Formation of a Geometric Pattern with a Mobile Wireless Sensor Network

Justin Lee, Svetha Venkatesh and Mohan Kumar

Department of Computing

Curtin University of Technology

March 22, 2004

Abstract

Mobile wireless sensor networks (MWSNs) will enable information systems to gather detailed information about the environment on an unprecedented scale. These self-organising, distributed networks of sensors, processors and actuators that are capable of movement have a broad range of potential applications, including military reconnaissance, surveillance, planetary exploration and geophysical mapping. In many of the foreseen applications the MWSN will need to *form* a geometric pattern without assistance from the user. In military reconnaissance, for example, the nodes will be dropped onto the battlefield from a plane and land at random positions. The nodes will be expected to arrange themselves into a predetermined formation in order to perform a specific task. Thus, we present algorithms for forming a line, circle and regular polygon from a given set of random positions. The algorithms are distributed and use no communication between the nodes to minimise energy consumption. Unlike past studies of geometric problems where algorithms are either tested in simulations where each node has global knowledge of all the other nodes or implemented on a small number of robots, the robustness of our algorithms has been studied with simulations that model the sensor system in detail. The simulations demonstrate that the algorithms are robust against random errors in the sensors and actuators.

Conventional sensor systems use one of two approaches to a sensing problem[12]. One approach is to use large complex sensors positioned far from the phenomena being observed. The other approach is to use several sensors with a carefully engineered placement and communications topology. There is another approach, however, that is starting to receive more research attention:

- use a large number of spatially *distributed* nodes (sensor units equipped with processing and communications hardware) *embedded* within or in close proximity to the phenomena being observed[7]; and
- use *cooperation* among the nodes to *autonomously* perform the sensing task.

Such a network is called a *wireless sensor network (WSN)*.

There is a field within WSNs that is highly ambitious and has only just begun to emerge. This area of research, called *mobile wireless sensor networks (MWSNs)*, presents many interesting research challenges. MWSNs will enable information systems to gather detailed information about the environment on an unprecedented scale. They have a vast range of potential applications, including

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

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

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

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

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

- Nodes are prone to failure or damage, especially in military applications.
- The user might be unable to manually configure the network, because (a) the operating environment is difficult to access or dangerous, as in military applications; or (b) the network must be deployed on short notice.
- The sensors and actuators are subject to error.
- Wireless communication requires a lot of power while the nodes have a limited supply of energy, restricting the amount of information that can be shared between the nodes.

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

Where do we begin in our investigation of motion coordination algorithms for MWSNs? What are the fundamental motion coordination problems that will be encountered in typical MWSN applications? We note that in many of the foreseen applications the given task must be carried out with the nodes arranged in a certain geometric pattern. Furthermore, in many applications where a certain geometric pattern is required, the MWSN must be deployed without assistance from the user. In military applications such as reconnaissance, for example, the nodes will be dropped from a plane behind enemy lines, landing at random positions. As explained by Balch and Arkin[2], the formation used has a major impact on performance in such tasks. Thus, in this paper, we investigate the problem of *forming* a geometric pattern from a given set of random positions.

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

Little research has been done on the problem of forming a geometric pattern with multiple mobile robots beginning from a random set of positions. Sugihara and Suzuki[17] and Défago and Konagaya[6] propose algorithms for forming a circle, but their algorithms require the robots to have global knowledge of all other robots. Using our algorithm this task can be performed with only partial knowledge of the other robots and this has been demonstrated with simulations.

The approach taken in this paper enables us to gain insight into the issues that would arise in real implementations of motion coordination algorithms. There are challenges that must be considered in the design of motion coordination algorithms if vision is used to locate other nodes, such as occlusion and image noise. Our study is unique in that the impact of some of the errors in the sensors and actuators are investigated in simulations that model the sensor system in detail.

The rest of the paper is as follows. A review of related work is given in Section 1. In Section 2, algorithms for forming a line, circle and regular polygon are presented. The robustness of the algorithms under various conditions is tested through simulations in Section 3. Our final conclusions for this paper are discussed in Section 4.

1 Review of Related Work

The field of MWSNs is relatively new, so literature relating specifically to this area is scarce. The following areas make a valuable contribution to research on MWSNs:

1. Wireless sensor networks
2. Cooperative mobile robotics

WSNs provide the main foundation for research on MWSNs, as MWSNs are essentially an extension of WSNs. *Cooperative mobile robotics* addresses the highly complex problems in motion coordination and cooperative behaviour which are introduced when the nodes have mobility. Akyildiz *et al.*[1] provide a comprehensive and recent survey of research in WSNs. A survey of research on cooperative mobile robotics is given by Cao *et al.*[3].

Sugihara and Suzuki[17] develop distributed algorithms for forming geometric patterns with mobile robots. They present the following algorithms:

- Algorithm CIRCLE: Forms a circle.
- Algorithm CONTRACTION: Given a parameter n, forms an n-sided polygon when $n \geq 3$ and distributes the robots uniformly along a line segment formed with Algorithm FILLPOLYGON when $n = 2$.
- Algorithm FILLCIRCLE: Distributes the robots uniformly within an approximation of a circle.
- Algorithm FILLPOLYGON: Distributes the robots uniformly within a convex polygon.
- Algorithm FOLLOW: Divides the robots into a specified number of groups.

Simulations show that sometimes a *Reuleaux's triangle* results from Algorithms CIRCLE and FILLCIRCLE. Also, "bubbles"—convex regions with no robots—sometimes appear within

the shape with Algorithms FILLCIRCLE and FILLPOLYGON. Chen and Luh[4,5] modify Algorithm CIRCLE to incorporate collision avoidance.

Yamaguchi uses feedback control laws to achieve a desired formation[20] and enclose a target using holonomic[19] or nonholonomic[18] mobile robots. Yamaguchi's approach is decentralised, well grounded mathematically and includes collision avoidance, however it is limited to formations that are already organised into some form of open chain.

Flocchini *et al.*[8–10], Prencipe[15] and Prencipe and Gervasi[16] study the computational issues of pattern formation. A model of time and motion is proposed which the authors believe more realistically models a system of robots. They also investigate how the agreement between the robots' local coordinate systems affects the solvability of a pattern formation problem. The following theorem summarises their findings:

1. With a common coordinate system of (that is, the robots agree on) both x and y directions and orientations, the robots can form an arbitrary given pattern[10]. This is equivalent to assuming that the robots are equipped with a compass.

2. With agreement on only one axis direction and orientation, the pattern formation problem is unsolvable when n is even, while it can be solved if n is odd[10].

3. With agreement on only one axis direction and orientation, an even number of robots can form only symmetric patterns that have at least one axis of symmetry not passing through any vertex of the pattern[9].

4. With no agreement at all, the robots cannot form an arbitrary given pattern[10].

Défago and Konagaya[6] propose a distributed algorithm for forming the smallest enclosing circle from a given set of initial locations. The motion planning of each robot is based on the relationship between its *Voronoi cell* and the smallest enclosing circle of the beginning

configuration. The drawback of this algorithm is that the nodes must have global knowledge of all the robot locations to compute the smallest enclosing circle.

Fredslund and Matarić[11] develop a simple algorithm for establishing and maintaining a formation where each robot, except the leader of the formation, follows a neighbouring robot. The algorithm is limited to formations with a single chain of positional dependencies and under the assumption that each robot's field of view is limited to $\pm 90^\circ$ from the front, a robot cannot follow another that is behind it. The leader of the formation is called the *conductor*, and the robot that a neighbouring robot chooses to follow is called the *friend* of that robot. Each robot except the conductor maintains a certain distance and bearing from its friend. If one end of the chain of friends is at the front of the formation, the formation is *noncentred*; otherwise, it is *centred*. A centred and noncentred formation are shown in Figure 1. Each robot is assigned a unique numerical ID, which is used to determine the appropriate friend. If a robot has an ID lower than that of the conductor, it chooses a friend with an ID greater than its own. Similarly, if a robot has an ID greater than that of the conductor, it chooses a friend with an ID lower than its own. For noncentred formations the conductor has the lowest ID, thus the robots will always choose friends with a lower ID for these formations. The soundness of the algorithm was demonstrated with simulations and experiments using four robots.

