Robot Localization

Jordan Croom

Abstract—In this paper, a simulation is presented leveraging an adaptive particle filter (or adaptive Monte Carlo filter) to localize multiple moving robotic vehicles using wheel odometry and lidar sensor inputs. Parameters are presented which allow the simulated robot to successfully navigate to a goal position using the ROS amcl and move_base packages. Using the same parameters, performance is compared between the two robot architectures.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

The problem of robot localization is a well-studied problem with numerous real-world applications. Self-driving car companies such as Waymo, Lyft, Uber, etc. rely heavily on localization algorithms fuse multi-modal sensor suites (lidar, radar, cameras, and IMUs) to maintain the intended vehicle trajectory. Drone companies such as Kespry and DJI use localization algorithms to estimate their position and heading using noisy, low-cost GPS sensors and IMUs. This paper attempts to address a similar but more limited problem in simulation. Given simulated maze made up of traffic barriers, use a lidar sensor and wheel odometry to localize a robot on a known map, and navigate to a prescribed goal location and orientation. By limiting the number and resolution of simulated sensors, some complexity is avoided while still addressing the core issues of the problem.

1.1 The Gazebo Platform

The Gazebo multiphysics simulation engine paired with the RViz visualization platform greatly decreases the complexity of simulating and testing complex physical systems to enable efficient, low-risk algorithm development. Using ROS (the robot operating system), Gazebo is able to emulate arbitrary robot architectures with a number of typical sensors, actuators, and simulated objects with which to interact. ROS, Gazebo, and RViz were leveraged in this case to demonstrate the performance of built-in adaptive particle filter (amcl) by effectively localizing a simple mobile robot and navigating a simple, simulated maze.

2 Background

The presented simulation requires the fusion of sources of position information: wheel odometry and a 2D laser range sensor (lidar). These information from these sensors were used to localize the simulated robot within a known map. Two popular algorithms were considered for the combined fusion and filtering problem, extended Kalman filter, and adaptive Monte Carlo localization.

2.1 Kalman Filters

Kalman filters have been in popular use ever since their heavy utilization in the early space program. The concept is fairly straightforward. By using multiple noisy measurements, each with know probability distributions, the Kalman filter estimates the joint probability distribution of the system state. The joint probability distribution tends to have higher certainty than an estimate of the state from an estimate based on a single sensor alone. The estimate of the current state and covariance estimates are then fed in to subsequent iterations in combination with new measurements. Standard Kalman filters make two key assumptions:

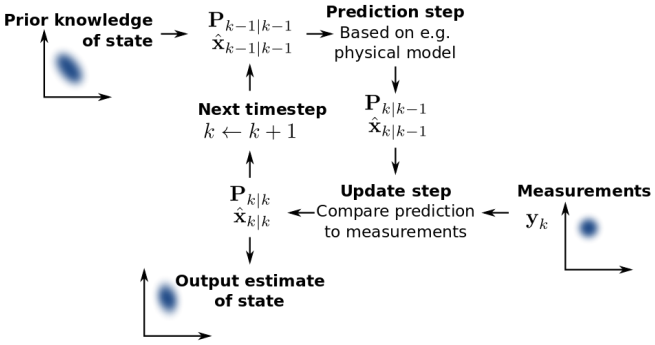

Fig. 1: Visual Outline of the Kalman Filter Algorithm

1) Sensor noise and/or observation uncertainty can be described by a unimodal Gaussian distribution.
2) The process being modeled is linear.

For most real world systems, the assumption of process linearity is too strict. As a result, the extended Kalman filter was developed to relax the assumption of process linearity by using a linearized model of the process to propagate covariance estimates through subsequent iterations of the algorithm. However, the state observations are still required to have a Gaussian noise profile.

