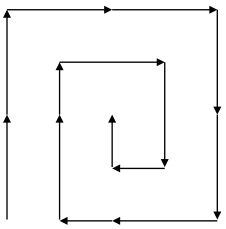
Shrinking Square Search for Swarm Robotic Systems

Montgomery College NASA Swarmathon Team A++

Student Members:
John Ajmian
Niko Dietz
Francesco Gallegos
Fernando Piedra
Michael Roa
Charles Varga

Academic Supervisor: Professor David Kuijt

Abstract
We address the issue of insufficient space covered by the wave pattern search algorithm by implementing an improved algorithm which places the rovers in corners and divides the search space up according to the number of rovers. The rovers search their designated areas in a shrinking square pattern in which the rovers repeatedly travel in straight lines which gradually decrease in length per 90º turn. This solves an issue of insufficient space covered, as it will become apparent in this report. In some of the previous algorithms with which we experimented, the rovers have more corners within the search areas, which makes turning significantly more time consuming and searching more inefficient, since the rovers have to check more often for collisions and make more turns to adjust. This algorithm is designed to minimize the time spent turning and maximize the time spent searching for collectibles.



This visual diagram of the shrinking square search pattern is not to scale.

Introduction
Our goal is to calibrate the given code in a manner that improves the code which we are provided. As we found in our experimentation, this is significantly more efficient than creating separate methods for our algorithm. Most of the calibration was done by modifying SearchController.cpp, and a few other files such as mobility.cpp. By modifying mobility.cpp, we were able to get a general idea of how each of the different methods affected the rover's movement, and we were able to calibrate the rovers' turns so that they make 90º turns when necessary and adjust their turning angle to get around obstacles and avoid collisions.

Related Work
We based our idea of a shrinking square search off of a searching tactic used by the United States Coast Guard: expanding square search. As indicated by Singh's *What is Expanding Square search as per IAMSAR?,* the expanding square search starts off at a base point and moves away from that base in perpendicular lines that increase in length per 90º turn [1]. We took this search method a step further in the context of this competition by reversing the pattern so that the rovers search in a shrinking square. We believe that this is a more efficient method in the context of this competition not only because the rovers are collecting cubes and returning them all to the central hub, but also because the rovers search the entire given field with this method, which makes the algorithm virtually free of errors. The expanding square search's purpose is to repeatedly expand the scope of the search area until the lost object is found; however, in this case, we know that we are bound to a map of limited size; therefore, we can use the advantage of knowing the map size by starting from the outside in, repeatedly shrinking the scope of our search area.

Methods

SearchController.h
These are the following variables that are used by SearchController.cpp: *init, prelim, started, rover, and movementTracker.* There

are several other variables that have been added to this file that are used nowhere else except for the constructor; however, we decided against the deletion of these respective variables for the purpose of possible future experimentation. The important ones serve the following purposes: *init* is a boolean that is set to true once the search is complete. *prelim* is a boolean regarding whether we are in a preliminary round, where there are only three rovers, or whether we are in a regular round, where there are six rovers. *started* is a boolean that is used as a way of checking if the rover has started its search pattern. The int *rover* is used to assign ID's to each of the rovers in the regular rounds. *movementTracker* is an int variable that is used as a way of tracking the rover's movement and determining which next maneuver is necessary in the search pattern.

SearchController::SearchController

We initialized each of the used variables from SearchController.h within the constructor for the purpose of enhancing the different instances of SearchControllers, whereas the original constructor only had a random number generator. For example, we initialized *init* to false within the constructor since the constructor is called when the search begins (i.e. when the rovers are in their initial position), and when *init* is set to true, the method where *init* is checked is not called again. Additionally, the *prelim* boolean is initialized to true so the SearchController automatically assumes that we are in the preliminary round unless it is determined otherwise. Likewise, *started* is also initialized to false. *movementTracker* is initialized to 1 in the constructor because in the *SearchController::Search* method, *movementTracker* is used in a switch statement. In the case that *movementTracker* equals 1, the rover migrates to the upper-left hand corner of the map to begin the search pattern.

SearchController::search

We have designed this method not only to be multipurpose, but also endpoint oriented, which means that movement within the method ends with a checkpoint followed by a turn, and the sequences of movements and turns all end in the central hub, at which point the search is complete. As a result, *goalLocation* plays a significant role in this method, particularly in the switch statement for *movementTracker.* The method is designed in a multipurpose manner for security purposes. Each of the steps within the method are dependent on the previous one.

The rovers automatically move to their starting position since *init* is set to false in the constructor. After that, the method checks if we are in a preliminary round by calling the *setArenaClient* and initializing *startSearchWidth.* If ROS is unable to call the client, *startSearchWidth* is initialized to 0. If the client is called, *startSearchWidth* is initialized to a non-zero value, and *prelim* is therefore set to true. Next, the rovers adjust their positions by calling the ServiceClient with the argument "get_pos_adjust." Each of the robots have different names under the string *robotName,* which makes it easier to assign them different starting positions. For additional security, we assign the different rovers ID's for non-preliminary rounds based on their positions on the field.

It is assumed at this point that the rovers are in their initial position of the search, so they can begin the shrinking square search pattern. There are two different if statements for preliminary rounds and non-preliminary rounds, but they each have the

same sequence of movements which are maintained by the value of movementTracker, which is why it is initialized to 1 in the constructor. Within each if statement, there is a switch statement for movementTracker, and every case, as it becomes apparent in the code, is carried out sequentially. In the case where movementTracker is equal to 1, the rover moves from the bottom-left hand corner of the field to the upper-left hand corner of the field (*goalLocation.x = currentLocation.x + ((7.5) * cos(goalLocation.theta)); goalLocation.y = currentLocation.y + ((7.5) * sin(goalLocation.theta));*). Then the rovers turn right and move to the upper-right hand corner of the field. The rovers continue this sequences of turns and movements to form a square which shrinks in size per repetition of this method. They continue this search pattern, collecting cubes along the way and dropping them off at the central hub, until they reach the central hub through the repetitions, at which point the search ends. With every repetition of this method, goalLocation is updated and returned.

Experiments

Due to the time constraint, only one experiment was able to be performed on this algorithm. Our experiment involved repeatedly testing the algorithm on the rovers and finding out how many targets were collected.

Results

The algorithm seemed to produce inconsistent, spread out results, as it becomes apparent in the chart below. Additionally, not more than 60 targets were collected in any of the five runs. The results produced a mean of $\mu=34.6$ targets and a standard deviation of $\sigma=10.644$ targets.

	A	B
1	Trial	Algorithm
2	1	52
3	2	34
4	3	33
5	4	23
6	5	40
7		

Conclusion

Although this algorithm was not developed to its fullest potential, our team did have many issues with determining an appropriate algorithm to use for the project. Alongside that we had many changes in membership, which made the time constraint a larger burden on us. But this entire activity was a learning experience for all of us, which opened our eyes to the reality of working in a professional environment. Time is not always going to be on your side to complete projects, but you have to be able to adapt in order to finish things off in a timely manner.

References

[1] marinegyaan.com/what-is-expanding-square-search-as-per-iamsar/, Web March 31, 2017.