|  |
| --- |
| **Rose-Hulman Institute of Technology** |
| **Arkin Final Report – Localization and Search** |
| **ECE425-Mobile Robotics** |
|  |
| Ander Solorzano & Ruffin White |
|  |

Rose Hulman Robotics Team CM 5000

5500 Wabash Avenue

Terre Haute, IN 47803

# ABSTRACT

The purpose of this final project is to demonstrate the integration of some of concepts learned in this quarter by creating localization and search algorithms for the CEENBoT platform. The localization task involves using sensor feedback with a navigation routine to determine the location of a lost robot in the world, the location of a fire (heat) source, and then rescue it by moving it to its home location. The CEENBoT robot uses a total of four IR range sensors, with a single IR sensor attached to each side of the robot, for detecting walls and/or close proximity obstacles. The robot uses these sensors to map the world, localize itself, and verify that it has entered or left a discrete cell in the world. Located in the front of the robot, the CEENBoT also uses a heat sensor to detect heat sources and move towards or away from them. Using a very efficient mapping and localization algorithm, our robot managed to constantly locate itself in the correct cell and orientation within 3-5 moves or turns. Rarely did our robot get lost while exploring due to non-systematic errors such as odometry or IR sensors errors. Using our methods described in this report, our CEENBoT robotic platform, named Arkin, managed to come up in second place during the competition.

Table of Contents

[ABSTRACT 1](#_Toc348885237)

[OBJECTIVE 3](#_Toc348885238)

[THEORY 3](#_Toc348885239)

[METHODS 3](#_Toc348885240)

[Localization 3](#_Toc348885241)

[Navigation 6](#_Toc348885242)

[RESULTS 7](#_Toc348885243)

[CONCLUSION AND RECOMMENDATIONS 7](#_Toc348885244)

# OBJECTIVE

In this final project, the CEENBoT robotic platform will use localization, mapping, exploration, and control technique learned and discussed in the class lectures. The robot world consists of a 6’ x 6’ boxed world consisting of 18”x18” cells. The world is therefore divided into a 4 x 4 array that we will refer to as gateways.

The robot will have a prior knowledge of the world and its starting orientation but it will not know where it is starting or where a potential heat source (i.e. the fire) is located. Once a user presses GO, the robot should start moving around the world and localize itself within 3 to 5 moves or turns. The length of time that it will take for the robot to localize itself depends on the uniqueness of gateways that the robot traverses. Once localized, the robot will then proceed to explore the world until it finds the heat source. Once the robot detects a heat source, it will finally proceed to its user-designated home position where it will report the location of the heat source.

# THEORY

# METHODS

## Localization

The localization method implemented within our robot utilizes a method of deductive calculations and deterministic expansions of possible candidate locations rather than by discrete probabilistic means. Using a tree structure to hold historical orientations, observed gateways, executed movements our robot uses a brute force algorithm to deduce possible origins and current locations. Upon initialization, the robot is already given a complete map depicting every cell, and every sells specific gateway (specifically the orientation of every possible wall with reference to a northern orientation). The robot initializes the tree by acquiring its root seed, or otherwise known as a starting location. It is not entirely accurate in referring to the starting location as always the root seed, as our tree resembles a dynamic queue with regard to the nature of a queue data structure, with the oldest root elements popped out and newer ones placed on top of the queue. In that respect the oldest element of the queue always serves as the root seed for the tree.

