ECE 5242 Intelligent Autonomous Systems

Project 3: SLAM

Alana Crognale arc232

Apr 13, 2020

Introduction

Simultaneous localization and mapping (SLAM) aims to create a mapping of both a robot's surrounding environment and the robot's location within this environment. SLAM has many applications, such as self-driving vehicles, planetary rovers, and autonomous drones. In this project, our goal is to use wheel encoder data and range sensor readings to construct a mapping and localization of an unknown indoor environment's walls and obstacles. Specifically, we will train sets of odometry, inertial, and 2D laser range scanner (LIDAR) readings in order to construct an accurate 2D occupancy grid map of the environment, as well as to visualize the robot's movement and pose within the environment. To do so, we will incorporate techniques including dead reckoning, occupancy grid algorithm, and particle filtering.

Problem Formulation

How can we implement SLAM with sets of odometry, inertial, and range readings to visualize an accurate 2D mapping of an unknown environment and a robot's trajectory within the environment?

Technical Approach

Wheel-Odometry Mapping of robot path:

Using encoder data from each of the four wheels, we first average the two outer (right) wheels together and the two inner (left) wheels together, converting the encoder ticks to change in degrees per time sample using the robot's radius of 0.127m, obtained from the platform configuration document. I assumed a coordinate system such that at time 0 and theta of 0 degrees, the robot is at origin (0,0) on the horizontal x-axis facing towards the right (positive). Then, we use dead reckoning to convert these change in degrees per time sample to maintain the cumulative robot's pose over time, calculating the theta parameter by dividing the change in outer and inner degree changes by the robot width of 0.7m (an initial width of 0.311m was used per the platform configuration document; however, due to aspects like friction, this width parameter was subsequently fine-tuned), and the X global and Y global coordinates by averaging the outer and inner degree changes and multiplying by cosine of theta at that time sample or sine of theta at that time sample respectively.

Initial Map:

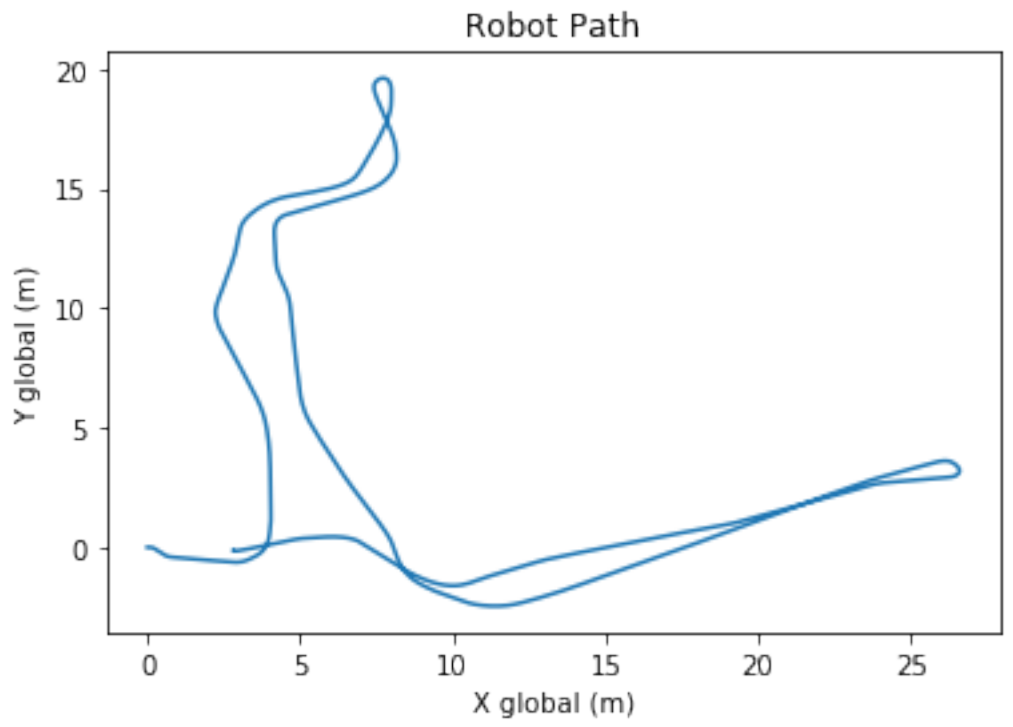
Next, we need to align the timestamps of the data taken from the LIDAR hits with the data taken from the wheel encoder, since the sensors record at slightly different sampling rates. This means that some of the LIDAR samples may be repeated when matching to the nearest encoder timestamp. Then, we initialize our map space such that each cell corresponds to a 0.05 meter by 0.05 meter space, ranging from -30 meters to 30 meters in both the x-axis and y-axis. Then, we convert the LIDAR hits in meters, which are taken relative to the robot pose, to the global map in meters using the previously calculated global robot coordinates. Once the global robot pose and global LIDAR hits have been converted from meters to map cell grids, we loop through each time sample and perform the occupancy grid algorithm: For every LIDAR hit, we add the log-odds value of 0.9 to that map cell. For every miss, calculated with the Bresenham line algorithm, we subtract the log-odds value of 0.7 from that map cell. I capped the maximum cell log-odds limit to 30 and minimum cell log-odds limit to -30 to ensure that the overall distribution maintains within a reasonable range for visualization, such that the darkest regions correspond to regions with the highest probability of being non-occupied, and the lightest regions correspond to regions with the highest probability of being occupied.

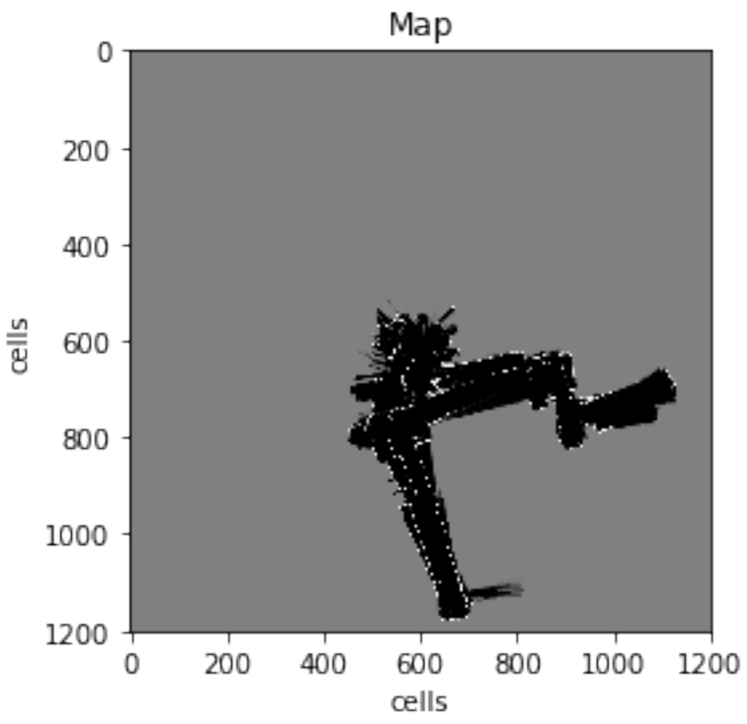
SLAM:

