WHO TOOK MY NEATO!?
QUANTITATIVE ENGINEERING ANALYSIS 2 FINAL PROJECT

James Jagielski, Krishna Suresh, and Evan Lockwood

May 10th, 2022

ABSTRACT

The objective of this project was to drive a NEATO robot through a contained area from a start position to an object in that area. For this project that was the Barrel of Benevolence. The rest of the objects in the contained area were obstacles that the NEATO had to avoid.

1. INTRODUCTION

We decided to challenge ourselves with all of the provided levels in the project as well as expanding to more modern SLAM and motion planning algorithms. For the first level, we started by characterizing each object as a single point allowing us to make a simple equation for the potential field. We approached the second level by using RANSAC to fit points to lines found on LIDAR scans. We then added each of those fitted lines to the overall function to build the potential field. Next, for Level 3 we took a similar approach to Level 2 but used a circle fitting version of RANSAC to estimate the goal position. For Level 4 we decided to take a more modern and optimal approach to motion planning by using the RRT* algorithm to create a waypoint list to the goal. Then we used a pure pursuit controller to follow the waypoints to reach the goal. Finally we reached the inspiration for the title of this report: The Kidnapped Robot Problem. For Level 5 we aimed to emulate a more realistic scenario where the robot does not know its starting position, rather has to estimate it by maximizing the probability of its position given the current LIDAR measurement. To accomplish this we implemented a Particle Filter which localizes the NEATO and then uses the same approach as Level 4 to navigate to the target.

2. LEVEL 1

2.1. Mapping and Planning

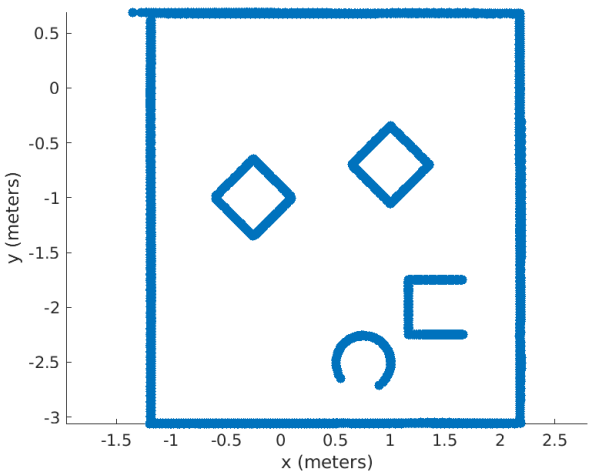

Fig. 1. Level 1 LIDAR scans mapped to global frame.

We took LIDAR scans from the reference frame of the LIDAR to the NEATO reference frame and then back to the global reference frame. The global reference frame is a fixed frame that allows us to model the area.

2.1.1. Point based objects

To represent objects we used a source defined by the equation:

$$v(x, y) = -ln(\sqrt{(x-a)^2 + (y-b)^2})$$

To represent the goal we used a sink defined by the equation:

$$v(x, y) = ln(\sqrt{(x-a)^2 + (y-b)^2})$$

Where a and b represent the source location in the global frame.

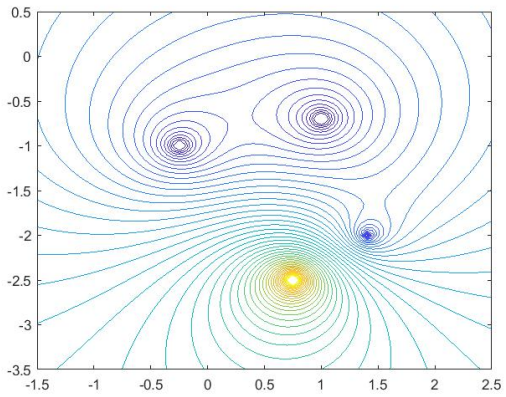

Fig. 2. Contour of the point based objects.

We initially approached this problem by taking the center point of each object and taking the sink equations and source equations to create one function that describes the entire area. This allowed us to simplistically model the room. For the first level this was the most effective way we found to reach the barrel in the arena.

