SELF-ASSEMBLY AND SELF-HEALING FOR ROBOTIC COLLECTIVES

by

Michael Rubenstein

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
COMPUTER SCIENCE

December 2009

Acknowledgments

First, I would like to thank my advisor, Dr. Wei-Min Shen, whose encouragement, advice, and support was essential for this thesis and the PhD. process in general. Next I would like to thank Dr. Ari Requicha, Dr. Cheng-Ming Chuong, and Dr. Peter Will, for their valuable feedback on this work. Thanks to Nadeesha, Jacob, Harris, Feili, and the other students at the polymorphic robotics lab for their willingness to listen to me and help when I was stuck on problems. Thanks to Wei-Min and Elise for the long hours of help with editing and grammar. Finally, thanks to my parents, family, and friends who encouraged me and made this all possible.

Contents

List of Figures

List of Tables

Abstract

In a large group of robots, which in the scope of this thesis is called a collective, the capabilities of the collective are often greater than the sum of the capabilities of the individual robots that make up that collective. This additional capability can be the result of the overall collective shape, and/or the robots' changing their behavior based on their location within the collective, defined as differentiation. These properties of the collective, namely the shape and the overall differentiation of robots in the shape (the desired spatial-temporal role pattern), can add capabilities to a collective, and therefore they are important to control. In the event that the shape of the collective and/or the spatial-temporal role pattern is damaged, then the collective can lose some or all of the capabilities gained by those properties. If the collective could re-gain the lost properties through self-healing, then it is possible to recover the lost capabilities as well. This self-healing could be possible if the robots are capable of moving to a new location within the collective, as well as changing their differentiated behavior.

In this thesis, a distributed control method called DASH (distributed self-assembly and self-healing) is presented, which empowers a collective of robots to robustly and consistently form and maintain a pre-defined shape and spatial-temporal role pattern. Furthermore, a control method called S-DASH (scalable distributed self-assembly and self-healing) allows the shape and spatial-temporal role pattern to be formed at a scale proportional to the number of robots currently in the collective. If this collective shape or spatial-temporal role pattern is damaged through the un-controlled movement, removal, or addition of some members of the collective, the existing members will recover the desired shape and spatial-temporal role pattern proportional to the new number of robots in the collective. The control methods are analyzed in terms of the class of acceptable shapes, how well it scales to the number of robots in the collective, and convergence to the desired shape and spatial-temporal role pattern. The control methods are then demonstrated in a simulated collective of simple robots.

Chapter 1: Introduction

This thesis presents a method for controlling a large group of identical robots in order to form (self-assemble), hold, and repair (self-heal) the shape of a group of robots, called a collective. This shape is at a size proportional to the number of robots in the collective (scalable). This thesis also presents a method to allow individual robots to take on roles based on their location within that shape (differentiation) and time. Both shape and differentiation are important collective properties, giving the collective of robots capabilities beyond that of the individual robots. The assumptions about these robots and their interaction with the environment are partly based on biological cells, and allow for a simple, low cost system. With these assumptions, the robots have only: local sensing of other robots, no *a priori* knowledge of starting location or orientation, and have the ability to move along the 2-D planar environment. To keep the robots simple, they are homogeneous, with identical programs and capabilities, including the lack of an ID, or unique name.

The work in this thesis can be divided into four main tasks. The first task is for the collective to self-organize a consistent coordinate system. The second task is for the collective to self-assemble into a pre-defined shape from a random starting configuration, as well as self-repair that shape in the event of damage to the collective such as the moving, addition or removal of a subset of robots. The third task is for the collective to determine the scale of the shape that is proportional to the number of robots currently in the collective. The fourth task is for an individual robot to choose a role based on its location within the shape, and with respect to time (spatial-temporal differentiation).

The first main task, described in chapter 5, is for the collective to self-organize a consistent coordinate system. The purpose of this coordinate system is to determine where the desired shape should be formed in the environment, and for each robot to know where it is located with respect to that desired shape. In this thesis, the coordinate system is formed in three stages. In the first stage, local seed robots are elected. In the second

stage, trilateration is used to develop many local coordinate systems, one for each local seed. In the final stage, these local coordinate systems are merged together to form a single consistent coordinate system. This thesis also presents a method which will allow robots to move in the environment without negatively affecting the coordinate system.

The second main task, self-assembling and self-healing the shape of a collective, is accomplished using a control method called DASH (distributed self-assembly and self-healing), described in chapter 6. DASH is a control method that runs on each individual robot in the collective. It takes as a input the desired shape of the collective, given to it in the form of a picture or pixel map. Given the desired shape, DASH will control the movement of each robot so that the robot will move into the shape while not blocking other robots from doing so. To accomplish this, DASH uses the location of the robot in the coordinate system, communication between neighboring robots, and a desired scale provided by S-DASH.

S-DASH (scalable distributed self-assembly and self-healing) is the third main task of this thesis, described in chapter 7, which adapts the scale of the shape so that it is proportional to the number of robots in the collective. This allows the collective to form the desired shape, while at the same time maintaining the density of robots in the collective. S-DASH will constantly update the scale of the shape, so that even in the event of the removal or addition of robots to the collective, the size of the shape will stay proportional to the current number of robots.

The fourth task for this thesis, described in chapter 8, is a distributed method to allow each robot to choose a role based on its location in the desired shape and time. The overall choice of roles of all robots in the shape is called the spatial-temporal role pattern. The desired pattern is given to each robot in the form of an image or pixel map, called the spatial-temporal role map. Using this spatial-temporal role map, the desired scale of the shape, and the robot's location in the coordinate system, this method allows each robot to choose a role that properly contributes to the desired pattern. Furthermore, if the location of a robot within the shape changes, for example due to damage, then the robot's role will also change to a role appropriate to its new position.

Finally, in chapter 9, this thesis provides examples of these methods running in a simulated robotic collective. This simulation demonstrates the collective forming shapes and spatial-temporal role patterns for various desired shapes. It also demonstrates the collective repairing its shape and spatial-temporal role pattern after various types of damage to the collective.

Chapter 2: Problem Statement

As its main objective, this thesis intends to solve two main problems related to robotic collective control. For both of these problems, and in the scope of this thesis, each robot is first given the desired shape, as well as the desired spatial-temporal role pattern (in the form of a spatial-temporal role map). The first problem is to produce a distributed control strategy for individual robots in the collective to self-assemble the desired shape and spatial-temporal role pattern. The second problem is to produce a control strategy for individual robots to self-heal the collective's shape and spatial-temporal role pattern in the case of damage to the collective. These two objectives will be combined together to produce a single control strategy that can both self-assemble and self-heal a pre-defined shape and spatial-temporal role pattern in a robotic collective. The capabilities of this combined control strategy are shown in Fig. 2.1. In this figure, a robot is represented by a circle, and its role is represented by the color of the circle. In Fig. 2.1(top), an initially random collection of robots self-assembles the desired star shape with a desired spatial-temporal role map where robots on the outside of the star take on a "red" role, and the robots on the inside of the star take on a "blue" role. In Fig. 2.1(bottom left), the collective is damaged by removing half of the star. Fig. 2.1(bottom) shows that even after this damage, the collective can autonomously reform the desired shape and spatial-temporal role pattern using the remaining robots in the collective.

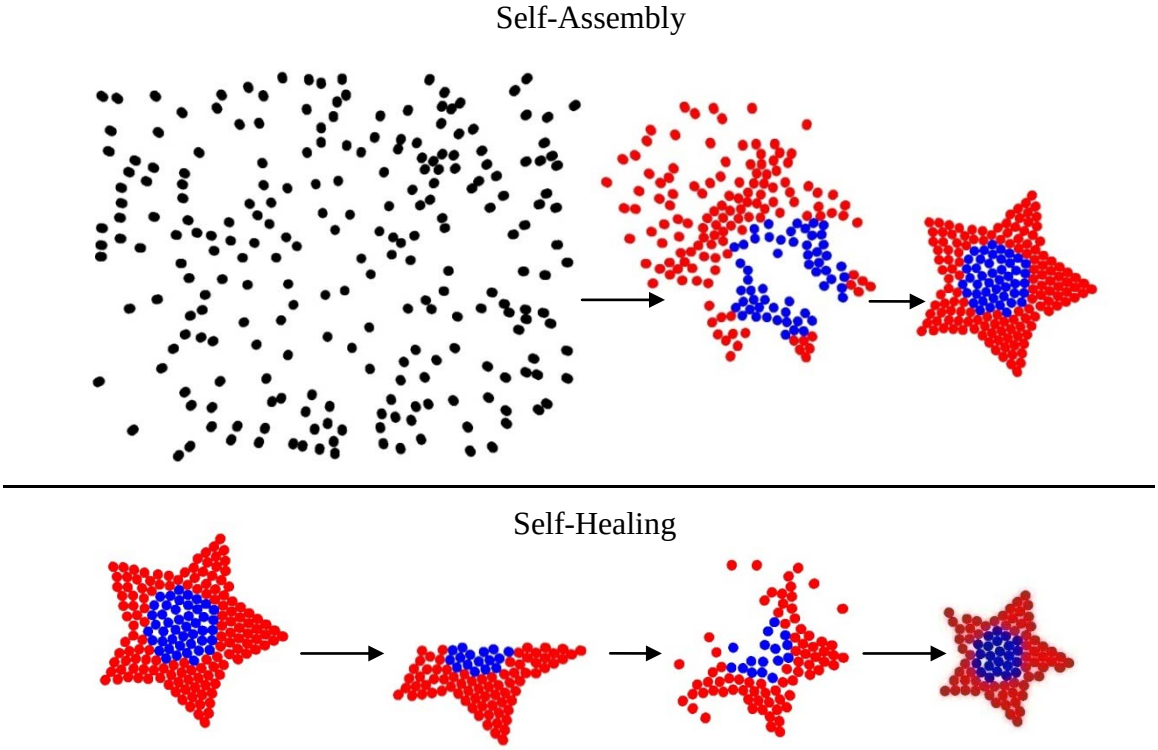


Figure 2.1. A demonstration of the self-healing and self-assembling capabilities of this combined control strategy. (top) A differentiated star shape is self-assembled. (bottom) That star is damaged and then self-heals to a smaller scale of the complete shape and role pattern.

The usefulness of these robotic collectives comes from their ability to develop *capabilities* which are not possessed by any of the individual robots that make up the collective. These capabilities arise from *properties* of the collective itself, and not directly from the properties of any one robot. One such property of a collective that gives it capabilities beyond that of the individual robots, is its shape. For the scope of this thesis, the shape of a collective is defined as the outline of the robotic collective, without any specific size. Another such property of a collective is the spatial-temporal role pattern, where all robots in the shape take on specific roles depending on their location within the collective.

2.1 Importance of Shape

The general idea of collective shape can be thought of as analogous to the shape of an animal, but composed of robots rather than cells. Often, the shape of an animal

plays a critical role in that animal's capabilities. For example, the shape of a bird helps give it the capability to fly, while flight is obviously not a capability of its individual cells. Similarly, the shape of a robotic collective plays a role in the collective's own capabilities, so it may be desired that the collective also be able to form and hold the property of shape, in order to maintain its capabilities. For example, a simple cube shaped robot (Fig. 2.2(A)) is not capable of holding water; however, if it is part of a collective that forms a water tight shape (Fig. 2.2(B)), such as a cup, then the collective of these robots gains the capability of holding water from its collective shape. In a second example, imagine a single SWARM-BOT (Sahin, et al. 2002) reaches a canyon-like obstacle with a goal on the other side (Fig. 2.3(A)). By itself, a single SWARM-BOT is not capable of crossing the canyon to reach the goal on the other side. However, if the SWARM-BOT joins a collective of other SWARM-BOTs, and forms a collective shaped like a bridge, the collective's shape enables it to cross the canyon and reach the goal (Fig. 2.3(B)). In a third example, a single Superbot robot (Chiu, Rubenstein and Shen 2008) needs to locomode as far as it can until its battery pack empties. As a solitary Superbot (Fig. 2.4(A)), it can only travel 200 meters until the battery is empty. If this single Superbot can form a collective with five other Superbot robots in the shape of a wheel (Fig. 2.4(B)), then it can move over 1000 meters until its battery pack depletes. In this case, the shape of the Superbot collective enables the collective to travel at least five times as far as any single Superbot can travel.

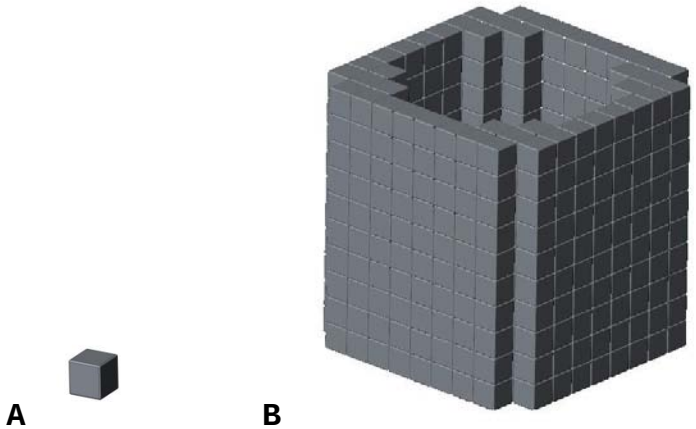


Figure 2.2. (A) A simple cube shaped robot. (B) A collective of cube shaped robots forming a cup. (Photo made with assistance from Akiya Kamimura)

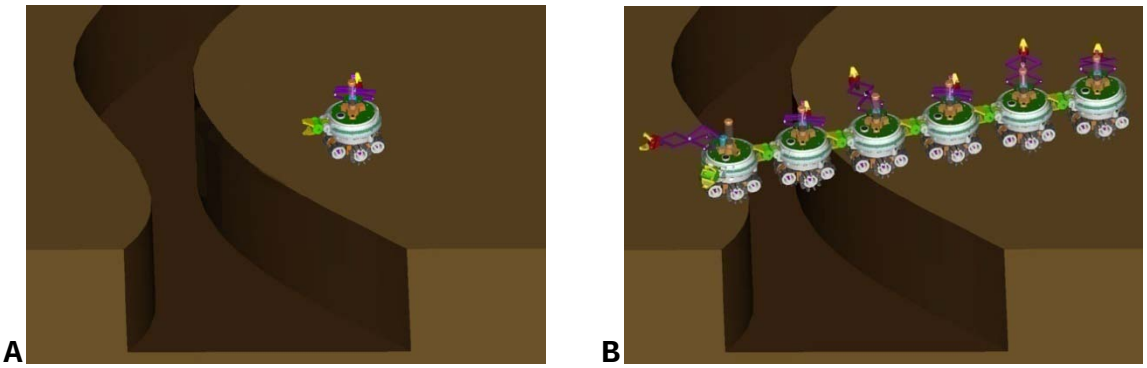


Figure 2.3. (A) A single SWARM-BOT with an insurmountable obstacle. (B) A collective of SWARM-BOTs crossing the same obstacle. From (Sahin, et al. 2002).

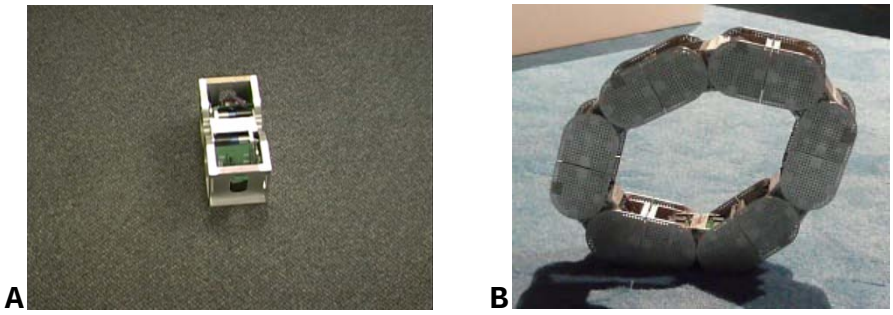


Figure 2.4. (A) A solitary Superbot module. (B) A wheel made of 6 Superbot modules.

2.2 Importance of Role

Besides shape, another way a collective achieves capabilities beyond that of the individual robots is when all robots take on a specific role, i.e. differentiation, according to a spatial-temporal role map. This differentiation does not cause the collective to lose its homogeneity just as a group of homogenous state machines remain homogenous even if they are not all in the same state at a given time. Differentiation is common in nature, such as when stem cells differentiate into distinct cell types such as skin cells, depending on their location within the body. This arrangement of cells allows skin cells to be located on the surface of the body where they are most effective at one of their main jobs, protecting the inside of the body from invaders. If, instead of arranging themselves on the surface, skin cells were just randomly distributed throughout the body without any regard to location, then they would lose their effectiveness.

In a second example of differentiation, consider an ancient military formation called the "testudo" or tortoise shell (Fig. 2.5). In this example, a collective of soldiers differentiate into two location-dependant roles. The spatial-temporal role map for the testudo formation dictates that the soldiers in the front row hold their shields in front of themselves, protecting against frontal projectile attacks, and that the soldiers in the back rows hold their shields on top of themselves, protecting against attacks from above. If the soldiers did not follow the spatial-temporal role map, then some soldiers in front would hold their shields above themselves, and some soldiers is the back rows would hold their shield in front of themselves. It is obvious that in this example, capabilities beyond that of an undifferentiated collective, or a single soldier, are developed by the collective when all soldiers follow the spatial-temporal role map.

Figure 2.5. An ancient military formation called the "testudo". Photo by Neil Carey, 2006.

This power from differentiation of cells in a biological organism and soldiers in a testudo formation can be used similarly in a robotic collective, making the collective more effective at a given task when it differentiates based on a spatial-temporal role map. For example, consider a collective of robots tasked to monitor the environment and then share that data with each other. In this collective, it might be useful for robots to differentiate into two roles, according to a spatial-temporal role map. This spatial-temporal role map would direct the robots on the periphery of the collective to choose a role dedicated to sensing the environment, where contact with the external environment is common. The spatial-temporal role map would also direct robots in the interior to choose a role dedicated to communication between sensing robots, where communication links tend to be shorter and contact with the external environment is limited. By following this spatial-temporal role map, robots will increase the effectiveness of the collective for its desired task.

This spatial-temporal role map can be extended to consider not only spatial information about where the robot is within the collective, but also temporal information. This extension of the spatial-temporal role map gives the collective further capabilities by directing individual robot roles to vary based on time as well as space. For example, in (Ishiguro, Shimizu and Kawakatsu 2004), a robot can choose to be in a role where its

body is expanded, or one where it is contracted. In this collective, the robots in the front first expand and then contract. As the front robots contract, the robots behind them expand. Then these robots contract, while the robots behind them contract. This continues along the whole shape, causing a wave of contracting robots to move from front to back. This spatial-temporal role pattern will cause the collective as a whole to move forward in a locomotion called protoplasmic streaming. In this case, it is easy to see that if the roles of these robots were static, the collective would not move; only when the role is allowed to vary in time does the collective gain the ability to move.

2.3 Importance of Self-Assembly and Self-Healing

Now that two collective properties which give additional capabilities to a collective have been identified, it would be useful for the robots to produce a collective with a pre-defined shape and a pre-defined spatial-temporal role pattern within that shape. Imagine there is a group of robots on the floor, randomly scattered without any particular shape. This is a typical starting situation for a robot collective. Upon turning on the robots, they are assigned to a goal through whatever means. This goal is something that no single robot is capable of completing individually; however, the robots are given a shape and spatial-temporal role map that would allow them, as a group, to complete the given goal. If they could form that shape and spatial-temporal role pattern from their initially random configuration (self-assembly), then they could use the additionally gained collective capabilities to complete the assigned goal. An advantage of self-assembly is that it is autonomous and independent of human input, allowing a collective to form a shape and spatial-temporal role pattern even in locations that are expensive or dangerous for humans to reach, such as space or deep-sea environments. A second advantage of self-assembly is if the number of robots in the collective is large, it would be difficult and time consuming for a human operator to assign locations and roles for each individual robot in the collective.

After a collective self-assembles into the desired shape and spatial-temporal role pattern, it would be desirable if the collective maintained these properties in order to maintain the collective capabilities gained. If an external action caused a perturbation of

some of the robots in the collective, changing its shape and/or its spatial-temporal role pattern, it is conceivable that this action (i.e. damage to the collective) could diminish or completely remove the capabilities gained from the collective properties. Some possible external actions considered in this thesis could include: removal of some robots, changing the location of some robots, or even the addition of some robots. If the collective could re-gain (self-heal) its shape and spatial-temporal role pattern after this damage, then it would also re-gain its capabilities gained from those collective properties. Similar to the case of self-assembly, self-healing also gives the collective the advantage of repair without human interaction, or when there are a large number of robots in the collective.

2.4 Requirements

In the natural world, self-assembly, and to a lesser degree self-healing, occur in many different systems, over many different physical and temporal scales. These natural systems not only provide an existence proof that self-assembly and self-healing is possible, but can also give a grounding as to what is required of the individual components that are part of the self-assembling and self-healing system. One example of these natural world systems is a biological animal, composed of cells. The cells in these animals are simple in terms of their individual capabilities. For example, there are many biological systems made of cells that are nearly identical to each other in every way, defined as homogenous. These identical cells even lack globally unique names or IDs. Furthermore, in these biological systems, each individual cell has limited sensing range, only capable of sensing nearby neighbors over a small portion of the entire collective. Each of these cells also has limited computational power, which is not capable of storing a large amount of information, or making many complex decisions in quick succession. Even with these limitations on the individual cell, there are many examples of self-assembly and self-healing in biological systems.

For the scope of this thesis, the capabilities of the individual robots in the collective will be partly modeled after these capabilities of cells in biological systems, and will also be constrained to the possible capabilities of simple mass-produced modern robots. For example, all the robots used to make up a collective will be identical to each

other in every way. This homogeneity of the robots includes hardware and software, and they even lack a unique global ID. Every robot in the collective has its own on-board computing power and can only communicate with nearby neighbors, making the collective distributed in nature. In addition, these robots tend to have very simple sensors only capable of sensing within a limited range, nothing complex enough to directly gather global information. These limited sensing capabilities constrain the robots to knowledge only about their local neighborhood, and they have no direct way of sensing beyond this neighborhood, making it difficult to react to events outside the robot's sensing range.

While collectives consist of large numbers of simple robots which individually have limited capabilities, it should now be apparent that if the robots can cooperate by forming a shape, as well as differentiating within that shape, it is possible to increase the capabilities of the collective beyond that of any individual robot. These collective capabilities can even be recovered after damage to the collective, such as robot removal, addition, or movement. This recovery of capabilities is a result of the collective re-forming its shape and spatial-temporal role pattern, even after damage has occurred.

Chapter 3: Challenges and Related Work

In order to achieve the described self-assembly and self-healing in a collective, a number of challenges and design questions must be addressed. In this chapter, some of these challenges and questions, as well as previous approaches to solving them, are discussed. These approaches stem mainly from biological, robotic, and other artificial systems.

3.1 Distributed Control

One challenge is how to design a controller for a distributed robotic collective. This can be difficult due to the distributed nature of the collective. There are three main types of controllers for a distributed system: centralized, distributed, and hybrid controllers.

One type, a centralized approach, is to elect a single leader, or provide a pre-selected external leader, and propagate the information needed to make control decisions to that leader. This leader will then send the control decisions to the relevant robots in the collective. One example of a centralized approach for assembly can be found in (Gilpin, et al. 2008). In that approach, the location of each robot is first sent to an external computer running a Matlab control program. In this program, the decision is made whether each robot is inside or outside the shape. If the robot is outside the shape, then the computer commands that robot to separate from the rest of the robots. While this centralized approach is generally easier to conceive and implement, it suffers from some major disadvantages. The first disadvantage is that the leader represents a single point of failure, so that if it is removed or damaged, the collective would, at least temporally, not function as desired. A second disadvantage is that in general, the number of messages required between robots to get information to and from the leader scales poorly as the number of robots increases.

The second type of controller for a distributed system is a distributed controller, in which each robot concurrently makes decisions for the collective, and the end result of all

the robots' decisions gives the desired collective behavior. For example, in (Spears and Gordon 1999), each robot moves away from its neighbors if it is too close, and towards them if it is too far away. These simple rules allow the collective to form a hexagonal lattice shape. To follow the rules for this example, a robot only needs to know the distance and bearing to its immediate neighbors, nothing more. Another example is in (Shen, Will, et al. 2004), where each robot only communicates with its neighbors, and then makes decisions based on that communication and its current state. This work also has very limited shapes that it can form. The distributed type of collective controller has the advantage that there is no single point of failure, making it more robust to damage. In general, the disadvantage to these controllers is that they can be more difficult to conceive and implement, especially if more complicated shapes are desired.

There is also a third type of controller, a hybrid between centralized and distributed. This hybrid controller requires a "seed" robot to start the controller, and once the controller is started, this seed is no longer needed. For example in (Stoy and Nagpal 2004), and (Arbuckle and Requicha 2004), a single robot is required to start building a coordinate system which guides the assembly process. Once the coordinate system has been started, both approaches no longer require a seed, and it can be replaced if necessary. This requirement of a "seed" presents a single point of failure for the collective, if only for a small amount of time. If the seed is damaged in the beginning of self-assembly, the controller will not produce the desired results. If the seed is damaged after the beginning of self-assembly, the controller will still produce the desired results.

3.2 Generality of Shapes

A second challenge is how to make the controller as general as possible in terms of the types of shapes that can be self-assembled and self-healed. The controller should be capable of forming as many shapes as possible, because the more shapes that the controller can produce in the collective, the more general the controller becomes. In (Spears and Gordon 1999), the collective is only capable of forming simple shapes and lattice patterns, nothing more complicated. Some approaches can only produce simple shapes described by a function. For example, in (Rubenstein and Shen 2008), the

function must be polar, and in (Hou, Cheah and Slotine 2009), they can form simple squares and circles. In (Yim, Lamping, et al. 1997), their method is capable of producing any shape that doesn't have an empty space that is surrounded by robots. An even more general approach is shown in the work by (Cheng, Cheng and Nagpal 2005), as well as (Kondacs 2003), which are capable of forming any collective shape. One drawback is that the size of the shape is predetermined, so if there are not enough robots for that size, the shape cannot be formed or repaired.

3.3 Generality of Spatial-Temporal Role Patterns

Similar to the generality of shapes, a third challenge is to make the controller as general as possible in terms of the spatial-temporal role patterns that can be produced. One example of a limited spatial-temporal role pattern can be found in (Sahin, et al. 2002), where a group of robots can form spatial-temporal role patterns such as a yellow circle with a blue center, or a rectangle with alternating blue and yellow stripes. Another example of limited possible spatial-temporal role patterns, but this time in biological cells, is shown in (Jiang, et al. 1999). In their work, it is shown that biological cells can form a simple feather bud spatial-temporal role pattern, independent of the cells' starting positions. A more complicated spatial-temporal role pattern can be formed in (Nagpal 2001), where a sheet of simulated cells can fold its shape into a pre-defined pattern, and then use this shape to produce a spatial-temporal role pattern on the sheet. This approach can form complex patterns on the sheet of cells, such as an electrical inverter schematic. Another example of a complex spatial-temporal role pattern formation is shown in (Shen, Salemi and Will 2002), where a collective of robots connect to form a modular robot collective. Each robot then chooses its role based on which neighbors are connected to it, and in what way are they connected. This form of differentiation allows a robot in the leg of the collective to behave as a leg, while a robot in the spine behaves like a spine, etc. A difficulty with all of these approaches is that they cannot form any general spatial-temporal role pattern without a human to hand-design the rules for forming the pattern. It would be more general if, given any spatial-temporal role map, the collective is capable of autonomously forming the spatial-temporal role pattern without human input.

3.4 Self-Heal Damage

A fourth challenge is to self-heal any damage applied to the collective shape and/or spatial-temporal role pattern. There are many types of damage that could occur in a robotic collective. For example, additional robots could be added, or some robots could be removed from the collective. More types of damage could include taking some robots and moving them from one location to another in the collective. Furthermore, robots could be partially damaged, by allowing communication but preventing movement, or vice versa. In one example, an approach with limited self-healing (Rosa, et al. 2006) cannot reform a desired shape if some robots are added or removed. This is because their approach requires knowledge of the initial starting positions of robots in the collective in order to produce the robot controller. If some robots are added or removed, then the initial starting positions of the robots are different than their positions after damage, and hence the controller cannot reform the shape. In the approach by (Yim, Lamping, et al. 1997), the shape can be recovered if robots are added to the collective; however, it cannot recover if robots are removed from the collective or repositioned. In (Stoy and Nagpal 2004), the shape can be healed if robots are added or removed from the collective; however, if a robot is moved from one location to another, then this approach will not be able to re-form the shape. In (Cheng, Cheng and Nagpal 2005), the collective is fully capable of reforming the desired shape in the event of the addition, removal, or movement of robots; however, if robots are partially damaged the collective cannot reform. One distinction to note is that, for this approach, the number of robots added to the collective has an upper limit, due to the collective reaching a maximum density of robots.

