the receiving node is already participating in another election region that has already reached the second competing stage, then it 'poisons' any and all of the election gradients that are in the first formation stage (line ii). If all election gradients are in the first stage then all but one of them, with the highest ID, are marked as poisoned (line i).

Poisoning a gradient is the method by which an election will submit and withdraw in the vicinity of another election that takes precedence. A gradient marked as poisoned will propagate this fact to all nodes closer to the gradient's source node (this is achieved through the ELECTION gradient's propagation rule, shown later in listing 4.2.5). When the election gradient's source node processes the timeout event for the second stage of the election, it stops the gradient and relinquishes all interest in holding the poisoned election.

Once a gradient has been stopped by the source node, the gradient's decaying process begins to remove the gradient from nodes' HomePages. The decaying process is part of the housekeeping of the gradient forwarding mechanism, as described in section 3.2.1.4, and is separate from the poisoning of gradients. The poisoning of gradients is part of the mechanism for obtaining an exclusive lock on a spatial region, specific to this algorithm; the decaying of a gradient is part of the general gradient forwarding mechanism for the eventual removal of stopped and unwanted gradients.

4.2.2 Election Stage 2: Competition Within Region

The second stage of the election involves the competition amongst nodes in the formed and non-poisoned election region. The timeout event for this stage was timed to make sure that the election region is fully formed and poison, if any, has already reached the source node.

Algorithm 4.2.4: HANDLETIMEOUT(*ELECTION_COMPETE*)

```
$eg \leftarrow$ GETGRADIENT($ELECTION, myState.nodeID$)
if $eg.isPoisoned = true$
  then STOPGRADIENT($ELECTION, myState.nodeID$)
  else
    $myState.gdopMetric \leftarrow$ GETGDOP(GETFILTEREDANCHORS())
    $eg.Constraint \leftarrow (g \in hostNode.$GETGRADIENTS($ELECTION$))
    $eg.stage \leftarrow COMPETING$
    $eg.isPoisoned \leftarrow false$
    $eg.winningMetric \leftarrow myState.gdopMetric$
    $eg.winningID \leftarrow myState.nodeID$
    SCHEDULETIMEOUT($ELECTION\_COMPLETE$)
```

When the election region is fully formed and no other nearby election has taken precedence, the source of the election gradient is manipulated to begin the second stage of the election (see listing 4.2.4). The manipulated gradient fills the entire election region – i.e. it is constrained to propagate on to each node that already contains the same election gradient.

Nodes involved in the election competition provide their GDOP value as a testament to their potential strength as a new anchor.[1] The GDOP value is calculated as per equation 4.1 in section 4.1.1.3 using the locations of the anchors that were used for estimating the node's location.

If another nearby election has taken precedence, the source node's gradient will be flagged as having been poisoned by another election. This poisoned source node does not continue the election and the poisoned gradient is stopped, removing the election region and freeing up state.

[1]The contender gradients can certainly include further distinguishing information describing the suitability of the node as an anchor, such as known degrees of hardware measurement error, if it is available

Algorithm 4.2.5: isBetterThan($ELECTION, newGrad, existingGrad$)

```
if $newGrad.state = COMPETING$
  then {
    if not $existingGrad.state = COMPETING$                                  (i)
      then return ($true$)
    if $newGrad.winningMetric < existingGrad.winningMetric$                  (ii)
      then return ($true$)
    if $newGrad.winningMetric > existingGrad.winningMetric$
      then return ($false$)
    if $newGrad.winningID > existingGrad.winningID$
      then return ($true$)
    return ($false$)
  }
  else if $existingGrad.state = COMPETING$
  then return ($false$)
if $newGrad.isPoisoned = true$ and $existingGrad.isPoisoned = false$         (iii)
  then return ($true$)
if $newGrad.isPoisoned = false$ and $existingGrad.isPoisoned = true$
  then return ($false$)
if $existingGrad.Hops = 0$ or $newGrad.Hops \geq existingGrad.Hops$          (iv)
  then return ($false$)
  else return ($true$)
```

Listing 4.2.5 shows how the election gradient's propagation is handled depending upon the stage of the election. If the election region is competing in the second stage then it is impervious to poison and gradients with better GDOP values are allowed to overwrite gradients with inferior values (line ii).

If a node with an existing election gradient in the formation stage receives an update of the same gradient but in the competing state (as a result of line i) then it will check if it is a better candidate for a new anchor. If the node is a better candidate with a lower GDOP value, the received gradient will be modified and re-forwarded (see listing 4.2.3 on line iii).

As the election region is forming, the propagating gradient is allowed to overwrite existing gradients, of the same ID, based on the default behaviour of preferring lower hops counts (line iv). However, if one of the gradients is poisoned, the poisoned gradients must be preserved and overwrite any non-poisoned gradients (line iii).

4.2.3 Election Stage 3: Completing Election

Eventually, the election gradient with the lowest GDOP value will have propagated to its fullest extent, covering the entire election region. Each node that entered the competition will review their situation after a timeout delay that allows for all the contending gradients to complete. If a node still has its own $ELECTION$ gradient, it has won the competition to become a new anchor node and emits a new anchor gradient with its estimated location. The remaining contender gradient is stopped.

Algorithm 4.2.6: HANDLETIMEOUT($ELECTION\_COMPLETE$)

```
for each $eg \in$ GETGRADIENTS($ELECTION$)
  do if $eg.winningID = myState.nodeID$
       then STOPGRADIENT($eg$)
            SCHEDULETIMEOUT($ADVERTISE\_NEW\_ANCHOR$)
```

The completion of the election in listing 4.2.6 involves just stopping the gradient to remove the election region, and scheduling a future timeout event to advertise the new anchor. The timeout is scheduled to occur after the gradient is completely stopped and removed, so when the new anchor gradient is propagated, new elections are not prevented from starting due to a continued presence of the used election region.

The introduction of new anchor nodes continues in this fashion until the entire connected network is covered in anchors and no nodes remain that are within acceptable limits for becoming new anchors.

4.2.3.1 Timing of Election Stages

Some care is taken over the scheduling of the timeout events for the different election stages in this algorithm. The time between the start of the first formation stage and the second competing stage takes into account the possibility of a worst case in overlapping election regions. This worst case involves a forming region poisoning another forming region at their extremities. Just before the poisoned gradient makes its way back to the source, the source node begins the second stage of the election. Now that the second stage has begun, the election region is impervious to the poison and continues the second stage through to its completion. As the election gradient in the second stage reaches the region's extreme edge overlapping with the first region, the first region's gradient is poisoned. The poisoned gradient of the first region then travels back to the first gradient's source node.

The time required between the first and second stages of the election must therefore allow for at least:

1. The first region to form.
2. The time for the poison to travel back to the second region's source node.
3. The time for the second region's newly competing election gradient to propagate to the edge of the region.
4. The time for the first election gradient's poison to travel back to its source node.

Each of these four events take time proportional to the radius of the election region, so the total time between the first and second stages can be calculated to be $4 * ElectionRadius * MaxMsgDelay$.

4.3 Summary

It has been shown that great hop-based ranging error is introduced from obstructions and concave perimeters in the network. This error significantly distorts the network shape of any anchor-less localisation algorithm which relies on hop-based ranging.

Hop-based ranging is dependent upon the shortest path, between two nodes, being relatively straight. Obstructions and network perimeters between the two nodes cause the shortest path to bend, as it must travel around the disconnected area. Other localisation algorithms using hop-based ranging have only been able to detect and compensate for this error through using special pre-configured anchor nodes. When no special anchor nodes are used, existing localisation algorithms produce a distorted coordinate system in concave networks.

This chapter has described a simple modification to the gradient primitive. As normal, gradients keep track of the number of nodes they have passed along the shortest path, but additionally keep track of how many of those forwarding nodes were flagged as being beside a network edge, or perimeter. As these modified gradients are forwarded past flagged edge nodes, they are then known to have unreliable hop counts for use as ranging measurements. It can be thought that these modified gradients automatically form two regions: one region containing nodes receiving good ranging measurements, where no network edges have been passed since the source node; and another region where all nodes receiving the gradient see that it has passed one or more edge nodes. The second type of region is akin to shadow of doubt cast over an entire area beyond a network edge (see figure 4.5). These received gradients with non-zero edge node counts are named shadowed gradients.

To our knowledge, no workaround to the missing clear line-of-sight in anchor-less hop-based localisation has been published before. This chapter has proposed a new localisation algorithm with just such a workaround. The new algorithm spreads a coordinate system throughout the network, from a single starting point, by electing new anchors along the way. The election of new anchors is guided by nodes' calculated geometric dilution of precision (GDOP) – a measure of the potential error in location estimation. Newly elected anchors emit gradients with their estimated locations attached. All nodes receiving anchor gradients will estimate their location, and its GDOP, from the node's entire list of unshadowed anchor gradients. That is, each node will filter out all shadowed gradients from multilateration, leading to a location estimate free from the distortion caused by any network perimeters or obstacles.

By using the new proposed algorithms from this chapter and chapter 3, together with the idea of shadowed gradients, a coordinate system can be formed without becoming distorted by gaps or corners in the network. This is particularly important in our scenario as many ceilings have corners, bends and non-convex boundaries. Avoiding this distortion allows a more accurate and representative coordinate system to be used to form a visual spatial positioning code – a subject covered in the following chapter.

Chapter 5

Simulation

This chapter primarily evaluates, via simulation, the edge detection and coordinate propagation algorithms described in chapters 3 and 4. The Anoto spatial pattern described in chapter 6 is then shown overlaid on the resulting simulated coordinate system, followed by a brief discussion of its properties and their implications in the context of an indoor positioning system.

Before examining the results, the simulator is briefly introduced and the reasoning behind the various network scenarios used in the simulations is explained.

5.1 Simulator and Network Scenarios

The simulator used is written in Java and builds upon a discrete event simulator called SSF, the Scalable Simulation Framework (Cowie, Liu, Liu, Nicol & Ogielski 1999, Cowie, Nicol & Ogielski 1999). This was originally chosen over MIT's Scheme based HLSIM (also known as Gunk), as it offered the potential to run very large parallel simulations on multiple computers and provided some support for network based models.

The only two entities involved in the simulations are the actual nodes themselves and a single coordinating entity that maps communications between the nodes according to the network topology. A node entity reacts to only two types of events: reception of a communication packet from a neighbouring node and notification of a previously scheduled timeout occurring.

The coordinating entity calculates a network topology from the locations of the nodes and the node communications range. Any nodes that have locations within communications range of each other become neighbours capable of bidirectional communications. When a node sends a communications event, it is handled by the intermediary coordinating entity which forwards it on to all the neighbouring nodes.

A few important simplifying assumptions are made in the simulations:

- The simulations involve nodes on a flat surface in two dimensions. See chapter 7 for a discussion on extending the algorithms to three dimensions.
- Events are dealt with in their entirety the instant they are received. The computation time is not accounted for.
- The communication medium imitates that of an omni-directional broadcast radio with an idealised circular attenuation model. The communications range remains constant for all nodes for the duration of the scenario and it remains constant regardless of the angle a neighbouring node lies relative to a broadcasting node.
- It is assumed that there is no message collision and all broadcast packets are received by all neighbours. The means of inter-process communication provided by the HomePages (see section 2.5.2), which is used by the algorithms, is tolerant of occasional packet loss due to its proactive periodically updating nature.
- The nodes are immobile and are always alive. See chapter 7 for a discussion on modifying the algorithms to allow for changes in topology and node death.

Most of the simulation scenarios shown in this chapter involve nodes placed randomly, with a uniform probability distribution, within an area of 100 by 100. The communications radius, R is set at 2.185097 units which leads to a width and height of up to 45 hops in a square network. The network density, measured by the average neighbourhood size, N, is varied by changing the number of nodes involved in the simulation from 5000 ($N = 7.5$) to 20000 ($N = 30$). By changing N rather than R to vary the network density it becomes easier to compare the results because the network width in terms of hops remains similar[1]. $N = 15$ is chosen as a middle point of reference as the network connectivity becomes sufficiently high for reasonable distance estimates based on hop counts (Bachrach et al. 2004).

The scenarios are split into convex and concave layouts. The convex layouts involve nodes' locations constrained to within a full square shaped region, whilst a variety of regions are used for the concave layouts. The same random number seed is used when generating the locations of the nodes in both types of scenario so they both contain the same network topology. The only difference between the two scenarios of the same seed and value of N is that the concave version omits the nodes that lie within its prohibited void area. As the topology remains identical for the same value of N, any differences in behaviour of the algorithms on the two scenarios can be attributed to the edges and void area introduced in the concave scenario.

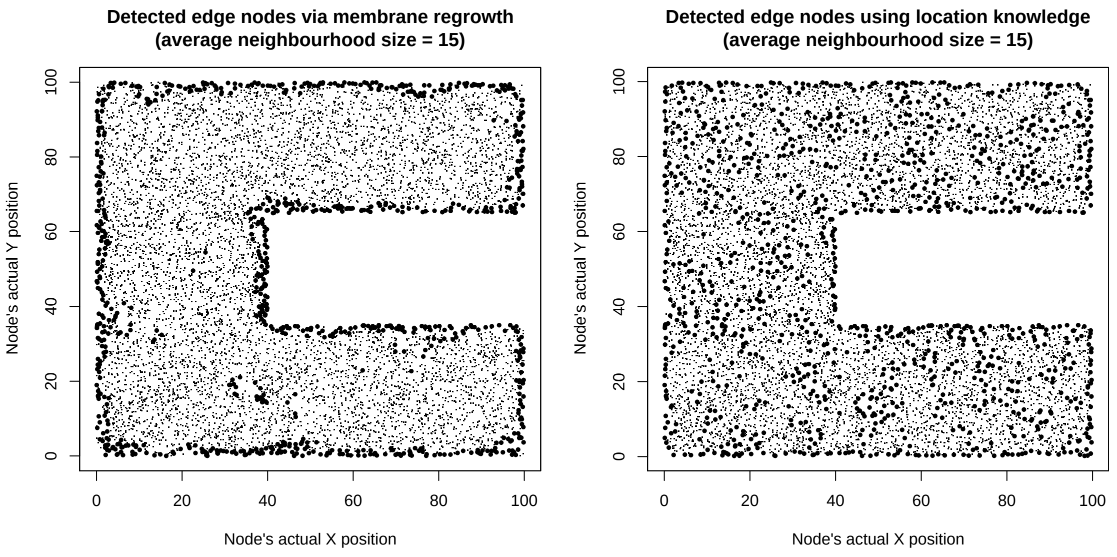
[1]In practice, increases in the network density by changing N causes the average hop distance to increase and consequently causes the network width in hops to decrease slightly. If the density were increased by changing R, the network width in hops would decrease drastically and cause problems when comparing the spatial phenomena displayed by the algorithms.

Where the simulation seed remains the same and N is increased by adding more nodes, the interconnect topology of existing nodes remains similar whilst the network becomes more dense with new nodes – making it easier to see the influence of network density on the algorithm's performance without the distracting effect of vastly different topologies.

5.2 Edge Detection Simulations

This section explores the performance of the proposed new edge detection algorithm, based on membrane regrowth, from section 3.2. The existing published algorithm, based on node location knowledge (detailed in section 3.1.1), of Martincic & Schwiebert (2006) is used here as a reference with which to compare the new proposed algorithm.

As a typical example of the general distinguishing characteristics of the two algorithms, figure 5.1 shows simulations of both algorithms on the same convex scenario, with N at 15.

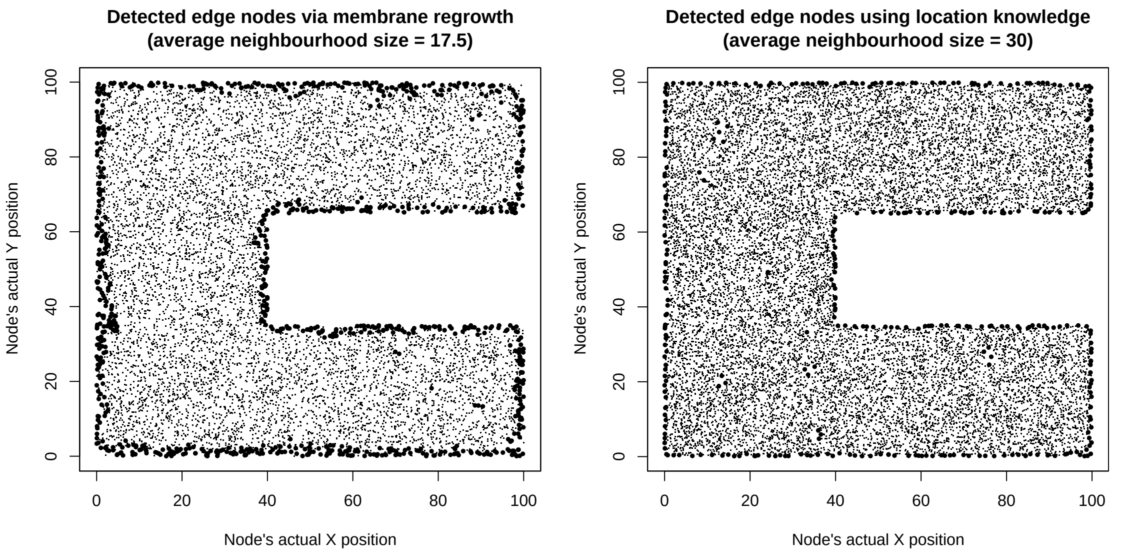


(a) Membrane regrowth. Cell radius is 7, membrane width is 3. (b) Using Martincic and Schwiebert's approach

Figure 5.1: Both edge detection algorithms running on 8176 nodes in a concave network layout with $N = 15$. Non-edge nodes are marked as a small dot. Detected edge nodes are indicated by a larger circle.

The first most striking difference that can be seen between the two algorithms is the sensitivity to small local disconnected areas in the network. The membrane regrowth algorithm (figure 5.1(a)) shows several clumps of nodes within the network detected as edges where the local area contains only very little or no nodes. In contrast, Martincic and Schwiebert's algorithm (figure 5.1(b)) is sensitive to small local changes in network density and edge nodes can be seen almost evenly spread throughout the network space.

The second most noticeable difference is in the width of the detected edge. Martincic and Schwiebert's algorithm tends to detect edge nodes at the very farthest point of the network's perimeter, consistently creating a very thin edge. The edge is so thin that a path along edge nodes can not be made without visiting nearby intermediate non-edge nodes. If such a path were important for an application, a process of thickening the edge could be performed by causing non-edge nodes to become edge nodes themselves if they have sufficient neighbours as edge nodes. The detected edge nodes found by the membrane regrowth algorithm shows a noticeably wider edge area, however it has a relatively high variance in this width.



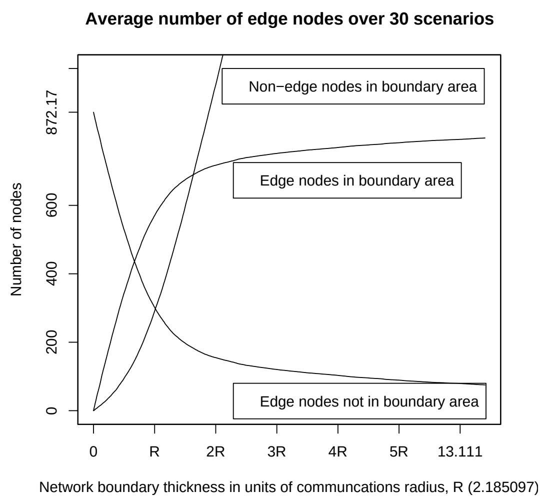
(a) Membrane regrowth with $N = 17.5$ (9543 nodes). Cell radius is 7, membrane width is 3.

(b) Martincic and Schwiebert at $N = 30$ (16339 nodes). Perfect location knowledge used.

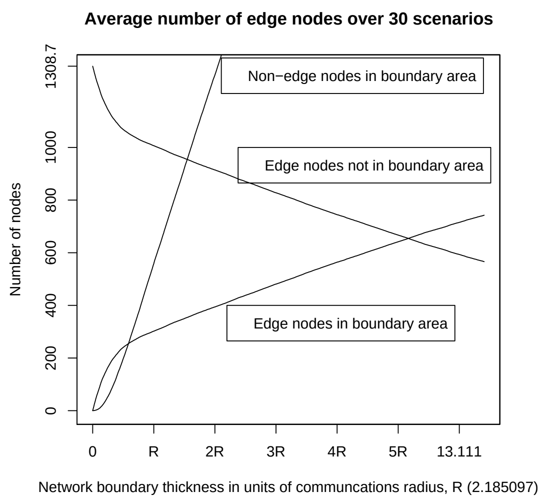
Figure 5.2: Both algorithms at network densities that provide results with few artifacts and low noise

The sensitivity of Martincic and Schwiebert's approach with N at 15, where edge nodes are detected throughout the network, can be considered as involving a high level of noise or false positives. At this density these detected edge nodes are not next to the network perimeter that we are interested in. Similarly, the membrane regrowth approach shows some artifacts detected within the network space which may not be wanted. Figure 5.2 shows the two algorithms running on a network with a density at which they perform with very little noise or artifacts. Martincic and Schwiebert's algorithm requires a high network density (around $N = 30$) to reduce most of the noise. The membrane regrowth algorithm can operate at lower densities (around $N = 17.5$) with little noise or artifacts and as will be seen later, it can also operate at lower densities ($N < 15$) with some changes to the cell radius and membrane width. A few undetected edge nodes, or false negatives, can be seen in the membrane regrowth approach, mainly in the extreme ends of the concave corners of the network.

In an effort to see the relative thickness of the edges detected, figure 5.3 shows the numbers of detected edges in convex scenarios that lie within an inner margin area beside the network perimeter. The boundary thickness in this figure is the width of the inner margin region used to count nodes. As the boundary thickness increases from 0, the number of edge nodes lying in the boundary area sharply rises at first because a high percentage of the nodes in the area are detected as edges. As the thickness increases further, the number of detected edge nodes in the area rises less sharply and an increasing number of non-edge nodes are found in the boundary. This indicates that in some parts of the boundary area, the detected edge lies within the boundary area in its entirety, whilst some other parts of the boundary still only capture part of the detected edge. Eventually the entire detected edge that we are interested lies within the boundary area, at which point any rise in edge node numbers is due to artifacts and noise.



(a) Membrane regrowth. Cell radius is 7, membrane width is 3.