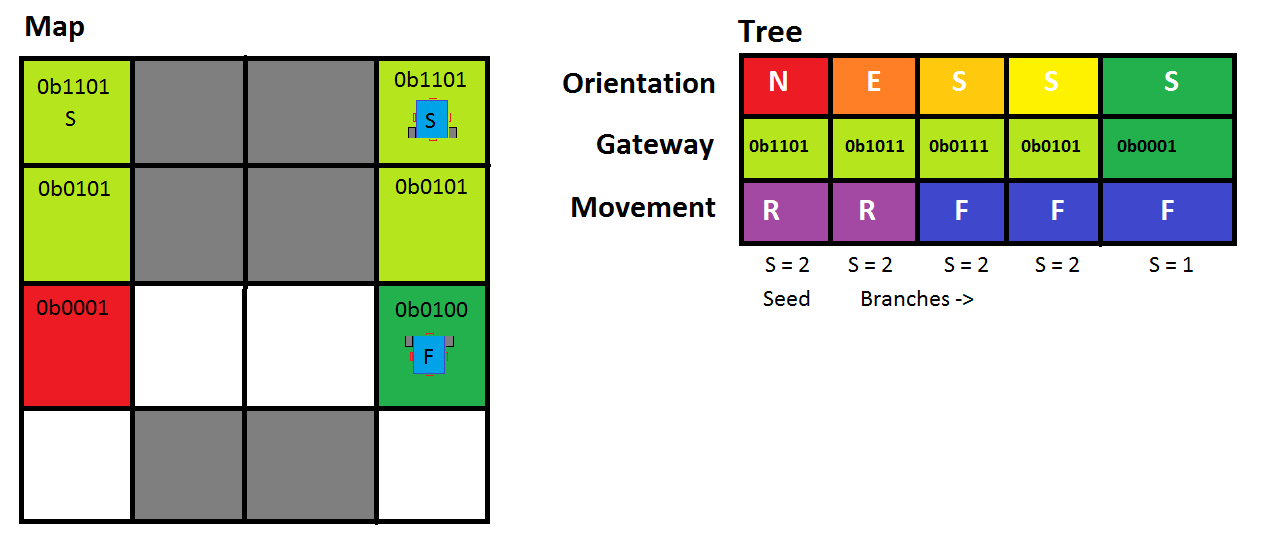


Figure : Localization Process – This figure shows the propagation moment of a kidnaped robot, the robots possible locations within the map, as well as is growing tree structure used to solve for its location from start to finish.

By inspecting Figure 1, one can observe a test case scenario where the robot is placed within a symmetrical environment in the top right corner. The robot is first placed within its environment in a front facing North orientation; it is possible to use the same localization process to localize from any starting orientation and will further be explored later in this document. The robot continues the localization process in a circular fashion of sensor acquisition, data processing, and motion execution. Once the tree has been initialized with the root seed using the sensor data, that specific gateway is then used as an ideal template to search the map for possible matches, or potential seeds. In this case the robot will find potential seeds that possess identical gateways. We use the number of resulting potential seeds as a method of determining our state of confusion; a single seed would entail that our exact position has been found as there is only one solution for our growing tree remains. Multiple seeds would entail that multiple solutions are still possible and that not enough branches have been gathered or that our tree does not contain enough uniquely identifiable features. While lastly zero seeds would entail that no solution has been found and that either our map data is incorrect and we have stumbled across uncharted territory or that in fact are recorded gateway data has succumbed to some sensor error.

Continuing with the test case scenario as shown within Figure 1, two potential seeds still exist, and as only one branch has been constructed, further measurements are still necessary for complete localization. Using our standard maze navigation algorithm as developed within our exploratory mapping algorithm, the robot continues to explore on the conditional premise of the behavior that follows; if the left wall is open, proceed to make a left turn, otherwise proceed to check the front wall if for if forward motion if possible, and should neither be the case, proceed to make a rightward spin in place. As shown in the figure, specifically the growing branches within the tree, the robot proceeds to spin right twice, then move forward twice. Up to this point, the two potential seeds that have been found remain suitable solutions for our current tree. It is only until we reached the fifth branch within out tree do we encounter a uniquely identifiable feature that is the deciding factor for this case scenario. Due to this specific mirrored symmetry of the particular gateway we currently observe, as well as the sequential pattern and history of our orientation and movement, we can thereby determine that there remains only one possible solution for our elongated tree.

In this specific example, I have chosen one of the starting points in orientations that would result in the longest path to localization, thus proving in this particular instance, that the minimum number of necessary branches for immediate localization, or localization as soon as possible, is five. However this algorithm is quite suited for larger or more symmetric maps and is the case where we maintain this parameter, the max length of our tree, configurable upon compile time. Another interesting note is our algorithms capacity to localize under fewer than five iterations. Given the case where the robot is placed within the middle of one of the T intersections, remaining of course is the assumption our robot has been initialized in a front facing North orientation. Due to the uniqueness that this specific gateway serves, our robot will be able to initialize its current location and a single observation of the surrounding gateway. Such is the case that where there is only one potential seed that will serve as a solution for our tree. And lastly, with credit to using a queue-based structure, should a sensor reading corrupt our tree, preventing immediate localization using the encumbered data, our algorithm will fail gracefully and it's temporarily prolonged state of confusion. In the case where all five branches have been grown and a single seed solution has not been found, a robot will continue to map and record its environment by replacing its oldest seed with the second oldest. In a given worst-case scenario, should a corrupted sensor reading occur during the construction of our fifth branch, only five more correctly correct recorded gateways would be necessary to repair our trees integrity, and thereby find a singular seed solution. With the inclusion of omni-orientational localization, a similar graceful recovery from orientation confusion would also be possible should an error in rotational optometry occur when traversing turns.