2.2 Particle Filters

Particle filters use a number of virtual elements (particles), each representing a potential location of the actual robot (or other system). In the naive implementation, particles are initialized randomly in the first step distributed evenly across the known map. Each subsequent iteration is make up of three processes [1]:

1) **Motion Update**: Each particle's state is adjusted according to the expected (commanded) change in state since the last step (e.g. change in position commanded by the wheel controller)
2) **Sensor Update**: The current sensor measurements are used to estimate the probability each particle represents

the robot's (or other system's) current state. This probability is translated to a weight value for each particle.

3) **Resampling**: N particles are selected from the current batch to be passed to the next iteration. Particles with higher weight are more likely to be propagated to the next iteration than those with lower weight.

The collection of particles is used to form a statistical distribution of the possible states of the system. This procedure requires no assumption of the nature of process or sensor noise (e.g. Gaussian) and has the additional benefit of being adjustable based on the processing capabilities of a given system simply by adjusting the number of particles used. This can even be optimized each iteration based on system load as is done in the adaptive Monte Carlo localization algorithm. In addition, the particle filter is capable of estimating the state of a system in which the initial estimated state is incorrect (a.k.a. the kidnapped robot problem).

2.3 Kalman Filter versus Particle Filter

Table 1 summarizes the properties of Kalman filters and particle filters. Each method has advantages in certain scenarios. The work presented in this paper exclusively utilizes particle filters.

TABLE 1: Extended Kalman Filter vs. Particle Filter

Property	Kalman Filter	Particle Filter
Convergence Speed (Linear Process)	Excellent	Good
Convergence Speed (Nonlinear Process)	KF: Diverges EKF: Possibly Unstable	Good
Computational Efficiency	Excellent	Poor - Good
Sensor Noise	Gaussian Only	Any
Guaranteed Convergence?	Linear Models Only	No
Guaranteed Stable?	No	Yes
Solves Kidnapped Robot Problem?	No	Yes
Implementation Difficulty	High	Low

3 Simulations

3.1 Achievements

Through modification of the relevant AMCL and costmap parameters, two simulated robots repeatably navigated to the goal position. The differences and similarities between these two simulated platforms are explored in subsequent sections.

3.2 Benchmark Model (Udacity Bot)

Udacity bot is comprised of a rectangular chassis, fore- and aft-mounted frictionless casters, and two side-mounted driven wheels. The robot also includes two sensors, a front-mounted RGB camera and a top-mounted +/-90 degree HFOV lidar. The lidar sensor collects point cloud information representing surrounding obstacles that can be used to determine the robot's location on the map and plan a navigable path to the goal position.

TABLE 2: Benchmark Model (Udacity Bot) Summary

Sensors	+/-90 Degree HFOV Lidar Front-facing RGB Camera
Total Mass	27kg
Track Width	0.35m
Wheelbase	0.30m
Configuration	2 Center-Mounted Wheels Fore- and Aft-Mounted Casters
Approximate Robot Envelope Dimensions	0.40m x 0.40m

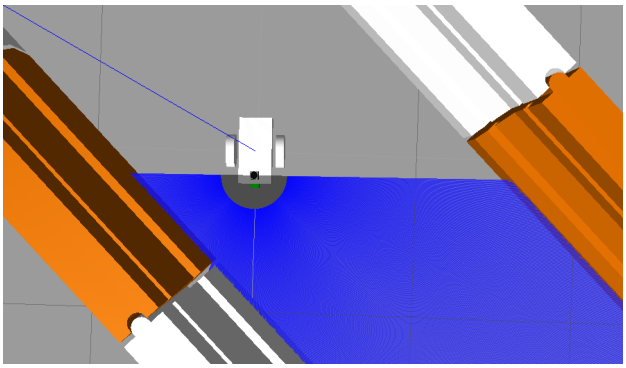

Fig. 2: Top View of Udacity Bot with Blue Lines Representing Lidar Distance Measurements

.

