Self Organizing Swarm With Shape Formation

Kristen Morse and John Kotson
University of Southern California
kmorse@usc.edu*, *kotson@usc.edu

1. Introduction

From the problem space of multi-agent dynamic robotic systems, shape formation presents many issues that are interesting to reason about. Questions arise when considering how the robots should communicate, what knowledge representation the robot should be initialized with, how the robot should envision its environment, how the goal is represented, and how the robot should decide that goal has been satisfied. Our paper takes insight from recent work on solutions to these questions, merges ideas behind that research together, and provides a simulation of a multi-agent robotic system that displays the algorithms we develop. Our work focuses not only on understanding the current research on shape formation, but also to provide a means for others to understand robotic systems. Creating a simple simulation of shape formation with code that is straight forward allows other researchers to visualize our algorithm and develop optimized versions of our shape formation algorithm or implement novel techniques on our system to test other cutting-edge theories.

More specifically, our work builds a simulator in Python that provides a binary bitmap of a specified shape to each robot, creates a target function that acts as the decision making process for each robot, uses a small group of robots to signify the coordinate framework for shape building, and allows each robot to terminate its process once it has reached the correct destination. Consequently, given a simple binary bitmap of a shape, our system builds a satisfiable solution to this goal.

2. Related Work

Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulation:

Arbuckle and Requicha formulate a method for self-assembly and self-repair using a multitude of agents. They compile a set of rules and are able to simulate shape formation with agents running the same instructions. We drew a lot of inspiration for our method of self-assembly from this paper; however, we do not delve into the realm of self-repair. In this paper the agents are all identical squares with orthogonal orientation. Every single agent also has an identical program running that is used for self-assembly. This paper also presents the idea of message passing between the multiple robots. While the message passing in our simulation is far less complicated, it does exist in the target assignment. This target assignment is the portion of

our simulation that tells the each agent which target it needs to navigate towards. Message passing is executed when one agent successfully navigates to its goal, at which point it will notify the next robot in the queue of the updated target location. Arbuckle and Requicha had four different types of messages, two for growing edges and vertices and two for acknowledging edges and vertices. Our simulation simply deals with one type of message that is sent from one agent to another. Arbuckle and Requicha also have a table of rules for dealing with every type of message. Due to the more simplistic model in our simulation we did not have a need for more than one rule, which is to simply go to the newly received target. We also derived our theory of starting with a seed from this paper. The authors start their simulation from the seed and build onward from there. Our simulation uses a very similar method in which our target function uses the seed as a coordinate basis for creating target locations. In conclusion, for our project we drew inspiration from this paper and used simplified methods from this paper to generate our simulations. [Arbuckle]

Programmable self-assembly in a thousand-robot swarm:

Rubenstein and researchers built a shape forming algorithm for a system of 1000 robots with the key ideas of relying on user-specified shapes, edge-following, gradient formation, and localization for self-organization. Rubenstein presents the idea that a group of robots within the swarm act as "seeds," such that the rest of the swarm processes the seeds' position as the origin for a new coordinate frame. As a result, a specified shape has a direction and origin for formation. In order to specify the shape, the researchers feed in an image, convert it to a binary bitmap, and give that bitmap to each robot in the system. Our research pulls heavily from this idea; however, Rubenstein relies on close neighbors and edge-following for localization in his algorithm while we rely on each robot's proximity to the seed and rules-based coordinate system for movement and localization. The algorithm discussed in Rubenstein's paper provides a useful framework for massive multi-robot swarm shaping; however, for the general robotics researcher, access to 1000 robots is not very possible. To bypass the limitations of development on a physical system, our simulator is able to provide a visualization of shape formation for a robotic swarm without requiring future researchers to buy massive amounts of physical robots. While our algorithm is not identical to that of the Rubenstein paper, we capture the ideas of localization, defined shapes, and the use of seed robots to create a coordinate framework and create a system that can reach a satisfiable version of the input shape. [Rubenstein]

Massively multi-robot simulation in stage:

Vaughan and researchers discuss a simulator called Stage written in C++ for the Robotic Operating System[Vaughan]; however it is widely known implementations in ROS are difficult to achieve due to lack of clear documentation and issues over different distributions of ROS. Stage is a powerful tool for robotics researchers; however, due to the high level of understanding necessary to simulate a coordinated robotic system, it is crucial that those with less

understanding have a different way of testing coordination theories. By writing our system in Python, creating easily modifiable functions such that other researchers can change or create algorithms quickly, and modeling our system in a simple fashion: with blocks, an obstacle free environment, we are able to produce a simulator that is user friendly to any developer. Our simulator contributes a tool for visualization of complex algorithms that is simple to understand and easy to develop. [Vaughan]

3. Approach/Implementation

In order to model a shape forming robotic system, our team chose to write a simulator in Python with Pygame. Pygame provided us with a simplistic way of displaying our methods in a two dimensional format.