3.5 Determining the Scale of the Shape

A fifth challenge is to determine the size (scale) of the collective shape. During the self-assembly process, or in certain types of damage, namely the addition and subtraction of robots, there are two options for the collective to choose a shape scale. Those options are fixed scale self-healing, or scalable self-healing.

The first option for determining the scale of the collective shape, fixed scale self-healing, is used in (Arbuckle and Requicha 2004). This method keeps the size of the shape the same, but either changes the density of robots, as in (Cheng, Cheng and Nagpal 2005), or grows new replacements, like (Kondacs 2003). Current robotic technology can only self-replicate in very controlled environments, for example (Eno, et al. 2007), so growing replacement robots is generally not yet feasible. In biological systems however, growing replacements is possible, and is seen in many cases of animal self-healing. A popular example of this type of self-healing is the starfish, more specifically the species *Linckia Diplax***.** If the starfish's arm is cut from the body, the amputated arm will re-grow a body, and the body will re-grow a new arm, resulting in two complete starfish from the original one (Kellogg 1904). Similarly, when the leg of a salamander is amputated, the salamander can re-grow the missing leg; however, in the case of the salamander, the leg will not re-grow a new body. With regards to the approach of changing the density of robots in a collective, there is an upper limit to this robot density (one can only fit so many robots in a fixed area), so there is a maximum number of robots that can fit inside the collective shape. If the collective grows beyond that number, it cannot correctly form the desired shape. Another drawback to the idea of changing robot density is that, for many collective robotic systems, such as reconfigurable robots (Yim, Shen, et al. 2007), the robots require a close physical connection to neighboring robots. This means that in general, it is advantageous for the density of robots in the collective to remain constant, irrespective of the size of the collective.

The second option for determining the scale of the collective shape is to adapt to the number of robots in the collective, known as scalable self-healing. This option adjusts the size of the shape proportional to the number of robots in the collective, keeping the robot density constant. This scalable self-healing is seen in nature, for example (Bode 2003), where a small invertebrate, the hydra, will reform its original shape after being cut in half, but at half of the original size. The hydra does this using morphallaxis, where the remaining cells move to repair the damaged shape. The hydra's use of morphallaxis is so good at self-repair, that it is suspected that the hydra can live forever (Martinez 1998). Since it does not require the production of more robots,

scalable self-healing lends itself nicely to robotic collectives, and has been recreated in robotic collectives, for example (Stoy and Nagpal 2004). A problem with this option is that because the scale is proportional to the number of robots in the collective, the robots must either directly or indirectly determine this number. This number is hard to find in a distributed collective, especially if the number of robots can change unexpectedly at any time.

3.6 Robot Identification

The sixth challenge is how robots identify themselves. There are three main options for the robots identify themselves: with globally unique identifiers (ID), with locally unique identifiers, or without any identifiers at all.

The first option is to use a globally unique identifier. With globally unique identifiers, the ID is guaranteed to be unique for all the robots in the collective. This form of identifier is useful in robot-to-robot communication, leader selection, and negotiation between robots. Since global ID is so useful, it is no surprise that it can be found in many approaches for controlling a robot collective, for example (Yim, Shirmohammadi, et al. 2007), and (Maxim, et al. 2008). The main drawback for globally unique IDs, is that when the robots are made or programmed, special care must be taken that each robot's ID is unique. Furthermore, it is generally accepted that the use of a global identifier is not used in biological cells.

The second option is to use a locally unique ID. Using this option, the unique ID is only guaranteed to be unique for a certain distance, usually one or two communication distances. As shown in (Nagpal 2001), this ID has the advantage that it can be created probabilistically at runtime. While this option is useful for negotiation between neighboring robots, it has the drawback that it is not very useful for leader selection or long distance communication.

The third option is to simply not require any ID for the robots in the collective. One advantage of not requiring an ID is that it can simplify the robots used by removing one requirement of those robots. Another reason not to use ID, is that is does not seem to be used in nature, which means that ID-free methods to control a robotic collective may

provide insight into how natural systems self-assemble and self-heal. Some examples of approaches that can self-heal and/or self-assemble even without ID include (Cheng, Cheng and Nagpal 2005), (Shen, Will, et al. 2004), and (Rothemund 2006).

3.7 Limited Sensing

A seventh challenge is that there is no guarantee that robots can perceive everything in the environment. The amount of sensor perception can vary from no perception to full perception. With no perception, each robot is only aware of its internal state, nothing about its external environment is known. With full perception, each robot is aware of the precise position, bearing, and orientation of all other robots in the collective with respect to itself. While it is ideal for each robot to have full perception of all relevant factors in its environment, this does not scale well as the number of robots becomes large. If an approach uses less than full perception, then it becomes more difficult for the robots to react to events, such as damage, which occur outside its perception.

In the approaches taken by (Nagpal 2001), (Kondacs 2003), as well as in simple cellular biological systems, each robot (or cell) has a more limited sensing capability. In these cases, each robot can only perceive neighboring robots that are within a range R. If a neighbor is closer to a robot than R, then that robot can measure the distance to that neighbor, but bearing is not sensed at any range, except perhaps for the biological cells. If the neighbor is farther than R, then it cannot be directly sensed by that robot.

In other approaches, such as (Arbuckle and Requicha 2004), (Shen, Will, et al. 2004), and (Stoy and Nagpal 2004), the sensing range of a robot is further limited to only neighbors that are in direct contact with that robot. In these approaches, the sensors can detect distance (always the distance between two touching robots), as well as bearing to the neighboring robots. This very limited sensing makes the robots more dependent on information gathered by communicating with neighboring robots.

3.8 Limited Communication Bandwidth

An eighth challenge is that communication bandwidth between neighbors is limited. Previous approaches have used different amounts of information communicated

among neighboring robots, varying from no information, all the way to large amounts of information. This information is generally communicated to the same group of neighboring robots that are within sensor range. Some approaches use no communication, such as (Spears and Gordon 1999) and (Shen, Will, et al. 2004), where only sensor values are used for controlling the robots. This results in very limited coordination between robots, which limits the types of shapes that can be formed by the collective. In (Cheng, Cheng and Nagpal 2005), a small amount of information is sent between robots in the form of each robot's location (x,y values). In other approaches, for example (Suzuki, et al. 2007), large amounts of data are communicated to neighboring robots. This information can include state information, sensor information, and robot coordinates. There are two main drawbacks with sending a large amount of information. The first drawback is that it increases the time and complexity of robot-to-robot communication. The second drawback is that it increases the amount of memory required for each robot to temporarily store the communicated information.

3.9 Robot Location

A ninth challenge is for robots to know where they are located, with respect to the desired shape, as well as neighboring robots. In many approaches, a consistent (accepted by all robots in the collective) coordinate system is used by each robot to aid in forming the desired shape. This coordinate system allows each robot to know its current location with respect to its neighboring robots, as well as where it is located in the desired shape. For example, in (Stoy and Nagpal 2004), an initial seed starts its coordinates at (0,0). Then, for all robots who see a neighbor with an already determined coordinate, they set their own location to be the coordinate of that neighbor, plus the vector between them and that neighbor. Eventually, all the robots will have a coordinate that corresponds to the vector between them and the seed robot. In this approach, the shape is known to be centered about the coordinate (0,0). This approach is a hybrid controller, and as discussed before, sufferers from a single point of failure. Another approach to developing a coordinate system can be found in (Grabowski and Khosla 2001). In this approach, each robot tries to minimize the difference between the measured distances

from its neighbors, and the distance between it and its neighbors based on their guessed locations. The minimization is done by updating its own guessed location. Over many iterations, the guessed locations for all the robots in the collective will converge to a consistent coordinate system. A difficulty with this method is that global errors in the coordinate systems are possible for some collective layouts. There are also approaches that don't use any coordinate system, for example in (Spears and Gordon 1999), and (Shen, Will, et al. 2004); however in this work, as a result of no coordinate system, the types of shapes that can be formed are limited.

3.10 Robot Movement

A tenth challenge for self-assembly and self-healing in a collective is deciding how to move each robot in order to assemble or heal the collective shape. Each robot must move to a location within the desired shape while at the same time avoiding other robots in the collective. Each robot must also try to move to a space in the desired shape that is not occupied by another robot. In many self-assembly approaches involving robot movement, if a robot does not take care in avoiding neighbors, then it is possible that some robots may become stuck outside the desired shape, or in an undesired location inside the shape. One way to avoid blocking by neighbors is to use a gas model for robot movement, for example in (Cheng, Cheng and Nagpal 2005). This gas model pushes robots away from each other if they are too close, avoiding collisions. Another way to avoid collisions is used in (Yim, Lamping, et al. 1997), where each robot reserves a goal location prior to moving to it. This way there is only one robot moving to a particular goal, which helps prevent overcrowding. A drawback to this method is that it will prevent self-healing in the case of robot movement. Another way is to use random movement to prevent robots from being stuck in undesired locations for too long, for example in (Zykov and Lipson 2007), (Bishop, et al. 2005). With this method, it may take a long time for stuck robots to become unstuck, increasing the time taken to self-assemble.

These ten challenges described above may not be the only challenges for controlling a robotic collective; however, they are all important to consider when designing the controller.

Chapter 4: Approach Overview

This chapter gives an overview of the approach this thesis will take towards solving the described self-assembling and self-healing problems. For that purpose, it is important to first describe what assumptions are made regarding which properties of the robots can be used, and their environment. Next, an overview will be given of the general approaches taken by this thesis to overcome the described challenges for the self-assembling and self-healing problems, given those assumptions about the robots and their environment.

4.1 System Properties

The control methods described in this thesis are designed to run separately on each individual robot of a robotic collective. In order for these control methods to operate as desired, the individual robots must meet some basic requirements. As described in the problem statement in chapter 2, these robot requirements are chosen to emulate natural self-assembling and self-healing systems. These requirements also keep the cost-per-robot down, in order to enable large scale manufacturing of these robots.

Each robot that forms the collective should be identical to all other robots in the collective, and all robots should meet the following seven requirements:

1. The first requirement for each robot is that it must be shaped like a cylinder, with radius R_{robot}, and this radius is known to all robots. The robot's environment is a 2D plane, and these cylinders are oriented such that their axis of symmetry is perpendicular to the 2D plane. This requirement can be visualized as analogous to a hockey puck on an ice rink, where the robot is a hockey puck, and the ice rink is the 2D plane.

2. Secondly, each robot must be capable of moving in a straight line to any location on the 2D plane if there are no obstacles or robots in the way. To do this, each robot has a "front" direction, in which the robot can move "forward". The robot can also "rotate" on its axis of symmetry to change the direction that the front of the robot is facing in the environment. With these two degrees of freedom, "forward" and "rotate", all locations on the 2D plane are directly reachable. The speed at which a robot can travel in the "forward" direction is defined as V_{robot}.
3. The third requirement for each robot is that each robot must have its own local clock. This clock runs at the same rate as other robots' clocks; however, it is not required that it be initially synchronized with any other robot's clock. The robot main control loop operates once for every tick of this clock.
4. The fourth requirement for each robot is that it must be able to sense the distance between itself and all neighboring robots that are less than R_{com} distance away. This sensing is updated once every main control loop.
5. The fifth requirement for each robot is that it must have on-board computational power, in order to run the control methods of this thesis. This computation will allow the robot's main controller to run one loop every time step.
6. The sixth requirement is that each robot must have sufficient memory to remember the desired shape, spatial-temporal role map, and temporarily stored data received from neighboring robots.
7. The seventh and final requirement is that all robots must be able to send and receive data to and from all neighboring robots that are located within R_{com} distance. This occurs once for every tick of the local clock.

These seven requirements are summarized in Table 4.1. The robots require nothing other than these seven requirements. No additional sensors such as a GPS, odometer, bump-sensors, cameras, etc. are necessary. Additionally, the robots do not require any unique global ID.

1. Cylinder shape.
2. 2D movement.
3. Local clock.
4. Send and receive data with neighbors.
5. Measure distance to neighbors.
6. Computational power.
7. Sufficient memory.

Table 4.1. A summary of robotic requirements.

In addition to the requirements of the individual robots, there are three assumptions about the environment in which the robots operate. The first assumption is that the environment is a 2D plane with no obstacles. The second assumption is that robots in this environment cannot pass through each other. This means that if a robot moves in a certain direction and collides with another robot, it must go around the other robot in order to continue in that direction. The third assumption about the environment is that the robots start in a randomly placed location and orientation, but in such a way that their communication graph is a well connected graph. If the robots start spaced too far apart, then it is possible that the robots will separate, forming more than one collective. It is important to note that the environment provides no position or orientation information to the robots or their sensors. A summary of these environmental assumptions can be found in table 4.2.

1. 2D plane.
2. Robots collide.
3. Initial start positions and orientations.

Table 4.2. Summary of environmental assumptions.

4.2 General Approach Overview

The approach in this thesis toward solving the self-assembly and self-healing problems can be broken down into four main methods. These four methods, as well as a

method for receiving messages, are executed in the main robot control loop, which runs once per robot clock tick. Hence, in the main robot control loop, the following five methods are executed. First, the robot records any new messages that it has received since the robot last checked its messages at the beginning of the last main control loop. Second, using the method described in section 4.2.1 and chapter 5, the robot updates its location in the self-organizing coordinate system. Third, the DASH controller is executed as described in section 4.2.2 and chapter 6, to form a desired shape at a given scale. Fourth, the S-DASH controller, described in section 4.2.3 and chapter 7, is executed to determine and update the scale used in DASH. Lastly, the robot chooses a spatial-temporal role as described in section 4.2.4 and chapter 8, to form a spatial-temporal role pattern according to the given spatial-temporal role map. A summary of this main controller loop is shown in table 4.3.

1. Record new messages.
2. Update location.
3. DASH.
4. S-DASH.
5. Choose role.

Table 4.3. A summary of the main control loop.

4.2.1 Self-Organizing Coordinate System

This section gives an overview of the first method, which allows all robots to self-discover a consistent global coordinate system. In this thesis, it is critical that each robot knows where the desired shape is located with respect to the robot's current location. Any one robot identifies the desired location for the shape using two numbers. The first number is the distance, D, between that robot and the desired location of the shape's center. The second number is the direction, θ, with respect to the robot's "forward" direction, in which the location of the desired shape's center lies. If the robot were to move from its current position by first rotating θ, and then moving forward distance D, it would place itself where it believes the shape center is located.

Furthermore, it is also necessary that each robot's location for the desired shape is consistent with where all other robots in the collective believe the desired shape is located. To understand this idea of consistency between robots with respect to the desired shape, consider the following experiment. First, all robots in the collective, with their own θ, D values, are commanded to rotate by their own θ, and then move by their value D. In this case, ignoring robot collisions, if all robots were to move to the same location in the environment, then all robots in the collective would have a consistent location for the desired shape.

One way to find the D value for each robot which results in a consistent location for the desired shape among all robots, is to use a coordinate system and assume the shape is centered at its origin. If each robot is given its location (x,y) in the coordinate system, for example using a Global Positioning System such as GPS (Hofmann-Wellenhof, Lichtenegger and Collins 1997), then each robot's D value is just its distance to the coordinate system origin, $\sqrt{x^2 + y^2}$. However, the method used to develop the coordinate system, and therefore the location for each robot, must be constrained to the given robotic and environmental assumptions, so complex systems such as GPS can't be used.

Given that the only available sensor to the robots is a simple, local range sensor, and the fact that the robots can communicate with neighbors, an approach based on trilateration will be used to develop the coordinate system. The principal of trilateration is a very common tool used to develop a coordinate system in wireless sensor networks and group robotics, for example in (Moore, et al. 2004). The general idea of trilateration is as follows. Imagine there exist three robots, positioned non-linearly, where all know their own location in the coordinate system. If there is a fourth robot which can see the three robots, can measure its distance to all three robots, and through communication can read those three robot's locations in the coordinate system, then that robot can compute its own location in the coordinate system, becoming localized. This process of using three robots with coordinates to compute a fourth, un-localized robot's coordinates, continues until there are no un-localized robots that can see three localized neighbors. As long as the collective's communication connectivity graph forms a trilateration graph,

(Eren, et al. 2004), which in this thesis is highly likely given the starting configuration of the robots, then trilateration will give all robots in the collective a location in the coordinate system.

For trilateration to begin, three robots must have their coordinates initialized using some process other than trilateration. The approach taken in this thesis, called trilateration seeding, is as follows. First, an initial robot chooses its location in the coordinate system to be (0,0). Next, this first robot assigns a location along the positive x axis of the coordinate system to a second robot, one of its neighboring robots, with an x value equal to the distance between the two, and a y value of zero. After that, a third robot that can see the first two robots, has two possible locations in the coordinate system. This third robot chooses the location that gives it a positive y value. At this point, there are three robots that now have a location in the coordinate system, so trilateration can commence. This approach to initialize these first robots is very similar to the approaches found in previous works such as (Moore, et al. 2004) and (Savarese, Rabaey and Beutel 2001).

Due to the distributed nature of the collective, it is very likely that multiple robots will attempt to start this trilateration seeding. This will result in multiple conflicting coordinate systems being developed by trilateration, with one coordinate system starting from each robot that started trilateration seeding. This thesis will present a method to resolve these conflicts, which will eventually create a single consistent coordinate system, no matter how many robots start trilateration seeding. The details of trilateration, trilateration seeding, and resolving coordinate system conflicts, will be presented in chapter 5.

The approach used in this thesis to find the θ value for each robot, which results in a consistent location for the desired shape among all robots, first requires the development of the coordinate system. Once the coordinate system is developed, the robots can discover θ by completing a three step process. In this process, a robot moves to three non-linear locations, and after each movement re-computes its location in the coordinate system. By recording both the commanded movement, (ie. "forward" and "rotate") as well as the new locations after each movement, the robot can compute the

value of θ. Additionally, with information gathered from the three movements, the value of θ for this robot can be computed for any location in the coordinate system, without any further movement. Further details for computing θ will be presented in chapter 5.

4.2.2 Self-Assembly and Self-Healing

The method in this thesis which controls the movement of the robots in the collective in order to self-assemble and self-heal the shape is called DASH (distributed self-assembly and self-healing). Running on each robot separately, DASH takes as its inputs: the location of the robot in the coordinate system, a description of the desired shape, the scale of the desired shape, and communication with neighboring robots. The output of DASH is a commanded movement for the robot, as well as possible messages to communicate to neighbors. The goal of DASH is to move each robot into the desired shape, while at the same time, not preventing other robots from entering that shape.

In this section, DASH is provided with two initial inputs when it begins running on the robots. Those two inputs are the desired shape to be formed, and the scale at which the shape is to be formed. The first given input is the desired shape, given to DASH in the form of a black and white image, called the shape pixel map. In this shape pixel map, locations that are part of the desired shape correspond to white pixels in the shape pixel map, while locations that are not part of the desired shape correspond to black pixels in the shape pixel map. Each pixel in the shape pixel map has a unique location, in terms of its x and y index. See Fig. 4.1(B) for example of these indices. These indices represent the number of pixels in the corresponding x or y direction that are between that pixel and the upper left corner of the shape pixel map (note: this is different from the upper left corner of the shape). This shape pixel map representing the desired shape is given to all robots at the start. A simple example of a 4x5 shape pixel map is shown in Fig. 4.1(A). In this example the desired shape (white pixels) is a rectangle with a height to width ratio of 3/2, because the shape is 3 pixels high and 2 pixels wide. Notice that this shape pixel map only describes the shape in terms of the ratio of x and y, but not the actual size of the shape.

The second input to DASH is the scale. This scale value tells DASH how big the desired shape should be formed. This scale is defined as the desired dimensions of one pixel from the shape pixel map, in terms of the robot radius, R_{robot}. For example if the scale value is 1.0, then for the shape pixel map in Fig. 4.1(A), the shape will be $3 \times R_{robot}$ high, and $2 \times R_{robot}$ wide. Similarly, if the scale value is 3.0, then for the shape pixel map shown in Fig. 4.1(A), the shape will be $9 \times R_{robot}$ high, and $6 \times R_{robot}$ wide. While the scale of the shape can vary when DASH is running on the robots, DASH cannot itself change the value of the scale. Therefore, in this section the scale is assumed to be a constant. However, with S-DASH, the scale of the shape is not assumed to be a constant, because the function of S-DASH is to choose the proper scale according to the number of robots currently in the collective.

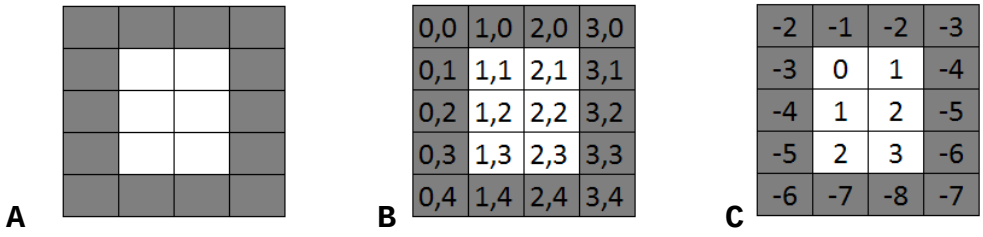


Figure 4.1. (A) A shape pixel map for the desired shape of a rectangle. (B) The X and Y index values for each pixel is displayed in the corresponding pixel. (C) The priority values of each pixel given by DASH, displayed in the corresponding pixel.

Using the coordinate system developed by all robots in the environment, DASH first virtually overlays the shape pixel map onto this coordinate system. This is done by assuming the upper left pixel in the shape pixel map, pixel (0,0), corresponds to and is centered on the location (0,0) in the coordinate system. All other pixels (x,y) then correspond to the location $(scale \cdot x \cdot R_{robot}, scale \cdot y \cdot R_{robot})$ in the coordinate system. For example, if the scale is 2.0, then the pixel (1,1) in the shape pixel map is centered at $(2R_{robot}, 2R_{robot})$ in the coordinate system; and the pixel (2,3) in the shape pixel map is centered at $(4R_{robot}, 6R_{robot})$ in the coordinate system.

With this virtual overlay of the shape pixel map on the coordinate system, each robot can use its location in the coordinate system to then determine which pixel in the shape pixel map it is closest to. If the robot is closest to a pixel that is inside the desired shape, then DASH gives a movement command that will move the robot in such a way that keeps the robot from blocking other robots that are trying to get into the shape. If the robot is closest to a pixel that is outside the desired shape, then DASH gives a movement command that will move the robot into the shape. These movement commands are generated by DASH, and are based on the robot's location in the shape pixel map. This generation of movement commands by DASH is deterministic, so for any given shape pixel map, all the DASH controllers use the same set of commands. This set of commands consists of a list of all pixels in the shape pixel map, and for each pixel, a direction in which DASH should move a robot if that robot is located closest to that particular pixel. Furthermore, if the robot cannot move in the commanded direction due to collisions with other robots in the collective, DASH provides alternative directions to move to avoid local minima where robots may become trapped. A more detailed explanation of DASH can be found in chapter 6.

4.2.3 Scalable Self-Assembly and Self-Healing

As described in section 3.5, there are two methods for setting the scale of the shape: fixed and adaptive (scalable). In this thesis, the scalable method is chosen, both to keep the robot density constant, and to best mach the given robotic capabilities, specifically the limited communication range. This approach is called scalable distributed self-assembly and self-healing, or S-DASH. With S-DASH, the robotic collective will automatically form its shape in the same way as with DASH, but now at a scale that is proportional to the number of robots currently in the collective.

To adjust the scale of the shape, S-DASH uses the property that in DASH, there is one location that is going to be the last spot filled by robots inside the shape, and this location is known (the details of this property are explained in chapter 6). For an example of this property, consider the desired shape pixel map shown in Fig. 4.1(A). DASH will set a priority value for each pixel in the shape pixel map shown in Fig.

4.1(C). The higher this value, the higher the priority DASH has to fill the pixel with robots. Based on this priority filling, the pixel in the upper left corner of the shape in Fig. 4.1(C), pixel (1,1), will be the last location filled by a robot or robots.

S-DASH uses the last filled location property of DASH to determine if the shape is too big. This is done by observing if the last filled location is not filled by a robot for long enough of a time, then S-DASH can be confident that all the robots are already inside the desired shape, but the scale is too big, as evidenced by the unfilled location. If the shape is too big, then S-DASH will decrease the scale of the shape. The mechanisms and details for how to decrease the scale, as well as how to detect if the last filled location is empty for long enough of a time, can be found in chapter 7.

In a similar way to the last filled location, S-DASH also makes use of the property that in DASH, there is one known location that is going to be the first spot outside the shape filled by robots, given the shape is already full (the details of this property are explained in chapter 6). For an example of this property, again consider the priority values of pixels in the shape pixel map, shown in Fig. 4.1(C). If there is no room for any more robots in the shape, then for any robots outside the shape, DASH will move them to the location with highest priority outside the shape, which in this example is location (1,0).

S-DASH uses this location outside the shape, which is the first filled if the shape is full, to determine if the shape scale is too small. This is done by observing if this location is filled by a robot for long enough of a time. If so, then S-DASH can be confident that all locations within the shape are already full, and therefore the scale is too small. If the shape is too small, then S-DASH will increase the scale of the shape enough to fit in the extra robot(s). The mechanisms and details for how to increase the scale, as well as how to detect if the first filled outside-of-shape location has a robot in it for a long enough time, can be found in chapter 7.

4.2.4 Robot Differentiation

Lastly, this thesis presents a method to control individual robot differentiation, in order to form an overall pattern of robot roles in the shape, i.e. the spatial-temporal role

pattern. Each robot chooses its role based on: its current location, the current scale of the shape, the robot's local clock, and the given spatial-temporal role map. The spatial-temporal role map is given to each robot at the beginning.

The spatial-temporal role map consists of a pixel map. This pixel map, called the differentiation pixel map, contains the same number of pixels, and has the same x and y dimensions as the shape pixel map (the pixel map that defines the collective shape). Furthermore, each location in the differentiation pixel map that corresponds to a white pixel in the shape pixel map has a list of roles associated with it. This list of roles represents which roles the robot should take if it is in the corresponding pixel in the shape pixel map. If this list has only one entry, then the robot will take on that role irrespective of time. If the list has more than one entry, then a robot in that location will have a role that varies in time. To vary its role in time, a robot will use its local clock to choose into which role in that list it should differentiate. For a spatial-temporal role pattern to emerge, this local clock must be synchronized with the local clocks of all the other robots in the collective. The details of this synchronization can be found in chapter 8.

This spatial-temporal role map defines robot roles in terms of a robot's location in the shape, so if the scale of the shape changes, the spatial-temporal role pattern must adapt to this change as well. For example, imagine a collective of robots forms a circle shape with a radius of $100 \cdot R_{robot}$. Furthermore, the collective has a spatial-temporal role map that instructs the robots that are closer than half the radius to the center of the circle to differentiate into role "A", and robots farther than half the radius to differentiate into role "B". Initially in this example, any robot which is closer than $50 \cdot R_{robot}$ to the center will take on role "A", and all others will take on role "B". If all robots farther than $50 \cdot R_{robot}$ from the center are then removed, then the collective will be shaped as a circle with radius $50 \cdot R_{robot}$, with all robots currently in role "A". For the spatial-temporal role pattern to be correct, now it must adapt to the new scale of the shape. To do this, robots that are farther than a distance of $25 \cdot R_{robot}$ from the center must switch roles from "A" to "B".

A full description of how this robot differentiation is implemented can be found in chapter 8.

Chapter 5: Self-Organizing Coordinate System

In this thesis, a self-organizing coordinate system is used in DASH, S-DASH, and for robot differentiation. In all three of these contributions, the coordinate system, along with the scale, is used by every robot in the collective to determine where in the desired shape it is located. Due to the importance of this coordinate system, first an overview of related work in developing a coordinate system in a distributed system will be presented. After describing the related work, the details of developing a self-organizing coordinate system for this thesis will be presented.

5.1 Related Self-Organizing Coordinate Systems

There is a large amount of previous work related to creating a coordinate system in a distributed collective. A large portion of this related work is in the field of wireless sensor networks, in particular, the sensor calibration problem. In the sensor calibration problem, a wireless sensor network develops a coordinate system, so that information gathered from the various sensors can have a geographical tag associated with it, giving more meaning to the data. Furthermore, the individual nodes in a wireless sensor network tend to have very simple sensors, so the approaches used in their work may be compatible with the robotic and environmental assumptions in this thesis. An overview of this related work can be found in: (Bachrach and Taylor 2005), (Yang and Liu 2007), and (Mao, Fidan and Anderson 2007). For the brief overview in this thesis, some of the most related works will be divided into two categories.