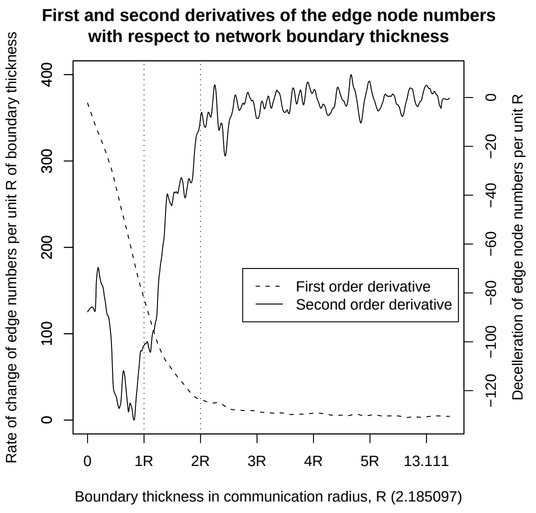
(b) Martincic and Schwiebert with perfect location knowledge

Figure 5.3: Number of detected edge nodes and the detected edge width from both algorithms. These numbers are the average from 30 convex scenarios with different seeds. Each scenario contained 10000 nodes with $N = 15$.

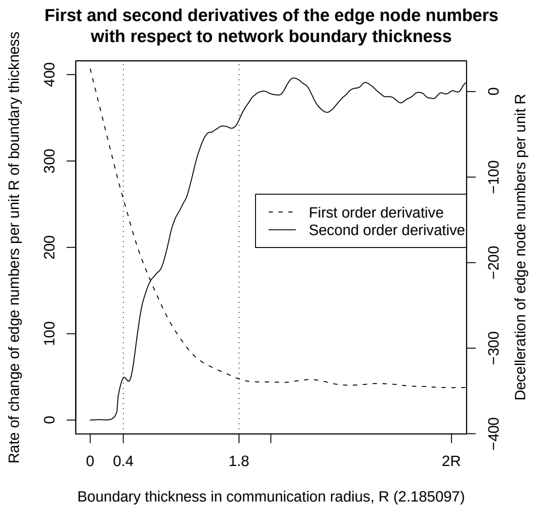
With N at 15, the membrane growth algorithm (figure 5.3(a)) shows that typically more than 870 edge nodes are detected throughout the network, most of which occur very close to the network perimeter and likely part of the edge we are interested in. Martincic and Schwiebert's algorithm (figure 5.3(b)) detects far more edge nodes at around 1300, most of which are not close to the network perimeter.

From the initial steep rise of edge nodes in Martincic and Schwiebert's algorithm, it appears that most of the edge that we are interested in is detected within a narrow boundary area of thickness $\frac{1}{2}R$. The membrane regrowth approach sustains this steep rise up to a boundary thickness of around $2R$, showing a relatively thick detected edge.

The rate at which the detected edge nodes in figure 5.3 rise is shown in figure 5.4. The first order derivative shown here is a regression fit, using LOWESS (Cleveland 1979), of the differences in the edge node numbers. The second order derivative is the non-smoothed differences of the first derivative.



(a) Membrane regrowth. Cell radius is 7, membrane width is 3.



(b) Martincic and Schwiebert with perfect location knowledge

Figure 5.4: First and second order derivatives of the detected edge node numbers (in figure 5.3) with respect to the boundary area thickness

In figure 5.4(a), the deceleration in edge node numbers with respect to the boundary thickness stays roughly around zero at thicknesses above $2R$. This zero deceleration implies that, on average and with a boundary of $2R$, the rest of the non-boundary network area is covered evenly with detected edge nodes that we are not interested in. The deceleration also sharply falls after roughly $1R$ until it reaches zero at $2R$, which implies that there is a variation in the detected edge thickness from roughly $1R$ to $2R$. At $2R$ and wider, the first order derivative stays around 10 edges per R – giving an indication of the number of false positives or artifacts present.

Figure 5.4(b) shows the thinner edge of Martincic and Schwiebert's algorithm which appears to have an edge less than $0.82R$ on average and varies between $0.18R$ to $0.82R$. At $0.82R$ and wider, the first order derivative stays around 40 edges per R.

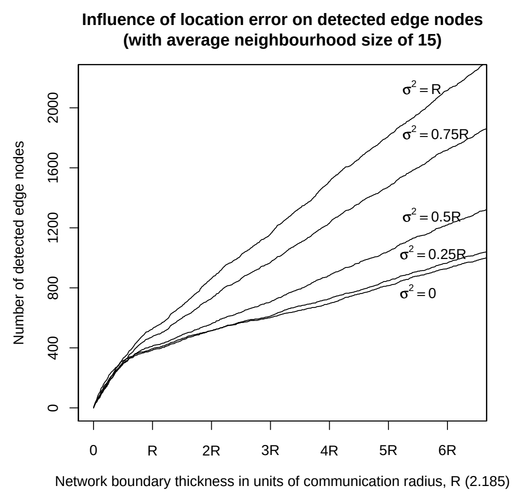
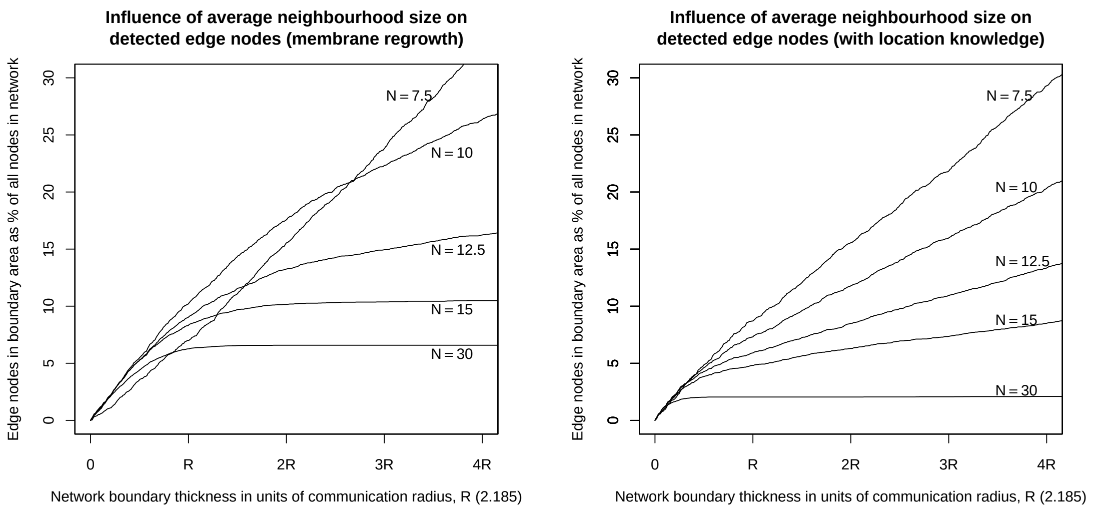


Figure 5.5: Martincic and Schwiebert's algorithm running with different degrees of node location error. Convex layout of 10000 nodes, $N = 15$

The results above show the use of Martincic and Schwiebert's algorithm with perfect knowledge of the nearby neighbouring node's locations, but most real life network deployments will fall short of this ideal situation. Figure 5.5 shows the algorithm running with varying amounts of node location error. There is little impact from small variations, such as $0.25R$ in node's location error. As the node location error increases, more noise or false positives are encountered as would be expected, but the edge itself is still perceptible. The membrane regrowth approach does not rely on knowledge of any node's location so is unaffected by this.



(a) Membrane regrowth. Cell radius is 7, membrane width is 3.

(b) Martincic and Schwiebert with perfect location knowledge

Figure 5.6: Different average neighbourhood sizes and their influence on both algorithms' edge detection. Concave layout, same simulation seed with varying node numbers.

The effects of different network densities is shown in figure 5.6. At low densities where N is 7.5 and below, both algorithms perform poorly and show no distinction between edge and noise. With N at 10, some improvement appears but still contains unusably high levels of noise in both algorithms. Where N is at 15, the membrane regrowth algorithm (figure 5.6(a)) contains a well defined edge and few artifacts or noise. At the same density, Martincic and Schwiebert's algorithm (figure 5.6(b)) still contains noise that would interfere with some applications such as the propagation of the coordinate system. With higher densities of N at 30, both algorithm perform very well with next to no artifacts or noise. At these higher densities, the two algorithms tend to provide a slightly thinner edge.

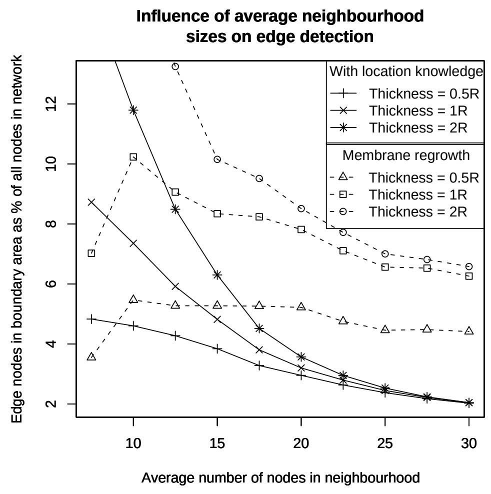
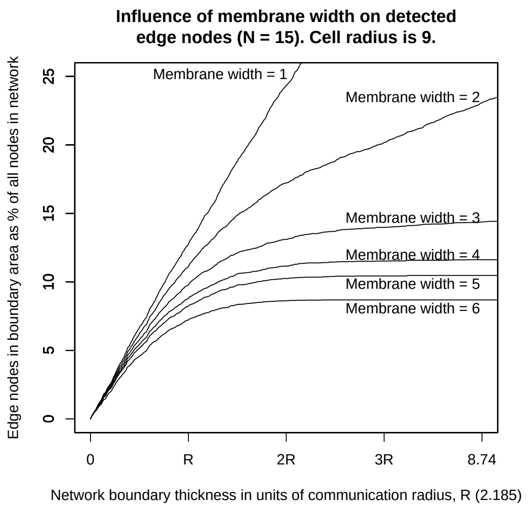


Figure 5.7: Both algorithms running with different average neighbourhood sizes. Membrane regrowth uses cell radius of 7 and membrane width of 3. Perfect location knowledge is used for Martincic and Schwiebert's algorithm.

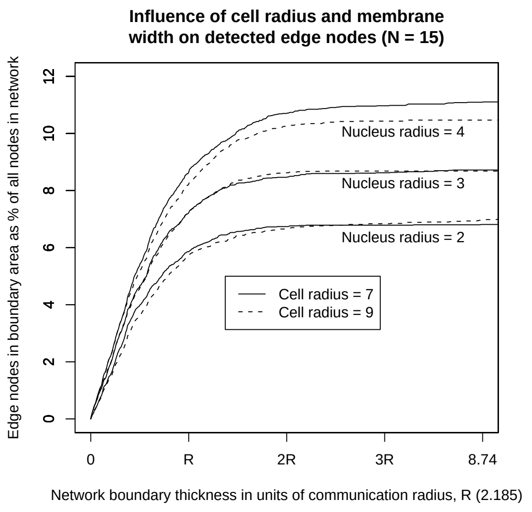
Figure 5.7 shows both algorithms together with different network densities. As N increases, both algorithms perform better. Also, it can be seen that generally the thickness of the detected edge shrinks as the neighbourhood density increases.

At low densities, both algorithms tend not to show a discernable edge as there is such a high number of artifacts. As the average neighbourhood increases to 25-30, a very clear and defined edge is produced by Martincic and Schwiebert's algorithm, as can be seen by the convergence of the percentages of nodes in different sized boundaries. The membrane regrowth algorithm does not converge in such a striking way, and instead continues to produce edges that vary slightly in their thickness at very high network densities. Again, in this figure, it can be that the membrane regrowth algorithm produces very thick edges compared to Martincic and Schwiebert's algorithm where very thin edges.

In figure 5.7, the membrane regrowth algorithm critically fails at very low densities where N is less than 10 because most of the cells' membranes are fragmented into two or more separate sections (and so dismissed). The membrane regrowth approach also operates poorly with N at 10 and 12.5 because the membrane width of 3 is too narrow to find a path within it due to the reduced network connectivity. This causes a somewhat premature regrowth of the membrane and too many edges to be detected. This can be alleviated through a different choice of membrane width and cell size as will be seen below.



(a) Different membrane widths



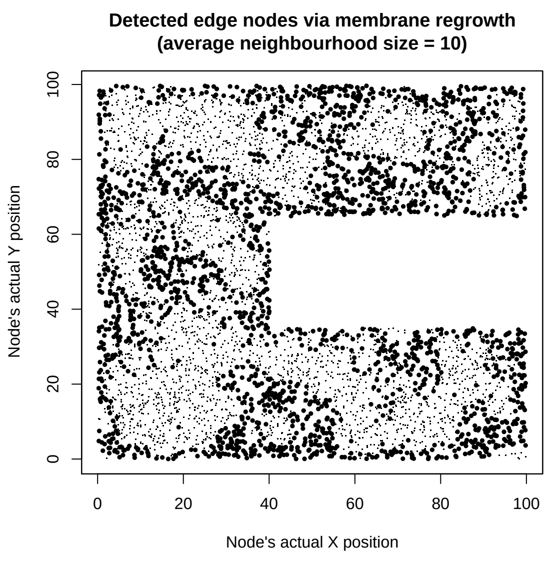
(b) Different cell radius and membrane widths

Figure 5.8: Shows effects of different cell radius and membrane widths on detected edge node numbers. The locations of cell leaders are kept the same.

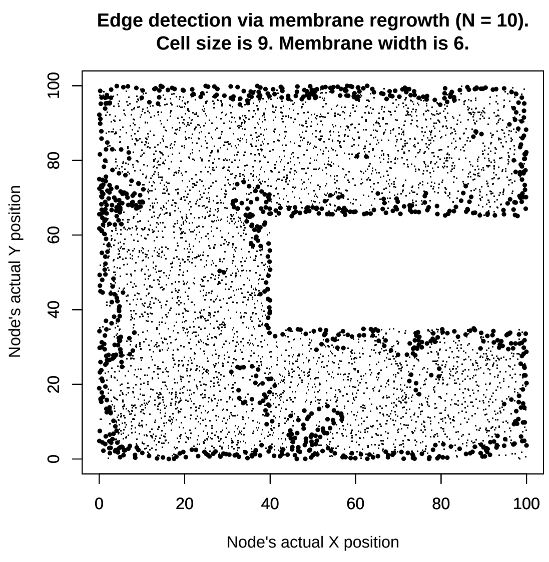
The effects of varying the width of the membrane whilst keeping the cell radius constant is displayed in figure 5.8(a). Widths of 1 or 2 are clearly inappropriate and create problems finding an encircling path along the membrane of a cell that is otherwise unobstructed by any network perimeter. This causes many artifacts and noise similar that seen in figure 5.9(a). With a width of 3 and above, the encircling paths can normally be found as the network is sufficiently dense with N at 15. With higher membrane widths, slightly fewer nodes near the network perimeter are detected as edges because the thicker membrane is no longer broken.

Figure 5.8(b) shows that it is actually the cell's radius minus the membrane width (i.e. the 'nucleus' radius) that governs most of the difference in detected edge nodes. The cell radius and membrane width can both be different, but if the 'nucleus' radius is the same, then very similar results are obtained.

By changing the cell radius and membrane width, it is possible to achieve more useable results from the edge detection at lower network densities. For example, figure 5.9(b) shows some convincing edges obtained with a low average neighbourhood size of 10 when the membrane width is substantially increased (and the cell radius suitably enlarged).



(a) Cell radius of 7, membrane width of 3



(b) Cell radius of 9, membrane width of 6

Figure 5.9: Obtaining better results for lower average neighbourhood sizes by using different cell radius and membrane widths

It should be noted that as the cell radius becomes larger, there is a greater potential for problems in very narrow network spaces. If a space is too narrow, cell membranes will become perforated by the network perimeter on both sides, causing two membrane fragments and the cell to be discarded. Martincic and Schwiebert's algorithm does not suffer from this problem.

So far, the results shown have assumed that the nodes are placed at random with a uniform probability density on the 2D surface. With this layout, the probability that any area of the surface contains exactly N nodes follows the Poisson distribution. This may well not be the case in our scenario, as nodes might be suspended within paint or some other medium that could cause the nodes to separate from each other resulting in a minimum distance between each node.

The method of separating node particles within a medium, such as paint, largely depends on the size and form of the particles. Nevertheless, a few potential approaches are suggested here:

- It could be possible to coat the particles in a surfactant before mixing them in the paint.
- A dispersant, or plasticising agent, could be added to the paint to aid in the separation of the node particles.
- Vibration could be applied to disperse the particles, perhaps through the use of acoustics or electromagnetic effects.
- Some other electrostatic or electrokinetic effects could be used to make the particles repel each other.

- If the medium were wallpaper, or some other pre-manufactured surface, it is straightforward to introduce a controlled dispersion of particles on the medium. For example, the manufacturing process of sandpaper has, for a long time, allowed a controlled spacing of the grains in open coat sandpapers.

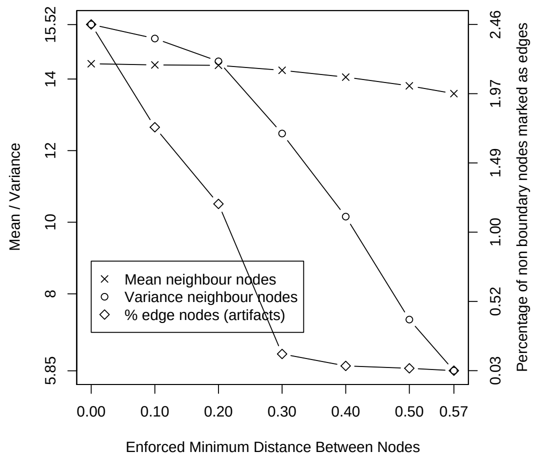


Figure 5.10: Shows the number of edge node artifacts and the variance in node neighbourhoods as the minimum distance between nodes is increased. N=15, 8176 nodes, concave layout. Boundary thickness assumed to be 2R

Simulations were run with different enforced minimum distances between nodes. The layouts for these simulations were created by randomly adding a node to the layout and checking the newly added node's distance to each of its neighbours. If the new node has any neighbour closer than the minimum distance, all of the nodes in that neighbourhood are removed and the process of adding each node with new random locations is resumed. This continues until the whole network space contains the required number of nodes.

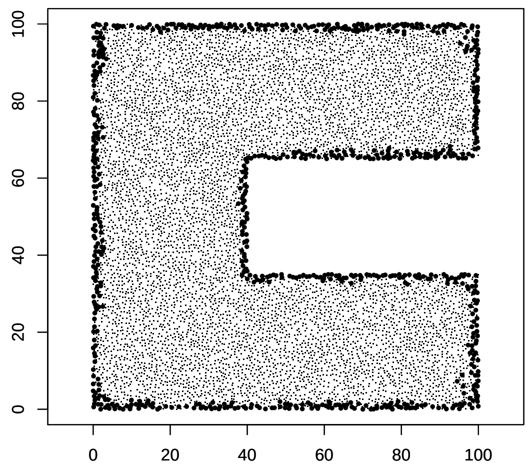


Figure 5.11: 8176 nodes with $N = 15$ and an enforced minimum distance of 0.57

Figure 5.10 shows the effect of enforcing different minimum distances between nodes. 8176 nodes are placed according to the concave 'C' shaped layout with

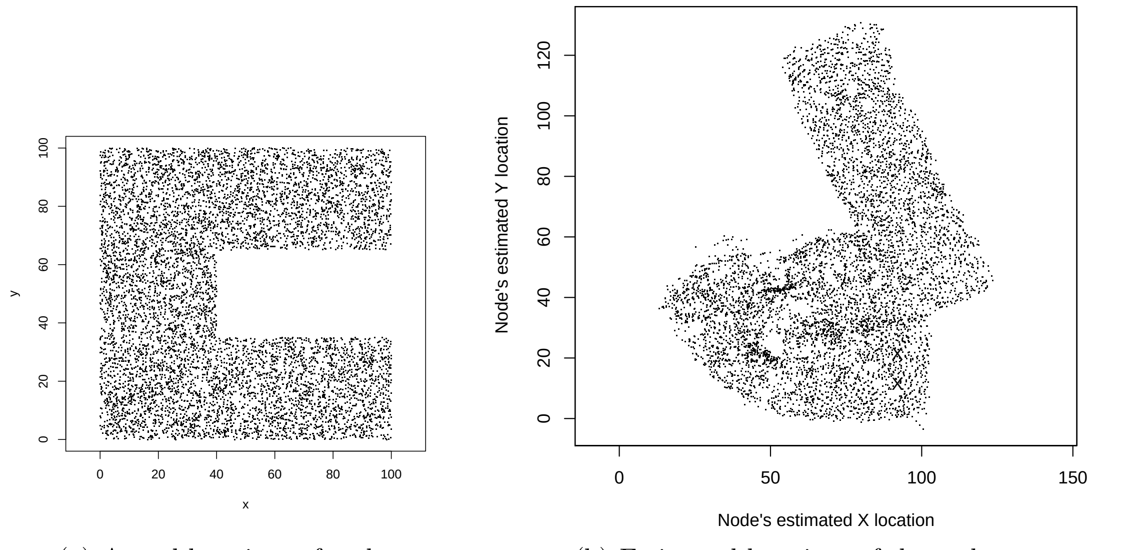
$N = 15$. As the enforced minimum distance is increased, the nodes become spread more evenly and the variance in the numbers of neighbouring nodes is decreased. This reduced variance in neighbourhood sizes has an effect on the edges detected by membrane regrowth. As the variance decreases, the membrane regrowth algorithm performs better and the number of edge nodes detected outside of the boundary area is greatly reduced.

Figure 5.11 shows the edges detected when the minimum distance between nodes is quite large (and the variation in neighbourhood sizes is very low). The distribution of nodes is seen to be more regular, and no artifacts can be seen.

5.3 Coordinate Propagation Simulations

This section looks to the performance and influencing effects on the coordinate propagation algorithm detailed in section 4.2.

By electing anchors by random, rather than using the GDOP metric to elect new anchors, there is a potential for large failures in the propagation of the coordinate system. If badly located anchors are elected, the coordinate system may fold in on itself, bend drastically, or the optimisation algorithm may even fail to converge. An example is in figure 5.12.