2 Algorithms

We present three algorithms:

Algorithm L: Forms a line parallel with the x-axis with the nodes separated by a specified constant distance S[1].

[1] In the thesis by Lee[13], this algorithm is labelled Algorithm CS/G.

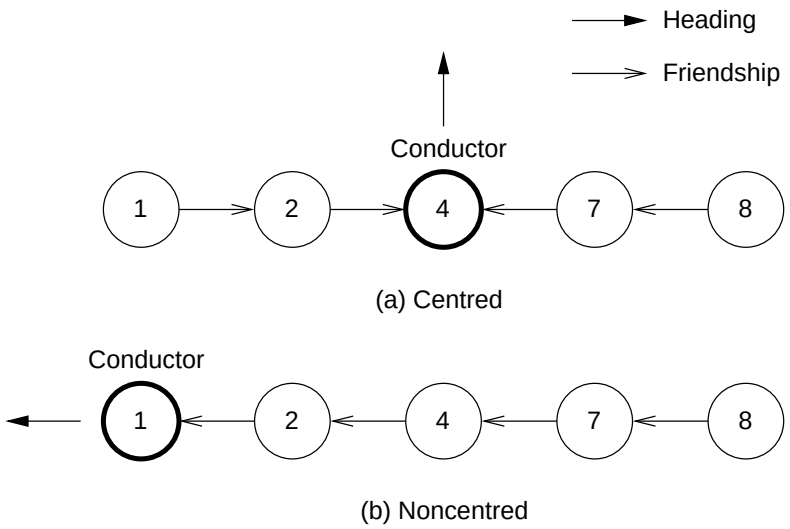

Figure 1: Algorithm of Fredslund and Matarić which uses "friendships" to form and maintain a geometric pattern.

Algorithm C: Forms a circle with a given radius R_c centred at a landmark.

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

Each algorithm is comprised of two components:

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

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

Each of these components is described below.

2.1 Vision

A number of assumptions have been made about the environment:

1. The ground is an infinitely large, perfectly flat plane.

2. The nodes are cylindrical in shape and always upright.

3. Each node is equipped with eight cameras positioned evenly on the circumference of the cylinder to achieve a 360° field of view, as illustrated in Figure 2. The views of adjacent cameras overlap to ensure coverage in all directions.

4. No other objects are present.

As a result of the above conditions, finding the nodes in the images is simply a matter of searching for objects approximately rectangular in shape centred on the horizon. The image was repeatedly segmented using a recursive algorithm. As illustrated in Figure 3, regions of interest are identified and segmented using histograms of (a) the number of feature pixels counted in each column, and (b) the number of feature pixels counted in each row of the segment. Once a node has been identified, the relative position is calculated using geometry. A detailed description of the vision algorithm is given by Lee[13].

2.2 Motion Coordination

2.2.1 Algorithm L

In the discussion that follows, it is assumed that the desired line runs parallel to the x-axis in the Cartesian plane.

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

x-velocity The approach of Algorithm L is for the nodes at each end to maintain a separation of S from their neighbour while the other nodes maintain an equal separation from their

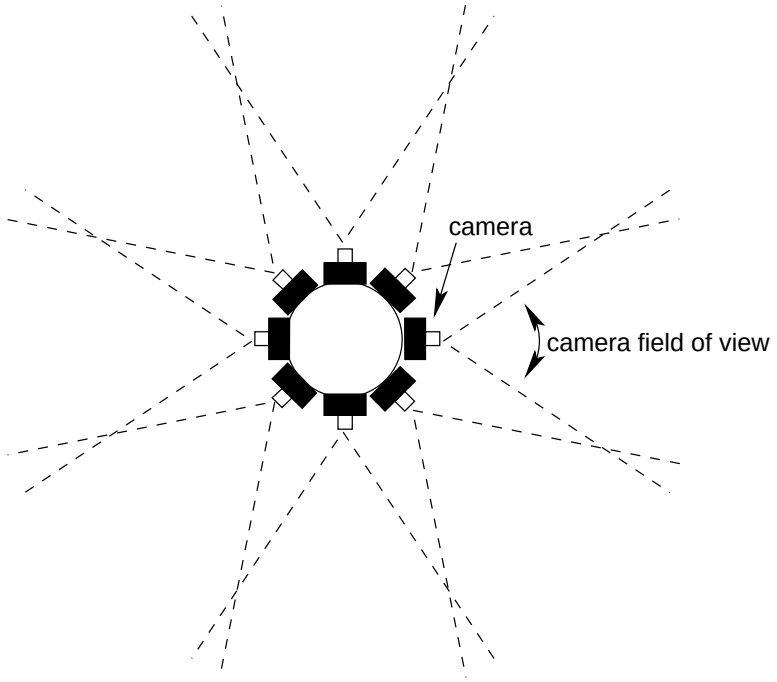

Figure 2: Using several cameras to achieve a 360° field of view.

neighbour on each side. When the system reaches a state of equilibrium, all neighbouring nodes are separated by S.

The algorithm is as follows. If another node is in-line with the node, that is, has the same x-coordinate, then move left or right, choosing the direction at random. Otherwise, if there is a neighbour on the left as well as on the right, move to the mid-point between them. If there is only a neighbour on one side, that is the node is at one end of the array, then maintain a constant distance S from the neighbouring node.

Since the aim is to concentrate on the fundamental issues of coordinating the motion of a group of robots that must rely on local information, the algorithm is designed to be as simple as possible. Hence, the magnitude of the x-velocity is set to a constant Δv_x when moving. When choosing the value of Δv_x and the motion constants used in the other algorithms, a decision is made based on the trade-off between convergence time and stability, as well as

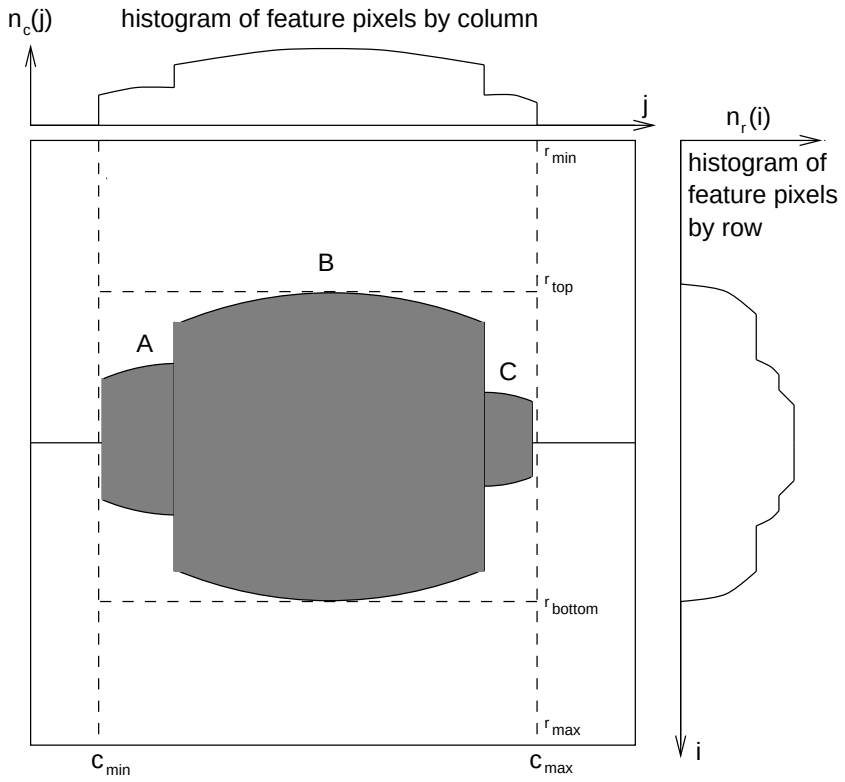

Figure 3: Isolating the nodes in an image using histograms of the number of feature pixels counted in the columns and rows.

other issues that are specific to the implementation of the mobility system.