One category of related work in developing a coordinate system in wireless sensor networks uses only communication (no sensing) between neighboring sensors. Using this communication as well as globally unique IDs, each sensor can send a message to a sensor outside its communication range, as shown in Fig. 5.1(A), where sensor “A” is trying to send a message to sensor “B”. A hop count is used with this message to discover the minimum number of sensors required to relay the message from the source sensor to the destination sensor. This minimum path is shown as a dotted line in Fig.

5.1(A). If an assumption is made that the distance between any two consecutive relays is approximately the maximum communication range, R_{com}, then the distance between sensors can be estimated as $(R_{com} \cdot (N_{sensors}-1))$, where $N_{sensors}$ is the number of sensors it takes to send a message from the source to the destination, inclusive.

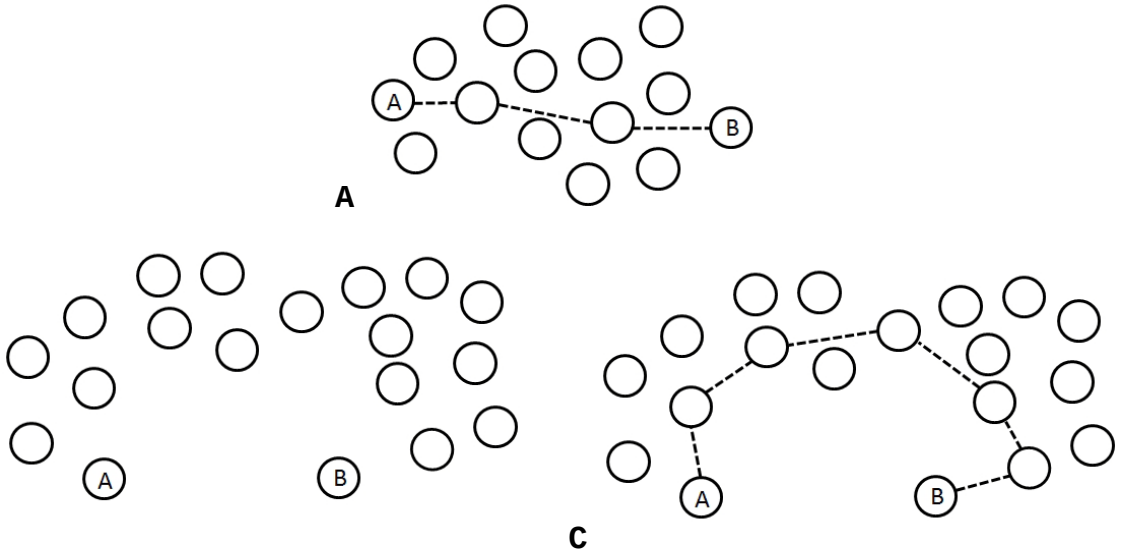


Figure 5.1. (A) Estimating distance using sensor-to-sensor routing "hops". (B) A network with large areas of no sensors. (C) The path of sensor-to-sensor hops between sensor A and sensor B, in the network from (B).

In (Niculescu and Nath 2003), this distance estimation technique, called "distance vector hops" or "dv-hops" for short, is used to estimate the distance between a sensor and three "landmarks". These landmarks are given their position in the coordinate system *a priori*. Using the known location of the landmarks, each sensor can triangulate its position in the coordinate system using dv-hops to estimate its distance to each landmark. Of course the drawback with this method is that the landmarks are given their position by some other means.

One problem with this kind of dv-hop based distance estimation occurs if the sensor network has large holes where no sensors are present, as shown in Fig 5.1(B). If this is the case, a dv-hop method to estimate distance would actually measure the shortest distance between two sensor nodes that avoids the area with no sensors, for example

shown in Fig. 5.1(C). In that case, the distance estimate from dv-hop is an over-estimation of the distance, resulting in an incorrect position of the sensor from triangulation. To solve this problem, (Li and Liu 2007) propose an approach that can measure the deviation between the reported dv-hop distance and the actual straight line distance.

A second problem with the dv-hop approach for measuring distance is that the distances it gives can be inaccurate. This inaccuracy stems from the fact that the distance between consecutive sensors along the dv-hop path are assumed to be a constant value of R_{com}. This assumption breaks down as the communication path between the sending and the receiving sensors deviates from a straight line path. The approach by (Nagpal 2003) reduces this error by averaging the dv-hop path length to the receiver among all the receiver's neighbors.

There are two main drawbacks to these three communication-only based approaches. The first is that they require some sensors to be initialized *a priori* with their location in the coordinate system. The second drawback is that they require a globally unique ID.

A second category of related work in developing a coordinate system for a wireless sensor network uses both communication and sensing distance between neighboring sensors. The basis for these approaches is to localize an un-localized sensor based on communication and constrained by the sensed distance between itself and localized neighbors. This newly localized sensor then can be used to aid in localizing more sensors. This wave of localizing sensors will sweep across the sensor network, until all sensors have become localized.

One difficulty with such an approach is that given some localized sensors, there may be more than one possible location for an un-localized sensor to choose as its own coordinates. For example in Fig. 5.2(A), if only a single localized sensor is available, say sensor "A", then an un-localized sensor, say sensor "B", can choose any location on a circle centered on the localized sensor, with a radius that is the sensed distance between the two sensors, D_{ab}. If only two localized sensors are seen by sensor "C", for example sensor "A" and sensor "B", then there are, in general, two possible locations for "C" to be

located, as shown by the red circles in Fig. 5.2(B). These locations are the intersections of two circles, centered on sensor "A" and "B", with a radius of the sensed distance between D_{ac} and D_{ab} respectively. One approach taken by (Goldenberg, et al. 2006) addresses this problem by remembering all the possible locations that a sensor can take, and pruning that list as more sensors become localized. Another approach by (Moore, et al. 2004) does not localize a sensor until it only has one possible location. Furthermore, because a distance sensor is used in this category, when that distance sensor is noisy, it will have negative effects on the development of the coordinate system. To mitigate the effects of this sensor noise, (Moore, et al. 2004) and (Yang and Liu 2008) choose the neighbors whose sensor noise will negatively affect the coordinate system the least.

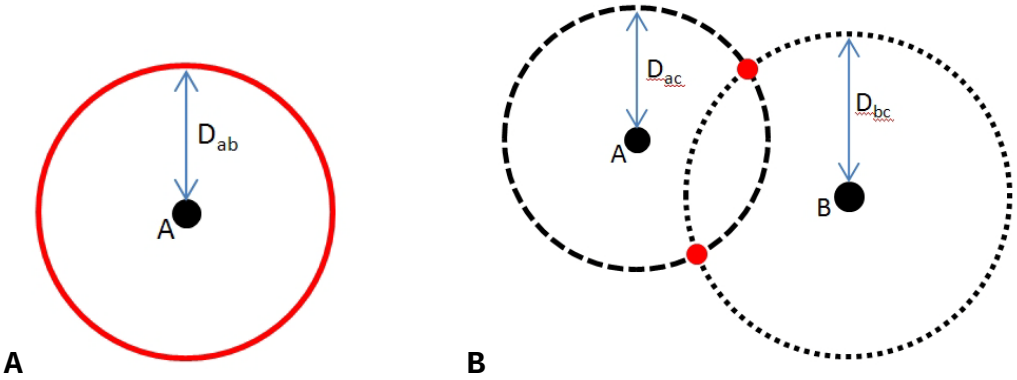


Figure 5.2. (A) Possible locations (red circle) with one localized sensor. (B) Possible locations (red dots) with two localized sensors.

Another approach to developing a coordinate system can be found in the field of modular robotics. This approach takes advantage of the physical connection between two neighboring robots to determine a precise location of a robot with respect to its connected neighbors. Using knowledge about the geometry of this connection, each robot can compute a vector which represents where it is located with respect to its neighbor. Then if an un-localized robot has a neighbor that is localized, it too can localize simply by adding this vector to the neighbor's position, and making this its own location in the coordinate system. An example of this approach can be found in (Stoy and Nagpal 2004). The main drawback to this type of approach is that the coordinate system cannot

adapt to damage caused by the repositioning of robots in the group because there is no mechanism provided for resolving conflicting coordinate systems.

In all three of these approaches to developing a coordinate system (communication only, communication and distance sensing, and physical connection), an initial startup phase is used to initialize the coordinate system. In the approach using physical connection, a seed is used to start the coordinate system. This is done by placing a single seed robot which has the coordinates (0,0), into an un-localized group. Once this seed is in place, the coordinate system can develop as described. As mentioned before, this seed is a single point of failure. In the communication only approaches, a group of seeds, called "landmarks" are placed sparsely in the environment, and each is given a correct location in the coordinate system. This use of landmarks assumes that each landmark can sense its position in the environment, something that is not part of the environmental assumptions in this thesis. In the communication and sensing approaches, various startup approaches are taken, with the most common being trilateration seeding. Trilateration seeding is described in section 5.2.1, and requires additional methods to merge the coordinate systems from each seed into a single coordinate system.

5.2 Developing a Coordinate System

The approach taken in this thesis to self-organizing a coordinate system is largely based on, and borrows heavily from, the work done in (Moore, et al. 2004), and to a lesser degree (Savarese, Rabaey and Beutel 2001) and (Howard, Mataric and Sukhatme 2001). The approach uses both communication as well as distance sensing to self-organize a consistent coordinate system. The approach starts by seeding the coordinate system using trilateration seeding, in a method closely related to the work in (Savarese, Rabaey and Beutel 2001), as well as others. Next, after an initial three or more robots have a location given to them via trilateration seeding, the method of robust quadrilaterals from (Moore, et al. 2004) is used to trilaterate the neighboring robots, but without using global ID's. During this process, the inconsistent coordinate systems from different trilateration seeds are merged to produce a single, consistent coordinate system. This is accomplished using a method similar to (Howard, Mataric and Sukhatme 2001).

Furthermore, the approach in the thesis allows robots to move while at the same time maintaining the coordinate system.

5.2.1 Trilateration Seeding

As described above, to start the trilateration process, at least three robots need to be localized in a coordinate system. In this thesis, this is accomplished by trilateration seeding, which is closely related to the approach in (Savarese, Rabaey and Beutel 2001). Trilateration seeding uses communication between robots, sensing the distance between robots, and a locally unique ID for each robot, to initially develop many local, isolated coordinate systems which are then used for trilateration.

Trilateration seeding starts by locally selecting a single seed robot to start the trilateration seed. Because these seed robots are selected locally, multiple seeds will form in the collective. Once these seeds are selected, they each recruit two other local reference robots to act as the second and third robot in their local coordinate system. This process of trilateration seeding can be seen in the collective shown in Fig. 5.3(A). In Fig. 5.3(B), robots that are seeds for a local coordinate system are shown as red; the robots that are reference robots for the seed robots are shown as green. All remaining robots are shown as black.

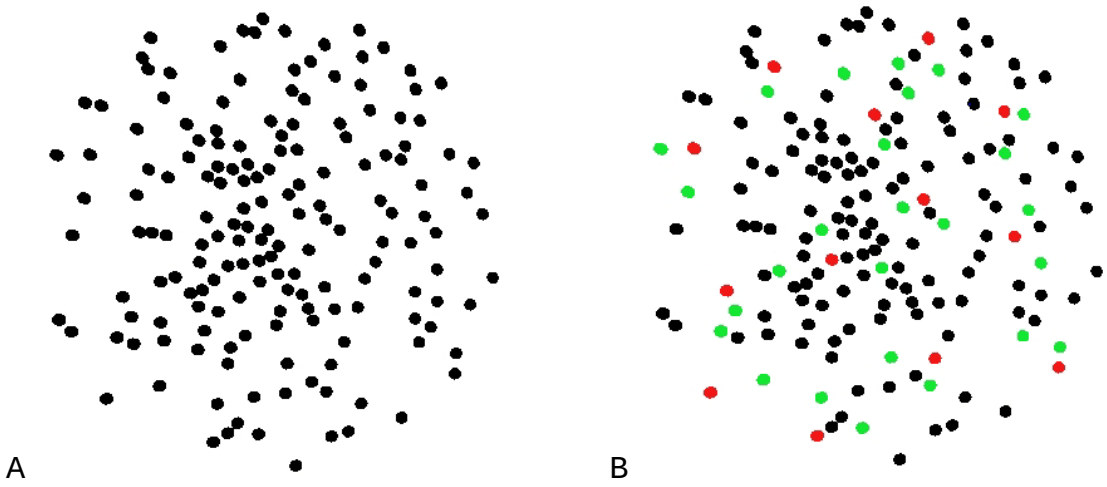


Figure 5.3. (A) Starting position for a collective. (B) Seed robots (red) for the collective, and reference robots for those seeds (green).

5.2.1.1 Local ID

Initially, all robots lack any form of ID; however, it is required, since a locally unique ID is used for developing the coordinate system. This locally unique ID guarantees that for any robot, all that robot's neighbors (robots within communication range) have an ID that differs from all other neighbors of that robot. To assign locally unique IDs to each robot, each robot does the following. First, it randomly chooses a number and assigns that number as its own ID. Then, for every loop of the main controller, it communicates its ID to all its neighbors. If a robot ever receives a message from two different robots, where both robots have the same ID, then that robot transmits a "change_id" message, which contains the value of those two robots' ID. If a robot ever receives a "change_id" message with a value that equals its own ID, then it randomly selects a new ID. See Fig. 5.4. for a summary of how to select local IDs. As long as the range of the possible randomly selected IDs is much larger than the number of neighbors a robot can have, this method for selecting locally unique IDs will quickly converge to a solution where every robot has a locally unique ID.

```
If( has_local_ID = FALSE )
{
    Generate_new_local_ID
}
If( received change_id message with value = my_local_ID)
{
    has_local_ID = FALSE
}
For( all neighbors i,j )
    If( neighbors_ID(i) = neighbors_ID(j) )
    {
          Send change_id message with value = neighbors_ID(j)
    }
```

Figure 5.4. Pseudo code for selecting a locally unique ID.

5.2.1.2 Selecting Seed Robots

To start the seeding process, each trilateration seed starts with a single robot, called the seed robot. These seed robots are selected so that they are spread out sparsely among the collective, while at the same time, each robot can sense at least two seed

robots, or one other if that robot itself is a seed. To select these seed robots, a two level approach is taken. This two level approach is used to ensure that each robot can sense at least two seed robots, or one other if itself is a seed robot.

The top level guarantees that each robot can see at least one seed, or is itself a seed. This top level uses the locally unique IDs to elect top level seed robots. In this election process, every robot sends an "elect_seed_top" message to all its neighbors, with its own local ID included in the message. If a robot receives an "elect_seed_top" message with a value greater or equal to its own ID, it does not become a top level seed robot. If a robot only receives "elect_seed_top" messages with values less than its own ID, it considers itself a top level seed robot.

The bottom level guarantees that each robot can see at least two seeds, or one if it itself is a seed. The bottom level seed election works by having every robot that is not a top level seed, and which can see only one top level seed and no bottom level seeds send out an "elect_seed_bottom" message to all its neighbors. This message contains its own local ID. If a robot receives an "elect_seed_bottom" message with a value greater or equal to its own ID, it does not become a bottom level seed robot. If a robot only receives "elect_seed_bottom" messages with values less than its own ID, it considers itself a bottom level seed robot. In the remainder of the thesis, both top and bottom level seeds are considered seeds. This election process for both top level and bottom level seeds will continue indefinitely, with robots sending out new election messages every loop of its main controller when no top or bottom level seeds are seen. This is in order to maintain that every non-seed robot sees two seeds, even in the case of movement, addition, or removal of robots in the collective.

5.2.1.3 Selecting Reference Robots

Once a robot seed is elected to start the trilateration seeding, that robot selects two neighboring robots, called reference robots, in order to have the three robots required to start trilateration. The seed robot (robot A) chooses the two reference robots (B and C) based on three criteria. First, robot B must be able to communicate with robot C. Second, the minimum internal angle, α, of the triangle for robots A,B,C, is larger than the

minimum triangle angle for any other choice of robots B and C. Thirdly, the value of α should be greater than α_{min}, which is a pre-defined minimum angle. This constraint on the minimum internal angles give the local coordinate system robustness to distance sensor noise, as described in (Moore, et al. 2004). All minimum angles in the triangle A,B,C, can be computed with the law of cosines, because all three distances, AB, AC, CB, are measurable.

5.2.1.4 Local Coordinate System

Once the seed robot has selected its two reference robots, it then assigns itself and its two reference robots a location in a local coordinate system. The seed takes a location of (0,0) in its local coordinate system. It then assigns robot B the location (D_{AB},0) in the local coordinate system, where D_{AB} is the distance between the seed robot and robot B. With the location for robot A (the seed robot) and robot B already assigned in the local coordinate system, there are two possible locations for robot C in this local coordinate system. The seed robot arbitrarily assigns robot C the one location that has a positive y value. This location is ($D_{ac} \cdot \cos(\alpha)$, $D_{ac} \cdot \sin(\alpha)$), where D_{ac} is the measured distance between the seed robot and robot C, and α is angle (B,A,C) between robots B,A,C respectively, which can be found using the law of cosines. This method is also used in other approaches such as (Savarese, Rabaey and Beutel 2001).

5.2.2 Trilateration

Once a seed and two references are localized in that seed's local coordinate system, then any robot (robot Z) that can see the seed (robot A) and two other robots (robots E,F) which are localized in that seed's local coordinate system, can also become localized in seed A's local coordinate system. To provide robustness to sensor noise, another constraint on the choice of robots E and F is that the minimum internal angle of the triangles with robots at the vertices, (A,Z,E), (A,Z,F), and (E,Z,F), is required to be greater than some preset minimum angle, α_{min}, again as described in (Moore, et al. 2004).

Once an un-localized robot, robot Z, can see neighbors E and F that are localized within A's local coordinate system, and the minimum internal angle is greater than α_{min}, robot Z can then also localize in seed A's local coordinate system. This localization uses

the positions of robots A,E,F as well as the distances between robots A,E,F,Z. If the location of reference robot E is located on the positive x axis in the local coordinate system, then the position of robot Z is given by the trilateration formula 5.1, where D_{12} is the measured distance between any robot 1 and robot 2, and (x_1,y_1) is the coordinates for robot 1 in the local coordinate system of robot A.

$$x_z = \frac{D_{AZ}^2 - D_{EZ}^2 + D_{AE}^2}{2 \cdot D_{AE}} \qquad y_z = \frac{D_{AZ}^2 - D_{FZ}^2 + x_F^2 + y_F^2}{2 \cdot y_F} - \frac{x_F \cdot (D_{AZ}^2 - D_{EZ}^2 + D_{AE}^2)}{y_F \cdot (2 \cdot D_{AE})} \tag{5.1}$$

In the case where robot E is not located on the positive x axis, the location values of robot E and F are first rotated by -θ, where θ is the angle between the positive x axis and the location of robot E. This rotation does not affect the E and F location values, as it is only used locally by robot Z for computation. After this rotation, the rotated coordinates of robot E and F are used in formula 5.1. Then the resulting location for robot Z from formula 5.1 is rotated by θ, and this result is used as the location of robot Z in robot A's local coordinate system. Now with robot Z localized, it can be used in trilateration to aid other non-localized robots in localization.

5.2.3 Merging Coordinates

After trilateration seeding, there will be many local coordinate systems, each one centered on its seed robot, and oriented with respect to that seed's reference robots. This results in many local coordinate systems in the collective that can differ by a translation (the origins are not aligned), rotation (x axes are not parallel), and possibly the "handedness" of the coordinate system (y axis is 90° ahead of the x axis, or 90° behind). For example, if the collective in Fig. 5.5(A) forms two seeds, robots 1 and 9, where robot 1 chooses robots 3 and 2 to be its reference robots, and robot 9 chooses robots 7 and 8 to be its reference robots, 2 local coordinate systems are formed. The local coordinate systems for robots 1 and 9 are shown in Fig. 5.5(B); they differ by a translation, a rotation, and have different handedness. To produce a single consistent coordinate system, which is needed for this thesis, there must be a mechanism for resolving these differences between local coordinate systems.

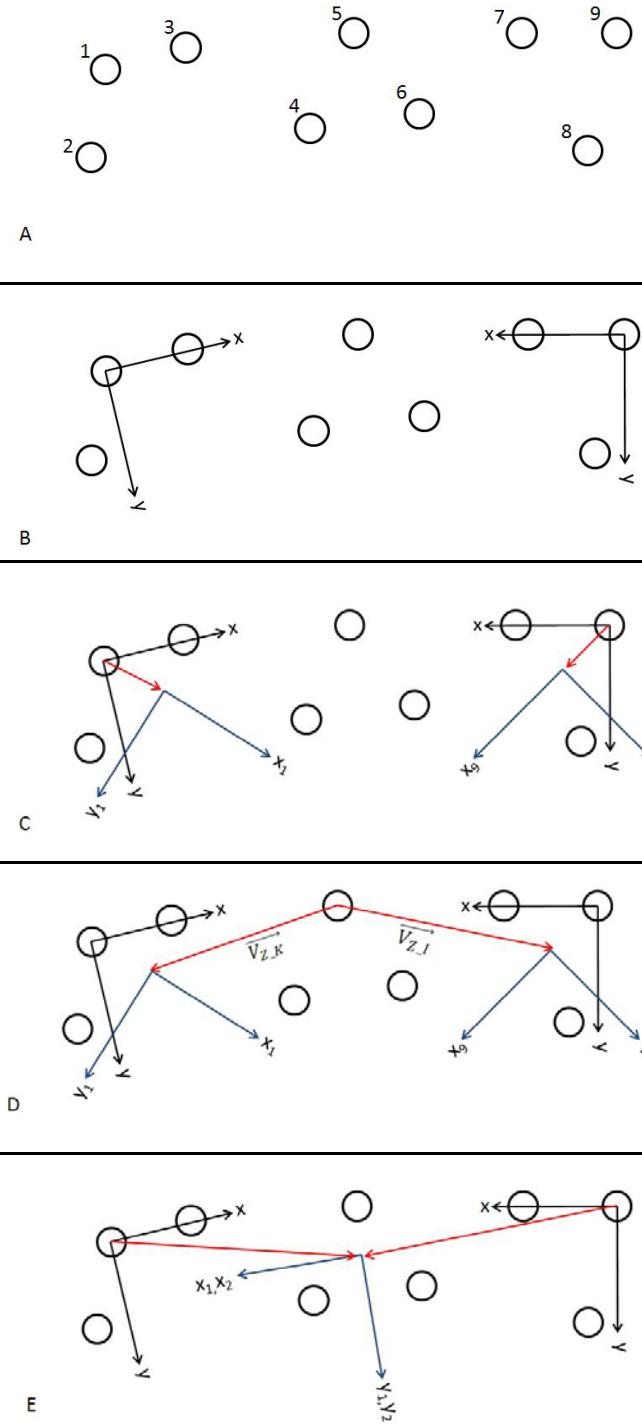


Figure 5.5. (A) An example collective. (B) A possible assignment of local coordinate systems. (C) Example transitional coordinate system (shown in blue) for seed robots 1 and 9. (D) Visualization of example $\overrightarrow{V_{Z\_J}}$ and $\overrightarrow{V_{Z\_I}}$ (shown in red) where Z is robot 5, K is robot 1, and I is robot 9. (E) Consistent transitional coordinate systems.

To resolve these differences between local coordinate systems, this thesis gives a method which modifies all the local coordinate systems so that they eventually become consistent with each other; i.e. their origins are located in the same position, have the same rotation and handedness. These modified local coordinate systems are a 3D coordinate system, called the "transitional coordinate system", which are initially independent from each other. The method called "coordinate merging" assigns and updates a vector $\overrightarrow{V_{seed}}$ and a 3D rotation matrix R_{seed} to each seed robot. The vector $\overrightarrow{V_{seed}}$ represents the vector in the seed robot's local coordinate system that points to the origin of that seed's transitional coordinate system. The rotation matrix R_{seed} represents the rotation that is required to rotate the seed's local coordinate system origin into the same orientation as its transitional coordinate system origin. Initially since no information is known about how to modify the local coordinate system, $\overrightarrow{V_{seed}}$ is set to the zero vector, and R_{seed} is set to identity. With those initial values of $\overrightarrow{V_{seed}}$ and R_{seed}, the origin of the seed's transitional coordinate system is at the same location and orientation as its local coordinate system. In Fig. 5.5(C), the origins of the transitional coordinate systems (shown in blue) for seed robots 1 and 9 are shown where both seeds have the following example values.

$$\overrightarrow{V_{seed}} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad R_{seed} = \begin{bmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The vectors $\overrightarrow{V_{seed}}$ for both seed robots are shown as a red arrow pointing from the seed robot to the origin of its transitional coordinate system.

Using $\overrightarrow{V_{seed}}$ and R_{seed}, a robot's location in a local coordinate system can be converted to its location in a transitional coordinate system as follows. If the robot is a seed robot, then its location in its own transitional coordinate system is $R_{seed} \cdot (-\overrightarrow{V_{seed}})$. If a robot, robot Z, is not a seed robot, then for every seed, "A", to which it is localized, it has a location in A's transitional coordinate system of $R_A \cdot (\overrightarrow{x_z} - \overrightarrow{V_A})$, where $\overrightarrow{x_z}$ is a

vector that represents robot Z's location in A's local coordinate system, and R_A, $\overrightarrow{V_{A,}}$ are the R_{seed} and $\overrightarrow{V_{seed}}$ values for seed robot A.

When the origins of the transitional coordinate systems for all seed robots in the collective are located in the same spot, and have the same orientation, for example, as shown in Fig. 5.5(E), then the transitional coordinate systems for all seeds are considered to be consistent. Furthermore, if the transitional coordinate systems are consistent, then so are all the robot locations in all transitional coordinate systems. The method of coordinate merging is designed so that this will occur. Coordinate merging does so by iteratively updating the $\overrightarrow{V_{seed}}$ and R_{seed} values for each seed every loop of the main controller.

5.2.3.1 Updating $\overrightarrow{V_{seed}}$

The update for each seed's $\overrightarrow{V_{seed}}$ value is chosen so that the origin location for each seed's transitional coordinate system moves closer to the origin of another seed's transitional coordinate system. Specifically, the new $\overrightarrow{V_{seed}}$ is chosen so that the origin location for each seed's transitional coordinate system is moved halfway between its old location and the location of the origin of another seed's transitional coordinate system. This averaging of transitional coordinate system origins over many iterations will result in all the transitional coordinate system origins being located in the same location.

To update each seed robot's $\overrightarrow{V_{seed}}$ in this way, the principal of a "merging" group consisting of three robots is used. These three robots must be able to communicate with one another, and form a triangle with a minimum internal angle greater than α_{min}. These three robots in the merging group must be localized in the local coordinate systems of at least two common seeds. For example, in Fig. 5.5, if the seed robot 1 can see robots 2 through 8, and the seed robot 9 can see robots 2 through 8, then robots 2 through 8 will have coordinates in the local coordinate system for seed 1 as well as for seed 9. Then, assuming that their minimum internal angle is greater than α_{min}, robots 4,5,6 can form a merging group for seeds 1 and 9. Whenever there are three robots that satisfy the constraints for a merging group, they will form one. This will cause many merging

groups to form, and it is possible that a single robot may be part of more than one merging group.

Any one robot in the merging group can communicate with the other robots in the group. Therefore it is possible to discover the location in the local coordinate system for the other two robots in the merging group with respect to the two common seed robots, seed robot "I" and seed robot "K". Using these values and its own local coordinate system values with respect to seed robot I and K, any robot in the merging group can compute R^I_K, which is a 3D rotation matrix that defines how to translate a vector in seed K's local coordinate system to that of robot I's.

Once R^I_K is known, then the location of the origin of robot K's transitional coordinate system in terms of robot I's local coordinate system can be found. To do this, first two vectors, $\overrightarrow{V_{Z\_K}}$ and $\overrightarrow{V_{Z\_I}}$, are computed. These vectors represent the vector from a robot in the merging group, robot "Z", to the origin of the transitional coordinate system of robot K or robot I, respectively. These vectors are in the local coordinate systems of those respective seed robots. An example of $\overrightarrow{V_{Z\_K}}$ and $\overrightarrow{V_{Z\_I}}$ is shown in Fig. 5.5(D), where Z is robot 5, K is robot 1, and I is robot 9. These vectors are the negative of robot Z's location in the respective local coordinate systems, minus $\overrightarrow{V_{seed}}$ for the seed robot of that coordinate system. Therefore $\overrightarrow{V_{Z\_K}} = -(\overrightarrow{x_{z\_K}} - \overrightarrow{V_K})$ and $\overrightarrow{V_{Z\_I}} = -(\overrightarrow{x_{z\_I}} - \overrightarrow{V_I})$, where $\overrightarrow{V_x}$ is the vector $\overrightarrow{V_{seed}}$ for seed robot X, and $\overrightarrow{x_{z_X}}$is location of robot Z in seed robot X's local coordinate system. Next, $\overrightarrow{V_{Z\_K}}$ is translated into a vector in I's local coordinate system by multiplying R^I_K by $\overrightarrow{V_{Z\_K}}$. Now the location that is halfway between $\overrightarrow{V_{Z\_K}}$ and $\overrightarrow{V_{Z\_I}}$ can be computed, which is $(R^I_K \cdot \overrightarrow{V_{Z\_K}} + \overrightarrow{V_{Z\_I}})/2$. To move the origin of robot I's transitional coordinate system to this new location, $(R^I_K \cdot \overrightarrow{V_{Z\_K}} + \overrightarrow{V_{Z\_I}})/2$, the value of $\overrightarrow{V_{seed}}$ for robot I should be updated to $(R^I_K \cdot \overrightarrow{V_{Z\_K}} + \overrightarrow{V_{Z\_I}})/2 + \overrightarrow{x_{z\_I}}$. This update will cause the origin of robot I's transitional coordinate system to move halfway between its old location and the location of the origin of robot K's transitional coordinate system. This update is communicated to robot I, which for every loop of its main controller, will choose at random one of the received updates for $\overrightarrow{V_{seed}}$ and apply that update.

