Replicating a thousand self-assembly robot swarm with software

Felix Cammaerts, r0663453
felix.cammaerts@student.kuleuven.be

May 31, 2020

Abstract

This paper discusses the work done to provide a software alternative to the programmable self-assembly in a thousand-robot swarm as discussed in (Rubenstein, Cornejo, and Nagpal (2014)). Replicating this work with software gives a few advantages: robots can not break down, less energy consumption, easier scaling and quick simulations. Disadvantages include: a simulation on a computer is a discrete simulation which means it can not fully replicate a real world scenario, another disadvantage can also still be found in the scaling as the resources of one machine still remain limited.
Keywords: *Kilobots; Swarm intelligence; Multi-agent systems*

1 Introduction

Self-assembly allows for nature to build complex forms despite the intelligence of one individual still being limited. Look for instance at ant colonies that are able to create ant hills with very complex networks within, sometimes these ant hills can reach up to over the height of a human being. Not a single one of these ants on itself is able to construct such a structure however due to their swarm intelligence they are able to generate a structure which surpasses the capabilities of each individual.

This idea has given the inspiration to perform something likewise using small robots, in which the resulting structure of the robots surpasses the capabilities of all individuals. Rubenstein et al. (2014) has created a swarm of a thousand small robots, also called Kilobots, which are capable of self-assembling into complex shapes. This is achieved by uploading a global map to the robots which contains the final structure which is to be reached. As soon as this global map has been uploaded to all robots they can start self-assembling and form the desired shape. This is done by 3 processes called edge-following, gradient formation & localization.

We tried to replicate this work using a software visualisation tool, we opted to use Visual Basic. We first discuss how we implemented the different parts of the simulation after which we will discuss the results. Finally we will discuss the differences between this simulation and its real world counterpart as well as the advantages and the disadvantages.

2 Methodology

2.1 Seeds & initialization

Just like in the original work we make use of seed robots that are already placed at their final location and have already been giving their final gradient, we also use 4 seed robots and

give them the same gradients as in the original work. (Rubenstein et al. (2014)) However the orientation is different from in the original work, there the robots have been ordered in a rhombus, here we ordered them in the shape of a square, however as a rhombus with equal sides is actually just a square, this is only a difference in orientation and this should not have any further impact. The other robots have been spawned a bit away from the seeds. See figure 1.

Figure 1: The initial position of the robots and the seed robots. The seed robots are green, the other robots are colored red.

2.2 Going to the seeds

The robots are allowed to move one by one to allow to see the movement of each individual robot more easily, in a first step a robot gets out of its initial position and goes towards the seeds. This path has been hard-coded as letting a robot move around randomly until they reach the seeds would take a significant amount of time. See figure 2.

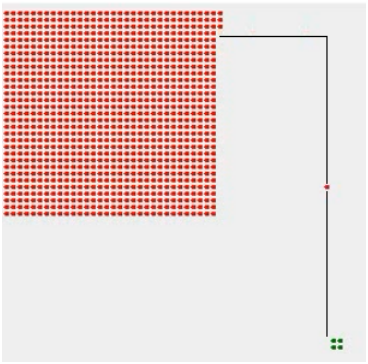

Figure 2: The first robot going to the seed robots. The path taken is marked with the black line, however this line is not made visible during the real simulation

2.3 Edge-following & gradient formation & localization

Once a robot has arrived next to the seeds it will start the edge-following procedure, here the robot stays close to robots that have already arrived. In the beginning this will only be the seeded robots. It will continue doing so until the next step would let the robot fall out of the global map, at this point the robot comes to a stop and updates its gradient in the

same way as in the original work. (Figure 3) At that point the robot will also become an arrived robot. When a robot can choose a direction because both directions would be close enough to already arrived robots, the robot will choose a random direction.

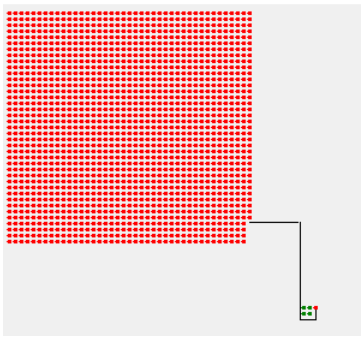

Figure 3: The full trajectory taken by the first robot. The path taken is marked with the black line, however this line is not made visible during the real simulation.

This process is then done for all other robots to form the final shape.

3 Results

There are multiple structures that we can try to create, we will start with more basic ones such as squares and circles before we continue to a non-convex shape.

