NASA Swarmathon Search and Rescue

Jose Medina, Andrey Khlapov, Stephanie Forero, Scott Jagolinzer, Sabri Tosunoglu

Department of Mechanical and Materials Engineering
Florida International University
Miami, Florida, U.S.A

jmedi089@fiu.edu, akhla001@fiu.edu, sfore009@fiu.edu, sjago001@fiu.edu, tosun@fiu.edu

ABSTRACT

This paper delineates research results that was carried out on swarm robots with the purpose of searching a large unknown area for specific objects. The robots used in this work operate using ROS or robot operating system. The search algorithm is coded in C++. Different search patterns were tested to see the efficiency of search of each algorithm as well as how the robot deals with errors accumulated in sensors. Search patterns include radial search, inward and outward circular spirals, rectilinear spirals, and grid-style search techniques.

Keywords

Multi-robot system, swarm robots, UGV, swarm controllers, architecture, search algorithm

1. INTRODUCTION

The NASA Swarmathon is a competition to develop cooperative robotics to revolutionize space exploration. These cooperative robots are known as swarmies.

Figure 1. Swarmies Begin Their Search of a Specific Area

Swarm robots are considered as a new approach to multi-robot systems. Based on its name, multi-robot systems consist of several amounts of robots that can physically move both alone and as a group. The main designs of swarm robots come from the behaviors of social insect such as ants. These behaviors are called swarm intelligence.

Figure 2. Swarmies are Inspired by the Behavior of Ants

When located in hard to reach environments, a feasible method of searching such areas can be swarm robots. By properly combining the shared intelligence of these robots entire areas can be mapped and searched faster than one could by themselves. The problem then arises in how one can properly search open-ended areas and collect meaningful information. The situation we are presented with is the NASA Swarmathon project. The requirements of the project are left as basic as possible to make the logic applicable to other situations. Assuring that the robots can rapidly search an area and collect object is the primary concern. Concerns that arise because of the first problem are making sure that the robot can avoid other robots and walls, and overcoming the shortcoming of hardware. Sensor reliability is important and must be tested to work well in conjunction with the code. Without a proper meshing of both then information the robots collect, as far as its location, will not have enough certainty to be trusted. Inaccuracies in the actuators may cause the robot to be off its intended location and areas of the importance may be missed throughout its search. For this the other main challenge is overcoming the sensor limitations and making sure the robot's logic can handle these inaccuracies.

The robots we will be using are limited to certain hardware as defined by NASA. This hardware includes 4 motors, three ultrasound sensors in the front of the robot only, a serial arm to grab

object, a camera, and a GPS sensor. We are also provided with a base code which is used to make the robot search. This base code can be altered to increase performance. Actual hardware cannot be modified and will be the final configuration.

2. Search Pattern Background

A. Global Maxima Search

The approach is useful when search region boundaries and target probability are known. The algorithm divides search area into grids and assigns a value of the integral of the probability under the grid to each cell. The algorithm generates trajectory to visit the cells with highest probability and clears the grid-cell once visited. The strategy provides trajectory with multiple overlapping segments due to the shift of global maxima across the search region [1].

B. Local Maxima Search

This strategy uses the same probability distribution across the grid-cell and clears the cells with maximum values within the local maxima-search results [1]. The trajectory for this search is less overlapping, but the strategy depends on the initial position and robot is tend to stay in the local maxima area.

C. Radial Search

The algorithm involves a repeated sequence of moving from the center and then making a return. Each new sequence starts with the new heading. The search strategy is greatly improved when outward and inward paths form a loop [2].

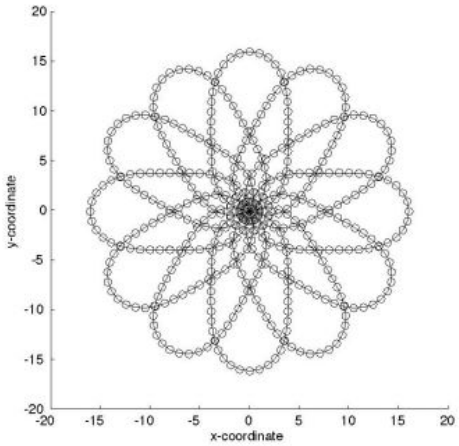

Figure 3. Radial Search

D. Bees Algorithm

This strategy models the foraging behavior of the colony of bees for the riches and closest food source. The algorithm that was proposed originally was a combination of random search and neighboring search. The goal of the search strategy was to find a single value which represents the global optimum. A variation of this strategy would be a Distributed Bees Algorithm (DBA), suitable for multiple objects search mission. The objective of this improvement was to attract more robots to the areas with higher target distribution. When robot finds a target, it communicates with other robots and sends target location to the robots in the range [3].