3.2.1 Packages Used

Two primary packages were used to enable the robot to navigate from its initial position to the specified goal position: AMCL (adaptive Monte Carlo localization) and move_base, a localization and navigation tool. move_base itself utilizes a number of independent packages. Of those, parameters for costmap_2d and base_local_planner were modified.

The packages used in the project should be specified as well as the topics received and published; the services it used and provided should also be addressed.

3.2.2 Parameters

Modified parameters for each package as implemented in the Udacity Bot simulations are summarized below:

- AMCL
  - min_particles: 25
  - max_particles: 100
  - transform_tolerance: 0.1
  - laser_min_range: 0.4
  - laser_max_range: 30.0
  - update_min_d: 0.15
  - update_min_a: 0.15
  - laser_max_beams: 120
- move_base
  - costmap_2d
    - overall parameters
      - obstacle_range: 5.0

            - raytrace_range: 30.0
            - transform_tolerance: 0.2
            - robot_radius: 0.4
            - inflation_radius: 0.8
            - observation_sources: laser_scan_sensor
            - laser_scan_sensor: sensor_frame: hokuyo, data_type: LaserScan, topic: /udacity_bot/laser/scan, marking: true, clearing: true

        * global_costmap

            - global_frame: map
            - robot_base_frame: robot_footprint
            - update_frequency: 3.0
            - publish_frequency: 3.0
            - width: 40.0
            - height: 40.0
            - resolution: 0.05
            - static_map: true
            - rolling_window: false

        * local_costmap

            - global_frame: odom
            - robot_base_frame: robot_footprint
            - update_frequency: 3.0
            - publish_frequency: 3.0
            - width: 5.0
            - height: 5.0
            - resolution: 0.05
            - static_map: false
            - rolling_window: true

    - base_local_planner

        * holonomic_robot: false
        * max_vel_x: 1.0
        * max_vel_theta: 3.0
        * controller_frequency: 15.0
        * meter_scoring: true
        * sim_time: 1.0
        * pdist_scale: 1.5
        * occdist_scale: 0.05
        * heading_lookahead: 3.0

While each of these parameters is needed to fully replicate the demonstrated performance, discussion is limited to parameters with particular impact on performance.

3.2.3 Key AMCL Parameters

Given the limited computing power of the platform on which the simulations were run, the number of particles used to in the AMCL algorithm had to be limited. AMCL uses an adaptive algorithm that dynamically assigns the number of particles instantiated at each iteration. Through experimentation, an upper limit of 100 particles and a lower limit of 25 particles was found to be a reasonable compromise between computational load and accuracy.

The transform tolerance was another key parameter in reducing computational load. By allowing the estimated pose to be maintained for 0.1 seconds (10 Hz update frequency), the AMCL algorithm was able to complete each iteration without timing out.

The min and max ranges of the laser at which a detected point would be added to the costmap were set to 0.4m and 30m, respectively. The 0.4 meter lower limit ensured the robot's structure would not be mistake for an obstacle. The 30m upper limit essentially implies no cap on the maximum detected range given the small size of the simulation map.

To maximize the accuracy of the AMCL particles, laser_max_beams, update_min_a, and update_min_d were particularly important. laser_max_beams sets the maximum number of lidar sensor evenly spaced beams are used to update the particle filter. As expected, increasing the number of detected points significantly reduces the covariance of the position and orientation estimates from the particle filter. 120 beams was experimentally determined to be a good compromise between increased accuracy and computational load. update_min_a and update_min_d set the minimum angular and translational motion, respectively, required before a particle filter update. Since the wheel odometry output is relatively noisy and inaccurate, reducing the motion required between lidar updates significantly improves particle covariance. If the wheel odometry output were prefiltered, this effect with likely be less pronounced.

3.2.4 Key Costmap Parameters

Obstacle_range and raytrace_range define how measurements from the range sensor are used in updating the map. Setting the obstacle range to 5 allows obstacles up to 5 meters away from the sensor to be added to the costmap. The raytrace range of 30 meters means the robot will clear the costmap if any point previously marked as an obstacle is determined to be clear and is less than 30 meters from the current sensor location.