2.1.2. Discretizing a line by n points

For each line that characterized pieces of objects we found the start and end points and then looped through the discrete points in between them. Since all the lines were linear we were able to use the equation,

$$y = mx + b, \tag{1}$$

which meant we could fit all the lines and find the coordinates of points to use the sink equation on. This allowed us to more accurately create obstacles and set us up for the next levels with the LIDAR scans.

2.1.3. Integrating a parameterized line

Below is the equation we used for creating a source for a line. We parameterized the line by integrating over the length of the line.

$$p(a_1, a_2, m, k) = -\int_{a_1}^{a_2} ln(\sqrt{(x-u)^2 + (y-um+k)^2})du$$

Where m and k represent the slope and y-intercept of the line respectively and a_1, a_2 represent the x values of the endpoints of the line.

For a vertical line, we shifted the parameterization to be over the y center point and fixed the x value at c:

$$p_v(b_1, b_2, c) = -\int_{b_1}^{b_2} ln(\sqrt{(x-c)^2 + (y-u)^2})du$$

2.1.4. Potential map of room

Below is the overall equation for the potential map:

$$\begin{aligned} f = &-\ln(\sqrt{(x-1.41)^2 + (y+2)^2}) \\ &-\ln(\sqrt{(x-1)^2 + (y+0.7)^2}) \\ &-\ln(\sqrt{(x+0.25)^2 + (y+1)^2}) \\ &+\ln(\sqrt{(x-0.75)^2 + (y+2.5)^2})) \\ &+p_v(-3.37, 1, -1.5) \\ &+p_v(-3.37, 1, 2.5) \\ &+p(-1.5, 2.5_2, 0, 1) \\ &+p(-1.5, 2.5, 0, -3.37) \end{aligned}$$

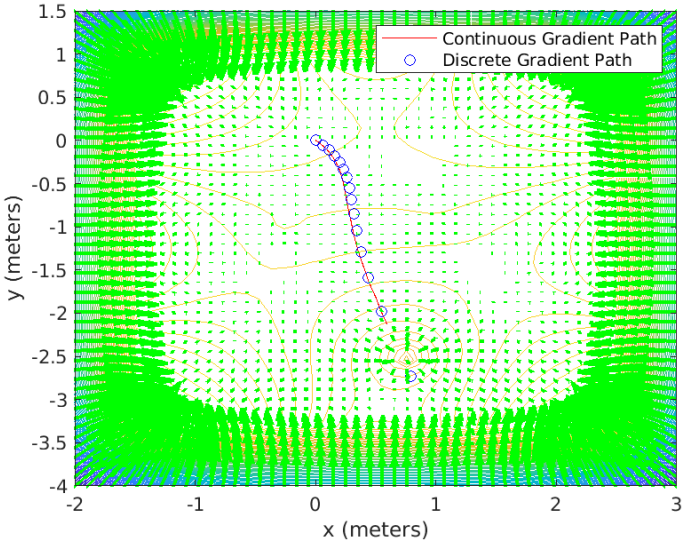

Fig. 3. Level 1 potential field on contour with path of gradient starting at (0,0).

2.2. Navigating

We decided to navigate the paths with a continuous method. This was utilizing what we learned in the previous Bridge of Doom project. This method takes the derivative of a curve at various points to calculate the tangent and normal vectors, which enables us to calculate the velocity and angular velocity as well. The normal vector describes which direction the NEATO's heading is going to point next.

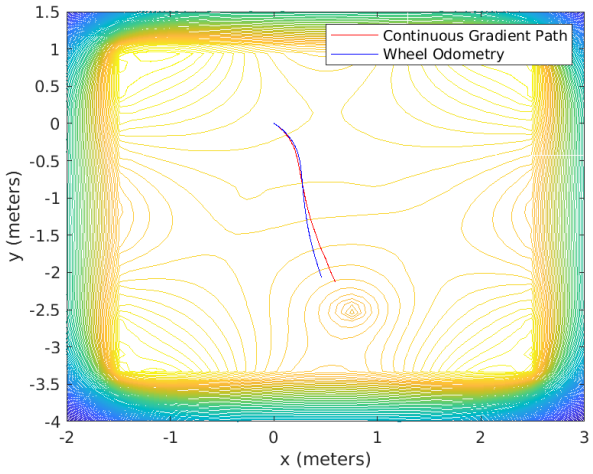

Fig. 4. Level 1 potential field on contour with odometry path.