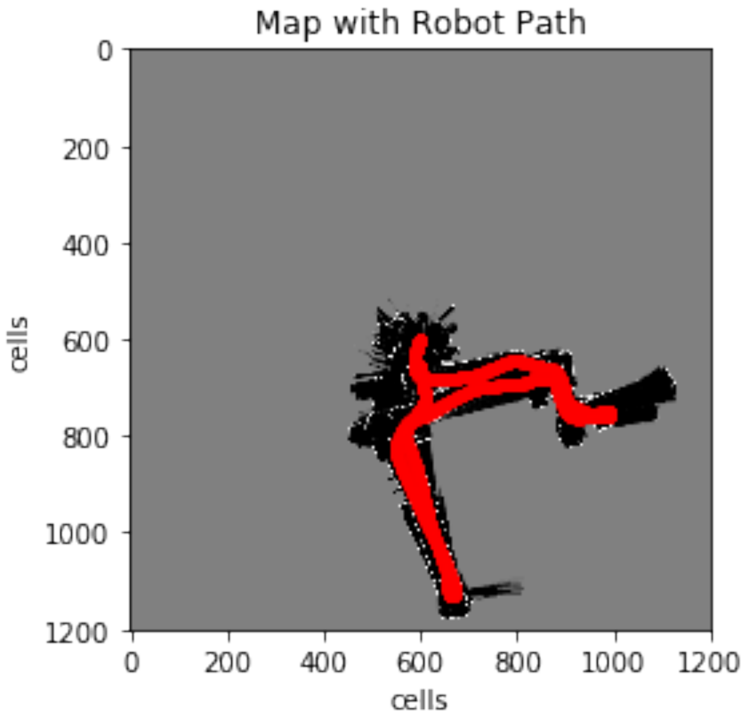
To implement particle filtering, we first initialize the number of particles to N = 70 in order to optimize the tradeoff between too few particles, which would essentially produce the same results as the odometry only map, and too many particles, which significantly increases computation time and memory. We initialize a set of weights for each particle to be equal to the log of 1/N. For every time sample, we add a small amount of noise to each of the averaged outer and inner encoder readings which will then carry over to the global robot pose readings. This noise was produced per each time sample for each particle based on a gaussian distribution with mean of 0 and standard deviation of 0.0001. For each time sample, we then update the new pose for each particle, convert these poses to global cell coordinates, convert the LIDAR hits and misses for each of the particles into global cell grids, and perform the occupancy grid algorithm similar to how we did in the case of one particle (on the first iteration, all particles have equal weights, so we can choose any of the particles to update the map). Once the map's log-odds values have been updated for a given time sample, we update the correlation (the sum of log-odds at the LIDAR hits for each particle) for each particle and add this to the current particle weights to update which particles are more closely aligned with the existing map and which particles have diverged more from the existing map. Note that the weight vector stores the log-odds value, so to get the true weight value, we can take the exponential, thus we then normalize the weights by subtracting the log of the sum of exponential values from each particle's current log-odds value. This is equivalent to dividing the exponential of each entry by the exponential of the sum of all entries, but the log-odds subtraction method avoids issues related to overflow and underflow, the potential to divide by 0, and it allows for log-odds values to be negative. After the weights have been updated, for every time sample we then choose the particle with the highest weight and update the map based on this particle's update. For every time iteration, we also check whether we need to resample the particles, which happens when the poses of the particles have diverged too much resulting in inaccurate map updates. We calculate the effective number of particles by dividing the square of the sum of all weights (exponentiating each entry to obtain the true weight) by the sum of the square of each weight. If this number drops below 30, we resample: ranging from 0 to 1, we represent the cumulative sum of all weights as a distribution and sample uniformly from 0 to 1 N times, replacing each particle with whichever particle this sampling corresponds to. Thus, particles with higher weights are more likely to be sampled, leading to more accurate maps. We then re-normalize and iterate.

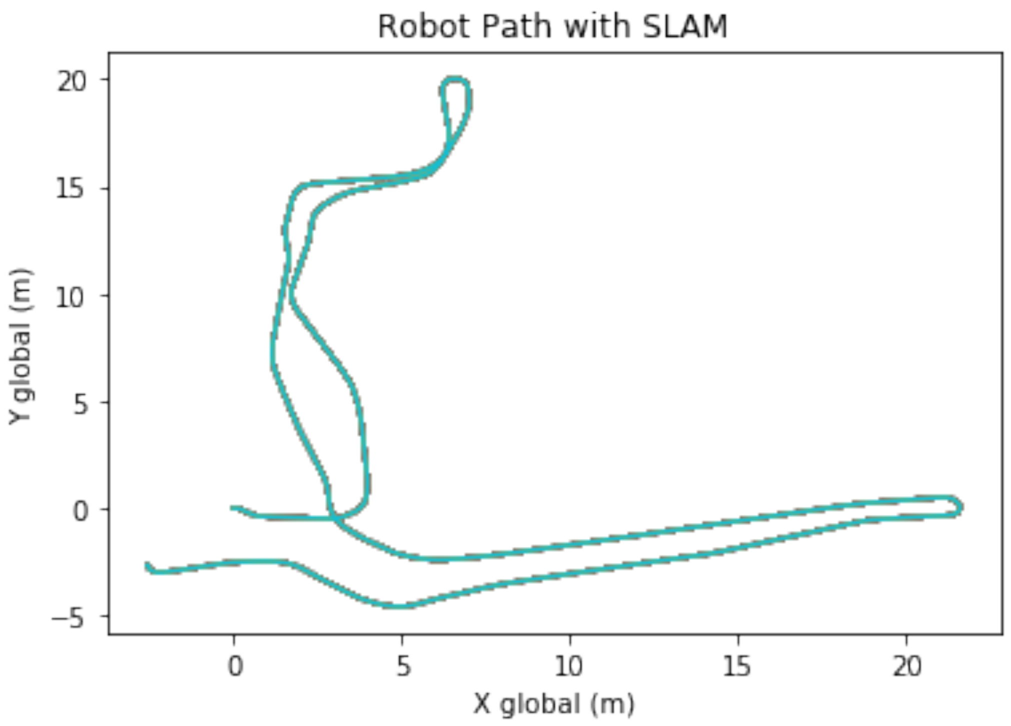
Results and Discussion

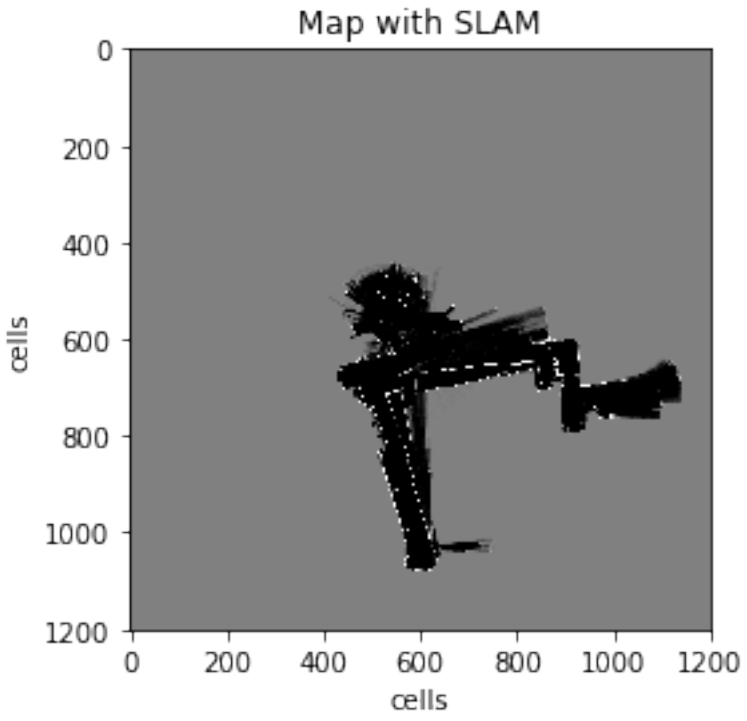
Map20:



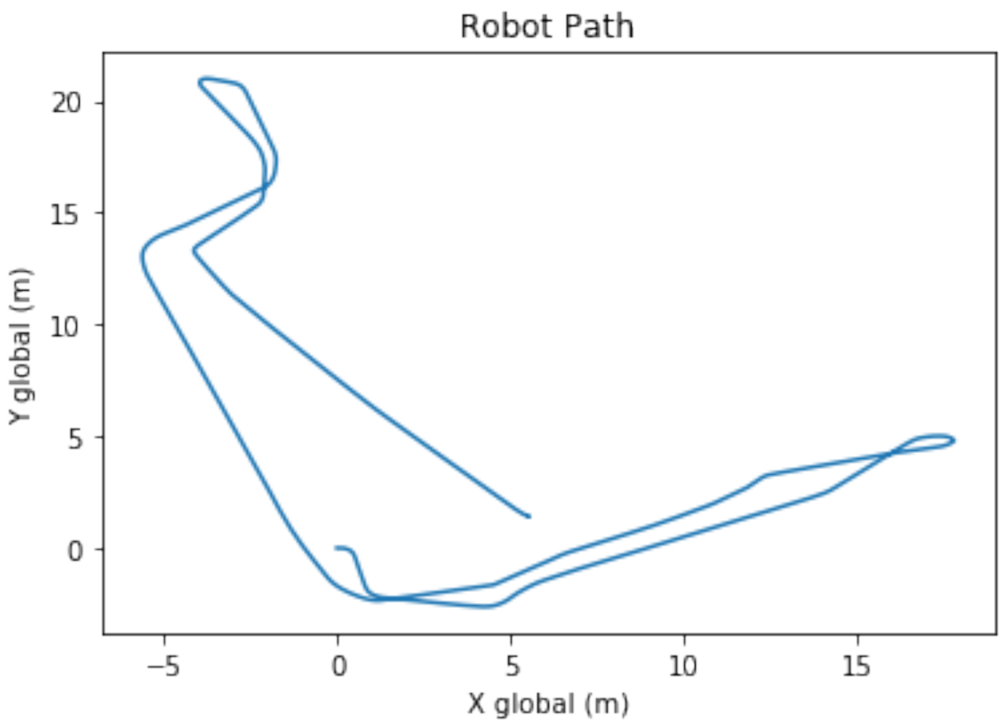

Map
cells
cells
0
200
400
600
800
1000
1200

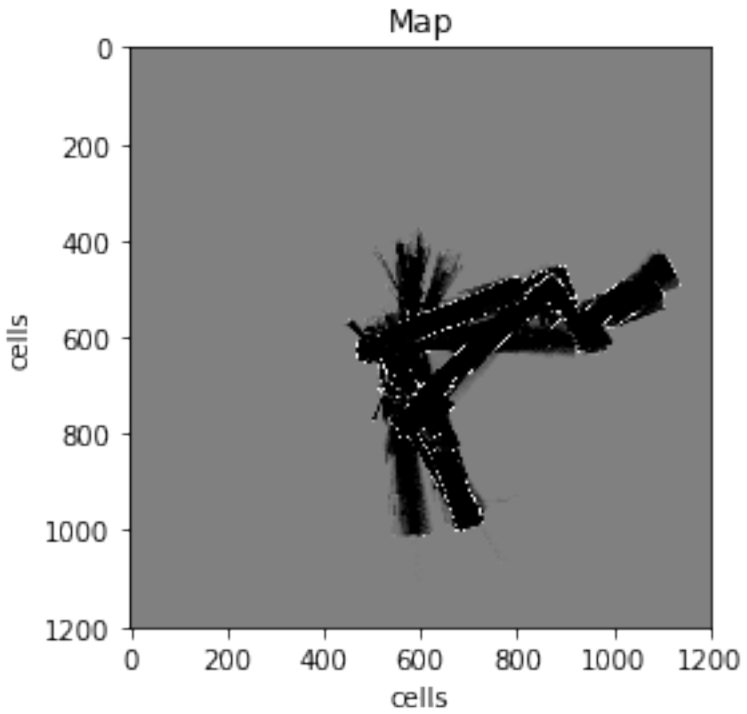

Map with Robot Path
cells
cells
0
200
400
600
800
1000
1200

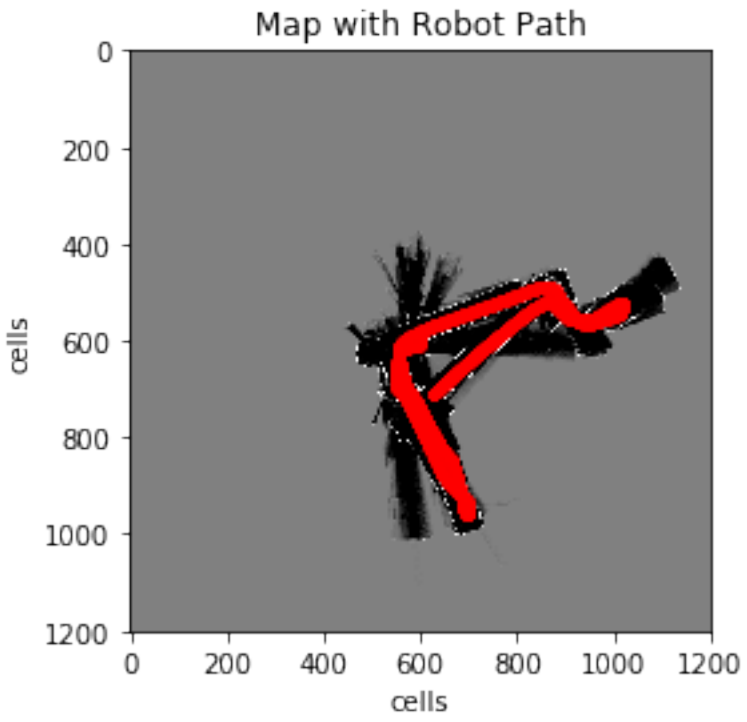

Robot Path with SLAM
Y global (m)
X global (m)
20
15
10
5
0
−5
0
5
10
15
20

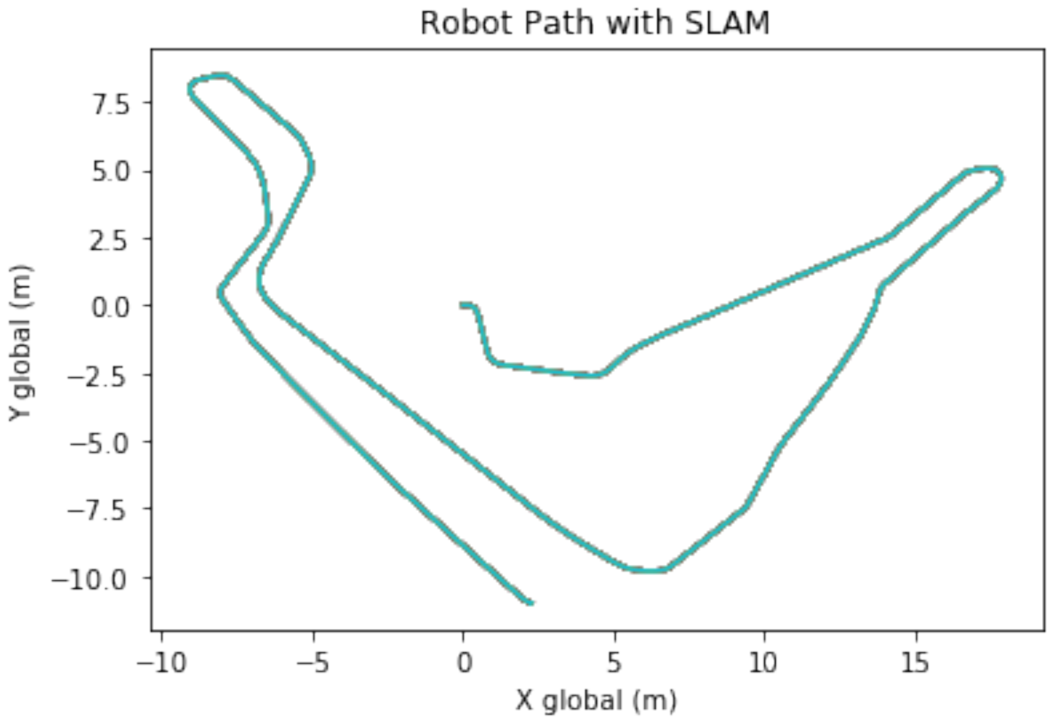

Map with SLAM
cells
cells
0
200
400