The particular details of this algorithm can be described as follows: upon every reiteration our localization subroutine we sequentially search every cell for a bit by bit similarity with the gateway of our current root seed. Should the potential seed be found, the seed is investigated and scrutinized by reconstructing our tree on top of the cell. This separate function applies the tree to the given cell within the map by reconstructing the past using what the robot has observed with the starting origin transposed onto the seed in question. During the path reconstruction, should a branch become invalid by either of two ways; the first a discontinuity and observed gateway and the gateway of the map result in a disagreement, or second should the recorded movements result in an out of bounds error, thereby inferring that the robot would have had to venture outside of the discrete representation of the map. Should the potential seed past the scrutinization of the current tree, the numeric seed index is then incremented. After all potential seeds have been investigated, and that the seed index has retained a value of one, we are thereby localized. Using the trick of using the global variables as local placeholders within the branch calculations, after a successful localization iteration, these global variables are left assigned the last location as dictated by the tree using the single found seed, thereby already reconstructing our current location without further computation.

In order to achieve omni-orientational localization, we merely expand our search by either using additional rotated versions of our current map, or simply rotating only the orientation values within our own tree. By rotating our entire tree and searching again for potential seeds using our root seed as an ideal template, we can abstract our state of confusion to assume a state of successful localization upon the event when we have achieved only one remaining seed among all the possible orientations. When we are found to be left with one remaining seed, we simply use that orientation offset that the solution is derived from to accommodate and correct our orientation with respect to the northern reference of the map. This does however could be regarded as a potential con in the regard that it does require additional computations and the possibility of even further localization attempts on evenly the most geographically unique gateways should any duplicates of such exist. If initial orientation is unknown, even the T junctions as identified before will themselves will seem identical due to their rotational symmetry, and will not remain distinguishable when using the previously exampled tree of a single branch.

## Navigation

Once the robot has been localized and its current cell location and orientation are then specified, our robots next achievement will be to autonomously navigate from its current waypoint to a predetermined goal. This goal could be specified on-the-fly by using our pre-existing function that calculates the deterministic metric map and cost values for all cells within the map given any arbitrary goal point. Once the metric cost map has been calculated I navigation algorithm proceeds to explore a path of least resistance. As opposed to our previous implementation of metric locomotion, specifically in regards to generating a pre-deterministic path and relying on its and fallible execution, our robot instead follows its nose, or makes navigational base decisions on a continuous point by point basis.

The metric cost map as a method for the robot to deterministically move from one cell to another in a pattern that advances the robot towards this goal and ultimately results the arrival to its destination. There are several methods for generating cost maps, one would include assigning each cell the value of the calculated Euclidean distance to the goal cell. However this method usually results in gravity wells, and instance where the robot is geographically close to its goal point, but due to an obstacle or barrier, achieving that goal would require in moving in a direction that would momentarily result a further displacement to the goal. Such predicaments have been solved by developing a more elegant approach in path planning, such within the implementation of an A\* algorithm. Based upon your perspective of the initial conditions, there can be several ways in implementing an A\*algorithm. In this specific case for the final competition, the robot is required to navigate back to its designated home, that is specified just prior to every match during the competition, after successfully finding and recording the location of the simulated fire. Whether the robot calculates its path using its current location after identifying the fire or calculates a path to the goal starting from any navigable region is merely a choice of preference, as either method would suffice in achieving useful end result. In our case of a timed competition, we've chosen to use the pre-deterministic approach to shave time after the time has been started. Although the benefits in time savings due to our clock frequency and computation time are marginal at best, this helps reduce the number of points of failure should ever our robot attempt to plan the path starting from the nonvalid coordinate or cell index.