Let

$$
\begin{aligned}
\text{neighbour\_in-line} &= \text{Boolean variable that is TRUE when another node has the same } x\text{-coordinate and FALSE otherwise;} \\
I &= \text{random variable that assumes a value of 0 or 1 with equal probability;} \\
\text{left\_neighbour} &= \text{Boolean variable that is TRUE when a neighbour is detected on the left and FALSE otherwise;} \\
\text{right\_neighbour} &= \text{Boolean variable that is TRUE when a neighbour is detected on the right and FALSE otherwise;} \\
d_{\text{left}} &= x\text{-distance of the left neighbour;} \\
d_{\text{right}} &= x\text{-distance of the right neighbour; and} \\
v_x &= \text{set } x\text{-velocity.}
\end{aligned}
$$

Algorithm 1 shows the algorithm for setting the x-velocity in Algorithm L.

Algorithm 1 Setting the x-velocity in Algorithm L.

if (neighbour_in-line) **then**
  $v_x \leftarrow (1 - 2I)\Delta v_x$
else if (left_neighbour **and** right_neighbour **and** $d_{\text{left}} < d_{\text{right}}$)
 or (left_neighbour **and not** right_neighbour **and** $d_{\text{left}} < S$)
 or (**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} > S$) **then**
  $v_x \leftarrow \Delta v_x$
else if (left_neighbour **and** right_neighbour **and** $d_{\text{left}} > d_{\text{right}}$)
 or (left_neighbour **and not** right_neighbour **and** $d_{\text{left}} > S$)
 or (**not** left_neighbour **and** right_neighbour **and** $d_{\text{right}} < S$) **then**
  $v_x \leftarrow -\Delta v_x$
else
  $v_x \leftarrow 0$
end if

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

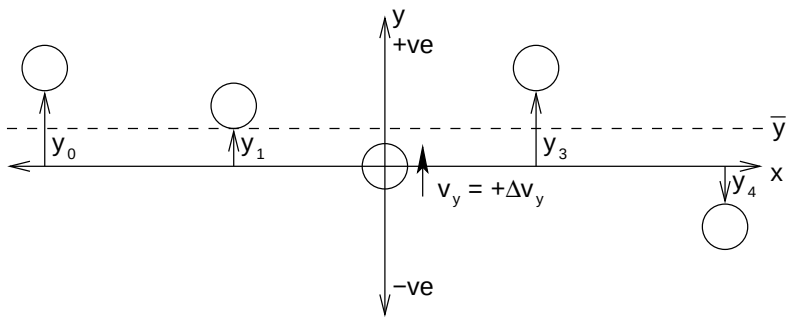

Figure 4: Using the average relative y-displacement of neighbouring nodes to align the nodes along the y-axis.

as illustrated in Figure 4. In the example shown in this figure the average is greater than zero, so the y-velocity of the node at the origin is set to Δv_y to align itself with its neighbours. The y-displacement is defined to be positive in one direction and negative in the other. The y-velocity is chosen according to the sign of the average y-displacement.

Let

$$\begin{aligned} y_i &= \text{relative } y\text{-displacement of neighbour } i; \\ n &= \text{number of known neighbours;} \\ \Delta v_y &= \text{magnitude of the } y\text{-velocity when moving forward or backward; and} \\ v_y &= \text{set } y\text{-velocity.} \end{aligned}$$

Algorithm 2 Setting the y-velocity in Algorithm L.

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

Note that the algorithm does not depend on each node having accurate knowledge of

the y-displacement of every other node. In fact, they only need to know the y-displacement of their left and right neighbours. However, the more global knowledge that is used in the decision the more efficient and robust the performance.

2.2.2 Algorithm C and Algorithm P/n

In the discussion that follows, for a given node i with polar coordinates (r_i, θ_i), as illustrated in Figure 5:

- The *clockwise neighbour* is the node with the closest θ-coordinate clockwise from θ_i and is labelled node $i - 1$.
- The *anticlockwise neighbour* is the node with the closest θ-coordinate anticlockwise from θ_i and is labelled node $i + 1$.
- The *immediate neighbours* are the clockwise and anticlockwise neighbours.
- θ_{cw} is the angle subtended by node i and its clockwise neighbour.
- θ_{acw} is the angle subtended by node i and its anticlockwise neighbour.

Algorithms C and P/n are analogous to Algorithm L in that the motion of each node consists of two components that are determined independently, and one of these components is responsible for achieving a symmetrical geometric relationship with the immediate neighbours. The motion of each node is considered in terms of the following two vector components, as illustrated in Figure 5, where the origin is the centroid of the nodes' current positions:

Radial component (v_{R}): A component *parallel* to a line from the origin to the node. v_{R} is set to move node i towards the edge of the shape. If $\rho(\theta_i)$ is the distance of the edge

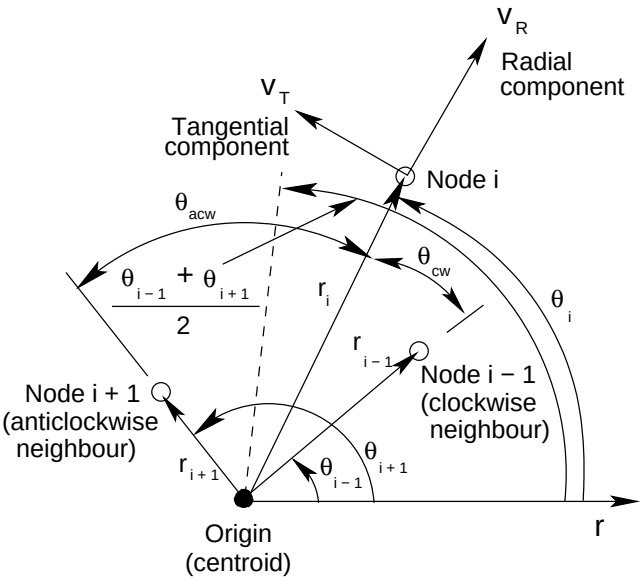

Figure 5: The *radial component*, v_{R}, and *tangential component*, v_{T}.

of the shape from the origin at an angle of θ_i, v_{R} is positive when $r_i < \rho(\theta_i)$, 0 when $r_i = \rho(\theta_i)$ and negative when $r_i > \rho(\theta_i)$.

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

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

For Algorithm C, $\rho(\theta_i) = R_{\mathrm{c}}$. For Algorithm P/$n$, $\rho(\theta_i)$ depends on θ_i and is determined as follows. Assume that the desired regular polygon is oriented so that one of the sides intersects the line $\theta = 0$ at 90°. First, we determine which side of the shape intersects a line drawn from the origin to the edge of the shape at an angle of θ_i with respect to the origin. We then determine the angle ϕ of a line from the origin that intersects this side of the shape

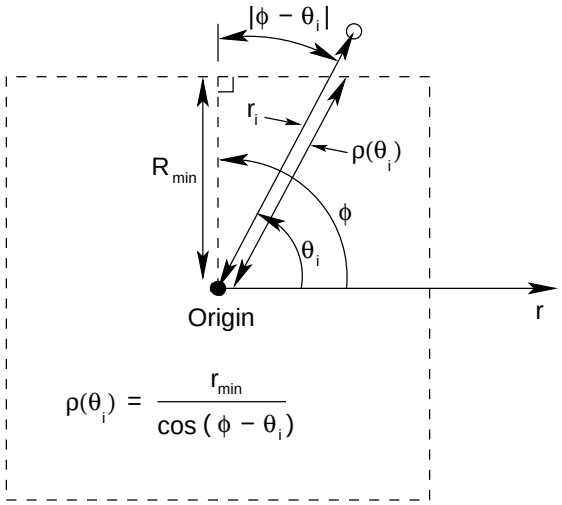

Figure 6: Calculating $\rho(\theta_i)$ when forming a regular polygon with Algorithm P/n.

at 90°. These two lines and the edge of the shape form a right triangle, where the vertex at the origin has an angle of $|\phi - \theta_i|$, as illustrated in Figure 6. Thus, using trigonometry, $\rho(\theta_i) = \frac{R_{\min}}{\cos(\phi - \theta_i)}$.