The robot_radius and inflation_radius parameters are used to determine what locations in the costmap are safely passable. The inflation radius (0.5 meters in this simulation) sets the distance away from a detected obstacle the path is deemed perfectly safe. Closer distances are negatively weighted in path planning with a magnitude in accordance with their proximity to the obstacle. The robot radius (in this case, 0.4 meters) simply provides an additional offset to account for the non-zero radius of the robot.

For both the global and local costmaps, a number of parameters were adjusted to reduce the computational load of the simulation. For this simulation, each costmap was updated and published at a rate of 3 Hz. This was experimentally found to be a reasonable compromise between performance and computational load.

The static_map and rolling_window parameters define how each map is updated and treated as the robot moves about the map. In the case of the global costmap, static_map is defined as true so that it is initialized based on the predefined simulation map. For the local costmap, this parameter is set to false so that it is initialized based on actual sensor readings. The rolling_window parameter defines whether the cost map region of interest moves with the robot or now. For the local costmap, this is set to true since the robot should keep track of its immediate surroundings to update the local costmap as it advances around the map.

3.2.5 Key Base Local Planner Parameters

The base local planner parameters set restrictions on what sort of paths to the goal position are acceptable. First, holonomic_robot is set to false since the differential-drive robot cannot move in any direction. Max_vel_x and max_vel_theta are defined at 1.0 m/s and 3.0 rad/s to allow for fairly quick navigation of the map toward the goal position. heading_lookahead was set to 3.0 meters

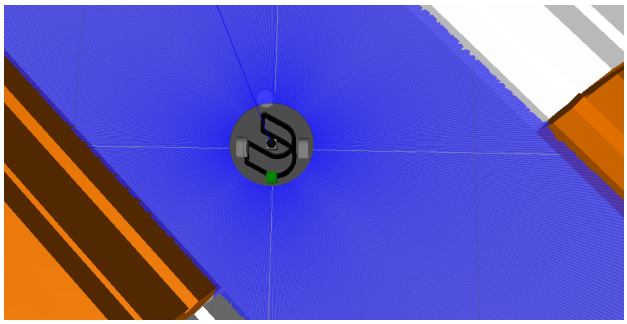

Fig. 3: Top View of Ultra Udacity Bot with Blue Lines Representing Lidar Distance Measurements

.

so that the robot would have adequate time to adjust heading, slow down, or stop given its commanded max velocity of 1.0 m/s.

Other parameters define how the planner should weight different properties of potential paths. pdist_scale defines how proximity to the nominal planned path should be weighted in the robot controller. A value of 1.5 was found to give reasonable path following results while also avoiding oscillations about the nominal path as the robot proceeds toward the goal position. occ_dist scale defines how much the controller attempts to avoid obstacles as it advances along the planned path.

Other parameters were again selected to be compatible with the processing power constraints of the system. sim_time defines how far ahead in seconds to simulate the path during each navigation iteration. controller_frequency sets the frequency at which the controller attempts to provide control inputs to the robot (e.g. adjusting velocity).

3.3 Personal Model

3.3.1 Model design

TABLE 3: Ultra Udacity Bot Summary

Sensors	+/-180 Degree HFOV Lidar Front-facing RGB Camera
Total Mass	18kg
Track Width	0.35m
Wheelbase	0.25m
Configuration	2 Fore-Mounted Wheels 1 Heavy Aft-Mounted Caster
Approximate Robot Envelope Dimensions	0.40m x 0.40m

Ultra Udacity Bot builds on the Udacity Bot design with a few modifications. Most importantly, the chassis styling has been upgraded. Secondly, the lidar unit has been modified to return the same number of point measurements as the original Udacity Bot but in a +/-180 degree HFOV rather than the original +/-90 degree HFOV. The intention of this modification was to allow for more certainty in robot position enabling the robot to sample a wider range of features. In order to support full +/-180 degree HFOV, the wheel radius is reduced by 50%.

