Automatic Scalable Size Selection for the Shape of a Distributed Robotic Collective

Michael Rubenstein, Wei-Min Shen

Abstract— **A collective of robots can together complete a task that is beyond the capabilities of any of its individual robots. One property of a robotic collective that allows it to complete such a task is the shape of the collective. One method to form that shape is to form it at a size proportional to the number of robots in that collective, i.e. scalably.**

In our previous work, scalably forming the shape of the collective required that each robot know the total number of robots in the collective. In this work we present a method called S-DASH, which now allows a collective to scalably form a shape without knowing how many robots are in the collective. Furthermore, S-DASH will change the size of the shape to reflect the addition or removal of robots from the collective. This paper also provides demonstrations of S-DASH running on a simulated collective of robots.

I. Introduction

In this paper we present a distributed solution to the problem of choosing the shape size for a collective of distributed robots (sometimes referred to as a swarm, group, or ensemble). It is possible for this robotic collective to complete a goal that cannot be completed by any of the individual robots, if the collective forms a specific shape. For example, imagine a single S-BOT [1] reaches a canyon-like obstacle with a goal on the other side. By itself, a single S-BOT is not capable of crossing the canyon to reach the goal on the other side. However, if the S-BOT joins a collective of other S-BOTs, and forms a collective shaped like a bridge, the collective's shape enables it to cross the canyon and reach the goal. In another example, a single Superbot robot [2] needs to locomode as far as it can until its battery pack empties. As a solitary Superbot, 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, 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 five times as far as any single Superbot can travel.

When the collective forms a shape, one challenge is to determine the size of the collective's 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 size. Those options are fixed size, or scalable.

Manuscript received March 10, 2010. Michael Rubenstein and Wei-Min Shen are with the Information Sciences Institute and Computer Science Department at the University of Southern California, Marina del Rey, CA 90292, USA. (website: www.isi.edu/robots phone: 310-448-8710; fax: 310-822-0751; e-mail: mrubenst@usc.edu , shen@isi.edu).

The first option for determining the size of the collective shape, fixed size, used in [3], is to keep the size of the shape constant, but either change the density of robots as in [4], or grow new robots until the fixed scale is reached, like [5]. Current robotic technology can only self-replicate in very controlled environments, for example [6], so growing new robots is generally not feasible. In biological systems however, growing replacements is possible, and is seen in many cases of animal self-healing.

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 [7], 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 the same, irrespective of the size of the collective.

The second option for determining the size of the collective shape is to adapt the size to the number of robots in the collective, known as scalable size selection. 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 [8], where a small invertebrate, the hydra, will reform its original shape after being cut in half, but at half the size. The hydra does this using morphallaxis, where the remaining cells move to repair the damaged shape.

Without requiring the production of more robots, this scalable size selection lends itself nicely to robotic collectives. For example scalable size selection has been used in [9-11], however only for a limited class of shapes.

A difficulty with scalable size selection is that because the size 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.

The task of choosing the size of the collectives shape is further complicated once properties of a robotic collective are considered. For example, to keep costs down, the individual robots generally have limited sensing range, so

directly counting the number of robots is difficult or not possible. Furthermore, many collectives are homogenous, even lacking a unique ID for individual robots. This lack of ID can hinder multi-hop communication between distant robots, making coordination between robots difficult.

In the work presented in this paper, we use a fully distributed method to scalably select the size of the collective shape, called S-DASH (scalable, distributed self-assembly and self-healing) for a large class of possible shapes. S-DASH builds upon our previous work called DASH (distributed self-assembly and self-healing), which is summarized in section 2 and can be found in [12]. Section 2 will also summarize the assumed properties and capabilities of the robots in this paper. The method of S-DASH is then presented in section 3. Next, in section 4, and examples S-DASH in use is demonstrated in a simulated collective.

II. DASH

The following section will first discuss the assumed properties and capabilities of the robots in this paper. Next it will summarize DASH, a distributed method to control each robot in a robotic collective, so that they form a desired shape at a given fixed scale.