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

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

Step 1. The origin is set to the centroid of the nodes' current locations. Algorithm C begins with

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

and Algorithm P/n begins with

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

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

Step 2. Step 2 is shown in Algorithm 3.

Algorithm 3 Step 2 for Algorithms C and P/n.

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

3 Experiments

3.1 Simulation Setup

The algorithms were tested by simulation. The paths of the nodes were computed in discrete steps according to the algorithm being tested. To simulate vision, the *POV-Ray* ray-tracer[14] was used to produce images of the scene as viewed through the side cameras of each node.

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

Image pixels: An error $\sigma_{E_{r,g,b}}$ was added to the red, green and blue values of each pixel.

Orientation of cameras: A second source of error in the node's local knowledge is the compass reading, which is needed to determine the orientation of the cameras. The impact of error in the compass reading can be investigated by either fixing the compass reading and randomly varying the orientation of the cameras, or *vice versa*. In these experiments the former approach was used, as illustrated in Figure 7. The orientation of the cameras was rotated by a random angle σ_{E_θ} while it is assumed in the vision system that they were pointing parallel to the x-axis.

Velocity: A node may move in one of eight directions depending on the x-velocity, which may be $-\Delta v_x$, 0 or $+\Delta v_x$, and the y-velocity, which may be $-\Delta v_y$, 0 or $+\Delta v_y$. Thus the velocity is expressed as a 2-D vector and an error $\sigma_{E_{V_x,V_y}}$ is added to both the x- and y-components.

The simulations were carried out under the following conditions:

- The nodes are cylindrical with a height of 0.3 distance units and radius of 0.15 distance units. The cameras are positioned on the circumference of the cylinder mid-way between the top and bottom of the node.

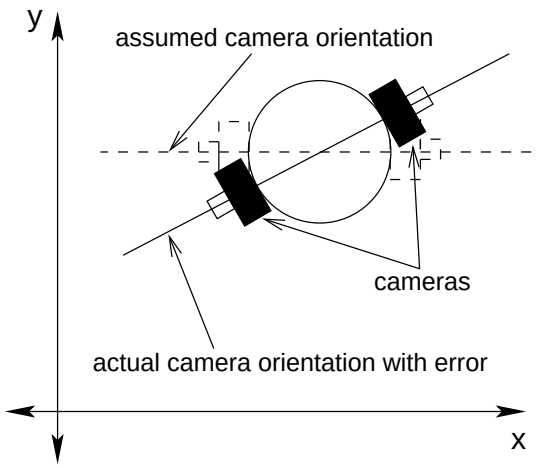

Figure 7: Error introduced into the orientation of the cameras.

- The "snapshot" taken by each camera is 100 pixels wide and 100 pixels high. The image plane is located 1 distance unit in front of the camera and is 1 distance unit wide and 1 distance unit high. The colour of each pixel is represented by RGB values ranging from 0 to 255. The chosen threshold value for deciding whether a pixel's colour is close enough to belong to a node is 50.
- For Algorithm L, $S = 1$ and $\Delta v_x = \Delta v_y = 0.05$.
- For Algorithms C and P/n, $R_{\mathrm{c}} = R_{\mathrm{min}} = 2$ and $\Delta v_{\mathrm{R}} = \Delta v_{\mathrm{T}} = 0.05$.
- Each simulation lasted for a duration of 100 time units.

Figure 8 shows the initial locations of the nodes.

3.2 Performance Analysis

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

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

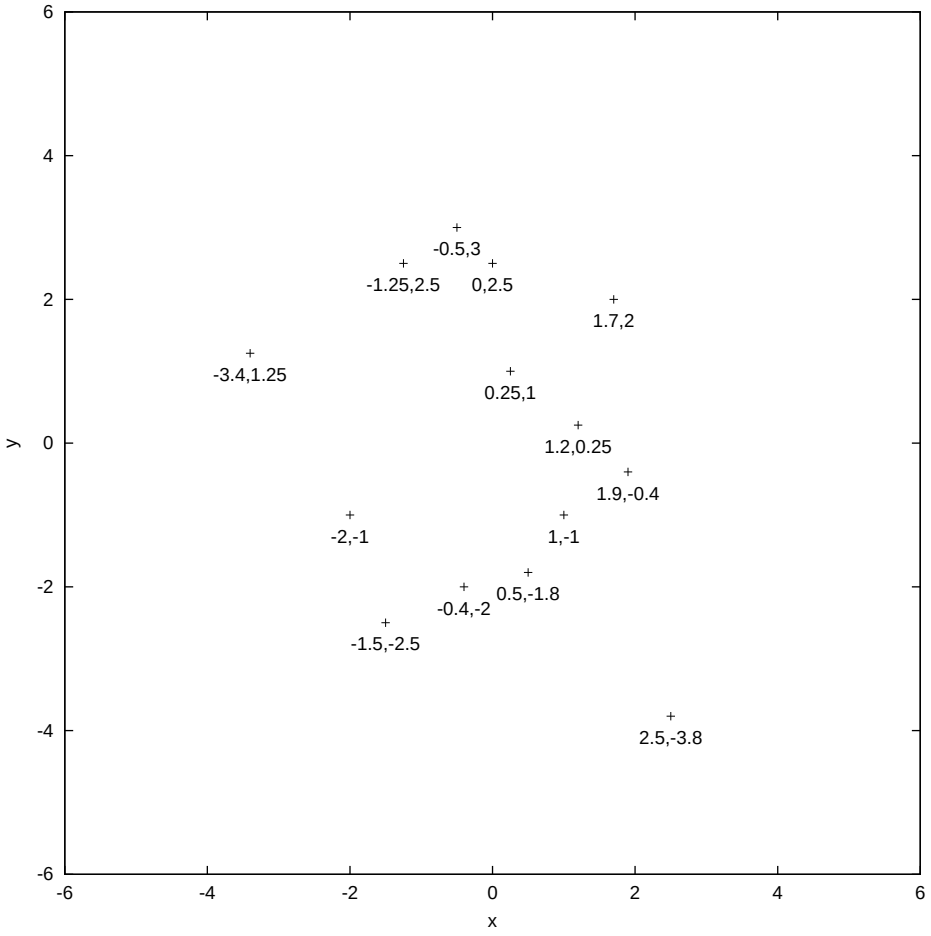

Figure 8: Initial locations of the nodes for the simulations.

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

These measures are defined below.

3.2.1 Error

The *error* $e(t)$ is defined as the average distance of each node i, $i = 0, \ldots, N-1$, where N is the number of nodes, from its corresponding *desired position* $(\hat{x}_i, \hat{y}_i)$ at a given time t.

When forming a line with Algorithm L, the *desired positions* are defined as the set of positions that are separated by S, have the same y-coordinate and minimise the sum of the square of the distance of each point from the corresponding node's current position (x_i, y_i). Thus the desired positions for Algorithm L are given by

$$\hat{x}_i = \frac{1}{N} \sum_{i=0}^{N-1} x_i - S\frac{(N-1)}{2} + Si \tag{1}$$

$$\hat{y}_i = \frac{1}{N} \sum_{i=0}^{N-1} y_i \tag{2}$$

for i, $i = 0, \ldots, N-1$.

For Algorithms C and P/n the desired positions are chosen to minimise the sum of the square of the angle between each node and its desired position with respect to the origin. Since it is important only that any given pair of neighbouring nodes subtend an angle of $\frac{2\pi}{N}$ with respect to the origin, the final angles of the nodes are rather arbitrary. Assume that N nodes each with polar coordinates (r_i, θ_i), for i, $i = 0, \ldots, N-1$, are indexed so that $\theta_0 < \theta_1$, $\theta_1 < \theta_2$, and so on until $\theta_{N-2} < \theta_{N-1}$. The desired polar coordinates of node i are

$$\hat{r}_i = \rho(\hat{\theta}_i) \tag{3}$$

$$\hat{\theta}_i = \frac{\sum_{i=0}^{N-1} \theta_i - \pi(N-1)}{N} + \frac{2\pi i}{N} \tag{4}$$

where $\rho(\hat{\theta}_i)$ is determined using Step 1 of the motion coordination algorithm for the shape that is desired.