3.3.2 Parameters and Packages Used

The same packages and parameters were used for Ultra Udacity Bot and Udacity Bot.

4 Results

4.1 Localization Results

Ultra Udacity Bot managed to reach the goal position in much less time than the original Udacity Bot. Results are summarized over 10 runs in the Table 4. Note that these performance numbers were gathered with slightly different parameter settings than reported in section 3.2.2. Deviating parameters are listed below:

- AMCL
  - update_min_d: 0.2
  - update_min_a: 0.52 ($\pi/6$)
  - laser_max_beams: 30
- move_base costmap_2d overall parameters
  - inflation_radius: 0.5

Sample runs for each robot configuration can be found at the following links:

- With Deviating Parameters (Above)
  - Udacity Bot Successful Run
  - Ultra Udacity Bot Successful Run
- With Parameters from Section 3.2.2
  - Udacity Bot Successful Run
  - Ultra Udacity Bot Successful Run

Figures 4 and 5 show images of both robot configurations after having reached the specified goal pose.

TABLE 4: Udacity Bot vs. Ultra Udacity Bot Performance

Run	Time to Reach Goal Position (sec)	
	Ultra Udacity Bot	Udacity Bot
1	23.160	38.676
2	23.322	39.153
3	23.332	39.401
4	23.652	31.290
5	21.491	41.289
6	23.962	32.444
7	23.793	35.753
8	23.844	34.626
9	21.671	35.614
10	23.047	35.802

4.2 Technical Comparison

The primary improvements to the modified robot model (ultra udacity bot) in comparison to the benchmark robot (udacity bot) were:

- Improved dynamic stability due to changes in wheel base and mass distribution

- Full 360 degree laser scan view

However, upon futher investigation, Ultra Udacity Bot did not seem to have improved localization accuracy, particle convergence speed, or even a difference in commanded velocity magnitude. The other primary difference between the two robot configuration is mass. In fact, when the mass of Ultra Udacity bot was adjusted to match Udacity Bot by increasing the mass of the rear caster to 16 kg, Ultra Udacity Bot did not move at all, despite the fact that both robots were modeled with zero friction in their casters. This anomaly requires more investigation into the physics of the two models. However, it appears that both robots are equally capable in terms of localization efficacy.

5 Discussion

While one might expect that replacing the +/-90 degree lidar sensor with a +/-180 degree lidar sensor would improve localization convergence speed and accuracy, results in the discussed simulations did not support this assumption. Though Ultra Udacity Bot did reach the navigation target significantly faster than Udacity Bot, this appears to be attributable to the dynamics of each robot rather than localization performance. Both robots shared similar localization performance, which seems to indicate that more comprehensive views of the surroundings with the same sensor are not necessarily sufficient to improve performance. Instead, it is possible that improved odometry may increase localization accuracy.

Some investigation into the kidnapped robot problem was also performed with very poor results. It is clear that some additional modifications to the algorithms would be required to successfully solve this problem. First, particles need to be initialized randomly across the entire map rather than clustered near the assumed start position of the robot. If the robot were to be kidnapped in the middle of a run, some method for instantiating particles far away from the current set would also need to be developed. Finally, the current navigation algorithm does not begin moving away from its current position unless a path can be found from its current position toward the goal position. In some cases, misalignment with the assumed global costmap could result in a trapped robot. An additional mode of recovery would need to be developed to command the robot to randomly search around the map until it can find a path to the goal. The current method of rotating in place is insufficient.

In general, these simulations provide a simplified view of a large number of applicable real-world scenarios. MCL/AMCL could be used in a similar manner with relatively few modifications to provide solutions for warehouse robots, hospital supply robots, vacuum robots, etc. In essence, the presented methods would be a good start for any problem involving a well-known local map and a fairly static, well-controlled environment.