2.3. Real Gauntlet

Fig. 5. Level 1 gauntlet setup.

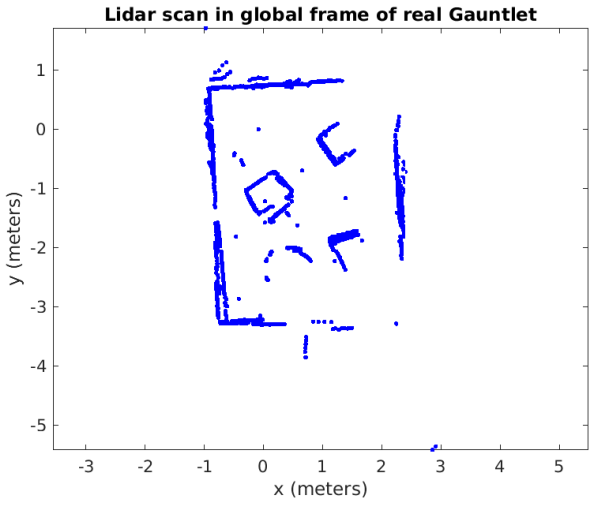

Fig. 6. Level 1 LIDAR scans of the real room mapped to global frame.

3. LEVEL 2

3.1. Mapping and Planning

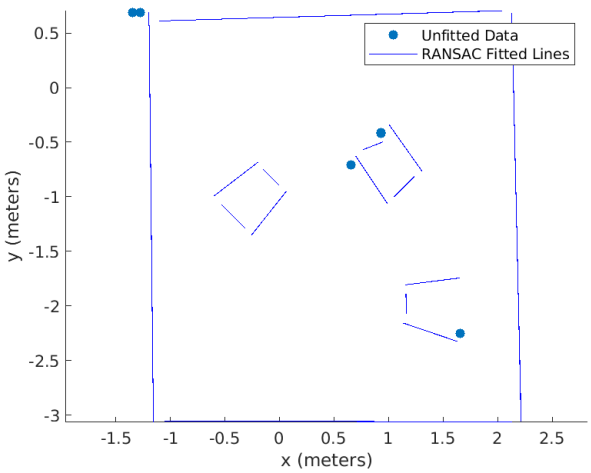

Fig. 7. Level 2 LIDAR scans mapped to global frame and lines fitted via RANSAC.

3.1.1. RANSAC for line detection

The LIDAR scans produce noisy results and we used RANSAC to filter the outliers from the data set. The RANSAC algorithm works by randomly choosing points and fitting a line to those points. We then set a range of error on either side of the fitted line and calculated how many points were in those thresholds. We then choose the fitted line with the most inliers within the threshold. Thus, the outliers for the data aren't considered in the fitting of the lines. We added some features to the RANSAC to be more accurate. We included a max gap between points, which allowed us to distinguish between two objects. Another addition was a maximum on the line length of the RANSAC.

3.1.2. Tuning integrated line

One issue we ran into with integrating over a line was the curved shape of the height of the line. To correct for this we multiplied the line by a scale factor determined by a quadratic.

3.1.3. Dynamic mapping

At each odometry position, RANSAC-filtered LIDAR data was mapped onto the global frame. The intended purpose of this was to created a potential field at each position to map the environment as the NEATO moved throughout the room. However, gathering and analyzing this information at each position was computationally heavy, making this an inefficient method of mapping the room.

3.1.4. *Potential map of room*

The equation for the potential map was created using the lines detected from RANSAC and with the same equations as Level 1.