E. Particle Swarm Optimization, variations and improvements.

PSO is one of the earliest and well-known search algorithms for swarm behavior, provided by Kennedy and Eberhart in 1995 [4]. It provided the inspiration for Robotic Darwinian Particle Swarm Optimization (PDPSO), Extended Particle Swarm Optimizations (EPSO) and many others. PDPSO consist of robots that move collectively in search area and work for optimal solution. Each robot characterized by the performance, position and heading. This strategy allows robots to overcome problems related to obstacle avoidance, robot dynamics, sub-optimal solutions and communication constrains [5]. EPSO algorithm does not use multi-hop connectivity compare to PDPSO and makes each robot to consider information only with robot with fixed radius of communication. This strategy also uses different obstacle avoidance approach.

3. Search Pattern Prototypes and Concepts

Since the search starts from initial position in the center of search area, then we propose that a spiral search algorithm to be a reasonable choice. This algorithm is very efficient and in theory provides complete coverage of circular area where radius of the circular area equals half distance of the search arena. In the spiral search algorithm, we have options to use two patterns: inward spiral and outward spiral. The inward spiral pattern first identifies the search region and boundaries and moves forward toward the center. The outward spiral pattern initialized from the center of the search area and expands outwards to cover entire region.

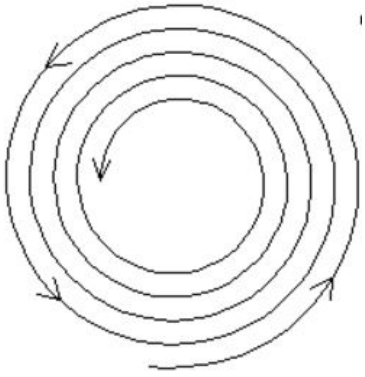

Figure 4. Inward Spiral

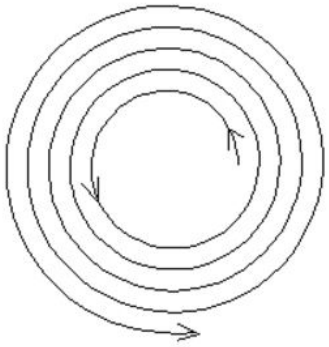

Figure 5. Outward Spiral

The inward spiral pattern takes much longer search time, but ultimately provides better search results. On the other hand, the outward spiral minimizes the search time, provides greedy search, but at the end the success rate is compromised.

Since it takes time for the Bot to bring the cube back to the collection zone, we propose to use outward spiral pattern. The Bots will collect maximum amount of cubes in the vicinity of collection zone which is also the starting point for the spiral.

The onboard web-camera which is used to find and identify cubes has a very specific parameter. It processes the pixels of the upcoming images left to right, top to bottom. Thus, in terms of the Bots view, the cube in the upper left corner of the screen will be identified first.

Considering this feature, we chose to have a counter-clock rotation for our spiral pattern. By doing so, the Bots will be always collecting the cubes on the "Inner" side of the circular path.

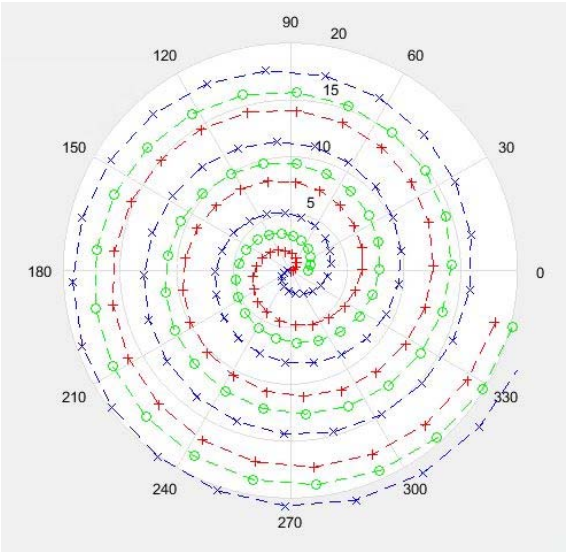

Figure 6. Counter-clockwise Outward Spiral Path for 3 Robots

It is important to note that Bots will start their spiral paths not side by side, but rather equidistant from the center point of the search area.

