ASSIGNMENT 2

Mapping an occupancy grid

Raphael Attfield
attfield@email.sc.edu

Overview
This report presents my approach and solution to implementing occupancy grid mapping in RO using C++. The program receives laser data from the robot and then updates the occupancy grid based on any detected obstacles. It uses my previously created random walk program to control the movement of the robot.

The Setup
I found the setup for this assignment to be much easier than for the previous one, mainly due to the fact that it was almost all the same as before, just with the additional step of having to also run the new program. Once I had started running the program and made sure it was set up correctly, I spent some time reading through the provided code and understanding it.

My approach
The laserCallback function:
This function is called every time new laser data is received. It starts by calculating the robots current position (‘robot_y’ and ‘robot_y’) based on its coordinates and the size of the occupancy grid canvas. It then marks the cells corresponding to the robots current position on the occupancy grid. It gets the number of scans (’num_rays’), it then calculates the angle increase ‘ray_angle_inc’ between each laser by using the field of view (‘scan_fov’) of the laser scanner. It then loops through each laser scan. For each scan it calculates the endpoint (‘ray_x’ and ‘ray_y’) aswell as the angle (‘ray_angle’).It calls the ‘rayTrace’ function I created, it then checks if the ray ended due to encountering an object or if the distance was too large.

The rayTrace function:
This function implements Bresenham’s line drawing algorithm, which is used for drawing straight lines between two points on a grid. It traces a line from one point to the next, while updating the canvas with information about free cells. It starts off be calculating dx and dy which aref the absolute difference between the x and y coordinates of the two points. It then infinitely loops until x1==x2 and y1==y2. For each iteration of the loop it checks if the current cell is within the boundaries of the occupamcy grid. If it is within the boundary and its value is ‘CELL_UNKNOWN’ it updates this to ‘CELL_FREE’.

Testing
To test the program I had two windows open, one was the canvas being created by the program and one was the stage, on which the robot was moving. Having both of them on screen at the same time enabled me to compare the two and see how well my program was mapping. Early on to test the mapping I decided to drag the robot into a large open space, this was to make sure that it was correctly identifying free and blocked areas. This is shown below in figure 1.

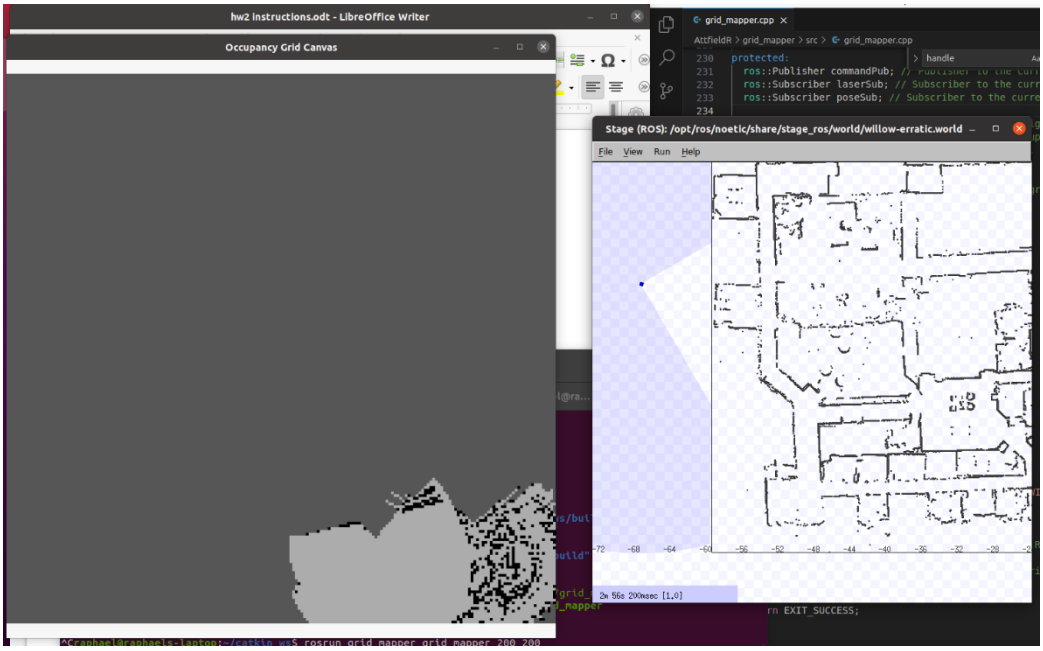

Figure 1

Problems with random walk

Some of the problems that I had in the previous assignment were also present in this one, that was due to this code, using my previously created random walk algorithm. Sometimes the robot would get stuck, or just explore the same sort of areas, and not really explore too much of the map. I fixed this issue by manually moving the robot around with my mouse to help create the full map.

The mapping

The mapping was quite successful. To create the full map I did have to manually move the robot around, but once I had done this I had successfully created a map of the world. My map isn't too accurate, and it doesn't always show the exact shape of objects, and it occasionally has some free space marked as occupied, but it creates a general map of the area very successfully.

Conclusion

Overall, I am quite happy with how my program worked in the end. The main issue I have with it is that it starts creating the map on the bottom right of the canvas rather than in the center. I tried to fix this issue however all my attempts just created more bugs in the mapping itself and I decided it was better to have a more accurate map that starts in the bottom corner, than a much less accurate map that is centered on the canvas.

Examples of the maps created:
The following examples are snapshots of the same map created over time.