600
800
1000
1200

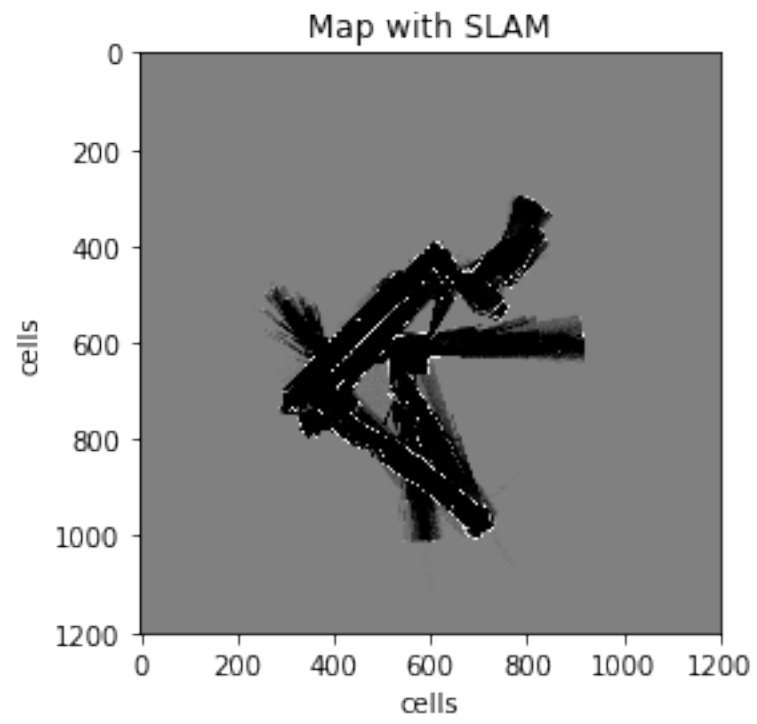
Map21:


Robot Path
Y global (m)
X global (m)
20
15
10
5
0
−5
0
5
10
15


Map
cells
cells
0
200
400
600
800
1000
1200

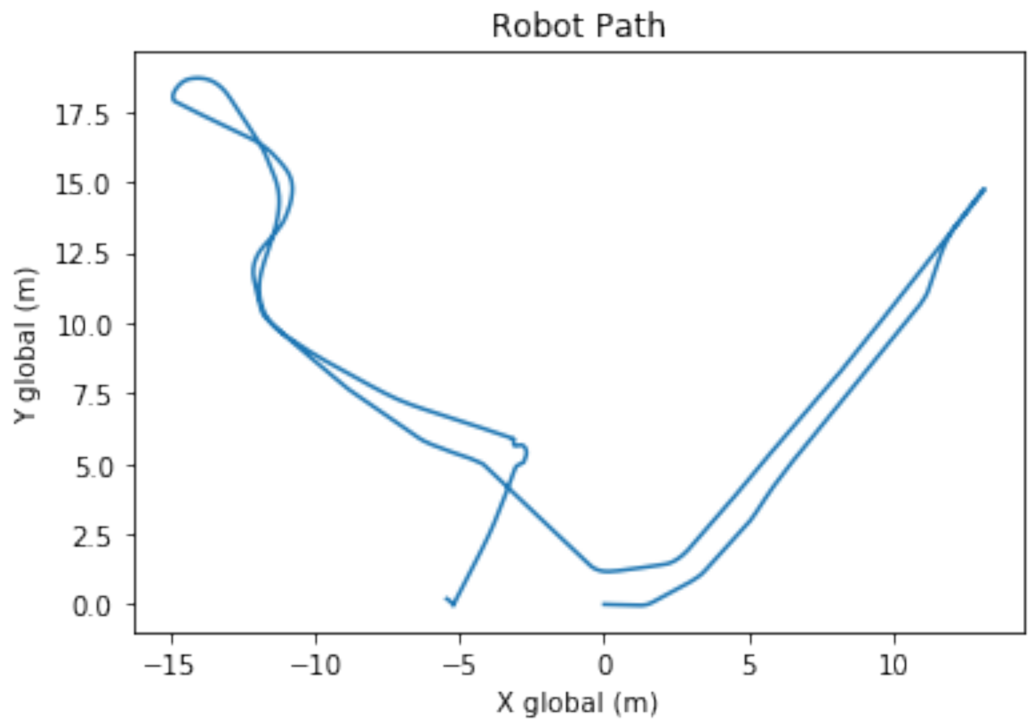

Map with Robot Path
cells
cells
0
200
400
600
800
1000
1200


Robot Path with SLAM
Y global (m)
X global (m)
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10
15