Once the cube is picked up and delivered to the collection zone the Bot can start a new spiral pattern, from the beginning or can resume the spiral where it left off.

As an option, the Bot could be programed in such way that after certain amount of returns to the collection zone it will increase either initial radius of the spiral with respect to time travelled from the center, or increase distance between the spiral turns. This option will help to deliver the Bot to the outer range of the spiral since most of the cubes within the close proximity of the center have been picked up.

Once the Bot reaches the boundary on the outer side of the spiral it will either change the direction to avoid the wall and continue the spiral path or could be programmed to use different algorithm to cover 4 corners of the square arena.

The spiral search is only efficient if there are no errors in navigation. Since our Bot is equipped with the onboard GPS with accuracy of 3-6 feet, we cannot rely on that feature for our spiral path. The only options left for us are encoders on each wheel. The accumulated distance with each turn of the spiral among with slippage of the wheels due to the turn will lead to the error in distance and consequently the path travelled. This might lead to the rapid decrease of the quality of coverage.

The collision avoidance behavior for the Bots should also be taken into consideration. At the beginning of the spiral the Bots will be close to each other. Once one of the Bots will identify and determine the cube, it will become a stationary, possibly blocking the spiral path of the other Bot. The obstacle avoidance algorithm for the bot could be made to assume the robot is a wall and alter its trajectory slightly.

Another possibility for the search pattern is too divide the known search area into different quadrants. Using the onboard compass of the robot, the quadrant assigned to each robot is determined based of what the original trajectory angle is. Once the robot has been assigned a quadrant, it begins snaking outward towards the wall.

This snaking motion will increase in height with each horizontal motion. The move movements will be strict linear movements of 0.5 to 1 meter increments. After each interval, the robot will scan its surrounding search for cubes. The intervals are kept small to minimize any missed cubes due to the limitation of the robot's camera.

4. Logic Structure

There exist multiple ways that this robot can be programmed to perform its desired application. The chosen structure for our robot is with State Machines.

State Machines or the method of using State Machines are defining each task the robot must perform within different cases. The main logic function for this method is the switch case statement.

Certain identifiers that the robot may encounter throughout its operation will prompt it to switch to a different "State" of operation. For our specific application, these prompts to switch to a different state can be obstacle detection, object detection, verification that the object has been collected, drop-off zone detection, and multiple others.

A basic example of the framework state machine logic can be seen in the following image.

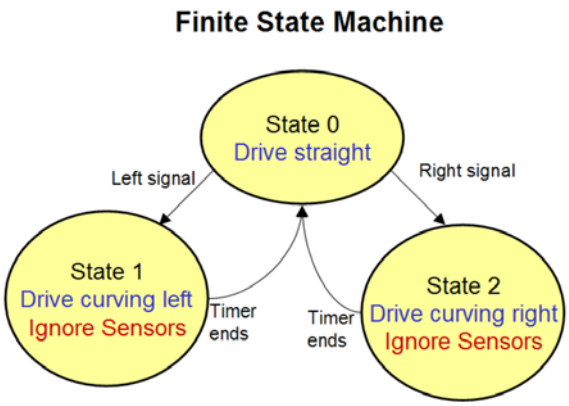

Figure 7. Example of State Machine Logic

The main benefit of programming in terms of state machines is that it allows the robot to continue looping into its function without being locked into that specific function. The robot can exit into a different state at any moment if the correct prompt is presented. The robot is also able to always revert to its original state and determine its following action. New states can also be added to the robot and the redefining of the rest of the code to accommodate this addition is not as strenuous.

5. Search Algorithm Testing

A Spiral Search

After narrowing down the potential algorithms to two potential patterns we begin the coding and simulation of the search. We begin with the outward spiral search pattern by using the below equations to affect the goalLocation.x and goalLocation.y variables. These equations are placed in the ”Transform” stage of the state machines for the robot.

goalLocation.theta = currentLocation.theta + (M_PI_2)/12;

goalLocation.x = currentLocation.x + ((0.01 + .03*theta)* cos(goalLocation.theta)) ;

goalLocation.y = currentLocation.y + ((0.01 + .03*theta) * sin(goalLocation.theta)) ;

theta = theta + (M_PI_2)/12;