3.1 Square

We initialize a square consisting of 1000 robots as the global map, see Figure 4. We also measure the total time that it took to finish creating the structure. In attachments there is also a video to show how exactly the assembly process went, this video is named `demo1.mp4` (this video has been sped up to 5 times the original speed). The total time needed to assemble the structure was 19 minutes and 12 seconds equating to 0.152 seconds per robot on average. The final structure that has been assembled is shown in figure 5.

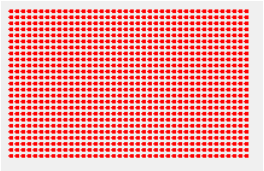

Figure 4: Global map of a square structure.

Figure 5: Final assembly of the square.

3.2 Quarter circle

We initialize a quarter circle consisting of 1293 robots as the global map, see Figure 6. We also measure the total time that it took to finish creating the structure. In attachments there is also a video to show how exactly the assembly process went, this video is named `demo2.mp4` (this video has been sped up to 5 times the original speed). The total time needed to assemble the structure was 18 minutes and 36 seconds equating to 0.863 seconds per robot on average. The final structure that has been assembled is shown in figure 7.

Figure 6: Global map of a quarter circle structure.

Figure 7: Final assembly of the quarter circle.

3.3 Full circle

We initialize a circle consisting of 1017 robots as the global map, see Figure 8. We also measure the total time that it took to finish creating the structure. In attachments there is also a video to show how exactly the assembly process went, this video is named `demo3.mp4` (this video has been sped up to 5 times the original speed). The total time needed to assemble the structure was 17 minutes and 28 seconds equating to 1.030 seconds per robot on average. The final structure that has been assembled is shown in figure 9.

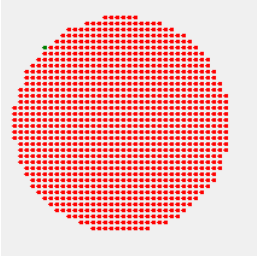

Figure 8: Global map of a circle structure.

Figure 9: Final assembly of the circle.

3.4 Non-convex shape

We initialize a non-convex shape consisting of 1127 robots as the global map, see Figure 10. We also measure the total time that it took to finish creating the structure. In attachments there is also a video to show how exactly the assembly process went, this video is named `demo4.mp4` (this video has been sped up to 5 times the original speed). The total time needed to assemble the structure was 16 minutes and 38 seconds equating to 0.886 seconds per robot on average. The final structure that has been assembled is shown in figure 11.

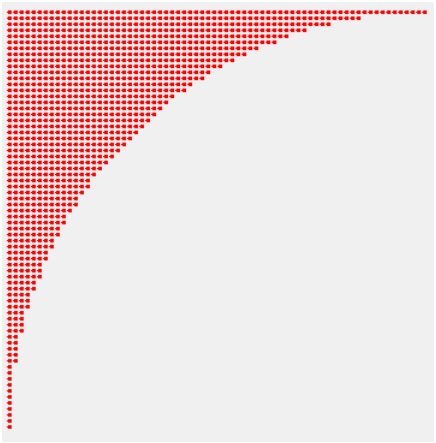

Figure 10: Global map of a non convex structure.

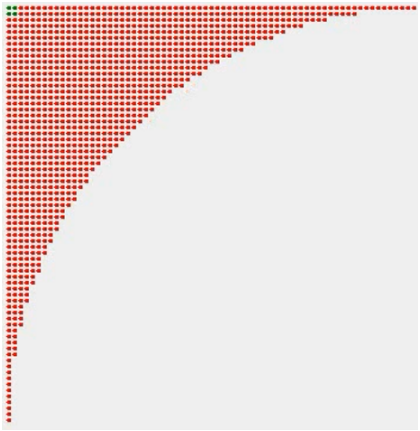

Figure 11: Final assembly of the non convex shape.

3.5 Multi-threaded

We also attempted to use a multi-threaded approach, in which each of the robots is a separate thread. However we were not able to run a lot of robots simultaneously as it turns out that the graphics in Visual Basic can at most be accessed by one thread at a time. This means that very early on during the simulation it looks like our screen froze because there a lot of robots which look stationary without having arrived at their destination this is due to the fact that the threads running those robots are waiting to access the graphics while other threads are using it. This multi-threaded approach is obviously closer to the real world example as all robots are active at the same time, however we do believe that the same results would be obtained as in the single threaded approach. Of course the multi-threaded approach will lead to a faster solution.

3.6 Interesting observations

There are some interesting observations that we noticed during the simulation, we would like to discuss them here.