5.2.3.2 Updating R_{seed}

The update for each seed's R_{seed} value is chosen so that the x and y axes for each seed's transitional coordinate system become parallel with the x and y axes of every seed's transitional coordinate system. Specifically, the new R_{seed} is chosen so that the x and y axes for each seed's transitional coordinate system are rotated halfway between the old orientation and the orientation of another seed's transitional coordinate system. This averaging of the orientations of transitional coordinate systems over many iterations will result in all the transitional coordinate system origins having the same orientation.

Using the same value of R_K^I computed by a merging group as before, a robot in the merging group can also compute $R_{parallel}$, which is the rotation required of robot I's transitional coordinate system origin so that it becomes parallel to that of robot K's. The computation of $R_{parallel}$ uses the fact that the rotation needed to rotate K's transitional coordinate system to its local coordinate system is $R_{seed\_K}^{-1}$, where R_{seed_K} is the R_{seed} value for seed robot K. The computation of $R_{parallel}$ also uses the fact that the rotation needed to rotate robot I's local coordinate system to its transitional coordinate system is R_{seed_I}, which is the R_{seed} value for seed robot I. Using these facts, a robot in a merging group can compute $R_{parallel}^{-1} = R_{seed\_I} \cdot R_K^I \cdot R_{seed\_K}^{-1}$.

The inverse of this matrix, $R_{parallel}$, shows how to rotate the origin of robot I's transitional coordinate system so that it becomes parallel to that of K's; however, it is desired to only rotate I's transitional coordinate system halfway to becoming parallel to that of K's. To do this half rotation, quaternion spherical linear interpolation (SLERP) is used (Shoemake 1985). SLERP computes a parameterized path to rotate from a starting rotation to an ending rotation in the shortest possible way. To find the halfway rotation of $R_{parallel}$, SLERP is used to find the rotation that is half way between the 3x3 identity matrix and $R_{parallel}$. This halfway rotation matrix, called $R_{half\_parallel}$, is then used to determine the update for the orientation of the origin of robot I's transitional coordinate system. This update, which is $R_{half\_parallel} \cdot R_{seed\_I}$, is communicated to robot I, which for every loop of its main controller, will choose at random one of the received updates for R_{seed} and apply that update.

5.2.3.3 Convergence of Transitional Coordinate Systems

The update for $\overrightarrow{V_{seed}}$ and R_{seed} as described above will result in a consistent coordinate system for all robots in the collective. To demonstrate this, it will first be shown that the update process for $\overrightarrow{V_{seed}}$ and R_{seed} for every seed in the collective will converge to values that give a single location and orientation for the origin of the transitional coordinate systems for all seeds. Then it will be shown why these values also result in consistent coordinate locations for all robots in the collective.

First it will be shown that the update for $\overrightarrow{V_{seed}}$ will cause the origin of all transitional coordinate systems to converge to a single location in the environment. To do this, first consider a graph where each origin of the transitional coordinate system is a node, placed in a location corresponding to the origin's position in the environment. On this graph, place a link between two nodes if their corresponding seed robots share a merging group. As described in chapter 4, all robots form a well connected communication graph; therefore when seeds are formed, this graph will also form a well connected graph. An example of this graph of the origins of the transitional coordinate systems is shown in Fig. 5.6(A). In this graph, there is a group of nodes called the hull group, which is located on the convex hull of the graph. Fig. 5.6(B) shows these nodes as red, and the convex hull is shown as a dotted line.

Using the averaging update for $\overrightarrow{V_{seed}}$ described in section 5.3.2.1, when an origin that has a corresponding hull node is updated, there are two cases for where its new position can be. In the first case, if the update was done with a node inside the convex hull, i.e. not part of the hull group, then its new position will also be inside the hull. This new position is guaranteed to decrease the area of the complex hull. In the second case, when the update is done with another node in the hull group, the updated node location will be on the convex hull. When any node on a convex hull moves along an edge in the convex hull, the area of the new convex hull will always decrease. To show this is true, consider any node on a convex hull, call it node "A". That node will always have two neighbors also on the convex hull, call them nodes "B" and "C". These nodes A,B,C form a triangle that is part of the area inside the convex hull, shown as green in Fig.

5.6(C). For any new location of node “A” that is on the edge of the convex hull, the height of the triangle labeled as “h” in Fig. 5.6(C) will decrease. The area of the triangle is ½•b•h, where “b” is the base, as labeled in Fig. 5.6(C), and since the base does not change, a decrease in “h” will always decrease the area of the triangle. When the area of the triangle decreases, the overall area of the convex hull also decreases. This means that when any node on a convex hull moves along an edge in the convex hull, the area of the new convex hull will always be smaller. Furthermore, any update of a node on the convex hull, whether updating with another node on the convex hull or one not on the convex hull, will cause the area of the hull to decrease.

When a node that is updated is inside the convex hull, its new position must remain inside the convex hull. For it to move outside the convex hull, the node that it was updating with would have to be on the outside of the convex hull, which by definition is not possible.

In summary, an update of a node on the convex hull will always cause the area of the convex hull to decrease. Furthermore, an update of a node inside the convex hull will never cause the area of the convex hull to increase. This means that after updating the nodes for a long enough time, the area of the convex hull will converge to zero. A convex hull with an area of zero means that all the nodes in the graph are in the same location. Therefore the update for $\overrightarrow{V_{seed}}$ will cause the origin of all transitional coordinate systems to converge to a single location in the environment.

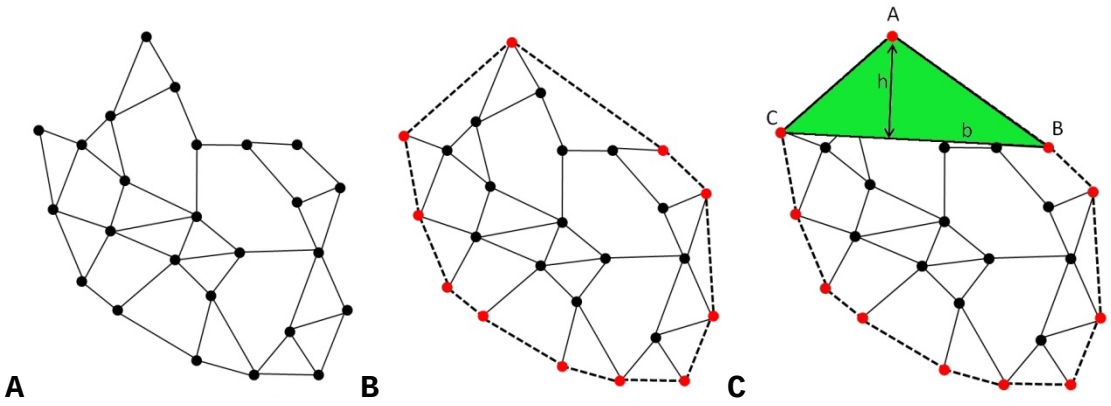


Figure 5.6. (A) A graph for the origin locations of each seed's transitional coordinate system. (B) The hull group (red), and the convex hull of the graph (dotted). (C) A triangle of three nodes in the hull group.

Next, it will be shown that the update for R_{seed} will cause the origin of all transitional coordinate systems to converge to a single orientation in the environment. For the case of R_{seed}, since the solution space is a closed manifold, its convergence will be shown experimentally. This experiment consists of 1000 runs. In each run, 25 values of R_{seed} are initialized at random, and in each step of a run, one R_{seed} value is updated according to the update method from section 5.2.3.2. After each step, the total error between the orientations of all 25 values of R_{seed} is computed. Fig. 5.7. shows the average value of the total error among all 1000 runs, for each step, as well as the variation between all runs, for select steps. This demonstrates experimentally that the update for R_{seed} will cause the origin of all transitional coordinate systems to eventually converge to a single orientation in the environment.

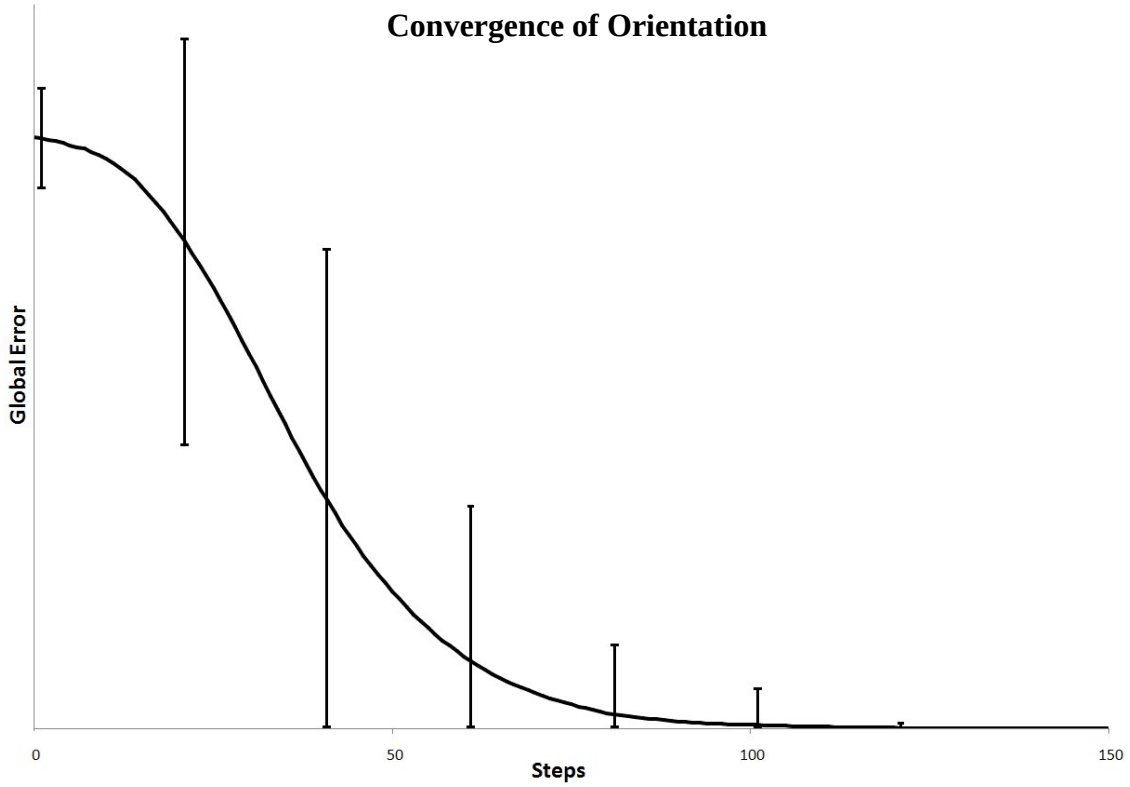


Figure 5.7. The average total error for 1000 experiment runs, as well as the variation for various steps.

Once $\overrightarrow{V_{seed}}$ and R_{seed} have converged, their values will result in a consistent coordinate system for all robots in the collective. This convergence of the orientation and position for the origin of each seed robot's transitional coordinate system means that if a robot can determine its location with respect to that origin, then its location will be consistent with all other robots that can do the same. All seed robots know their position with respect to that origin, which is $-\overrightarrow{V_{seed}}$. For all other robots in the collective, their position with respect to that origin is $R_z \cdot (\overrightarrow{x_z} - \overrightarrow{V_z})$, where $\overrightarrow{x_z}$ is that robot's location in seed Z's local coordinate system, and $\overrightarrow{V_z}$, R_z is seed robot Z's $\overrightarrow{V_{seed}}$ and R_{seed} respectively. These positions are the robot's locations in a seed's transitional coordinate system, so by using a robot's location in a transitional coordinate system, this location is consistent with other robots in the collective.

5.2.3.4 Rotating to 2D

After many updates of the R_{seed} and $\overrightarrow{V_{seed}}$ values for the seed robots in the collective, the transitional coordinate system for each seed robot will converge to be consistent with all other seed robots' transitional coordinate systems. This will give each robot a location in a consistent coordinate system; however, that location is in 3D, while the robots only exist in a 2D environment. For example, in Fig. 5.8(A), the collective is in a grid pattern in the x,y plane of the environment, while the coordinates of each robot in the transitional coordinate system are in all three dimensions, as shown in Fig. 5.8(B). The work in this thesis requires that the robots' coordinates be located in the x,y plane, so a method called "coordinate system rotation" is used to compute a consistent coordinate system that is constrained to the x,y plane, for example as shown in Fig. 5.8(C).

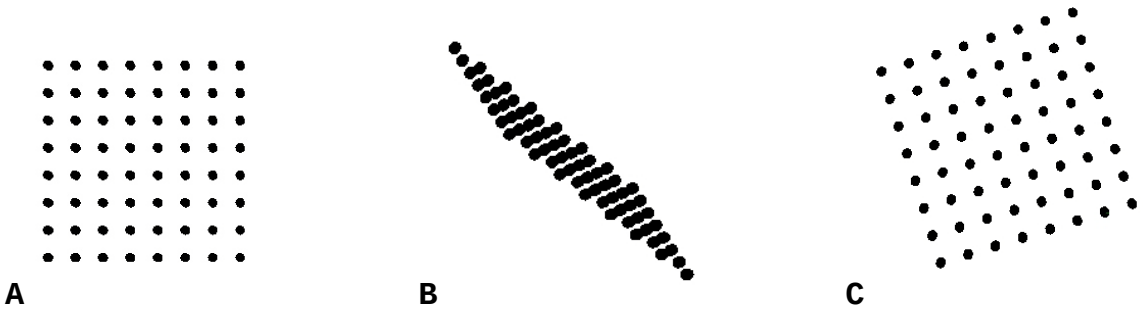


Figure 5.8. (A) The collective in the environment. (B) The 3D transitional coordinates of all robots, projected on the x,y plane for visualization. (C) The 2D rotated transitional coordinates after the coordinate system rotation.

The coordinate system rotation method uses the fact that while the coordinates of each robot in the transitional coordinate system are in all three dimensions, they all exist on the same arbitrary plane in 3D. If the direction of this plane, a normal vector, can be computed, then the transitional coordinate system can be rotated so that this normal vector has no x or y component, effectively forcing all robots to be on a plane that is parallel to the x,y plane. Once parallel to the x,y plane, this plane can be projected onto the x,y plane, giving the robots a consistent coordinate system in the x,y plane.

Every robot in the collective can compute the normal vector for the arbitrary 3D plane of robots in the transitional coordinate system. This is done by each robot. A

robot, robot "Z", looking at two neighboring robots, robots "A" and "B", computes two cross products between vectors $\overrightarrow{V_{ZA}}$ and $\overrightarrow{V_{ZB}}$, where these are vectors from robot Z to robots A or B. This pair of cross products is the result of $\overrightarrow{V_{ZA}} \times \overrightarrow{V_{ZB}}$, and the result of $\overrightarrow{V_{ZB}} \times \overrightarrow{V_{ZA}}$. Set a vector $\overrightarrow{V_{cross}}$ to be the result of whichever of these two cross products has a positive z value. Next, each robot can compute a rotation matrix R_{cross}, which, when multiplied by the vector $\overrightarrow{V_{cross}}$, gives a result with no x and y component. Each robot can then use this R_{cross} rotation matrix to rotate its location vector in the transitional coordinate system, where the location vector is defined as the vector that starts at the origin of the transitional coordinate system and ends at the robot's location in that transitional coordinate system. The x and y values of this rotated location vector are then set as the robot's location in the consistent coordinate system. This location is what the robot uses in the rest of the thesis when referring to a robot's location in the coordinate system, or in the self-organized coordinate system.

5.2.4 Movement

The development of a coordinate system as described above only considers stationary robots; however, in this thesis, it is imperative that robots move in order to assemble the collective shape. If a robot moves, then its current location in the coordinate system becomes incorrect, especially without odometry available. Furthermore, if that robot is a seed, or a reference for a seed, this movement can affect the coordinates of other robots which rely on that seed or reference for localizing themselves.

To prevent movement from negatively impacting robots' locations, this thesis provides a framework to deal with robot movement. The basic idea is that when a robot moves, it loses all previous information about its location in the coordinate system, and it stops being a seed or reference. After the movement is complete, this robot will rely on its neighbors to re-localize itself in the coordinate system.

There are two parts to this framework. The first part controls when a robot is allowed to move. The second part controls how the coordinate system updates after a robot has completed its move.

The first part, which controls the robot movement, is needed to prevent a total loss of the coordinate system. This loss would occur if every robot in the collective moved at once, because they would all lose their coordinate system information, and there would not be any localized neighbors. If this were to occur, then a new coordinate system would form, which would be different from the starting coordinate system. This new coordinate system could possibly result in drastic changes of location for each robot, and as a result of DASH, the collective would reform the shape in a different location, taking time to do so. To prevent this from occurring, only a subset of robots in the collective are allowed to move at any one time. A simple way to implement this is that for every loop of the main controller, if a robot wants to move, allow it to move with a probability of P_{move}. If the value of P_{move} is chosen to be in the range between 0.0 and 0.3, then experiments with this range have been found to prevent total loss of the coordinate system.

The second part of the framework to allow movement of robots without affecting the coordinate system is updating the coordinate system during robot movement. When a robot moves, it clears all its coordinate system information. Once that robot completes its movement, then it will re-localize itself using trilateration. If this robot was a seed before it started moving, then another seed will be elected to take its place. This new seed, call it robot "A", will be elected according to the standard election process outlined in section 5.2.1.2. However, instead of selecting a $\overrightarrow{V_{seed}}$ set to the zero vector, and an R_{seed} set to identity for this new seed, a more intelligent initialization for these values can be chosen. The new $\overrightarrow{V_{seed}}$ should be chosen as the negative of robot A's location in another seed's transitional coordinate system. This choice will place the origin of that new seed's transitional coordinate system in the same location as that of other existing seeds' transitional coordinate systems.

The choice of the R_{seed} for the new seed is chosen so the origin of the new seed's transitional coordinate system is aligned with the origin of other exiting seeds' transitional coordinate systems. The new seed does this by looking at the transitional coordinate system locations for its two reference robots and itself. These locations are in another robot's transitional coordinate system, say seed robot "B". The rotation required

to rotate these three robots' locations in robot A's local coordinate system to their location in B's transitional coordinate system is set as the initial value of R_{seed} for robot "A". This allows an initial choice for R_{seed} that will allow the origin of the new seed robot's transitional coordinate system to be aligned with that of other seeds' transitional coordinate systems.

This method for finding a better initialization for $\overrightarrow{V_{seed}}$ and R_{seed} and the use P_{move} to keep some robots stationary, will prevent the coordinate system from changing during robot movement. This will allow the collective shape to form in a stable manner.

5.2.5 Robot Orientation

The previous sections address how to give a location to each robot described as its (x,y) position in the coordinate system; however, it does not provide the robot information about its orientation in this coordinate system. This orientation describes how the robot's location in the coordinate system will change when the robot commands a "forward" movement. This orientation can be described with two values, θ_x and Ω. θ_x represents the angle between the robot's "forward" direction and the positive x axis of the coordinate system. Ω is a binary variable which represents the "handedness" of the coordinate system. If Ω = SAME, then if the robot where to rotate in a clockwise direction, its direction in the coordinate system would also rotate in a clockwise direction. If Ω=DIFFERENT, then if the robot were to rotate in a clockwise direction, its direction in the coordinate system would rotate in a counter-clockwise direction.

To determine the values of θ_x and Ω, a robot needs to move twice and measure how these movements affect its location in the coordinate system. The system to measure θ_x and Ω works as follows. First, the robot records its current position in the coordinate system, (x_0,y_0). Next, it commands a movement in its "forward" direction by distance D. After this movement, it records its new position, (x_1,y_1). It then rotates by an arbitrary angle that is between the range ±10° to ±170°. This constraint on the angle prevents the three locations measured from being on a line. This rotated angle is recorded as θ_r. After rotating by θ_r, the robot makes another movement in the new "forward" direction by distance D, and records its new position after movement, (x_2,y_2).

After these movements, two values can be calculated, θ_1 and θ_2, as shown in equations 5.2.

$$\theta_1 = atan\left(\frac{y_1 - y_0}{x_1 - x_0}\right) \quad , \quad \theta_2 = atan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{5.2}$$

After the movements, the current angle between the robot's forward direction and the x axis of the coordinate system, θ_x, is set to θ_2. If $\theta_1 + \theta_r = \theta_2$, then Ω is set to the value SAME, else $\theta_1 - \theta_r = \theta_2$ and Ω is set to the value DIFFERENT. These values of θ_x and Ω can also be updated every time the robot moves by remembering the last two previous positions and the previously commanded rotation values.

5.2.6 Sensor Noise

In real world examples, it is possible that the method for sensing robot-to-robot distance is noisy. This noise causes the distance measured to differ from the actual distance. In developing the coordinate system in this thesis, there are two tools used to deal with this sensor noise. These tools are the averaging of multiple sensor readings, and the use of robust quadrilaterals (Moore, et al. 2004).

The first tool is to use averaging of multiple sensor readings between a pair of robots to refine the measured distance between those two robots. This assumes two things, that the sensor noise is an additive zero mean noise, and that multiple measurements can be made in the time between robot movements. If the noise is an additive zero mean noise, then a better estimate of the actual distance is to average a large number of measurements, and use this result as the distance value. This distance value can be further refined until one of the two robots moves, and in that case, the averaging of sensor values will restart when the robot stops moving. This averaging greatly improves the accuracy of measured distances, and the limited robot movement from section 5.2.4 gives robots the opportunity to make multiple measured distance readings to contribute towards these averages.

The second tool used to develop the coordinate system in the presence of sensor noise is to use the robust quadrilateral approach for trilateration. The robust quadrilateral

approach is designed to reduce localization errors due to sensor noise. This is done by not localizing an un-localized robot, robot "A", using three other robots, "E", "F", and "G", if the minimum internal angle of triangles (E,F,G), (A,E,F), (A,E,G), and (A,G,F) is less than α_{min}. For details on this method, see (Moore, et al. 2004).

See chapter 9 for demonstrations of a simulated robotic collective using this method to form a coordinate system.

Chapter 6: DASH, Distributed Self-Assembly and Self-Healing

To review, DASH (distributed self-assembly and self-healing) is a robotic controller that runs on each robot in the collective. The objective of DASH is to self-assemble a desired shape in a robotic collective, and then in the event of damage, self-heal. In the case of DASH, this is all done at a constant (non-varying) scale of the shape. Only with the use of S-DASH, presented in chapter 7, does the scale of the shape vary with the number of robots. This chapter on DASH will consist of seven main sections. The discussion on DASH will first start with the description of the inputs used. Second, the generation a gradient map, which is used by the controller, is discussed. Third, the method each controller uses to determine where it is located in the desired shape will be presented. The fourth section shows how to generate a gradient direction from the gradient map. Fifth, the five modes of robot movement are described. The sixth section gives a description of how the controller chooses which mode to set the robot to. The seventh and final section will analyze the effectiveness and convergence of the controller.

6.1 Inputs to DASH

To function properly, DASH requires four inputs. Those inputs are the desired shape, the desired scale of the shape, the robot location in the self-organized coordinate system, and messages from neighbors. The desired shape is given to each robot in the form of a black and white image, called the shape pixel map. In this shape pixel map, white pixels represent a location in the desired shape, and black pixels represent locations not in the desired shape. Each pixel in the shape pixel map can be indexed by two numbers, (x,y), where x represents how many pixels are above this pixel in the pixel map, and y represents how many pixels are to the left of this pixel in the pixel map. For example, the pixel in the upper left corner of the shape pixel map can be indexed to by (0,0). Fig. 6.1 shows some example shape pixel maps for the desired shape of a filled circle (A), a ring (B), and a teapot (C). There are two constraints to these shape pixel maps. The first constraint is that the desired shape must be a connected shape. A

connected shape is defined as a shape that for every location in that shape, there is at least one path that is fully contained within the shape to all other points in that shape. An example of a non-connected shape pixel map, and therefore a shape that is not allowed, can be seen in Fig. 6.1(D). A second constraint is that all pixels on the outside border of the shape pixel map must be black. Besides these two basic requirements, there are no other fundamental constraints to the types of permissible shape pixel maps.

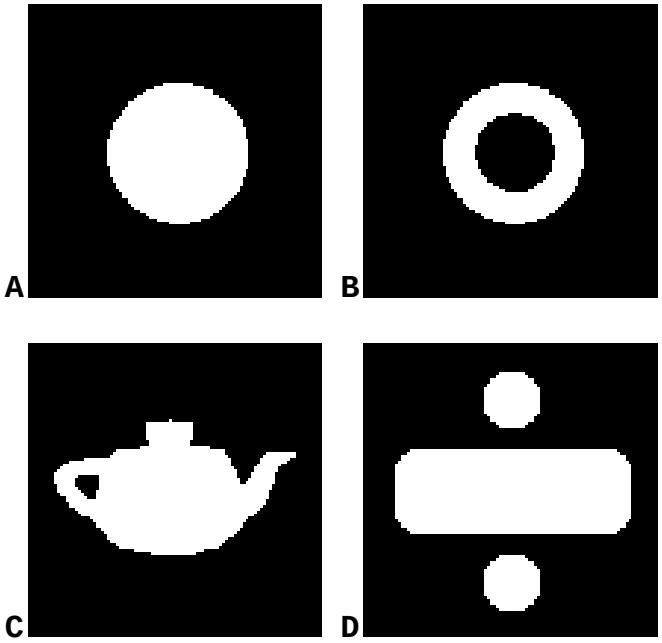


Figure 6.1 (A-C) Examples of acceptable shape pixel maps. (D) An example of an un-acceptable shape pixel map.

The second input to DASH, the scale, represents how big the shape should be when formed by the collective. More precisely, scale is represented by the variable S_f which defines, in terms of the robot radius, R_{robot}, how big each pixel in the pixel map should be when the collective forms the desired shape. For example, consider the desired shape of a square, which is represented by a 3x3 block of white pixels in the shape pixel map. If this desired shape is formed at a scale where S_f=2.0, then each pixel will be $2 \cdot R_{robot}$ wide and $2 \cdot R_{robot}$ long, giving the shape as a whole the dimensions of $6 \cdot R_{robot}$ wide and $6 \cdot R_{robot}$ long.

The third input to DASH is the current location (x,y) of the robot. This current location is taken from the self-organized coordinate system. More details of the self-organizing coordinate system can be found in chapter 5.

6.2 Generation of Gradient Map

Prior to moving the robot, DASH must first convert the shape pixel map into a form that can be used by the controller. This form is called the gradient map. This gradient map only needs to be generated once, when the robot is initially turned on and given the shape pixel map. The method used to generate the gradient map is deterministic, so if every robot is given the same shape pixel map, then they will all generate the same gradient map. To start generating the gradient map, the shape pixel map is segmented into groups of connected pixels with identical colors. Due to the shape pixel map constraints given in section 6.1., there will be only one segment that includes white pixels, which is called the *shape segment.* For example, consider the shape pixel map in Fig. 6.2(A). This shape pixel map has a shape segment that is shown as the solid white segment in Fig. 6.2(B). There will also be one segment called the *external segment,* shown as the solid black segment in Fig. 6.2(B). This external segment includes the pixel (0,0), which is the upper left most pixel in the shape pixel map. Due to the constraint that every pixel on the outside border of the pixel map must be black, the external segment will completely surround the shape segment. There are further possible segments, called *trapped segments*, if there are black pixels in the shape pixel map that are completely surrounded by the shape segment. Three examples of trapped segments are shown as vertical, horizontal and diagonal striped line segments in Fig. 6.2(B).

In each segment, there is one "starting pixel", which is used later to generate the gradient map. For all segments except the external segment, the upper left most pixel in that segment is chosen as a starting pixel. For the external segment, the starting pixel is chosen to be the pixel in the external segment that is immediately above the starting pixel for the shape segment. For example, in Fig. 4.1(A), the starting pixel for the shape segment is (1,1), and the starting pixel for the external segment is (1,0).