In the above equations, the orientation of the robot, defined by goalLocation.theta, is defined by adding whichever angle one chooses. In order to maintain a tight uniform circle, the angle chosen as what is essentially a step size is 7.5 degree increments. This could be made smaller or larger if one chooses. The rectangular coordinates of the goal location are defined using altered version of the Archimedes spiral equations. The general form of these equations is the following.

$$x_{component} = (a + b\theta)cos(\theta)$$
$$y_{component} = (a + b\theta)sin(\theta)$$

The additions of the currentLocation variable to both equations are done in order to make sure that goalLocation coordinates are relative to the actual position of the robot. The more important variables of interest are the “a”, “b”, and “theta” variables. These variables in tandem control the spacing in between loops of spiral and how tight of a spiral it is. The “theta” variable will have the same step size as the goalLocation.theta variable. Manipulating those above variables help change the shape of the spiral which will directly influence search time and area coverage.

Initial spiral search patterns were done with larger turning angles in order to avoid slippage of the robot wheels by using only rectilinear movements. The angle increment is responsible for how circular of a spiral one gets. The two angle measures tested for uniform expanding spiral were 45 degree increments and 7.5 degree increments. At the 45-degree increment setting, larger values of “a” and “b” are required in order to get a good balance between search area and time spent searching. An “a” and “b” value of greater than 0.1 is recommended for the 45-degree setting in order to get a good balance between the area covered by each robot and the time it takes it to reach the end of the area. For the 7.5-degree setting, “a” and “b” values less than 0.05 are recommended.

The following is an example of the rectilinear spiral and the circular spiral.

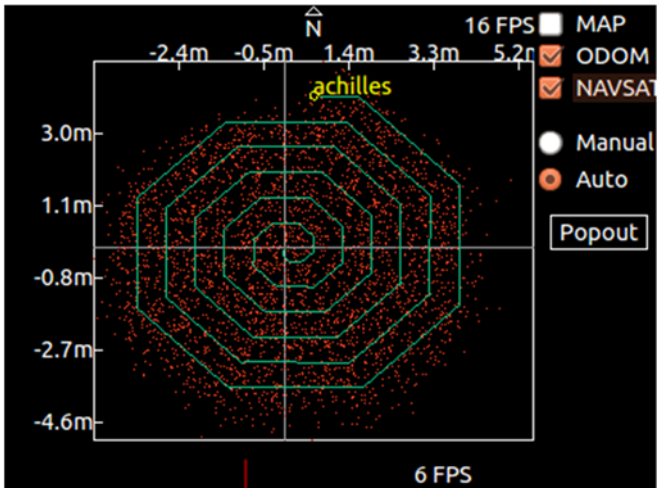

Figure 8. Rectilinear Spiral

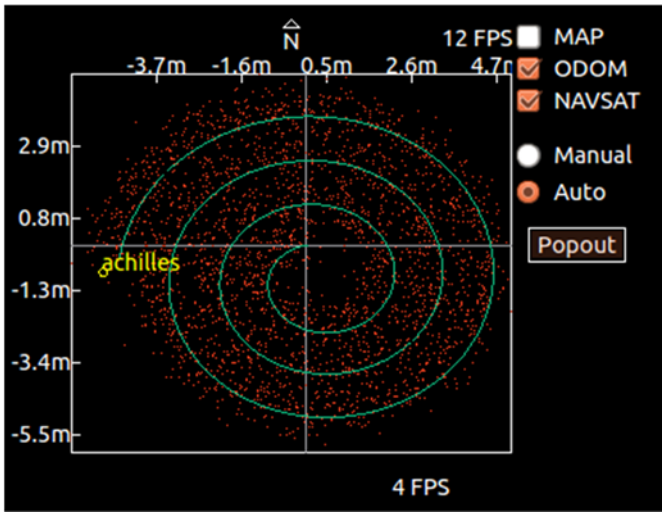

Figure 9. Circular Spiral

In the above examples, values of 0.1 and 0.05 are used, respectively for each spiral type. Each one was tested at the beginning of its range and how much area was covered was compared to the time it took. The rectilinear spiral was tighter at its minimum range, this however can be fixed easily. The circular spiral has good spaces in between its loops, to make it tighter the values can be reduced. Even

taking into account the spacing in between each loop, it takes the robot longer to search in the rectilinear spiral than in the circular spiral. This is because it takes the robot longer to turn to its orientation because it is larger; 45 degree turns compared to 7.5 degree turns. This can also be seen in the following two images.

