Frans Fourie

Intelligent Autonomous Systems

Robot Localization and Mapping

Contents

Introduction 4
Problem Formulation 4
Technical Approach 4
Odometer path 4
Lidar Mapping 5
Generating the Occupancy Grid and Map 6
Particle Filter 7
Adding Noise 8
Correlation Calculation 9
Particle Resampling 9
Results 10
Part 1 Results 10
Part 2 Results 15
Part 3 Results 21
Discussion 25
Ghost Hallway removal 26
Conclusion 27

Table of Tables

Table 1: Constant path calculation parameters 4
Table 2: Path calculation formulas 5
Table 3: Lidar data conversion formulas 6
Table 4: Map hyperparameters 7
Table 5: Hyperparameter list 8
Table 6: Correlation calculation formulas 9

Table 7: Particle Resampling test formulas..... 9

Table of Figures

Figure 1: Part 1 Robot Path for file 20 11
Figure 2: Part 1 Robot Path for file 21 12
Figure 3: Part 1 Robot Path for file 22 13
Figure 4: Part 1 Robot Path for file 23 14
Figure 5: Part 1 Robot Path for file 24 15
Figure 6: Part 2 Occupancy Grid and Robot Path for file 20 16
Figure 7: Part 2 Occupancy Grid and Robot Path for file 21 17
Figure 8: Part 2 Occupancy Grid and Robot Path for file 22 18
Figure 9: Part 2 Occupancy Grid and Robot Path for file 23 19
Figure 10: Part 2 Occupancy Grid and Robot Path for file 24 20
Figure 11: Part 3 Occupancy Grid and Robot Path for file 20 21
Figure 12: Part 3 Occupancy Grid and Robot Path for file 21 22
Figure 13: Part 3 Occupancy Grid and Robot Path for file 22 23
Figure 14: Part 3 Occupancy Grid and Robot Path for file 23 24
Figure 15: Part 3 Occupancy Grid and Robot Path for file 24 25
Figure 16: part 2 file 23 has a ghost hallway 26
Figure 17: part 3 file 23 ghost hallway removed 26
Figure 18: part 2 file 21 has a ghost hallway 27
Figure 19: part 3 file 21 ghost hallway removed 27

Introduction

This report documents the mapping and localization of a robot given odometer, gyroscope and lidar data. The first step of this project is to trace the path the robot moved using only the odometer data. The second step is to create a 2D map of the environment in which the robot moved using the provided lidar data along the path generated from the odometer data. The final step is then to perform simultaneous localization and maximization of the robot using a particle filter. The goal of this step is to correct errors in the path obtained from only odometer data and then to obtain a more accurate lidar map of the environment using this new map.

Problem Formulation

Given odometer, gyroscope and lidar data use this data to implement a SLAM algorithm (Simultaneous localization and mapping). This should provide you with a path of the robot's movements as well as a map of the area in which the robot moved. The SLAM algorithm should improve the robot path leading to a more accurate environment map then the map obtained when using the raw odometer data for the robot path. The SLAM algorithm should also remove incorrect objects that are present in the original environment maps such as ghost hallways.

Technical Approach

Odometer path

In order to create a map of the robot's movements using odometer data the first task is to find a few parameter values. Firstly, the distance traveled every tick must be calculated, every tick of the odometer is an axle rotation thus the robot travels the length of the robot wheel circumference divided by 360. Secondly, the width of the robot is needed, or a derivative of the width is needed that compensates for slippage during turning by the robot. Next the number of tics traveled by the front and rear wheel on each side needs to be averaged. By then using the number of tics traveled by each side of the robot as well as the distance of each tick it is possible to calculate the distance moved by each side of the robot. This is an approximation given that wheel slippage occurs, and the robot can get stuck while the wheels are still turning. The below table shows the constant robot parameters used and how they were calculated.

Table 1: Constant path calculation parameters

Parameter	Formula	Value

Wheel diameter	Data Sheet	254
Tick Travel Distance	(pi*(Wheel diameter))/360	2.21657
Robot Width	Estimation	733

Next for every data point the direction in which the robot is moving needs to be calculated as well as the distance the robot traveled forward. Now by subtracting the distance traveled by the left side of the robot from the distance traveled by the right side and then dividing this by the estimated width of the robot we obtain the radian amount the robot turned during that data point. Then adding the robots right and left sides movement distances together and dividing this by 2 we get the distance moved by the robot. Now by multiplying this movement distance by sin and cosine of the previously calculated radian amount we get the amount moved in the y and x direction. The following table shows all the formulas used in calculating the path traveled using only the odometer data.

Table 2: Path calculation formulas

Parameter	**Formula**
(Right/Left) Wheel ticks	(Front + Back (Right/Left) wheel ticks)/2
(Right/Left) distance move	(Right/Left) Wheel ticks * Tick Travel Distance
Moved Distance	(Right + Left distance moved)/2
Theta	(Right - Left distance moved)/Robot Width
Movement in X direction	Moved Distance * cos (Theta)
Movement in Y direction	Moved Distance * sin (Theta)

By then repeating this for every data point and plotting all these data points on a map we have the path given by the odometer data.

Lidar Mapping

To create an environment map based on lidar data the first step is to match the lidar data to the odometer movement data. This is done by looking thru the odometer timestamps as well as the lidar data time stamps and finding the lidar data time stamp that best matches the odometer time stamp for each data point. The lidar data with that time stamp is then associated with that specific odometer data point.

Depending on which data set is longer one Lidar data point can be assigned to multiple odometer data points or some lidar data points aren't assigned to any odometer data points.

The Lidar data is in polar coordinate format with each data point having an angle in terms of the robot and then a distance at which an object was detected from the robot. To start using the Lidar data the data needs to be converted to cartesian coordinates this is done by first calculating the x and y positions of the data points using only the lidar angle so the x and y positions are in terms of the robot position and angle. Next the coordinates are rotated according to the robot angle at that data point to get the lidar coordinates in terms of only the robot position. Lastly the robot coordinates at that data point is added to all the lidar data points for that time stamp in order to make the lidar data global and not reliant on the robot. The following table contains the equations used to make the Lidar coordinates Global coordinates.

Table 3: Lidar data conversion formulas

Parameter	**Formula**
Lidar X-Cor in terms of robot	scan * cos (angle)
Lidar Y-Cor in terms of robot	scan * sin (angle)
Lidar X-Cor global angle	X-Cor * cos (robot theta) - Y-Cor * sin (robot theta)
Lidar Y-Cor global angle	X-Cor * sin (robot theta) + Y-Cor * cos (robot theta)
Global Lidar X-Cor	Lidar X-Cor + Robot X-Cor
Global Lidar Y-Cor	Lidar Y-Cor + Robot Y-Cor

Generating the Occupancy Grid and Map

All the data has been converted to meters. In order to now generate the occupancy, grid each robot position is taken with its corresponding lidar data and all the data points are multiplied by 10 this is in order to increase the resolution of the map. This gives the map higher accuracy while still allowing for large enough bins to group data seen as the same together. Now for every lidar data point associate with a specific robot location the data pair is sent to the Bresenham algorithm which returns the integer values of all the data points in a straight line between a lidar data point and the robot location. Now considering the lidar detected an object at a certain distance from the robot it means that in a straight line from the lidar to that point there are no obstacles. So, on the occupancy grid for every location returned by the Bresenham algorithm we indicate a miss and subtract 0.7 from the likelihood of an object being in that location. Except for the very last location returned by the algorithm since that is a hit indicating an object,

we add 0.9 to the likelihood of there being an object in that location. This is repeated for every lidar data point associated with every robot location.

In order to get more accurate results if a Lidar data hit is within 0.05m of the robot it is disregarded considering it is probably the robot detecting itself. Also, the certainty of an object existing or not existing was set to -20 and 20. Thus, all values below -20 was converted to -20 and all values above 20 where converted to 20. After the occupancy grid was completely generated all values below 0 where set to 0 as to represent no object or unknown. Since the bounds of the occupancy grid is thus [-20,20] we can interpolate all the values in the occupancy grid from this range to a range of [0, 255] and convert the occupancy grid to a gray scale map using the new values as the pixel intensities. This new gray scale map can then be displayed. Thus, a hit or object is indicated by white on the map and a miss or unknown is indicated by black. The following table lists the hyperparameters used for generating the occupancy grid and map.

Table 4: Map hyperparameters

Hyperparameter	**Value**
Grid discretization (cell width)	0.1m
Log-odds increase for a "hit"	0.5
Log-odds decrease for a "miss"	0.1
Min/Max cap for map log-odd	20
Minimum Lidar Range	0.05m

Particle Filter

The particle filter works by adding noise to the odometer data according to a set of parameters. A certain number of particles are then created each taking the position of a different one of these noise points at the start. By then checking which particle shows the best correlation to map we already have we can select a particle to use to further create the map. Then noise is added to each particle again and again the correlation is evaluated. This process is repeated until the particles fail the resample test and the particles are resampled. The new particles are then again evaluated according to correlation and noise is added. With the particle at each step with the highest weight being used to further generate the map. The following is an extensive list of all the hyperparameters used in the particle filter.