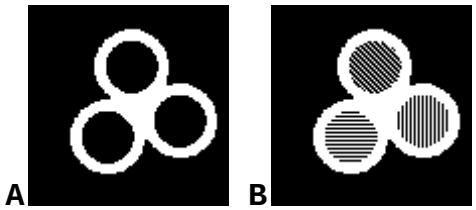
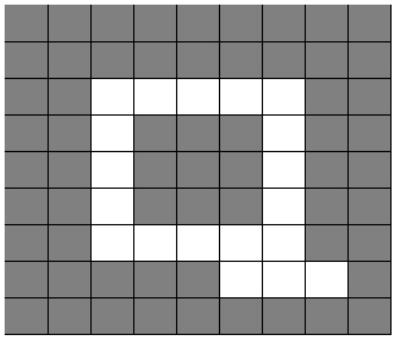


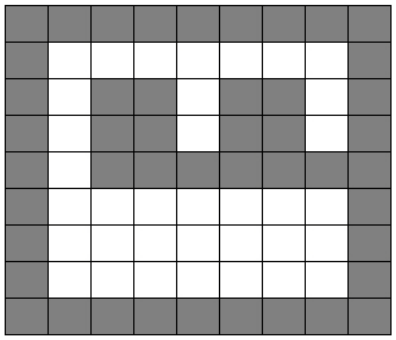
Figure 6.2 (A) The shape pixel map of the desired shape. (B) The shape pixel map segmented into 5 regions.

Once the segments and starting pixels are identified, the gradient map can be created. The gradient map is an array with the same dimensions as the shape pixel map. Furthermore, this gradient map has one integer entry for every pixel in the pixel map. Each location in the gradient map corresponds to the pixel in the same location in the shape pixel map. The values in the gradient map are set as follows. For the external and trapped segments in the pixel map, do the following. For each pixel in those segments, calculate the "Manhattan" distance of the shortest path between that pixel and the segment's starting pixel. This shortest path is constrained to be fully within the corresponding segment. The value for this pixel in the gradient map is then set to -(path_length + 1). For the shape segment, a similar approach is used. For each pixel in the shape segment, calculate the "Manhattan" distance of the shortest path between that pixel and the shape segment's starting pixel. This shortest path is constrained to be fully within the shape segment. The value for this pixel in the gradient map is then set to this path length.

When a gradient map is generated in this manner, any location in the gradient map with a negative value is located in a trapped or external segment. If the gradient value is positive or zero, then it is within the shape segment. When a location has a gradient map value of -1, then it is located at a starting pixel for an external or trapped segment. Two examples of gradient maps generated from a pixel map are shown in Fig. 6.3.



-4	-3	-4	-5	-6	-7	-8	-9	-10
-3	-2	-3	-4	-5	-6	-7	-8	-9
-2	-1	0	1	2	3	4	-9	-10
-3	-2	1	-1	-2	-3	5	-10	-11
-4	-3	2	-2	-3	-4	6	-11	-12
-5	-4	3	-3	-4	-5	7	-12	-13
-6	-5	4	5	6	7	8	-13	-14
-7	-6	-7	-8	-9	8	9	10	-15
-8	-7	-8	-9	-10	-11	-12	-13	-14



-2	-3	-4	-5	-6	-7	-8	-9	-10
-1	0	1	2	3	4	5	6	-11
-2	1	-22	-21	4	-19	-18	7	-12
-3	2	-21	-20	5	-18	-17	8	-13
-4	3	-20	-19	-18	-17	-16	-15	-14
-5	4	5	6	7	8	9	10	-15
-6	5	6	7	8	9	10	11	-16
-7	6	7	8	9	10	11	12	-17
-8	-9	-10	-11	-12	-13	-14	-15	-16

Figure 6.3. (Left) Two examples of a 9x9 shape pixel map. (Right) The gradient map generated from that shape pixel map.

6.3 Location in Shape Pixel Map

The first step for DASH's robot controller is to use the coordinates of the robot, as well as the scale of the shape, to find where the robot is located with respect to the shape, and where the shape is or will be formed. DASH accomplishes this by virtually overlaying the desired shape in the coordinate system. This is done by assuming the upper left pixel in the shape pixel map, pixel (0,0), corresponds to and is centered on the location (0,0) in the coordinate system. All other pixels (x,y) are then centered on the location ($S_f \cdot x \cdot R_{robot}$, $S_f \cdot y \cdot R_{robot}$) in the coordinate system. For example, if the scale is 2.0, then the pixel (1,1) in the shape pixel map is centered at ($2R_{robot}$, $2R_{robot}$) in the coordinate system, and the pixel (2,3) in the shape pixel map is centered at ($4R_{robot}$, $6R_{robot}$) in the coordinate system.

Using this virtual overlay of the shape pixel map in the coordinate system, a robot can use its own coordinates to find which pixel in the shape pixel map is closest to its current coordinates. This closest pixel is considered the robot's current location in the shape pixel map. If this closest pixel is a white pixel, then the robot considers itself inside the shape segment, otherwise it is outside the shape segment.

6.4 Location and Gradient Direction in Gradient Map

After finding the robot's location in the shape pixel map, it is simple to find its location in the gradient map. The robot's location in the gradient map is simply the pixel in the gradient map that corresponds to the robot's current location in the shape pixel map.

Once the controller has determined where the robot is located (x_{index}, y_{index}) in the gradient map, it uses that location, as well as its four neighboring grid locations in the gradient map (up, down, left, right), to determine how to move. Those four neighboring grid locations are used to determine the maximum gradient direction of the gradient map around the robot's current location in the gradient map. The computation of this direction is shown in (6.1), where gm(x,y) returns the gradient map entry for location (x,y).

$$\theta_{gradient} = atan\left(\frac{A}{B}\right) \qquad where, \tag{6.1}$$
$$A = gm(x_{index}, y_{index} + 1) - gm(x_{index}, y_{index} - 1)$$
$$B = gm(x_{index} + 1, y_{index}) - gm(x_{index} - 1, y_{index})$$

If the robot has a x_{index} or y_{index} that is outside the virtual overlay of the shape pixel map, it will not have a valid entry in the gradient map. If that is the case, the maximum gradient direction reported from equation (6.1) will return a direction pointing to the center pixel in the shape pixel map.

6.5 Modes of the Controller

Once the location of the robot in the gradient map and the direction of the gradient are known, the DASH controller can choose one of five modes. This mode can

be one of the following types: gradient following, trapped robot messaging, trapped robot movement, random movement, and stop movement.

6.5.1 Gradient Following Movement Mode

The gradient following movement is a simple movement of the robot in the direction of the maximum gradient direction for the robot's current location. Prior to movement, the robot will first check to make sure that moving in this direction will keep it inside the shape. If it will cause the robot to move outside of the shape, then the movement will not be attempted, and the robot will set a variable named "move_complete" to the value FALSE. If the movement will not move the robot outside of the shape, then the robot first rotates to face the maximal gradient direction, and then commands a movement of distance D_{move}. If the robot attempts this move, (see section 5.2.4 as to why this move might not be attempted) the robot will check to see if the commanded movement was actually accomplished by the robot. This check is done by comparing the coordinates of the robot before movement with its coordinates after movement. If the distance between the before and after coordinates is close to the value D_{move}, then the controller considers the last commanded movement to have been completed. If the robot completed the last commanded movement, then it sets move_complete to TRUE; if the movement was not completed, it sets move_complete to FALSE.

6.5.2 Trapped Robot Messaging Mode

When a robot is stuck in an internal blockade starvation position, (explained in 6.7), the approach in this thesis uses messaging to help un-trap that robot. This messaging consists of generating a "trapped robot message". This message contains the shape pixel map location of the robot ($x_{trapped}$,$y_{trapped}$). It is sent to itself, as well as all neighboring robots who meet the following conditions: first the neighbor's y location in the shape pixel map, ($y_{neighbor}$) should be less than or equal to $y_{trapped}$, and second, the neighbor's x location ($x_{nieghbor}$) should be in the range $(x_{trapped} - 2 \cdot T_{width}) < x_{neighbor} < (x_{trapped} + 2 \cdot T_{width})$, where T_{width} is a predefined constant. Furthermore, if any robot receives the "trapped robot message", no matter what its

current mode, that receiving robot will further propagate the message. The message is propagated in a similar manner, where the receiving robot forwards the message to all of its neighbors that have a y location in the shape pixel map less than that of the receiving robot, and a x location that is in the range $(x_{trapped} - 2 \cdot T_{width}) < x_{new\_neighbor} < (x_{trapped} + 2 \cdot T_{width})$, where $x_{new\_neighbor}$ is the x value of the neighbor of the robot that is propagating the trapped robot message, and $x_{trapped}$ is the location found in the trapped robot message.

This way of generating and propagating the trapped robot message will form a tunnel of robots that receive the message. This tunnel is $2 \cdot T_{width}$ wide, and in the negative y direction from where the robot that started the trapped robot message is located. This is shown in Fig. 6.4, where the robot that started the trapped robot message is green; any other robot that received the trapped robot message is red, and all other robots are black.

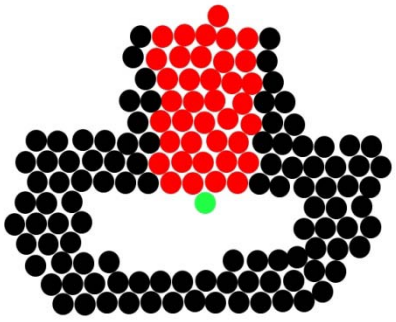


Figure 6.4. Receivers, shown in red, of the trapped robot message, generated from the green robot.

6.5.3 Trapped Robot Movement Mode

The trapped robot movement is also used to help move a robot in the collective out of an internal blockade starvation position. This is done by attempting to move the robot from its current shape pixel location of (x,y) to a shape pixel with a lesser y value, such as the location of (x,y-1). With this movement, the robot first rotates to face toward the shape pixel (x,y-1), and then commands a movement of distance D_{move}. If the robot attempts this move (see section 5.2.4 as to why this move might not be attempted), the

robot will check to see if the commanded movement was actually accomplished. This check is done by comparing the coordinates of the robot before movement, with its coordinates after movement. If the distance between the before and after coordinates is close to the value D_{move}, then the controller considers the last commanded movement to have been completed. If the robot completed the last commanded movement, then it sets the variable move_complete to TRUE; if the movement was not completed, it sets move_complete to FALSE.

6.5.4 Random Movement Mode

The desired effect of random movement mode is simply to move in a random direction, without taking the robot out of the shape, for D_{move} distance. To do this, the robot first generates a random direction to move. If the robot is inside the desired shape, and moving D_{move} distance in that random direction will move it out of the desired shape, then the robot does not move, and just sets its move_complete to TRUE. If the robot is not in the desired shape, or the random move will not take it outside the desired shape, then DASH commands the robot to rotate to face the random direction of movement, and then to move D_{move} distance forward. If the robot attempts this move, (see section 5.2.4 as to why this move might not be attempted), then unlike previous movements, it sets its move_complete to TRUE, without checking if the move was actually completed.

6.5.5 Stop Movement Mode

In this movement mode, the robot simply stops moving. In this mode, move_complete is always set to TRUE.

6.6 Choice of Mode

Using DASH, each robot monitors four variables to determine which of the five modes it should be operating in currently. The first variable is “gradient_map_value”, which represents the current value of the gradient map pixel at which the robot is currently located. The second variable is “in_trapped_segment”, which is TRUE if the robot is currently in a trapped segment, as defined in section 6.2, and FALSE otherwise. The third variable is “move_complete”, which was defined in section 6.5. The fourth

variable is called “trapped_robot_message_recieved” which is set to FALSE if it has been longer than 2 cycles of the controller since a trapped robot message has been received. If the robot has received a trapped robot message in less than 2 cycles of the controller program, then it will set the “trapped_robot_message_recieved” variable to one of two values. If the robot has a x value in the pixel shape map that is $(x_{trapped} - T_{width})<x<(x_{trapped} + T_{width})$, where $x_{trapped}$ is the value in the trapped robot message, then it will set the “trapped_robot_message_recieved” variable to MOVE. Otherwise, it will set the “trapped_robot_message_recieved” variable to STOP.

There are five modes in which the controller can operate. Each mode is described in section 6.5. These modes are gradient_following_movement_mode (section 6.5.1), trapped_robot_messaging_mode (section 6.5.2), trapped_robot_movement_mode (section 6.5.3), random_movement_mode (section 6.5.4), and stop_movement_mode (section 6.5.5). The mode is chosen as follows. First, if the robot’s current gradient map value is -1, and the robot is in a trapped segment in the shape pixel map, then the robot will go into trapped_robot_messaging_mode. This messaging mode can occur before a movement mode in any time step. Next, if move_complete is FALSE, the robot will go into random_movement_mode. If move_complete is TRUE, one of three modes is chosen, based on the variable trapped_robot_message_recieved. If trapped_robot_message_recieved is FALSE, the robot will go into the gradient_following_movement__mode. If trapped_robot_message_recieved is MOVE, the robot will go into trapped_robot_movement_mode. If trapped_robot_message_recieved is STOP, the robot will go into stop_robot_movement_mode. These mode choices are summarized in pseudo-code in Fig. 6.5.

```
if( gradient_map_value = -1 ) and ( in_trapped_segment = TRUE )
{
       trapped_robot_messaging_mode()
}
if( move_complete is TRUE )
{
       if( trapped_robot_message_received = FALSE )
       {
              gradient_following_movement_mode()
       }
       else if( trapped_robot_message_received = MOVE )
       {
              trapped_robot_movement_mode()
       }
       else if( trapped_robot_message_received = STOP )
       {
              stop_movement_mode()
       }
}
else if( move_complete is false )
{
       random_movement_mode()
}
```

Figure 6.5. Pseudo-code for how the DASH controller chooses its mode in every time step.

6.7 Analysis

The goal of DASH is for the collective to form and heal a desired shape at a given scale. For the robot collective to have fully formed or healed the desired shape, every robot must be in a location that is within the shape when the scale is chosen appropriately with respect to the number of robots in the collective. To show that all robots will move into the shape, it will first be shown that without considering interference from other robots, a robot running DASH in any location will always move into the shape. Next it will be shown that even when other robots are in the environment, a robot running DASH will still make it into the shape, no matter its location.

6.7.1 Single Robot Case

First, when a robot is on its own, there is no possibility of collisions or blocking from other robots. In this case, a single robot will always operate in gradient_following_mode or trapped_robot_movement_mode, because there are no

collisions with other robots to trigger random_movement_mode, and no messages that will trigger stop_movement_mode.

In this case, if the robot is ever located in a trapped segment, and the gradient value is -1, then it will operate in trapped_robot_movement_mode, and move in the negative y direction (move up in the shape pixel map). Because the location in trapped segments with a gradient of -1 is the location of that trapped segment's seed, this location will be the upper left most location in that trapped segment; therefore, moving up will immediately move the robot into the shape segment.

If a solitary robot is located anywhere in a trapped segment other than where the gradient value is -1, then the robot will operate in the gradient_following_mode. In this mode, the solitary robot will move in a direction that increases the gradient value of its location. Given there are no local maxima in the gradient map for the non-shape segments, then this mode of movement will always place the robot inside the shape segment, where the only local maxima of the gradient are located, or in the location of a trapped segment's seed. It was shown above that being at a trapped segment's seed will always lead to moving into the shape, so all that needs to be shown is that there are no local maxima outside the shape segment. That proof is as follows.

Proposition: There are no local maxima in the gradient map for trapped or external segments.

Proof: The starting seed, S, for the external or trapped segment is not a local maximum, because it is always immediately adjacent to a location in the shape segment which has a higher value in the gradient map. For any other location, L, in the segment, there is a shortest path $L \to L_1 \to L_2 \to \cdots \to S$ from L to the starting seed S, where L_1 is an immediate neighbor of L. Due to the fact that every sub-path of a shortest path is also a shortest path for its respective start and finish points, the shortest path from L_1 is $L_1 \to L_2 \to \cdots \to S$, which is 1 less than the shortest path from L. The values in the gradient map for the external or trapped segments are set to be - (the length of the shortest path from that location to the starting seed + 1), so every value in the gradient map for this segment must have an immediate neighbor in the gradient map with a higher value, and therefore is not a local maximum. ■

Once the single robot is inside the shape segment, it will only use the gradient_maximization_movement or random_movement_mode, which are defined as movements that will not move the robot out of the shape (see section 6.5). Now that it has been shown that a single robot will always move into the shape, the case of multiple robots will be addressed.

6.7.2 Multi-Robot Case

The second possible case to look at which shows that every robot will move into the shape, is when more than one robot is trying to move into, or stay in, the desired shape. This differs from the single robot case because neighboring robots may impede the motion of any one robot, blocking its entrance into the shape. This blocking can be divided into two categories: local blocking, and blockade starvation.

Local blocking occurs when a robot is unable to move into a desired position directly due to interference from a neighboring robot, but a small detour would allow the robot to reach this position. For example, in Fig. 6.6, robot 1 would like to move to the position marked by the grey circle labeled 2; however, it is locally blocked, as direct movement is not possible, due to collision with the robot labeled 3. This local blocking can be resolved with simple random motion. For instance, in Fig. 6.6 when the collision between 1 and 3 occurs, the controller for robot 1 will switch to random movement mode and execute a movement in a random direction. Eventually, this random movement will allow the robot to move away from robot 3, and then the robot can move directly into the location labeled 2.

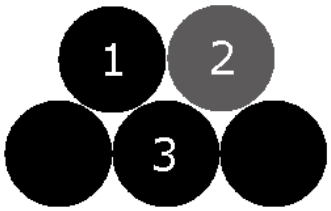


Figure 6.6. Robot in local blocking. Black circles are robots, and the grey circle labeled 2 is a desired location for the robot labeled "1".

The other category of blocking, called *blockade starvation*, can also prevent robots from entering the desired shape. Blockade starvation is when an area in the shape segment cannot be filled by a robot because the behavior of other robots inside the shape prevents the robot from reaching this area. There are two types of blockade starvation: internal and external. An example of external blockade starvation is shown in Fig. 6.7(A). In this form of blockade starvation, a robot in the external segment cannot move inside the shape, thus preventing the shape from being fully formed. An example of internal blockade starvation is shown in Fig. 6.7(B). Here, a robot in a trapped segment is not capable of entering the desired shape, also preventing the shape from fully forming.

When external blockade starvation occurs, the empty area in the desired shape does not include any location that corresponds to a local minimum of the gradient map. This is because the generation of the gradient map in the shape segment guarantees that there is no local minimum, which can be proved as follows.

Proposition: There is no local minimum in the gradient map for the shape segment.

Proof: The starting seed, S, for the shape segment is not a local minimum, because it is always immediately adjacent to the starting seed for the external segment, which has a lower value in the gradient map. For any other location, L, in the shape segment there is a shortest path $L \to L_1 \to L_2 \to \cdots \to S$ from L to the starting seed S, where L_1 is an immediate neighbor of L. Due to the fact that every sub-path of a shortest path is also a shortest path for its respective start and finish points, the shortest path from L_1 is $L_1 \to L_2 \to \cdots \to S$, which is 1 less than the shortest path from L. The values in the gradient map for the shape segment are set to be the length of the shortest path from that location to the starting seed, so every value in the gradient map must have an immediate neighbor with a lower value, and therefore is not a local minimum. ■

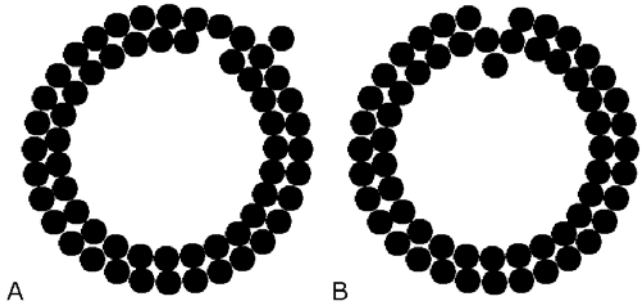


Figure 6.7. (A) Example of external, and (B) internal blockade starvation for the desired shape shown in Fig. 6.1(B).

As described above, the empty area in the shape segment cannot contain a location corresponding to a local minimum in the gradient map. If a robot immediately borders this empty area, and that robot has a corresponding gradient map entry lower than that its neighboring empty space (which is part of the empty area), then by the gradient_following_movement_mode, the robot will move into the empty space. Robots will continue to move into the empty area until either the empty area is filled with robots, or there are no robots next to the empty area that have a gradient map value less than that of the neighboring empty spaces. If the latter is true, then, because there are no local minima, the empty area must include the starting pixel for the shape segment. This starting pixel for the shape segment is always on the outside of the shape segment, which a robot in the external segment can reach. If this is the case, this empty space is no longer an instance of external blockade starvation.

When internal blockade starvation occurs, a robot is trapped is in a trapped segment. By using the gradient_following_movement mode, it will eventually reach the starting seed for that segment, where the gradient is -1. Once it reaches this point, according to the DASH controller from section 6.6, it will start transmitting the trapped robot message. This message will create a “tunnel” above the trapped robot, which will make space for the trapped robot inside the shape segment. This tunnel formation can be seen in Fig. 6.8. The tunnel has 2 parts. The first part is the inside portion, where robots move upward, making space for the trapped robot. These robots are colored blue in Fig. 6.8. These upward moving robots have an x value that is in the range

$(x_{trapped} - T_{width})<x<(x_{trapped} + T_{width})$, where $x_{trapped}$ is the x value of the trapped robot. The second part of this tunnel is the wall of the tunnel, where robots stop moving. These robots are shown in red in Fig. 6.8. The purpose of these robots is to prevent robots other than the trapped robot from entering the tunnel. These wall robots have an x value that is in the range $(x_{trapped} - 2 \cdot T_{width})<x<(x_{trapped} + 2 \cdot T_{width})$, but not in the range $(x_{trapped} - T_{width})<x<(x_{trapped} + T_{width})$. Eventually, this tunneling will cause the trapped robot, shown as green in Fig. 6.8., to move into the shape, removing the internal blockade starvation. Once the trapped robot is inside the shape, it will stop sending the trapped robot message, which in turn will cause the tunnel to disappear. Fig. 6.8(A) shows a trapped robot generating the trapped robot message. Next, in Fig. 6.8(B-C) the tunnel creates space for the trapped robot, and finally in Fig. 6.8(D) the trapped robot is able to move into the shape segment.

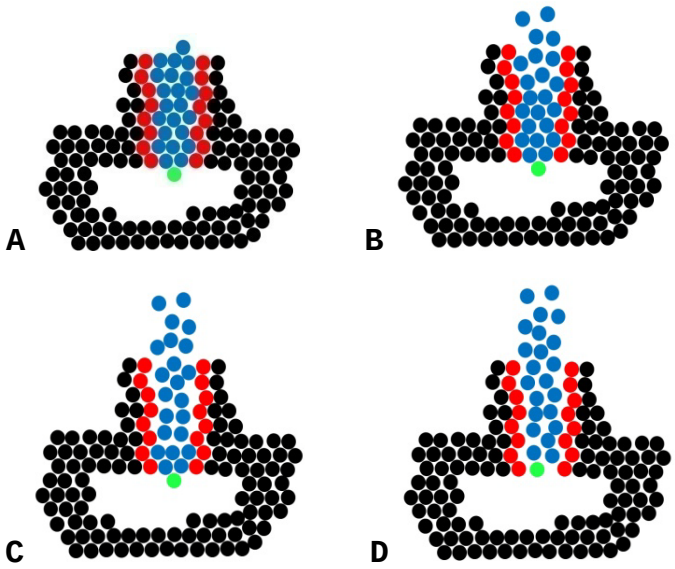


Figure 6.8. Tunneling to remove an internal blockade starvation condition.

There are two side effects as a result of this behavior to remove the internal blockade starvation. The first is that it may introduce multiple empty locations inside the shape above where the trapped robot was located; however, these locations are easily filled with robots using the gradient_following_movement once the tunnel disappears.

The second side effect is that while some robots were moving to create space for the trapped robot in the tunnel, they may have moved into an external or trapped segment. If they moved into the external segment, the gradient_following_movement will direct them back into the desired shape. If they moved into a trapped segment and cannot move back into the desired shape, they will also need to use this trapped robot behavior by generating another trapped robot message. It is important to note that while this trapped robot behavior may create more trapped robots, it only creates them in locations above the original trapped robot. This means that there is no feedback cycle where a robot trapped in a specific segment will cause more trapped robots in that segment. Without this problem of feedback, the trapped robot behavior will quickly cause the number of trapped robots to reduce to zero.

Now it has been shown that for both the single robot and the multi-robot cases, any robot, no matter its position, will eventually reach a location that is in the desired shape by using DASH. This means that DASH provides a means for self-assembling and self-healing a collective's shape. Examples of DASH forming a desired shape in a simulated robotic collective can be found in chapter 9.

Chapter 7: S-DASH, Scalable Distributed Self-Assembly and Self-Healing

Scalable distributed self-assembly and self-healing, or S-DASH, is a method that runs in each robot, in addition to DASH, to dynamically vary the scale of the shape in order to reflect the number of robots that are currently in the collective. It does this by providing DASH an appropriate scale value, instead of the constant scale value used in chapter 6. To accomplish this scalable self-assembly and self-healing, this S-DASH controller has three main parts. The first part is a method to come to a consensus with the other robots in the collective as to the scale of the shape. The second part is a way to reduce the scale of the shape if the shape is too big for the current number of robots in the collective. The final part of S-DASH is to increase the scale of the shape if the shape is too small for the current number of robots. Besides presenting these three parts of S-DASH, this chapter will also present analysis of S-DASH's ability to dynamically vary the scale of the shape.

7.1 Scale

Each robot has a variable, S_f, which represents that robot's belief as to the appropriate scale of the shape. This variable is used in chapter 6, 7, and 8 to determine the robot's location in the shape. To properly form the shape, this value of S_f must be updated so that it is consistent among all robots, i.e. every robot agrees to the same scale. Furthermore, the S_f value should be able to change, so that the scale can adjust to changes in the number of robots. To accomplish this consistency and flexibility of S_f, its value will be updated as follows. Every time step, each robot sets its value of S_f to be the average S_f value of all its neighbors, including its own value. After this update, each robot will communicate its new S_f value to its neighbors. This update, summarized in Fig. 7.1, is applied once every cycle of the robot's main loop, unless the conditions described in sections 7.2 and 7.3 apply.

```
Total_scale = Sf
Number_neighbors = 1
For( all neighbors i)
{
       Total_scale = Total_scale + neighbor_i's_Sf
       Number_neighbors = Number_neighbors + 1
}
Sf = Total_scale / Number_neighbors
```

Figure 7.1. Method for scale update.

7.2 Scale Reduction

For the reduction of the scale, S-DASH uses the fact that when using DASH to form a shape, the seed location for the shape segment is the last location to be filled in the desired shape because it has the lowest gradient value of all locations within the shape. This means that if the shape is too big, then the shape seed location will continuously be un-occupied by robots. For example, consider the desired shape map given in Fig. 7.2(A). In this shape, the seed for the shape segment is located at the very top of the desired shape, which in this case is the lid of the teapot. When the scale is too big for the number of robots, as shown in Fig. 7.2(B), then there will not be any robots near the seed of the shape segment.

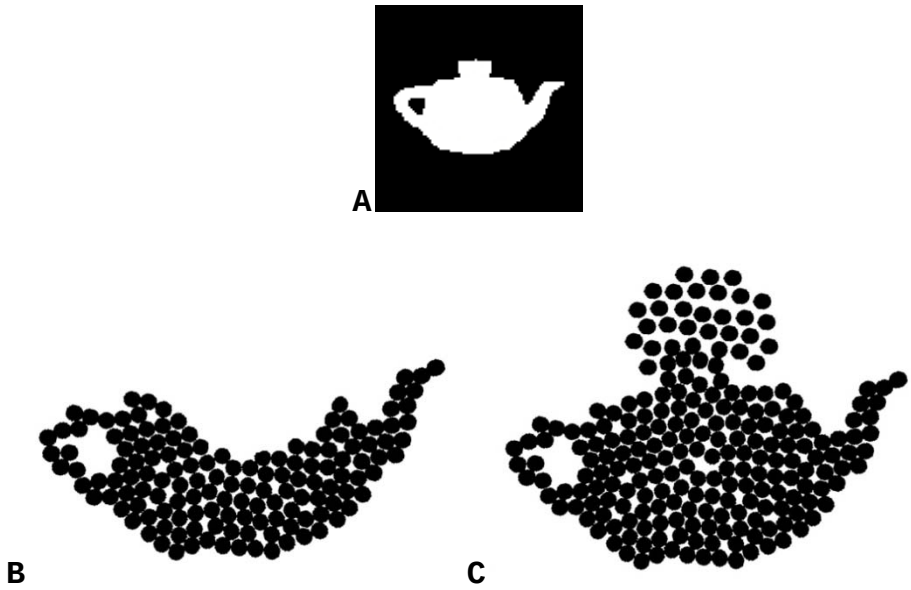


Figure 7.2. (A) The desired shape pixel map, a teapot. (B) The collective forming the desired shape at too large a scale. (C) The collective forming the desired shape at too small a scale.