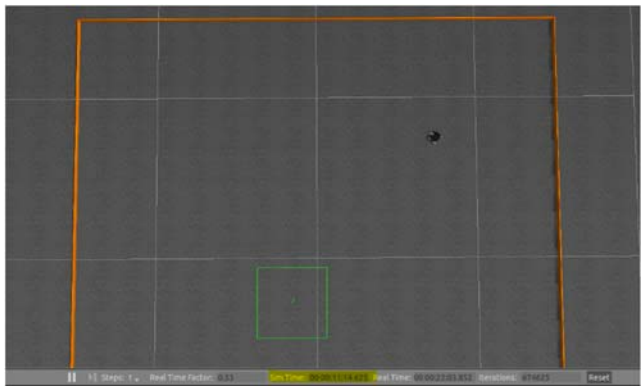

Figure 10. Rectilinear Spiral Simulation Time

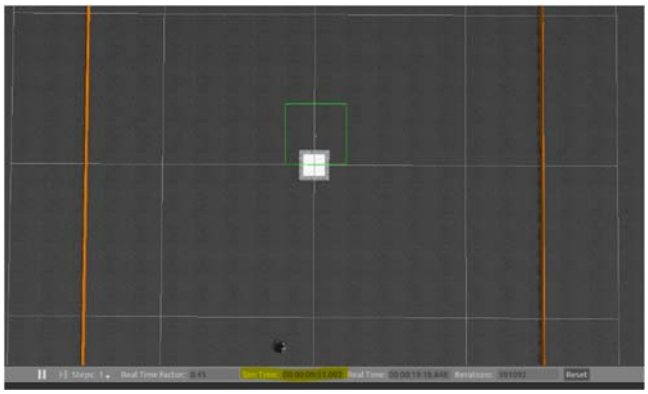

Figure 11. Circular Spiral Simulation Time

The time taken by the circular spiral is less than the time taken by the rectilinear spiral and the circular spiral was still able to cover more area than the rectilinear spiral. This however can be remedied significantly for the rectilinear spiral by increase the space in between each loop. This will aid in time used up to search the same amount of area. The time for the rectilinear is still slightly longer then the circular spiral because of the larger turning angle after each increment.

With regards to the space in between the loops, it is most optimal to have a space of 1.5 to 2 meters in between each loop. This is done to accommodate the capabilities of the robot's camera. The camera has a visual area of about 1.5x1.5 meters. With a space in the above specified range, the chances of finding a cube are increased and any cubes not seen by the camera by one robot can be found by one of the other robots that loops adjacent to it. This will help with both time without sacrificing covered area. The rectilinear spiral also has the added benefit of not accumulating as much error in its movements s the circular spiral does. Rough surfaces, such as gravel which has many individual rocks, may pinch wheels and twist the robot slightly and deviate it from its desired destination. This happens more often when the robot attempts to twist as it moves forward as in the case of the circular spiral. So, keeping linear movement and turning separate reduces the buildup of error throughout the search.

The return function for either one of the spiral patterns chosen were the straight shot to the base option. This was chosen over the reset option because of time constraints. The reset option is more secure in returning to the base, however it will take up too much time after multiple collections.

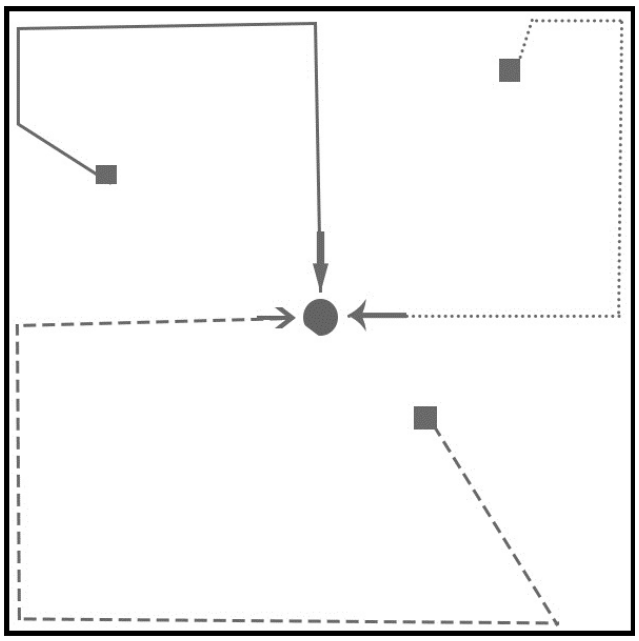

Figure 12. Return to Base Reset Option

The straight shot to base option is far more time efficient but depending on how much error has been accumulated throughout the search the robot may miss its destination. In order to correct this, once the robot has reached its assumed goal destination its spins around 90 degree and moves forward, spin another 90 degrees and move forward and continues this pattern. The forward motion is done at varying increments in order prevent the robot from moving in a localized circle. The robot repeats this looking for the markers that indicate the drop off area. In recent testing, the assumed goal location has not been far off from the actual drop off location so this method has worked quickly on each return.