In order to implement a pre-deterministic path planning approach, thereby rendering a cost map that will serve as a path solution to the goal starting from any other location, we need to utilize a fundamental technique extended in any traditional A\*algorithm; recursion. By developing a recursive algorithm given only a starting index and current accumulated distance, then uses those parameters to assign values to the global cost map and recursively call itself on neighboring cells. In practice, we simply began by calling a recursive function upon the cold location as well as an accumulated distance of one. The recursive function will then assign they keenly the distance value it has received to the corresponding cell or index within the cost map given that additional starting index received. It will then do a for neighborhood search, first verifying that the neighboring cell is not out of bounds with respect to the street map, and second that the value of that cell with regard to the cost map has not already been assigned. If the neighboring cell in question passes both cases, then the recursive function as again called upon that specific location. There's one additional step that we must include within our recursive function, and that is to include the case of obstacles. In reality the very first thing our recursive function performs is to observe the Gateway of it starting location using a predefined map. Because the recursive function has been called the first place, we can already assumed that the cell location of the Gateway we wish to observe is a valid index and not out of bounds. Before deciding whether to sign the corresponding cell within the cost map the current accumulated distance we were given, we check to see if the relevant Gateway we are located in is bounded by all walls. If the Gateway we observe is completely enclosed, then that location is treated as an obstacle in us assigned a uniquely high cost value of 99, and that no instances of recursive call are placed are placed upon the objects neighboring cells. The second action is in order to prevent our path planner traversing through obstacles and designating deceitful paths.

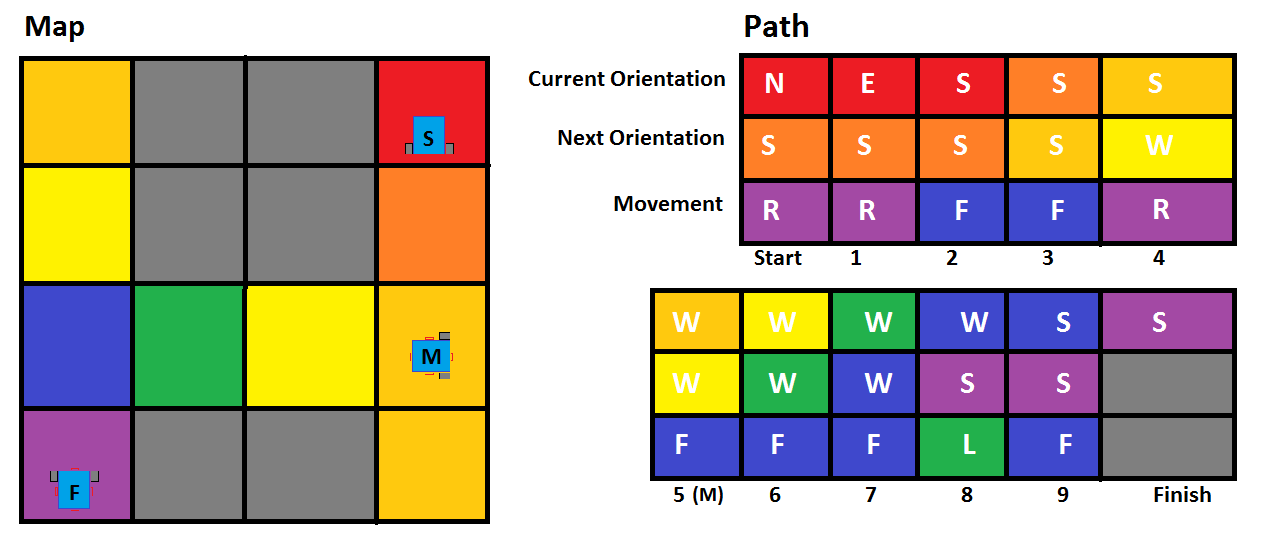
There are a few assumptions one must carefully consider when implementing such a method; the first is that the uniquely high cost value you assign to objects, given that the objects are completely non-traversable, is that the cost value should be greater than the value of the largest or longest conceivable accumulated distance within the map. This would prevent the robot attempting to traverse through a non-navigate able obstacle rather than proceeding into the next cell of a substantially lengthy path. The second assumption is that by discriminating only completely enclosed regions as obstacles, we neglect to observe the path between two navigable regions separated by a single wall as non-navigate able. In future or a more extensive implementations, it would be wise to have the algorithm account for the existence of walls separating boundary regions. This however would also call for our simple maze algorithm by which we navigate from cell to cell to also recognize this environmental trade when attempting to traverse a path of least resistance leading from a cell from a given cost value to that of one containing a lower value. An additional reason this was not implemented was out of the lack of necessity; given the environment within the competition and that every traversable area within the map was only segmented by completely enclosed obstacles and not by single wall barriers. This allowed us to assume a path planning structure complexly to that comprehendible to a basic occupancy grid, while simultaneously maintaining a descriptive feature method in order to record gateways in which were soaked beneficial for our prior localization algorithm. 