A. Assumptions

The robots are simple and homogeneous. Each robot is shaped like a simple 2D circle, with radius R_{robot}. They exist in a 2D planar environment. The robot is capable of moving in its local x direction, at velocity V_{robot}, as well as rotating about its center, perpendicular to the plane. A robot cannot share the same space as another robot, and is not capable of pushing any robots. All robots in the collective are identical and indistinguishable from each other in every way, even lacking a unique ID.

Communication and ranging between neighboring robots is possible. Each robot can communicate to any of its neighbors who are within a certain distance (R_{com}). During this communication, the range between the two robots can also be measured

Robots have a consistent coordinate system. The collective has a shared coordinate system that is known by all robots. This enables each robot to precisely know its location in the coordinate system, in terms of (X,Y). This coordinate system is developed from a local, distributed method based on robust quadrilaterals [13]. The details of how this coordinate system is produced are omitted from this paper for the sake of brevity. When the robot moves, it can also determine the angle between its local x direction and the x direction of the shared coordinate system, as described in [12].

B. DASH

The following is a summary of DASH, which guarantees that a collective of robots can form any given simply connected shape at a fixed scale. For a more detailed description see [12]. The DASH control method uses an identical controller that runs in each robot. Each robot controller is given a full description of the desired collective shape, in the form of a picture, or pixel map. When DASH is run on each robot in the collective, the robots will self-assemble to form the desired shape at a fixed size, and self-repair if the shape is damaged. Initially there is no information about how large to form the shape, so an initial guess of shape scale, S_{f_start} is used.

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. Using those inputs, DASH generates a commanded movement for the robot, so that each robot will move to a location that is inside the desired shape at the given scale. At the same time, this movement will not prevent other robots from entering the desired shape.

The movement of robots using DASH also provides two locations in the desired shape that are important for S-DASH. The first location is called the shape seed. This shape seed location will be the last location inside the shape to be occupied with robots. This means that if there is any other location in the shape that is not occupied by robots, then that unoccupied location will become filled by robots before the shape seed is filled with robots. The second important location for S-DASH is the location, called the external seed. This is the first location outside the shape that will be filled by robots if there is no room for any more robots inside the desired shape. For example, if there is no empty space in the desired shape, and there is one robot located outside the desired shape, that robot will move to the external seed location.

III. S-DASH

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 fixed scale value used in section 2. 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 of S-DASH 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.

A. Scale

Each robot has a variable, S_f, which represents that robot's belief as to the appropriate scale of the shape. Specifically, each pixel in the given picture of the desired shape will have dimensions of $S_f \bullet R_{robot}$ when formed by the collective. 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 fit the number of robots. To accomplish this consistency and the 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. 1 is applied every cycle of the robot's main controller loop, unless the conditions described in sections III.B or III.C apply.

```
Total_scale = Sf
Number_neighbors = 0
For( all neighbors i)
{
  Total_scale = Total_scale+neighbor_i's_Sf
  Number_neighbors = Number_neighbors + 1
}
Sf = Total_scale / (Number_neighbors + 1)
```

Figure 1. Pseudo code for scale update.

B. Scale Reduction

For the reduction of the scale, S-DASH uses the fact that when using DASH to form a shape, the shape seed is the last location to be filled in the desired 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 given in Fig. 2(A). In this shape, the shape seed 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 2(B), then there will not be any robots near the shape seed.

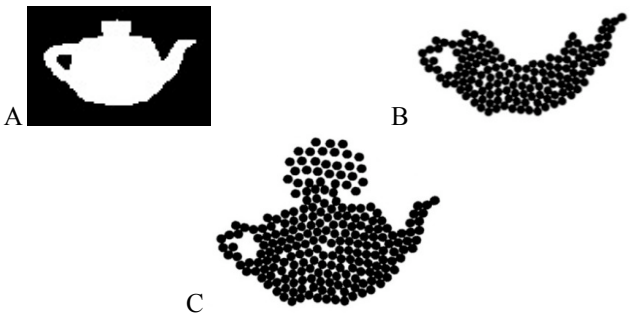


Figure 2. (A) The desired shape, a teapot. (B) The collective forming the desired shape at too large a scale. (C) The collective forming the desired shape at too small a scale.