The first one is the fact that sometimes a robot would jump over the position it was expected to arrive at. This would happen due to all neighboring positions already having robots arrived at, leaving a hole in the structure. Instead of filling this hole the robot would continue past it as it would still believe it is edge-following because it remains close enough to already arrived robots. An example of this is shown in figure 12. The black line indicates the edge of the global map and the red dots indicate the positions of the robots that have already arrived. The green dot indicates the robot that is currently edge following. The robot is expected to go one step to the top as this would lock it into its final position. However the robot would continue to the right if it came from the left or vice versa, this happens because to the robot itself it still looks like it is edge-following due to the next robot to the left/right still being in range to not be able to detect the hole. We solved this problem by letting robots check for these kind of holes in their vicinity as soon as the robot has been edge-following for longer than a certain period of time.

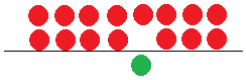

Figure 12: Robot jumps over hole in structure. The red circles are robots that have already arrived at their final location, the green circle is the robot currently edge-following and the black line indicates the border of the global map.

Secondly it is possible that a robot that arrives at its final position will block future robots from filling a hole that it neighbors to. An example of this is shown in figure 13. Here the robot believes it has reached its final destination as moving downwards would lead to it falling off the global map. However the hole above the robot remains unfilled, to make matters worse this robot has prevented any future robots from filling by blocking the last entry way.

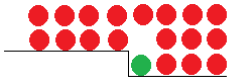

Figure 13: Robot blocks hole from being filled in by any future robots. The red circles are robots that have already arrived at their final location, the green circle is the robot currently edge-following and the black line indicates the border of the global map.

4 Discussion and Conclusions

All structures were perfectly assembled, there are a few reasons to this which all can be attributed to the fact that it is a computer simulation. Firstly all movements are discrete movements i.e. there is a step size with which the robots move, this means that robots are unable to collide with each other which was the case in the original work. Secondly it is impossible for a robot to break down as it is a computer simulation, this is however possible in the real world where this would imply that all robots would have to move around the broken robot possibly leading to some disorientation.

There are quite a few advantages to working with a simulation however, as mentioned above it is impossible for robots to break down and they can also not collide with each other. It is also easier to scale up in software than in the physical world, adding more robots only requires a few changes to the code where as in the real world we would have to make new robots which comes at a much higher cost. Even though we did not check the electricity bill we are quite confident that our robots come at a lower price than the 14$ stated in Ackerman (2014). Secondly the energy consumption of virtual robots is much lower, there is however still a small energy cost to running the computer but this is a lot smaller than the cost of a 1000 small physical robots running for a couple of hours.

Uploading a new global map is also easier in software as it is possible to send this information immediately to all robots, instead of having to upload it all each robot separately which would be the case in the real world.

It is also possible to make a simulation in which only the result would be shown, in this way the simulation can be sped up incredibly much as it is the visualization that slows

done the entire process the most. Running it completely back-end will therefor allow for very fast computation times compared to showing every move of every robot during assembly.

The main disadvantage is that a computer simulation is unable to fully replicate its real world counterpart and in this case it is no different. As said before all steps are discrete while in the physical world movement is continuous, robots cannot fail in software and they can also not collide. These are all things which could happen in the physical world.

The scalability can also be considered a disadvantage in this case, even though it is easy to add new robots, as long as these robots run on the same computer, the resources will remain limited. This means that after adding a certain amount of robots, using a multi-threaded approach the performance will start to degrade. In the real world this problem is circumvented by the fact that each robot contains enough processing power to compute the necessary information needed for its own movement. It would however be possible to split up the needed resources across multiple computers allowing for a better performance.

One last difference from the computer simulation to the real world counterpart is that in the real world the seed robots are places next to all of the other robots, meaning that they can immediately start edge following. In our simulation the robots are spawned a bit away from the seeds for easier debugging purposes and an easier implementation, also the fact that the virtual robots travel at such a high speed allows us to do this without further complications. The seeds in the original work had also been placed as a rhombus whereas we placed them as a square, however we do not believe this makes a difference.

There are a lot of potential uses for Kilobots as discussed in (*Programmable Robot Swarms* (2019)) ranging from underwater robots to flying pollinating microrobots. A computer simulation for these robots and the environments that these robots are made to function in can be helpful to optimize robot behaviour and for testing purposes. The scaling of an existing robot swarm can also easily be tested with computer simulations. However a real world application is still very different from a simulation and such a system is only fully tested when it is proven to be functional in the real world environment.

References

Ackerman, E. (2014, Aug). *A thousand kilobots self-assemble into complex shapes.* Retrieved from `https://spectrum.ieee.org/automaton/robotics/robotics-hardware/a-thousand-kilobots-self-assemble`

Programmable robot swarms. (2019, Sep). Retrieved from `https://wyss.harvard.edu/technology/programmable-robot-swarms/`

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014, Aug). *Programmable self-assembly in a thousand-robot swarm.* American Association for the Advancement of Science. Retrieved from `https://science.sciencemag.org/content/345/6198/795`