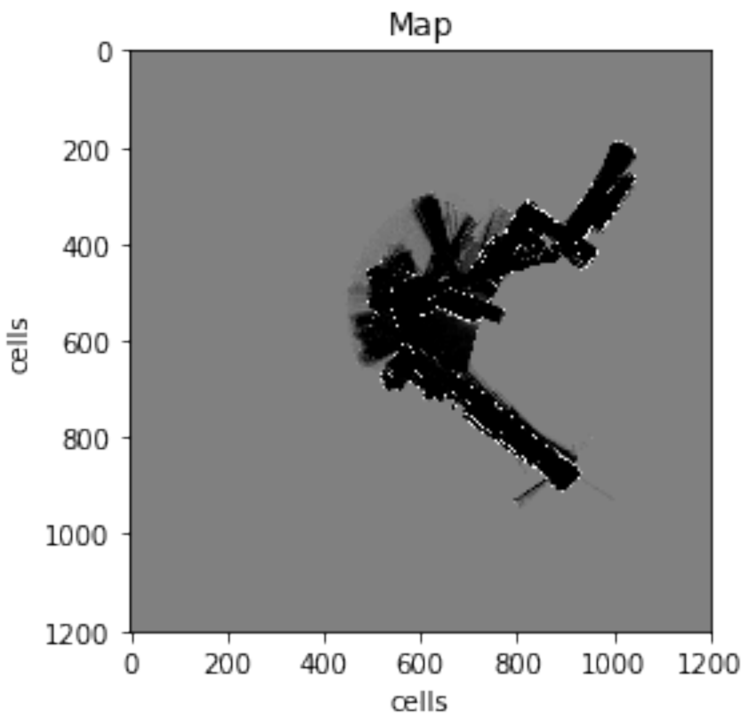


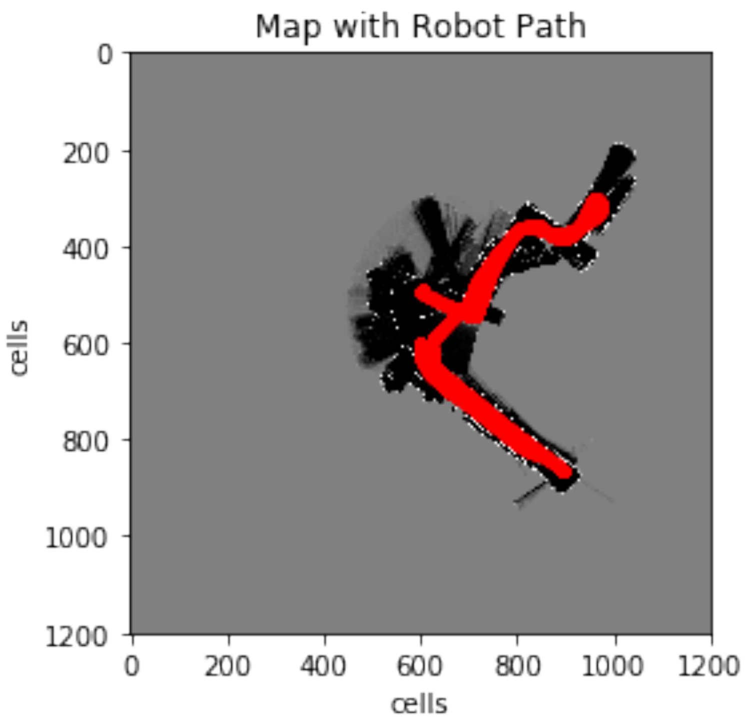
Map with SLAM
0
200
400
600
800
1000
1200
cells
0
200
400
600
800
1000
1200
cells

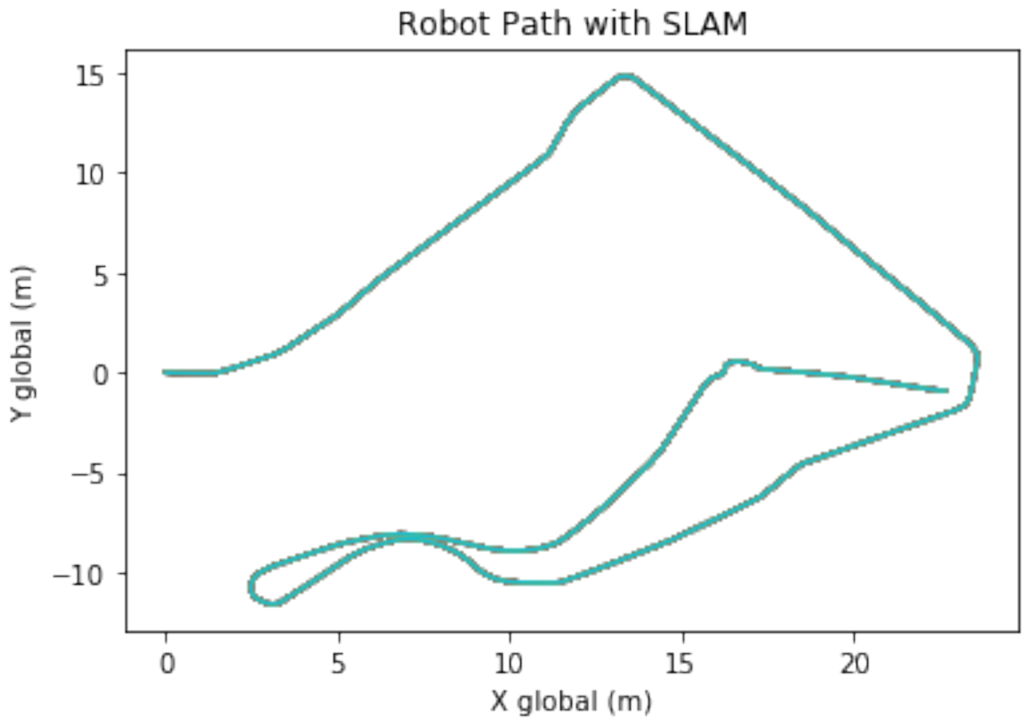
Map22:

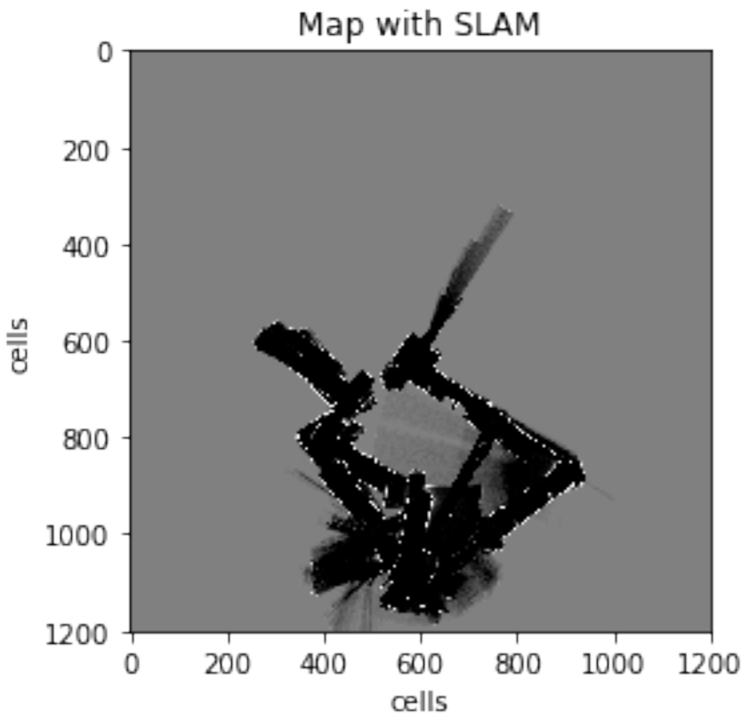


Robot Path
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Y global (m)
−15
−10
−5
0
5
10
X global (m)