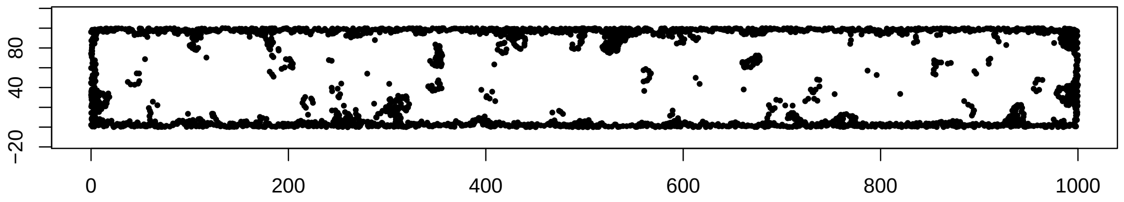


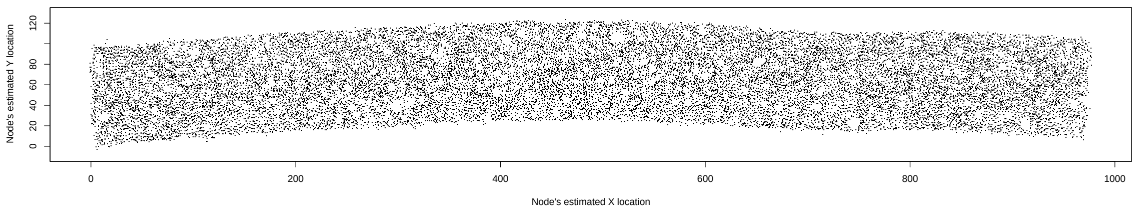
(a) Actual locations of nodes. (b) Estimated locations of the nodes.

Figure 5.12: Potential catastrophic propagation failure when not using the GDOP metric for electing new nodes.

Other than this tendency for catastrophic collapse, there is little qualitative difference between using the GDOP metric and not using it during propagation. The metric serves primarily as an important guard against the degenerate case where anchors are nearly collinear as opposed to significantly improving the estimation accuracy.

In this section, the election region for new anchors in the simulations is 8 hops radius; however, the election region has no real effect on the resulting coordinate system beyond 4 hops radius. A radius of 8 hops was chosen to give reasonably large election regions and choices for new anchors whilst keeping the number of simultaneous nearby elections low (as only one election is allowed in one area).


(a) Actual locations of detected edges via membrane regrowth (cell radius 7, membrane width 3)


(b) Propagated estimated coordinates with detected edges

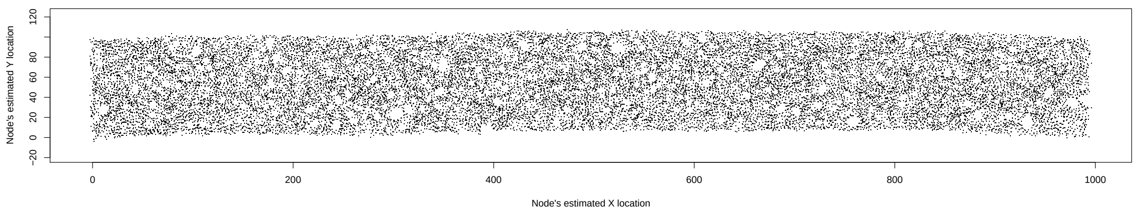
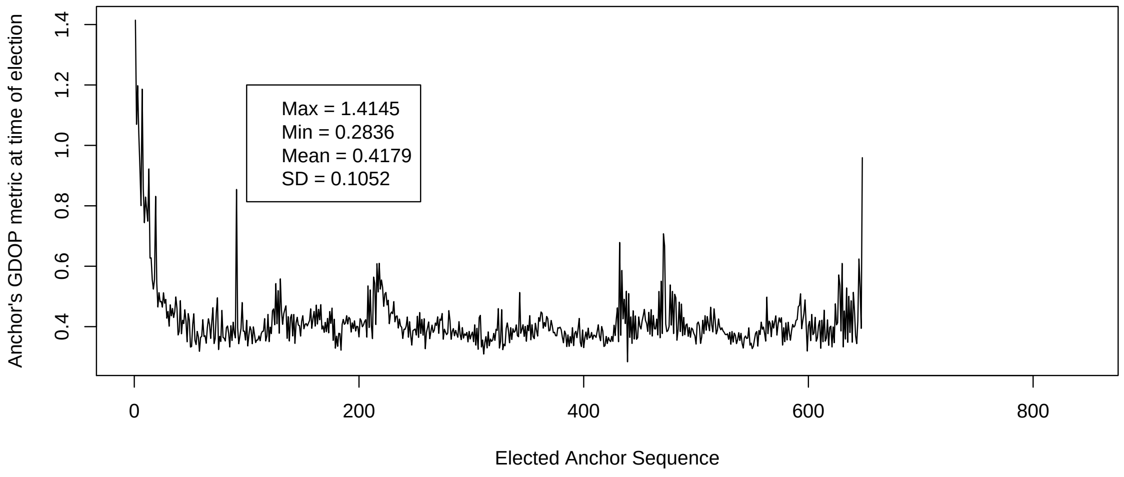

(c) Propagated estimated coordinates without detected edges

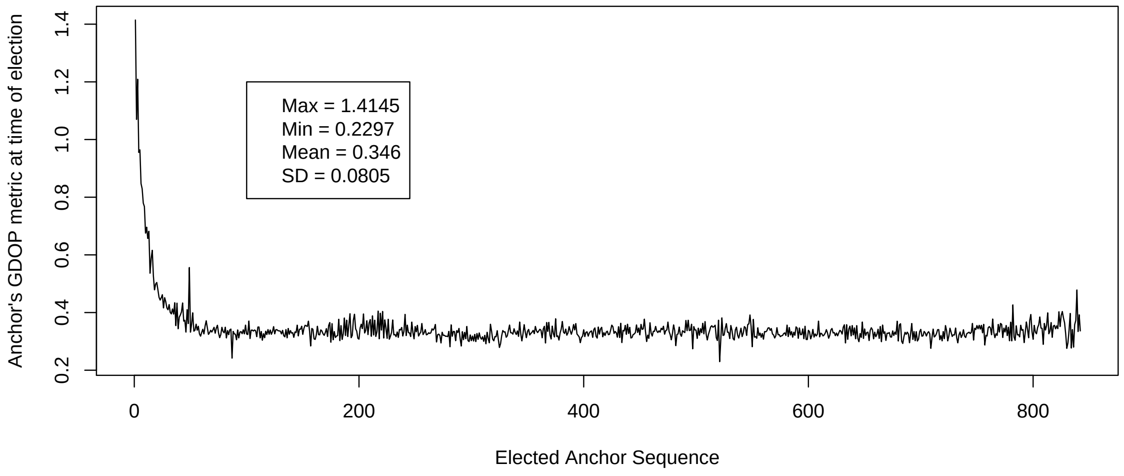
Figure 5.13: Coordinate system formed along long narrow convex corridor. 30000 nodes ($N = 15$, $R = 3.989423$)

The propagation of coordinates over a greater distance is shown in 5.13. The scenario here is a straight corridor ten times wider than it is in height (approximately 25 hops in height and 250 hops in width) and the initial two anchors begin at the extreme left of the corridor. In both cases of using the edge information and disregarding all edges, a useable coordinate system is gained and does not show any severe skew over the distance travelled.

Here it is seen that filtering out all shadowed anchor gradients (in figure 5.13(b)) from use in multilateration actually results in a slightly worse coordinate system than if the edge influenced anchor gradients were not filtered out (as in figure 5.13(c)). From looking at the detected edges in figure 5.13(a), there are a considerable number of small disconnected areas detected across the entire corridor. Each of these small obstacles has caused several shadowed anchor gradients to be filtered out from their use in multilateration. From this observation, it can be deduced that the error resulting from using range estimates influenced by such such small obstacles is less than the error resulting from using fewer anchors in multilateration.



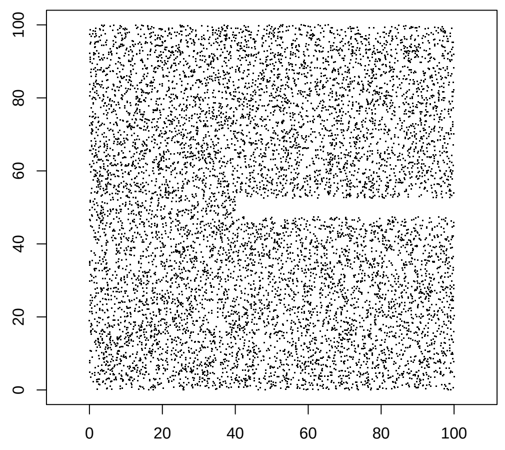
(a) With detected edge information and shadowed anchor gradients filtered out



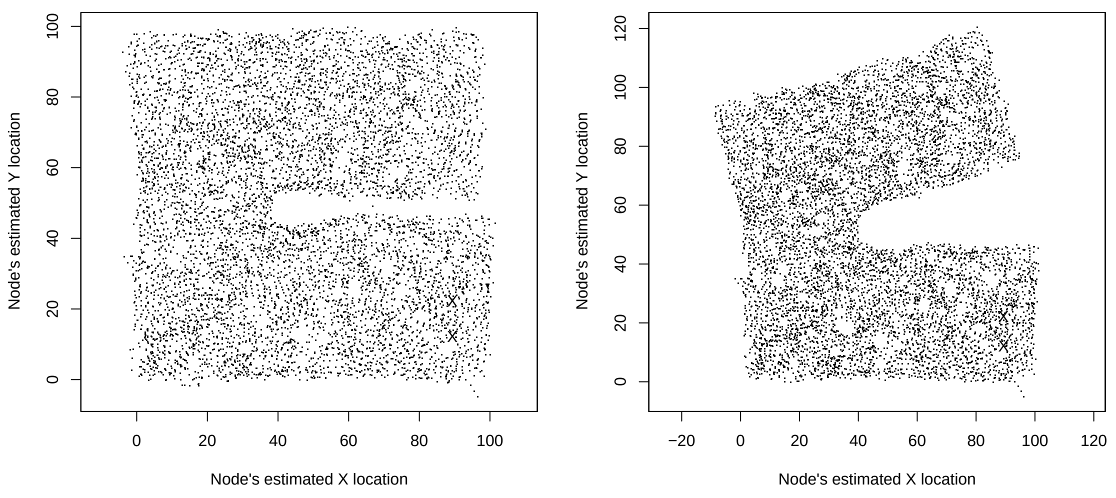
(b) Without detected edges and no anchor gradient filtering

Figure 5.14: GDOP metric of new elected anchors, in the order they were elected, in the narrow corridor scenario of figure 5.13

There are not only fewer anchors used in multilateration, but fewer anchors overall when the edge information is used. The number of anchors elected in figure 5.13(b) is 648 whilst in figure 5.13(c) 842 anchors were elected. This is because anchors will not be elected on or beside any known edge nodes. Less anchors involved in multilateration also leads to lower GDOP metric values as can be seen in figure 5.14. In these figures, the GDOP metrics for new anchors begin at a very high level, as only the initial anchors are visible; however, as more anchors are elected, the average GDOP metric for new anchors begins to reduce and stabilise. When filtering out shadowed gradients, the GDOP metrics of elected anchors is of a poorer quality on average than if the gradients were not filtered. The quality of the newly elected anchors in figure 5.13(b) varies to a much greater degree than in figure 5.13(b). At points where many edge obstructions cause many anchors gradients to be filtered out, the GDOP metric increases significantly.



(a) Actual node locations



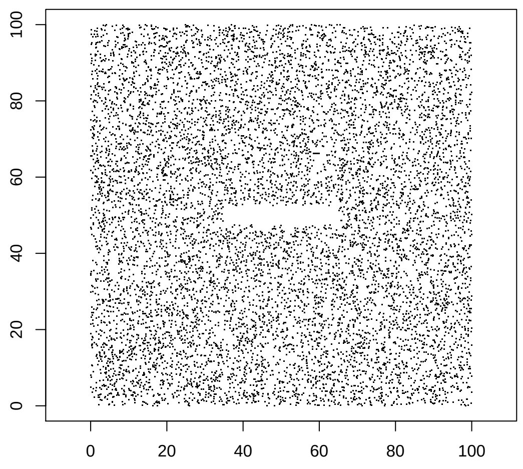
(b) Propagated coordinates with detected edges via membrane regrowth (cell radius 7, membrane width 3)

(c) Propagated coordinates without detected edges

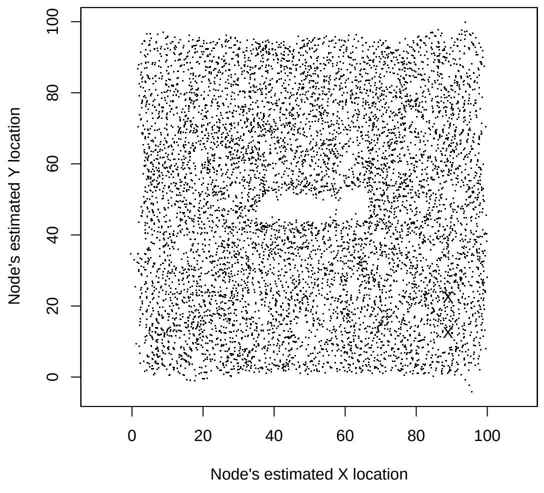
Figure 5.15: Coordinate systems formed with and without detected edges. 9694 nodes, thin concave layout ($N = 15$). Anchor gradients forwarded for 20 hops.

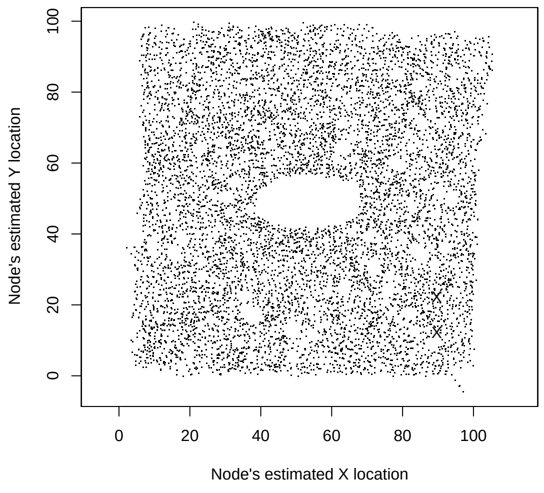
Figure 5.15 shows the effect of filtering shadowed anchor gradients in a concave network. In figure 5.15(c), when no filtering is performed, the resulting estimated coordinate system becomes significantly bent by the edge's influence. This bending is more pronounced when the anchor gradients are forwarded over more than 20 hops. If the network involved several edges such as this, the estimated coordinate system would no longer resemble the actual shape of the network.

With the filtering of shadowed gradients in figure 5.15(b), the edge influence is eliminated and the coordinate system retains its resemblance to the actual network shape.



(a) Actual node locations





(b) Propagated coordinates with detected edges via membrane regrowth (cell radius 7, membrane width 3)

(c) Propagated coordinates without detected edges

Figure 5.16: Large thin rectangular obstacle in an otherwise convex layout. 9844 nodes, ($N = 15$).

New anchors are elected and added to the coordinate system simultaneously. It can be the case that the coordinate system propagates in several different directions at the same time. Figure 5.16 shows a coordinate system that propagated along two different directions and later rejoined and merged with itself on the far side of an obstacle. Without filtering shadowed gradients as in figure 5.16(c), the shape of the obstacle is misrepresented by the estimated coordinate system, but the small error does not stop the estimated coordinate system from joining together.

5.4 Summary

This chapter has looked at the results of simulating the proposed edge detection and localisation algorithms in a discrete event based simulator.

Generally, where nodes were deployed in the network scenarios, their locations were chosen using a uniform random distribution. This leads to the numbers of nodes in independent neighbourhoods being distributed according to a Poisson distribution. The Poisson distribution of node numbers in the neighbourhoods is a fairly unforgiving layout, but a good test of the algorithms under difficult network deployments. The mean average numbers of nodes in neighbourhoods were varied to show the effect of the network density on the algorithms.

The proposed edge detection algorithm was investigated whilst using Martincic & Schwiebert's (2006) algorithm as a reference. Two main properties of the algorithms became apparent and were explored: the width of the detected network edge, in numbers of nodes; and the number of unwanted nodes flagged as edges (artifacts, or false positives) in areas which were not thought to be close to the real edge of the network.

The proposed algorithm exhibits far fewer artifacts than Martincic & Schwiebert's (2006) algorithm and operates well at much lower network densities. The width of the proposed algorithm's detected edge is much thicker than that detected by Martincic & Schwiebert's (2006) algorithm. For the purposes of shadowed gradients, very thin edges can prove problematic as some gradients can pass by the network edges without being shadowed, if there are too few nodes flagged as edges.

It was shown that the some of the parameters of the proposed edge detection algorithm, such as the membrane width and cell size, can have an influence on the numbers of artifacts. In networks with a very low density, increasing the membrane width and cell size can markedly improve edge detection, though at a cost of further communications.

Different distributions of numbers of neighbourhood nodes were also explored, in order to show the effect that the variance in neighbourhood numbers has on the algorithms. As the neighbourhood node numbers move from a Poisson distribution, (with high variance) towards a uniform distribution (with low variance), the edge detection algorithm performs better. Where the variance is low, no artifacts occur.

The GDOP metric is used by the proposed localisation algorithm when deciding between new anchors to elect. It is shown that if new anchors are elected at random, rather than using this guiding GDOP metric, there is a great risk of electing bad anchors. Bad anchors can cause the propagating coordinates system to 'fold' in on itself, incorrectly and badly representing the network shape.

A warning is given for cases where there are high numbers of incorrectly flagged edge nodes, or nuisance artifact nodes, in very narrow areas. In such a situation, too many anchor gradients can be shadowed, resulting in lower quality location estimates in convex network layouts. This is mostly due to the simplistic nature of controlling the anchor density, through enforcing new anchors to be a minimum distance from existing anchors. If the anchor density were allowed to be increase in areas where more anchors are shadowed, this problem is likely to disappear. See section 7.1.2.2 for possible future work and remedies to this situation.

The filtering out of shadowed anchor gradients, from the multilateration, is shown to be greatly beneficial in concave network layouts. It eliminates much of the distortion introduced by the hop-based ranging measurements that are influenced by the network edges. To the best of our knowledge, no other hop-based localisation algorithm, without pre-configured anchors, has achieved this robustness to concave network layouts.

Chapter 6

Positioning Patterns

It is conceivable that a gel or paint containing sufficient numbers of small computing particles suspended within it could be painted over a ceiling of a room in a building, thus forming an amorphous computer (Butera 2002). The particles could then collaborate to agree upon a coordinate system spanning the entire surface. An encoding of each particle's coordinates can be performed, yielding a virtual spatial pattern overlaying the entire surface. This spatial pattern has the intended property that each small, arbitrarily located, portion fully encodes its exact coordinates and orientation within the coordinate system. Once the particles know which segment of the pattern they form part of, they can stop computing and discharge ink or some other chemical to permanently mark their part of the pattern upon the ceiling. Using a cheap simple camera, such as those found in an average mobile phone, an image of the pattern can be taken. This image can be quickly analysed and the glyphs decoded, providing a position and orientation context awareness in an indoor setting.

This chapter describes three existing methods of creating a two-dimensional pattern formed of simple identifying glyphs, or marks.

All of the spatial patterns that will be looked at are based on watermarking paper with location information. Each of the inventors of the patterns have envisioned a small handheld device, in the form of a digital pen, to facilitate the capturing of handwriting, and other spatially aware applications, for this augmented paper (Jared et al. 2001, Wang et al. 2005, Pettersson 2001, Pettersson 2006). Such a pen is equipped with a small camera capable of rapidly reading these patterns and determining its position on the page. Furthermore, each of these spatial patterns similarly encode the locations by carefully interleaving bit sequences folded into a known area of space. It is the method of interleaving these bit sequences, and the form of the glyphs making up the sequences that primarily distinguishes the different encodings from each other.

Firstly, Microsoft's recent M-Array (Wang et al. 2005) is shown as a simple way to provide embedded location information in spatial codes. This involves glyphs that form a maze-like pattern over the paper. The orientation of the pattern can not be determined from the code itself, but rather it is guaranteed from the disambiguating form of the glyphs themselves.

The second type of spatial codes explained are a particular form of PARC's DataGlyphs (Hecht 2001) that provides location information. DataGlyphs use small elongated diagonal slashes as the glyphs which maintain an unobtrusive visual appearance. These glyphs encode location information by means of an address carpet (Jared et al. 2001). Whilst these address carpets enclose cells containing further information, the focus here is just on the method of providing location information.

Thirdly, Anoto's pattern (Pettersson 2001, Pettersson 2006) is explained. This uses a faint dotted pattern that assumes a uniform greyscale appearance similar to PARC's DataGlyphs and Microsoft's M-Array encodings. Each piece of paper has a different placement of dots arranged on a fine imaginary grid which defines its location within a 60 million square kilometre area - the size of the pattern space. This location can be resolved by looking at just a set of 6×6 dots in 2cm^2. Redundancy within the encoding itself enables the orientation to be determined.

After the patterns have been described, some limiting requirements that a ceiling painted amorphous computer might force on the pattern are discussed. A new pattern is created that is more suitable for forming the spatial pattern to be used in our ceiling scenario.

In chapter 5 it is shown, through a simulation, how such a spatial pattern and positioning system could be created using the resulting coordinate system from the algorithms proposed in chapters 3 and 4. The properties and issues with an indoor positioning system whose infrastructure is created from a paintable computer utilising the this new pattern is explored in chapter 7.

6.1 Spatial Position Coding Patterns

The DataGlyph address carpet and M-Array technologies both rely upon a particular class of bit sequences called Pseudorandom Noise (PN) codes. These PN codes are briefly described here first.

Pseudorandom Noise codes are binary counters generated in hardware using a Linear Feedback Shift Register (LFSR). An n-bit LFSR is a shift register containing n registers and generates cyclic bit sequences through successive shifts. What follows below is a description of an LFSR in a Galois configuration (rather than the alternative Fibonacci LFSR).

Certain designated registers take their values as the XOR of both the output (or last) register's bit and the preceding register's bit when a shift occurs. The output register's bit is also fed back as the input to the first register. The list of these feedback registers using this XOR logic is called the tap sequence.

In algebraic terms, the contents of the LFSR's registers can also be represented as the binary coefficients of a polynomial. For example, an LFSR with contents 10110 corresponds to the polynomial $x^4 + x^2 + x$. The XOR in the LFSR carries out coefficient arithmetic in the Galois field $GF(2)$. The tap sequence can also be represented as a polynomial of degree n, called the characteristic polynomial, which has the form

$$f(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_2 x^2 + c_1 x + c_0$$

where n is the number of registers and c_i is either 1 or 0 depending of the presence or absence, respectively, of a tap on register i. c_n and c_0 are always 1.