3.2.2 Efficiency

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

$$q(t) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{|\mathbf{p}_i(t) - \mathbf{p}_i(0)|}{\ell_i(t)} \tag{5}$$

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

3.3 Results

3.3.1 Simulation Traces

Simulation traces of Algorithm L are shown in Figure 9. The progressive positions of the nodes after every ten time units are shown in the plot. The desired positions at the end of the simulation are also plotted. The lines formed are not quite parallel with the x-axis because of occlusion. When the nodes are nearly aligned along the y-axis each node's immediate neighbours occlude all the other nodes in the array. If, for example, the left neighbour is slightly in front by a distance Δy for each node, then the node at the left end of the array will be a distance $(N-1)\Delta y$ ahead of the node on the right end of the array.

Simulation traces of Algorithms C, P/3 and P/4 are shown in Figures 10, 11 and 12. Note that the vertices of the final shape can appear cut-off even if the nodes are at their desired positions because neighbouring nodes are always directly joined by a line and the set of desired positions does not necessarily include the vertices of the desired shape. Due

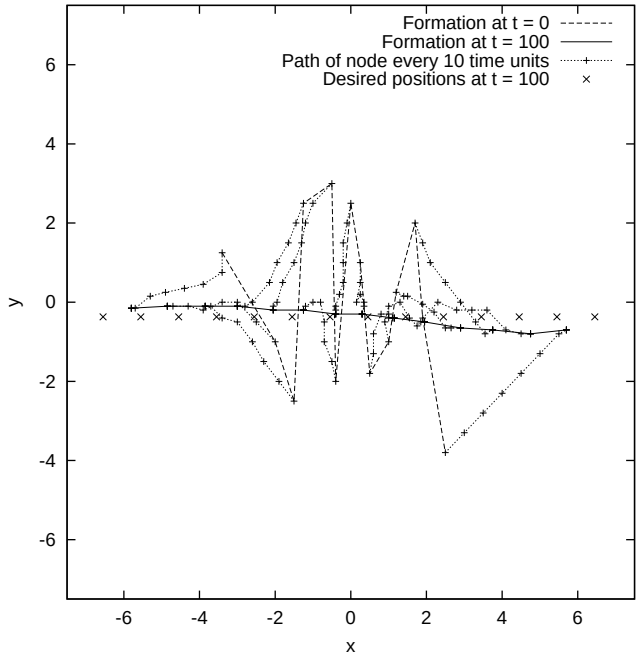

Figure 9: Simulation trace of Algorithm L.

to occlusion, the polygons were smaller than desired. The immediate neighbours of a given node, except for neighbours belonging to other sides, occlude the other nodes forming the same side. Since all the nodes forming the same side except the immediate neighbours are not included in the calculation of the centroid, the centroid is calculated to be further away than what it really is. As a result, there is a tendency for node i to finish with $r_i < \rho(\theta_i)$. The final error for Algorithm P/3 was greater than that for Algorithm P/4 because the triangle formed had longer sides than those of the square and therefore was the most affected by occlusion.

3.3.2 Robustness

The error and efficiency for each algorithm in simulations with $\sigma_{E_{r,g,b}} = 0$, $\sigma_{E_\theta} = 0^\circ$ and $\sigma_{E_{V_x,V_y}} = 0$ is shown in Figures 13 and 14, respectively.

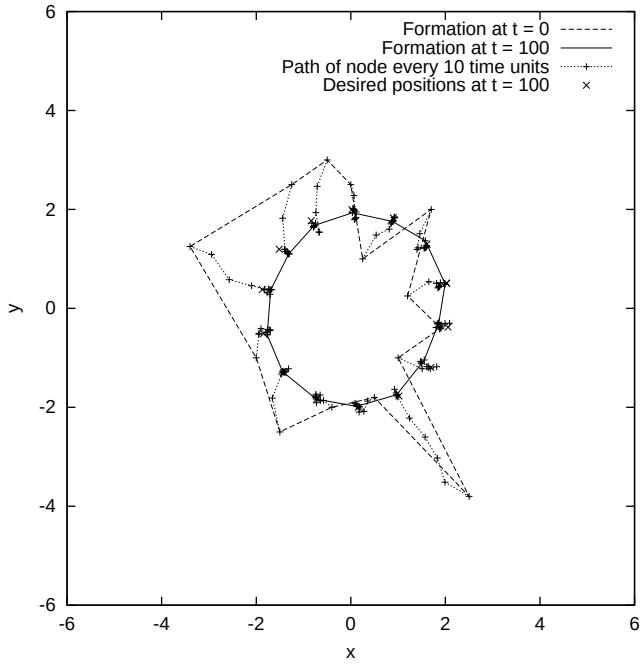

Figure 10: Simulation trace of Algorithm C.

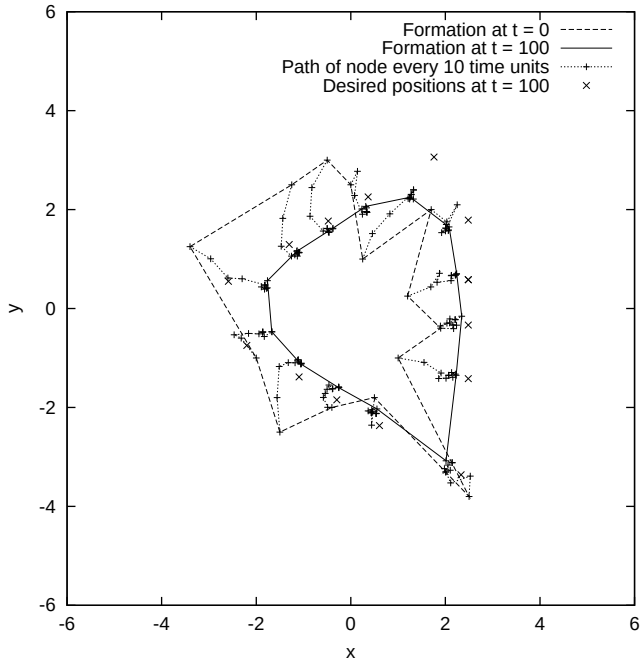

Figure 11: Simulation trace of Algorithm P/3.

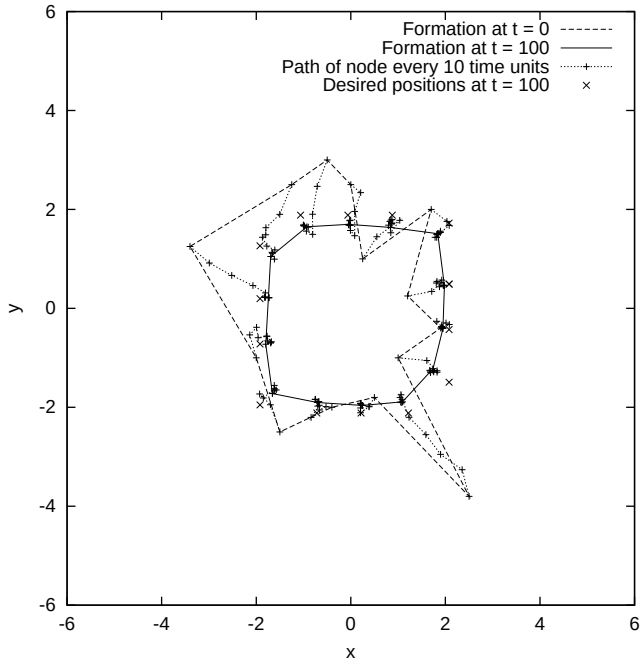

Figure 12: Simulation trace of Algorithm P/4.