Table 5: Hyperparameter list

Hyperparameter	**Value**
Robot width parameter	733
Number of particles	60
Min effective number of particles before resampling	Particle Count/1.5 = 40
Standard deviation for theta (noise at each timestep)	0.004 rad
Standard deviation for movement (noise at each timestep)	0.0004 m
Grid discretization (cell width)	0.1m
Log-odds increase for a "hit"	0.5
Log-odds decrease for a "miss"	0.1
Min/Max cap for map log-odd	20
Minimum Lidar Range	0.05m

Adding Noise

The first step in implementing the Particle filter is to generate noise that can be used to disperse the particles. To ensure the noise encompasses all the possible locations and directions the robot could have truly been moving in on each data point noise is added to the movement as well as the orientation of the robot which will heavily impact lidar data which is reliant on the robot orientation. The start is to define the number of particles to use as well as the standard deviation for the noise generation for the orientation and the standard deviation for noise generation of the movement. Now when the odometer data gets loaded and processed to give the x and y coordinates of the robot as well as the angle the robot is heading noise will be added. Instead of getting three one dimensional arrays containing the x, y and theta data respectively the program gets 3 two dimensional arrays with column count equal to the number of particles chosen. The first column of each array will contain the unaltered data that was used for the path without the particle filter. Random normal distribution will be obtained with center at the original data points and standard deviation as defined at the start of the program. From these normal distributions the number of particles minus one will be taken, these values will then form the entries of the other columns of the arrays.

The lidar data will be loaded and processed as normal with the only difference being there will now be a set of lidar data for each particle that is based on that particles position and orientation.

Correlation Calculation

To calculate the correlation of each particle at each timestamp to that of the map generated up until that point we look at the lidar hits associated with it. Start by looking at the map at the locations where the particle says there are hits and seeing if there are hits in those locations according to the map. The next step is counting the amount of particle hits that correspond to map objects and adding these together. This gives each particle a count of hits with higher count meaning greater correlation. These counts are then all divided by the largest count in the set which gives the particle(s) with the highest correlation a count of 1 this value is then multiplied by the weight of the particle up until that point. Each particle starts with a weight of 1. The following is the formulas used to update the weights.

Table 6: Correlation calculation formulas

Parameter	**Formula**
Particle Correlation Count	If (particle hit = map hit): Particle Correlation Count += 1
Particle Correlation	Particle Correlation Count/highest (Particle Correlation Count)
Particle Weight	Particle Old Weight * Particle Correlation

Particle Resampling

Particle resampling is divided into two parts firstly the test to see when it is necessary to resample and secondly the actual resampling of particles. For the first part the test is done by summing all the weights then squaring the answer and dividing this by the square of all the weights summed together. This gives the number of effective particles we have left; this number is then compared to the number of particles divided by 1.5 and if the number of effective particles is smaller than the particles are resampled. For the first part the following formulas are used.

Table 7: Particle Resampling test formulas

Parameter	**Formula**
Number of effective Particles	Square(sum(weights))/sum(square(weights))
Resampling Threshold	Particle Count/1.5

For the second part if the number of effective particles drops below the threshold the particles need to be resampled. Resampling takes place by multiplying each particles weight by 10000 to get all the significant digits of the particle weights to be integer values. Adding all these integers together a large number line is created with each number corresponding to an old particle with the segments belonging to particles that had a high weight being larger than those that had small weights. Randomly selecting numbers from this line and seeing to which old particle the number corresponds and setting the new particle to the value of that old particle and doing this for the number of particles, the particles are resampled. The weights off all the new particles are also reset back to 1 so they have equal probability.

Results

The following are the results obtained from the project, the files 20, 21 and 23 where training data and the files 22 and 24 where testing data.

Part 1 Results

The following is the results obtained from part 1 of the project. It shows the robot path as determined by using only the odometer data.