The agents in this simulation are represented by the boxes that are not green or gray. Each agent has the ability to steer in both the x and y direction. This steering functionality is fundamental for our simulation as it allows the agents to navigate around the simulated arena. For our agent model we are assuming that the robots are equipped with a certain set of laser sensors that can detect objects in all directions of the robot. These sensors allow the robots to detect collisions with one another. After detection of the collision the robots will then take the necessary steps to avoid and recover from this collision. This system is made exponentially easier by the one by one process of our simulation, since only one robot is moving at a time. We chose this method for this purpose and because it provides a more cost effective solution to the overall problem. All agents in the simulation are running an identical program. Each agent waits for its turn to start navigating to the target and passes execution to the next agent upon reaching this target. The next agent then navigates to his desired target and the process repeats for every agent. The target navigation system allows us to create rules for the agents that will eventually lead to construction, or the formation of a given shape. By specifying a different target for every agent we are able to construct any given shape, assuming that we have the adequate amount of agents needed to complete this shape.

We simulated our environment as a space with no obstacles. The environment within the simulation contains a gray shape that represents the binary bitmap each robot has access to. In a physical environment, this shape would not be displayed but for visualization purposes, this shape allows researchers to know whether or not the system has satisfied its goal. Our green "seeds" are displayed as squares similar to the squares of the robot. We model seeds as green place markers on the environment. In a real world scenario, the green squares would be physical markings on the floor of the environment.

Figure 1: The first simulation where the targets are aligned so that the agents form a square. This simulation did not have optimized target selection.

We use these models to implement shape formation by providing each robot with a global goal in the form of a binary bitmap of each specified shape (line, square, or horseshoe). A target function decides the order in which each target location should be satisfied. In each shape, the target function changes slightly. For example, in the line shape (as seen in Figure 2), we wrote the target function such that our robots fill in the shape sequentially from left to right. One by one, each robot is randomly chosen to fill in the next target destination. Upon a robot's predecessor updating the target value, the next robot will find its desired target using the information from the target function and its own knowledge of the frame of reference (using the seed robots as the origin for the shape formation coordinate frame). Next, it will change its X and Y velocities in order to reach its destination, update the target function, and terminate its process. After all robots have terminated a satisfiable shape is visible in the environment.

Figure 2: The second simulation for forming a line shape. The targets for this simulation are in sequential order, from left to right. This is a slightly optimized way to form the shape.

4. Analysis(Results, Evaluation & Discussion)

Results:

The results of our simulation are shown in the videos in the appendix. We were able to generate shapes using our agents for all shapes that we attempted. Pictures of the desired shapes can be seen in the figures one through four. Each figure shows a different state of the simulation and a different target shape. Figure 1 is the initial simulation we set out to achieve. The green boxes represent the seed and the gray boxes represent the desired positions of the agents. We did not use any sort of optimization techniques for this simulation. Figure 2 shows our simulation of forming a line shape. This simulation used a slightly optimized target selection technique. The targets were selected in order from left to right. This simulation shows what happens when there are more agents than target positions. Once the shape is completely formed the next agent will simply perform a random walk looking for a new target. This is the preferred behavior in case the shape would change later in the simulation. Figure 3 demonstrates the horseshoe shape that we generated. In this simulation we purposefully did not create enough agents to form the shape completely. The final result was that the same horseshoe shape was created but on a different scale. This

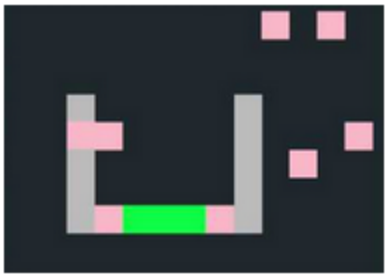

Figure 3: This simulation forms a horseshoe shape. The number of robots in this simulation is not enough to completely fill the shape. The agents are able to form the general shape; however, the formed shape is not the same size as the bitmap shape.

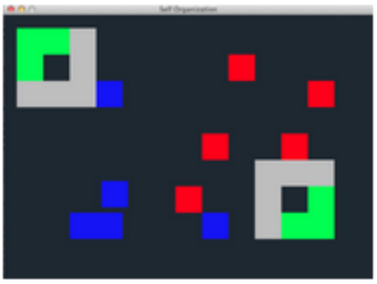

Figure 4: Our final simulation is to form two different squares. The blue agents form the square shape in the upper left. The red agents form the square in the bottom right. Both groups form shapes at the same time.

goes to show that our simulator is robust and can deal with the problem of not having enough agents to completely solve the problem. Figure 4 is the final simulation that we ran using our simulator. In this simulation we broke the agents into two separate groups, red and blue. This is a solution to the one by one approach problem, which creates a huge bottleneck in the program. By having two groups we can run the one by one approach for each group, so that two agents, one from each group, are moving at the same time. Splitting the agents into two groups greatly increases the speed at which a shape can be formed, so long as the shape is broken down into multiple smaller shapes. Complex shapes can be formed much more efficiently using this method. Overall, we have shown that our simulator is capable of generating shapes of all types and can even be broken into groups to enhance the efficiency.