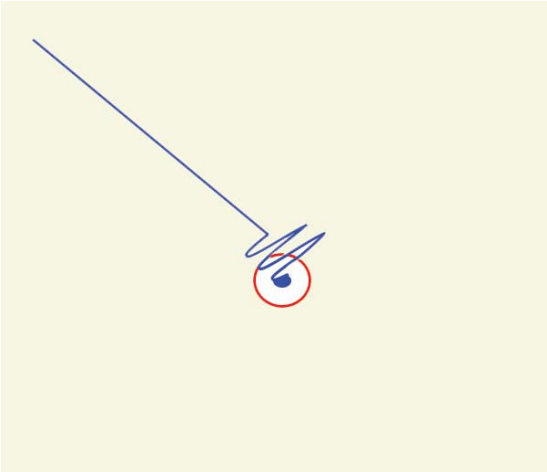

Figure 13. Return to Base Straight Shot Option

After each return the robot currently backs away and begins its spiral search once again. This function is currently very inefficient, to improve we intend to program a function to have the robot return to where it left off, the can also be subject to error but a rough estimate will still be better than beginning a search from the beginning.

All the logic coded for the spiral algorithm was placed in the first State Machine. This done to prevent over-correction by the robot. The state machine that controls the actual inputs to the motors verifies its goal location will be as accurate as possible by correcting its orientation and errors in it, as it drives forward.

Figure 14. Simulation Result of One of the Zone's Pathing with the Little Circles Representing the 360 Sweep at Each Space

These corrections are small and generally are not noticed. However, by placing the equations that control the shape of the search pattern in this state the robot tries to resolve the goal location to a new goal location based on the compass at every step instead of after it completes its motion. While this may not seem like an issue what happens is that the robot attempts to be too accurate, far more than it is capable of being. Because of this instead of moving forward as one would expect it to, the robot pivots in place indefinitely without making any forward motion. It appears to be stuck in place. By placing the goal location definition in the first state machine, the robot will not attempt to correct itself to a continually defined goa location. Instead it will define its new goal location after it has finished its motion.

6. CONCLUSION: FINAL SEARCH ALGORITHM

The chessboard search method allowed for us to relax the tolerances on positioning to a level that we felt could be reliably maintained and the rovers are accurate enough to make 1 meter cuts in a given direction. This method allows for us to track the rovers in terms of a 1 meter by 1 meter space that they occupy instead of a coordinate that could accumulate a tracking error over the course of the run.

Through testing in both simulation and on the physical rovers, we feel confident that the rovers can detect, pick up, and return the cubes in a quick and efficient manner with minimal need to have a positional recalibration. The one concern is a scenario when multiple rovers return at the same time and detect each other. However, that case should be infrequent enough that if it were to happen and they were sent to recalibrate it should minimally impact the performance.

7. REFERENCES

[1] Meghjani, M., Manjanna, S., & Dudek, G. Multi-target rendezvous search. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). doi:10.1109/iros., 2016.

[2] Russell, R. A., TASTI Follows Chemical Trails with its Robot Tongue. IEEE International Conference on Robotics and Biomimetics. doi:10.1109/robio., 2006.

[3] Senanayake, M., Senthooran, I., Barca, J. C., Chung, H., Kamruzzaman, J., & Murshed, M.. Search and tracking algorithms for swarms of robots: A survey. Robotics and Autonomous Systems, 75, 422-434. doi:10.1016/j.robot.2015.08.010, 2016.

[4] Eberhart, R., & Kennedy, J. (n.d.). A new optimizer using particle swarm theory. MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science. doi:10.1109/mhs., 1995.

[5] Couceiro, M. S., Vargas, P. A., Rocha, R. P., & Ferreira, N. M. Benchmark of swarm robotics distributed techniques in a search task. Robotics and Autonomous Systems, 62(2), 200-213, 2014.