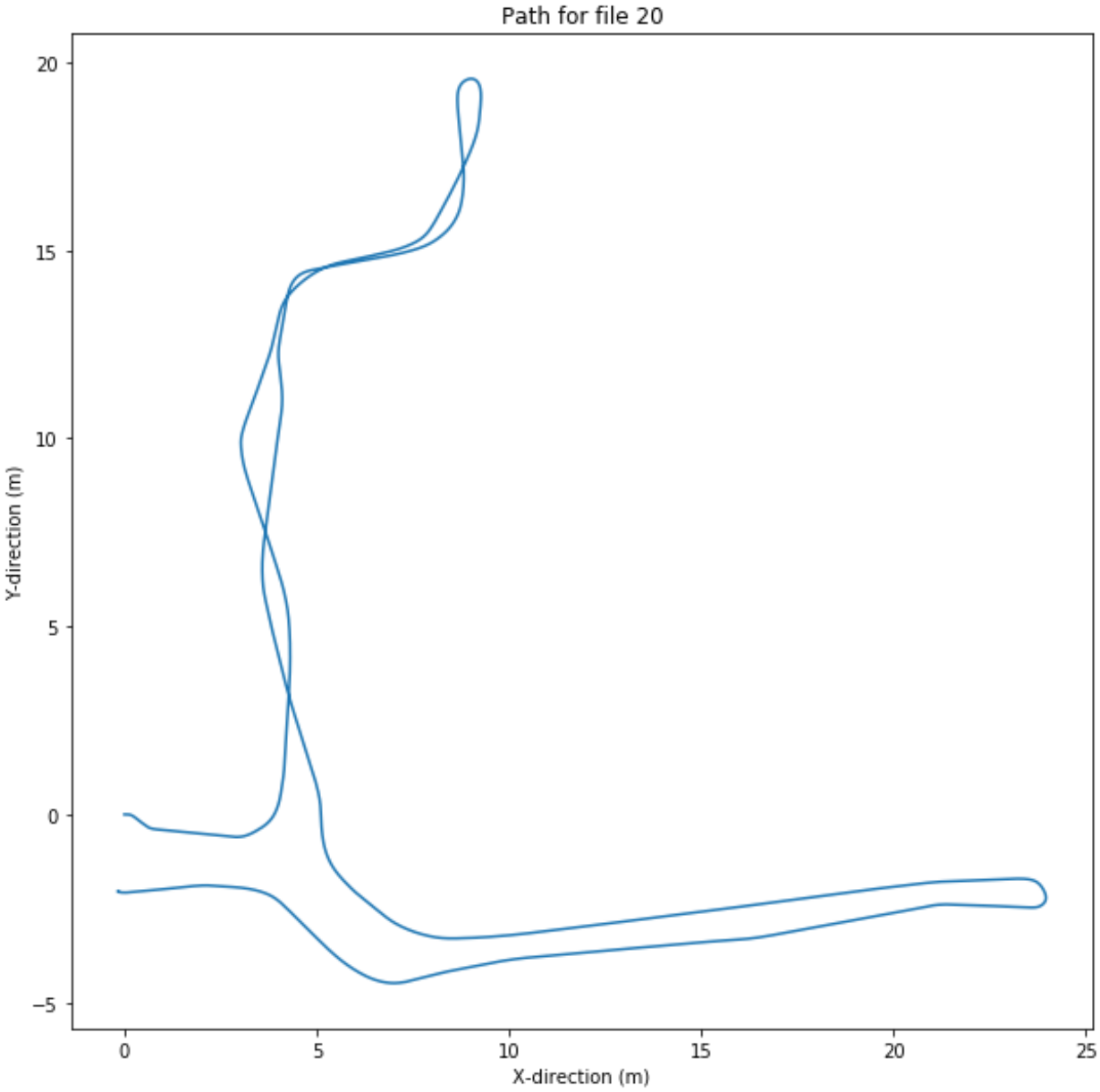

Figure 1: Part 1 Robot Path for file 20

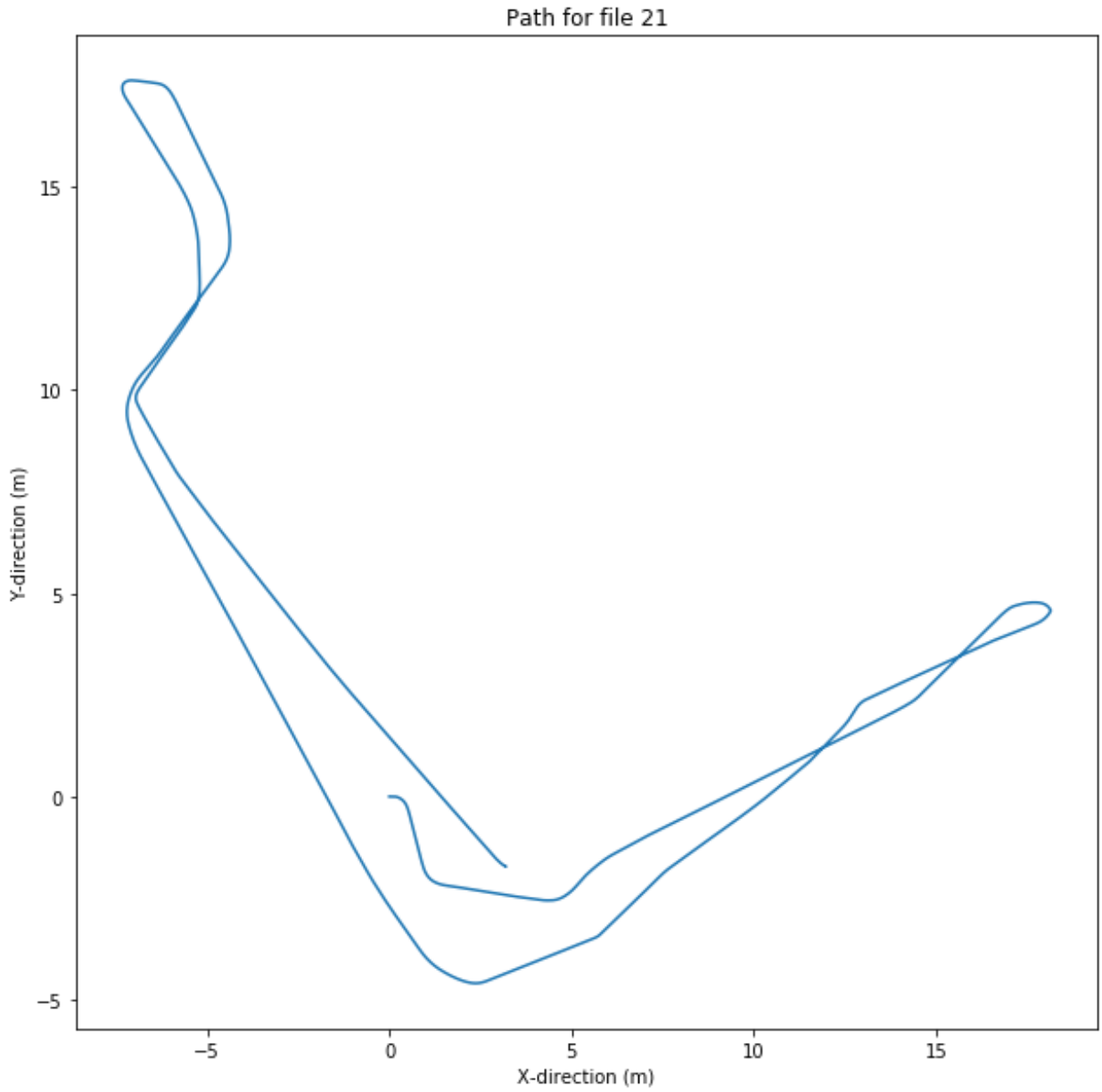

Figure 2: Part 1 Robot Path for file 21

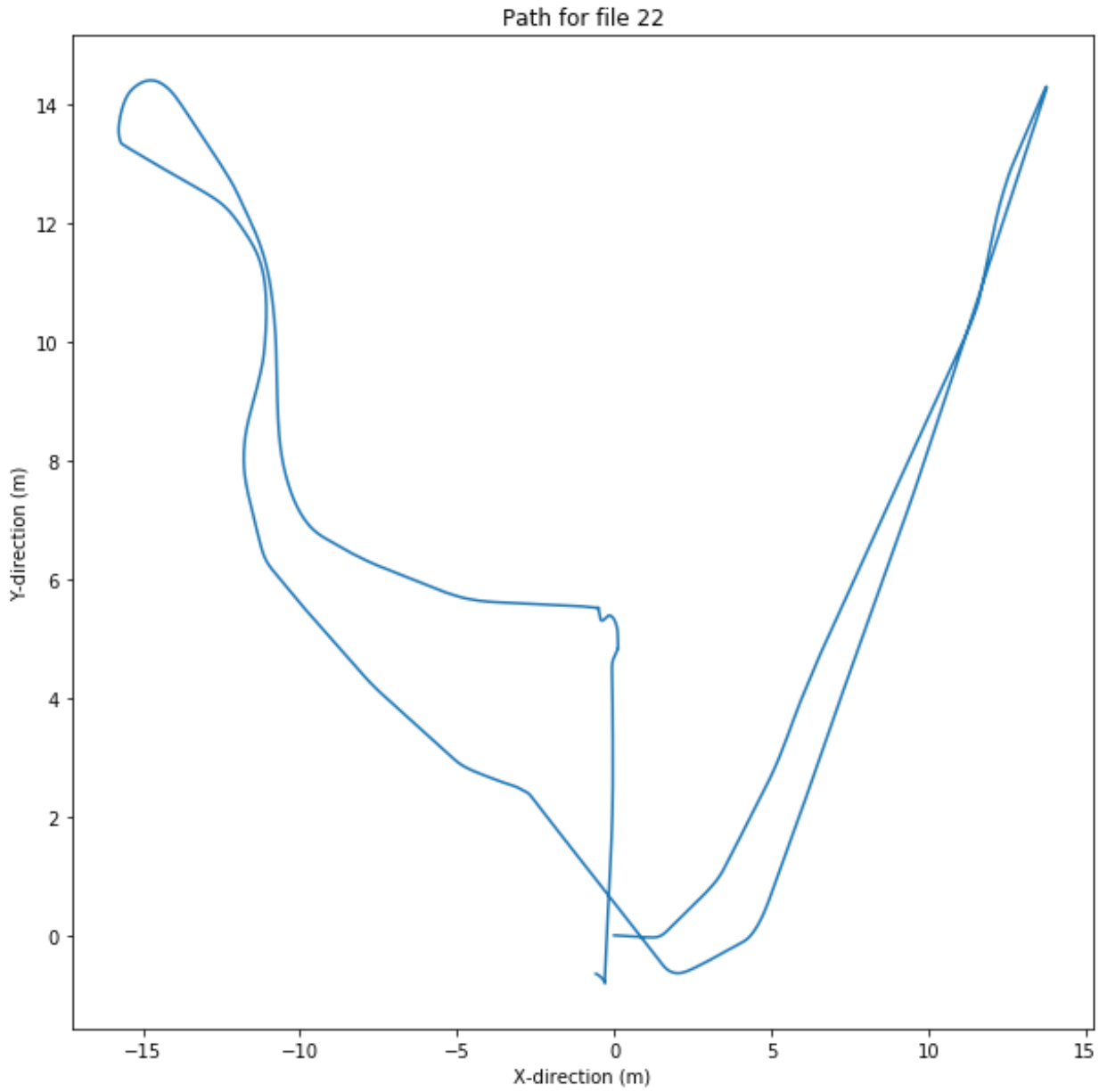

Figure 3: Part 1 Robot Path for file 22

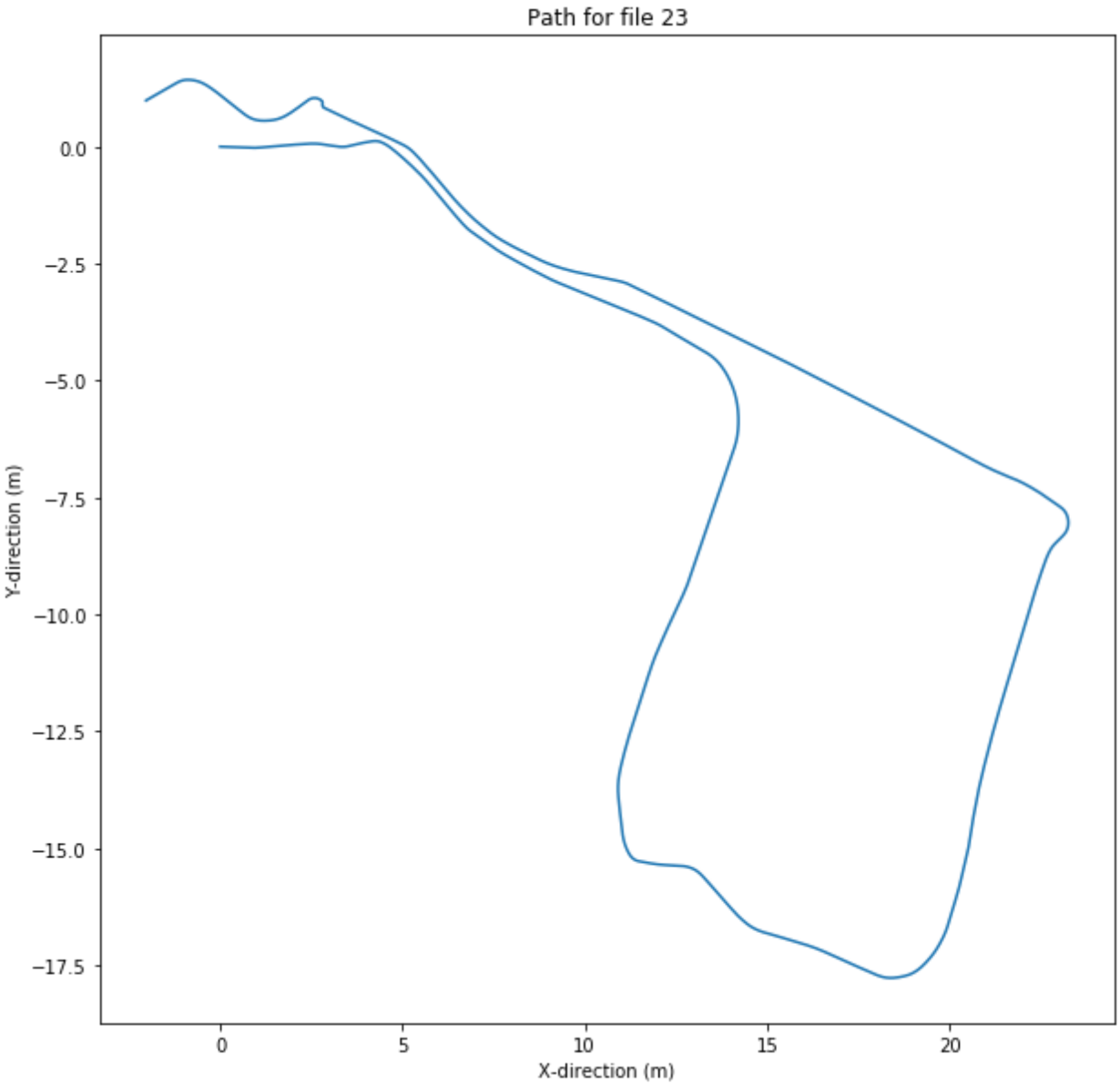

Figure 4: Part 1 Robot Path for file 23