6 Conclusion / Future work

In this report, two robotic platforms with integral lidar sensors were evaluated in a simulated localization and navigation challenge. While one robotic platform performed significantly better in this task, this result appears to be attributable not to differences in sensor architecture but to differences in dynamic properties between the two models. The addition of a +/-180 degree lidar sensor did not noticeably improve particle filter convergence speed or overall accuracy when compared to performance with a +/-90 degree lidar. However, both platforms performed sufficiently reliably that this basic architecture should be capable of deployment into similar real-world scenarios. However, modifications would be required to improve reliability and recovery in unexpected scenarios. Future work will continue efforts to improve overall localization performance. One promising avenue is the integration of an extended Kalman filter on the odometry result combined with the integration of an IMU sensor. With increased odometry accuracy, the convergence speed of the particle filter is expected to converge more quickly. Additionally, a more powerful simulation platform (i.e. a better computer) would allow for further exploration of the algorithm parameters as memory and computation constraints would be reduced.

References

[1] S. Thrun, D. Fox, W. Burgard, and F. Dallaert, “Robust monte carlo localization for mobile robots,” *Artif. Intell.*, vol. 128, pp. 99–141, May 2001.

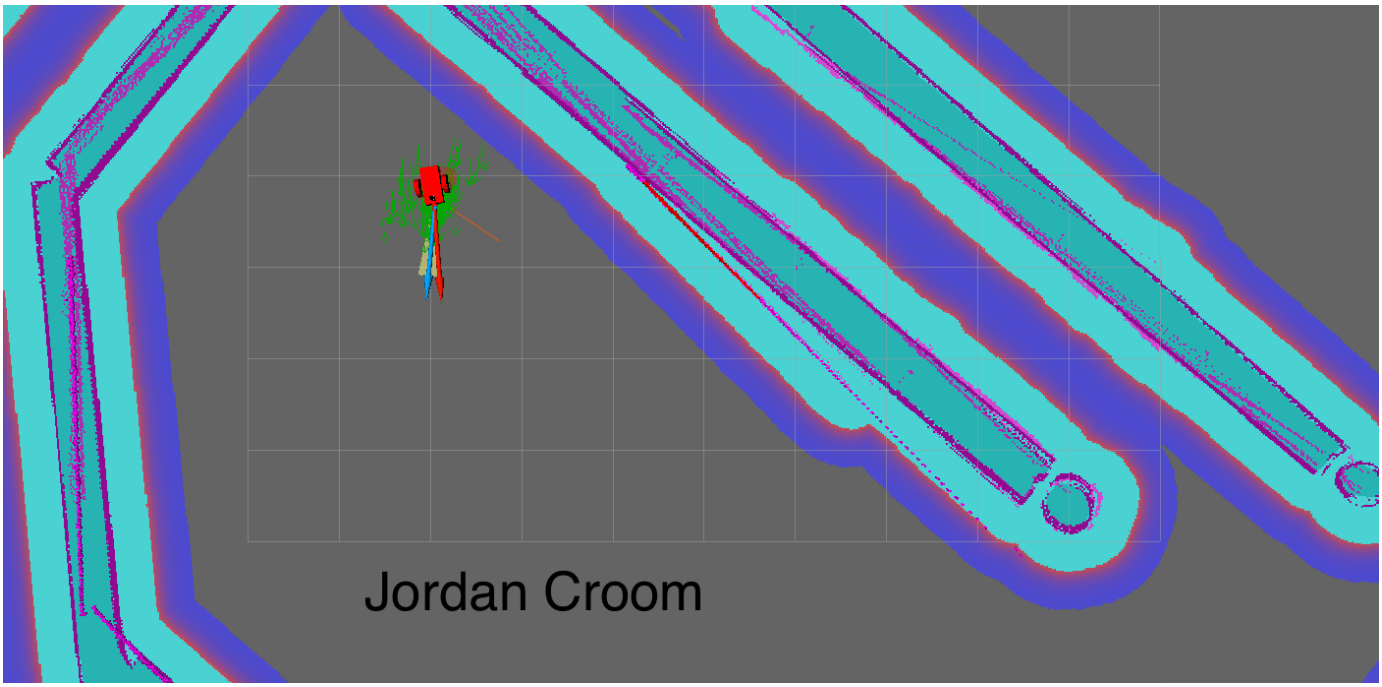

Fig. 4: Udacity Bot at the Goal Position

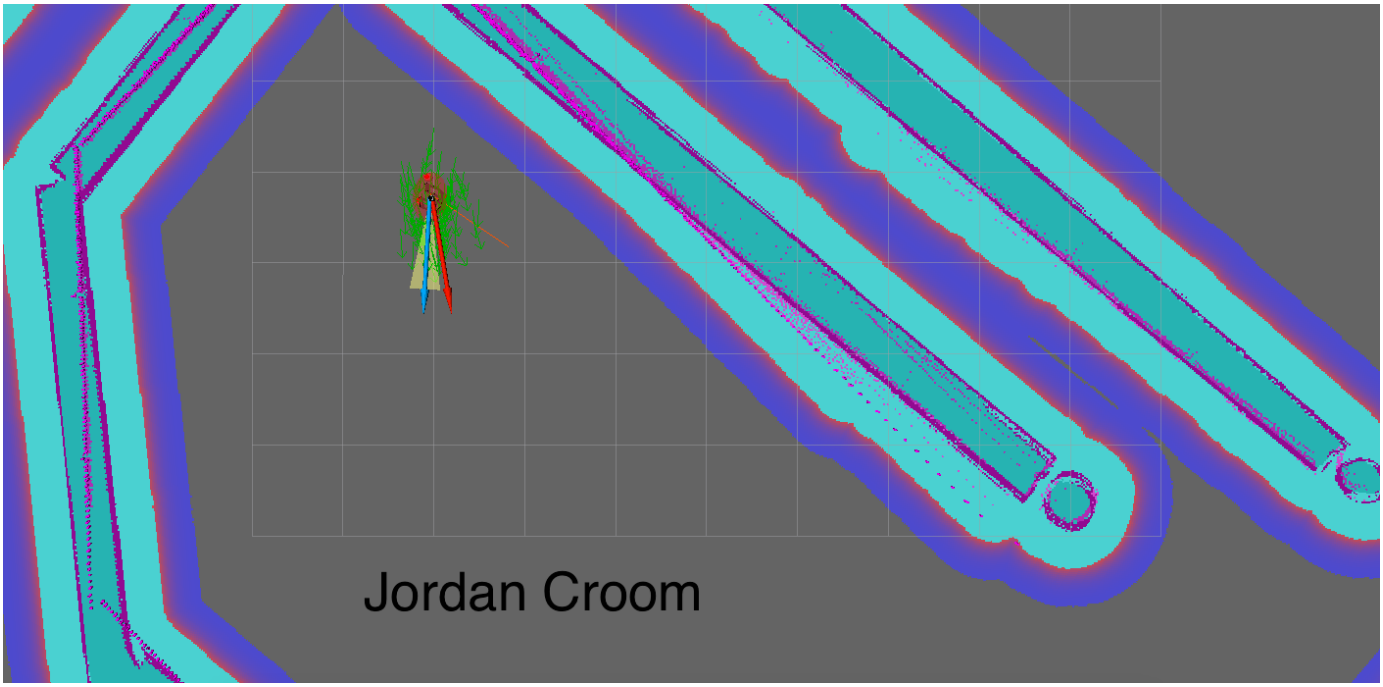

Fig. 5: Ultra Udacity Bot at the Goal Position