A shift of a Galois LFSR appears just like any normal shift when the last register contains 0. When the last register contains 1, the LFSR will appear as though the normal shift is followed by each tapped register being flipped from 0 to 1 and vice versa[1].

For example, a shift of 10111 (where the rightmost bit is the last register) in a LFSR with the characteristic polynomial $x^5 + x^2 + 1$ is performed by shifting 10111 right once to obtain 01011. As the last register contained 1 before the bitwise shift, the tapped registers are flipped by (01011 XOR 10010), where 10010 represents the tap sequence of the characteristic polynomial. The final content of the registers after shifting the LFSR is 11001, and the single bit of output was 1. The output bits from subsequent shifts form the bits in the cyclic sequence.

Certain configurations of the tap sequence can result in the LFSR producing a maximal length bit sequence, or m-sequence (W.Clark & Weng May 1994). These cyclic m-sequences have the maximum possible period of $2^n - 1$ bits. If the characteristic polynomial yields such a sequence with the maximum period, it is called a primitive polynomial. It is known that there are primitive polynomials for all n.

Other than random noise like qualities, m-sequences have some interesting properties:

It is due to these properties that PN codes are used in many areas including spread spectrum radio communication (Peterson et al. 1995), cryptography (Schneier 1996), and error detection and correction (Berlekamp 1968).

The following sections describe how PN codes are used to create a two-dimensional address spaces. The third positioning code described, called Anoto, uses non-maximal cyclic sequences, but which share property 3 of PN codes.

[1]In contrast, a Fibonacci LFSR has a different configuration of XOR gates and the feedback mechanism is always triggered, regardless of whether the bit in the last register is 0 or 1.

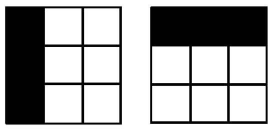
1. An m-sequence has a period of $2^n - 1$.
2. An m-sequence contains precisely 2^{n-1} ones and $2^{n-1} - 1$ zeroes.
3. Each subsequence of length n bits is unique within the entire m-sequence. This is called the 'window property' and is the most useful as the position within the entire code can be determined by looking at only a small section of it.
4. Every possible subsequence of length n is present in the m-sequence. This excludes the subsequence containing all zeroes as this is an illegal state[a].
5. The autocorrelation of the sequence is 1 when the sequence is aligned with itself and $-1/n$ when the sequence is offset by one or more bits.

[a]A related code with an extra 0 is called a DeBruijn code

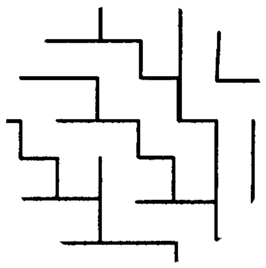
Figure 6.1: Properties of maximal length pseudorandom noise codes, or m-sequences.

6.1.1 M-Array Maze Pattern

Microsoft's M-Array is a pattern that is hardly discernable at a normal reading distance from the paper, but close up it looks much like a maze due to the glyphs that constitute it. Each glyph is 0.5mm^2 in size and encodes one bit with a short line at either $0°$ or $90°$. One corner of the glyph is always missing, as shown in 6.2(a), so they can be unambiguous with respect to their orientation when placed beside other glyphs.



(a) 1-bit oriented glyphs



(b) The M-Array's pattern formed of the glyphs

Figure 6.2: Gyphs encoding one of two possible bits are shown in 6.2(a). The grid lines are not visible but they show here the resolution required to identify the glyphs correctly. When placed beside other glyphs, their orientation is discerned by identifying the unused bottom right hand corner. These glyphs are combined together into a maze like pattern in 6.2(b).

The M-Array is formed by folding a single n-bit m-sequence, m, into a rectangular area of width w by height h. w and h are chosen to be relatively prime and so that all the bits in the entire sequence m covers the whole area exactly (m's period is $w * h$). The bits in the m-sequence are coded consecutively and diagonally within the area. Whenever the bounds of the area are reached, the subsequent bits in the sequence are wrapped around and continue on the opposite side. More specifically, a two dimensional array b_{xy} containing the pattern's bits in the x and y dimensions is constructed by

$$b_{xy} = m_j$$

where $x = j \bmod w$, $y = j \bmod h$, and $j = 0, \ldots, 2^n - 2$. m_j is the bit in the sequence m at index j.

With this pattern, any n by n window of bits can uniquely identify the position within the whole pattern space. By taking n bits of the diagonally coded subsequence in this window and correlating with the m-sequence, the sequence index j is found (this uses properties 3 and 4 of m-sequences listed in figure 6.1). From j, the coordinates can be calculated as $(x = j \bmod w)$ and $(y = j \bmod h)$. The area covered by the entire pattern space is dictated by the period of m. In terms of glyphs, the total coverable area is $2^n - 1$ glyphs.

In practice, a window smaller than n by n can locate the position by using a combination of several different pieces of information. All of the diagonal subsequences within the window may be used together to help determine their sequence positions through property 2 of figure 6.1 and the fact that each index j of these subsequences must be encoding coordinates that all lie within the same window area. The M-Array system also uses images of text and other artifacts on the page, together with a history of recently calculated locations, in order to help identify the current location.

6.1.2 Glyph Address Carpets

Given a two-dimensional area, addressing in one dimension can be provided by placing glyphs representing the bits in a maximal length bit sequence consecutively along one axis. The m-sequence is chosen to have a period length that will span the entire area required. The glyphs are placed in a line at constant intervals along the axis with the first glyph at the origin. This line is then repeated on every row of the other axis.

The glyphs can be as simple as black and white squares representing a 1 or 0 respectively, or more sophisticated representations of many bits. In the case of PARC's DataGlyphs, short lines rotated by +/-45° around 0.25mm in length encode single bits. These glyphs, shown in figure 6.3, were chosen to provide a neutral appearance on paper and to allow easy reading by machine.

An object's position along this axis can be established by identifying the glyphs in its local area and converting them to a bit matrix. Assuming the matrix's orientation is correct, a subsequence, s, of the m-sequence is extracted. By using properties 3 and 4 of m-sequences listed in figure 6.1, the sequence index, j, of

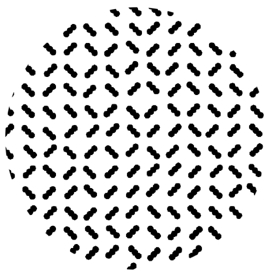


Figure 6.3: Enlarged view of 1 bit encoding glyphs forming a DataGlyph pattern

the m-sequence can be obtained by searching through the entire sequence for the subsequence s (essentially a sliding window correlation). The distance of the matrix from the pattern's origin is calculated by multiplying the constant interval distance between glyphs and the sequence index j.

In its current form the layout of the code is not sufficient to determine the orientation while it remains ambiguous as to whether it is oriented at 0 or 180 degrees. For two-dimensional coordinates and orientation information, some additions are required to the pattern. Glyphs encoding another m-sequence, with a different characteristic polynomial, are placed along the same axis, between the rows of the first sequence. The bits of the second m-sequence are shifted by a constant phase on each progressive row. If the indices of the first sequence reflect the extent in the x dimension, then the distance along the y axis is the degree to which the second sequence on the row below has been shifted (and multiplied by the appropriate glyph interval distance).

This address carpet now provides 2D location information by looking at any small section of at least 4 rows of n bits, where n is the degree of the m-sequence's characteristic polynomial. The perceptible patterned appearance of the first sequence can be alleviated by shifting both sequences by a different constant phase on each progressive row (for example, the two interleaved sequences could be shifted by 2 bits in opposite directions). By using suitable glyphs, such as elongated slashes, the visual appearance would take on an unobtrusive random quality.

6.1.2.1 Orientation

When the glyphs are converted to a bit matrix, the orientation is first resolved to either 0 or 180 degrees. This is deduced by correlating the bits of the odd and even rows with each subsequent odd and even row, after shifting by the appropriate phase shift. This correlation value is then compared to the same correlation performed on a bit matrix rotated by 90 degrees. Out of the rotated and unrotated bit matrices, the matrix with the highest correlation value is known to be aligned with either 0 or 180 degrees.

Table 6.1: An example bit matrix of glyphs with different orientations. The bits of two m-sequences, a and b, are shown. Sequence a has a phase shift of 1, and sequence b has a phase shift of -1

(a) Correct orientation

a_0	a_1	a_2	a_3	a_4	a_5
b_2	b_3	b_4	b_5	b_6	b_7
a_1	a_2	a_3	a_4	a_5	a_6
b_1	b_2	b_3	b_4	b_5	b_6
a_2	a_3	a_4	a_5	a_6	a_7
b_0	b_1	b_2	b_3	b_4	b_5

(b) Rotated by 90 degrees

b_0	a_2	b_1	a_1	b_2	a_0
b_1	a_3	b_2	a_2	b_3	a_1
b_2	a_4	b_3	a_3	b_4	a_2
b_3	a_5	b_4	a_4	b_5	a_3
b_4	a_6	b_5	a_5	b_6	a_4
b_5	a_7	b_6	a_6	b_7	a_5

(c) Rotated by 180 degrees

b_5	b_4	b_3	b_2	b_1	b_0
a_7	a_6	a_5	a_4	a_3	a_2
b_6	b_5	b_4	b_3	b_2	b_1
a_6	a_5	a_4	a_3	a_2	a_1
b_7	b_6	b_5	b_4	b_3	b_2
a_5	a_4	a_3	a_2	a_1	a_0

(d) Rotated by 270 degrees

a_5	b_7	a_6	b_6	a_7	b_5
a_4	b_6	a_5	b_5	a_6	b_4
a_3	b_5	a_4	b_4	a_5	b_3
a_2	b_4	a_3	b_3	a_4	b_2
a_1	b_3	a_2	b_2	a_3	b_1
a_0	b_2	a_1	b_1	a_2	b_0

Table 6.1 shows the four different possible orientations of the bit matrix. The bits from the odd and even rows of table 6.1(a), which is in the correct orientation, are:

Odd rows	Even rows
$a_0\ a_1\ a_2\ a_3\ a_4\ a_5$	$b_2\ b_3\ b_4\ b_5\ b_6\ b_7$
$a_1\ a_2\ a_3\ a_4\ a_5\ a_6$	$b_1\ b_2\ b_3\ b_4\ b_5\ b_6$
$a_2\ a_3\ a_4\ a_5\ a_6\ a_7$	$b_0\ b_1\ b_2\ b_3\ b_4\ b_5$

Correlating the first odd row with the second odd row, after having shifting the second odd row by sequence a's phase shift of 1, yields a perfect correlation value of 1. Likewise, correlating the second and third odd rows, with the third row shifted by 1, also results in a correlation value of 1. If these rows were instead correlated after having shifted the subsequent row by sequence b's phase shift of -1, the resulting correlation value would be close to 0 (see property 5 of PN Codes in figure 6.1). In a similar fashion, the even rows, which contain sequence b, correlate perfectly with each other when shifting with b's phase shift of -1. This high correlation of the odd and even rows is also true when the bit matrix is incorrectly oriented at 180 degrees to normal (as in table 6.1(c)).

If the matrix were oriented at 90 or 270 to normal, the correlation would be very low, regardless of the phase shift, because the rows and columns of the matrix have been transposed and the two sequences are intermingled. For example, the bits of the first and second odd rows from table 6.1(b) (matrix rotated by 90 degrees) are the low correlating sequences of $b_0a_2b_1a_1b_2a_0$ and $b_2a_4b_3a_3b_4a_2$ respectively. As a result of this low correlation, a simple test for whether the orientation is aligned to 0 or 180 is to compare the correlation values of both the captured bit matrix, and a copy of the bit matrix rotated by 90 degrees.

Once the orientation of a bit matrix is resolved to 0 or 180 degrees, it is known which phase shift correlates highly for the odd and even rows – i.e. the direction, in which the odd and even rows are shifting, is known. With this knowledge, all the odd rows, and separately all the even rows, may be combined to create a sequence of bits slightly longer than the width of the captured bit matrix. For example, all the odd rows from table 6.1(a) would combine to form the 8-bit long subsequence $a_0a_1a_2a_3a_4a_5a_6a_7$, whereas each individual row contains a sequence only 6-bits long.

With the longer combined subsequences from the rows in the bit matrix, the orientation can be fully determined by correlating these subsequences both forwards and backwards within the full m-sequence. The directions with the highest peak correlations (the positions of the subsequences within the whole m-sequence) are then used to disambiguate the orientation to one of 0 or 180 degrees.

6.1.2.2 Scale

The total area that may be covered by the two entwined codes is dictated by the m-sequence's period. Two n-bit m-sequences can cover a unique addressing area of $2^n - 1$ bits in width and $2^{n+1} - 2$ in height when using the scheme above. If the pattern is allowed to continue beyond this area, then the pattern starts to repeat and the addresses will become ambiguous.

The area covered by this simple pattern is relatively small, but it can be increased by introducing a third m-sequence on every third row which purely serves to label the address space. This labelling m-sequence's relative starting position can distinguish the address space as one out of a set of $2^n - 1$ such labelled address spaces which may be tiled over a surface in some pre-determined fashion. The window area required to determine the location will increase by one row but the resulting scalability of the pattern is much improved and the addressable area is shown later in this chapter in table 6.5.

6.1.3 Anoto Dot Pattern

The Anoto pattern, as shown in figure 6.4, can with a window just larger than 6 by 6 glyphs packed in an area of 2mm^2 determine a unique position within the pattern space of 4.6 million km^2.

The pattern is comprised of dots offset from the intersections of a virtual grid. Each glyph is a square containing a dot placed left, right, above or below the centre and encodes two neighbouring bits of a bit matrix. The consecutive bits of a known cyclic bit sequence are placed downwards on every even column of the bit matrix. The starting position, or index within the bit sequence, at the top of each column

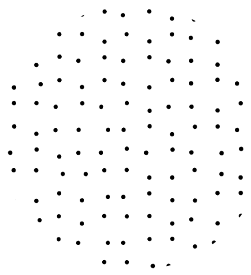


Figure 6.4: Anoto's dot pattern. Each glyph encodes two bits; one bit from a vertical coordinate sequence (giving x coordinates) and one bit from a horizontal sequence (giving y coordinates)

is selected such that the difference, or shift, in one position from neighbouring columns' positions gives the extent in the x dimension. In a similar fashion the y coordinate is given from bit sequences laid out horizontally over the rows of the matrix, with each consecutive bit placed in the next odd column to the right. It is the starting positions of these y codes, known after finding the x coordinate, that give the extent in the y dimension.

Table 6.2: Example of bit matrix extracted from 4 by 4 glyphs

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
R_0	1	1	1	0	1	1	1	0
R_1	1	1	0	1	1	0	1	1
R_2	1	0	1	1	1	0	1	1
R_3	1	1	0	1	1	1	0	0

As an illustrative example, if a window of 4 by 4 glyphs is used, a bit matrix of 8 by 4 bits can be extracted, such as in table 6.2.

Table 6.3 shows the bits from the even columns C_0 through C_6 which encode the x dimension codes. If a 7 bit long cyclic sequence such as 1110101 is used for both the x and y codes, the example has encoded the positions of 6, 2, 6 and 0 in the sequence for the vertical x codes. The differences between the neighbouring x codes are -4, 4 and -6, which modulo 7 gives 3, 4 and 1. These differences between the positions in the bit sequence of the x codes provide three digits in different bases which give the extent in the x direction.

The pattern is created such that the x codes' index difference between the first, second and third columns is in the range 3 to 6 and the difference[2] between the third and fourth columns is 1 or 2. This constraint on the range of the differences between neighbouring x codes' positions is repeated in this way for all columns in the pattern space.

[2]A difference of 0 is not used to avoid an obtrusive appearance where the columns' glyph positions are very similar

Table 6.3: X bit codes running down the even columns in the bit matrix

	C_0	C_2	C_4	C_6
R$_0$	1	1	1	1
R$_1$	1	0	1	1
R$_2$	1	1	1	1
R$_3$	1	0	1	0
Seq. pos.	6	2	6	0
Diffs.		-4	4	-6
Diffs. mod 7		3	4	1

Any four neighbouring x codes are guaranteed to have one of the three differences being the smallest. This smallest difference constitutes the least significant digit of a three digit number, in mixed base, that identifies that particular window of four x codes. This mixed base number increases by one for every three neighbouring columns' x codes. The least significant digit is in base 2 whilst the other two digits are in base 4, allowing 32 unique numbers to be encoded.

In the example above, the differences of 341 yield the mixed base number 010 ($[3-3][4-3][1-1]$) which indicates that these four x codes reside at the 3rd ($(0*8)+(1*2)+(0*1)=2$) such window. This window begins at the 7th column of glyphs and ends at the 10th column of glyphs, where the window's least significant bit is encoded by the 9th and 10th glyph columns' difference.

Table 6.4: Y bit codes running along the rows of the odd columns in the bit matrix

	C_1	C_3	C_5	C_7	Seq. pos.	Start pos.
R$_0$	1	0	1	0	2	3
R$_1$	1	1	0	1	1	2
R$_2$	0	1	0	1	3	4
R$_3$	1	1	1	0	0	1

After the extent in the x dimension is decoded, the y codes (see table 6.4) are investigated and the starting indices of the sequences are calculated. In the example, the horizontal y codes along rows R_0 through R_3 in the odd columns have sequence positions of 2, 1, 3 and 0. The differences of these indices is not used for recovering the y dimension's mixed base position. Instead, the current indices together with the known x dimension are used to calculate the positions at which the y codes began at the start of the y axis.

The starting indices of these y codes are calculated by $(2-6) \bmod 7 = 3$, $(1-6) \bmod 7 = 2$, $(3-6) \bmod 7 = 4$, and $(0-6) \bmod 7 = 1$, where 6 is the extent in the x dimension that these y codes are seen (i.e., the bit matrix started at the 7th column of glyphs in the pattern). The starting indices provide a four digit number in mixed base that increments every four rows in the y dimension. The constraint on the starting positions is similar to those on the x codes' differences. The first, second and third rows all will have a starting index between 2 and 6, whilst every fourth row's index will represent the least significant digit having a value between 0 and 1. A four digit number in this mixed base allows for 250 unique windows for each set of four y code rows.

The window identified by the mixed base number 1021 (obtained from the y codes' starting indices of $(3-2)$, $(2-2)$, $(4-2)$, and $(1-0)$) begins at the 217th glyph row $((1*50)+(0*10)+(2*2)+(1*1)$ is the 55th window in the y direction. Each window is 4 rows in height).

6.1.3.1 Orientation

The example above has assumed that the seen glyphs were originally oriented correctly. The orientation can be known from a combination of the glyphs themselves and from a redundancy in the cyclic bit sequence used.

The form of the two-bit glyphs used is such that they are interpreted into different bit pairs depending on whether they were oriented at 0, 90, 180 or 270 degrees to their correct orientation. If the orientation was not initially correct, the interpretation of the glyphs leads to a different sequence of bits in the bit matrix that are reversed and inverted versions of the original sequence. The 7 bit sequence was chosen as it has, at most, two zeroes in any 5 bit long subsequence.

If the bits in the subsequence were inverted due to an incorrect orientation, this could be detected by looking at an extra glyph immediately outside of the 4 by 4 glyph window and noticing that there are an impossible number of zeroes in the 5 bit subsequence. In such a case, a rotation by a different multiple of 90° should be tried.

6.1.3.2 Scale

The area this type of pattern covers is governed by the period of the cyclic bit sequence used and the number of bits required to determine the index of a subsequence within the sequence. With the 7 bit long sequence in the example above, 6 possible differences in index numbers can be used to encode the x coordinates in a mixed base number (base 4, base 4, base 2) that addresses up to 96 glyph columns (32 different sets of 3 differences over 96 even columns of bits). There are 7 different possible starting index positions for a 7 bit long sequence, so the y codes can address in a mixed base number (base 5, base 5, base 5, base 2) up to 1000 glyph rows (250 window sets of 4 rows). This allows every 4 by 4 set of glyphs to encode a unique position within 96000 different such positions.

The addressable area can be increased a little further by changing the range of values of the digits in the mixed base number. The least significant digit can always be identified by being the lowest number. In the example the x code's least significant digit is in the range of 1 to 2 and together with the other two numbers effectively labels 32 different windows of 3 columns. If the lowest digit amongst the three is set to be 1 then the other numbers can be put in the range of 2 to 6. This allows for just 25 windows of 3 columns in the x direction; however, 16 additional windows can be added after these first 25 by changing the least significant bit to be 2 and the other three numbers to be in the range 3 to 6. By adding additional windows in this fashion, the area is extended by 9 windows, then 4 and finally 1 window. In total this provides 55 addressable windows in the x direction compared to just 32. This is an increase in width from 96 to 165 glyphs in width.

After both the x and y windows' position has been determined, the starting indices of the x codes can be calculated as the offset from the top of the addressable space is now known. The area can be increased yet further by letting the x codes' starting indices vary whilst keeping the differences the same. There are seven possible starting positions for the x codes which can be used to label the 165 by 1000 glyph region as one out of seven such glyph regions. Given a known layout of these glyph regions, the area is increased sevenfold to provide $1,155,000$ different unique positions from a 4 by 4 set of glyphs.

6.2 A Pattern Suitable for an Amorphous Ceiling

Unless the density of nodes is extremely high, the accuracy of the glyphs created by a ceiling painted with an amorphous computer are likely to be of a low quality at reasonably small sizes. It may be the case that some parts of the glyph are not even formed. This precludes any use of larger composite glyphs that must be decoded in their entirety, such as 2D bar codes, as the chances that some critical piece of the glyph being ill-formed or missing is high. Additionally, any complex glyph requiring an exact form is also likely to be formed unreliably. Complex glyphs with many different forms also increase the capturing devices' sensitivity to interference.

The three patterns described in section 6.1 each have a 'floating' window property. This means that any small portion, or window, of the pattern is used to decode it, rather than requiring the whole pattern in its entirety as with 2D barcodes. This floating window property has two advantages over large complex composite glyphs for use in an amorphous ceiling setting. Firstly, they reduce the area required in a captured image to decode the position as any partial part of the surface contains a useable window, whereas a system based on composite glyphs requires the whole composite glyph in the image. Composite glyphs that encode the position, such as barcodes, would also require a delineating boundary to separate it from other encoded positions, which would increase the area required. Secondly, when given a slightly larger area than the floating window size itself, the redundancy of the extra information can be used to obtain a position even if some of the glyphs are rendered illegible (through damage or a bad formation). In the case of composite glyphs, if some part of the glyph is illegible, the entire glyph becomes unusable.