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

1) Detecting Un-Occupied Shape Seed

The distributed mechanism for detecting an un-occupied shape seed is based on a variable called $T_{un\text{-}occupied}$. This variable is updated in the following way. Every loop of each robot's 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's body overlaps into the shape seed location, then it sets its $T_{un\text{-}occupied}$ value to be zero. A robot uses its location in the coordinate system to determine if this overlap occurs. Fig. 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.

Updating $T_{un\text{-}occupied}$ in this manner will give the following effects. If any robot is in the shape seed location, 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 shape seed location. If no robots are in the shape seed location, then every robot in the collective will start increasing its value of $T_{un\text{-}occupied}$ by one, for every loop of its main controller. Eventually 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 the next section.

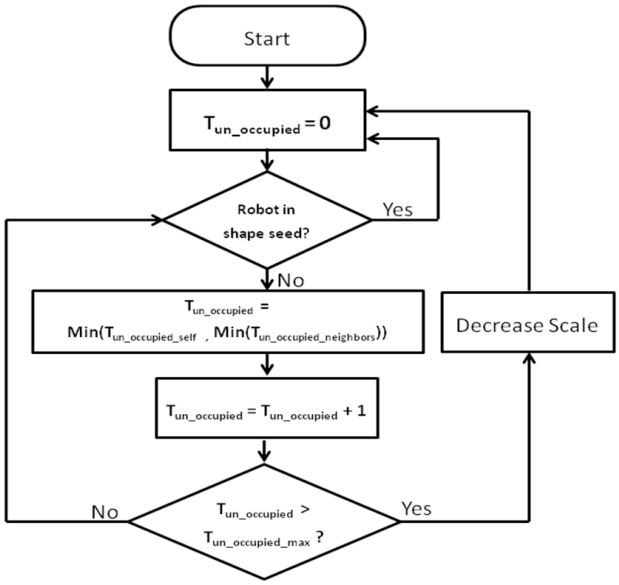
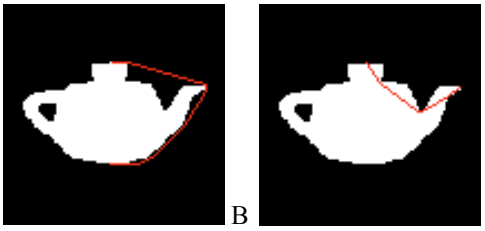


Figure 3. Flow chart for updating $T_{un\text{-}occupied}$.

2) Wait Time

Before reducing the scale of the shape because the shape seed does not contain any robots, S-DASH must first make sure that all the robots in the collective have had a chance

to move into the shape. If any robots are outside the shape and moving to get into the shape, then the method in section III.B.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 $T_{\text{un-occupied-max}}$ variable. The value of $T_{\text{un-occupied-max}}$ is based on three 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 maximum external path length, or $L_{\text{external\_path}}$, which is defined as follows. For all locations that are on the external boundary of the shape, find the shortest path between that location and the location of the shape seed, which stays outside of the shape, and set $L_{\text{external\_path}}$ to be the largest of these shortest paths. An example of the longest, shortest path for the shape in Fig. 2(A) is shown as the red line in Fig 4(A). With these three factors known, $T_{\text{un-occupied-max}}$ is computed to be $T_{\text{un-occupied-max}} = (S_f \cdot L_{\text{external\_path}}) / (V_{\text{robot}})$. This is the worst case distance that a robot must travel to get into the empty space in the shape seed, $(S_f \cdot L_{\text{external\_path}})$ divided by the average speed at which the robot travels, V_{robot}.



A B

Figure 4. (A) Visualization of $L_{\text{external\_path}}$. (B) Visualization of $L_{\text{internal\_path}}$.

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 $T_{\text{un-occupied}} \geq T_{\text{un-occupied-max}}$. The robot or robots that initiate the scale reduction do two things. The first thing they do is set their $T_{\text{un-occupied}}$ value to zero. This has the effect of inhibiting other robots from further trying to reduce the scale by resetting the $T_{\text{un-occupied}}$ values in the collective. The second thing that the robot or robots initiating scale reduction do is that for $T_{\text{un-occupied-max}} / 2$ cycles of the main controller loop, they don't use the scale update from section III.A; instead they report to their neighbors that their scale is a new, lower scale, $S_{\text{f\_new}}$, 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 $T_{\text{un-occupied-max}} / 2$ cycles, the scale update returns to the method from section III.A.