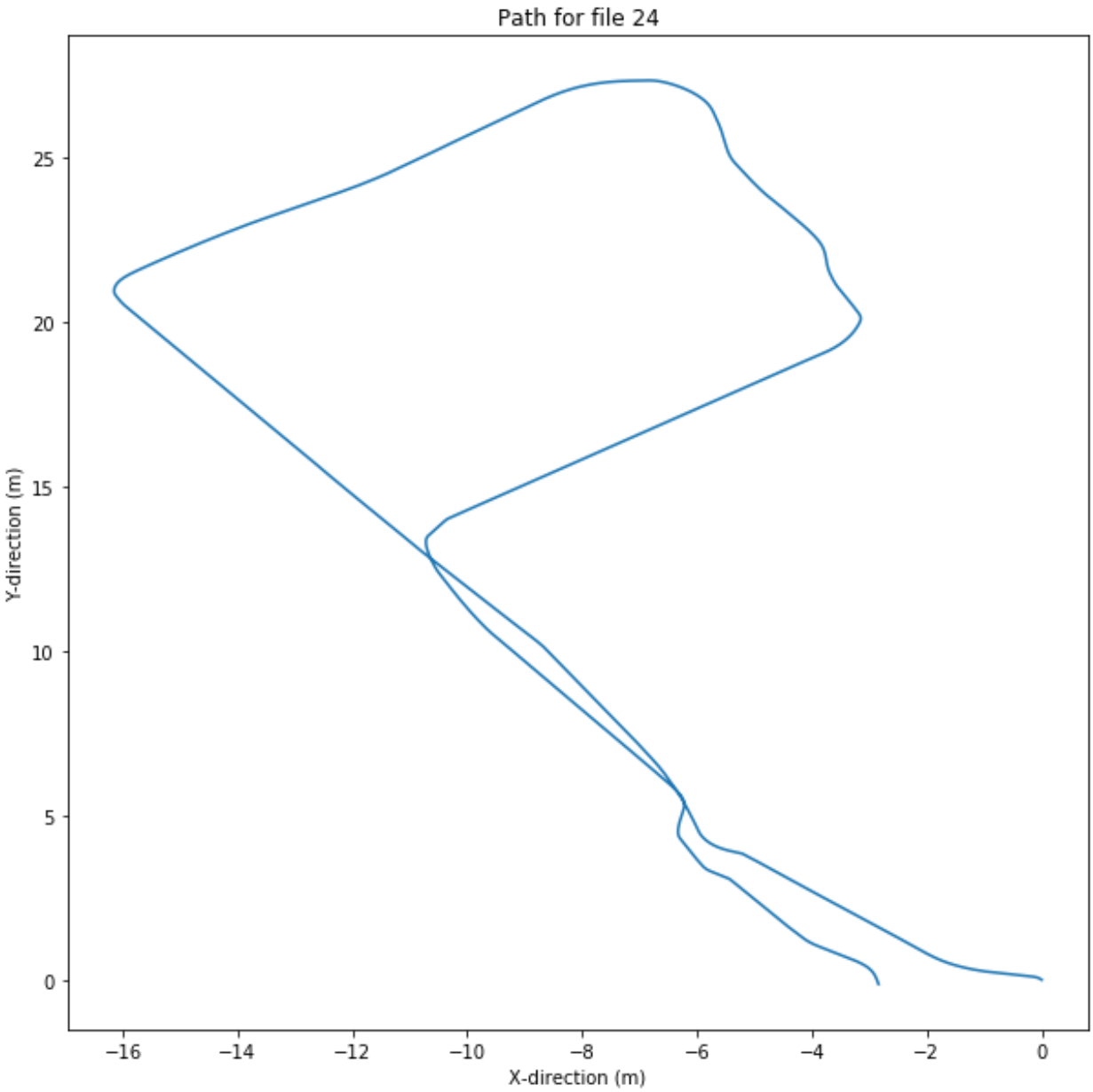

Figure 5: Part 1 Robot Path for file 24

Part 2 Results

The following is the results obtained from part 2 of the project. It shows the robot path as determined by using only the odometer data with the map generated using the corresponding Lidar data. The robot path is indicated in blue inside the map.