Due to the random placement of nodes in a ceiling painted with an amorphous computer, it is not guaranteed that some small area will be covered by any nodes. Also, it is not guaranteed that one node is not in error. In order to form a mark on the ceiling with some reasonable likelihood, the intended mark must be formed over a reasonably sized area and not by not just one node, but by a number of nodes in the area. This means that the resolution available for forming a pattern on the painted ceiling is likely to be low, unless the node density is extremely high.

The glyphs used in the three patterns are all reasonably small and simple glyphs; nevertheless, they still require a certain resolution to be properly defined. For example, the 1-bit glyphs used in the M-Array pattern require that nine parts of each glyph be identified in order to correctly decode the glyph and its orientation. The Anoto pattern's 2-bit glyphs require a still higher resolution.

The ideal glyph would be one with the greatest amount of information for the resolution required to represent it, whilst still being as unambiguous as possible. The smallest minimalist glyph would be a simple square encoding a single bit (for example, by being coloured either black or white). The glyphs can be arranged in a rough checkerboard like pattern. Such an arrangement would also mean that the decoding of an image of these low quality glyphs would be greatly simplified. Any small skew in a checkerboard pattern is not likely to greatly hinder the analysis of the relative positions of the glyphs and their correct relative positions in a lattice.

Simple 1-bit square glyphs can be easily recognised by a capturing device. The shape itself is easily detected and there is only one shape to detect. The shape's simplicity also allows for it to be detected and recognised even if part of it is damaged or ill-formed. As there are only two colours of the shape, for example black and white (or reflecting and absorbing infrared), faster image processing may be performed using monochromatic images rather than colour images. A greater contrast between the two colours can also help to identify the glyph.

The 1-bit glyphs used by the DataGlyph pattern can be replaced by simple 1-bit square glyphs while only sacrificing some of the property of being visually unobtrusive. Both Anoto and M-Array's glyphs are specifically chosen to help determine the orientation of the pattern to decode, so they may not be directly replaced with the simpler 1-bit square glyph without another mechanism by which the orientation may be determined.

The potential scale of each of the three patterns is shown in table 6.5, assuming that they all used 1-bit glyphs of the same size that each provide the correct orientation. From this table it can be seen that the Anoto pattern outperforms the other two in terms of the addressable area from the window size used, even with its sequence's redundancy for orientation.

With the 1-bit glyphs used in table 6.5 the minimum window area required to decode the pattern is rectangular, rather than square, in the case of the DataGlyph and Anoto patterns. This is because only four rows of sequences are required in unlabelled DataGlyph patterns to decode the position, so when the m-sequence length increases beyond 4, more columns than rows are required in the the minimum window area. With Anoto, as a direct consequence of moving from 2-bit glyphs to 1-bit glyphs, the two sequences are multiplexed on the rows, leaving twice as many columns as rows (as is seen by the bit matrix back in table 6.2).

It is desirable to have a square minimum window area, in order to increase the chance that a simple camera pointed at the ceiling will capture a whole window area, and to reduce the redundancy in captured windows.

Table 6.5: The addressable area is given for the four spatial patterns, under different sequence lengths. Also shown is the minimum captured window area required to decode the address, which is not necessarily square. This assumes each glyph is 1cm^2 in size, encodes 1-bit, and has orientation features (if required for the pattern). The number in brackets is the length, in bits, of the subsequence used to decode the position within the whole bit sequence. The Amorphous Ceiling Pattern has two subsequence lengths - the first is for the x and y codes, and the second is for the labelling code.

Window	M-Array	Address Carpet		Anoto	Amorphous Ceiling Pattern		
Area		Unlabelled	Labelled		Sgl. Label	Dbl. Label	Trpl. Label
16cm^2		450cm^2 (4)					
20cm^2		1922cm^2 (5)					
24cm^2		0.79m^2 (6)	1.01m^2 (4)				
25cm^2	31cm^2 (5)						
28cm^2		3.23m^2 (7)					
30cm^2			8.94m^2 (5)				
32cm^2		13.01m^2 (8)					
33cm^2				231m^2 (4)			
36cm^2	63cm^2 (6)	52.22m^2 (9)			37.8m^2 (3,6)		
40cm^2		209.3m^2 (10)					
42cm^2			75m^2 (6)				
44cm^2		838m^2 (11)					
48cm^2		3353.8m^2 (12)					
49cm^2	127cm^2 (7)		614.5m^2 (7)				
64cm^2	255cm^2 (8)		4974m^2 (8)		1.1km^2 (4,8)	280km^2 (4,8)	
72cm^2			40030m^2 (9)	10271Mm^2 (6)			
81cm^2	511cm^2 (9)						
90cm^2			0.32km^2 (10)				
99cm^2			2.57km^2 (11)				
100cm^2	1023cm^2 (10)				0.438Mm^2 (5,10)	448Mm^2 (5,10)	0.46Gm^2 (5,10)
120cm^2			20.6km^2 (12)				
121cm^2	2047cm^2 (11)						
144cm^2	4095cm^2 (12)						

In summary, a pattern more suitable for use in positioning from an amorphous ceiling would have the following properties:

1. Smallest square window area providing the largest addressable area
2. Use simple 1-bit square glyphs arranged in a checkerboard lattice
3. Provide window orientation not from the glyphs but from the code itself

The Anoto pattern clearly satisfies property 1 and would be ideal for use in our scenario if it also satisfied properties 2 and 3. Unfortunately, Anoto requires special glyphs and a carefully chosen redundant cyclic code to provide the captured window's orientation information. When these glyphs are translated into 1-bit square glyphs, this orientation information is lost and the window area is no longer square but rectangular. The DataGlyph code satisfies properties 2 and 3, but does not have such a large addressable area from a small window. The remainder of this section describes a new pattern which satisfies all three properties and is based on elements of both the Anoto and DataGlyph patterns.

6.2.1 Moving from 2-bit Anoto glyphs to 1-bit Square Glyphs

If the Anoto pattern were to use 1-bit square glyphs without the orientation information, the glyphs can be laid out according to the bit matrix as shown in table 6.2 or table 6.6(a). This layout is rectangular.

Anoto requires a redundant cyclic to discern the orientation of its glyphs. Unlike Anoto's 2-bit orientated glyphs, these 1-bit square glyphs do not provide orientation information. Another means of providing orientation, when using 1-bit glyphs, will be used and explained in section 6.2.2.

For 1-bit glyphs, the shorter redundant cyclic code of Anoto may be replaced by a longer m-sequence with a maximum period, increasing the addressable area. For example, where an Anoto style pattern might use a redundant 4-bit cyclic sequence with a period of 7 to gain help during orientation, it is now possible to instead use a 4-bit m-sequence with the full cyclic period of 15.

6.2.2 Labelling, Orientation, and Square Window Areas

If another m-sequence is now introduced into this new code on every second row (in much the same way as the DataGlyph pattern's labelling), the window area is now square having doubled in height. Table 6.6(b) shows the layout of the window area after the introduction of such a sequence. The starting index of this new m-sequence can be found after decoding the x and y code window numbers of the Anoto portion of the code. This starting index is used to further label the area as one out of $(2^n) - 1$ such areas (for an n-bit m-sequence). These labelled areas can then be tiled beside each other to increase the addressable area, with some consideration for captured windows spanning the edges of the tiling (see section 6.2.4).

It can be noted that the subsequence length of the x and y codes is half that of the labelling codes, as is shown in figure 6.5 by the two numbers within brackets.

Table 6.6: Example multiplexing of horizontal, vertical and labelling sequences.

(a) Shows the bits of four vertical x subsequences ($x1$, $x2$, $x3$, $x4$, running from top down to bottom) and four horizontal y sequences ($y1$, $y2$, $y3$, $y4$, running from left to right). The subscript shows the offset from the start of that subsequence. The bits of the $x3$ and $y2$ subsequences are shown in bold.

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
R$_0$	$x1_1$	$y1_1$	$x2_1$	$y1_2$	$\boldsymbol{x3_1}$	$y1_3$	$x4_1$	$y1_4$
R$_1$	$x1_2$	$\boldsymbol{y2_1}$	$x2_2$	$\boldsymbol{y2_2}$	$\boldsymbol{x3_2}$	$\boldsymbol{y2_3}$	$x4_2$	$\boldsymbol{y2_4}$
R$_2$	$x1_3$	$y3_1$	$x2_3$	$y3_2$	$\boldsymbol{x3_3}$	$y3_3$	$x4_3$	$y3_4$
R$_3$	$x1_4$	$y4_1$	$x2_4$	$y4_2$	$\boldsymbol{x3_4}$	$y4_3$	$x4_4$	$y4_4$

(b) Shows the horizontal and vertical sequences from table 6.6(a). A labelling sequence, b (with a phase shift of 2), is introduced on every other row

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
R$_0$	$x1_1$	$y1_1$	$x2_1$	$y1_2$	$x3_1$	$y1_3$	$x4_1$	$y1_4$
R$_1$	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
R$_2$	$x1_2$	$y2_1$	$x2_2$	$y2_2$	$x3_2$	$y2_3$	$x4_2$	$y2_4$
R$_3$	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
R$_4$	$x1_3$	$y3_1$	$x2_3$	$y3_2$	$x3_3$	$y3_3$	$x4_3$	$y3_4$
R$_5$	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}
R$_6$	$x1_4$	$y4_1$	$x2_4$	$y4_2$	$x3_4$	$y4_3$	$x4_4$	$y4_4$
R$_7$	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}

This new labelling m-sequence can easily be distinguished from the Anoto portion of the code as it will be identical on each subsequent row (the correlation of these rows is 1). If no identical rows are seen in the captured window, then the orientation must be rotated by +/-90°.

To fully disambiguate the window's orientation, the labelling m-sequence is shifted by a constant phase on each subsequent row[3]. The orientation of a captured window, or bit matrix, may now be fully determined by looking to the direction in which these labelling sequences are shifted (as described in section 6.1.2.1 for DataGlyph patterns).

6.2.3 Multiple Labels

As the window size is increased, it becomes possible to add a second labelling m-sequence with different starting positions between the first labelling m-sequence. This increases the address space by another $(2^n) - 1$. For example, in table 6.6(b), there is room in the window for another labelling sequence – two of the four rows of the labelling sequence could be used for the second sequence. Similarly, as the window size is increased, there becomes more space for further labels. An extra label can be added so long as there is always at least two rows of each labelling sequence in any captured window (in order for the labels to be identified and the orientation to be determined).

The window size and resulting address space for this new code is shown in table 6.5. With double labelling and a window size of just $8*8$ glyphs in 64cm^2, a position

[3]In fact, this gives the potential to increase the length of the labelling m-sequence as the reconstructed labelling code subsequence can be greater than the width of the window. This added complexity is not explored here.

can be located within a 280km^2 area. Some considerable addressable area is lost due to the redundancy of repeating the labelling code on several rows. Nevertheless, even with this compromise to gain orientation information, a substantial improvement in addressable area by an order of magnitude is achieved over the DataGlyph pattern which only covers an area of 4974m^2 from the same window area.

6.2.4 Tiling Labelled Areas and Edge Conditions

If the labelled areas are tiled beside each other, with no vertical offset, there is a problem when decoding a captured window that spans the edge of two different labelled areas. Two different labelling sequences from two adjoined areas will be decoded, yielding a spurious position in the labelling sequence, and consequently a spurious label. Additionally, the x and y sequences would decode to a spurious location within the area. With this method of tiling, it is not possible to detect this error when capturing only the minimum window area[4].

A 1-bit vertical offset can be introduced when tiling the labelled areas. For example, every other horizontal tile can be offset vertically by one bit. The offset causes windows captured at the abutment of two tiled areas to be indecipherable. This is because the two different labelling sequences are no longer confused with each other, but are instead confused with the x and y codes. This mixing causes a non-correlation of the labelling sequences, so the labelling sequences can not be identified within the captured bit matrix. This flags up an erroneous condition.

To recover from the erroneous edge condition, the width of the captured window must be increased until a subsection of the captured window contains unperturbed rows of the labelling sequences which can be identified through positive correlation. The width for which the captured window must increase on one side is equal to the number of columns that overlap the edge on the opposite side of the window. An alternative way to recover from this error condition is to keep capturing new windows in different nearby areas until one is found without an error.

6.3 A Proposed Glyph Address Carpet Indoor Location System

This section proposes a possible indoor positioning system using the self-organised glyph address carpet as the system's infrastructure. The theoretical properties and classification of such a system are introduced in the following subsections. An example embodiment of the system is given in chapter 7 from a simulation using the coordinates formed from the algorithms proposed in chapters 3 and 4.

An address carpet can provide location information even when only a very small portion of it is seen. The floor or ceiling of a room can be covered with this location pattern perhaps by laying a specially made covering, placing patterned tiles,

[4]The error could be noticed if the captured window was large enough to observe that the positions of the decoded x sequences are not be consecutive

printing indelible marks on the surface, or even by having self-organised computing particles dye or mark the surface with the correct glyph. Once the pattern is in place, it becomes fully passive and no communications or computing is necessary by such an infrastructure. The pattern can be inconspicuous and in some circumstances could even be marked by ink that only reflects light outside of the human visual spectrum.

It is envisioned that the location sensing device itself could be a small CMOS camera. CMOS cameras with reasonable resolution use between 250 and 375 milliwatts, are currently commercially available in sizes less than 1cm^2 and with mass production can be priced very reasonably. This CMOS camera would be integrated with a wearable computer or handheld computer such as a mobile phone.

The camera could comfortably be attached to a shoe directed at a floor based pattern, or in the case of a ceiling, a phone camera simply needs to be pointed upwards. Wherever a person walks, the mobile computer would decipher the glyphs on the floor or ceiling to reveal both the location and the orientation of the device. It is reasonable to say that wherever the device points is the point of interest and attention. Whilst orientation can to some extent be estimated by looking to a recent history of the trajectory of movement, such estimations can fail if a person turns on the spot. This system provides accurate orientation information whenever it is visible.

Whilst the address carpet covers only a two dimensional area, there is nothing to stop different areas of the pattern, or differently labelled patterns being associated with the different floors of a building.

6.3.1 System Properties

Different location systems each tackle diverse problems and applications. Classification is important to order different location systems with respect to their significant similarities and differences and this classification is invaluable when deciding upon the correct technology for a particular application. Hightower & Borriello's (2001) taxonomy provides a useful classification of indoor location systems for used in pervasive computing. The properties, under this taxonomy, of our proposed glyph carpet indoor location system are enumerated in the sections 6.3.1.1 to 6.3.1.6 below.

6.3.1.1 Physical Position and Symbolic Location

The pattern is resolved into a physical address rather than a symbolic location. Symbols relating to rooms, floors, or any arbitrary area may nevertheless be labelled from some information source external to the pattern infrastructure. As the pattern remains fixed to its surface after deployment, the position for each area will not change so a static database can easily be created to annotate physical areas of the pattern space with symbolic locations.

Whilst the pattern is fixed to the surface, the surface itself may move. For example, surfaces attached to trains or elevators. In such cases the area covering the surface could be annotated with this special meaning, however it is the application or pervasive computing middleware that provides this understanding rather the infrastructure itself.

6.3.1.2 Relative and Absolute Positions

Each location device determines an absolute position within the address carpet, rather than the relative position to some other other object's position. It would be up to the application or some pervasive computing middleware to provide this relative location information.

When creating the address carpet through the proposed algorithms in this thesis, the coordinate system created is not related to any other external coordinate system. If two separate disjoint ceilings were covered, the axis of the two coordinate systems would not be oriented in the same direction. The only similarity would be size of the glyphs.

Some simple applications, such as movement tracking within a room, may only require relative coordinate systems. However, if the orientation of disjoint areas must be identical, or if the location within some external coordinate system is to be known, then the application or middleware must apply a known affine transform to the decoded address carpet coordinates.

6.3.1.3 Localised Location Computation

Generally in indoor positioning systems the responsibility for computing a mobile device's location is either held by the infrastructure in the environment or by the mobile device itself. In the former case, the infrastructure must be secure and trusted if privacy is to be maintained. If it is the device that computes the location, privacy can be guaranteed as no other entity can glean another's location from the infrastructure itself; it is only if the localising device discloses the location, that the location can be known by others.

No maintenance or power is required by the address carpet infrastructure once deployed. The infrastructure is entirely passive and devoid of any computation. The locating device alone, rather than the infrastructure, computes the location and no communication between the device and the infrastructure is made. This ensures privacy and prevents eavesdropping.

Many useful applications do require knowledge of different device's locations though. Without any centralised point in the infrastructure providing global device location knowledge, extra effort must be expended by the application or pervasive middleware to discover other device locations.

6.3.1.4 Accuracy and Precision

Accuracy is the ability of a measurement to match the actual value of the quantity being measured, whereas precision is the ability of a measurement to be consistently produced.

If a position has been successfully resolved then the accuracy is reliant upon how exact the mapping is between the theoretical location of the glyph mark on the floor and its actual physical mark. It is not difficult to conceive of a sub-centimetre position accuracy and orientation to within a few degrees when using perfectly formed and high resolution glyphs.

Whenever the precision falls short of 100 percent, it is attributable to one of these causes:

- A sensing failure. The camera's view is obstructed or its resolution is too low to discern glyphs correctly at such a distance.
- An infrastructure failure. The glyphs themselves are either malformed, have been severely damaged, or have been destroyed rendering them illegible. If only minor damage to the glyphs has been sustained, the location may still be resolved if a wide enough view of the local pattern is available. If damage to the glyphs is inevitable, error checking and correcting can be incorporated, with some redundancy, into the spatial pattern itself.

Accuracy is influenced by the definition of the glyphs themselves. The location of the glyph mark will probably be determined by its centre of gravity. If a glyph is recognised but the defining edges of it are difficult to pinpoint exactly, then there will be an uncertainty when determining its nominal centre; the glyph's centre of gravity may be displaced from its intended nominal centre. Precision in identifying the placement of the glyph will be lowered if they are not straight, if the sharpness and clarity is imperfect, or if the resolution of the camera itself is limited.

The precision of the pattern's markings over the space could also influence the accuracy of the system. A captured image window of glyphs from the checkerboard pattern is analysed to identify the glyphs and their respective centres of gravity. It can be the case, due to an oblique coordinate system used, a distorted camera lens, or badly formed glyphs, that the resulting lattice of centres is also oblique. This brings in an uncertainty as to how the ideal regular lattice of glyph's nominal centres should be mapped to the oblique lattice of glyph's centres of gravity.

The distance of the camera from the pattern's markings can influence the precision. In addition to low resolutions, the distance and angle at which the camera is pointed towards the ceiling can complicate the way in which the captured window image is interpreted. When the camera is not completely perpendicular to the ceiling, a window will be captured that is some distance away from the device's actual 2D location.

A remedy for images captured at an angle may be found in software by recognising the perspective of the captured image (Tsai 1987, Zhang 2000) together with the distance of the glyphs from the camera[5]. From the perspective and distance, the offset of the device from the decoded window's location can be calculated. Unfortunately, the accuracy of such a software approach may be aggravated in cases where the captured window itself is badly formed or oblique. Hardware assistance from cheap tilt switches could be used, or the camera could be mounted such that it always points directly upwards.

6.3.1.5 Scale

The scale of the address code area was investigated in section 6.2 and shown capable of covering 280km^2 with a window size of 8cm by 8cm and a relatively small glyph size of 1cm^2. This can easily cover all of the ceilings in the largest office building or even the largest factory. With a 10cm by 10cm window area of the same sized glyphs, the entire addressable area can cover the whole surface of Jupiter roughly 7.5 times over.

Some further investigation into glyph sizes and scaling issues is made in chapter 7 in light of the simulation results of the propagated coordinate system.

Unlike other systems, the infrastructure performs no active role and performs no communication with the mobile devices. There is no theoretical scaling limit on the numbers of devices that may simultaneously use the infrastructure.

6.3.1.6 Limitations and Cost

In the case of an amorphous computer painted over a ceiling, the cost of the paint with suspended computing particles could vary greatly depending on the form of hardware used by the particles. As no devices at such a scale are yet available, no estimate of the costs involved can be made.

There are visibility limitations when the camera's sight is blocked by an obstruction or inhibited by extreme lighting conditions. The camera may also suffer from other visual effects such as colour bleed, motion blur and radial distortion.

If a large area of the pattern is destroyed, it becomes a dead spot in the address space. Presuming the nodes involved in forming the pattern are long since drained of power, there is no simple way to maintain the pattern. If the nodes were active however, the dead spot could be covered over with new nodes which automatically fill in the dead area and recreate the pattern.

[5]The distance may be determined to an extent from the known size of the glyphs, the camera's known resolution and depth of field

6.4 Summary

An amorphous computer is comprised of unreliable nodes in an unknown topology. Nodes in this kind of network are used to create machine readable marks, or glyphs, in a visual positioning system. The function of the glyphs is to represent the bits in a 2D spatial positioning code – a small window area of which yields the coordinates of that window's location on the ceiling.

As nodes are very small and simple devices, the discharged ink from not just one, but many nodes, will comprise each single glyph. This, together with the unreliable nature and potentially uneven distribution of the nodes, puts a design pressure on the form of the glyphs to be as simple as possible. The more simple a glyph is, the higher the chances are that it can be correctly identified, through machine vision techniques, when its form is less than perfectly created. The glyph form chosen was that of black and white squares, encoding a single bit, arranged along a checkerboard lattice.

Three existing visual positioning codes, constructed from cyclic bit sequences, have been detailed and their scaling properties investigated. These codes have been used mainly in the area of digital paper, to provide a pen with its location on the paper, but the codes may just as easily be used on other surfaces with other camera equipped devices. The surface of a ceiling is potentially very large, so it is desirable for the positioning codes to have the largest addressable area from the smallest locating window area.

The M-Array code shows a simple use of cyclic bit sequence for encoding positions, but it is unusable in our scenario for two reasons. The area covered by the M-Array code scales poorly with the window area required to decode the position. M-Array also relies on the form of its glyphs to provide orientation information, but the glyph form would take up nine times the space of our simple glyphs.

The DataGlyph code has better scaling properties than the M-Array code, but is still not suitable for very large surfaces. However, the code uses very simple single bit glyphs, instead providing orientation from the layout of the sequences in the code itself.

The Anoto code covers a very large area from a very small locating window size. Unfortunately, it requires large glyphs, and a redundancy in the cyclic bit sequence, to provide the orientation of the locating window. Like the M-Array, each glyph would take up nine times the space of our simple glyphs, leading to a very large locating window area.

