* 1. Programming the Robot

The Lego Mindstorm came with custom preinstalled software. This software did not meet our needs. It used a block GUI to program the brick and was not robust enough for Monte Carlo Localization. After some research, a Java operating system, Lejos, was found that could be installed on the brick. This system erased what the brick previously had and installed a version of Java. A regular Java programming suite could now be used to program the brick. It is important to note that no one in this group is a computer science major. The group is made up of mechanical engineers and learning how to program at this level was its own challenge.

The first step was connecting to the brick using a Bluetooth connection. The only options were a Bluetooth connection or a USB connection. Using USB would mean the brick would always need to be connected to the laptop. This was a limitation that was not acceptable. It took trial and error to try and open the ports needed to send integers to and from the brick. The key was to get the computer and brick to do the opposite of each other. If one was sending data, the other had to be listening for data. The table below shows what calls the computer would make that what calls the brick would make it response. The segment below would ask the brick to use its sonar and record the distance it reads. The brick then returns this value to the laptop and the laptop then prints it out in the form of ‘temp.’ By passing numbers back and forth, the two computers can be regulated and not override each other.

|  |  |
| --- | --- |
| Laptop | Brick |
| *conn* = **new** NXTConnector();  *dos* = *conn*.getDataOut();  *dis* = *conn*.getDataIn();  *dos*.writeInt(5);  *dos*.flush();  **int** temp = *dis*.readInt();  System.*out*.println(temp); | *btc* = Bluetooth.*waitForConnection*(); *dis* = *btc*.openDataInputStream();  *dos* = *btc*.openDataOutputStream();  n = *dis*.readInt();  **int** temp = *uss*.getDistance();  *dos*.writeInt(temp);  *dos*.flush(); |

After the laptop was communicating with the brick, the laptop had to create a grid that represented the environment. While the computers may not need this display, the humans running the program needed it so they could understand what the computers were doing. The grid would be identical to the physical course that was constructed. On the screen, 5 pixels is the equivalent of 1 centimeter. In the screen shot below, each small square represents 1 centimeter. The larger squares are 5 centimeters on each side and match the physical course. Any red square represents a wall. The user can easily click a cell to change it into a wall. This screen is interactive.