When the scale is being reduced, it is known that at least the shape seed location is un-occupied by robots. From [12], it is known that the area this location takes up in the environment is equal to $(S_{\text{f\_old}} \cdot R_{robot})^2$, where $S_{\text{f\_old}}$ is the scale of the shape prior to the start of reduction, and R_{robot} is the radius of a robot. Because this location 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 $(S_{\text{f\_old}} \cdot R_{robot})^2$, the area of shape seed. The area of the shape at the old scale is just NUM_PIX $\cdot (S_{\text{f\_old}} \cdot R_{robot})^2$, and the area of the shape at the new, smaller scale is just NUM_PIX $\cdot (S_{\text{f\_new}} \cdot R_{robot})^2$. Therefore, the new scale, $S_{\text{f\_new}}$, should be $S_{\text{f\_old}} \cdot \sqrt{1 - \frac{1}{\text{NUM\_PIX}}}$, where NUM_PIX is the number of pixels of the desired shape, in the given picture describing the shape.

This process of reducing the scale when the shape seed is unoccupied will continue until the shape seed is permanently occupied.

C. Scale Increase

To increase the scale of the shape, S-DASH uses the fact that when using DASH to form a shape, the external seed is the first location to be filled by robots if the shape is completely filled by other robots. This means that if the shape is too small, then the external seed will continuously be occupied by robots. For example, consider the desired shape map given in Fig. 2(A). In this shape, the external seed is located just above the very top of the desired shape. When the scale is too small for the number of robots, as shown in Fig 2(C), then as a result of DASH there will continuously be robots in the external seed location, trying to get into the shape.

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

1) Detecting Occupied External Seed

The distributed mechanism for detecting a continuously occupied external 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 its location in the coordinate system to see if it is located in the external seed. 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 $T_{\text{occupied-max}}$, 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" to all its neighbors.

The "moved_into_shape_message" is initiated by one of two events. The first event is when a robot moves from outside the shape to inside the shape. 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 robots outside the desired shape. 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.

2) Wait Time

Before increasing the scale of the shape because the external seed contains robots, S-DASH must first make sure that enough time is given for DASH to fill any empty locations in the desired shape. This is because if there are any empty spaces in the shape, then the method in section III.C.1 might pre-maturely decide the scale of shape is too small, where in fact if robots where given time to move into those empty spaces, it would be at the right scale. S-DASH makes sure there are no empty spaces in the shape 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 three factors, and represents the upper bound of the time DASH will take to fill empty locations in the desired shape 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 maximum internal path length, or $L_{internal\_path,}$ which is defined as follows. For all locations on the outside edge of the desired shape, find the shortest path between that location and the external seed location which stays inside the shape, and set $L_{internal\_path}$ to be the largest of these shortest paths. An example of the longest, shortest path for the desired shape in Fig. 2(A) is shown as the red line in Fig 4(B). With these three factors known, $T_{occupied\text{-}max}$ is just computed to be $T_{occupied\text{-}max} = (S_f \cdot L_{internal\_path}) / (V_{robot})$. This is the worst case distance that a robot must travel to get into the empty location in the shape, $(S_f \cdot L_{internal\_path})$ divided by the average speed at which the robot travels, V_{robot}. See figure 5 for a summary on the update for $T_{occupied}$.

1) Increasing Scale

Once the collective has waited long enough for the empty volumes in the shape to be filled, 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 III.A; 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 III.A.