A new proposed positioning code was described and its scaling properties explored. This new code draws upon the method used in the Anoto code to provide very large addressable areas from small windows. However, the method for determining the locating window's orientation from the code sequences alone, rather than the glyphs, is borrowed from the DataGlyph code. This combination yields a code which:

- Uses the simplest square glyphs.
- Has a very large addressable area which is easily capable of uniquely covering all ceilings in every city.
- Requires a reasonably small locating window area, which could easily be captured by camera.

The potential properties of an indoor location system, built upon this new proposed positioning code, were enumerated. The properties were placed in the taxonomy of Hightower & Borriello (2001), which serves as a useful classification for location systems used in the pervasive computing domain.

Chapter 7

Discussion and Conclusion

This chapter evaluates the proposed edge detection and coordinate formation algorithms in the context of our original hypothetical scenario: Can an indoor positioning infrastructure be formed from a ceiling painted with an amorphous computer?

The limitations of the experimental approach taken in investigating the algorithms are first acknowledged, followed by a discussion of the general properties and issues involved in an indoor positioning system of this form. The specific limitations of the algorithms are investigated together with some potential remedies.

The chapter then outlines a few potential directions for future work and concludes with a brief summary and a list of research contributions.

7.1 Critical Analysis

7.1.1 General Issues

There is an issue with regards to using a simulated model to capture the properties and behaviour of a future system whose embodiment is not yet known. Ideally, the model's behaviour would have been compared with an existing real world system's behaviour to give confidence to the model's validity. As this is not possible with today's hardware, an attempt was made to keep the number of assumptions down to a low number and not specific to any one particular hardware implementation. The assumptions are reasonable for and fit well with today's typical sensor network hardware; nevertheless, future hardware platforms could vary widely in their characteristics and potentially some platforms may not be suitable for the proposed algorithms.

7.1.1.1 Communications Method and Hop Based Ranging

A compromise was made with the communications model used. The most far reaching assumption made is that of a perfect circular communications range. This is clearly idealised, even with today's sensor network hardware, and any potential future hardware implementations may have very different properties.

This circular range is more suited to RF, ultrasound and potentially chemical or biological diffusion based communications hardware. Small variances or anisotropicity in the communications range will have an effect on the error in estimated ranges based on hop counts. The proposed algorithms will be tolerant of small differences and likely will degrade in a fashion similar to that seen by lowering the average node neighbourhood.

Other hardware methods such as directed light or randomly scattered communication fibre links may have a more unpredictable and greater variance in range and anisotropicity. If the communications with neighbours is arranged in a highly structured or orderly fashion (such as the Von Neumann neighbourhood in cellular automata), other ranging effects with respect to direction might become visible and the use of gradients or even standard multilateration algorithms may not be suitable. The effects of these more varied means of communication would need to be explored in further simulations.

The simulations have assumed a hop-based ranging, but more sophisticated hardware ranging support may be available, such as RSSI. In such cases the distance-vector based propagation of gradients still operate, but the range estimations can be refined with the hardware supported hop-distances.

7.1.1.2 Unreliable Nodes

Some communication methods may be unreliable. For example, in a shared communications medium, packets sent simultaneously from nearby nodes may collide with each other and not be received correctly. The simulations did not model this phenomena and any implementation must cope with the situation. Whilst acknowledgements could be sent in receipt of important messages, this would lead to an increase in the traffic and complexity of communications. The use of the HomePages for communications between neighbouring nodes has been shown to be tolerant of any occasional lost update (Butera 2002) because another regular update will follow a short random time afterwards.

Nodes may be destroyed, be entirely inoperable, or be only in operation intermittently. This behaviour can change the network topology significantly and was not modelled in the simulations. As they currently stand, the success of the algorithms over the entire network is not dependent on any one nodes' sustained availability; however, the loss of certain nodes will affect the local performance of the algorithms. For example, the loss of certain elected nodes during the execution of the edge detection algorithm may cause the algorithm to cease for that one cell region, though it may be the case that the cell is redundant and other cell regions cover the same area. The loss of an anchor node during the formation of the coordinate system would eventually result in the anchor gradient being removed and a reduction in the anchor density in that local area.

It is possible to build in greater robustness to failures of key nodes. Some potential remedies for each algorithm to cope in these situations will be mentioned later in this chapter. The adaptation of the algorithms to the changing topology of mobile nodes is a possible direction for future work and is outlined later in this chapter.

7.1.1.3 Node Neighbourhood Density

The average node neighbourhood has been assumed to be similar throughout the whole network. It may be the case, when nodes are suspended in paint, that differing qualities of the surfaces over which the paint is covered can cause the network density in these areas to be dissimilar. This could be due to either a change in the spacing of the nodes, or a change in the nodes' communications medium. For example, several coats of paint in one area can affect that area's node density; RF propagates in air differently to water (which may be important in the case of a painted bridge). These large changes in average network density in the same network have not been explored in the simulations and are an area for future research.

The expected distance travelled in a single hop is affected by such changes in node density. In localisation algorithms that use pre-configured anchor locations, there is an opportunity to adjust the measured distances between the anchors according to the known distances between the anchors. In our case, there are no pre-configured known locations and no global knowledge of the variance and distribution in node density. If the expected inter-hop distance is purely a function of the average node neighbourhood, there may be some promise in each node in the network estimating its local average and variance in numbers of neighbouring nodes. As gradients are propagated across the space in the network, the estimated range based on hop counts can be replaced by the accumulated inter-hop distance estimates of each forwarding node.

7.1.1.4 Size and Scale

The form of future node hardware could vary greatly in size and in the numbers of participating devices. The simulations have focused on node neighbourhood densities of around 15, as this seems to be a reasonable minimum for hop-based distance estimates. It could be the case that with future hardware, extremely dense neighbourhoods with hundreds or thousands of neighbours are normal. The algorithms in this thesis, without modification, would place an unreasonable demand on each individual node's resources in such large neighbourhoods. One potential solution could be to selectively ignore most of the neighbouring nodes, or to aggregate them into virtual nodes, in order to bring the neighbourhood size down.

The actual physical area covered by a neighbourhood will have an impact on the address codes. If the neighbourhood size is too large, then too few reliably formed glyphs can be packed into a reasonably sized area for capture by camera. If the neighbourhood area is very large, then the network scenario shown in figure 7.1 could represent that of an entire room. Such a case is less interesting to us as a single floating code window can span the width of the whole room, amounting to little more than a large barcode tagging the room. If the neighbourhood area is extremely small, the network scenario shown in figure 7.1 could represent just a square centimetre or less.

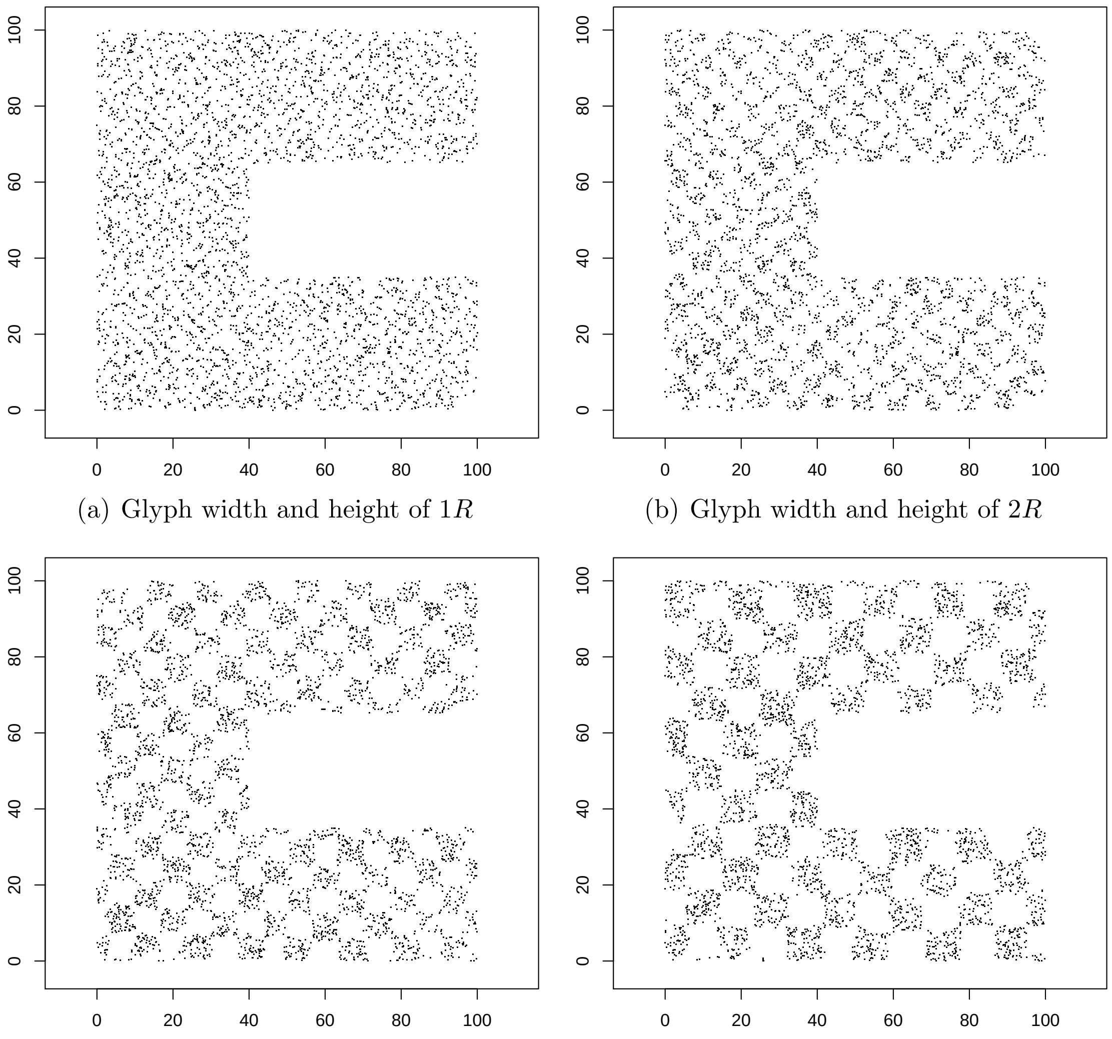
In the case of very small neighbourhoods, the whole ceiling would be covered by extremely large numbers of nodes, perhaps billions or more. Hardware constraints limited the simulations to 30000 nodes in a network scenario that reached roughly 250 hops in length. The simulations have shown the algorithms to operate successfully with 30000 nodes, but this would only cover a very small part of the ceiling when the neighbourhood area is very small. It is possible that there is a long range behaviour, not seen in these simulations but becomes evident in much larger networks involving far greater lengths in terms of hops. The small errors in multilateration whilst propagating the coordinate system from one point is accumulated as the coordinate system spreads – over 250 hops this error is shown to be easily manageable, but still larger simulations are needed to show that this scales well to much larger networks.

7.1.1.5 Glyph Sizes

Figure 7.1 shows the actual locations of nodes that are participating in a checkerboard glyph pattern based solely on their estimated locations. The physical size of the nodes involved may vary as might the extent to which ink is discharged from the nodes; nevertheless, the figure shows differing numbers of nodes involved in forming glyphs.

Figure 7.1(a) shows glyphs formed from nodes that believe they lie within a checkerboard square of one communication hop's width. In some locations, there are no nodes present to form part of the glyphs. As so few nodes are involved in forming the glyphs, little confidence can be had in whether a lone node is in fact a glyph or whether it has erroneously discharged its ink. With so few nodes comprising a glyph, it is difficult to estimate the centre of gravity of the glyph and the decoding of the address code is hindered. Figure 7.1(b) shows glyphs that are more likely to be identified correctly, but still lack definition and in some cases less than half of the expected glyph is covered by nodes.

As the glyph size is increased, in terms of the communication's radius, the glyphs become more clearly defined. Figure 7.1(c) shows glyphs three times larger than the communication's radius which are likely to be easily identified and decoded.



(a) Glyph width and height of $1R$

(b) Glyph width and height of $2R$

(c) Glyph width and height of $3R$

(d) Glyph width and height of $4R$

Figure 7.1: Different sized glyphs relative to the communication's radius, R. Formed using the estimated coordinates of 8176 nodes ($N = 15$ with a Poisson distribution)

With yet larger glyph sizes as in figure 7.1(d), they become well defined with a sharp edge and a good sense of the glyph's centre can be gained. At four times the communication's radius, the glyhps can still be identified even with small segments of them devoid of nodes (as is occasionally the case with the somewhat unforgiving Poisson distribution of nodes).

The node hardware may be such that the node neighbourhood areas are physically very large and produce very large glyph sizes. A possible remedy might be to have a heterogenous mixture of nodes suspended in the paint by mixing in much smaller and greatly simplified nodes. Or possibly to have a second coating of paint with a very dense mixture of small simplified nodes. The larger nodes could perform edge detection and coordinate formation whilst the miniature nodes could be used solely to mark the glyphs, using the locations from several of the nearest larger nodes.

7.1.1.6 Glyph Size Example

In machine vision, an optimistic number of pixels for recognising a simple dot would be 3 by 3 pixels, but a safer number is 10 by 10 pixels. For a 8 by 8 glyph window, a safe minimum captured image resolution is 80x80. To allow for different orientations, fixtures and unpainted areas on the ceiling, an image of a larger area could be captured to increase the chances of having at least one full code window available for decoding. An area three times this minimum size fits well into the typical low, yet quickly processed, mobile phone resolution setting of 320x240.

As a rough indication, an unmodified mobile phone camera with a field of view of 40°, held high at 1.5m from the ground, will capture a surface area of 0.5m^2 on a very low ceiling of 2.2m, and will capture a surface area of 1.45m^2 on a high ceiling of 3.5m. This captured window area, when the glyphs are roughly 5cm^2 in size, would mean that a captured image, at the low resolution of 320x240, would comfortably contain around 10 by 10 glyphs on a very low ceiling and 29 by 29 glyphs on a high ceiling.

7.1.1.7 Glyph Colour and Appearance

There is a requirement that the glyph markings have sufficient contrast with the background of the surface in order to easily differentiate the individual glyphs. Complementary colours provide the most contrast and white next to black would give an optimum contrast between two neighbouring areas, but this is likely to result in the ceiling taking on a distracting appearance.

The glyph markings need not be in the visual spectrum as most CMOS cameras have an exceptional sensitivity to near-infrared and a good sensitivity to near-UV light. Normally these camera's spectral response is modified by filters to exclude this light, but it is equally possible to fit the camera with visible-blocking and IR/UV pass filters to block all but the non-visible light. A visual positioning system using glyphs which absorb or reflect this non-visible light would be visually unobtrusive and could even continue to operate in the dark.

7.1.1.8 Non-Planar Ceilings

The simulations have assumed that the ceiling itself is flat and all nodes are coplanar. This is not always the case and many ceilings have joists, fixtures and inclinations, which can introduce some new issues.

A small effect could arise in areas where the painted surface covers corners and edges. Certain types of node communication hardware could have problems here. For example, with RF, a tight corner could cause the spherical range of the communications hardware to allow links between nodes that are physically close to each other in three dimensions, but more distant when following the surface itself. This could cause a change in the node density where corners exist.

A more concerning problem is the disturbance of the coordinate system. The actual hop distance across the bent surface of a fixture is greater than the hop distance across the surface beside the fixture, across a flat surface. This discrepancy, due to the unaccounted third z dimension, will cause a distortion in the coordinate system around features such as fixtures.

Still more serious is the perceived address code as seen by the camera in areas where large fixtures and inclinations exist. Inclinations will cast a perspective on the captured image. If the localising mobile device has some hardware inclinometer to deduce its camera's angle to the ceiling, then the perspective can be corrected and known to be due to an inclined ceiling. If the localising device uses the perspective of the captured image to deduce the camera's angle to the ceiling through software, then the calculated angle will be incorrect and the deduced location incorrect. Large fixtures and joists will cause parts of the address code to be hidden from the camera, complicating or perhaps even prohibiting decoding of the location in that area.

7.1.1.9 Completely Disjoint Ceiling Areas

Each ceiling's formed coordinate system and their decoded addresses will require an arbitrary transformation to conform to some external coordinate system. For example, each separate ceiling in a building will have created its own coordinates, unrelated to any other ceiling. Each of these coordinate systems may be transformed into a shared coordinate system as might be used in a pervasive application spanning a whole building. This puts some demand on the configuration requirements of pervasive applications where such a shared coordinate system must be used.

Each separate covering must be started with an initial starting position in the address space that is sufficiently distant from all others. This ensures they will not overlap each other and do not encode ambiguous locations, but requires a management of the address space used over the entire set of painted areas. When the code window size of 10x10 glyphs is used, the address space is extremely large (see section 6.3.1.5 in chapter 6). It is quite possible that randomly chosen starting positions in the address space will be sufficient for many applications.

The transformation of the formed coordinate system into the shared coordinate system used by the application must be learned after deployment. This calibration of the application adds only a little to the configuration requirements of the system, but it can be exacerbated when a single ceiling does not have a single coating of paint covering it.

A relatively simple process of taking location readings from three measured points on each separate surface can be performed after the address code has been formed. Alternatively, three nodes can be configured with their absolute coordinates and carefully placed at measured points on the ceiling before the coordinate system is formed.

7.1.2 Specific Issues

This section looks at the specific issues involved in both the edge detection algorithm and the coordinate propagation algorithm. Each issue is accompanied with some potential avenues for improvements and future research.

7.1.2.1 Edge Detection: Cell Leader Redundancy

The proposed perimeter detection algorithm relies on a sufficiently high number of cells in order to cover and detect all of the network perimeter. There is a large level of redundancy involved as many of the cells will have broken membranes along the same edge of the network perimeter. Many cells will serve little purpose other than to establish that there are no edges in their membrane regions. Additionally, even with fairly high densities of cell leaders, there are still some small sections of the network perimeter that are not detected and covered by the membrane regrowth.

In its current form, the cell leader election of the perimeter detection algorithm will typically result in around $\frac{1}{N}$ of all nodes as cell leaders (where N is the average node neighbourhood). This is because in each local neighbourhood there will be exactly one cell leader, the node with the highest ID. A slower, but more powerful approach to achieving a regular packing of cell leaders is to inject mutually repellant mobile code particles into the network. The particles can each radiate a gradient whose intensity acts as a repellant force to steer away other nearby mobile code particles. The particles can then migrate on to nodes further away from other mobile particles. More mobile particles can be injected at random intervals until the required density is achieved. This is similar to a gaseous dispersion technique described in more detail by Butera in (Butera 2002).

In the scenario of a painted ceiling, there is no benefit from reducing computation and communication in selective areas of the system. However, in other uses of the algorithm, such as in large sensor networks, other algorithms may executed on the nodes continually, and it may be important to conserve as much battery life as possible. A reduction in the number of cells in some areas will save battery life by reducing communications and computation.

The number of initial cell leaders can be reduced, if their density and arrangement remains such that the network edges of interest, with a high probability, are all detected by at least one cell. Once an edge is detected, additional cell leaders could be elected in the vicinity of the perforated cell. Good candidates for these new cell leaders could be the perforated cell's two membrane endpoint nodes that lie at the extremes of the newly discovered perimeter. This will provide a cell density and subsequent communication and processing cost that is concentrated at the local points of interest, along the perimeter of the network. This adaptive cell placement would also ensures that all of the network perimeter will eventually be covered, leaving no undetected patches. A drawback with this approach is that the convergence time for the whole process becomes unknown. The convergence time would be dependent upon the length of the network edges and the number of cells that detected the edges.

7.1.2.2 Coordinate Formation: Filtering Anchors and Anchor Density

One of the most serious sources of localisation error arises where insufficient numbers of anchors are available for multilateration. The available anchors are reduced by filtering out those whose gradients have passed edge nodes (the shadowed gradients). In narrow areas or areas involving many obstacles, far fewer anchors than normal are available as a result of this filtering. To overcome this requires a more sophisticated method of selecting the anchor density than the current simplistic approach of enforcing a minimum anchor distance. If it can be observed during the coordinate propagation that the multilateration accuracy is becoming compromised by too much filtering, the anchor density could be increased further in that area to compensate.

Other research has explored a similar problem in adapting the anchor density through the selective disabling of anchors in networks populated very densely with pre-configured anchors (Bulusu et al. 2001, Tarnacha & Porta 2003). This might prove a useful starting point for further research.

Another potential solution to this could be to monitor new potential anchors' GDOP metric and compare this to the average GDOP of previous anchors. Spikes can be seen in the new anchors' average GDOP metric (as in figure 5.14 of chapter 5) as the coordinate system propagates. These spikes show that some newly elected anchors have far greater GDOP values than other recent nearby anchors. When the filtering of shadowed gradients is enabled, there are far more spikes and local rises seen in this figure which seems to indicate that relatively bad anchors are forced to be chosen as a result of the acceptable minimum distance between existing anchors.

The minimum distance between anchors is used as a mechanism to ensure that the propagation of the coordinate system makes some reasonable progress. In narrow areas between edges and obstructions, it is more of a hindrance and hazard to the location estimation accuracy. If the GDOP values of potential new anchors is seen to be higher and deviate greatly from other nearby recently elected anchors, the enforced minimum distance could be relaxed and a more dense distribution of anchors allowed in these narrow areas.

7.1.2.3 Coordinate Formation: Error Accumulation

As the coordinate system propagates further, the error involved in new anchor's location estimates will have accumulated from all the previous anchors' estimation error. With a good model of the error involved in hop-based ranging measurements, this error can be estimated and tracked as the coordinate system spreads.

Measurements of error can be used in some optimisation algorithms to allow ranging measurements with larger error to have less influence in the location estimation (Liu et al. 2006). Already, the edge count of shadowed gradients can serve as a indication of the degree of error involved in the hop-based range measurement. Rather than the proposed wasteful approach of eliminating all shadowed gradients regardless, they could be included into the location estimation and potentially increase the estimation accuracy.

7.1.2.4 Coordinate Formation: Overlapping of Propagated Coordinates

A propagated coordinate system over great distances may accumulate error and slowly become considerably bent. One potential unfortunate situation may arise from this where the estimated coordinates on the same surface begin to coincide at physically different points on the surface. Two extremely long parallel and close corridors could result in this situation if the coordinate system bends considerably during propagation. Different physical locations would be ambiguous in the resulting estimated coordinate system.

A potential solution to this situation could be to map the propagated coordinate system and discover any locations that correspond to two different physical locations. In cases where the coordinate systems does overlap, the localising device could disambiguate the location with the use of this map and a history of recent movements within the map.

7.1.2.5 Coordinate Formation: Non-Robust Optimisation Algorithms

Where the coordinate system has propagated separately over a long distance and then rejoins to itself, there may be a large gulf between the estimates due to the accumulated error over the distance. If this gulf is too large, there may be problems with some optimisation algorithms not being robust to the large differences in ranging measurements and anchor locations.