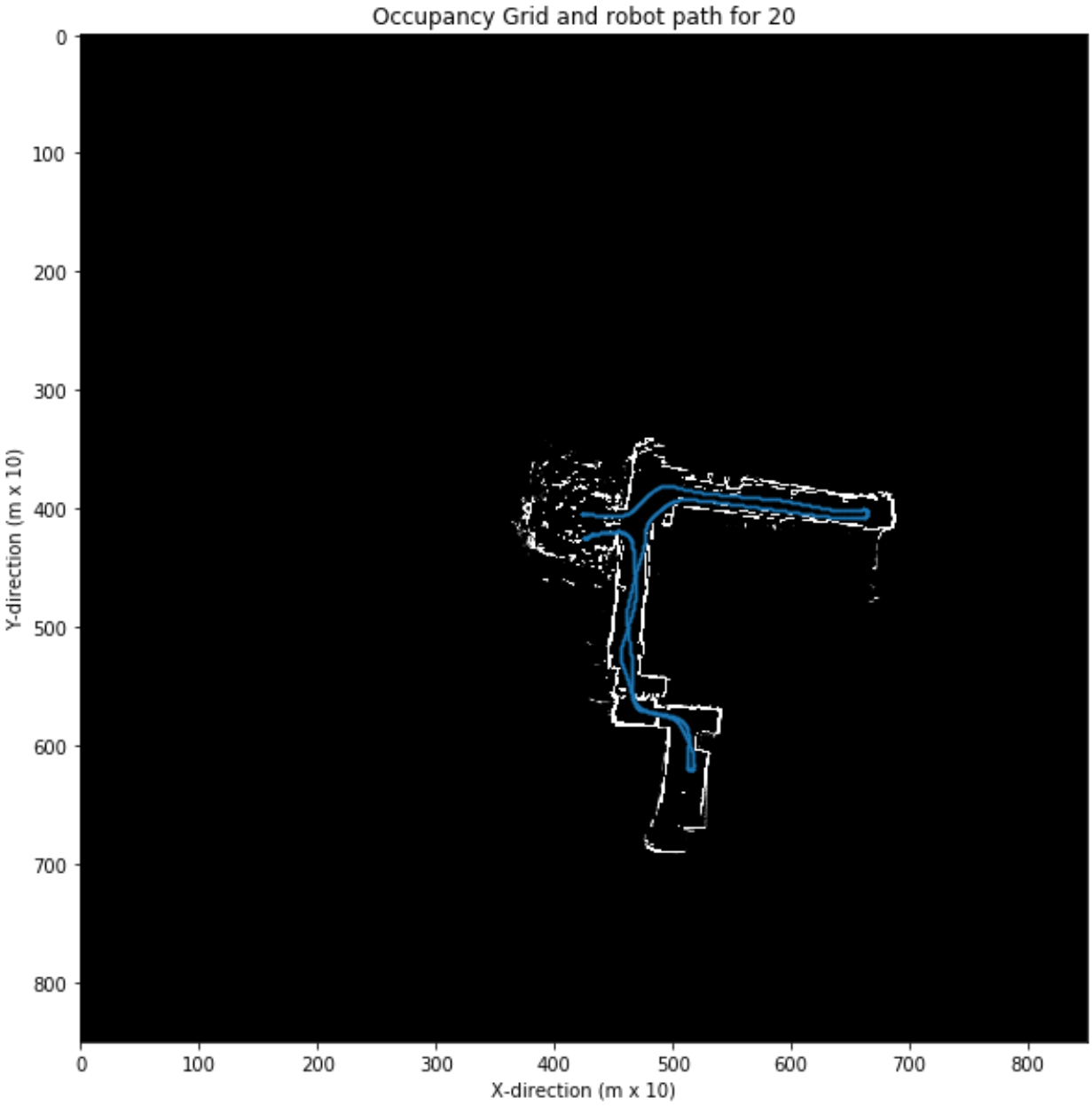

Figure 6: Part 2 Occupancy Grid and Robot Path for file 20

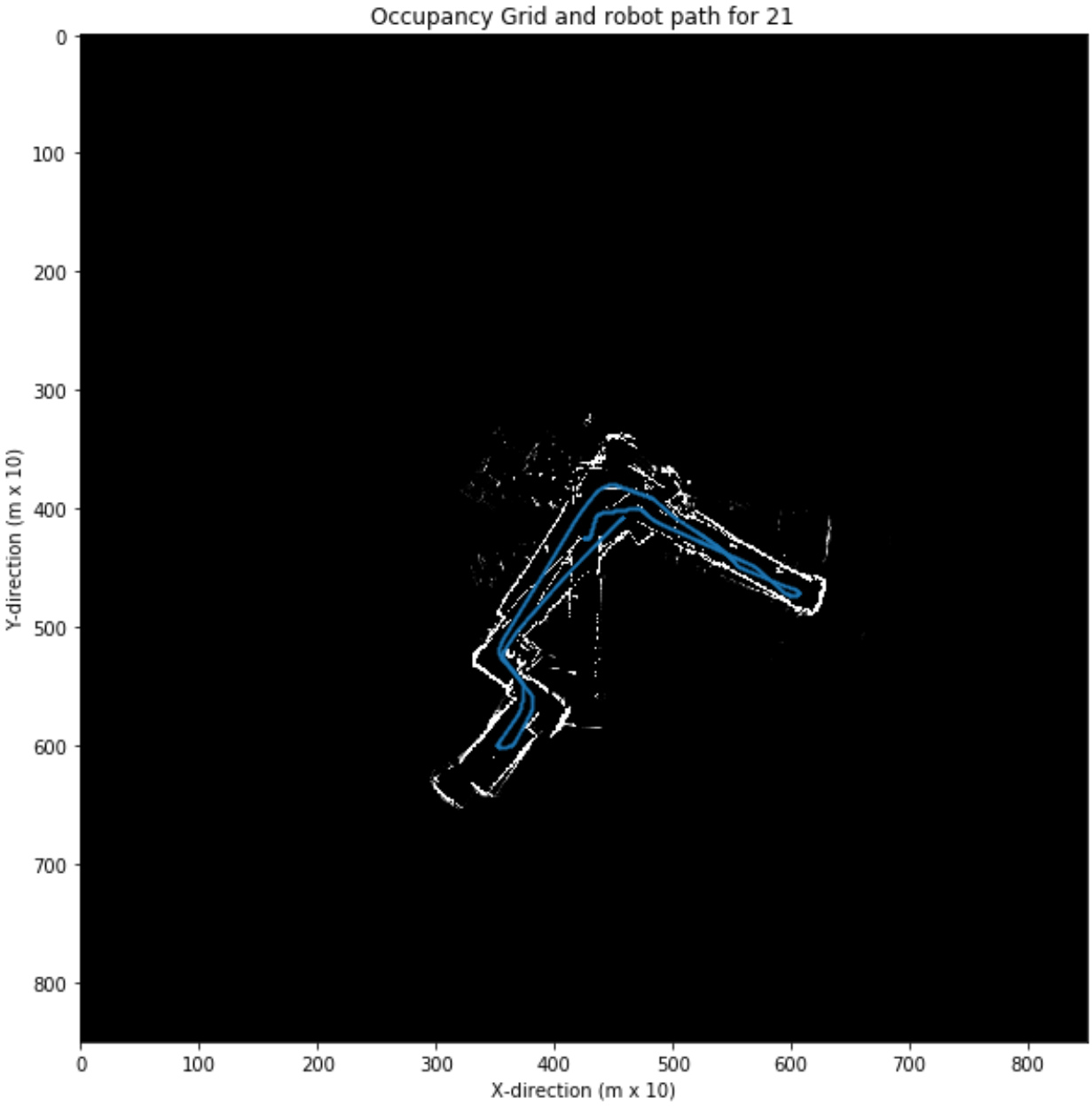

Figure 7: Part 2 Occupancy Grid and Robot Path for file 21

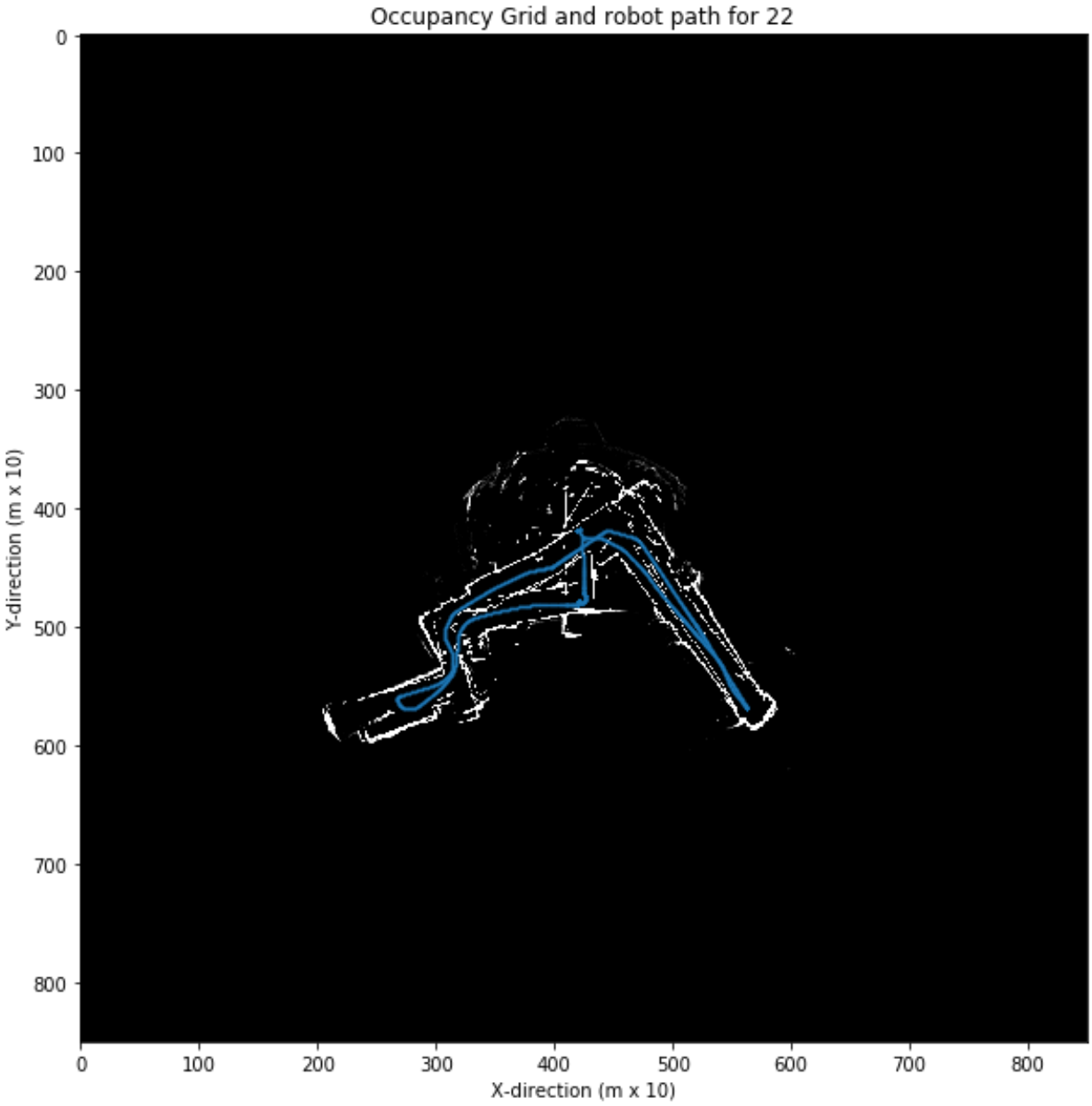

Figure 8: Part 2 Occupancy Grid and Robot Path for file 22

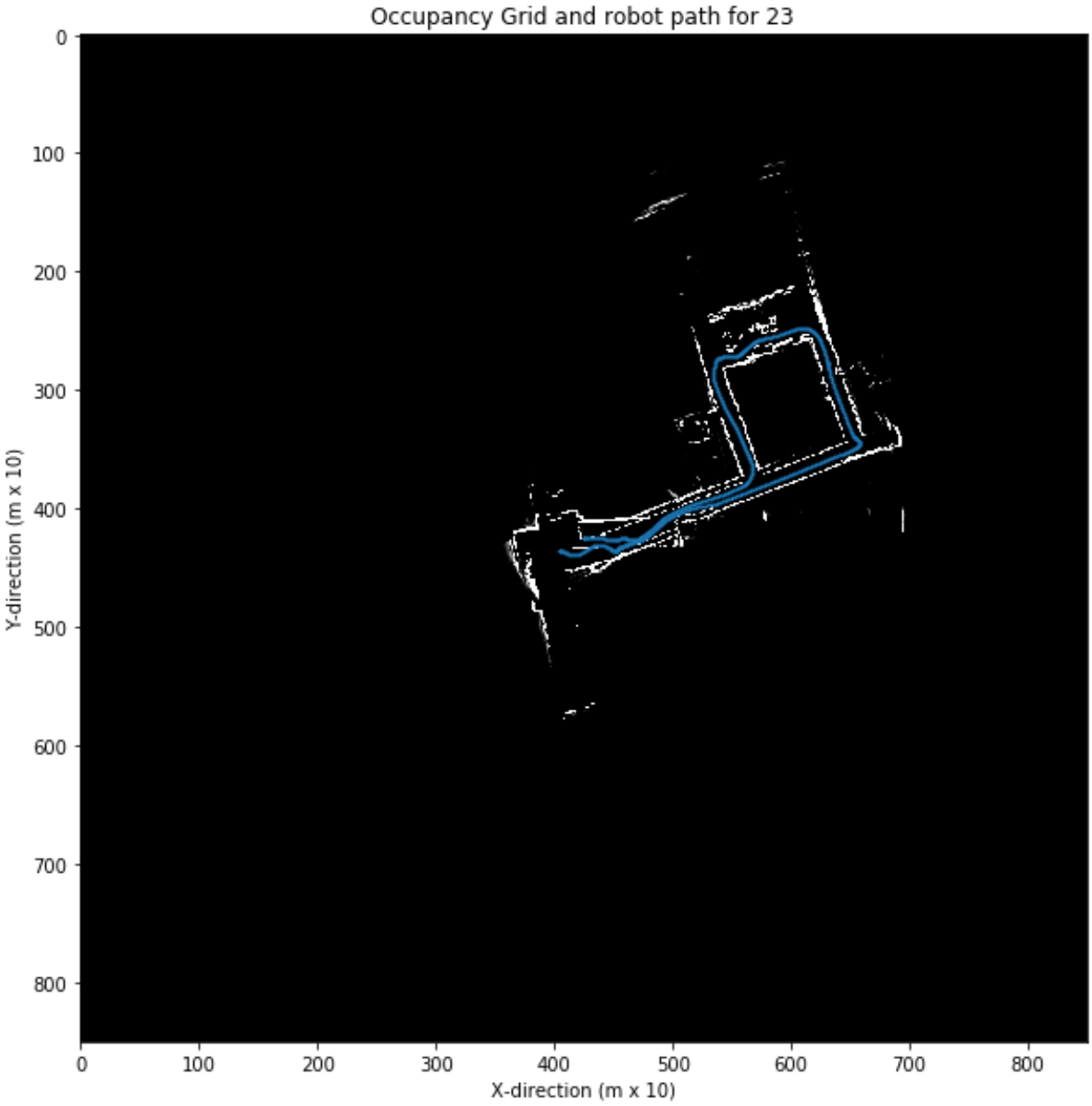

Figure 9: Part 2 Occupancy Grid and Robot Path for file 23

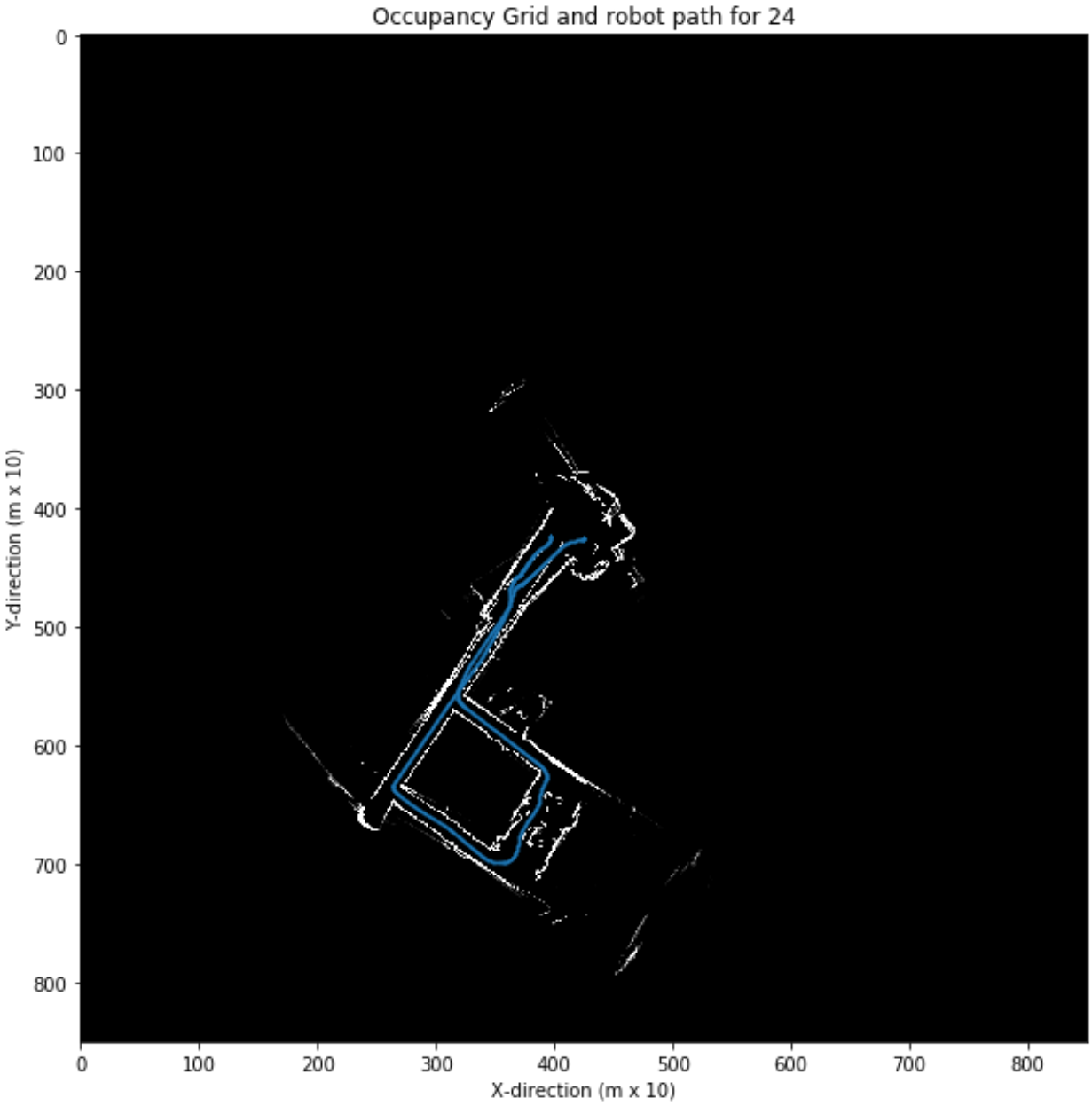

Figure 10: Part 2 Occupancy Grid and Robot Path for file 24

Part 3 Results

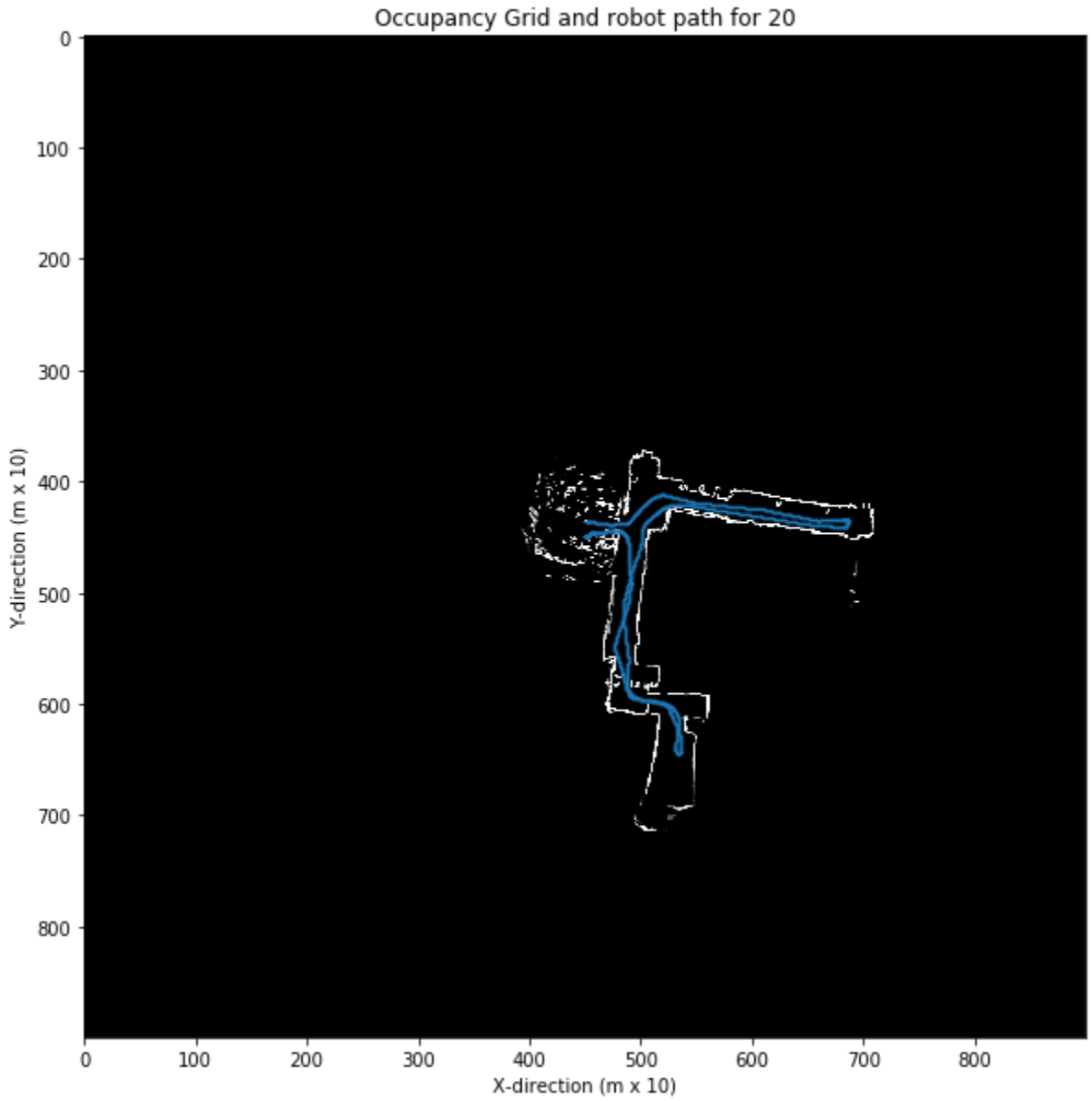

Figure 11: Part 3 Occupancy Grid and Robot Path for file 20

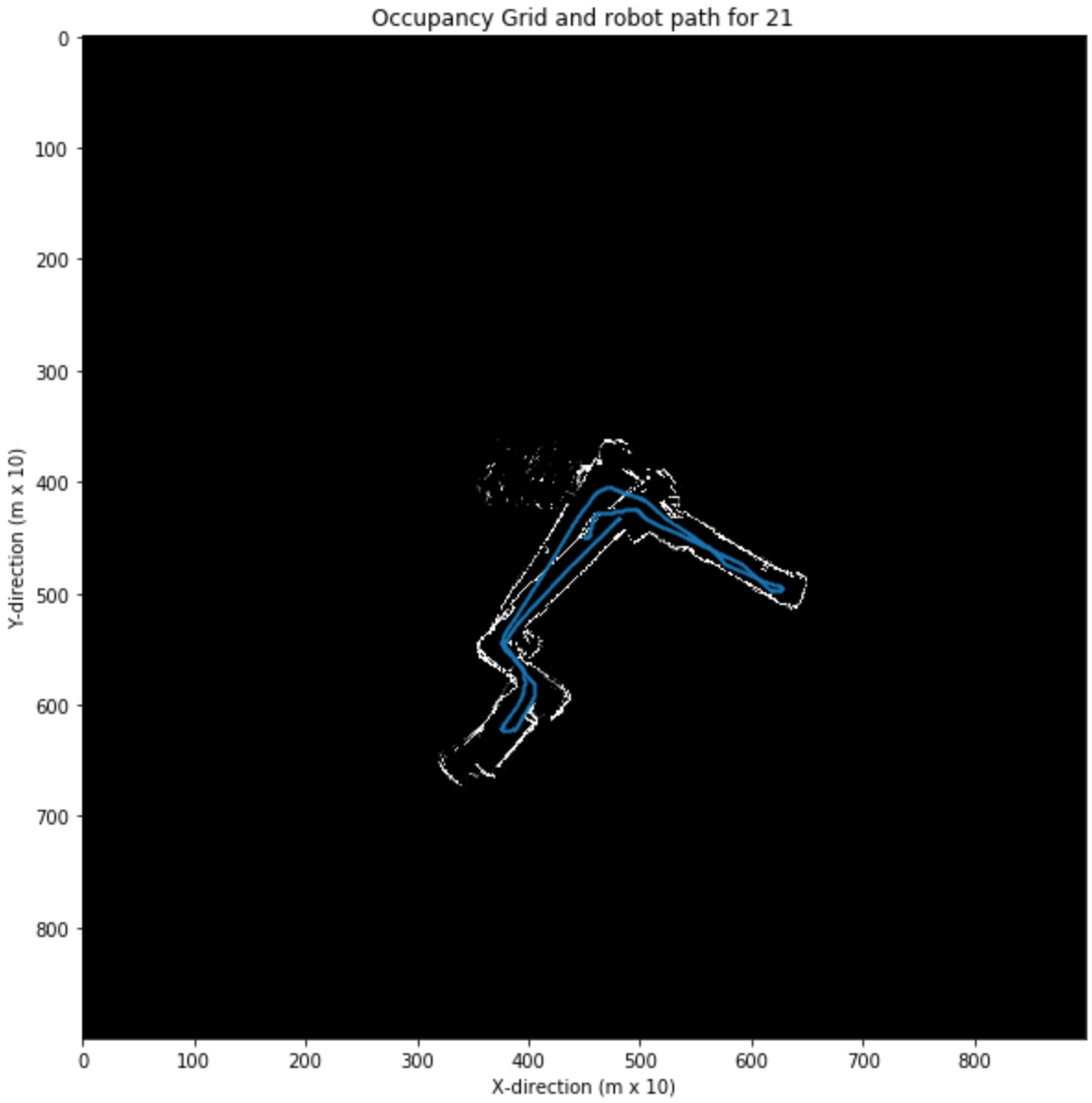

Figure 12: Part 3 Occupancy Grid and Robot Path for file 21

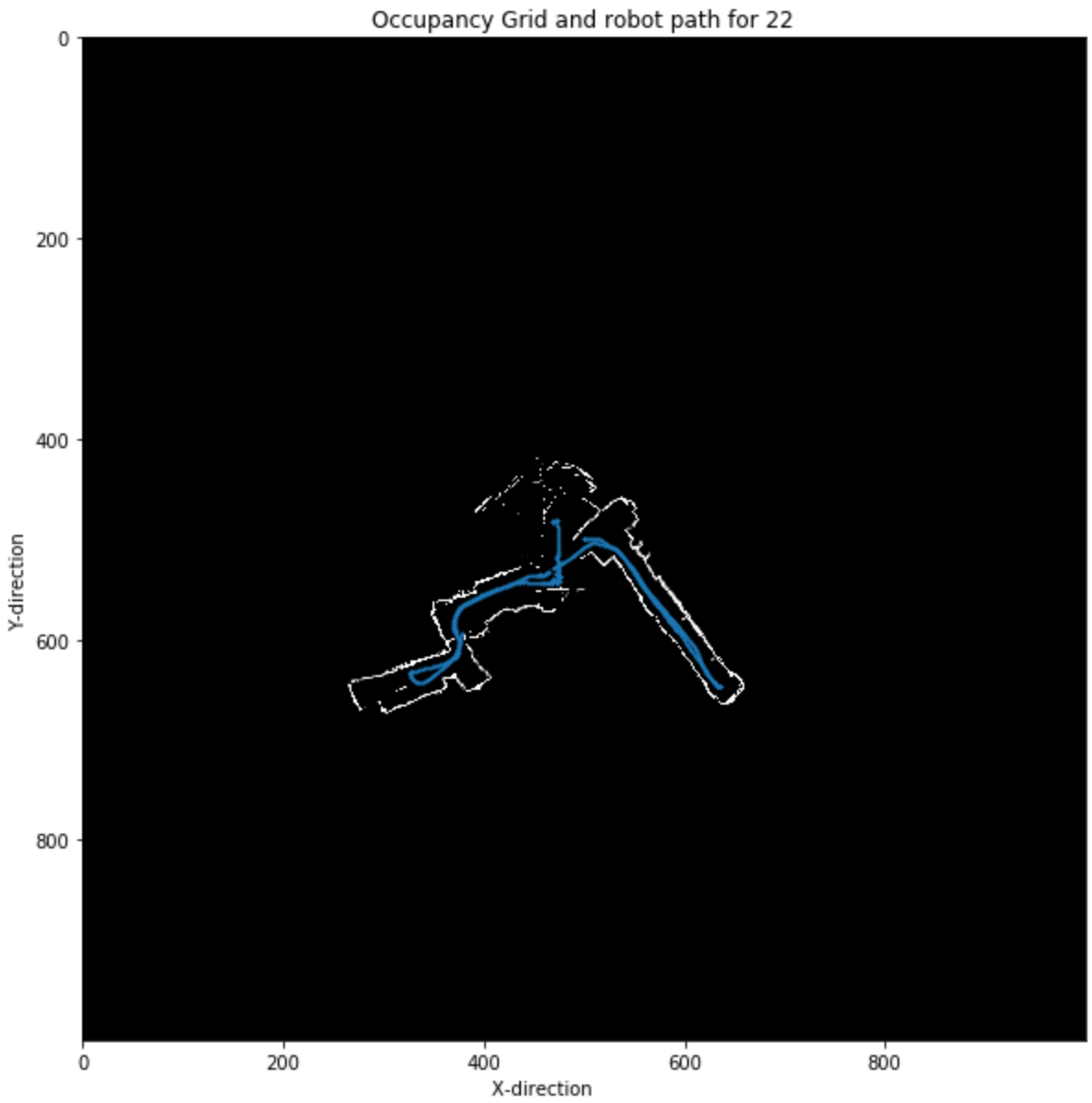

Figure 13: Part 3 Occupancy Grid and Robot Path for file 22

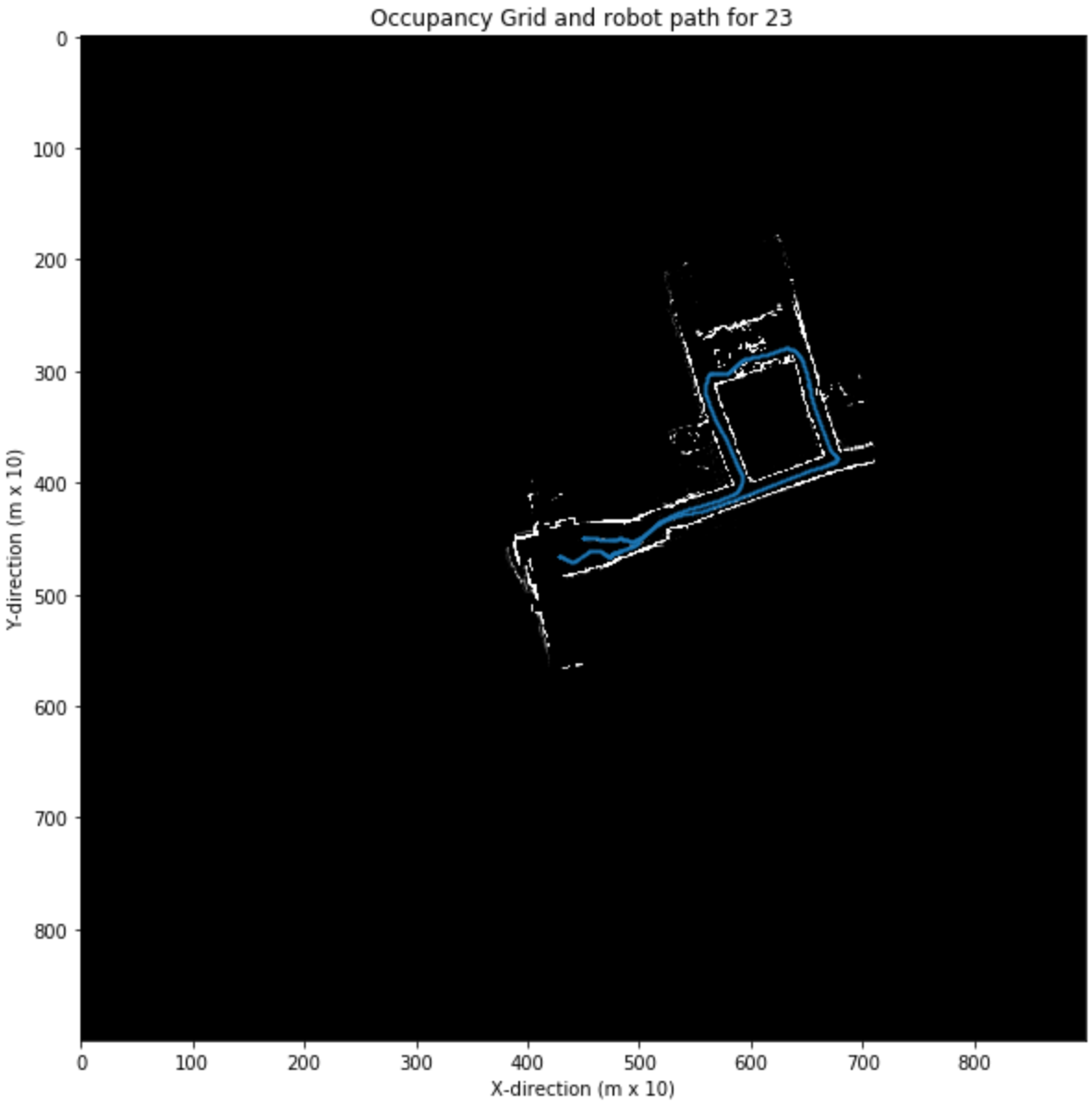

Figure 14: Part 3 Occupancy Grid and Robot Path for file 23

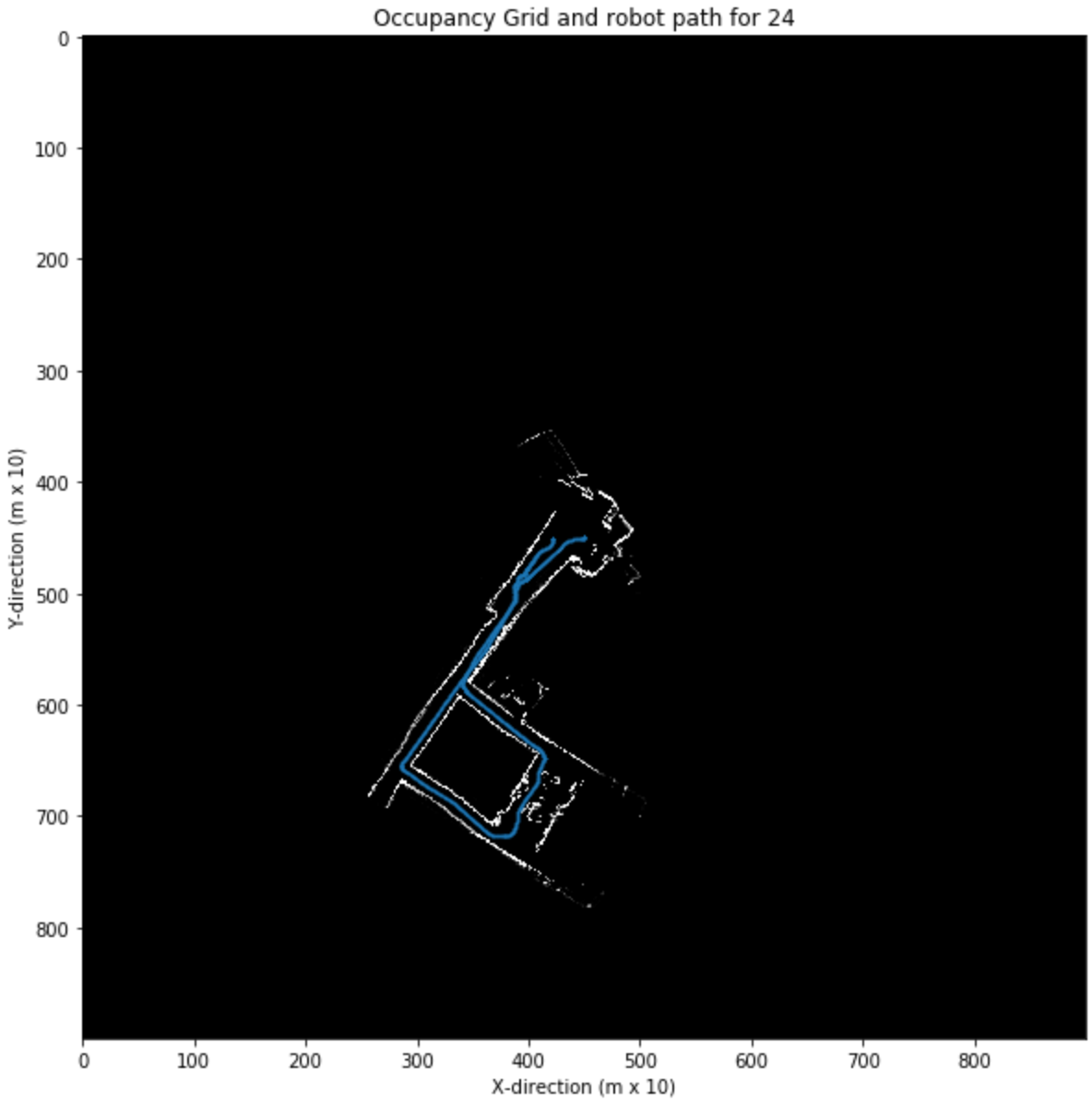

Figure 15: Part 3 Occupancy Grid and Robot Path for file 24

Discussion

When comparing the results obtained in part 2 and part 3 of the project the effects of the particle filter can clearly be seen. There are a lot of slight robot path adjustments that has been made by the particle filter which has led to more