RoboND: Robot Localization Project

Matthew Dewar

Abstract—In this project two mobile robot models are created and evaluated within a maze environment to be able to navigate to a predefined goal position. Both robots contain a camera and a laser scanner to process its environment for use in the navigation stack. Adaptive Monte Carlo Localization (AMCL) is used on both robots to localize within the global environment. Parameters of both the AMCL, and the Move-Base packages are tuned to optimize navigation performance.

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

1 Introduction

A critical ability for a mobile robot is for it to be able to locate itself within an environment, asking the question "Where am I?". While localization may be fairly trivial in a perfect environment with perfect sensors, in the real world noisy sensors with large uncertainties, and compounding errors from odometry data drastically complicate the problem. Moreover, multiple sensors will most likely need to be used and compounded in order to achieve an accurate enough sensor measurement, labeled as sensor fusion.

The three most common problems in localization are position tracking, global localization, and the kidnapped robot problem. Position tracking involves the robot knowings its initial position, and then estimating its pose as it moves around the environment [1]. With global localization the robot's initial pose is unknown and it must determine its position relative to the ground truth map. The global localization problem is harder than the position tracking problem. Lastly, the kidnapped robot problem is the same as the global localization problem, however the robot may be kidnapped at any time, and replaced within the environment.

For this project the global localization project will be addressed creating a program using the Robot operating System [2], along with the Adaptive Monte Carlo Localization (AMCL) package for localizing, [3], and the move_base package for interacting with the navigation stack on the robot [4]. This program will be able to globally localize two different robots within a supplied environment (Figure. 1), and navigate the robot to the goal position determined by Udacity.

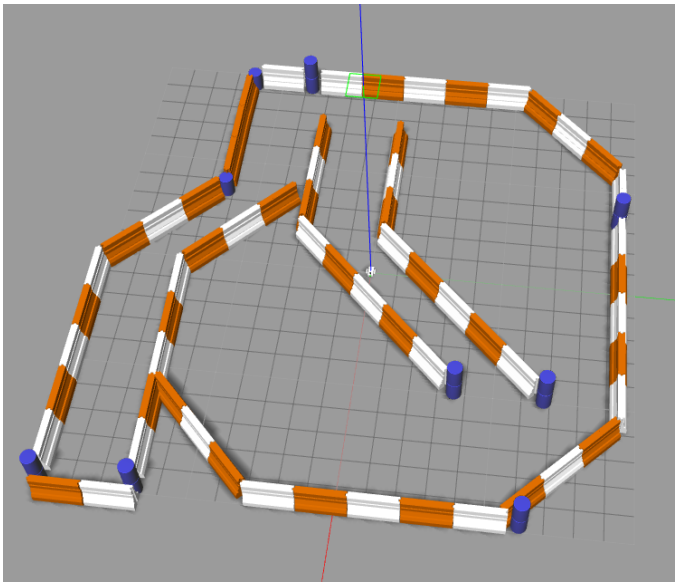

Fig. 1. Environment for testing the navigation and localization stack for both the benchmark and custom robots.

2 Background

The typical navigation stack for a mobile robot consists of odometery and sensor data used in conjunction with a ground truth map, which is then fed into the move_base node which outputs motion commands to the base controller to move the robot (see Figure. 2). Inside the move_base node, local and global costmaps are developed and divided into unoccupied regions, walls, and obstacles. An optional localization node can then be added into the navigation stack to locate the robot within the ground truth map. Porting this stack over to a real world robot, localization will also help with the noisy and uncertain data the robot will obtain. The two most used algorithms for localization are the Extended Kalman filter (EKF), and the Adaptive Monte Carlo Localization (AMCL).