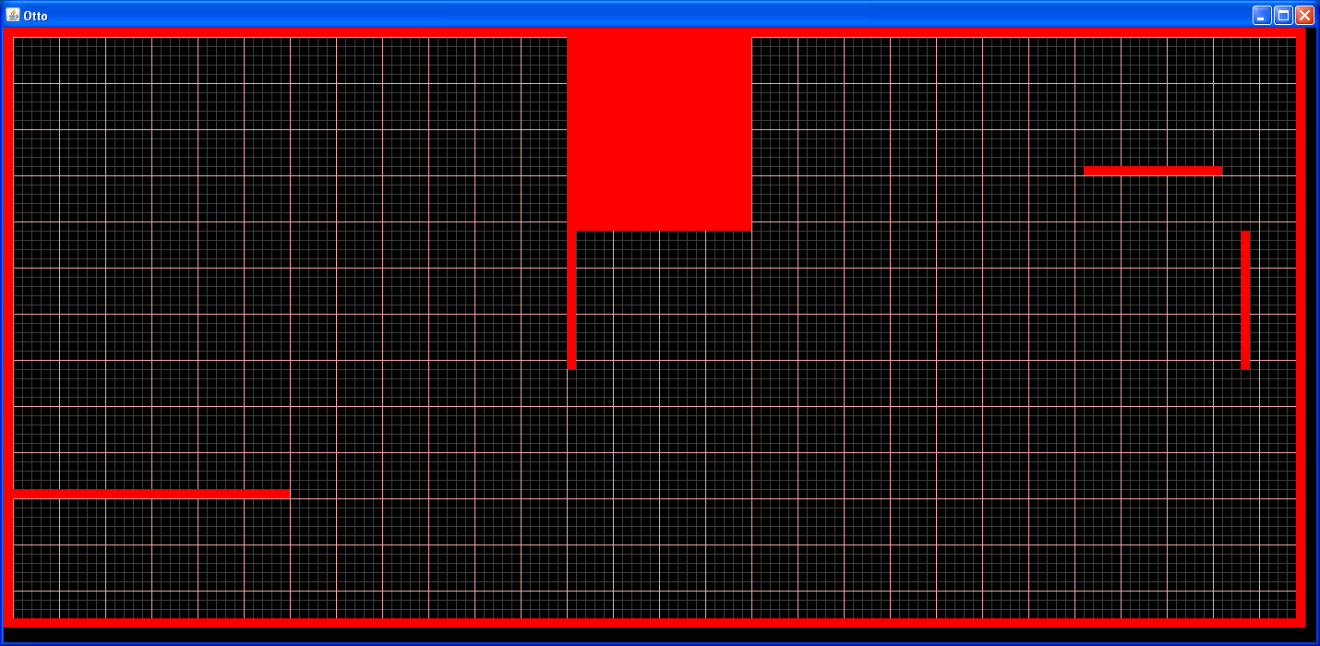


Figure - Screen shot of user interface.

With the laptop now showing an interface and also being able to communicate with the robot, two major obstacles still existed. The first was localizing itself so it knew where it was. The second was getting it to travel the course once it knew where it was. These two obstacles were divided up among two people on the team.

* 1. A\* Navigation

A\* navigation was chosen to navigate the robot once it knew where it was. This algorithm was first used in 1968 but was based on a 1959 algorithm created by Edsger Dijkstra (wiki A\*). A\* uses heuristics to be efficient and effective. Heuristics is often defined as common sense or an educated guess. While it is easy for a human to use common sense, it can be much more difficult to program a robot to use common sense. Heuristics attempts to do this.

This process divides the environment into cells called nodes. Each node is then assigned three costs. The table below talks about each cost. Whichever node has the lowest cost is the best solution.

|  |  |  |
| --- | --- | --- |
| A\* Star Costs | | |
| Name | Symbol | Description |
| Exact Cost | G | Exact distance from the starting point |
| Heuristic Cost | H | Estimated distance from the end point |
| Total Cost | F | = G + H |

Gamegardens.com provided an excellent example of pseudo code describing how A\* is implemented. That portion of code is included below. From this outline, code written in Java and customized to our course was created and could solve how to get from one point to another. The code also took into account walls and how much room was needed for the robot to get by. This was added on our own.

create the open list of nodes, initially containing only our starting node

create the closed list of nodes, initially empty

while (we have not reached our goal) {

consider the best node in the open list (the node with the lowest f value)

if (this node is the goal) {

then we're done

}

else {

move the current node to the closed list and consider all of its neighbors

for (each neighbor) {

if (this neighbor is in the closed list and our current g value is lower) {

update the neighbor with the new, lower, g value

change the neighbor's parent to our current node

}

else if (this neighbor is in the open list and our current g value is lower) {

update the neighbor with the new, lower, g value

change the neighbor's parent to our current node

}

else this neighbor is not in either the open or closed list {

add the neighbor to the open list and set its g value

}

}

}

}

References

* [http://en.wikipedia.org/wiki/A\*\_search\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)
* <http://wiki.gamegardens.com/Path_Finding_Tutorial>
* <http://www.policyalmanac.org/games/aStarTutorial.htm>

Like most programming projects, trial and error was used with almost every step. For the Bluetooth connecting, the parameters had to be isolated and used step by step to ensure the team understood what was happening and where the code was breaking. It was discovered that the Bluetooth Stack that came on the laptop being used was not compatible so different software had to be used.

All of the components in the GUI are measured using pixels. The robot moved in millimeters. The team worked in centimeters. This created room for error and conversion mistakes. There were several times a number was in pixels when it needed to be in centimeters. Having the program display the values enabled us to see what was happening and to correct for these errors.

Monte Carlo had several errors with how it calculated its weights and found a percentage. Because of the scale of these errors, it was discussed in the Monte Carlo section of this paper.