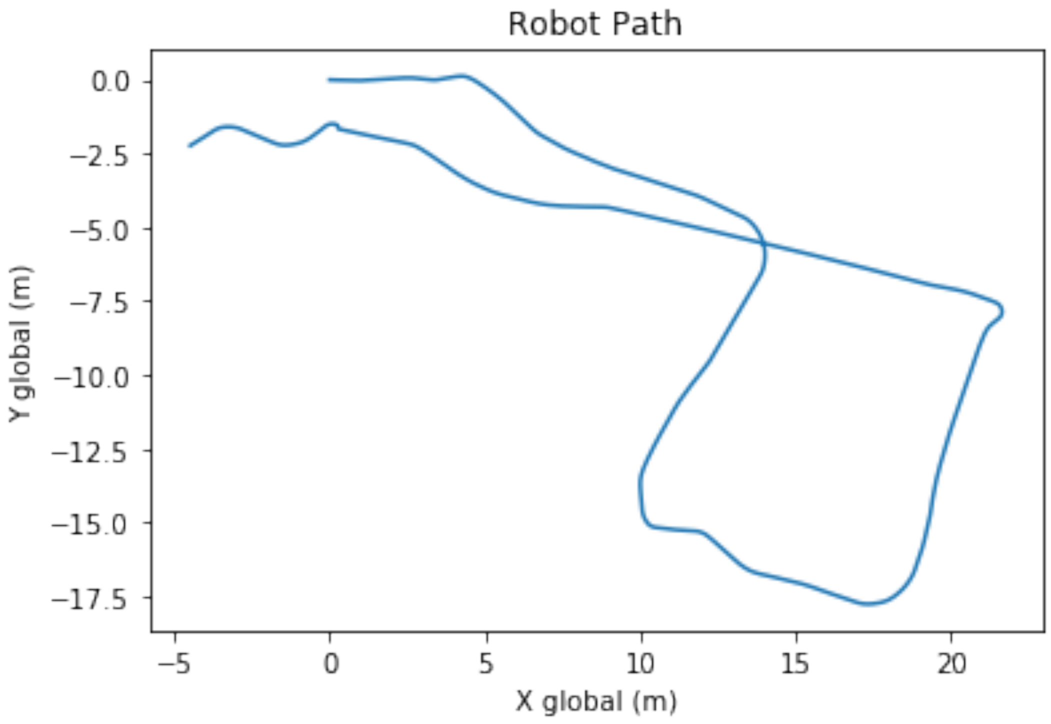

Map
cells
cells
0
200
400
600
800
1000
1200

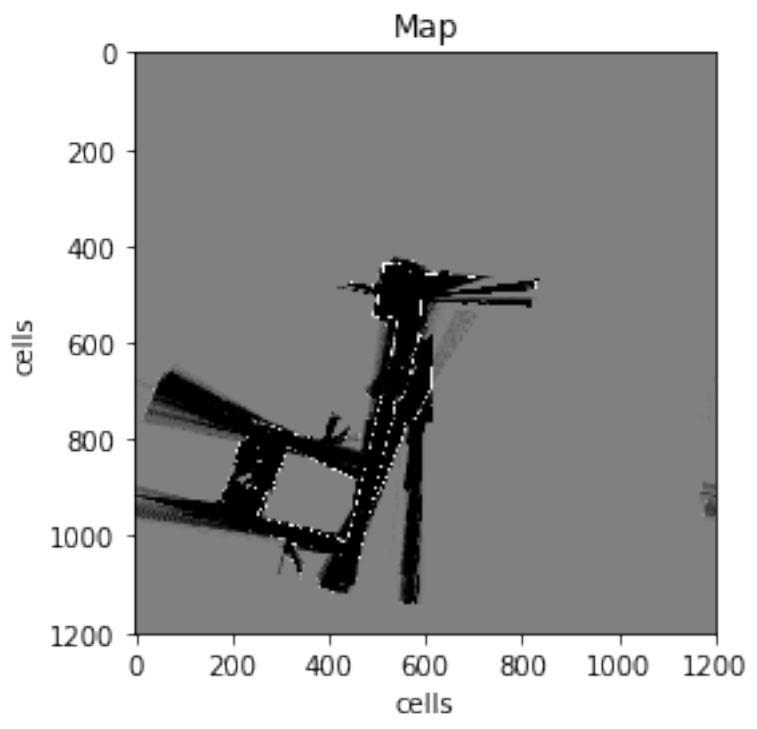

Map with Robot Path
cells
cells
0
200
400
600
800
1000
1200

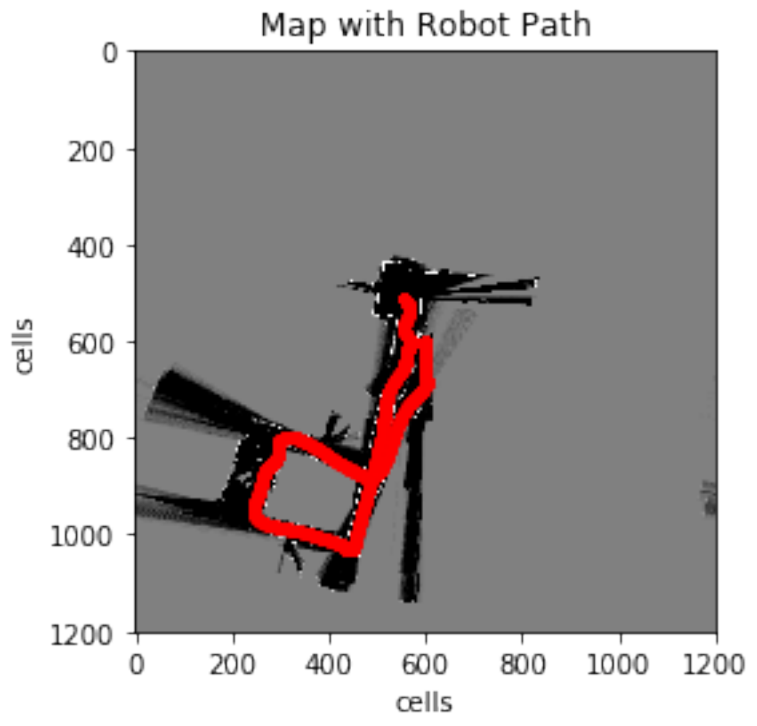

Robot Path with SLAM
Y global (m)
X global (m)
15
10
5
0
−5
−10
0
5
10
15
20

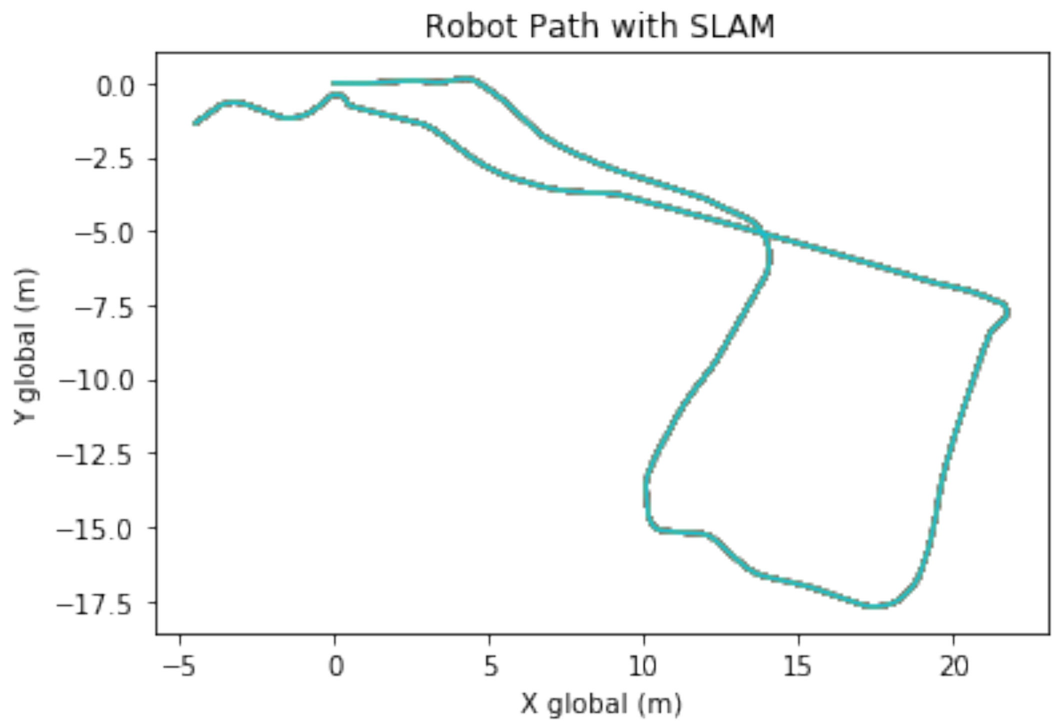

Map with SLAM
cells
cells
0
200
400
600
800
1000
1200