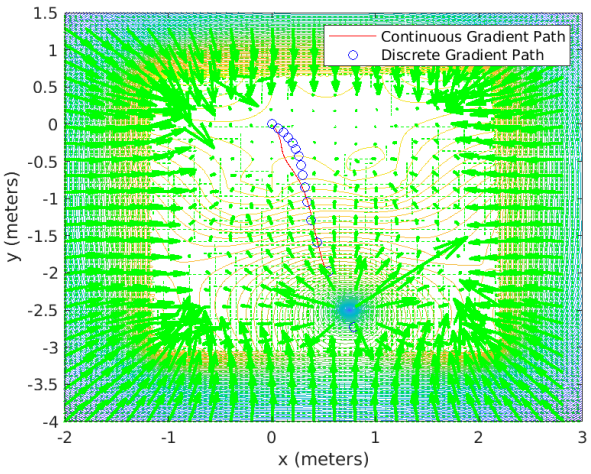

Fig. 8. Level 1 potential field on contour with path of gradient. It took 6 seconds for the simulated NEATO to reach the goal.

3.2. Navigating

The navigating for Level 2 was done in the same method as Level 1.

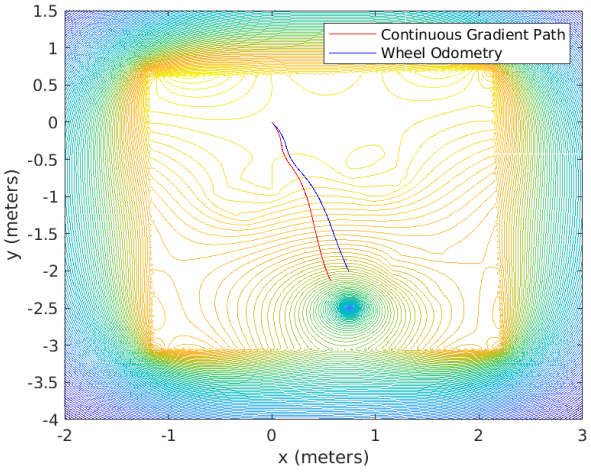

Fig. 9. Level 2 potential field on contour with odometry. It took 6 seconds for the simulated NEATO to reach the goal.

4. LEVEL 3

4.1. Mapping and Planning

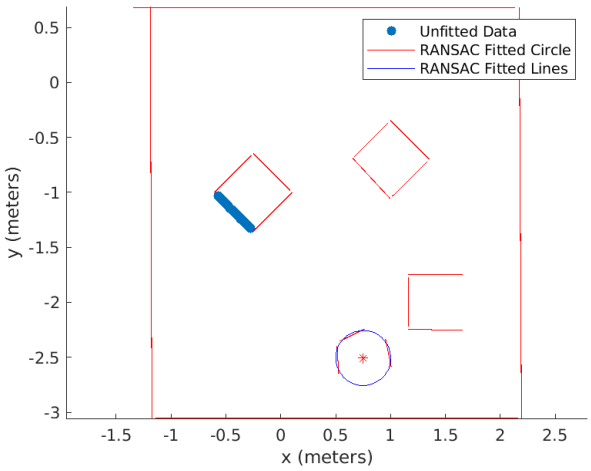

Fig. 10. Level 3 LIDAR scans mapped to global frame with lines and circles fitted via RANSAC.

4.1.1. *RANSAC of circle*

RANSAC of a circle follows the same principles as RANSAC of a line. The RANSAC of a circle randomly chooses three data points and fits a circle to them. There is then a set range around the circle to determine if points are close enough to the fitted line. We then determined the fitted circle that has the most amount of points that lie within the range. The fitted circle with the most inliers is the closest fit to the data set and doesn't take into account any outliers.

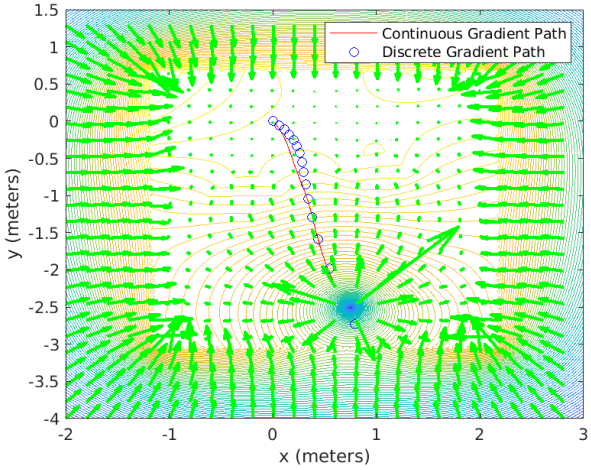

Fig. 11. Level 3 potential field on contour with odometry.