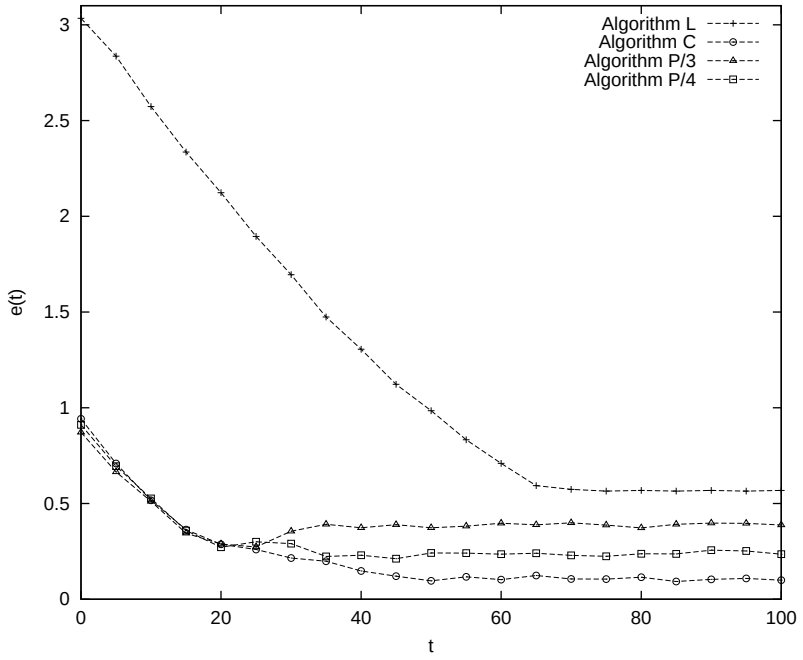

Figure 13: Error with all error parameters set to 0.

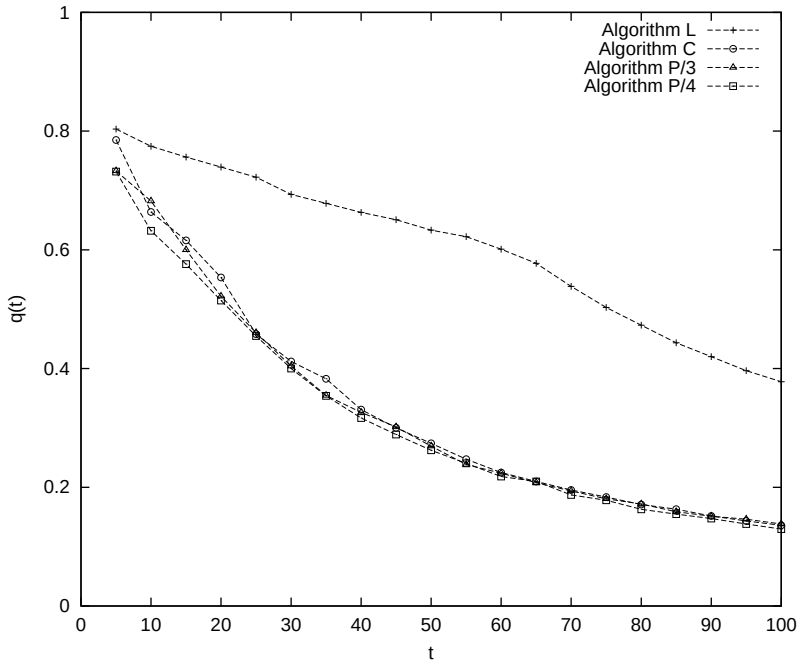

Figure 14: Efficiency with all error parameters set to 0.

Robustness Against Image Noise The error over time for each pattern-formation algorithm using $\sigma_{E_{r,g,b}} = 40$ is shown in Figure 15. The efficiency over time for each of these simulations is shown in Figure 16.

The error for Algorithms C, P/3 and P/4 initially decreased as the more distant nodes moved closer to the centroid and then rose at about $t = 15$ because each node i continues to move closer to the centroid than the desired distance $\rho(\theta_i)$. The vision algorithm tends to exclude "noisy" segments of pixels in distant nodes and fragment distant nodes with "noisy" vertical segments, causing the calculated location of the centroid to be further away than the real location. Hence the final shape is smaller than desired.

The error varied noticeably with shape because the accuracy of the calculated relative position of a node decreases with distance and the average distance between neighbouring desired positions varied with the shape. The average distance between neighbouring desired positions for the circle was lower than that for the square, thus when the nodes were close

to their desired positions they tended to locate their neighbours more accurately in the formation of the circle than in the formation of the square. Likewise, when the nodes were close to their desired positions they tended to locate their neighbours more accurately in the formation of the square than in the formation of the triangle.

The image noise did not have such a great impact on the performance of Algorithm L. The rate of convergence, that is the time taken for the system to become relatively stable, was lower when image noise was added. Ignoring small movements, the nodes settled at their final positions at about $t = 90$, while in the simulations with no image noise they settled at their final positions at about $t = 65$.

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

Robustness Against Error in Orientation of Cameras The error over time for each pattern-formation algorithm using $\sigma_{E_\theta} = 60°$ is shown in Figure 17. The efficiency over time for each of these simulations is shown in Figure 18.

Compared to the results for $\sigma_{E_{r,g,b}} = 40$, the impact on performance when $\sigma_{E_\theta} = 60°$ was greater for Algorithm L, but less for Algorithms C, P/3 and P/4. The rate of convergence for Algorithm L was much lower than that for $\sigma_{E_{r,g,b}} = 40$. The efficiency of the algorithms was significantly lower than that for $\sigma_{E_{r,g,b}} = 40$.

Robustness Against Error in Velocity The error over time for each shape-formation algorithm using $\sigma_{E_{V_x,V_y}} = 0.1$ is shown in Figure 19. The efficiency over time for each of these simulations is shown in Figure 20.

In terms of error, there was not much difference in the performance of the algorithms between this setting and $\sigma_{E_\theta} = 60°$. However, as expected, the efficiency was lower than

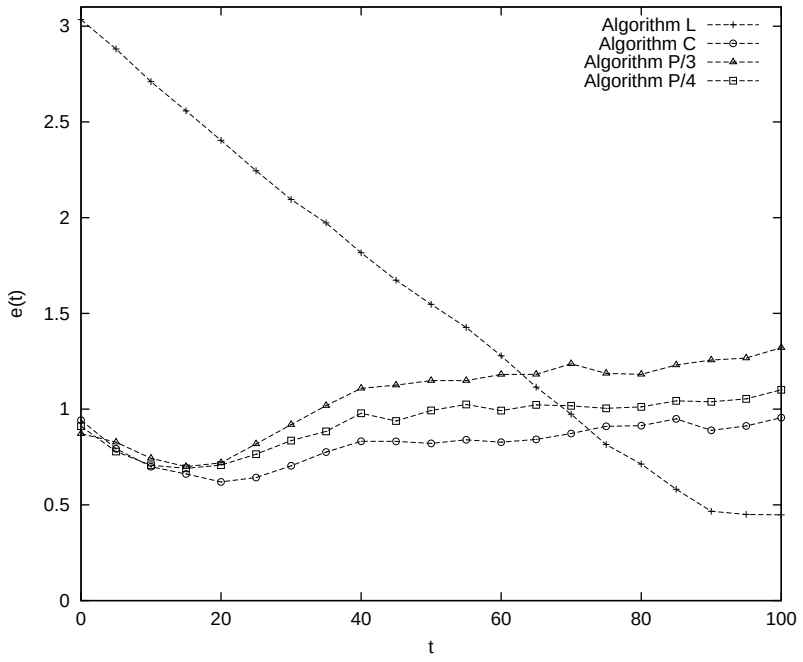

Figure 15: Error with $\sigma_{E_{r,g,b}} = 40$.