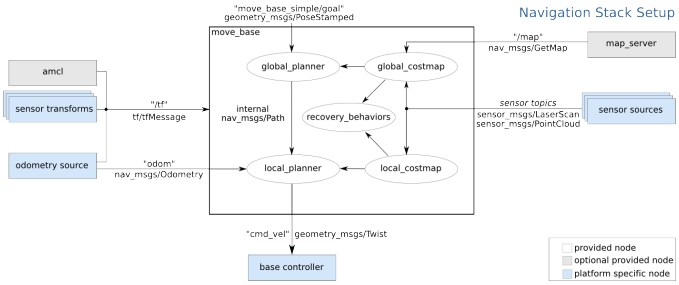

Fig. 2. Navigation stack for the robot, using odometry and sensor data along with AMCL into the move-base node where the robot sensors and the ground truth map are used to create a global and local costmap where velocity, and orientation commands are then sent to the base controller for motion.

2.1 Kalman Filters

The Kalman filter is an algorithm commonly used in guidance, navigation, and control systems for filtering out statistical noise and other inaccuracies from measurement data, producing an estimate of the systems state [5]. This state estimate is obtained by providing an initial estimate, whereby sensors measurements can gain knowledge about our environment increasing confidence in the system's state (Figure 3), where control actions cause some change in the system's state and a new state prediction is made. In robotics this filtering approach can be used to predict the current location of the robot with appropriate data allocated from multiple range finding sensors (i.e. lidars, and depth cameras). However, the traditional Kalman filter is limited to linear motions and measurement functions, whereby the Extended Kalman Fitler (EKF) is employed to linearize the nonlinear functions approximated via Taylor series [6]. To adapt the EKF to multidimensional cases, Jacobians are employed to linearized the measurement and state covariances.

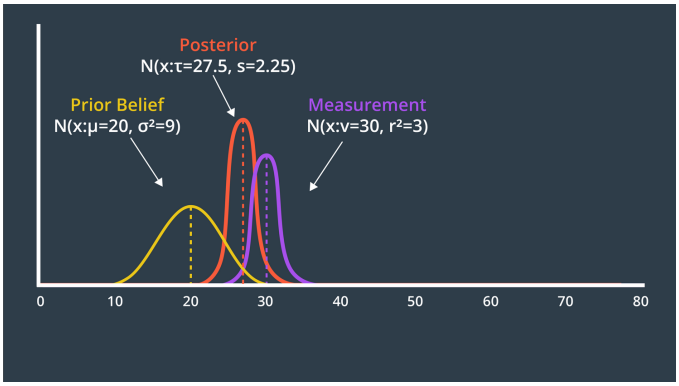

Fig. 3. A prior belief of a robots location can be improved by providing a measurement update to the prior belief of the system, where a more confident posterior belief can result.

2.2 Particle Filters

In Monte Carlo Localization the robots position and orientation are represented by a set a particles distributed around the map predicting the state of the robot. The initial pose of the robot can be known or unknown, where a recursive Bayes filter can measure the posterior state through a prediction step, and then a measurement update step. In the prediction step, motion commands are given to the particles and updated on the map. Then in the measurement update step, sensor measurements to mapped landmarks are compared to each particles position. Each particle is then given a weighted probability that it could be the robots current location, and the particles are then redistributed in the next time step based on this probability [7]. The accumulated probability mass of the particles is used as the current pose of the robot at that time step. In Adaptive Monte-Carlo Localization (AMCL) "the number of particles are adapted over time using Kullback-Leibler Divergence Sampling (KLD-Sampling) to determine the number of particles that are required at each time step" [7]. After many time steps the posterior state of the particles should become more confident and cluster around a single location, given an accurate estimate of the current robot's pose, Figure 4.

