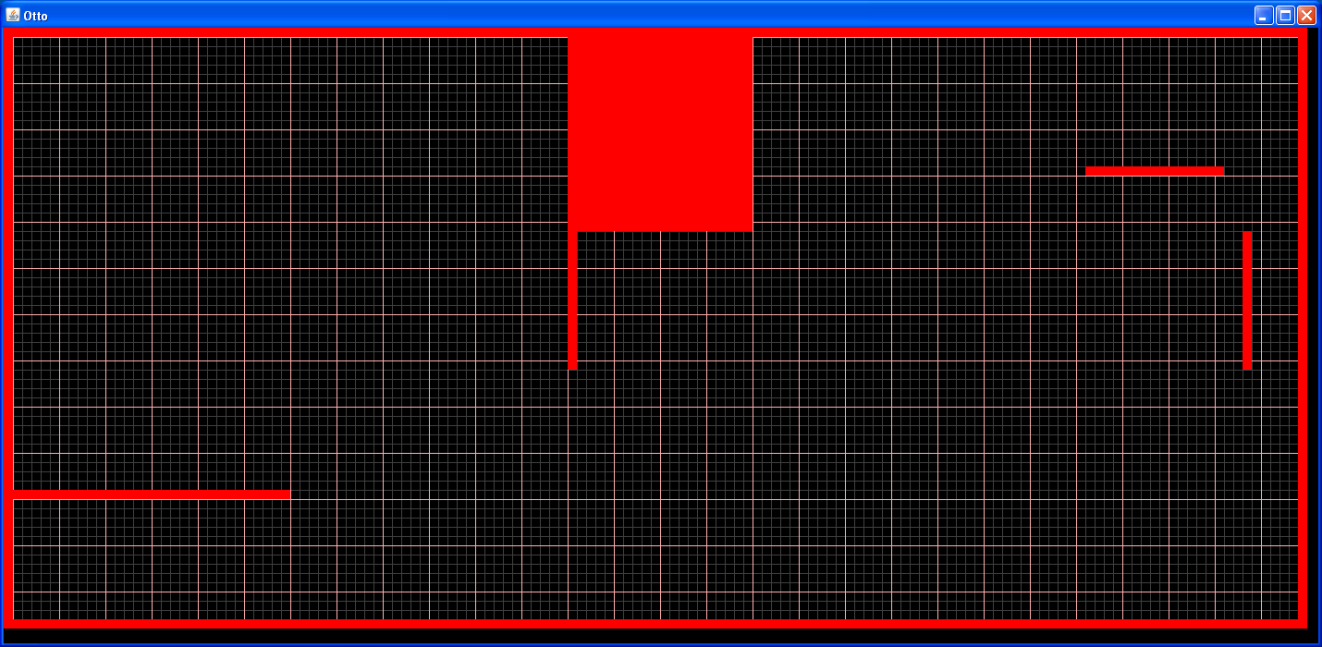
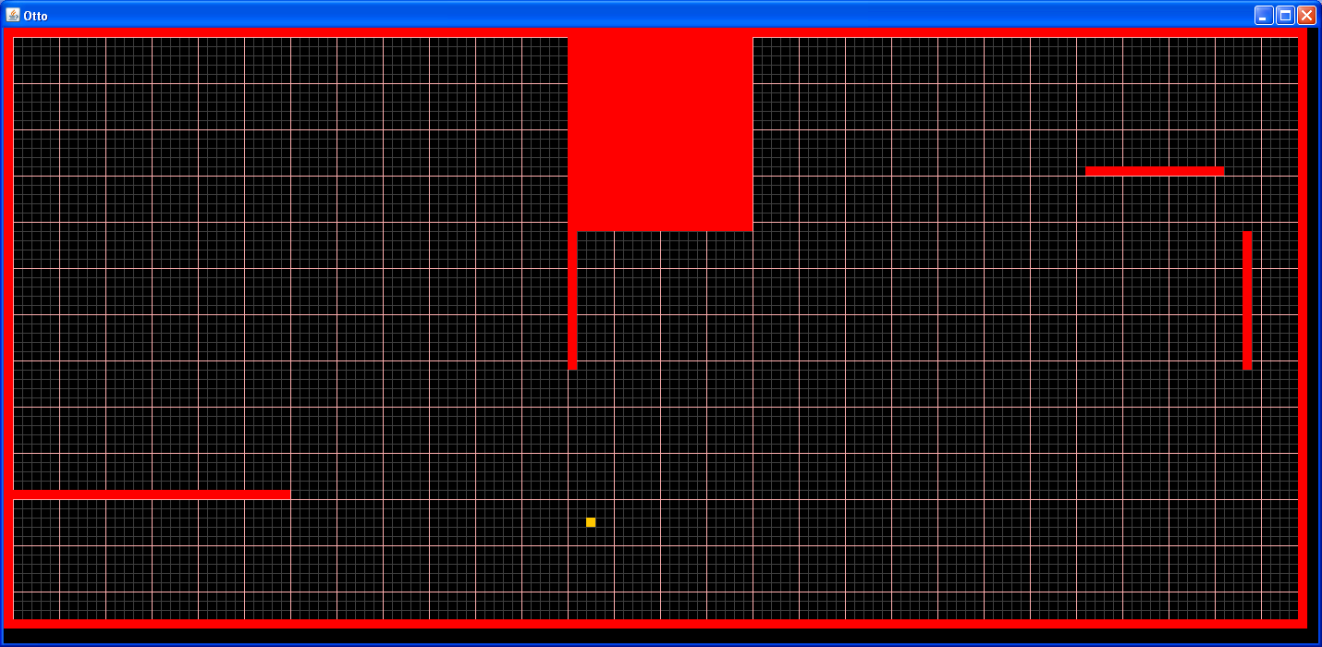
* 1. Programming Monte Carlo Localization
     1. Theory