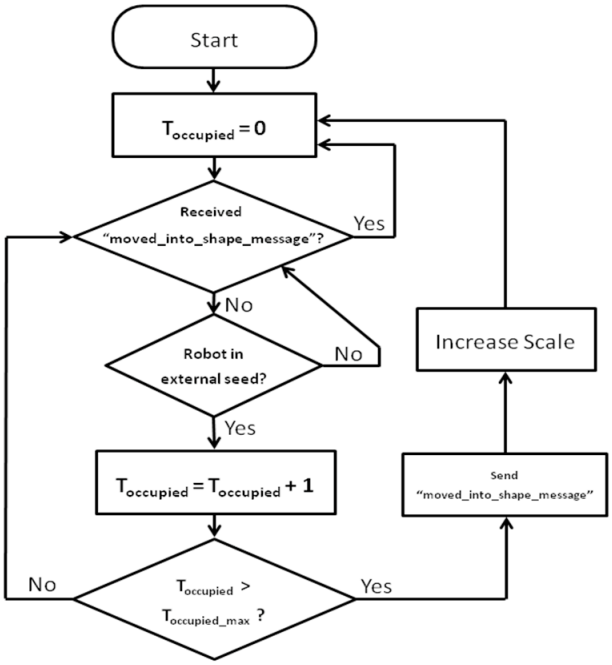


Figure 5. Flow chart for updating $T_{occupied}$

When a robot decides to increase the scale, it should increase it by 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 external seed location.

IV. DEMONSTRATION IN SIMULATION

To demonstrate the behavior of S-DASH, it is run on a simulated collective of robots. In this simulation, each robot is running its own DASH controller to form the desired collective shape, and its own S-DASH controller to determine what the scale of that shape should be. Each robot is shown as a black circle, which is not allowed to overlap with any other robot's circle. All robots meet the assumptions given in section II.A. During each time step of the simulation, the robots are stepped in a random order. During a robot step, that robot reads the messages it has received since its last step, runs DASH and S-DASH, commands its movement, and communicates messages to its neighbors.

In Fig. 6, a simulated collective of robots are tasked to form a five pointed star. Initially, the starting scale, S_{f_start}

is too small for the number of robots in the collective, as shown in Fig. 6(A). Fig. 6(B-C) show the size of the shape increases, as S-DASH increases the scale. Finally, in Fig. 6(D) the shape is large enough to fit all robots, so the scale stops increasing. Next, in Fig. 6(E) the top half of the collective is removed completely, and as a result the scale of the shape is now too big for the current number of robots in the collective. S-DASH will then reduce the scale of the shape, as show in Fig. 6(F-G), until the shape again is at the correct scale to fit all the remaining robots of the collective, Fig. 6(H). This demonstrates the ability of S-DASH to automatically scale the shape to the number of robots, even if that number changes.

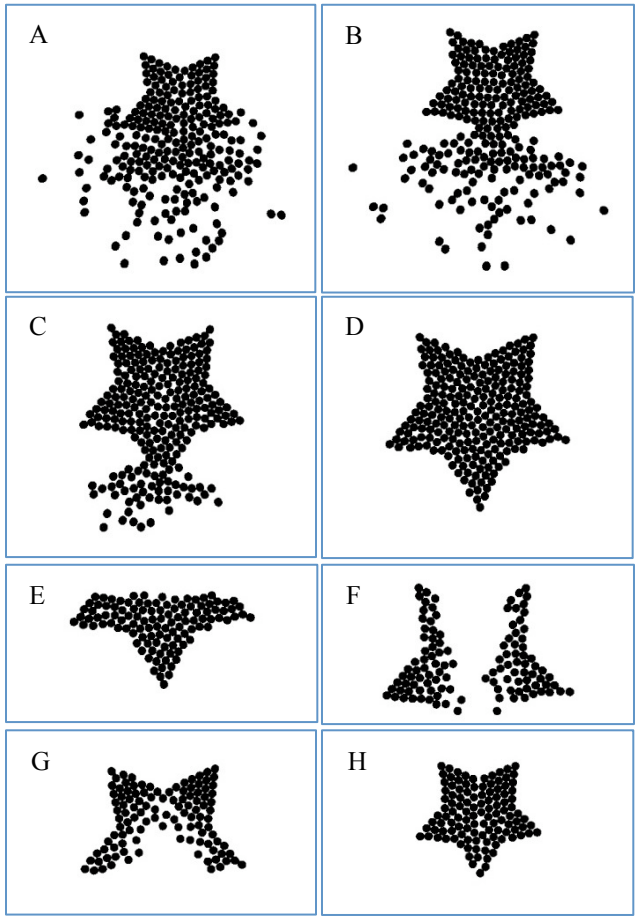


Figure 6. A demonstration of S-DASH adjusting collective scale for the desired shape of a star. (A) An initial formation at too small a scale. (B-C) S-DASH increasing the scale. (D) The shape reaches the correct scale. (E) Half the robots are removed. (F-G) S-DASH reduces the scale. (H) The shape reaches a new correct scale.