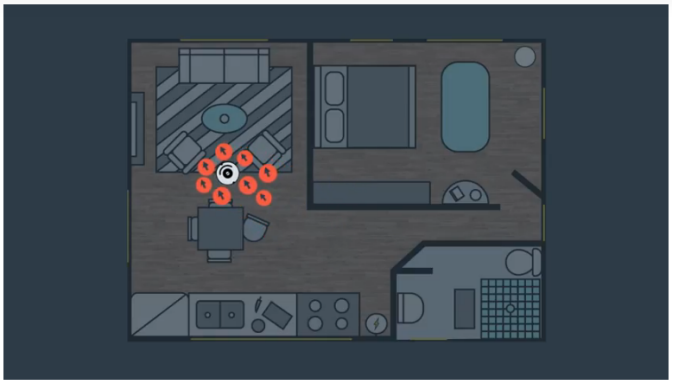

Fig. 4. Illustration of AMCL after multiple time steps where the particles are seen to converge and mass upon a single location on the map.

2.3 Comparison / Contrast

The AMCL can be seen to have many advantages over the Extended Kalman Filter, illustrated in Figure 5. The EKF is limited to a linear Gaussian state space, where the AMCL is not. The AMCL is easier to implement, and the computational memory and resolution of the AMCL can be controlled by tuning parameters related to particle quantity and distribution. The EKF on the other hand is useful for systems that require higher resolutions, and limited in computational power. In this paper AMCL will be chosen as the localization algorithm due to its ability to handle nonlinear states.

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	✓	✓✓
Efficiency(time)	✓	✓✓
Ease of Implementation	✓✓	✓
Resolution	✓	✓✓
Robustness	✓✓	x
Memory & Resolution Control	Yes	No
Global Localization	Yes	No
State Space	Multimodel Discrete	Unimodal Continuous

Fig. 5.

3 Simulations

Within ROS, Gazebo was chosen to simulate the environment which the robot will navigate. A Premade environment located in */maps/jackal_race.world* will be used to test the robot's navigation abilities. RViz was then used to visualize both the robot, and the sensor measurements, as well as to view the particle pose array created with the AMCL package, and the cost maps developed with the Move_Base package. The proper RViz configuration, and gazebo environment will load by issuing the command:

```
roslaunch udacity_bot udacity_world.
    launch
```

For the robot supplied by Udacity, and:

```
roslaunch tank_bot tank_world.launch
```

For the custom robot made by the author.

Tuning parameters for the local planner, and cost maps used in the navigation stack can be found in the $/config/$ folder for both the udacity_bot, and tank_bot repositories. They are broken up into four files:

- local_costmap_params.yaml
- global_costmap_params.yaml
- costmap_common_params.yaml
- base_local_planner_params.yaml

Tuning of the parameters located in these files will be discussed below and in subsequent sections.

URDF files for the baseline robot model and the custom model are found in the $/urdf/$ file for each robot, where they are broken up into a $/.xacro/$, and a $/.gazebo/$ file in order to define simulation parameters for the drive controller, camera, and laser range finder.

3.1 Achievements

The baseline robot successfully completed moving to the navigation goal in 1 minute and 47 seconds. Multiple runs of the navigation goal script were tested, and were all successful. 2D nav goals around the map were tested, with the baseline robot successfully moving around the environment.

3.2 Benchmark Model

3.2.1 Model design

The Udacity-bot seen in Figure. 6 consists of a 0.4m X 0.2m X 0.1m rectangular body with two 0.1m radius wheels on either side, and front and back caster balls for stability. Sensors on the robot consist of a front facing camera on the front of the robot's body, and a Hokuyo Scanning Laser Rangefinder on top of the robot. The full description of the robot can be found in $udacity\_bot/urdf/udacity\_bot.xacro$, and $udacity\_bot/urdf/udacity\_bot.gazebo$.