natural robot paths and better environment maps being generated. Some of the best effects the particle filter has had is in removing ghost hallways and improving the definition in the map. Also, it became clear that there where slight deviations in the results obtained when running the particle filter each time. Thus, some instances when it was running it greatly improved the map or some

parts of the map where other times the improvements where less. This can be due to the limited particles being used thus some instances the noise generated match the real-world values we are trying to find better then other instances. This can probably be solved by slightly increasing the standard deviations used and greatly increasing the number of particles used.

Thru experimentation it became clear that more particles where always better but that a trade of needs to happen since increasing the particles drastically decreases performance. A factor that also greatly impacted the results of the particle filter where the standard deviations chosen to add noise to the odometer data. A trade off was also necessary here as for a higher standard deviation a higher number of particles where necessary to ensure good results. A trade off was also needed in terms of picking a standard deviation that could fix small as well as large errors. On data sets which only had minor error having a small standard deviation allowed for small adjustments to be made by the filter which led to very good results but in data sets with large errors this failed as the standard deviation was to small to allow for the large errors to be corrected. It also became clear that the ratio of hit increase to miss decrease had to be increased for the filter. As for the filter to work it needs a well-defined map with solid objects.

Ghost Hallway removal

An example of ghost hallway removal can be seen in file 21 and 23.

Figure 16: part 2 file 23 has a ghost hallway

Figure 17: part 3 file 23 ghost hallway removed

From the above figures it can clearly be seen that the particle filter worked and removed the ghost hallway that was present in file 23.

Figure 18: part 2 file 21 has a ghost hallway

Figure 19: part 3 file 21 ghost hallway removed

From the above figures it can clearly be seen that the particle filter worked and removed the ghost hallway that was present in file 21.

Conclusion

From this project it is clear that it is possible to plot a ruth estimate of a robot path using only odometer data but that this should never be taken as the true path of the robot. In some of the data sets it is alarming the error that can exist in the odometer data such as in data set 22 which if taken as the truth adds a completely new hallway and turn to the map which does not exist. The effectiveness and necessity of making use of a SLAM algorithm such as a particle filter when using raw sensor data such as odometer data is clearly shown. The particle filter corrects the mistakes in the path obtained when using only the odometer data such as in file 22 where it corrects the path putting the robot path back on the real map eliminating the fake hallways. The particle filter also removes artifacts from the generated maps such as ghost hallways which were detected because the true robot orientation did not agree with the calculated orientation at that point. The filter removes these by correcting the robot orientation along the path. Sometimes when the filter is running it just works better then other times this can be attributed to generated noise matching the desired real-world values better than other times. This is an easy problem to solve by simply increasing the standard deviation slightly and greatly increasing the number of particles.

This was not done in this project due to the computational cost this would add as it would greatly increase the run time of the algorithm.