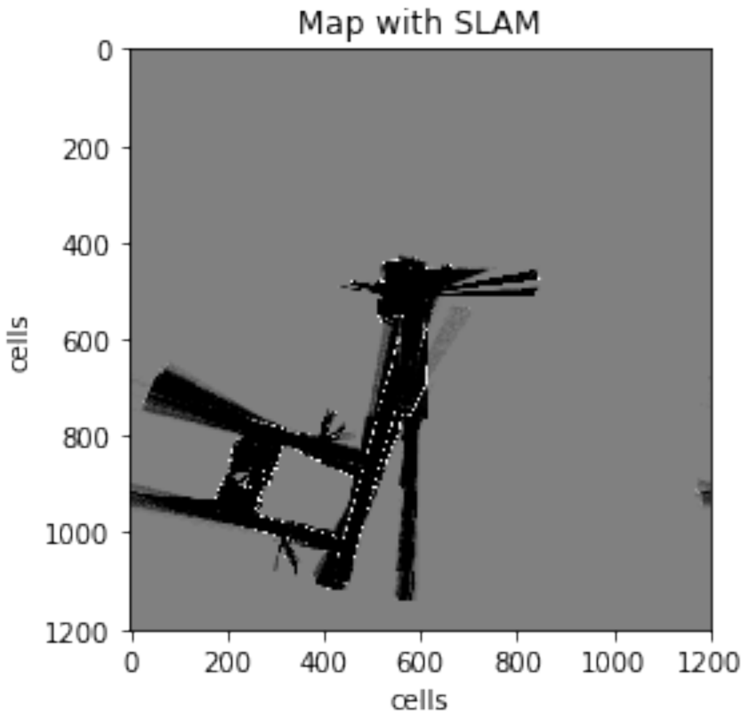
Map23:


Robot Path
Y global (m)
X global (m)
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−5
0
5
10
15
20

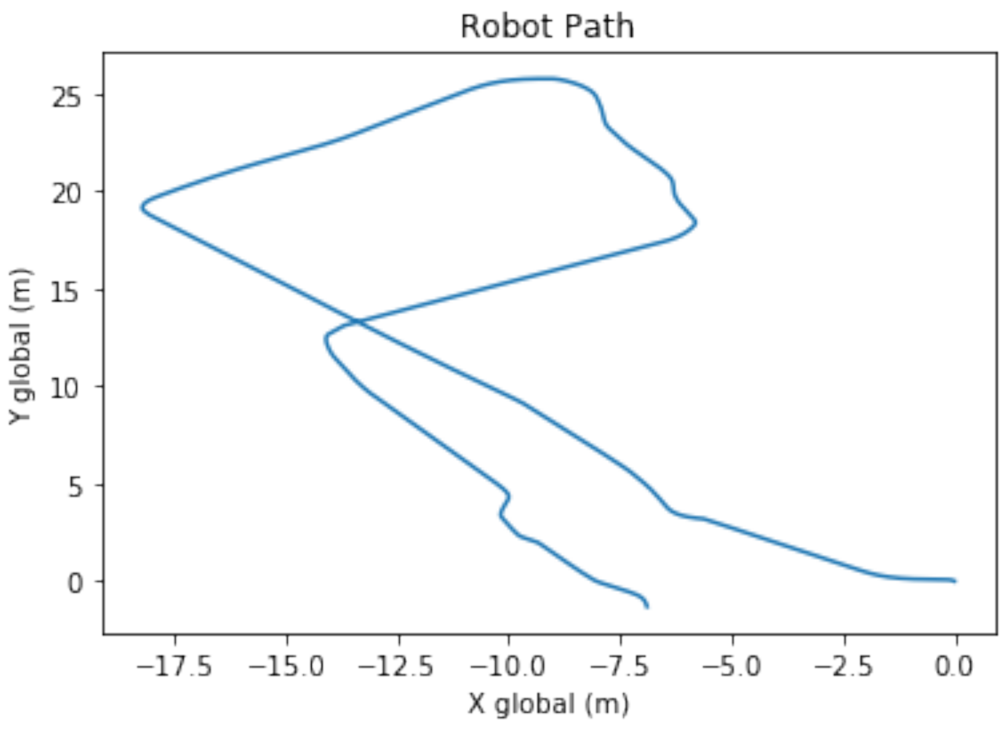

Map
cells
0
200
400
600
800
1000
1200
cells
0
200
400
600
800
1000
1200

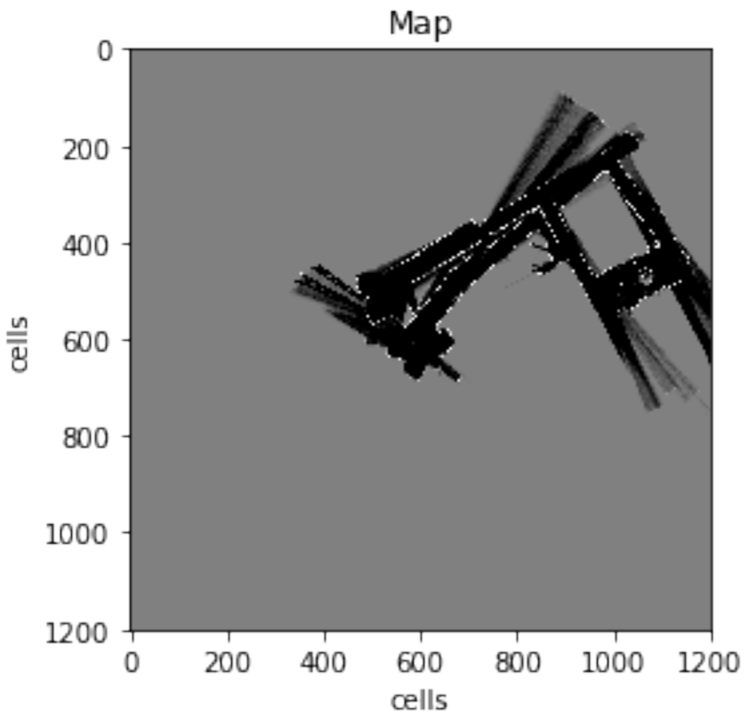

Map with Robot Path
cells
0
200
400
600
800
1000
1200
cells
0
200
400
600
800
1000
1200