Using the simulation, it is also demonstrated that S-DASH stably finds a correct scale, irrespective of the starting scale guess, S_{f_start}. To do this, the simulator is run four times, and in each run S-DASH is given a different value of S_{f_start}. Fig. 7 shows that in these four runs, the average value of S_f for the collective will reach the correct value, irrespective of the starting value S_{f_start}.

CONCLUSION

This paper presented S-DASH, a method for automatically adjusting the scale of a collectives shape. In the future, we hope to implement S-DASH in conjunction with DASH on a collective of real robots. Additionally we would like to adapt S-DASH for self-reconfigurable robots.

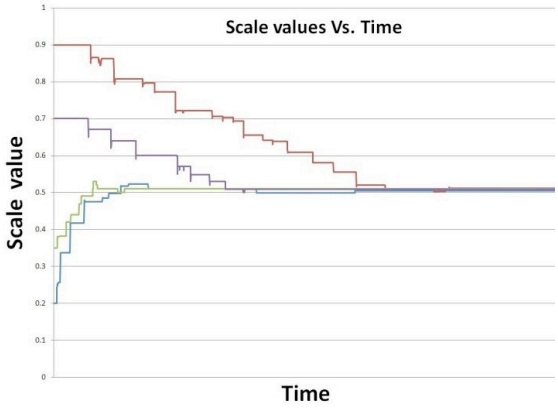


Figure 7. The scale value vs. time for the same collective with four different values of S_{f_start}.

ACKNOWLEDGMENT

We would like to thank the students at the Polymorphic robotics lab for their assistance with this work.

REFERENCES

[1] O'Grady et al. "SWARMORPH: Multi-robot Morphogenesis using Directional Self-Assembly". IEEE Transactions on Robotics 2008.

[2] Chiu, H., Rubenstein, M., Shen, W-M. "Deformable Wheel'-A Self-Recovering Modular Rolling Track." Intl. Symposium on Distributed Robotic Systems. Tsukuba, Japan, 2008.

[3] Arbuckle, D., Requicha, A. "Active Self-Assembly." International Conference on robotics and automation . New Orleans, LA, 2004.

[4] Cheng, J., Cheng, W., Nagpal, R. "Robust and Self-Reparing Formation Control for Swarms of Mobile Agents." National Conference on Artificial Intelligence (AAAI '05). 2005.

[5] Kondacs, A. "Biologically-inspired Self-Assembly of 2D Shapes, Using Global-to-local Compilation." *International Joint Conference on Artificial Intelligence (IJCAI)*. 2003.

[6] Eno, S., 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.

[7] Yim, M., et al. "Modular Self-Reconfigurable Robot Systems: Challenges and Opportunities for the Future." *IEEE Robotics and Automation Magazine*, 2007: 43-52.

[8] Bode, H. "Head Regeneration in Hydra." *Developmental Dynamics*, 2003: 226:225-36.

[9] Stoy, K., Nagpal, R. "Self-repair and Scale-independant Self-reconfiguration". IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Sept. 2004.

[10] Nagpal, R. "Programmable Self-Assembly: Constructing Global Shape Using Biologically-Inspired Local Interactions and Origami Mathematics", thesis june 2001.

[11] Rubenstein, M., Shen, W-M. "A Scalable and Distributed Approach for Self-Assembly and Self-Healing of a Differentiated Shape". AAMAS, May 2008.

[12] Rubenstein, M., Shen, W-M. "Scalable Self-Assembly and Self-Repair In A Collective Of Robots" IROS, Oct 2009.

[13] Moore, D. Et. al. "Robust Distributed Network Localization with Noisy Range Measurements". Sensys 2004.