Evaluation:

Evaluation of results was done by determining how well the agents formed the given shape. Imperfections, such as small cracks, exist in the simulation but they are extremely limited and the shape is identifiable to the user. In addition, efficiency of the algorithm can be improved by optimizing the target function with a shortest path implementation that maps robots to the closest target locations.This will decrease each agent's travel distance and will greatly reduce the runtime for the shape formation to finish. On average, each shape formation simulation takes under thirty seconds. Furthermore, increasing the number of robots is not beneficial to the speed of the simulation, in fact it is the exact opposite. Due to the one by one nature of the simulation process, increasing the number of robots makes the simulation increasingly more time consuming. As a result, is best to use the smallest required number of agents to adequately complete the shape. These results would also be very difficult to port to physical robots, since the simulation is written in Python using the Pygame library. However, even though the actual code used for the simulation would not port, the same methods would work on physical robots and would generate the same results.

The approach that we use has many benefits and we chose to use certain methods so that these advantages would be inherent. One of the major benefits to our approach is the readability of the code. We specifically chose Python for this reason. The code is easy to comprehend and is also very modifiable. Our program is straightforward for developers. By modifying the target selection process, one may test alternate algorithms for the path planning or task assignment of each robot. Consequently, our system can be utilized by future researchers in order to test more complex or novel approaches to the shape formation task. One asset of our approach is that it allows the user to input a bitmap of a shape for the agents to form. As a result, given an image that has previously been processed into bitmap form, theoretically our system could form the shape of any image.

5. Conclusion

Lack of simplicity and accessibility can be limiting factors in research. We focused on these two common pitfalls in our implementation for the purpose of delivering a user-friendly and understandable model of a multi-robotic swarm system. This project is open source and available to the general public. Consequently, our simulation is available for development and as a basis of understanding self organization in swarms for other researchers. By providing a tool that has application for developers and researchers alike, we contribute a new way of understanding a complex topic to the scientific community.

6. Future Work

There is a plethora of future work available for this project that is outside the scope of our project. Firstly, one expansion on our project would be to greatly increase the amount of robots used for every simulation. This would allow for larger and more complex shape formations to take place; however, many problems arise with the increase in robots. Python is not as robust as many other languages and therefore does not scale very efficiently. If the number of agents were to be increased drastically, a backend written in C++ would have to implemented to handle all of the individual agents and to help scale the efficiency with the number of agents. Secondly, future work could include getting input from the user in real time to generate the desired shape vertices. Python, and specifically Pygame, provide a method for getting mouse clicks from the user and the location of where the user clicked. This could then be used to allow the user to “paint” the shape they wanted to be formed exactly. A better user interface for choosing different options for shape formation could also be implemented to create better aesthetics for the simulator. Lastly, the future work we would like to eventually accomplish is to generate more algorithms for the selection of targets and target formation. This would allow the user to select which method they would like the program to use in order to generate the target locations for each agent. In addition, using the Python Image Library, we intend to write a function for image processing such that any image can be translated into a binary bitmap for initialization in this simulator.

7. Appendix:

Github Repo for project: https://github.com/kmorse/SelfOrganizationSimulation

To compile and run our code simply type "python 2.7-32 '*nameOfFile*'.py" into a terminal window that is in the directory of the simulation files. Make sure that you have both Python 2.7 and Pygame installed before you attempt to run the simulations. We have included four different simulations so feel free to run them all yourself.

Video of simulation of square shape: https://www.youtube.com/watch?v=eSP3PhJhB8I
Video of simulation of line shape: https://www.youtube.com/watch?v=k3ZArB_Hr9o
Video of simulation of horseshoe shape: https://www.youtube.com/watch?v=3ZtvqiYla6s

8. References

Arbuckle, D. J., and Aristides AG Requicha. "Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations."*Autonomous Robots* 28.2 (2010): 197-211.

Rubenstein, Michael, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm." *Science* 345.6198 (2014): 795-799.

Şahin, Erol. "Swarm robotics: From sources of inspiration to domains of application." Swarm robotics. Springer Berlin Heidelberg, 2005. 1020.

Lerman, Kristina, Alcherio Martinoli, and Aram Galstyan. "A review of probabilistic macroscopic
models for swarm robotic systems." Swarm robotics. Springer Berlin Heidelberg, 2005. 143152.

Vaughan, Richard. "Massively multi-robot simulation in stage." *Swarm Intelligence* 2.2-4 (2008): 189-208.