Figure : Cost-based Navigation - A rendition of the robots navigational movements from it starting location to its final goal as superimposed on top of a heat map representing the cost value within occupancy grid.

Within Figure 2 is an example of the robot traversing from a starting point in the top right corner to its end goal location at the bottom left of the map. As the robot begins already localized within this example, the robot is able to immediately proceed to its goal location using the navigational maze method as proudly described within the localization section. Along with the visualized occupancy grid, a table representing the robots attributes during each step in its path are also visualized with respect to its iterative process of achieving it subgoal or the neighboring cell of the lowest cost.

# RESULTS

WRITE HERE RUFFIN

# CONCLUSION AND RECOMMENDATIONS

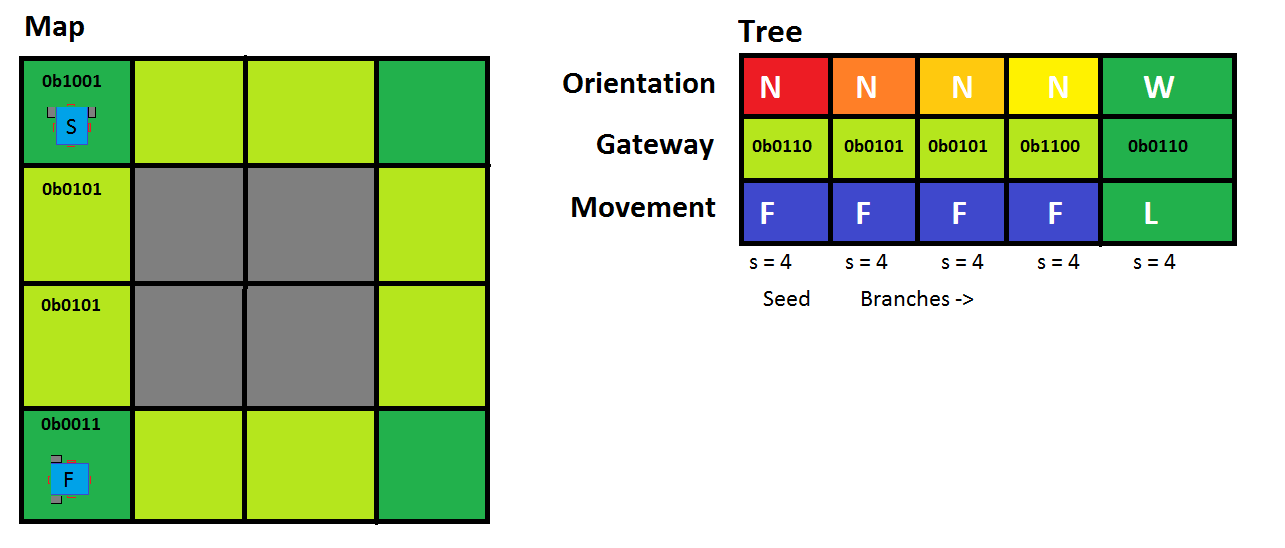


Figure : Localization Process – A case scenario where the world is rotationally symmetric and whereby assuming the starting orientation to be unknown or arbitrary will render it impossible to completely localize due to the complete symmetry.

Within Figure 3 is a specifically troubling scenario where the navigable regions within the environment are in total rotationally symmetric. Given the localization methods described within the prior sections and the specific techniques of using a queue-based structure and deterministically solving for one's location; here is a specific counterexample that renders what we have discussed quite helpless in the efforts of localization when one's initialized orientation is unknown. This map along with any other rotationally symmetric environments, among which there are many given a 5x5 occupancy (such as the navigational believe inverse of the map represented within Figure 3) is a suitable incentive for introducing additional environmental sensors. Such a sensor could include a digital compass or magnetometer in which to read current orientation utilizing Earth's magnetic fields in order to localize in such symmetric or featureless environments.

WRITE HERE RUFFIN