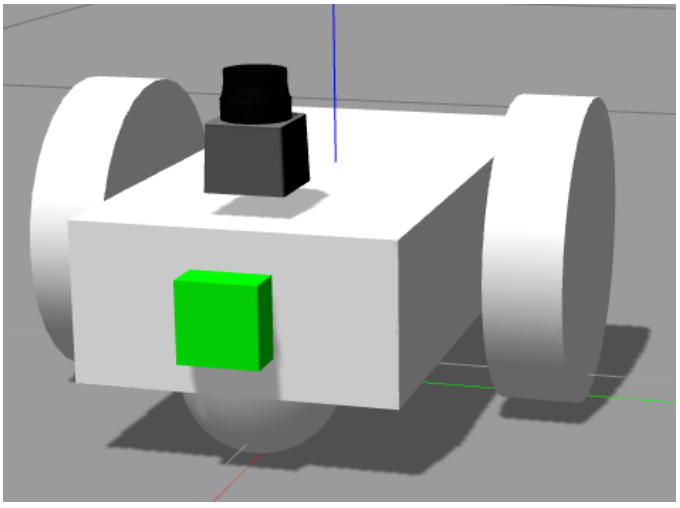

Fig. 6. Udacity_bot rendered in Gazebo.

3.2.2 Parameters

Optimal parameters for the benchmark model can be divided into relevant parameters for the AMCL node, Table. 1, and for the costmaps nodes of the move-base navigation package, Table. 2. Minimizing the number of particle in the adaptive AMCL helped increase the certainty and the performance of the AMCL only after a few time steps. Transform tolerances defining the latency between transforms in both the AMCL, and costmap node were set just high enough to avoid a transform timeout error. The inflation radius was set appropriately for the robot to be able to navigate within tight spaces, but not run into any obstacles. Relatively high values for the obstacle range and inflation range were used to give the robot a high enough field of view to be able to make the best navigation planner. Update and publish frequencies are mainly dependant on system performance and how quickly the costmaps can be updated. For this report, validation and testing was performed on a local machine with a Intel Core i5 CPU at 3.2GHz with 8GB of RAM. A full list of parameters can be found in the $udacity\_bot/config/$ directory.

TABLE 1
AMCL parameters for the benchmark model.

AMCL Node Parameters	
min particles	50
max particles	200
KLD error	0.1
KLD-Z	0.99
transform_tolerance	0.2
recovery_alpha_slow	0.001
recovery_alpha_fast	0.1
AMCL Laser	
laser_min_range	0.1
laser_max_range	30.0
laser_max_beams	50
laser model type	likelihood field
AMCL Odometery	
odom_model_type	diff
odom_alpha	0.2

TABLE 2
AMCL parameters for the benchmark model.

Parameters	**Global**	**Local**
global frame	map	odom
robot base frame	robot_footprint	robot_footprint
update frequency	10	10
publish frequency	10	10
width	5	5
height	5	5
resolution	0.07	0.05
obstacle range	3	3
raytrace range	5	5
transform tolerance	0.5	0.5
robot radius	0.5	0.5
inflation radius	0.6	0.6

3.3 Personal Model

3.3.1 Model design

The personal robot designed for this project, named Tank-bot, was made to provide a platform where this software package could be adapted for outdoor environments where a variable landscape, and objects that the robot could travel over are handled. Tank-bot consists of three wheels on each side with a rounded-bottom body to be able to roll over minor obstacles. Initially the front and back wheels were set higher up the body than the middle wheels to be able to adapt to variable terrain; however, for the environment in Figure. 1 the robot was not able to navigate appropriately. Therefore all of the wheels were set to same height with the differential drive set to the rear wheels. The robot model was created in Blender, where the collision, inertial, and joint objects were created with a robot model creation plug-in called Phobos [8]. The Phobos environment for creating URDF files can be seen in Figure. 7. A camera, and laser scanner were used for robot sensors similar to the benchmark robot with the camera in the front, and the laser scanner on top. Figure. 8 shows the Tank-bot rendered in blender.