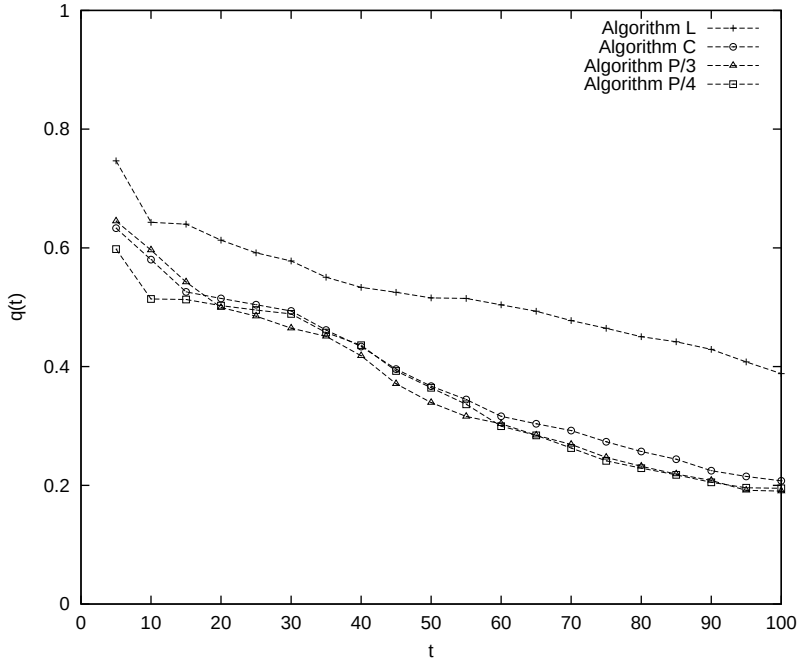

Figure 16: Efficiency with $\sigma_{E_{r,g,b}} = 40$.

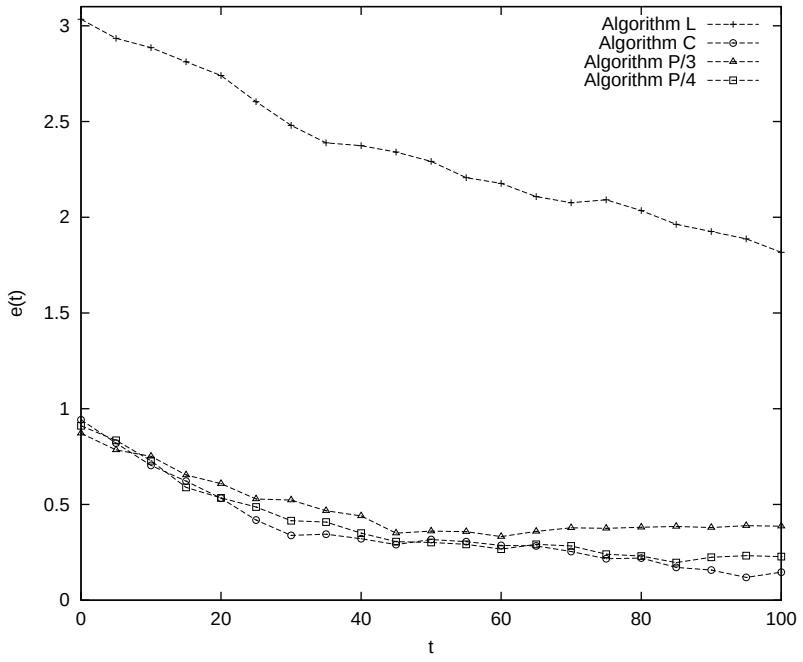

Figure 17: Error with $\sigma_{E_\theta} = 60°$.

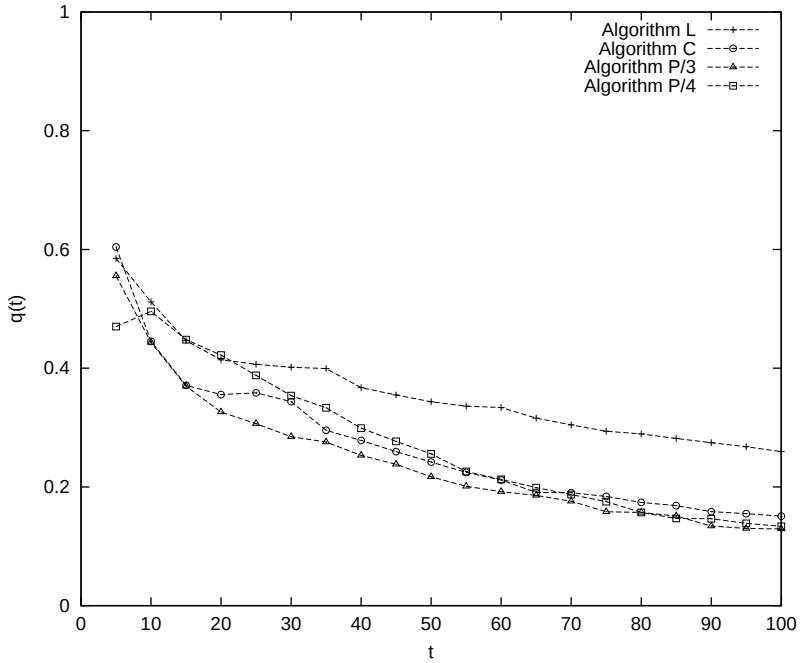

Figure 18: Efficiency with $\sigma_{E_\theta} = 60°$

that plotted for $\sigma_{E_\theta} = 60^\circ$ because of the unnecessary movement that is introduced by the velocity error.

Robustness Against All Types of Error Combined Figures 21 and 22 show the error and efficiency for simulations carried out with $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15^\circ$ and $\sigma_{E_{V_x,V_y}} = 0.05$. The performance of the algorithms was similar to that when $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 0^\circ$ and $\sigma_{E_{V_x,V_y}} = 0$, only a little more degraded.

4 Conclusion

In this paper we presented algorithms for forming a line, circle and regular polygon with mobile nodes beginning from random positions. These algorithms are distributed and only use locally obtained sensor information, making them suitable for implementation in MWSNs, where communication is costly in terms of energy consumption. Unlike past approaches to the formation of geometric patterns which were discussed in Section 1, the practical value of the algorithms was demonstrated with simulations that tested the robustness by introducing random errors into the system. The pattern formation algorithms presented in this paper have many potential defence, security and scientific applications where an MWSN must form a geometric pattern to carry out a reconnaissance, security or surveying task.

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

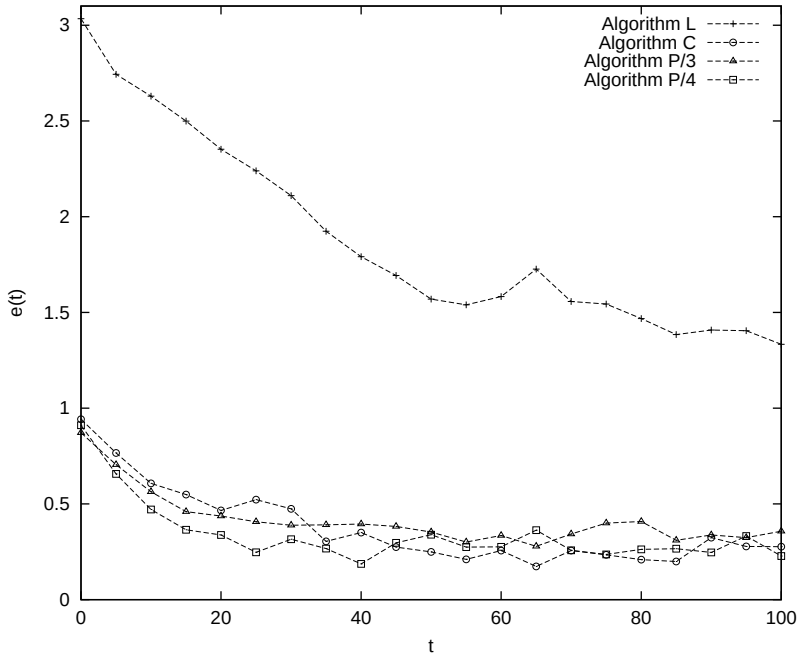

Figure 19: Error with $\sigma_{E_{V_x,V_y}} = 0.1$.