To reduce the scale after observing the continuously unoccupied shape segment seed, S-DASH must do three things. First, it must have a distributed mechanism to alert all robots when the starting seed is not occupied. Second, it must have a way to determine how long to wait before the seed is considered continuously un-occupied. Thirdly, it must have a mechanism for reducing the scale by an appropriate amount.

7.2.1 Detecting Un-Occupied Seed

The distributed mechanism for detecting an un-occupied shape segment seed is based on a variable called $T_{un\text{-}occupied}$. This variable is updated in the following way. Every loop of the controller main loop, a robot's value of $T_{un\text{-}occupied}$ is compared to that of all its neighbors. If any neighbor has a value lower than the robot's, the robot will set its $T_{un\text{-}occupied}$ value to be its neighbors' value + 1. If none of the neighbors' values are lower, then the robot will set its new value of $T_{un\text{-}occupied}$ to be 1 plus its old value. If the robot is located in the seed location of the shape segment (see section 6.2), then it sets its $T_{un\text{-}occupied}$ value to be zero. Fig. 7.3. summarizes how $T_{un\text{-}occupied}$ is updated. After $T_{un\text{-}occupied}$ is updated, the new value is communicated to all neighboring robots.

```
if( robot is in seed location for shape segment )
{
        T_un-occupied = 0
}
else
{
        Temp = T_un-occupied
        For( all neighbors )
                If( neighbors T_un-occupied < Temp )
                {
                        Temp = neighbors T_un-occupied
                }

        T_un-occupied = Temp
}
T_un-occupied = T_un-occupied + 1
```

Figure 7.3. Method for updating $T_{un\text{-}occupied}$.

Updating $T_{un\text{-}occupied}$ in this manner will give the following effects. If any robot is in the seed location for the shape segment, then every robot in the collective will have a value of $T_{un\text{-}occupied}$ which is equal to the number of communication hops from that robot to the one in the starting seed location. If no robots are in the seed location, then every robot in the collective will start increasing their value of $T_{un\text{-}occupied}$ by one, once for every loop of the main controller. Eventualy one or more robots in the collective will have their $T_{un\text{-}occupied}$ value reach the value $T_{un\text{-}occupied\text{-}max}$. Once this occurs, the seed has been un-occupied for long enough for S-DASH to consider the shape too big. How $T_{un\text{-}occupied\text{-}max}$ is chosen is explained in section 7.2.2.

7.2.2 Wait Time

Before reducing the scale of the shape because the shape seed location is not occupied by any robots, S-DASH must first make sure that all the robots in the collective are currently inside the shape. If any robots are outside the shape and moving to get into the shape, then the method in section 7.2.1 might pre-maturely decide that the scale of shape is too big, where in fact, if those robots moved into the shape, it would be the right scale. S-DASH makes sure all robots are in the shape by giving them ample time to move into the shape. The time is set through the use of the $T_{un\text{-}occupied\text{-}max}$ variable. The value of $T_{un\text{-}occupied\text{-}max}$ is based on four factors, and represents the upper bound of the time

a robot will take to get into the shape. The first factor is the current scale value, or S_f. The second factor is the speed of robot movement, V_{robot}. The third factor is the probability that the robot will move if commanded to do so, P_{move}, where P_{move} is defined in section 5.2.4. The fourth factor is the maximum external path length, or $L_{external\_path}$, which is defined as follows. For all pixels that are on the boundary between the shape segment and the external segment, find the shortest path between that pixel and the starting pixel which stays outside of the shape segment, and set $L_{external\_path}$ to be the largest of these shortest paths. An example of the longest, shortest path for the shape pixel map in Fig. 7.2(A) is shown as the red line in Fig. 7.4(A). With these four factors known, $T_{un\text{-}occupied\text{-}max}$ is computed to be $T_{un\text{-}occupied\text{-}max} = (S_f \cdot L_{external\_path}) / (V_{robot} \cdot P_{move})$. This is the worst case distance that a robot must travel to get into the empty space in the shape segment, $(S_f \cdot L_{external\_path})$ divided by the average speed at which the robot travels, $(V_{robot} \cdot P_{move})$.

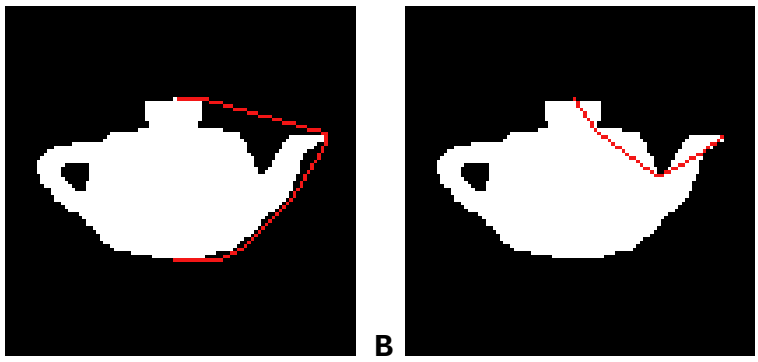


Figure 7.4. (A) Visualization of $L_{external\_path}$. (B) Visualization of $L_{internal\_path}$.

7.2.3 Reducing Scale

Once the collective has waited long enough for the starting seed location to be filled, the collective should go ahead and reduce the scale. This scale reduction is initiated by any robot whose $T_{un\text{-}occupied} \geq T_{un\text{-}occupied\text{-}max}$. The robot or robots that initiate the scale reduction do two things. The first thing they do is set their $T_{un\text{-}occupied}$ value to zero. This has the effect of inhibiting other robots from further trying to reduce the scale by resetting the $T_{un\text{-}occupied}$ values in the collective. The second thing that the robot or

robots initiating scale reduction do, is that for $T_{un\text{-}occupied\text{-}max}$ / 2 cycles of the main controller loop, they don't use the scale update from section 7.1; instead they report to their neighbors that their scale is a new, lower scale, S_{f_new}, the value of which will be shown shortly. This has the effect of decreasing the S_f values of all the robots in the collective. After the $T_{un\text{-}occupied\text{-}max}$ / 2 cycles, the scale update returns to the method from section 7.1.

7.2.3.1 Choosing a New and Smaller Scale

When the scale is being reduced, it is known that there is at least one pixel of the shape segment that is un-occupied by robots, which is the starting pixel of the shape segment. The area that this pixel takes up in the environment is equal to $(S_{f\_old} \cdot R_{robot})^2$, where S_{f_old} is the scale of the shape prior to the start of reduction, and R_{robot} is the radius of a robot. Because this pixel is un-occupied, the scale of the shape can safely be reduced so that the area of the shape at the new scale is equal to the area at the old scale, minus $(S_{f\_old} \cdot R_{robot})^2$, the area of one pixel. The area of the shape at the old scale is equal to NUM_PIX $\cdot$ $(S_{f\_old} \cdot R_{robot})^2$, and the area of the shape at the new, smaller scale is equal to NUM_PIX $\cdot$ $(S_{f\_new} \cdot R_{robot})^2$. Therefore, the new scale, S_{f_new}, should be $S_{f\_old} \cdot \sqrt{1 - \frac{1}{\text{NUM\_PIX}}}$, where NUM_PIX is the number of pixels in the shape pixel map's shape segment.

This process of reducing the scale when the starting pixel is unoccupied will continue until the starting pixel is occupied.

7.3 Scale Increase

To increase the scale of the shape, S-DASH uses the fact that when DASH is used to form a shape, the seed location for the external segment is the first location to be filled by robots if the shape segment is completely filled by robots. This means that if the shape is too small, then the external segment seed location will continuously be occupied by robots. For example, consider the desired shape map given in Fig. 7.2(A). In this shape, the seed for the external segment is located just above the very top of the teapot. When the scale is too small for the number of robots, as shown in Fig. 7.2(C), then there will continuously be robots in the external segment seed location trying to get into the shape segment.

To increase the scale after observing the continuously occupied external segment seed, S-DASH must do three things. First, it must have a distributed mechanism to alert all robots when the external segment seed is continuously occupied. Second, it must have a way to determine how long to wait before the seed is considered continuously occupied. Thirdly, it must have a mechanism for increasing the scale by an appropriate amount.

7.3.1 Detecting Occupied Seed

The distributed mechanism for detecting a continuously occupied external segment seed is based on a variable called $T_{occupied}$. This variable is updated in the following way. Every loop of the controller main loop, a robot checks to see if it is in the external segment seed (as defined in section 6.2). If so, it increases its $T_{occupied}$ value by one. If the robot receives a message called "moved_into_shape_message", then the robot will set $T_{occupied}$ to zero. Furthermore, when a robot's $T_{occupied}$ is greater than $T_{occupied\text{-}max}$, then the external seed is considered to have been occupied for long enough, and that robot will initiate a scale increase, as well as transmit the moved_into_shape_message.

The moved_into_shape_message is initiated by one of two events. The first event is when a robot moves from the external segment into the shape segment. The second event is when a robot's $T_{occupied}$ is greater than $T_{occupied\text{-}max}$. In both these events, the purpose of this message it to reset the $T_{occupied}$ value of the robots in the external segment. Furthermore, to prevent infinite loops of this message, it contains a hop count value,

which is used to give the message a limited life time. Whenever a neighbor receives a moved_into_shape_message, it will decrease the hop count in the message by one, and then only if the hop count is greater than zero, will it re-transmit the message. The robot that creates the moved_into_shape_message initializes the hop count to have the value $5 \cdot \frac{S_f \cdot R_{robot}}{R_{com}}$. This initialization of the hop count will allow the message to travel far enough to clear the $T_{occupied}$ value for the robots near the external segment seed.

7.3.2 Wait Time

Before increasing the scale of the shape because the external segment seed contains robots, S-DASH must first make sure that enough time is given for DASH to fill any empty locations in the shape segment, for example those caused by external blockade starvations. This is because if there are any empty spaces in the shape segment, then the method section in 7.3.1 might pre-maturely decide the scale of shape is too small, where in fact if robots moved into those empty spaces, it would be the right scale. S-DASH makes sure there are no empty spaces in the shape segment by giving DASH ample time to move robots into those spaces. This time is set through the use of the $T_{occupied\text{-}max}$ variable.

The value of $T_{occupied\text{-}max}$ is based on four factors, and represents the upper bound of the time DASH will take to fill empty volumes in the shape segment with robots. The first factor is the current scale value, or S_f. The second factor is the speed of robot movement, V_{robot}. The third factor is the probability that the robot will move if commanded to do so, P_{move}. The fourth factor is the maximum internal path length, or $L_{internal\_path,}$ which is defined as follows. For all pixels that are on the boundary between the shape segment and the external segment, find the shortest path between that pixel and the shape segment starting pixel which stays inside the shape segment, and set $L_{internal\_path}$ to be the largest of these shortest paths. An example of the longest, shortest path for the shape pixel map in Fig. 7.2(A) is shown as the red line in Fig. 7.4(B). With these four factors known, $T_{occupied\text{-}max}$ is computed to be $T_{occupied\text{-}max} = (S_f \cdot L_{internal\_path}) / (V_{robot} \cdot P_{move})$. This is the worst case distance that a robot

must travel to get into the empty volume in the shape segment, ($S_f \cdot L_{internal\_path}$) divided by the average speed at which the robot travels, ($V_{robot} \cdot P_{move}$).

7.3.3 Increasing Scale

Once the collective has waited long enough for the empty volumes in the shape segment to be filled, and if the seed location in the external segment is still occupied, the collective should go ahead and increase the scale. This scale increase is initiated by any robot whose $T_{occupied} \geq T_{occupied\text{-}max}$. The robot or robots that initiate the scale increase do two things. The first thing they do is send out a moved_into_shape_message. This has the effect of inhibiting other robots from further trying to increase the scale by resetting the $T_{occupied}$ values of robots in the collective. The second thing that the robot or robots initiating scale increase do, is that for $T_{occupied}/2$ cycles of their main controller loop, they don't use the scale update from section 7.1; instead they report to their neighbors that their scale is a new, larger scale, S_{f_new}, the value of which will be shown shortly. This has the effect of increasing the S_f values of all the robots in the collective. After the $T_{occupied}/2$ cycles the scale update returns to the method from section 7.1.

7.3.3.1 Choosing a New and Larger Scale

When a robot decides to increase the scale, it should increase it enough so that there will be room in the shape for an extra robot. The amount of room this extra robot takes up is $\pi \cdot R_{robot}{}^2$, so the area of the shape should be increased by that amount. To increase the area of the shape by that amount, the scale S_f must change from the old value, S_{f_old}, to a larger value, S_{f_new}, where $S_{f\_new} = \sqrt{S_{f\_old}{}^2 + \frac{\pi}{NUM\_PIX}}$.

This process of increasing the scale will continue as long as robots occupy the seed location in the external segment.

7.4 Analysis

For the methods in S-DASH to indeed result in the formation of the shape at the desired scale, not only should the collective adjust the size of its shape based on the number of robots as described in sections 7.1-7.3, but it should do so in a stable manner.

For S-DASH to form the scale in a stable manner, the scale should converge to a stable value, i.e. should not oscillate, assuming the number of robots in the collective does not change. This stability of the scale from S-DASH is dependent on three variables: $T_{occupied\text{-}max}$, $T_{un\text{-}occupied\text{-}max}$, and S_{f_new}.

If $T_{occupied\text{-}max}$ and $T_{un\text{-}ccupied\text{-}max}$ are chosen to be too small, then the robots that change the scale of the shape will do so too quickly. If this choice is too quick, then the effects of a scale change will not have time to reach the robots before they decide to change the scale again. For an example of this, in Fig. 7.5(A) the robot shown in red is outside the shape, and as a result increases the scale of the shape. As shown in Fig. 7.5(B-C), the rest of the robots start making room for the red robot by moving to new locations as the result of larger scale, and this extra space starts moving towards the red robot. However in Fig. 7.5 (D), because of a value of $T_{occupied\_max}$ that is too small, the red robot again increases the scale of the shape. By the time the effects of both shape increases make it to the red robot in Fig. 7.5(E), the shape is too big for the number of robots. This then leads to the scale being decreased, and if $T_{un\text{-}ccupied\text{-}max}$ is also too small, a similar result would occur, where the scale ends up smaller than needed, with some robots outside the shape. This process would oscillate back and forth, never allowing the shape to converge to a constant scale.

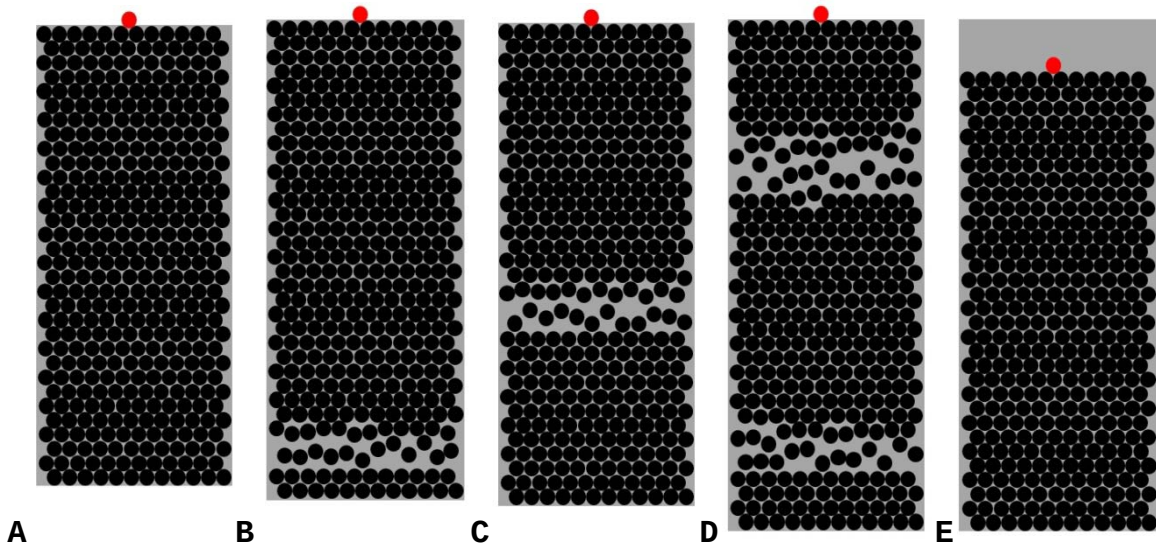


Figure 7.5. (A) A robot (red) outside the shape (grey) increases the scale. (B-C) Robots in the collective start to move. (D) The robot increases the scale again. (E) The robot finally moves into the shape (grey), which is now too big.

To prevent this oscillation, the values of $T_{occupied\text{-}max}$ and $T_{un\text{-}ccupied\text{-}max}$ are chosen based on how long it takes the effects of a change in shape to propagate back to the location where the robot that originated the shape change was located. For example, as the result of a scale decrease, it is possible for robots that were once inside the shape to find themselves on the outside of the shape. To propagate the effects of this scale decrease, i.e. more of the shape is filled by robots, these robots need to move back inside the shape. In the worst case, these robots need to travel $S_f \cdot L_{external\_path}$ distance to get back inside the shape, and the expected value of the time it takes the robots to travel this far is $(S_f \cdot L_{internal\_path}) / (V_{robot} \cdot P_{move})$. This means that if the robots set $T_{un\text{-}occupied\text{-}max} = (S_f \cdot L_{external\_path}) / (V_{robot} \cdot P_{move})$, then even in the expected worst case, this will still be enough time for the effects of a scale decrease to make it to the starting seed of the shape segment.

As the result of a scale increase, it is possible for robots that were inside the shape to move to a new location that is now also inside the shape, but before the scale change, was not. This movement can create an empty location inside the shape segment, which another robot that borders this empty location will move into. As robots continue to

move into newly formed empty locations, they will in effect cause the empty location to move in the shape segment, towards the starting seed of the external segment. The worst case distance that this empty location must travel to reach the location of the robot that initiated the scale increase is the longest internal path of the shape segment. This path has a distance of $S_f \cdot L_{internal\_path}$. Because this empty hole moves at the same rate as the robots, it is expected to take approximately $(S_f \cdot L_{internal\_path}) / (V_{robot} \cdot P_{move})$ time to travel this far. This means that if the robots set $T_{occupied\text{-}max} = (S_f \cdot L_{internal\_path}) / (V_{robot} \cdot P_{move})$, then in the expected worst case, there will still be enough time for the effects of a scale increase to make it to the starting seed for the external segment.

The choice of S_{f_new} can also affect the stability of the scale value. In the case of a decrease in scale, if the S_{f_new} is chosen incorrectly, then the resulting new shape could be too small for the number of robots currently in the collective. To choose this S_{f_new} correctly when decreasing the scale, the area of the shape must not be decreased by more than the unoccupied space in the shape. Due to the mechanism for decreasing the scale of the shape, it is known that there is at least one pixel of the shape pixel map that is unoccupied (the starting pixel for the shape segment). This means that there is at least $(S_{f\_old} \cdot R_{robot})^2$ worth of empty space in the shape segment, where S_{f_old} is the scale of the shape before reduction. To reduce the shape size by this area, the scale should be set to $S_{f\_old} \cdot \sqrt{1 - \frac{1}{\text{NUM\_PIX}}}$. By reducing the scale by this amount, the scale can be decreased without the risk of reducing it too small.

Similarly, in the case of an increase in scale, if the S_{f_new} is chosen incorrectly, then the resulting new shape could be too large for the number of robots currently in the collective. To choose this S_{f_new} correctly when increasing the scale, the area of the shape must not be increased by more than the area that will be occupied by the robots that are currently outside the shape. When the scale of the shape is increased, it is known that there is at least one robot outside the shape segment. This means that if the area of the shape is increased by $\pi \cdot R_{robot}^2$, there will be robots available to fill this space. To increase the shape size by this area, the scale should be set to $\sqrt{S_{f\_old}^2 + \frac{\pi}{\text{NUM\_PIX}}}$. By

increasing the scale by this amount, the scale can be increased without the risk of increasing it too much.

By following these methods for choosing $T_{occupied\text{-}max}$, $T_{un\text{-}occupied\text{-}max}$, and S_{f_new}, S-DASH will reach a stable scale appropriate to the number of robots in the collective. Examples of S-DASH adapting the shape scale to the number of robots in a simulated robot collective can be found in chapter 9.

Chapter 8: Robot Differentiation

In addition to collective shape, this thesis presents a method to control individual robot differentiation, in order to form an overall pattern of robot roles in the shape, i.e. the spatial-temporal role pattern. Through differentiation, a robot chooses its role based on: the current scale of the shape, the robot's local clock, communication with neighboring robots, and the given spatial-temporal role map.

8.1 Spatial-Temporal Role Map

The spatial-temporal role map defines how the robots should differentiate based on their location in the desired shape and their current time. This spatial-temporal role map is given to each robot to start with in addition to the shape pixel map. It is given as a pixel map, called the differentiation pixel map, and has the same x and y dimensions as the shape pixel map. Furthermore, each pixel in the differentiation pixel map corresponds to the pixel in the shape pixel map that has the same x and y location. In the case where just a spatial role pattern is required, each pixel in the differentiation pixel map contains an integer value, which represents a role. When a location is outside the desired shape, it has a default role of "-1". An example of a spatial role map is shown in Fig. 8.1(A). In this example, the spatial pattern is for a square shape, where robots in the center take on role "1", and robots on the periphery take on role "2". In the case when a spatial-temporal role pattern is desired, each pixel in the differentiation pixel map has a list of paired numbers. These pairs contain an integer, which represents a role, and a real number, which represents the time to choose that role. An example of a spatial-temporal role map can be seen in Fig. 8.1(B). In this example, at time=0, the robots in the left half of the square should choose role "1" and the robots in the right half of the square should choose role "2". At time=1, the robots in the top half choose role "1" and the robots in the bottom half choose role "2".

-1	-1	-1	-1	-1	-1
-1	2	2	2	2	-1
-1	2	1	1	2	-1
-1	2	1	1	2	-1
-1	2	2	2	2	-1
-1	-1	-1	-1	-1	-1

A

-1	-1	-1	-1	-1	-1
-1	(0,1) (1,1)	(0,1) (1,1)	(0,2)(1,1)	(0,2)(1,1)	-1
-1	(0,1) (1,1)	(0,1) (1,1)	(0,2)(1,1)	(0,2)(1,1)	-1
-1	(0,1)(1,2)	(0,1)(1,2)	(0,2)(1,2)	(0,2)(1,2)	-1
-1	(0,1)(1,2)	(0,1)(1,2)	(0,2)(1,2)	(0,2)(1,2)	-1
-1	-1	-1	-1	-1	-1

B

Figure 8.1. (A) An example spatial role map. (B) An example spatial-temporal role map.

8.2 Spatial Role Pattern

In the case where the collective is just given a spatial role map, i.e. there is no temporal component, then each robot uses a method similar to that in DASH to determine which role it should choose. This is done by the robot first determining where it is located in the differentiation pixel map, which is the same location in which it is located in the shape pixel map. Recall that the robot's location in the shape pixel map is determined by virtually overlaying the shape pixel map in the coordinate system, which was done by assuming the upper left pixel in the shape pixel map, pixel (0,0), corresponds to and is centered on the location (0,0) in the coordinate system. All other pixels (x,y) are then centered to the location ($S_f \cdot x \cdot R_{robot}$, $S_f \cdot y \cdot R_{robot}$) in the coordinate system. For example, if the scale is 2.0, then the pixel (1,1) in the shape pixel map is centered at ($2R_{robot}$, $2R_{robot}$) in the coordinate system, and the pixel (2,3) in the shape pixel map is centered at ($4R_{robot}$, $6R_{robot}$) in the coordinate system. Using this virtual overlay of the shape pixel map in the coordinate system, a robot can use its own coordinates to find which pixel in the shape pixel map is closest to its current coordinates. This closest pixel is considered the robot's current location in the shape pixel map, and therefore its current location in the differentiation pixel map. Once its location is known

in the differentiation pixel map, then the robot can look at the number located at that location to determine its role.

8.3 Temporal Role Pattern

In the case where the collective is given a spatial-temporal role map, i.e. there is both a spatial and temporal component to role selection, each robot will also need to synchronize in time with the other robots in the collective. This synchronization will allow the spatial-temporal role pattern displayed on the collective to also be synchronized with respect to time.

This synchronization of time between robots is accomplished through a mechanism that uses communication between neighboring robots. First, in every main loop of the controller, each robot will receive the value of the local clock of all of its neighbors. Second, it will set its own local clock to be the average of all its neighbors' local clocks. After this update of its local clock, the robot will then communicate its new local clock time to all of its neighbors. When the local clock for each robot is updated in this manner, over a period of time, the local clocks will become synchronized, allowing for a synchronized spatial-temporal role pattern to emerge in the collective.

Examples of a simulated collective forming spatial-temporal role patterns can be found in chapter 9.

Chapter 9: Simulated Validation

This chapter will focus on demonstrating the control methods described in this thesis. This is done by implementing and running them on a simulated robotic collective. First, this chapter will describe the details of the simulation environment used for these demonstrations. Next, it will provide results from this simulation for developing the coordinate system, DASH, S-DASH, and robot differentiation. Finally, this chapter will provide a demonstration that shows all the control methods operating together on the simulated collective.

9.1 Simulation Environment

The simulation is designed to provide a test bed for demonstrating the control methods described in this thesis, without having to build an actual robotic collective. The simulation consists of two parts, one robot controller per robot in the collective, and a single world controller which ensures that robots do not move through each other.

For every robot in the simulation there is one corresponding robot controller. This robot controller is responsible for executing the control methods described in this thesis, namely developing a coordinate system, DASH, S-DASH, and differentiation. For every execution of the robot controller, it will execute a single cycle of the main control loop, as shown in table 4.3. All robot controllers are running separately, but start identical to each other in every way.

The other part of the simulation is the world controller which has three main functions. The first function is to control the locations of all robots. To do this, it has full knowledge of the location and orientation of all robots; however, this information remains hidden from the robot controllers. The world controller models each robot as a circle in the x-y plane that has a radius R_{robot} which is centered on that robot's current location. Whenever a robot controller commands a movement, that command is sent to the world controller. The world controller then determines if that movement will cause the moving robot to move into a neighboring robot. If it does not, then the world

controller will move that robot by the commanded movement. If it does, then the world controller will not move the robot. The robot controller does not receive any feedback whether the movement occurred or not.

The second function of the world controller is to facilitate sensing and communication between neighboring robots. Whenever a robot controller decides to transmit a message to its neighbors, the robot controller will send that message to the world controller. Then the world controller will store that message for all robots that are located closer than R_{com} from the transmitting robot. Whenever a robot controller checks to see if it has received any new messages, the world controller will give that robot controller all messages it has stored for that robot since the last time the robot checked for messages. Furthermore, whenever a message is transmitted, the world controller includes in that message the distance between the transmitting robot and the receiving robot.

The third function of the world controller is to control the execution of the robot controllers. It runs each robot controller once per step of the world controller. The order that the robot controllers run is chosen at random.

In addition to these three functions, the world controller also provides a visual output to the user of the current location of all robots in the environment in the form of an overhead view. This output is simply a picture with circles of radius R_{robot} drawn for every location of a robot in the simulation. The color of each circle is chosen by the individual robot controllers and represents their current chosen role. Fig. 9.1 shows an example visual output for a random group of robots.