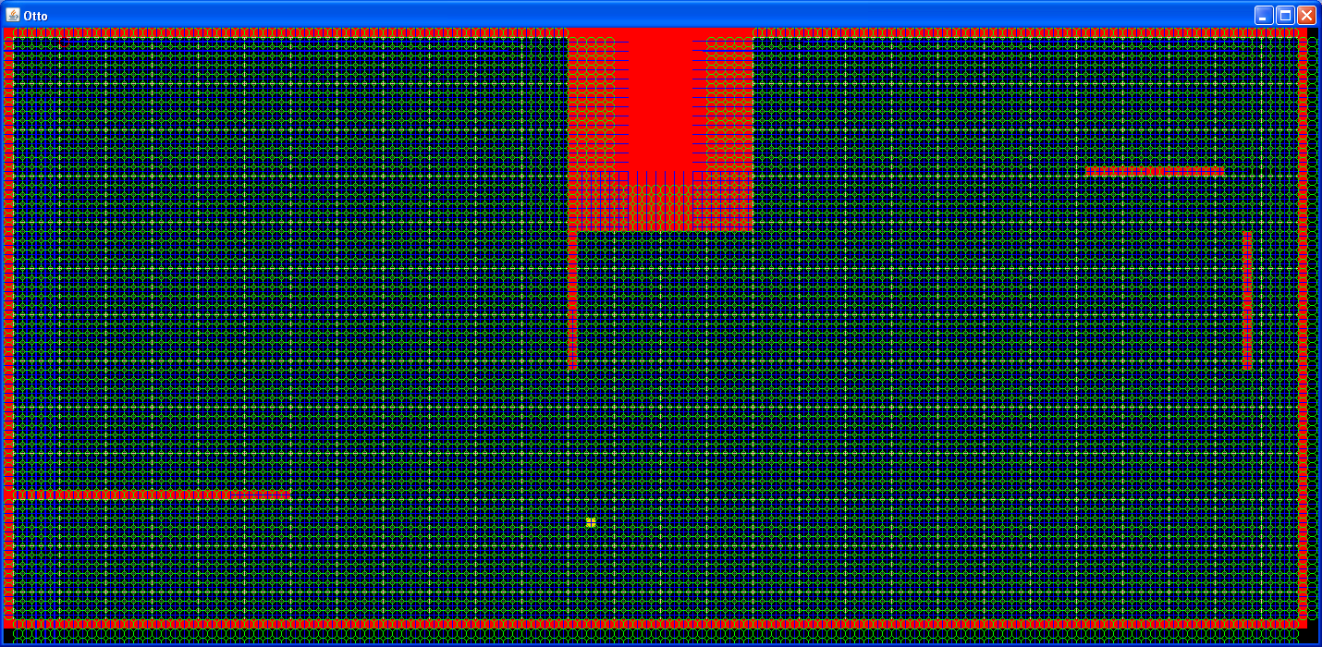
To begin Monte Carlo Localization (MCL) the processing computer must generate a large number of theoretically random possible poses, consisting of location and heading, within a known boundary environment which represent the possible locations that the robot may occupy. These can be thought of as the computer’s random guesses at where the robot might be within the known boundary. In actuality only one of these poses represents the actual location of the robot but at the beginning each pose generated by the computer has an equal chance of being the correct location. In order to eliminate certain poses the robot must move to a new location, update its current pose guesses and then resample the poses in a manner which keeps only those poses which have the highest possibility of being the correct location. This possibility of being the correct location is known as the pose’s weight. This weight is found by monitoring the difference between a sonar reading from the robot to the nearest obstructions and the pose property that represents this reading for each pose. The closer the pose reading to the nearest obstruction is to the robots reading the higher the weight of that pose and the more likely it is the correct guess at the robots location. Repeating this resampling process over the course of several random moves applied to the robot and the poses will eventually converge all of the poses into a single location that, based on the pose weight, has the best chance of representing the location of the robot.