Even if the optimisation algorithm is robust, the coordinate system could become distorted at the point where they meet. This distortion could be evened out by refining and updating existing anchors' estimated locations when they receive new or updated anchor gradients. These refinements would propagate out into the coordinate system until the distortion has been corrected.

One potential alternative to using refinement updates is to force a logical barrier at the point of the join. This would effectively split the coordinate system into two separately spreading coordinate systems in this area, each not being allowed to spread into the other's area.

A logical boundary eliminates the problem of the robustness of the optimisation algorithm, because the election of new anchors would disallow the use of anchor gradients from the neighbouring coordinate system. However, there will continue to be a gulf between the locations of the two propagating coordinate systems; no attempt is made to consolidate and rejoin the two systems, which could lead to the systems diverging further[1]. Additionally, the logical boundary acts as an artificial perimeter, which has a detrimental effect on the location estimates of anchors nearby to it because of the reduced number of anchors available to it. Also, there may be an added complication where several joins meet, leading to a very fragmented coordinate system.

Another alternative could be to form a third coordinate system at the point of the logical barrier, overlaying the two split coordinate systems. This could propagate around the logical barrier and back to the earlier parts from which the split coordinate systems originated from. This third coordinate system could potentially act as a bridge between the two split systems, after aligning itself with both of them, and allow the reconsolidation of the two systems into one. The downside of this is the clear cost of memory, time, further communications, and complexity of code.

7.2 Future Work

This section describes a number of interesting directions for future research. Some directions look at the algorithms operating in different networking environments, such as mobile networks, three dimensional networks, and networks with highly variable densities. Other directions look at variations and alternatives to the algorithms, which make them useful for different applications than have been covered in this thesis.

7.2.1 Mobility

The nodes in the simulations have been assumed to be static. The scope for mobility of nodes in the specific application presented in this thesis, that of painted ceilings, is remote. Mobility is only likely to occur whenever a shearing or other damaging movement of the surface occurs, such as during an earthquake. Nevertheless, this type of mobility may be of interest in other application areas involving longer running, active, surfaces. For example, the structural integrity of bridges or buildings could be monitored, allowing the location and area of any damage to be reported.

In a more general sense, mobility is possible in any environment where the nodes are not guaranteed to stay still, or the nodes are by their nature mobile.

[1]This divergence could nevertheless be alleviated by retrospectively mapping the formed coordinate system, as mentioned in section 7.1.2.4.

Sensor networks are deployed in a great variety of different environments, including both static and mobile environments, in which nodes may be mobile. In static environments, the nodes may be shifted or disturbed by the wind, or scuffed by wildlife. Nodes may be deployed in environments which have an aspect of mobility, such as sensor shifting sand dunes, erupting geysers, at the bottom of the sea, or in rivers. In all of these environments, it can be critical to the sensor network's application to understand the new locations of the nodes, and to understand the new coverage and perimeters of the sensor network.

The nodes themselves, rather than the deployment environment, might be mobile. For example there can be applications for knowing the perimeters and locations of nodes within large mobile ad hoc networks, such as large groups of mobile robots, or even in large crowds of people.

In a mobile network the topology can be constantly changing and some spatial structures, such as regions formed from gradients, will distort over time. To retain the integrity of these structures, they must be continually maintained with a vigour matching that of the speed of topological change.

A fundamental building block used in the proposed algorithms is the gradient. This amorphous primitive has been shown in previous research to be adaptive to changes in the network topology (Clement & Nagpal 2003, Nagpal 2003). Gradients can be continually maintained through regular updates of each node's HomePage. A node can periodically check through its gradients and remove old gradients that have not recently been updated. This housekeeping allows a node emitting a gradient to move through the network and for the gradient's emissions to follow it.

7.2.1.1 Edge Detection

In the edge detection algorithm, the regions of interest are the cell and its membrane. These two regions are formed from the same gradient emitted from one node, the cell leader. As nodes are mobile, the new members will join these regions and old members will be removed as a natural consequence of the gradient's regular maintenance.

The proposed algorithms have assumed a static network whose edges need be detected only once. In a mobile network, a continual maintenance of the edge nodes are needed. The edge nodes are detected from the path taken by a mobile token from one membrane endpoint node to the corresponding membrane endpoint on the other side of the break. In a mobile environment, this greedy forwarding based token passing should be made with sufficient frequency for the degree of network mobility involved. A decay and eventual removal of the edge nodes should occur after some time, as is the case with the housekeeping required of mobile gradients. Edge nodes that have not been flagged as edges for a long time should relinquish their status as an edge node.

Endpoint nodes are members of the cell's membrane region, which itself may move and change to the point where the current endpoints are no longer members. Also, as the cell regions move in the network, a previously detected break in one of the membrane segments could be filled with nodes, leading to a fully formed unperforated cell and the absence of endpoint nodes.

Membrane leaders and satellites both need to be persistent entities within the membrane. The elected nodes may cease to lie within the membrane region, so possibly a handover to another nearby node that does lie on the region could be performed. Perhaps, a general region of nodes could be upheld as the entity itself in a similar fashion to the persistent nodes described by Beal (2003).

Some care should also be taken over the density of cells. The network's mobility may be such that the cells become sparse in some areas and cease to cover the edges of the network. A possible remedy to this could be for the cells to exert gradients that attract and repel other cells' migration movements, as described earlier in section 7.1.2.1.

7.2.1.2 Coordinate Formation

The estimated location of the moving anchors can drift over time. With continually maintained anchor gradients, each node will need to re-estimate its location when the ranging measurements have changed. The resource cost of this in terms of power and computation could be high, so it could be beneficial to introduce a lethargy in these estimation updates, proportional to the degree of network mobility. Each node could keep track of the hop counts and anchor locations as they were at the time of each update. If a node's hop counts to anchor nodes move beyond acceptable limits, the location could be re-estimated. Similarly, anchor nodes could update their anchor gradients only if their location estimate changes beyond some significant margin. This lethargy effectively acts as a hysteresis to avoid unnecessary and potentially problematic location updates.

Anchors within the network have been elected according to their favourable GDOP metric. As the anchors move through the network, this metric will change and in some cases will become unfavourable. In areas where the average GDOP metric is low, more anchors could be elected in the best available spots to bring the average GDOP metric up.

Similarly, some areas may accumulate a dense population of anchors over time and it might prove profitable in terms of resource consumption to lower the density by selectively removing some anchors, as discussed in section 7.1.2.2.

7.2.2 Three Dimensions

Nodes may be deployed within three dimensions. With some tweaks, the proposed algorithms can also operate in three dimensions.

Nodes could be deployed within concrete or some solid structure, in which it is useful to know the location and perimeters of the structure. For example, an application such as damage awareness could locate and evaluate the extent of damage to solid structures.

Nodes could also be deployed within a body of water, such as the sea bottom or in rivers. Nodes could be deployed in gel or foam. Nodes suspended in gel or foam, for example, could be flushed down a tube or some otherwise inaccessible area, and the algorithms could help to discover the area's shape and the location and nature of obstructions or blockages.

The idea of gradients propagating over neighbouring nodes on a surface, in increasing concentric circles, naturally extends to three dimensions. In three dimensions, each node's communication radius is replaced with a communication sphere. Gradients propagate through three dimensional space, with each neighbouring node forwarding the gradient to their neighbours, as in the two-dimensional case. Increasing concentric spheres of increasing hop counts appear in three dimensions. The tweaks for the two proposed algorithms to handle this case are outlined below.

7.2.2.1 Edge Detection

The cell leader can propagate a gradient as normal, resulting in a spherical cell region with the outermost shell becoming the membrane. The problem of detecting edges remains similar to the two-dimensional case – that of finding a perforation in the cell membrane. Rather than just one ring circling the cell through the membrane, several rings could circle the cell through the spherical membrane. If there are breaks in any of these rings, then an edge has been detected. The steps that might be taken to form rings around a sphere might be as follows:

1. Elect, or select one node on the spherical membrane, preferably on the outermost part of it. This node acts as an antipodal point, or polar node, for the sphere.
2. Send a gradient from this pole node throughout the spherical membrane.
3. Elect, or select one node on the far side of the sphere with the greatest hop count from the polar node. This node acts as the opposite antipodal point on the sphere.
4. Send a gradient from this second, opposite, antipodal node throughout the spherical membrane.
5. All the nodes roughly equidistant in hops between the two poles, collectively form an "equatorial" great circle, or ring, around the sphere.
6. From this point, another great circle could be formed in a similar way by electing two nodes on the opposite sides of the equatorial ring. These two equidistant nodes could then emit two gradients, throughout the spherical

membrane, to use for forming a “longitudinal” great circle, perpendicular to the equatorial great circle.

7. The two points at which the equatorial and longitudinal rings meet can be used as a further two antipode nodes of another great circle. This third great circle would be roughly perpendicular to both the first longitudinal great circle and the equatorial great circle.
8. The surface of the sphere is now divided into 8 triangles. Further divisions could be made, if wanted, by forming smaller latitudinal and longitudinal rings. For example, a small ring could be formed on the surface where the hop count is around 25% (of the shortest path between the two antipodal nodes) from one antipode and 75% from the opposite antipode.

The reparation of the membrane and the flagging of nodes as edges is more tricky because a whole surface of nodes closest to the edge must be found, rather than just a line of nodes as in the two dimensional version. If there are enough of these rings, perhaps running along different lines of latitude and longitude, then any reparations of the broken rings would result in a good number of edge nodes across the surface being detected.

It is possible that a close knit mesh could be built up across the membrane. Where the perforations exist, a straight regrowth of the mesh could be made to the other side of the break in the mesh. A midpoint node on the far side of the membrane could be elected as the node which is the furthest away from all of the mesh endpoints. To account for concave edges perforating into the cell, this midpoint node can send a gradient throughout the cell. This gradient can act as a repellant force urging the regrown mesh to be pushed further towards the actual edge, in a similar way that an elastic balloon in a confined space might be inflated.

7.2.2.2 Coordinate Formation

Little needs to be changed for the coordinate propagation algorithm to be extended to three dimensions. Anchor gradients would provide hop counts in increasing spheres. Instead of the minimum of three anchors in two dimensions, we require four or more different anchor gradients in order for multilateration to be possible in three dimensions. The calculation of the GDOP metric is performed similarly to the two dimensional case, but is now more akin to that used in the well known GPS system.

7.2.3 Density Variations

The ranging measurements used in the simulations have been based purely on integral hop counts. There is an advantage to using Takagi & Kleinrock’s (1984) formula for estimating the expected inter-node distance based on the average node neighbourhood when this is below 15 (Nagpal et al. 2003). Similarly, when the node degree is greater than 15, an advantage is seen from a smoothing of the hop counts (Nagpal et al. 2003). Smoothing involves nodes exchanging their hop counts with neighbours and taking the average of the values. After a few exchanges, the smoothed hop counts provide a greater degree of precision than the integral hop counts.

If the average numbers of nodes in a neighbourhood varies across the network, applying these two techniques can introduce further and considerable error. However, as mentioned briefly in section 7.1.1.3, the expected inter-node distances according to the current perceived neighbourhood density can be accumulated in the forwarded gradients. This could result in improvements in the ranging accuracy when node densities change or when node's communication ranges change.

7.2.4 Faster Convergence of Coordinate System

The coordinate formation algorithm requires time proportional to the size of the network for the coordinates to completely cover the whole network – this type of algorithm is classed as an iterative localisation algorithm. Concurrent algorithms, on the other hand, such as distributed MDS (Ji & Zha 2004), do not require the spreading of anchors across the network and rely instead on the merging of small neighbouring coordinate systems that were all formed locally at the same time throughout the network.

Concurrent localisation algorithms show great promise and construct a coordinate system faster than iterative methods; however, there are risks involved in incorrect merges of neighbouring coordinate systems and there are still difficulties with concave networks. An interesting research avenue would be to introduce the edge detection algorithm to such localisation schemes and to investigate the use of the GDOP metric as a guard against incorrect merging of neighbouring coordinate systems.

7.2.5 Pattern Repair by Watching Paint Dry

If substantial damage is incurred by the pattern then the address space could contain dead spots where no decoding is possible. Maintenance of the pattern could be accomplished by covering the damaged area with another coat of paint and configuring at least 3 special anchor points at the edges of the damaged area. These configured points would have locations corresponding to the underlying dead spot's coordinates and would be used when forming the new coordinate system. After the coordinate system spreads, the pattern would be recreated with the same address space.

A pattern could be formed badly due to some environmental effects during its creation or perhaps due to an unusually sparse density of nodes. In this thesis, a node's power has been assumed to be depleted and any ink permanently marked after the pattern has been deployed. This might not be the case and nodes could be re-invigorated with an external power source such as light from a torch, or heat from a hairdryer. In the case of a bad glyph formation, the node density could also be topped up by simply painting on new nodes and then reinvigorating all the nodes in that area with a torch or hairdryer, allowing the new nodes to negotiate with the old to complete the glyph form.

Some care must be taken for old nodes reappearing in the network and for new nodes being added. If the power source is local only to a general area, then gradients from unpowered anchor nodes must be handled gracefully in the presence of unpowered forwarding nodes. If the specified corrective reference points for the repaired surface area are significantly different to the surrounding area, then the surrounding area must also be re-invigorated to allow it to adjust to the changes and the new anchors.

7.2.6 On Demand Edge Detection

The membrane regrowth edge detection algorithm described in section 3.2 of chapter 3 can be modified to provide a more on-demand detection of edges. This modified version enables any two nodes to discover if the shortest path between them is influenced by a network edge or obstacle without the use of shadowed gradients. This approach may be favourable in some situations, such as when these hop-based distance measurements are only rarely needed or possibly when high node mobility means the standard algorithm incurs a large continual overhead to maintain complete and accurate edge information.

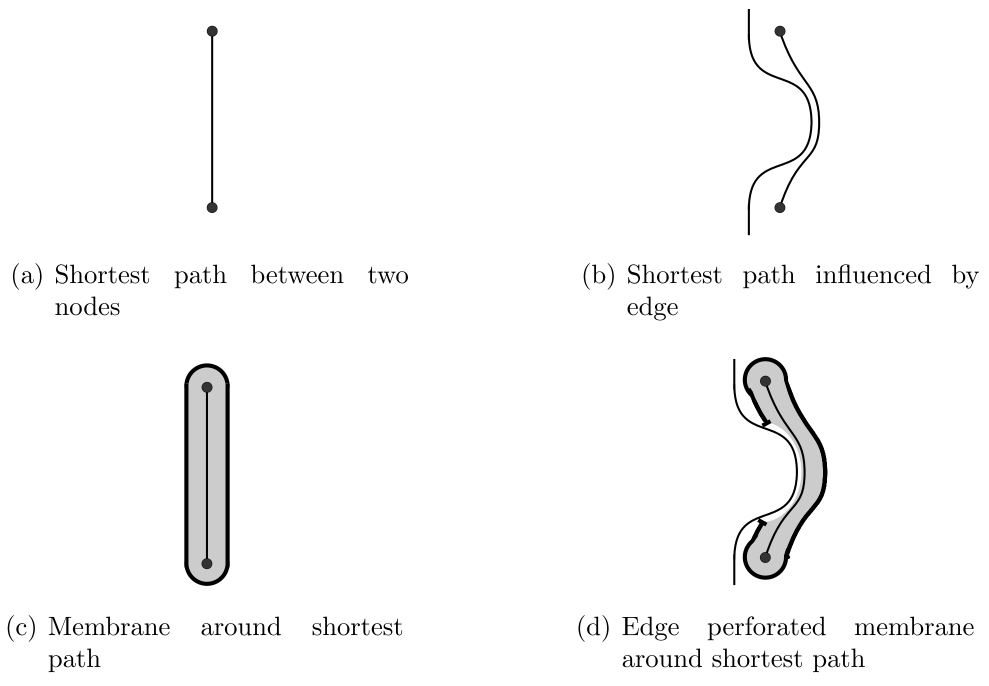

(a) Shortest path between two nodes
(b) Shortest path influenced by edge
(c) Membrane around shortest path
(d) Edge perforated membrane around shortest path

Figure 7.2: Modified edge detection algorithm

Figure 7.2(a) shows the shortest path between two nodes, obtained in the normal fashion. The shortest path shown in figure 7.2(b) is influenced by the network edge and would be longer than expected. Without any knowledge of location or of the network edges, it can not be known if the shortest path is reflective of the actual distance between the two nodes and no discrimination can be made between the case in 7.2(a) and that in figure 7.2(b).

If each node along the shortest path participated as one of multiple sources in a single gradient, a tube like gradient is achieved. The outer nodes of this tube can form a membrane or halo around the shortest path. As with the unmodified version of the edge detection algorithm, if this membrane contains a complete cycle around the membrane then there is no network edge nearby (see figure 7.2(c)). If the membrane is broken and the shortest path hugs an edge or obstacle in the network, as shown in figure 7.2(d), then no cycle will be found. With a disconnected membrane, the shortest path can not be trusted to be straight and accurate enough for ranging measurements.

7.2.7 Alternative Edge Detection

There may be other alternative methods for detecting network edges without location knowledge. One method could be to simulate waves over the space in the amorphous computer and apply signal processing to detect edges. An energy saving model has previously been investigated by Rauch (2003) and exhibits all the main properties of waves, including reflection. Using the same principle that a radar uses, edges might be detected by nodes emitting waves and listening for their echoes. However, such an approach would be very expensive in both computation and communication.

7.3 Brief Comparison of Infrastructure with Existing Systems

The Smart Floor (Orr & Abowd 2000) and ORL Active Floor (Addlesee 1997) both similarly provide location information through an infrastructure that covers an entire room's surface such as a floor. Like these and other systems, the painted infrastructure is unobtrusive and need not even be perceptible.

Like the infrared Active Badges (Ward et al. 1997) and other visual systems, the positioning is susceptible to degradation caused by extreme lighting conditions and visual occlusion.

Unlike many existing indoor location systems, there is no RF or ultrasonic communication between the infrastructure and the localising mobile devices. This avoids any problems with interference and any physical limits on the numbers of localising devices in any area.

As in the Cricket system (Priyantha et al. 2000, Priyantha 2005), the location computation is local to the mobile devices. This supports privacy of location and also results in a decentralised system where the infrastructure has no computational or administrative bottlenecks. Other systems, such as Active Bats (Ward et al. 1997), use the infrastructure to compute the location and are susceptible to these problems.

The proposed address code allows a single mobile device to precisely reproduce orientation information. Many of the existing systems derive the orientation through less than reliable means. Systems can employ a second order orientation derived through a recent history of movement (as might be the case with a typical GPS receiver) but these do not detect changes in orientation when standing still. Some systems, such as Active Bats (Addlesee et al. 2001), use two mobile localising devices fixed to each other, yielding a reasonable orientation accuracy related to the system's location accuracy; however, it is potentially expensive. Some systems have gleaned some orientation information through the relative RSSI measurements from an array of receivers, such as (Priyantha 2005), but this can be prone to error.

It is often the case that the indoor positioning infrastructures are very costly to either install or maintain, with some exceptions such as the Pinger (Hull et al. 1997*b*) and Cricket systems. Whilst the future is an unknown quantity, an amorphous computer looks to have the potential to be very cheap and the installation of the infrastructure using a paintable amorphous computer involves little more than a simple DIY project. Once deployed, the infrastructure is fully passive and requires no maintenance or power. Mobile camera phones are ideal localising devices after some software installation, and they are often already available to most people. This can eliminate much of the hardware cost involved.

7.4 Summary and Conclusion

This thesis has followed the problem of creating an indoor positioning infrastructure, suitable for pervasive computing applications, formed using an amorphous computer. The amorphous computer is comprised of many small computing devices, capable of communicating locally with each other, suspended within paint and covered over a ceiling.

The important and desirable characteristics for a positioning system for pervasive applications include:

- Ability to operate indoors
- Low cost: of infrastructure, installation and maintenance
- Give accurate and precise locations
- Provides orientation
- Able to cover a large area and handle many people simultaneously
- Retain user privacy
- Using a transparent, familiar or easy to use technology for the user

The form of the indoor positioning infrastructure targeted in this thesis and which aims to achieve these characteristics, is a machine visible patterned surface that encodes the positioning information. Part of this pattern covering a ceiling can be captured by low-powered mobile devices equipped with a suitable camera, such as the now ubiquitous mobile camera phones.

Several existing patterns providing addressing information from the domain of digital pens have been evaluated, all of which are formed using a small set of very simple glyphs. An assumption is made that the glyphs created from the computing devices on an amorphous computer will be prone to ill defined forms. This assumption motivates the pursuit of a pattern which uses only the most minimal of glyph forms. Based on techniques from existing patterns, a new address code pattern was proposed, which is more suited to encoding both position and orientation information using only the simplest square glyphs.

The positions encoded by the address code are taken from a reference coordinate system. This coordinate system is propagated across the amorphous computer from a single point, providing each node with hop-based distance ranges relative to each other. This thesis has proposed a new sensor network localisation algorithm which uses knowledge of the edges of the network, including large obstructions, to reduce the hop-based ranging error resulting from no clear line-of-sight between ranging nodes. The calculated geometric dilution of precision of reference points in the coordinate system is used as a measurement of fitness when introducing new reference points.

Distance vector gradients are the mechanism used to make hop-based ranging measurements. The clear line-of-sight requirement in these measurements is enforced by filtering out any gradients which are known to have passed through nodes flagged as being on the edge of the network. A new algorithm was described in this thesis that can detect obstacles and edges of the network without requiring any location knowledge.

The main research contributions of this thesis are:

- A new distributed edge detection algorithm that requires no location knowledge.
- A distributed localisation algorithm that requires no configuration, predetermined anchors or location knowledge at all. Coordinates are propagated by selecting geometrically favourable nodes as new anchors.
- Localisation within dense concave networks by identifying and avoiding the hop-based ranging error introduced by the network perimeter and obstacles. This allows localisation in networks with convex perimeters and networks with significant obstacles where the clear line-of-sight between nodes is interrupted.

- An outline of a new spatial addressing code that may be used in an indoor positioning system. This new code is based on existing spatial codes, but is designed to be more suitable for use with the low precision marks as might be formed by an amorphous computer.

In the area of sensor networks, the problem of having no line-of-sight in ranging measurements is one of the most limiting and difficult to overcome. This thesis has shown a remedy for this problem in the case of hop-based ranging measurements, where previously there has been no method for either detecting the error or for correcting the error without substantial prior manual configuration.