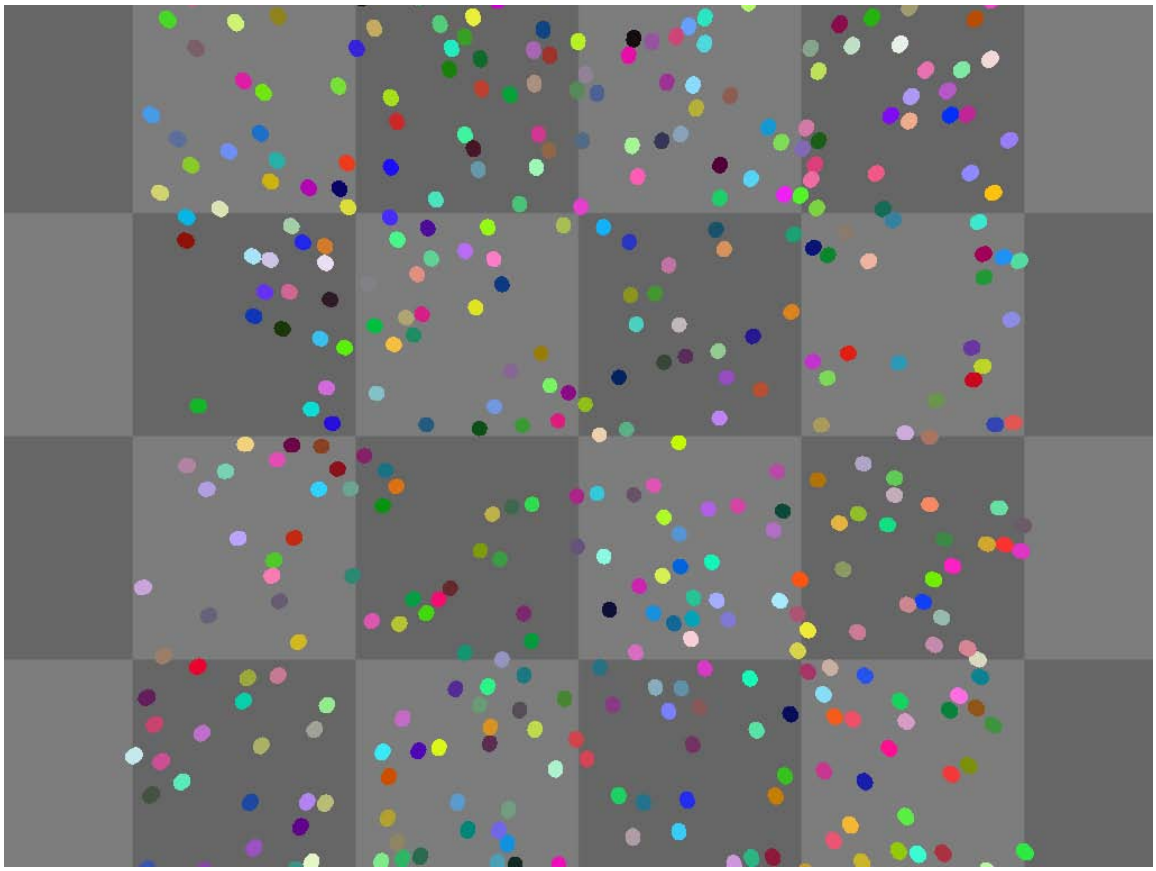


Figure 9.1. An example visual output from simulation.

9.2 Coordinate System

The coordinate system that is developed by the collective is used in all the methods in this thesis, DASH, S-DASH, and robot differentiation; however, this section will show the operation of the coordinate system working independently from those methods. This section will give two demonstrations related to the coordinate system. First, it will demonstrate that the coordinate system assigns locations to each robot that are consistent with the measured distances between robots and the coordinates of those other robots. Second, this section will demonstrate the effects robot movement have on the coordinate system.

To measure the consistency of each robot's location in the coordinate system, an error metric called the consistency error is used. The consistency error looks at all

possible pairs of robots, and sums the difference of the actual distance between the two robots and the distance between their locations in the coordinate system. This consistency error is computed as $\sum_{\text{all robots I,K}}\left(\left|D_{I,K} - \sqrt{(x_I - x_K)^2 + (y_I - y_K)^2}\right|\right)$ where: $I \neq K$, $D_{I,K}$ is the distance between robots I and K, and (x_I, y_I) is the location of robot I in the coordinate system. To test the consistency of the coordinate system developed by the robots, the following demonstration is run in the simulation. The simulation is run 10 times, and for each run, 100 robots are placed at random in a $50 \bullet R_{robot}$ by $50 \bullet R_{robot}$ area. These robots are only tasked to develop a coordinate system; they are not allowed to move. For every step of the world controller, the consistency error is computed. Fig. 9.2 shows the consistency error vs. world controller steps for each simulation run. This demonstrates that the method for producing the coordinate system in the collective is reliable at choosing coordinates for each robot that are consistent with the coordinates of other robots in the collective.

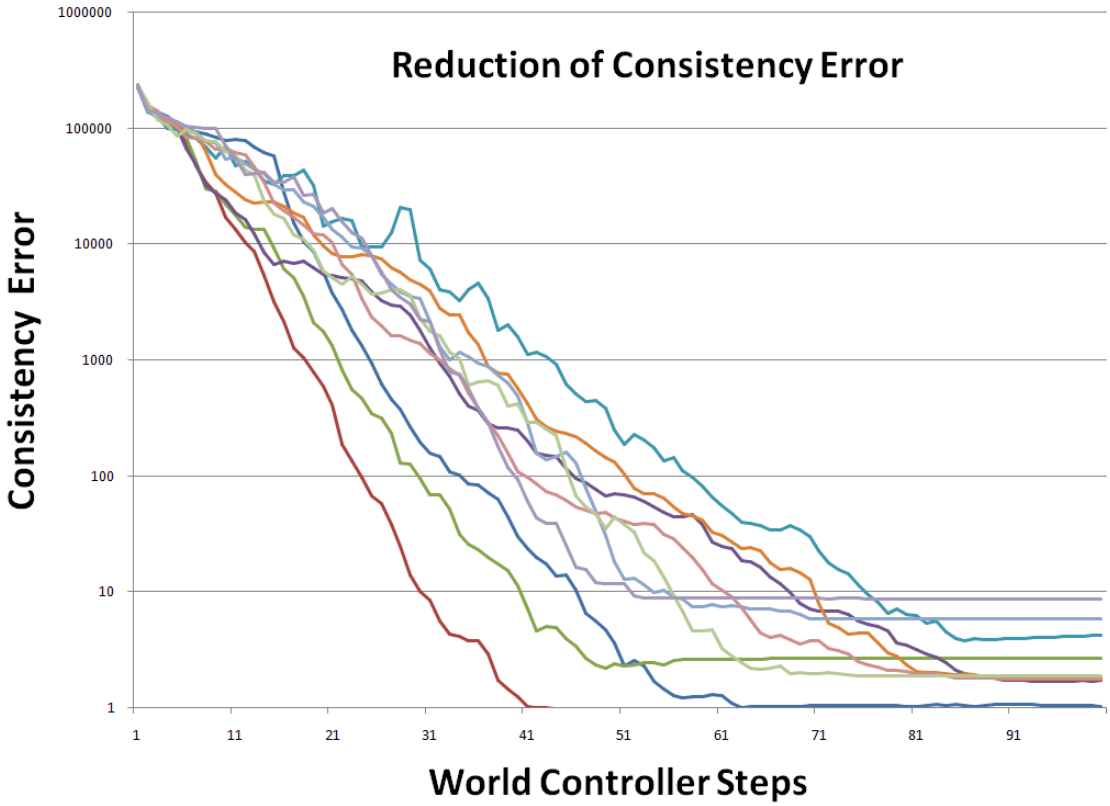


Figure 9.2. The reduction of consistency error over time as a result of the coordinate system update from chapter 5.

In DASH and S-DASH, when a robot moves, it needs to do so without negatively affecting the stability of the coordinate system for neighboring robots. To demonstrate that robot movement does not negatively affect the stability, the following experiment is run. First, 100 robots are placed at random in a $50 \bullet R_{robot}$ by $50 \bullet R_{robot}$ area. Then these robots are given 100 time steps of the world controller to converge on a consistent coordinate system. Between time step 100 and 150, each robot moves with a probability P_{move}. After step 150, the robots are no longer allowed to move. For every time step, the consistency error is recorded. This experiment is repeated ten times, and the consistency error vs. time for each experiment is shown in Fig. 9.3. Note that during the time steps 100-150 during robot movement, the consistency error jumps above a thousand. This is because when a robot moves, it temporarily keeps its old location in the coordinate system until it can re-localize. This causes the consistency error to artificially increase

during robot movement; but because the moving robots do not actually use their location in the coordinate system until they have become re-localized, this does not affect the coordinate system. Notice that immediately after movement stops, the consistency error drops back to or below the levels prior to movement, indicating that the movement did not negatively affect the coordinate system.

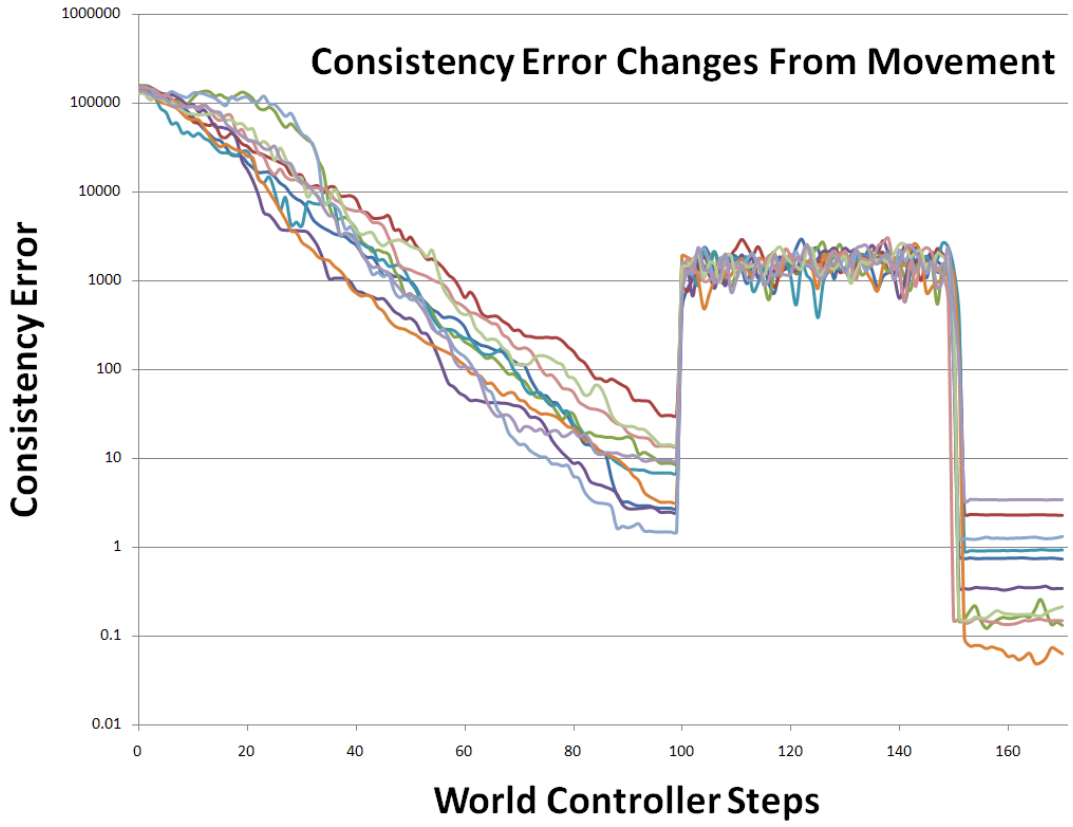


Figure 9.3. The consistency error for 10 simulation runs. During the world controller's steps 100-150, the robots were commanded to move.

9.3 DASH

The main result of DASH is that it allows the collective to both self-assemble and self-heal a given shape at a given scale, both of which will be demonstrated in this section. Since scaling the size of the shape is part of S-DASH, not DASH, the examples

in section 9.3 will be given a fixed scale *a priori* which is appropriate for the number of robots in the collective.

First, self-assembly using DASH will be demonstrated. In Fig. 9.4(A), 400 robots are placed at random in a $60 \bullet R_{robot}$ x $60 \bullet R_{robot}$ area in the simulated environment. They are given the desired shape shown in Fig. 9.5(A). As seen in Fig. 9.4(B-F), using DASH, all the robots in the collective are able move to self-assemble the desired shape. Note that the orientation of the shape in Fig. 9.4 does not match the orientation shown in Fig. 9.5(A). This is because the coordinate system has no knowledge about the orientation of the visual output of the world controller, and therefore is not necessarily in the same orientation. For further demonstration, the results of self-assembly in the collective for the desired shapes in Fig. 9.5(B-E) are shown in Fig. 9.6.

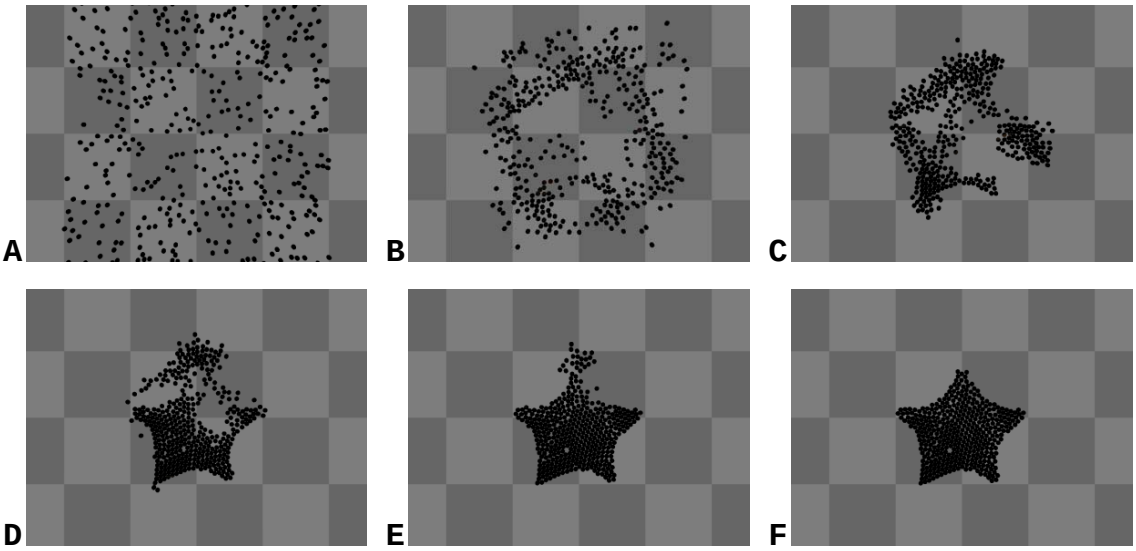


Figure 9.4. A collective forming the desired shape from Fig. 9.5(A). The collective starts from a random placement (A), and using DASH with a fixed scale forms the desired shape (B-F).

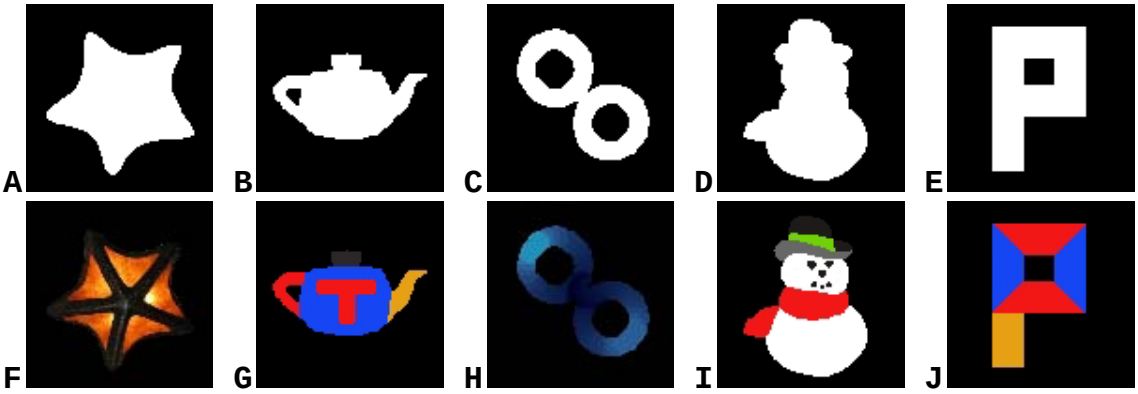


Figure 9.5. Some test shape pixel maps (A-E), and their corresponding differentiation pixel maps (F-J).

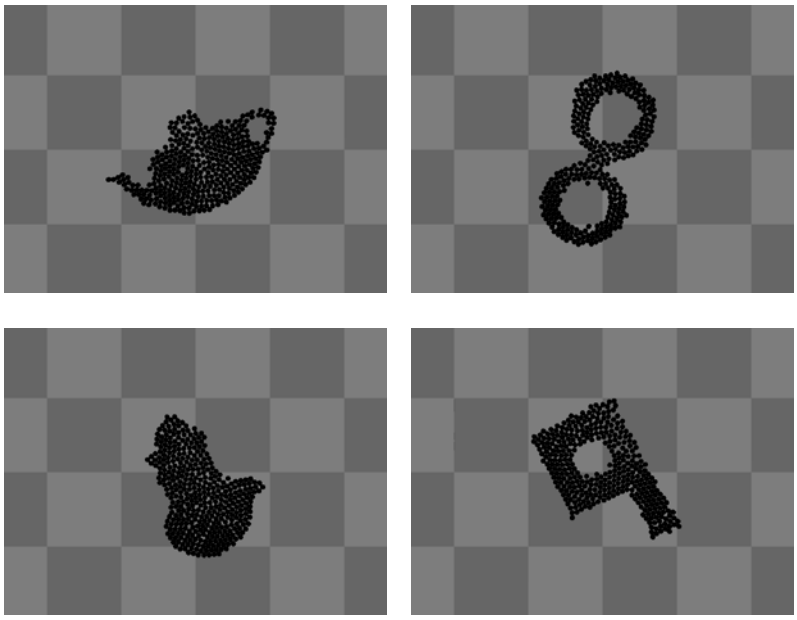


Figure 9.6. Collectives forming the desired shapes from Fig 9.5(B-E).

Next, to demonstrate self-healing, after the shape is self-assembled, some of the robots in the collective are moved to a new location, thus damaging its shape. In Fig. 9.7 the collective of robots from 9.4(F) is damaged by moving some of the robots in the collective, as shown in Fig. 9.7(B). Using DASH, the robots then re-form the shape, as shown in Fig. 9.7(C-F). More examples are shown as the collectives from Fig. 9.6 are

also damaged by moving robots. The damage of these collectives, and the new collective shape after self-repair for these further examples, are shown in Fig. 9.8.

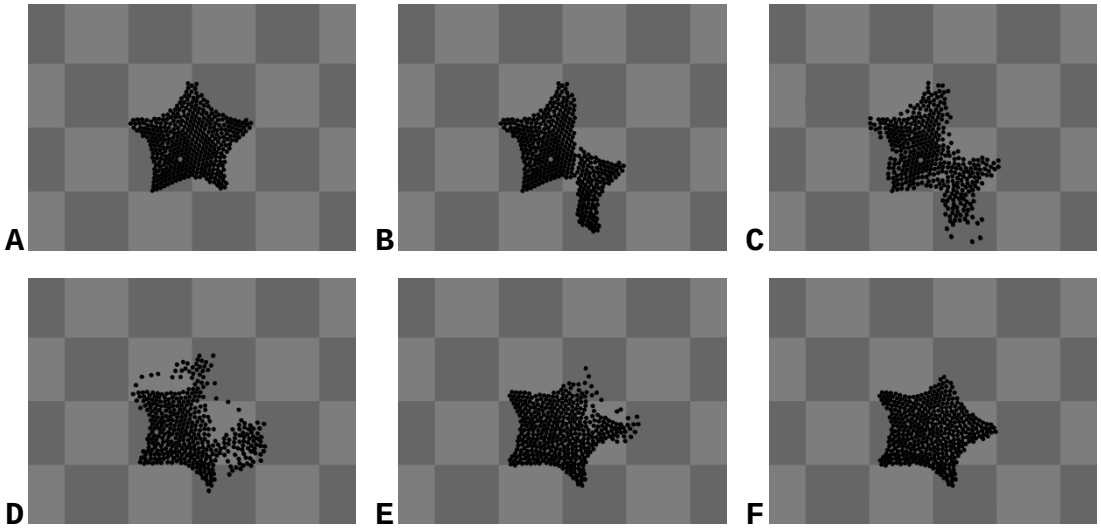


Figure 9.7. After forming the desired shape (A), a collective is damaged by shifting a portion of the robots (B). Using DASH, the collective reforms the desired shape (C-F).

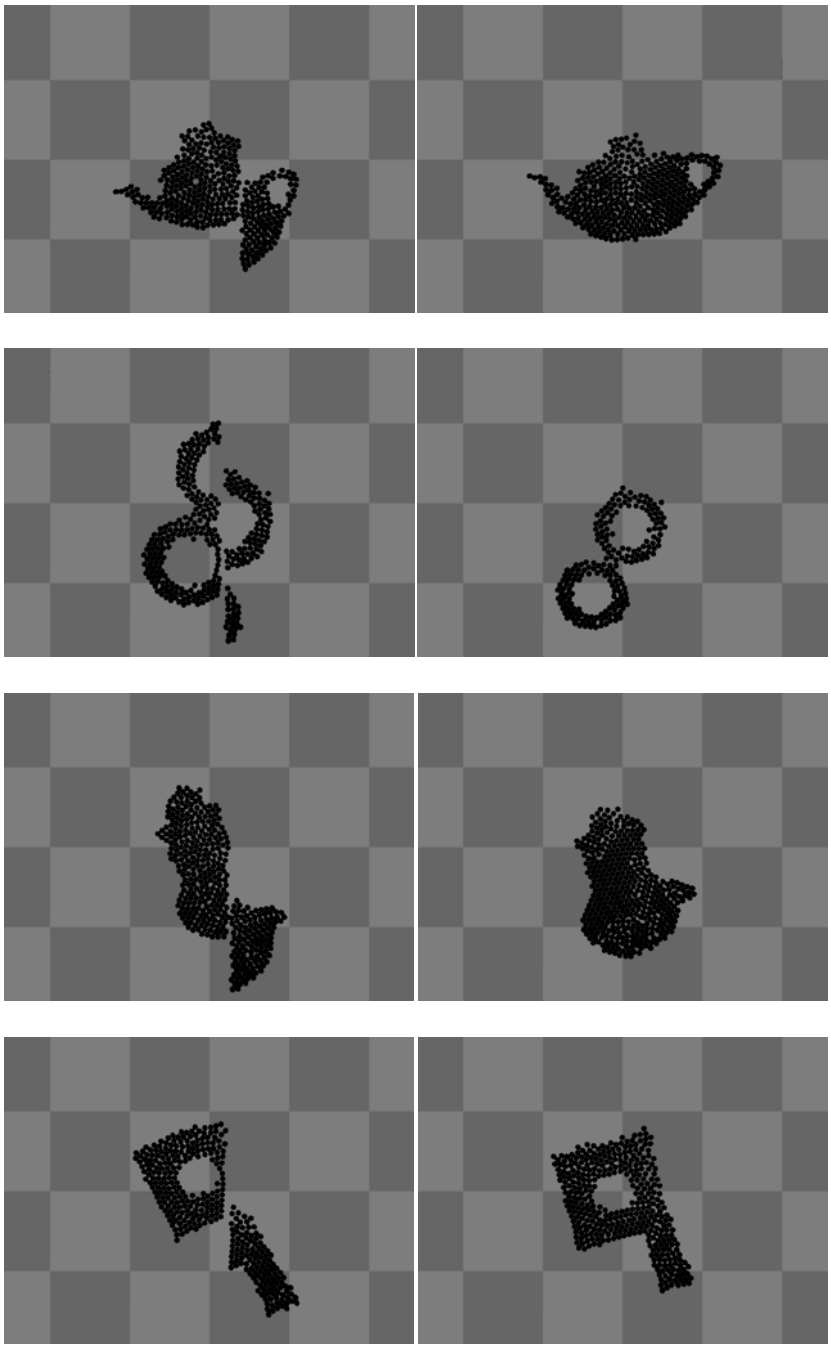


Figure 9.8. Additional demonstrations of shifting damage (left) to the collective from Fig. 9.5(B-E). The collective then self-heals the damage using DASH (right).

9.4 S-DASH

The main result of S-DASH is that it allows the collective to adjust the scale of its shape to fit the number of robots that are currently available. To demonstrate that S-DASH can find the correct scale of the shape, a simulated collective of 100 robots is run four times; each time it is given the same desired shape. However, for each of the four runs, the initial scale, S_f is set to a different value. Fig. 9.9 shows how the scale value for the collective changes over time. It shows that for each of the starting S_f values, S-DASH will converge to the same scale value. In all four of these demonstrations, the value that S_f converges to is the same value. This value allows all robots to be inside the shape.

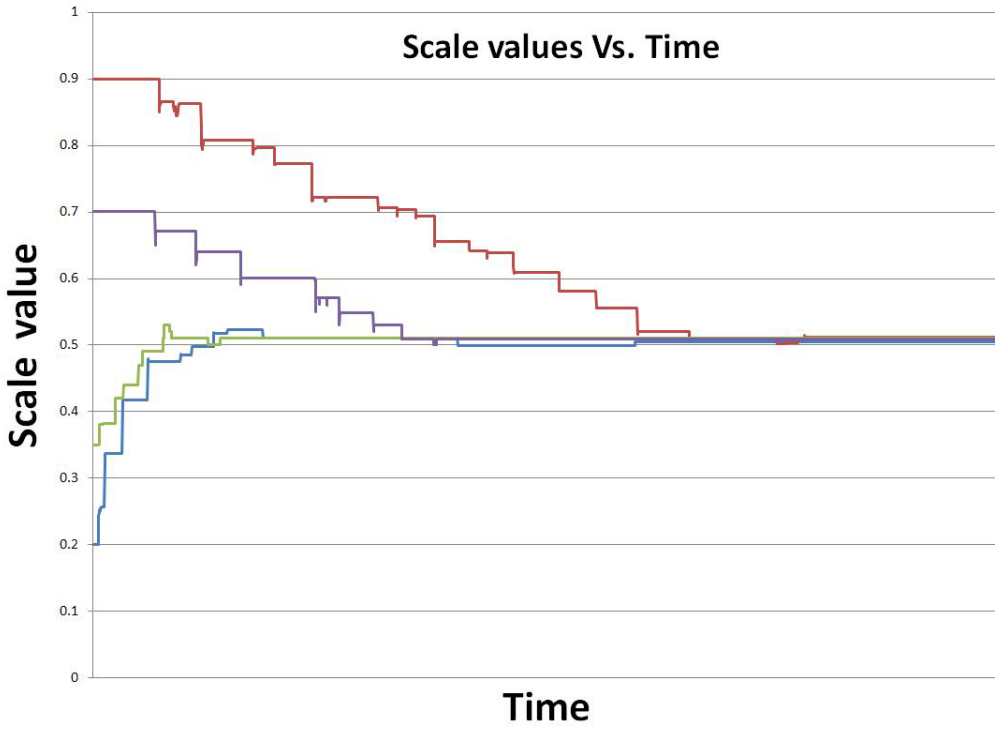


Figure 9.9. S-DASH adjusting the scale value (S_f) of the same collective for four different starting values.

To provide a further example of the collective adjusting is scale using S-DASH, see Fig 9.10. In this example, 400 robots are given the desired shape shown in Fig. 9.5(A); however, their initial S_f value is too small. In Fig. 9.10(A-C), the scale of the shape is increased by S-DASH since there are robots in the external segment seed

location. Finally, S_f reaches a value that allows all robots to be inside the shape, as shown in Fig. 9.10(D).

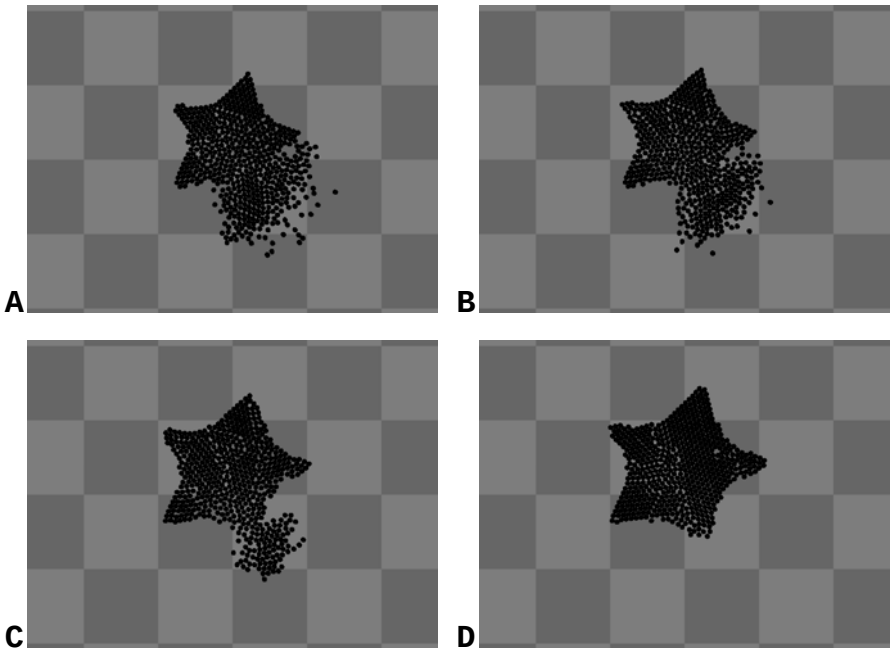


Figure 9.10. A demonstration of S-DASH forming the desired shape from Fig. 9.5(A). Initially the shape is too small for the number of robots (A); however, S-DASH increases the scale (B-D) until the shape fits all the robots in the collective (D).