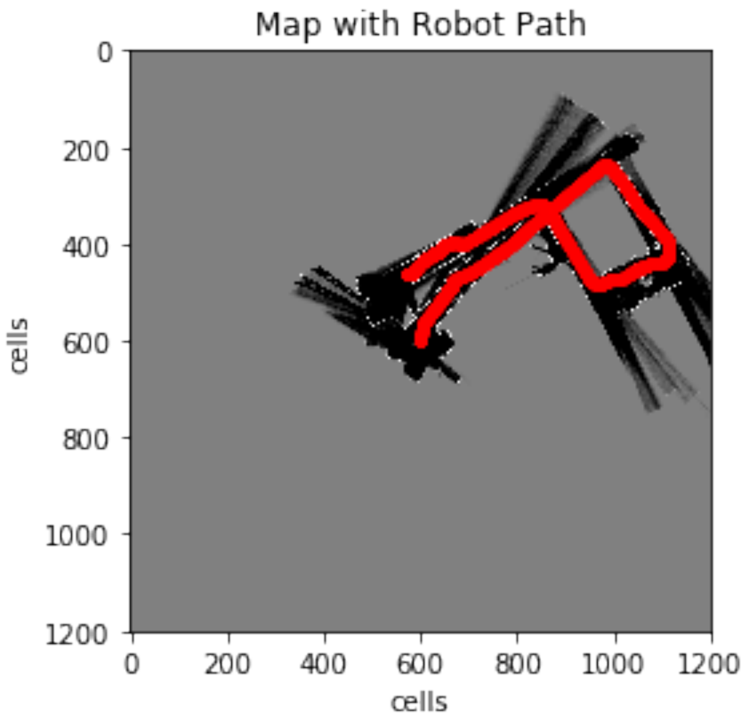

Robot Path with SLAM
Y global (m)
X global (m)
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−5
0
5
10
15
20

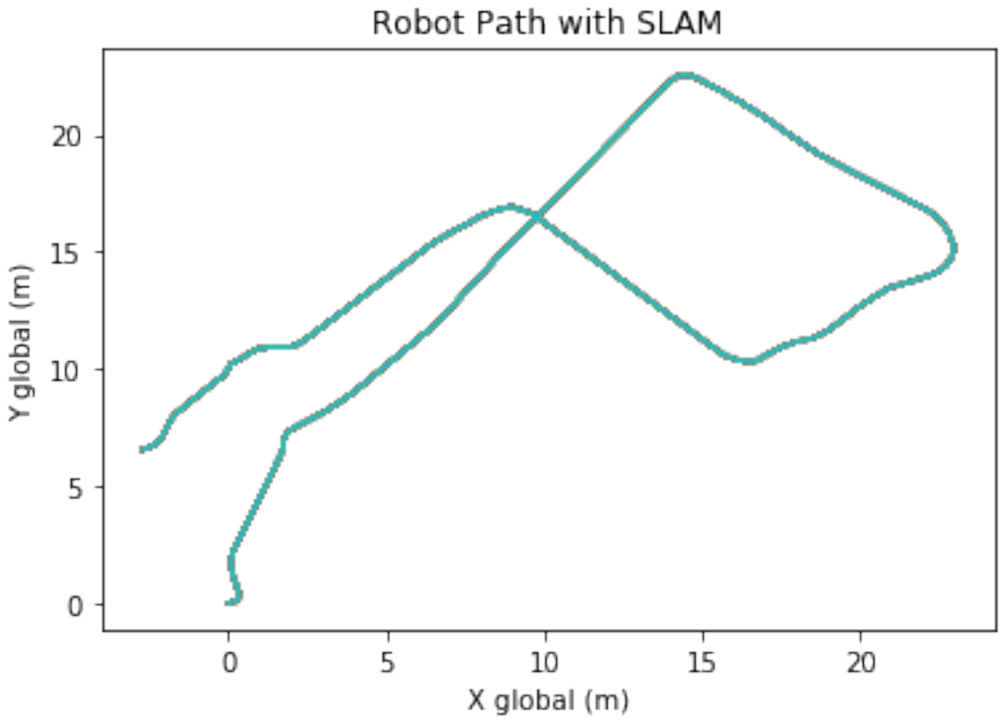

Map with SLAM
cells
cells
0
200
400
600
800
1000
1200

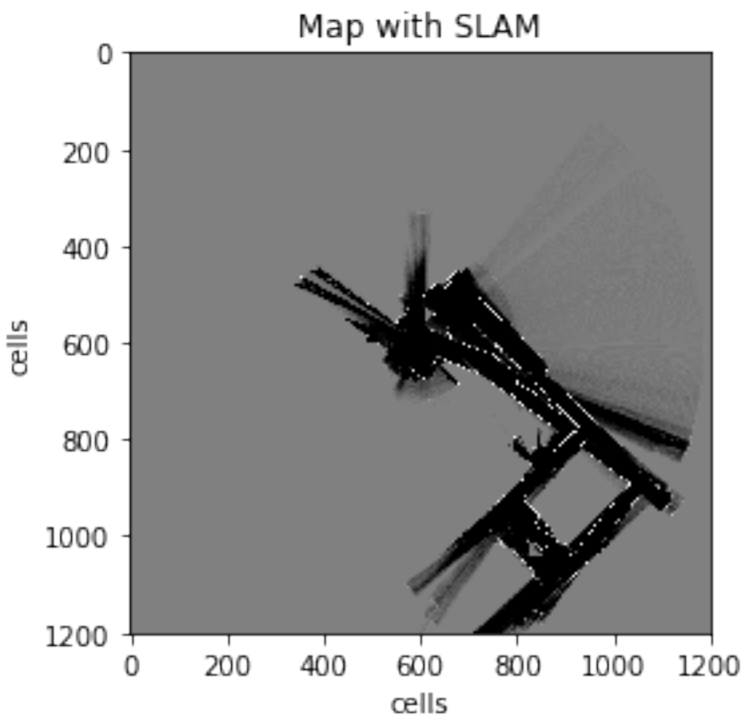
Map24:


Robot Path
Y global (m)
X global (m)
25
20
15
10
5
0
−17.5
−15.0
−12.5
−10.0
−7.5
−5.0
−2.5
0.0


Map
cells
cells
0
200
400
600
800
1000
1200


Map with Robot Path
0
200
400
600
800
1000
1200
cells
0
200
400
600
800
1000
1200
cells


Robot Path with SLAM
20
15
10
5
0
Y global (m)
0
5
10
15
20
X global (m)



We can see that the non-SLAM maps do a fairly good job of visualizing the space, although we can see clear ghost hallways and corners that could be sharper where a variety of real-world scenarios could have affected the readings, such as a wheel slipping, uneven terrain, noisy or otherwise distorted readings, etc. The SLAM results produced are less than optimal, where many of the maps look very similar to the single particle version or have actually rotated and created additional ghost hallways and distortions. This is likely due to a poor optimization one of the hyperparameters, such as the noise added to the encoder readings, number of particles, and resampling criteria. While I tried continuing to tune these parameters, particularly the noise and resampling/reweighting criteria, due to limited time, I was unable to find the key values to produce the desired map results. For example, I found that initially, the standard deviation value in my gaussian noise distribution was too high such that the particles required resampling nearly every iteration, however if the added noise values are too small, the particles would all have close to equal weights and there would be little benefit to adding the particle filtering step at all. I decreased the noise parameter so that the particles ended up resampling approximately every 10-100 time iterations, however, I saw no improvement in the map. With more time, I would first more thoroughly debug each step of the algorithm to make sure that the code written is representing the intended algorithm, and then if this is all correct, I would try tuning other parameters in addition to the noise, such as number of particles, max/min log-odds cap, the log-odds values used to increment the map, the map resolution, and the resampling threshold.