In the area of amorphous computing, the augmented gradients with accumulated edge proximity knowledge add a further degree of awareness of the network's geometry. It has been shown in this thesis that the shape of the deployed network has the potential to affect the geometric algorithms running within it. Whilst, a topological robustness of spatial structures has already been demonstrated on an amorphous medium in the face of node and region failures (Coore 1999, Nagpal 2001), this thesis has demonstrated it is also possible to attain some level of geometric robustness. Through a rudimentary geometrical awareness, a geometrical robustness can be achieved not just in the case of a previously functioning area of nodes failing, but also in the starting case where the space occupied by nodes is initially unknown and irregular.

The use of these edge detection algorithms can extend beyond this thesis' focus on maintaining a geometric robustness for localisation. Specifically, they could be a useful aid in obstacle avoidance algorithms in ad hoc routing protocols. This would help avoid the build up of congestion around obstacles or the network perimeter, particularly in geographic forwarding schemes.

Bibliography

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Knight, T. F., Nagpal, R., Rauch, E., Sussman, G. J. & Weiss, R. (2000), 'Amorphous computing', *Communication of the ACM* **43**(2), 74–82.

Addlesee, M. (1997), 'ORL Active Floor', *IEEE Personal Communications* **4**(5), 35–41.

Addlesee, M., Curwen, R., Hodges, S., Newman, J., Steggles, P., Ward, A. & Hopper, A. (2001), 'Implementing a sentient computing system', *IEEE Computer* **34**(8), 50–56.

Ailamaki, A., Faloutsos, C., Fischbeck, P. S., Small, M. J. & VanBriesen, J. (2003), 'An environmental sensor network to determine drinking water quality and security', *SIGMOD Record* **32**(4).

Akyildiz, I., Su, W., Sankarasubramaniam, Y. & Cayirci, E. (2002), 'Wireless sensor networks: A survey', *Computer Networks* **38**(4), 393–422.

Antifakos, S., Michahelles, F. & Schiele, B. (2002), Proactive instructions for furniture assembly, *in* 'Ubicomp 2002: Ubiquitous Computing'.

Bachrach, J., Nagpal, R., Salib, M. & Shrobe, H. (2004), 'Experimental results and theoretical analysis of a self-organizing global coordinate system for ad hoc sensor networks', *Telecommunication Systems Journal* **26**(2–4), 213–233.

Bahl, P. & Padmanabhan, V. N. (2000), RADAR: An in-building RF-based user location and tracking system, *in* 'INFOCOM (2)', pp. 775–784.

Beal, J. (2003), Persistent nodes for reliable memory in geographically local networks, Technical Report MIT AI Memo 2003-011, MIT.

Beal, J. (2005), Amorphous medium language, *in* 'Large-Scale Multi-Agent Systems Workshop at AAMAS 2005'.

Beal, J. & Bachrach, J. (2006), 'Infrastructure for engineered emergence on sensor/actuator networks', *IEEE Intelligent Systems* **21**(2), 10–19.

Beal, J. & Gilbert, S. (2004), Rambonodes for the metropolitan ad hoc network, *in* 'International Conference on Dependable Systems and Networks'.

Beal, J. & Sussman, G. (2005), Biologically-inspired robust spatial programming, Technical Report MIT AI Memo 2005-001, MIT.

Berlekamp, E. R. (1968), *Algebraic Coding Theory*, New York: McGraw-Hill.

Bose, P., Morin, P., Stojmenović, I. & Urrutia, J. (2001), 'Routing with guaranteed delivery in ad hoc wireless networks', *Wireless Networks* **7**(6), 609–616.

Bren, A. & Eisenbach, M. (2000), 'How signals are heard during bacterial chemotaxis: Protein-protein interactions in sensory propagation', *Journal of Bacteriology* **182**(24), 6865–6873.

Brumitt, B., Meyers, B., Krumm, J., Kern, A. & Shafer, S. (2000), EasyLiving: Technologies for intelligent environments, *in* 'Handheld and Ubiquitous Computing, 2nd International Symposium', pp. 12–27.

Bulusu, N., Heidemann, J. & Estrin, D. (2000), 'GPS-less low cost outdoor localization for very small devices', *IEEE Personal Communications Magazine* **7**(5), 28–34.

Bulusu, N., Heidemann, J. & Estrin, D. (2001), Adaptive beacon placement, *in* 'IEEE 21st International Conference on Distributed Computing Systems', pp. 489–498.

Butera, W. (2002), Programming a Paintable Computer, PhD thesis, MIT Media Lab.

Cheng, K.-Y., Tam, V. & Lui, K.-S. (2005), 'Improving APS with Anchor Selection in Anisotropic Sensor Networks', *ICAS-ICNS* **0**, 49.

Chintalapudi, K. & Govindan, R. (11 May 2003), Localized edge detection in sensor fields, *in* 'Proceedings of the First IEEE ICC Workshop on Sensor Network Protocols and Applications', IEEE Computer Society, pp. 59 – 70.

Cho, Y. & Neumann, U. (1998), Multi-ring color fiducial systems for scalable fiducial tracking augmented reality, *in* 'Virtual Reality Annual International Symposium', p. 212.

Clement, L. & Nagpal, R. (2003), Self-assembly and self-repairing topologies, *in* 'Multi-Agent Systems, RoboCup Australian Open'.

Cleveland, W. S. (1979), 'Robust locally weighted regression and smoothing scatterplots', *Journal of the American Statististical Association* **74**, 829–836.

Coore, D. (1999), Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer, PhD thesis, MIT Department of Electrical Engineering and Computer Science.

Coughlan, J., Manduchi, R. & Shen, H. (2006), Cell phone-based wayfinding for the visually impaired, *in* '1st International Workshop on Mobile Vision, in conjunction with 9th European Conference on Computer Vision'.

Cowie, J., Liu, H., Liu, J., Nicol, D. & Ogielski, A. (1999), Towards realistic million-node internet simulations, *in* '1999 International Conference on Parallel and Distributed Processing Techniques and Applications'.

Cowie, J., Nicol, D. M. & Ogielski, A. T. (1999), 'Modeling the global internet', *Computing in Science and Engineering* **1**(1), 42–50.

de Ipia, D. L., Mendona, P. R. S. & Hopper, A. (2002), 'TRIP: A low-cost vision-based location system for ubiquitous computing', *Personal and Ubiquitous Computing* **6**(3), 206–219.

Dellaert, F., Burgard, W., Fox, D. & Thrun, S. (1999), Using the condensation algorithm for robust, vision-based mobile robot localization, *in* 'IEEE Computer Society Conference on Computer Vision and Pattern Recognition'.

Dey, A. K. & Abowd, G. D. (2000), Towards a better understanding of context and context-awareness, *in* 'CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-Awareness'.

Doherty, L., Pister, K. & Ghaoui, L. (2001), Convex position estimation in wireless sensor networks, *in* 'IEEE Infocom'.

Doussis, E. (1993), An Ultrasonic Position Detecting System for Motion Tracking in Three Dimensions, PhD thesis, Tulane University.

Elnahrawy, E., Li, X. & Martin, R. P. (2004), The limits of localization using signal strength: A comparative study, *in* 'IEEE Sensor and Ad hoc Communications and Network Conference (SECON 2004)'.

Figueroa, F. & Mahajan, A. (1994), 'A robust navigation system for autonomous vehicles using ultrasonics', *Control Engineering Practice* **2**(1), 49–59.

Finn, G. G. (1987), Routing and addressing problems in large metropolitan-scale internetworks, Technical Report ISI/RR-87-180, Information Sciences Institute.

Forsyth, D., Mundy, J. L., Zisserman, A., Coelho, C., Heller, A. & Rothwell, C. (1991), 'Invariant descriptors for 3-D object recognition and pose', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(10).

Foy, W. (1976), 'Position-location solutions by taylor series estimation', *IEEE Transactions on Aerospace and Electronic Systems* **12**(2), 187–194.

Fuhrer, M., Nygard, J., Shih, L., Forero, M., Yoon, Y.-G., Mazzoni, M., Choi, H. J., Ihm, J., Louie, S., Zettl, A. & McEuen, P. (2000), 'Crossed nanotube junctions', *Science* **288**(5465), 494–497.

Harter, A. & Hopper, A. (1994), 'A distributed location system for the active office', *IEEE Network* **8**(1).

He, T., Huang, C., Blum, B. M., Stankovic, J. A. & Abdelzaher, T. F. (2005), 'Range-free localization and its impact on large scale sensor networks', *ACM Transactions on Embedded Computing Systems* **4**(4), 877–906.

Hecht, D. L. (2001), 'Printed embedded data graphical user interfaces', *Computer* **34**(3), 47–55.

Hightower, J. & Borriello, G. (2001), 'Location systems for ubiquitous computing', *Computer, IEEE Computer Society Press* **34**(8), 57–66.

Hill, J. & Culler, D. (2002), 'MICA: a wireless platform for deeply embedded networks', *IEEE Micro* **22**(6), 12–24.

HP (2006), 'HP unveils revolutionary wireless chip that links the digital and physical worlds', Press Release: Hewlett-Packard Development Company, Palo Alto, California, US, July 17.

Hull, R., Neaves, P. & Bedford-Roberts, J. (1997*a*), Towards situated computing, *in* '1st IEEE International Symposium on Wearable Computers', IEEE Computer Society, p. 146.

Hull, R., Neaves, P. & Bedford-Roberts, J. (1997*b*), Towards situated computing, *in* 'ISWC '97: Proceedings of the 1st IEEE International Symposium on Wearable Computers', IEEE Computer Society.

Jared, D. A., Flores, L. N., Hecht, D. L., Stearns, R. G. & Chang, K. H. (2001), *Methods and apparatus for robust decoding of glyph address carpets*, United States Patent: US 6,208,771 B1. United States Patent and Trademark Office.

Ji, X. & Zha, H. (2004), Sensor positioning in wireless ad-hoc sensor networks with multidimensional scaling, *in* 'IEEE INFOCOM', pp. 2652–2661.

Jin, J., Thomas, G. A., Niblett, T. & Urquhart, C. (1997), A versatile camera position measurement system for virtual reality TV production, *in* 'International Broadcasting Convention', pp. 284–289.

Kleinrock, L. & Silvester, J. (1978), Optimum transmission radii for packet radio networks or why six is a magic number, *in* 'IEEE National Telecommunications Conference', pp. 4.3.1–4.3.5.

Kondacs, A. (2003), Biologically-inspired self-assembly of 2D shapes, using global-to-local compilation, *in* 'International Joint Conference on Artificial Intelligence (IJCAI)'.

Kong, J., Franklin, N., Zhou, C., Chapline, M., Peng, S., Cho, K. & Dai, H. (2000), 'Nanotube molecular wires as chemical sensors', *Science* **287**(5453), 622–625.

Kwon, Y., Mechitov, K., Sundresh, S., Kim, W. & Agha, G. (2005), Resilient localization for sensor networks in outdoor environments, *in* '25th IEEE International Conference on Distributed Computing Systems', pp. 643–652.

Liu, J., Zhang, Y. & Zhao, F. (2006), Robust distributed node localization with error management, *in* 'ACM MobiHoc '06'.

Livstone, M. S., Weiss, R. & Landweber, L. F. (2006), 'Automated design and programming of a microfluidic DNA computer', *International Journal on Natural Computing* **5**(1), 1–13.

Martincic, F. & Schwiebert, L. (2004), Distributed perimeter detection in wireless sensor networks, Technical Report WSU-CSC-NEWS/03-TR03, Wayne State University.

Martincic, F. & Schwiebert, L. (2006), Distributed perimeter detection in wireless sensor networks, *in* 'Third International Conference on Networked Sensing Systems (INSS'06)'.

Martinez, K., Padhy, P., Elsaify, A., Zou, G., Riddoch, A., Hart, J. K. & Ong, H. L. R. (2006), Deploying a sensor network in an extreme environment, *in* 'Sensor Networks, Ubiquitous and Trustworthy Computing, Taiwan', pp. 186–193.

Nagpal, R. (2001), Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics, PhD thesis, MIT Department of Electrical Engineering and Computer Science.

Nagpal, R. (2003), A catalog of biologically-inspired primitives for engineering self-organization, *in* 'Engineering Self-Organising Systems', pp. 53–62.

Nagpal, R., Shrobe, H. & Bachrach, J. (2003), Organizing a global coordinate system from local information on an ad hoc sensor network, *in* 'IPSN'.

Niculescu, D. & Nath, B. (2001), Ad hoc positioning system APS, *in* 'IEEE Globecom'.

Niculescu, D. & Nath, B. (2003*a*), 'DV bsaed positioning in ad hoc networks', *Journal of Telecommunication Systems* **22**(1-4), 267–280.

Niculescu, D. & Nath, B. (2003*b*), Trajectory based forwarding and its applications, *in* '9th annual international conference on Mobile computing and networking (MobiCom '03)', ACM Press, pp. 260–272.

Nowak, R. & Mitra, U. (2003), 'Boundary estimation in sensor networks: Theory and methods', *Information Processing in Sensor Networks: Second International Workshop, IPSN 2003, Palo Alto, CA, USA, April 22-23, 2003. Proceedings* **2634/2003**, 80–95.

Orr, R. & Abowd, G. (2000), The smart floor: A mechanism for natural user identification and tracking, *in* '2000 Conference on Human Factors in Computing Systems (CHI 2000)'.

Parashar, M. & Hariri, S. (2004), Autonomic computing: An overview, *in* 'Unconventional Programming Paradigms', pp. 257–269.

Pascoe, J. (1998), Adding generic contextual capabilities to wearable computers, *in* 'The Second International Symposium on Wearable Computers', IEEE Computer Society, pp. 92–99.

Paun, G., Rozenberg, G. & Salomaa, A. (2006), *DNA Computing: New Computing Paradigms*, Springer.

Peterson, R. L., Ziemer, R. E. & Borth, D. E. (1995), *Introduction to Spread Spectrum Communications*, Prentice Hall International Editions.

Pettersson, M. P. (2001), *Encoded paper for optical reading*, International Patent WO 01/26032 A1. World Intellectual Property Organisation.

Pettersson, M. P. (2006), *Position determination II - graphic*, United States Patent Application Publication: US 2006/007641 A1. United States Patent and Trademark Office.

Picard, R. (1997), *Affective Computing*, MIT Press.

Pottie, G. & Kaiser, W. (2000), 'Wireless integrated network sensors', *Communications of the ACM* **43**(5), 51–58.

Priyantha, N. B. (2005), The Cricket Indoor Location System, PhD thesis, MIT Department of Electrical Engineering and Computer Science.

Priyantha, N. B., Chakraborty, A. & Balakrishnan, H. (2000), The Cricket Location-Support System, *in* '6th ACM MOBICOM', Boston, MA.

Priyantha, N. B., Miu, A. K. L., Balakrishnan, H. & Teller, S. (2001), The cricket compass for context-aware mobile applications, *in* '6th ACM MOBICOM Conference'.

Randell, C. & Muller, H. (2001), Low cost indoor positioning system, *in* 'UbiComp 2001: Ubiquitous Computing', pp. 42–48.

Rauch, E. (2003), 'Discrete, amorphous physical models', *International Journal of Theoretical Physics* **42**(2), 329–348.

Reichenbach, F., Born, A., Timmermann, D. & Bill, R. (2006), Splitting the linear least squares problem for precise localization in geosensor networks, *in* 'GIScience', pp. 321–337.

Rekimoto, J. & Ayatsuka, Y. (2000), CyberCode: designing augmented reality environments with visual tags, *in* 'DARE 2000 on Designing Augmented Reality Environments', ACM Press, pp. 1–10.

Romer, K. & Mattern, F. (2004), 'The design space of wireless sensor networks', *IEEE Wireless Communications* **11**(6).

Rueckes, T., Kim, K., Joselevich, E., Tseng, G. Y., Cheung, C.-L. & Lieber, C. M. (2000), 'Carbon nanotube-based nonvolatile random access memory for molecular computing', *Science* **289**(5476), 94–97.

Ryan, N. S., Pascoe, J. & Morse, D. R. (1998), Enhanced reality fieldwork: the context-aware archaeological assistant, *in* V. Gaffney, M. van Leusen & S. Exxon, eds, 'Computer Applications in Archaeology', British Archaeological Reports, Tempus Reparatum, Oxford.

Saha, S., Chaudhuri, K., Sanghi, D. & Bhagwat, P. (2003), Location determination of a mobile device using IEEE 802.11 access point signals, *in* 'IEEE Wireless Communications and Networking Conference (WCNC)', pp. 1987–1992.

Savarese, C., Rabaey, J. & Langendoen, K. (2002), Robust positioning algorithms for distributed ad-hoc wireles sensor networks, *in* 'USENIX Annual Technical Conference', pp. 317–327.

Savvides, A., Garber, W., Adlakha, S., Moses, R. & Srivastava, M. (2003), On the error characteristics of multihop node localization in ad-hoc sensor networks, *in* 'Second International Workshop on Information Processing in Sensor Networks (IPSN'03)', pp. 317–332.

Savvides, A., Han, C. & Srivastava, M. (2001), Dynamic fine grained localization in ad-hoc sensor networks, *in* 'Fifth International Conference on Mobile Computing and Networking (Mobicom 2001)', pp. 166–179.

Savvides, A., Park, H. & Srivastava, M. B. (2002), The bits and flops of the n-hop multilateration primitive for node localization problems, *in* 'First ACM International Workshop on Sensor Networks and Applications'.

Schilit, B., Adams, N. & Want, R. (1994), Context-aware computing applications, *in* 'IEEE Workshop on Mobile Computing Systems and Applications', pp. 85–90.

Schneier, B. (1996), *Applied Cryptography, Second Edition*, John Wiley & Sons.

Shang, Y., Ruml, W., Zhang, Y. & Fromherz, M. P. (2003), Localization from mere connectivity, *in* 'MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing', ACM Press, New York, NY, USA, pp. 201–212.

Shang, Y., Ruml, W., Zhang, Y. & Fromherz, M. P. (2004), 'Localization from connectivity in sensor networks', *IEEE Transactions on Parallel Distributed Systems* **15**(11), 961–974.

Shang, Y., Shi, H. & Ahmed, A. (2004), Performance study of localization methods for ad-hoc sensor networks, *in* 'IEEE International Conference on Mobile Adhoc and Sensor Systems', pp. 184–193.

Spirito, M. A. (2001), 'On the accuracy of cellular mobile station location estimation', *IEEE Transactions on Vehicular Technology* **50**(3), 674–685.

Stojmenović, I. (2002), 'Position-based routing in ad hoc networks', *IEEE Communications Magazine* **40**(7), 128–134.

Takagi, H. & Kleinrock, L. (1984), 'Optimal transmission ranges for randomly distributed packet radio terminals', *IEEE Transactions on Communications* **32**(3), 246–257.

Tarnacha, A. & Porta, T. L. (2003), E-STROBE: an adaptive beacon activation algorithm for sensor localization, *in* 'IEEE 58th Vehicular Technology Conference', Vol. 5, pp. 2951–2955.

Tennenhouse, D. (2000), 'Proactive computing', *Communications of the ACM* **43**(5), 43–50.

Theraulaz, G., Gautrais, J., Camazine, S. & Deneubourg, J.-L. (2003), 'The formation of spatial patterns in social insects: From simple behaviours to complex structures', *Philosophical Transactions. Royal Society of London* **316**, 1283–1282.

Tsai, R. Y. (1987), 'A versatile camera calibration technique for high-accuracy 3-D machine vision metrology using off-the-shelf TV cameras and lenses', *IEEE Journal of Robotics and Automation* **RA-3**(4), 323–344.

Wang, J., Dang, Y., Wang, Q., Chen, L. & Ma, X. (2005), *Strokes localization by m-array decoding and fast image matching*, United States Patent Application Publication: US 2005/0201621 A1. United States Patent and Trademark Office.

Want, R., Hopper, A., Falcao, V. & Gibbons, J. (1992), 'The active badge location system', *ACM Transactions on Information Systems* **10**(1), 91–102.

Want, R., Pering, T. & Tennenhouse, D. (2003), 'Comparing autonomic and proactive computing', *IBM Systems Journal* **42**(1), 129–135.

Ward, A., Jones, A. & Hopper, A. (1997), 'A new location technique for the active office', *IEEE Personal Communications* **4**(5), 42–47.

Warneke, B. (2003), Ultra-Low Energy Architectures and Circuits for Cubic Millimeter Distributed Wireless Sensor Networks, PhD thesis, UC Berkeley.

Warneke, B., Last, M., Leibowitz, B. & Pister, K. (2001), 'Smart dust: Communicating with a cubic-millimetre computer', *Computer Magazine* pp. 44–51.

W.Clark & Weng, L.-J. (May 1994), 'Maximal and near-maximal shift register sequences: Efficient event counters and easy discrete logarithms', *IEEETC: IEEE Transactions on Computers* **43**(5), 560– 568.

Weiser, M. (1991), 'The computer for the 21st Century', *Scientific American* **265**(3), 66–75.

Weiser, M. (1993), 'Some computer science problems in ubiquitous computing', *Communications of the ACM* **36**(7), 74–83.

Wood, A., Stankovic, J. & Son, S. (2003), JAM: a jammed-area mapping service for sensor networks, *in* '24th IEEE Real-Time Systems Symposium', IEEE Computer Society, pp. 286–297.

Youssef, M., Agrawala, A. & Shankar, A. U. (2003), WLAN location determination via clustering and probability distributions, *in* 'IEEE Pervasive Computing and Communications Conference (PerCom)', p. 143.

Zhang, B., Sukhatme, G. & Requicha, A. (2004), Adaptive sampling for marine microorganism monitoring, Technical Report CRES-04005, Center for Robotics and Embedded Systems, University of Southern California.

Zhang, Z. (2000), 'A flexible new technique for camera calibration', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(11), 1330–1334.

Appendix A

Source Code for Simulator, Simulations, and Algorithms

Please find a CDROM attached to the back cover. The CDROM has the following contents:

\src\devices	Java source code for the simulator itself, including the basic framework for communication between nodes.
\src\sims	Java source code for the simulations and the algorithms.
\src\sims\bubbles	Java source code for the membrane detection algorithm and simulation.
\src\sims\bubbles	Java source code for the coordinate propagation algorithm and simulation.
\src\sims\edgedetection	Java source code for the simulation of Martincic and Schwiebert's edge detection algorithm.
\bin	Java class files, compiled under jdk1.6.
\lib	Library dependencies.
\readme.txt	Some basic details for getting started.
\example.sh	Unix bash script that will run some example simulations.
\example.bat	Windows batch file that will run some example simulations.