This demonstration shows S-DASH increasing the scale to fit the number of robots, but does not demonstrate S-DASH decreasing the scale. To show a scale decrease, one way is to remove some robots from the collective. For example, in Fig. 9.11(A), the collective has already formed the desired shape at an appropriate scale; however, in Fig 9.11(B), the top half of the robots in the collective are removed. Fig. 9.11(C-H) shows the shape of the collective reducing in size as S-DASH decreases S_f, until the shape properly fits the remaining robots, as shown in Fig. 9.11(I). Further demonstrations of S-DASH reducing scale due to the removal of robots from the collective are shown in Fig. 9.12. In this figure on the left, after the collective has formed the desired shape, the top half of the robots are removed, and on the right, S-DASH reforms the shape at a smaller scale.

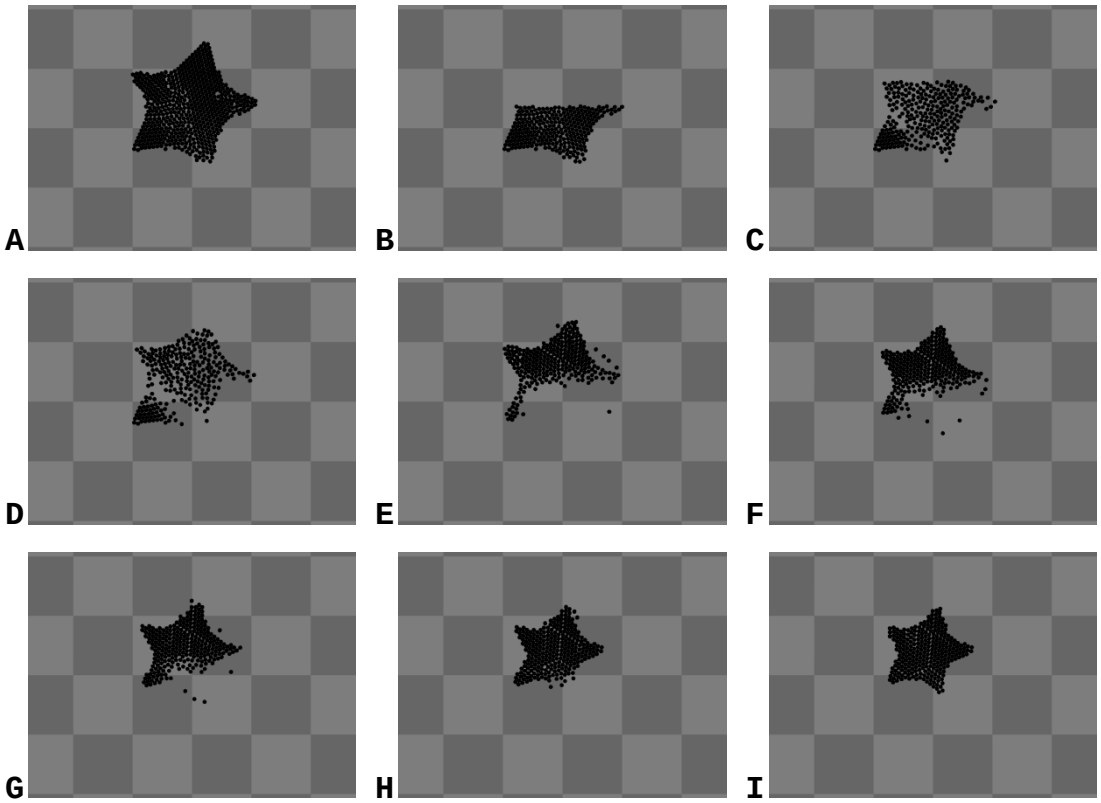


Figure 9.11. After forming the desired shape (A), half the robots are removed (B). The collective then uses S-DASH to reduce the scale of the shape (E-H). Eventually, the scale reduces to a value that is appropriate for the number of robots (I).

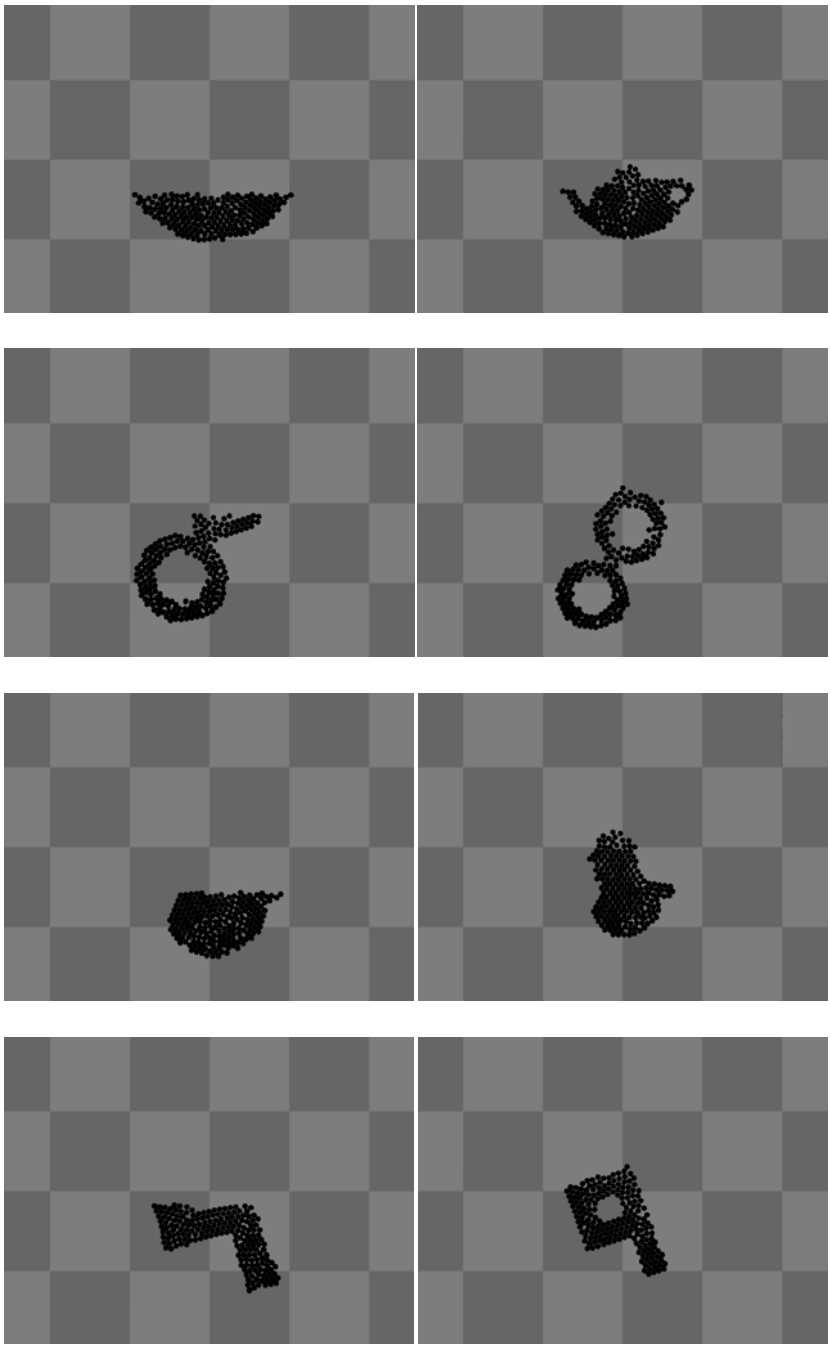


Figure 9.12. Further examples of the removal of the top half of the robots from Fig. 9.5 (B-E) (left). After damage, S-DASH self-heals the shape at a new scale appropriate to the new number of robots (right).

9.5 Robot Differentiation

The purpose of robot differentiation described in this thesis is to form a spatial-temporal role pattern in the collective. Since these patterns are defined based on a robot's location in the desired shape, the patterns should adjust according to the scale of the shape. Fig. 9.13 gives examples of a collective forming the desired shapes and patterns from Fig. 9.5(F-J). This figure shows the pattern forming on two different scales for each of the desired shapes, and the spatial role pattern scales to fit the size of the shape. The method for differentiation can also form spatial-temporal role patterns. For example, the collective is given the spatial-temporal role map in Fig. 9.14(A). This role map has six different patterns that it displays based on the robot's current time. If its time is less than ten, then the robot uses the left most pattern to choose its role; if time is less than twenty but greater than ten, it uses the second left most pattern to choose its role, etc. This will cause the collective to display the letters "T"-"E"-"A"-"P"-"O"-"T", switching letters every 10 time steps. Fig. 9.14(B) shows the collective in increments of ten steps of the world controller. This demonstrates the collective forming a spatial-temporal roll pattern, specifically the one in Fig. 9.14(A).

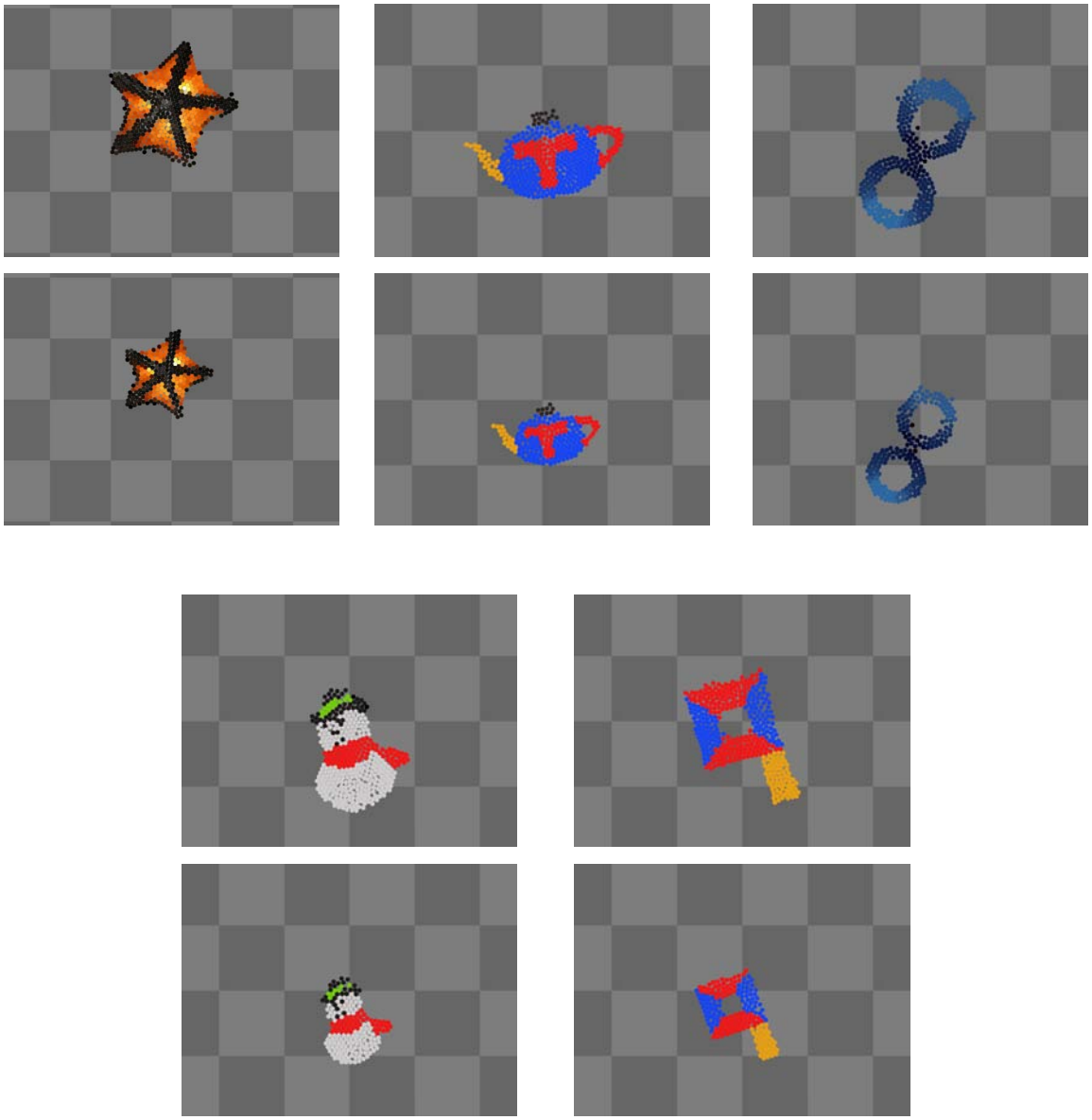
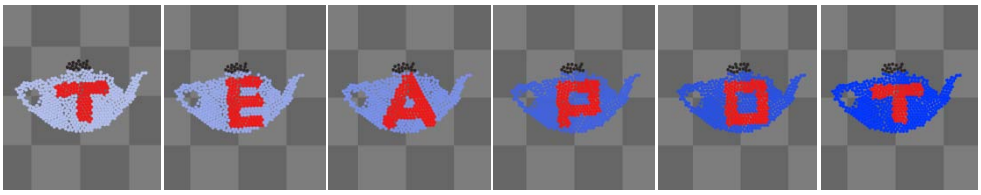


Figure 9.13. The simulated collective displaying a spatial role pattern for the given differentiation pixel maps from Fig. 9.5(F-J). Each pattern is shown on two different scales of the desired shape.

A

0-10 10-20 20-30 30-40 40-50 50-60

B



0 10 20 30 40 50

Figure 9.14. The given spatial-temporal role map for a collective (A). Each portion of this role map represents a desired role pattern for the time given below it. A series of pictures of a collective forming the desired role pattern from (A) can be seen in (B). The numbers below show the number of world controller time steps since the leftmost image was captured.

9.6 Overall Demonstration

Finally, an overall demonstration is given that shows DASH, S-DASH, and robot differentiation all working together. In Fig 9.15(A), the collective starts from random positions in the environment. It is given the shape pixel map from Fig. 9.5(A) and differentiation pixel map from Fig. 9.5(F). The collective then forms the desired shape and pattern to an appropriate scale, shown in Fig. 9.15(B-E). Then as shown in Fig. 9.15(F), the right most 30% of the robots are shifted down a distance of $20 \bullet R_{robot}$, and the top most 30% are removed; out of the remaining robots, another 20% are also removed at random. After this damage, the collective reforms the shape and pattern at a new scale appropriate to the current number of robots Fig. 9.15(G-L).

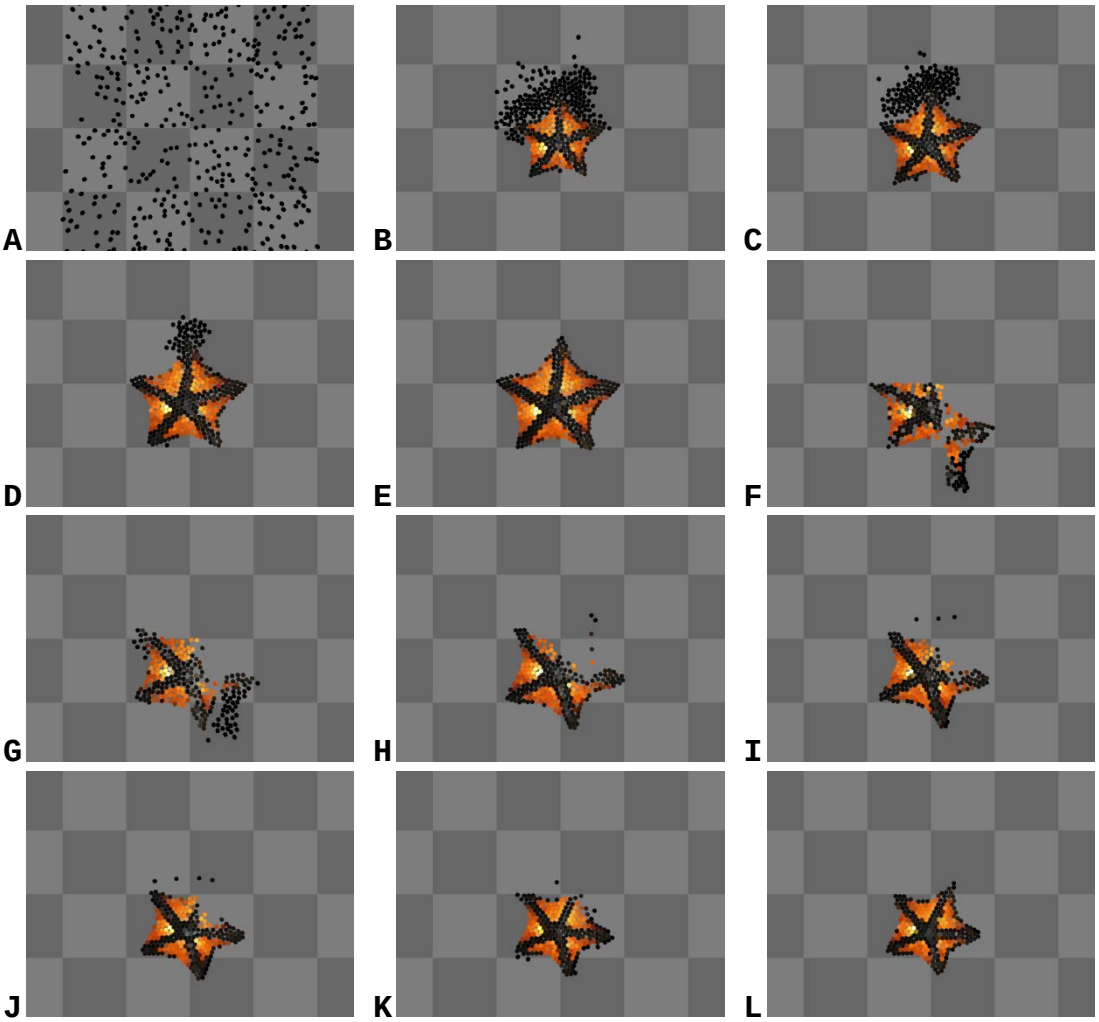


Figure 9.15. A collective starts from random positions (A). The collective forms the desired shape from Fig. 9.5(A) and pattern from Fig. 9.5(F) at an appropriate scale (B-E). In (F), the right most 30% of the robots are shifted down a distance of $20 \bullet R_{robot}$, the top most 30% are removed, and a further 20% are removed at random. The collective then reforms the shape and pattern at a scale appropriate to the new number of robots (G-L).

Chapter 10: Conclusion and Future Work

This thesis presented four contributions to the control of a robotic collective. They are DASH, S-DASH, the development of a coordinate system, and robotic differentiation. Together, these four contributions can allow a collective of distributed robots to self-assemble and self-heal a desired shape and spatial-temporal role pattern with an adaptive scale. This can allow the robots, as a collective, to complete a goal that cannot be accomplished by any of the individual robots alone.

The first contribution, DASH, is a method for a randomly distributed collective of robots to self-assemble into any given connected shape. These robots use information gathered solely from communication with neighboring robots, control of their local actuation, knowledge of the desired shape, and their location in a consistent coordinate system, to perform this self-assembly. The robots are identical in every way, contain no knowledge of their starting position, and do not require any other form of environmental awareness. Furthermore, if the collective shape is damaged, DASH will self-heal the collective back to the desired shape.

The second contribution, S-DASH, allows the shape formed by DASH, as well as the spatial-temporal role pattern, to be scaled according the number of robots in the collective. S-DASH uses information gathered solely from communication with neighboring robots, a robot's location in the consistent coordinate system, and knowledge of the desired shape, to perform this scalable self-assembly. In the event that the number of robots in the collective changes, S-DASH will adjust the scale of the shape to reflect the new number of robots.

The third contribution is a distributed method for developing a consistent coordinate system for a collective of robots, largely based on trilateration and robust quadrilaterals extended from (Moore, et al. 2004). This localization method is critical for DASH, S-DASH, and robot differentiation. This method requires no centralized control, initialization, global sensing, or global IDs. These relaxed constraints improve both the robustness of the swarm to damage and ease possible robot design requirements.

The fourth contribution is a method for individual robots within a collective to differentiate into various roles, depending on their location within a shape, as well as their local time. To do so, these robots use information gathered solely from communication with neighboring robots, knowledge of the spatial-temporal role map, and their location in the coordinate system, to choose this role. Furthermore, if robots are added or removed from the collective causing the scale to change, then the spatial-temporal role pattern will also scale accordingly.

There are a few possible research directions to continue and expand the work in this thesis. One possible extension to this work is to implement it on a real robotic collective. This could mean implementation on a currently existing collective, or the design of a robotic collective to meet the requirements set out in this thesis. Another possible direction to continue this research is to increase the constraints on robot movement. Currently the only constraint is that robots must avoid moving through other robots, similar to those of mobile robots. A further constraint could be that robots can only move into specific lattice locations, while staying connected to neighboring robots. These types of constraints can be found in reconfigurable robotic systems, and could allow this work to be applied to such a system.

Works Cited

- Arbuckle, Daniel, and Aristides Requicha. "Active Self-Assembly." *International Conference on Robotics and Automation.* New Orleans, LA, 2004.
- Bachrach, Jonathan, and Christopher Taylor. "Localization in Sensor Networks." In *Handbook of Sensor Networks*, by Ivan Stojmenovic. Wiley , 2005.
- Bishop, J., et al. "Programmable Parts: A Demonstration of the Grammatical Approach to Self-Organization." *Intelligent Robots and Systems.* 2005.
- Bode, Hans. "Head Regeneration in Hydra." *Developmental Dynamics*, 2003: 226:225-36.
- Cheng, Jimming, Winston Cheng, and Radhika Nagpal. "Robust and Self-Reparing Formation Control for Swarms of Mobile Agents." *National Conference on Artificial Intelligence (AAAI '05).* 2005.
- Chiu, Harris, Michael Rubenstein, and Wei-Min Shen. "Deformable Wheel'-A Self-Recovering Modular Rolling Track." *Intl. Symposium on Distributed Robotic Systems.* Tsukuba, Japan, 2008.
- Eno, Steven, et al. "Robotic Self-Replication in a Structured Environment without Computer Control." *IEEE International Symposium on Computational Intelligence in Robotics and Automation.* Jacksonville, FL, 2007.
- Eren, T., et al. "Rigidity, Computation, and Randomization in Network Localization." *INFOCOMM.* 2004.
- Gilpin, Kyle, Keith Kotay, Rus Daniela, and Iuliu Vasilescu. "Miche: Modular Shape Formatio by Self-Disassembly." *The International Journal of Robotics Research*, 2008: 345-372.
- Goldenberg, David, et al. "Localization in Sparse Networks Using Sweeps." *MobiCom.* Los Angeles, CA, 2006.
- Grabowski, Robert, and Pradeep Khosla. "Localization Techniques for a Team of Small Robots." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* 2001.

- Hofmann-Wellenhof, B., H. Lichtenegger, and J. Collins. *Global Positioninging System: Theory and Practice.* Springer Verlag, 1997.
- Hou, S. P., C. C. Cheah, and J. J. Slotine. "Dynamic Region Following Formation Control for a Swarm of Robots." *2009 IEEE International Conference on Robotics and Automation.* Kobe, Japan, 2009.
- Howard, Andrew, Maja Mataric, and Gaurav Sukhatme. "Relaxation on a Mesh: a Formalism for Generalized Localization." *IROS* . Wailea, Hawaii, 2001.
- Ishiguro, Akio, Masahiro Shimizu, and Toshihiro Kawakatsu. "Don't Try to Control Everything!: An Emergent Morphology Control of a Modular Robot." *Intelligent Robots and Systems.* 2004.
- Jiang, Ting-Xin, Han-Sung Jung, Randall B. Widelitz, and Cheng-Ming Chuong. "Self-Organization of Periodic Patterns by Dissociated Feather Mesenchymal Cells and the Regulation of Size, Number and Spacing of Primordia." *Development*, 1999: 4997-5009.
- Kellogg, Vernon L. "Restorative Regeneration in Nature of the Starfish Linckia Diplax (Müller and Troschel)." *Journal of Experimental Zoology*, 1904: 353-356.
- Kondacs, Attila. "Biologically-Inspired Self-Assembly of 2D Shapes, Using Global-to-local Compilation." *International Joint Conference on Artificial Intelligence (IJCAI).* 2003.
- Kurazume, Ryo, and Shigeo Hirose. "An Experimental Study of a Cooperative Positioning System." *Autonomous Robots*, 2000: 43-52.
- Li, Mo, and Yunhao Liu. "Renderd Path: Range-Free Localization in Anisotropic Sensor Networks with Holes." *MobiCom.* Montreal, Canada, 2007.
- Mao, Guoqiang, Baris Fidan, and Brian Anderson. "Wireless Sensor Network Localization Techniques." *Computer Networks 51*, 2007: 2529-2553.
- Martinez, Daniel E. "Mortality Patterns Suggest Lack of Senescence in Hydra." *Experimental Gerontology*, 1998: 217-225.
- Maxim, Paul, et al. "Trilateration Localization for Multi-Robot Teams." *International Conference on Informatics in Control, Automation and Robotics,*

Special Session on Multi-Agent Robotic Systems. 2008.

- Moore, David, John Leonard, Daniela Rus, and Seth Teller. "Robust Distributed Network Localization with Nosiy Range Measurements." *SenSys.* Baltimore, 2004.
- Nagpal, Radhika. "Programmable Self-Assembly: Constructing Global Shape Using Biologically-Inspired Local Interactions and Origami Mathematics." *Thesis.* MIT, 2001.
- Nagpal, Radhika, Howard Shrobe, and Jonathan Bachrach. "Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network." *ispn.* 2003.
- Niculescu, Dragos, and Badri Nath. "DV Based Positioning in Ad Hoc Networks." *Telecommunication Systems*, 2003: 267-280.
- Rosa, Michael De, Jason Campbell, Seth Goldstein, Padmanabhan Pillai, and Peter Lee. "Scalable Shape Sculpting via Hole Motion: Motion Planning in Lattice-Constrained Modular Robots." *IEEE International Confrence on Robotics and Automation (ICRA).* Orlando, Florida, 2006.
- Rothemund, Paul W. K. "Folding DNA to Create Nanoscale Shapes and Patterns." *Nature*, 2006: 297-302.
- Rubenstein, Michael, and Wei-Min Shen. "A Scalable and Distributed Approach for Self-Assembly and Self-Healing of a Differentiated Shape." *International Conference on Intelligent Robots and Systems.* Nice, France, 2008.
- Sahin, Erol, et al. "SWARM-BOT: Pattern Formation in a Swarm of Self Aseembling Mobile Robots." *International Conference on Systems, Man and Cybernetics.* Hammamet, Tunisia, 2002. 145-150.
- Savarese, Chris, Jan Rabaey, and Jan Beutel. "Location in Distributed Ad-Hoc Wireless Sensor Networks." *Acoustics, Speech, and Signal Processing (ICASSP).* Salt Lake City, Utah, 2001.
- Shen, Wei-Min, Behnam Salemi, and Peter Will. "Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable

Robots." *IEEE Trans. on Robotics and Automation*, 2002.

- Shen, Wei-Min, Peter Will, Aram Galstyan, and Cheng-Ming Chuong. "Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms." *Autonomous Robots 17*, 2004: 93-105.
- Shoemake, Ken. "Animating Rotation with Quaternion Curves." *International Conference on Computer Graphics and Interactive Techniques.* San Francisco, Ca, 1985.
- Spears, William M, and Diana F Gordon. "Using Artifical Physics to Control Agents." *Information Intelligence and systems.* Bethesda, MD, 1999. 281-288.
- Stoy, Kasper, and Radhika Nagpal. "Self-Repair Through Scale Independent Self-Reconfiguration." *International Conference on Intelligent Robots and Systems.* Sendai, Japan, 2004.
- Suzuki, Yosuke, Norio Inou, Hitoshi Kimura, and Michihiko Koseki. "Reconfigurable Group Robots Adaptively Transforming a Mechanical Structure." *International Conference on Intelligent Robots and Systems.* San Diego, CA, 2007.
- Yang, Zheng, and Yunhao Liu. "A Survey on Localization in Wireless Sensor Networks." 2007.
- Yang, Zheng, and Yunhao Liu"Quality of Trilateration: Confidence Nased Iterative Localization." *ICDCS.* 2008.
- Yim, Mark, Babak Shirmohammadi, Jimmy Sastra, Mike Park, Mike Dugan, and C. J. Taylor. "Towards Robotic Self-reassembly After Explosion." *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.
- Yim, Mark, et al. "Modular Self-Reconfigurable Robot Systems: Challenges and Opportunities for the Future." *IEEE Robotics and Automation Magazine*, 2007: 43-52.
- Yim, Mark, John Lamping, Eric Mao, and Geoffrey Chase. "Rhombic Dodecahedron Shape for Self-Assembling Robots." *Xerox PARC technical Report R9710777.* 1997.

- Zykov, Victor, and Hod Lipson. "Experiment Design for Stochastic Three-Dimensional Reconfiguration of Modular Robots ." *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2007.