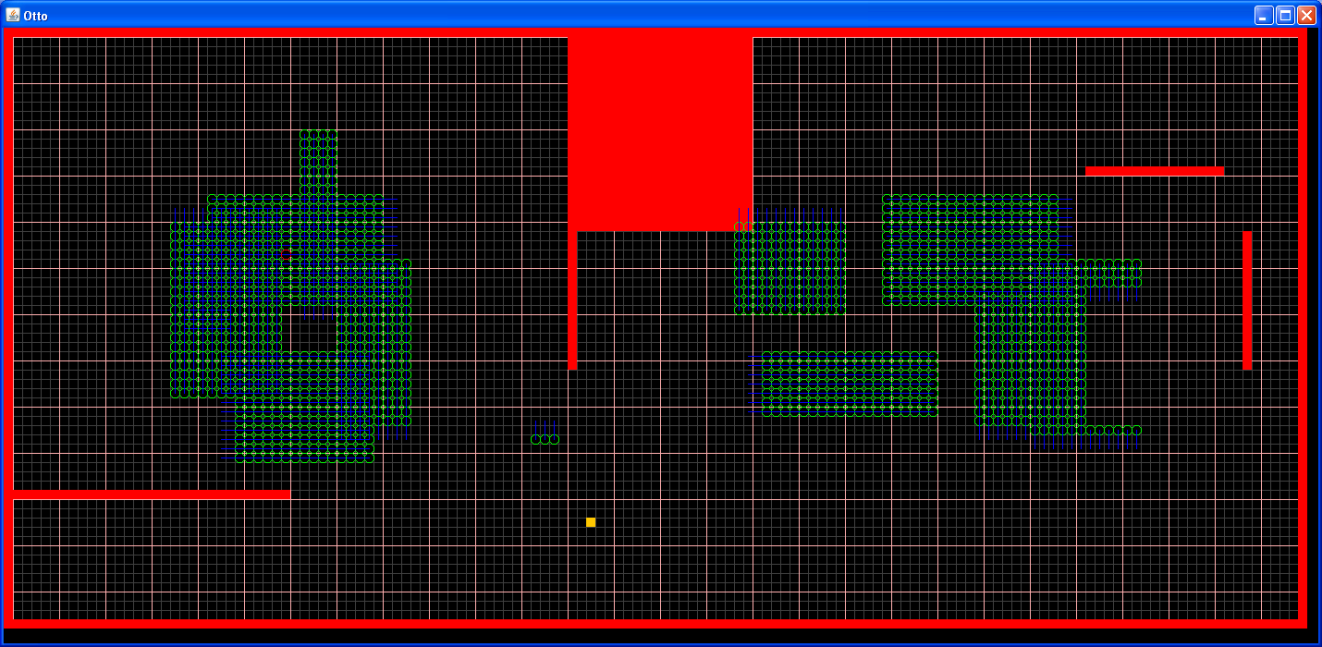
Initial Boundary



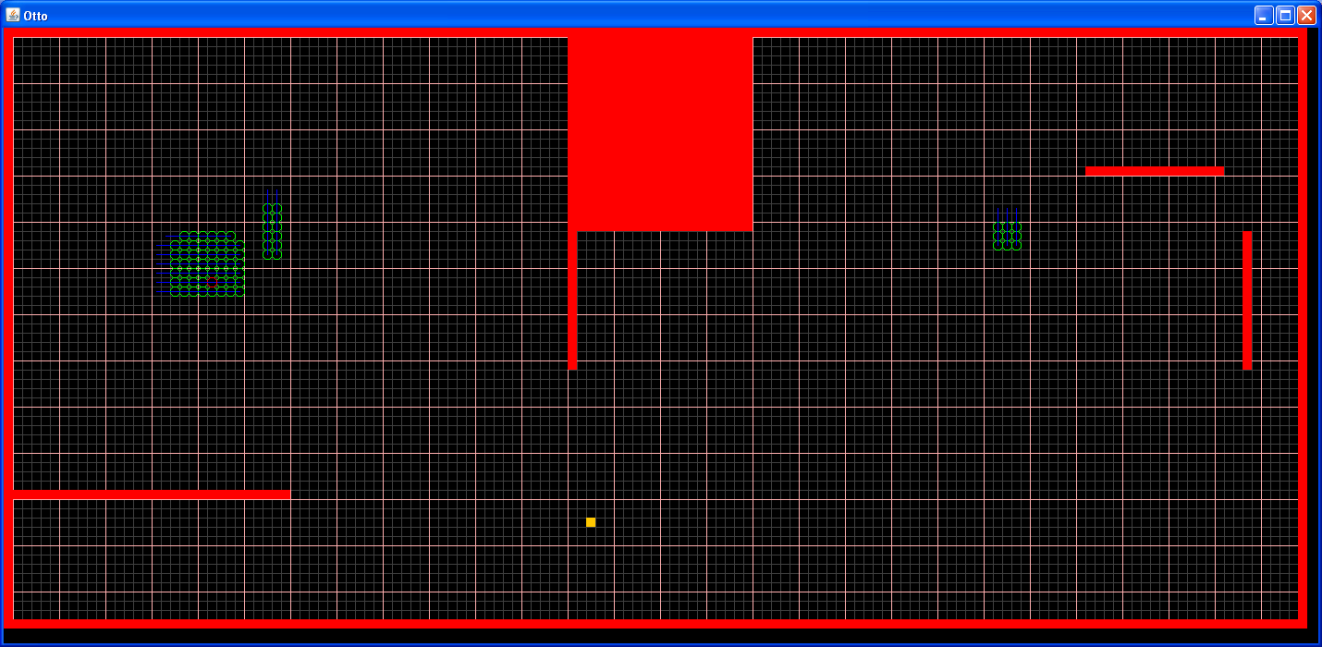
Initial Boundary w/ End Point Selected



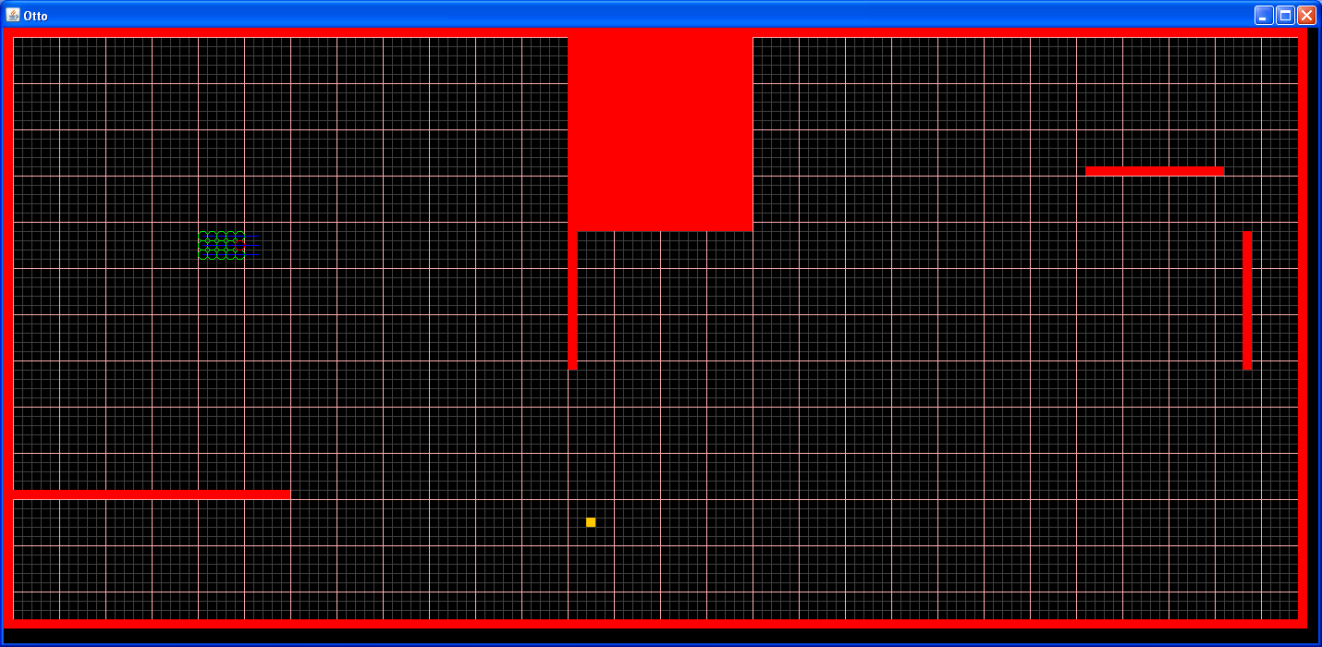
Initial Particle Population



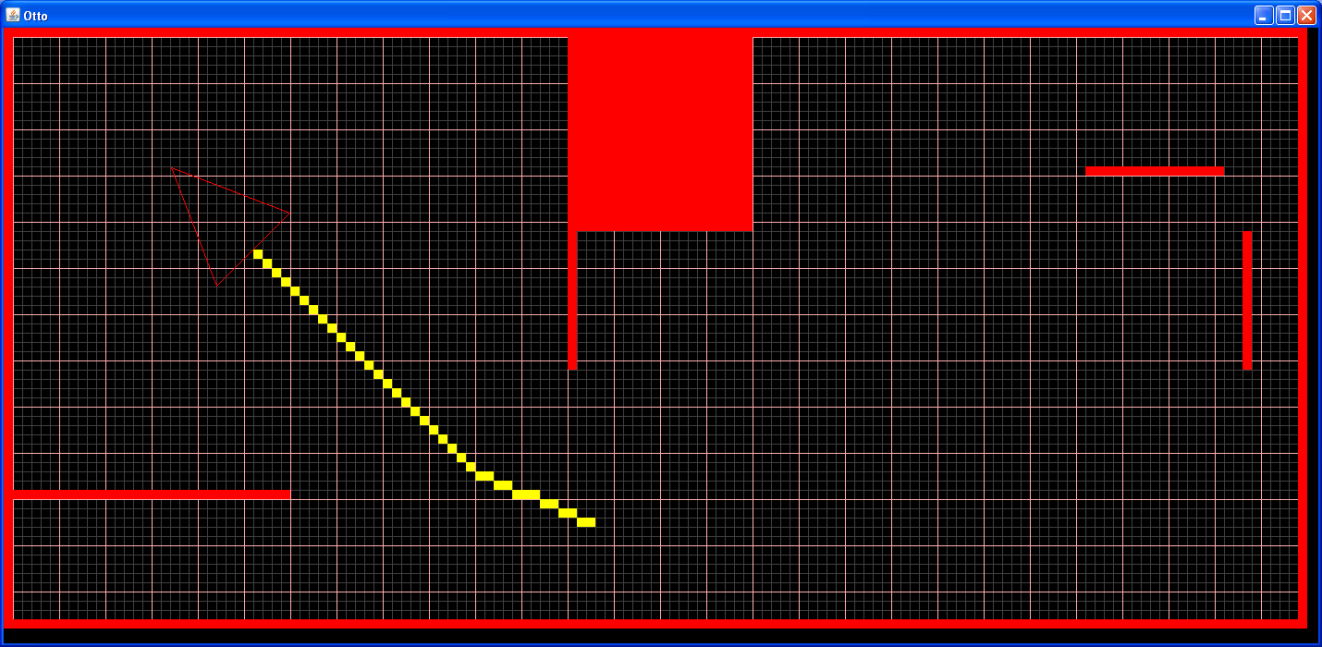
Converging Particles Resample 1



Converging Particles Resample 2



Converging Particles Resample 3



MCL Localization Completed

* + 1. Programming MCL

Programming the Lego NXJ robot very closely followed the MCL theory with a few minor customizations to optimize the process for this specific project. All computation was completed on a laptop PC while the robot simply completed the moves sent to it from the PC code and sent back the sonar readings to the nearest obstruction in the known environment. The PC and robot communicated over a Bluetooth connection with a simple communication protocol uploaded to the robots processing unit (NXJ brick).

The PC program followed a series of simple steps that will be outlined below and are broken down into more detail in their corresponding sections. These steps included:

1. Generate Particles: Generating the pose set to represent the computer’s guess at the possible locations of the robot.
2. Apply Move: Applying a move to the robot and to the poses in order to update the guesses of the computer at the robot’s location.
3. Take Reading: Taking a sonar reading from the robot to be used in the weight calculation for the poses.
4. Calculate Weight: Comparing the sonar reading from the robot to the known reading assigned to each poses and calculating the possibility that this pose is the correct guess at the robots location by analyzing the difference between these readings.
5. Resample: Updating the pose sets through a resampling process which checks if a given poses has a weight calculation higher than a random number in a specified range and if true, creating a copy of this pose so that its chances of being the correct robot location increase in the next sample.
   * + 1. Generate Particles

The process of standard Monte Carlo Localization (MCL) begins with the computer generating a random set of poses consisting of location and heading to represent the possible locations that the robot may occupy. To customize the project to limitations of the hardware a special pose class was created in Java with specific methods defined for interacting with each pose as well as retrieving information. For the purposes of this project the MCL code required that four poses be created in each cell defined within the environment. These four poses had the same x and y coordinates, however they each had a unique heading in one of the four cardinal directions (East: 0°, North: 90°, West: 180°, South: 270°). Note that this is greatly simplified from standard MCL as the headings of each pose are limited to only four different directions. This was due to the fact that the Lego Mindstorm sonar sensor was only capable of taking readings at locations perpendicular to the environment borders which for this project were all square. The poses were also defined by four integers in each of the cardinal directions representing the distances from each pose to the nearest obstruction. The code defining this process can be found in the Appendix section in the otto.java class file under the method “generateParticles()”.

* + - 1. Apply Move

After the pose sets have been created through the particle generation method the MCL algorithm requires that the robot make a series of moves in order to begin the process of converging the poses. For the purposes of this project the move called is a 90° rotation to the right and then if possible a forward move that covers a random integer distance between 1 and 6 centimeters. After each rotation the robot will go through the other steps of taking a reading, calculating the weight of each pose and then resampling the pose set as described in the section below. After the reading to the next obstruction is taken, however, the robot decides whether it has enough room to make a forward move. This is decided by checking to make sure the nearest obstruction is at least 20 centimeters ahead of the robot. If this condition is true then the robot knows it has enough room to make a forward move and then turn again. The purpose of making forward moves in between turns is to help the poses converge quicker by creating greater variance between the robot reading and the pose reading. This process of changing the move type also helps to eliminate local symmetry within the environment.