4.2. Navigating

The navigating for Level 3 was done in the same method as Level 1.

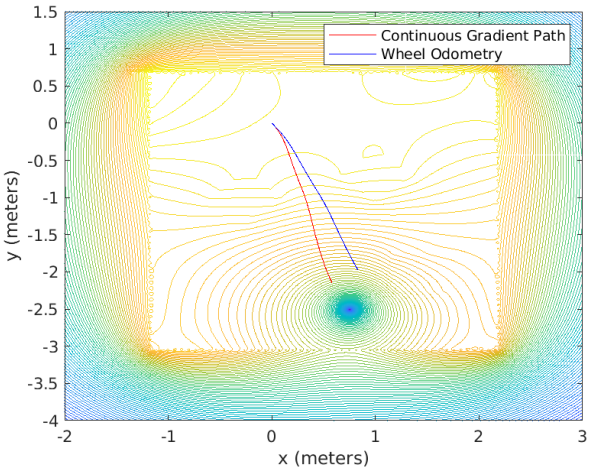

Fig. 12. Level 3 potential field on contour with odometry. It took 6 seconds for the simulated NEATO to reach the goal.

5. LEVEL 4

In Level 4, we build upon Level 1 by using the path planning algorithms RRT and RRT* to generate a list of waypoints from the NEATO's starting position to the location of the Barrel of Benevolence in order to find the optimal path between the NEATO and the end goal.

5.1. Mapping and Planning

5.1.1. RRT

Rapidly-exploring random trees (RRT) is a path planning algorithm used to create a tree of nodes for a robot to travel between in order to locate the shortest path between points A and B. The process is simple: nodes are randomly generated and a path is created between the newly generated node and the next closest node. But randomly generating nodes is time consuming and computationally expensive, so some logic and parameters are put in place to prevent any redundancies or points that are clearly impossible/inefficient to reach. For example, a maximum connection distance is set to make sure that nodes are not expanding too far into undesirable areas and if there is an obstacle between the new node and the next nearest node, then the node is not added. RRT is computationally fast compared to other path planning algorithms such as A* due to the maximum connection distance parameter, which decreases the amount of nodes necessary to find a valid path. However RRT is not the most optimal way to find the best path to the goal. Nodes are being randomly generated, so while it essentially guaranteed it will find a valid path between A and B, the chances of it beings the most optimal are slim.

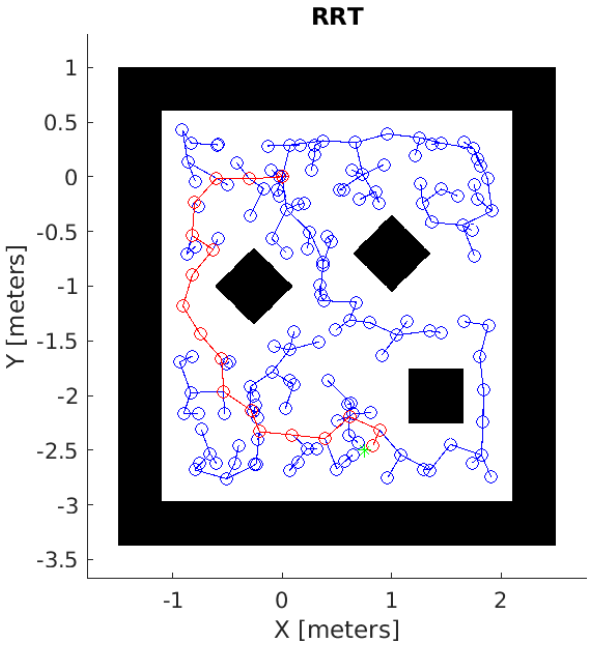

Fig. 13. Level 4 RRT Graph

*5.1.2. RRT**

RRT* builds upon the RRT framework and allows for an asymptotically optimal algorithm by incurring a higher computational cost. RRT* addresses the issues with RRT by allowing for edge rewiring to minimize the cost to each node. This is computed by finding each node in a specified search radius and checking if a re-connection allows the node to have a shorter distance to the start.