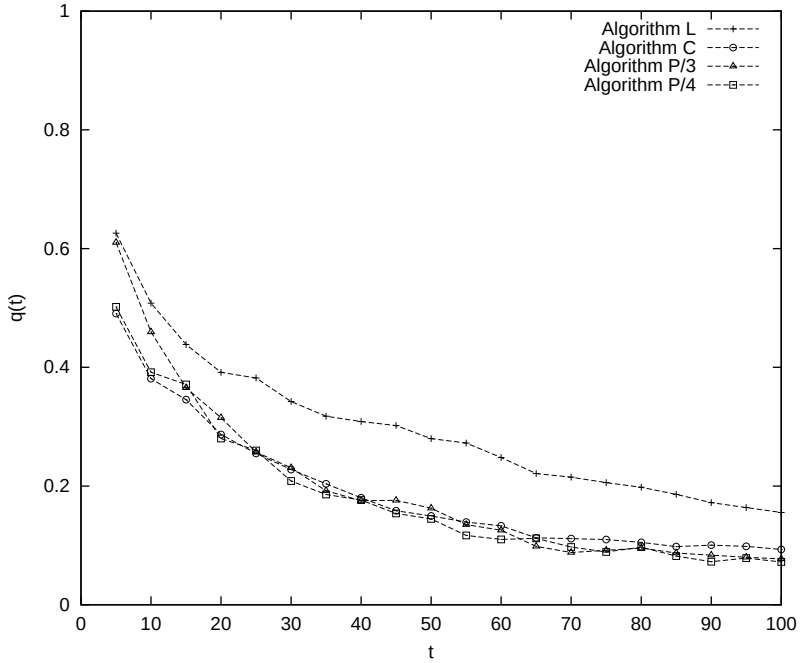

Figure 20: Efficiency with $\sigma_{E_{V_x,V_y}} = 0.1$.

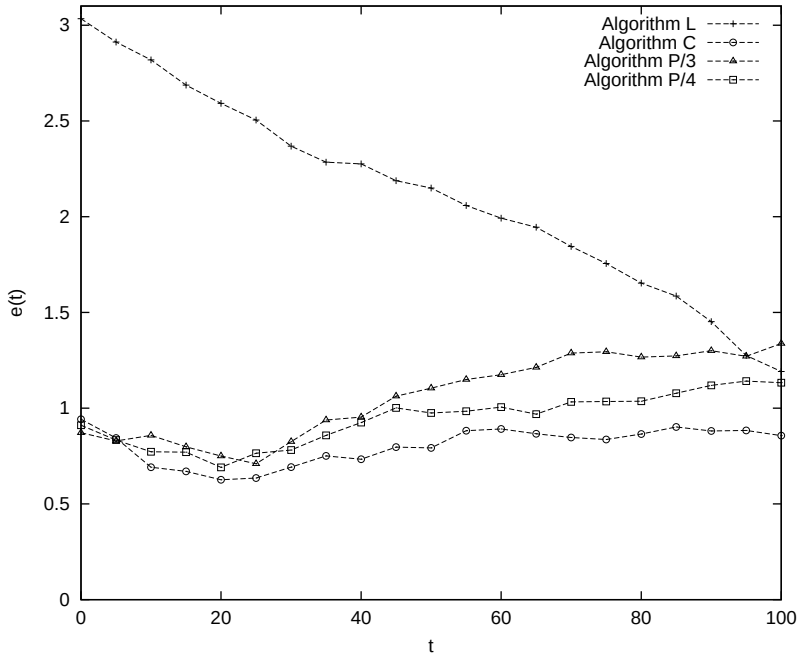

Figure 21: Error with $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15°$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

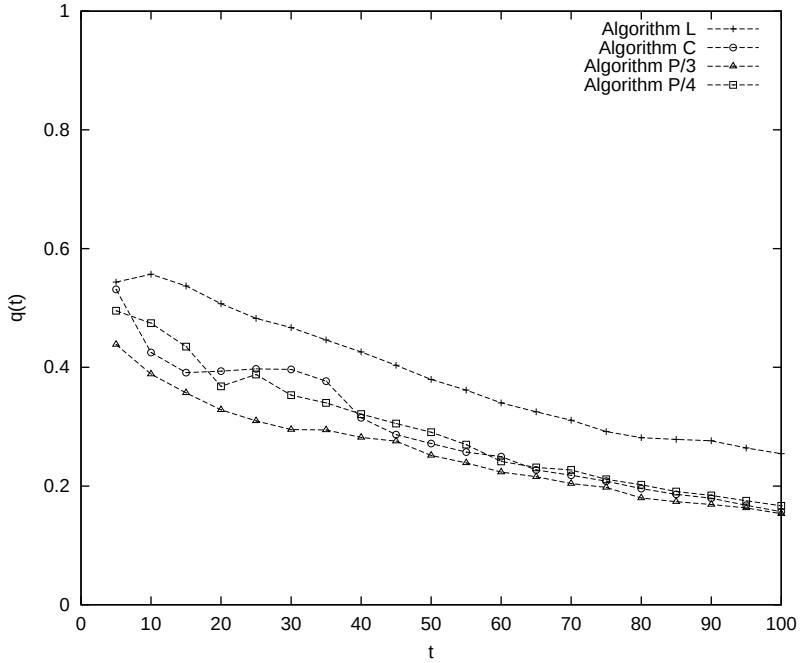

Figure 22: Efficiency with $\sigma_{E_{r,g,b}} = 40$, $\sigma_{E_\theta} = 15°$ and $\sigma_{E_{V_x,V_y}} = 0.05$.

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

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

5 Acknowledgements

I would like to thank Prof. Svetha Venkatesh and Dr. Mohan Kumar for their valuable assistance in designing the experiments and preparing this manuscript.

References

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

[2] T. Balch and R. C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998. URL `http://www.cc.gatech.edu/ai/robot-lab/online-publications/formjour.pdf`.

[3] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27, March 1997. URL `http://www.bol.ucla.edu/~fukunaga/cooperative-robots-survey-journal.pdf`.

[4] Q. Chen and J. Y. S. Luh. Coordination and control of a group of small mobile robots.

In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2315–2320, 1994.

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

[6] X. Défago and A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proceedings of the 2nd ACM Annual Workshop on Principles of Mobile Computing (POMC'02)*, pages 97–104, Toulouse, France, October 2002. URL `http://www.jaist.ac.jp/~defago/files/pdf/DK02.pdf`.

[7] Deborah Estrin, Ramesh Govindan, and John Heidemann. Embedding the Internet. *Communications of the ACM*, 43(5):38–41, May 2000. URL `http://www.isi.edu/~johnh/PAPERS/Estrin00a.pdf`.

[8] P. Flocchini, G.Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2000)*, pages 480–485, Dearborn, MI, USA, October 2000. URL `http://sbrinz.di.unipi.it/~peppe/Papers/IV2000.ps`.

[9] P. Flocchini, G.Prencipe, N. Santoro, and P. Widmayer. Pattern formation by autonomous robots without chirality. In *Proceedings of the VIII International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001)*, pages 147–162, June 2001. URL `http://sbrinz.di.unipi.it/~peppe/Papers/Sirocco01.ps`.

[10] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Proceedings of the 10th International Symposium on Algorithms and*

Computation (ISAAC 99), volume LNCS 1741, pages 93–102, December 1999. URL http://sbrinz.di.unipi.it/~peppe/Papers/ISAAC99.ps.

[11] J. Fredslund and M. J. Matarić. A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, October 2002. URL http://www.daimi.au.dk/~chili/Formations/journalpaper.pdf.

[12] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000. ISBN 1-58113-197-6. URL http://doi.acm.org/10.1145/345910.345920.

[13] Justin T. C. Lee. Distributed motion coordination for mobile wireless sensor networks using vision. Master's thesis, Department of Computing, Curtin University of Technology, Perth, Australia, March 2003. URL http://adt.curtin.edu.au/theses/available/adt-WCU20031201.132347/.

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

[15] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. In *Proceedings of the Fourth European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, pages 185–190, May 2001. URL http://www.cs.unibo.it/ersads/papers/prencipe.ps.

[16] G. Prencipe and V. Gervasi. On the intelligent behavior of stupid robots. In *Proceedings of the AI*IA Workshop on Robotics (GLR)*, September 2002. URL http://www-dii.ing.unisi.it/aiia2002/paper/ROBOTICA/gervasi-aiia02.pdf.

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

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

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

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