Applying the move to the robot involves selecting a number between 1 and 4 according to the move protocol defined by the Bluetooth communication file that is uploaded to the robots computing unit (1: Forward, 2: Backward, 3: Left, 4: Right). It is also important to note that the move that is applied to the robot must also be applied to each pose that is contained in the pose set. Applying this move to the poses means not only updating the x and y coordinates based on the pose heading but also updating the integers in the pose representing the distances to the nearest obstruction in each of the four cardinal directions. Updating these is also based on the heading of the poses which is itself changed with each turn move completed. The code for applying moves to the poses and robots can be found in the “applyMove()” method in the otto.java file as well as the “applyMove()” method in the Pose.java class in the Appendix section of this paper.

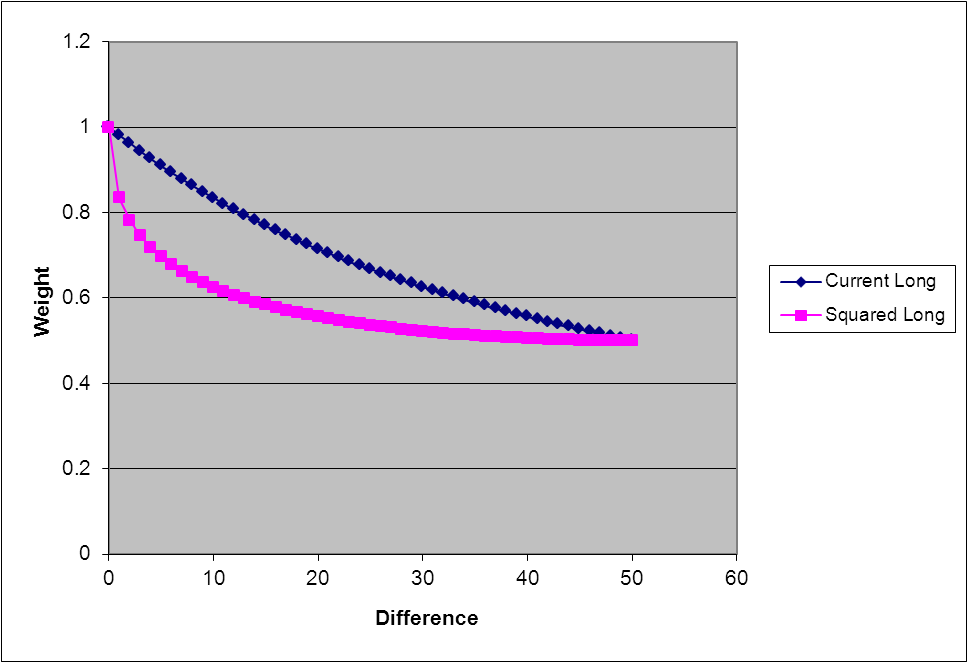
* + - 1. Take Reading

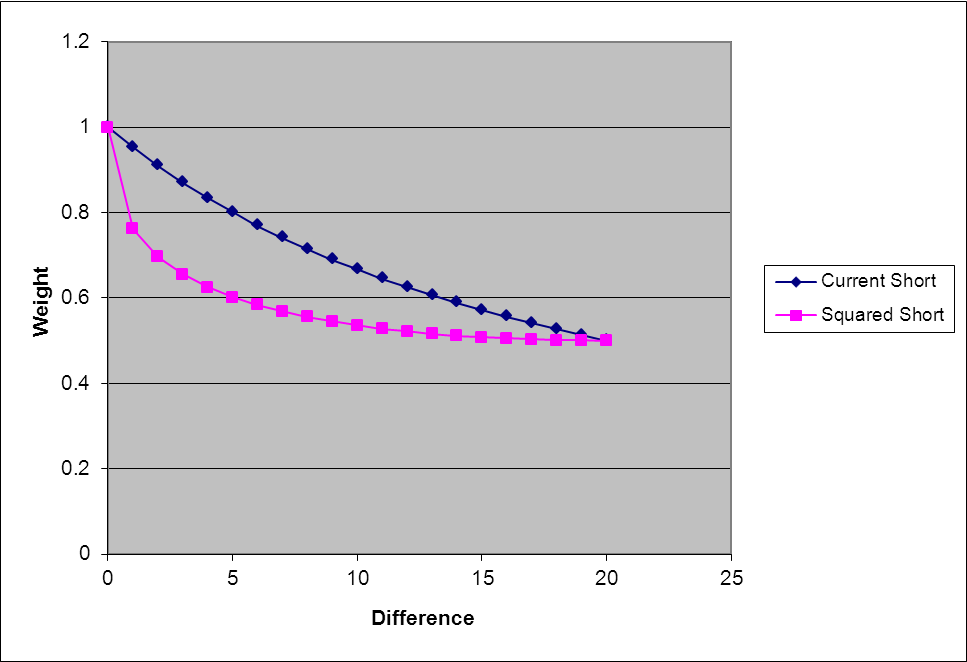
The take reading method is used for the sole purpose of telling the robot to take a sonar reading to the closest obstruction. It is accomplished by using the communication protocol previously discussed where the integer 5 is sent to the robot and tells the sonar sensor to take the reading and the data out stream flushes the reading from the robot to the PC. This method must also be paired with a while loop in which the reading must be less than 255 to break the loop and continue with the MCL algorithm. The method for taking the reading can be found in the “takeReading()” section of the otto.java file with its implementation being in the “solve()” method of the otto.java file.