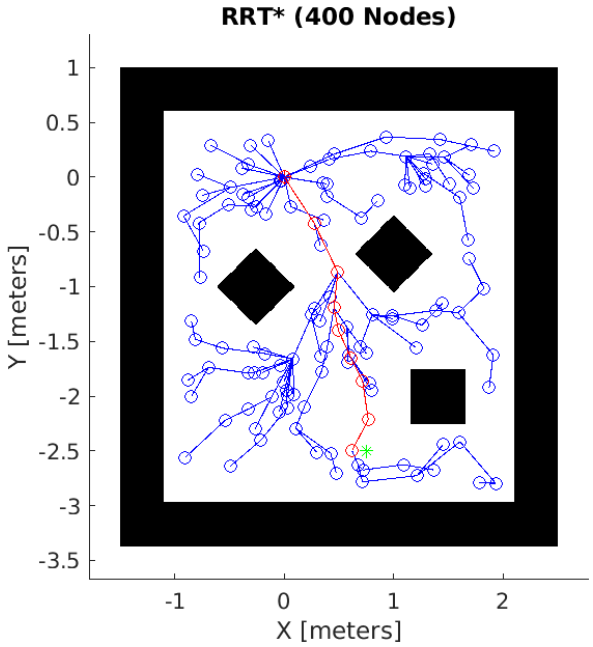

Fig. 14. Level 4 RRT* Graph

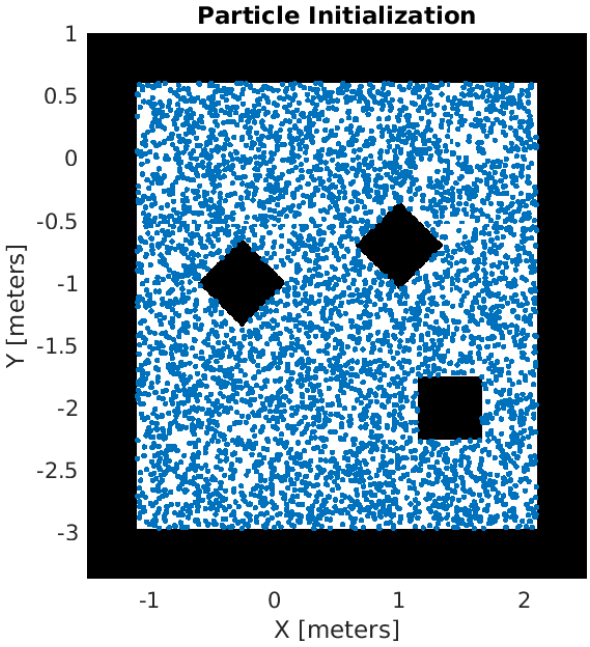

Fig. 15. Particle Filter Initialization, the starting state of the particles in the environment all equally weighted.

5.2. Navigating

5.2.1. Pure pursuit controller

Pure pursuit is a path following algorithm which attempts to smoothly pass through a list of waypoints by using a following distance to define a pointing vector to the path from the current position.

6. LEVEL 5

In Level 5, we strip the NEATO of its starting position, making it necessary for it to estimate its position based on LIDAR data. Using a Particle Filter and RRT* (from Level 4) to localize and plan the path, respectively.

6.1. Mapping and Planning

To solve this localization challenge, we decided to use a particle filter to approximate the non-linear probability distribution of the robot position given the lidar scan. A particle filter works by first defining a set amount of particles randomly throughout the environment:

Each particle represents a possible location for the robot with equal likelihood. Then a scan is taken of the environment and compared with a simulated scan taken from each particle. This gives a likelihood of whether that the particle is a good estimate of the true position or not and allows us to readjust the weighting of the particles. Next we resample the probability distribution based on the newly computed weights:

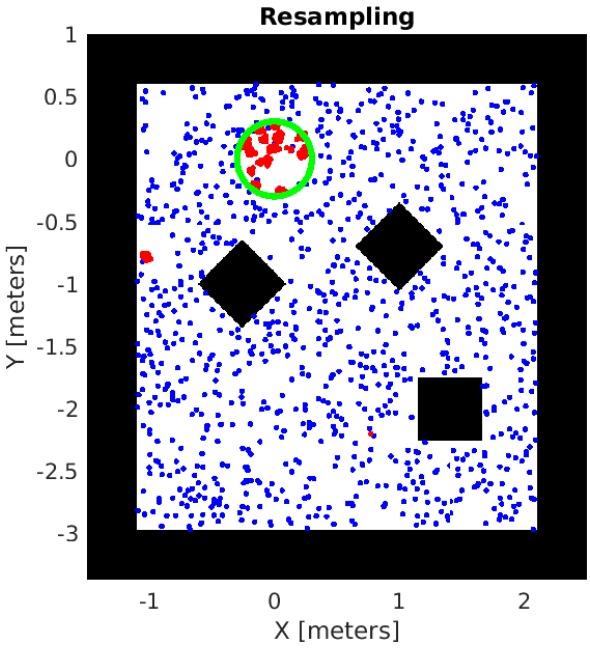

Fig. 16. Particle Filter Resample, particles within the circle are given a greater probability so when the particles are resampled, more end up near that radius.

Next we allow the robot to drive for a specific time and track the relative odometry delta. We then apply this odometry change to each of the particles and repeat the weighting and resampling process. This leads to the particles eventually converging to the true robot location:

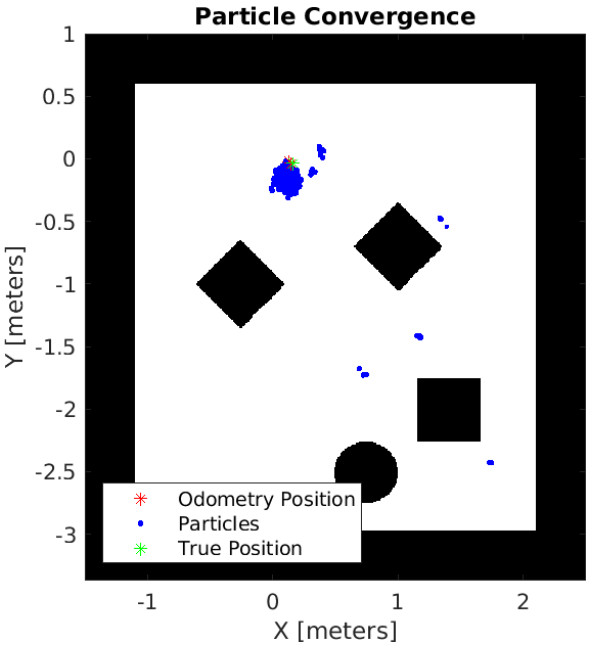

Fig. 17. Convergence: after repeatedly applying the odometry, calculating likelihoods and resampling, the particles converge to the true position of the robot.

6.2. Navigating

After the particles converge we planned to use the same path planning and following algorithm from Level 4.

7. RESULTS

Videos of results can be found at this playlist: `www.youtube.com/playlist?list=PLGzlZQ95yENrl-q4h6xQOG5QZb9IsN_YP`

Funny video: `https://www.youtube.com/watch?v=Iq2PXgALWTE`

All code written can be found at `https://github.com/krish-suresh/qea_2_gauntlet`.

8. REFERENCES

[1] Columbia University, *Differential Drive Robots*, CS W4733 NOTES, `http://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf`.

[2] MATLAB, *Autonomous Navigation Playlist*, `https://www.youtube.com/playlist?list=PLn8PRpmsu08rLRGrnF-S6TyGrmcA2X7kg`.

[3] Cyrill Stachniss, *MSR Course - 07 Particle Filter (Stachniss)*, `https://www.youtube.com/watch?v=uYIjB93oAUo`.

[4] EmaPajic, *TurtleBot-Localization*, `https://github.com/EmaPajic/TurtleBot-Localization`.

[5] Jose-Luis Blanco, *Resampling methods for particle filtering*, `https://www.mathworks.com/matlabcentral/fileexchange/24968-resampling-methods-for-particle-filtering`