* + - 1. Calculate Weight

The calculate weight method is used to determine if a given pose is a good guess at the actual position of the robot. This weight calculation begins with a difference squares difference between the reading that the robot takes from the sonar sensor and the particle reading of the pose as a function of its heading as defined by the equation . This difference is then used in the equation to normalize the value such that the weight will always be a number between 0 and 1 with the smallest differences yielding the highest weights.

Using a difference square difference method was shown to be superior to simply using a percentage difference method defined by the equation . This is due to the fact that the squared difference method copies more poses with smaller difference and less poses with larger difference in an exponential fashion while the percentage difference method is mostly linear in its selection of poses. This can be seen in the excel plots below where the Current line represents a linear selection and the Squared line represents the exponential selection. Note that this test was done for selection of higher weighted poses for both long and short differences in distance.





Another advantage to using this method is that is helps to compensate for compounding physical movement errors accrued in the process of having the robot make multiple moves before convergence. One important realization in the project was the idea to use modular math in reference to the angles when selecting the particle reading to be used in comparison with the robot reading. This is because the angle or heading of the robot and thus the particles may exceed 360 degrees depending on how many turns it takes to converge the pose set. Using modular math with respect to the heading will always keep this heading between -360 degrees and 360 degrees which can be handled more easily by the MCL computations. The specific code used for this section can be found in the “calculateWeight()” method of the Pose.java class found in the Appendix of this paper.

* + - 1. Resample

The resample method in the MCL algorithm uses the weights of each pose to decide what particles from the original pose set will be copied into the new pose set that will define the next round of moves and resampling. The logic of this resampling is as follows:

* The original pose set is copied into a new temporary pose set that will be used as a reference to copy poses and is then cleared
* A while looped is created and is set to break only when the new pose set that will be created is of the same size as the original pose set.
* A random number between 0.5 and 1 is selected and then used as the lower limit of the weight filter used to screen each particle in the temporary set.
* If the particle being analyzed has a weight that is greater than the random number a copy of this particle is placed into the new set and the loop continues to the next particle until the size of the new pose set if the same as the original pose set.

This method causes multiple instances of high weight poses to be created in a pose set of a constant size while the poses with too low of a weight based on their particle readings compared to the robot reading will be eliminated from the pose set representing the guesses at the robots location. The code for this section can be found in the “resample()” method of the otto.java file in the Appendix section.

Note that this entire process for converging the location guesses is run until the number of poses that are the same is at least 25% of the total pose set size. This number was determined after multiple trials of MCL where the solution was reduced down to a reasonable cluster but failed to converge to one specific location due to low weights as a result of errors in the robots physical movements. 25% of same particles is a high enough factor of the total pose set size to constitute a majority of particles.

Some of the limitations of using this method are that the more moves needed to reduce the possible locations of the poses, the more error that will be introduced into the readings based on physical errors involved in the robots movement. For this reason the convergence criteria can be altered to optimize the solution to limit the number of moves needed to reduce the number of possible locations and thereby increase the number of same poses in the pose set.