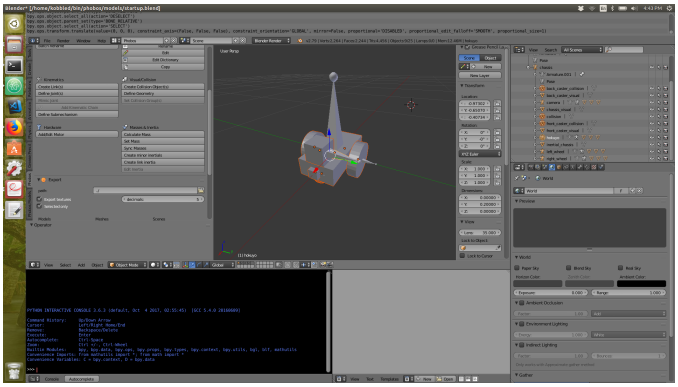

Fig. 7. The Phobos add-on environment for blender used to export URDF files for use as a robot model.

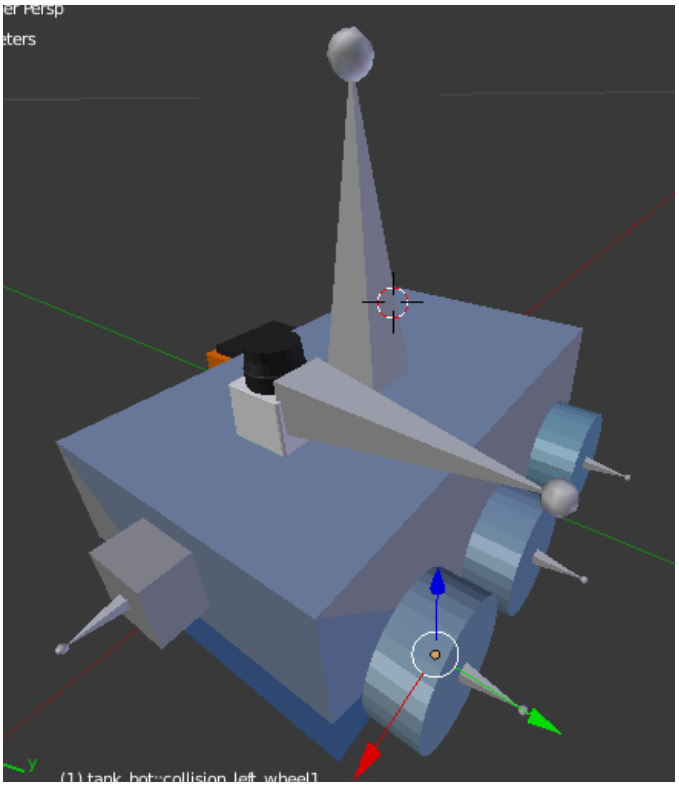

Fig. 8. Tank_bot rendered in Blender using the Phobos add-on to visualize the URDF file.

3.3.2 Parameters

AMCL parameters in 3, and costmap parameters in 4 for the Tank-bot had to be modified compared to the benchmark model in order for the robot to be able to move within the environment. Without the modifications in the parameters provided, the robot was unable to navigate during any 2D nav goal operations. Even with these modification the tank-bot has difficulty localizing and navigation in its environment from its initial pose. A full list of parameters can be found in the $tank\_bot/config/$ directory.

TABLE 3
AMCL parameters for the personal model.

AMCL Node Parameters	
min particles	25
max particles	200
KLD error	0.05
KLD-Z	0.99
transform_tolerance	0.2
recovery_alpha_slow	0.001
recovery_alpha_fast	0.1
AMCL Laser	
laser_min_range	0.1
laser_max_range	30.0
laser_max_beams	50
laser model type	likelihood field
AMCL Odometery	
odom_model_type	diff
odom_alpha	0.2

4 Results

4.1 Localization Results

4.1.1 Benchmark

For parameter validation and testing the 2D Nav Goal option in RViz was used to attempt to move the robot to a given position determined by the user. Figure. 9 shows the use of the 2D Nav Goal option, and where the goal pose was located during testing. Prior to proper parameter optimization, Figure. 9 shows an AMCL particle cloud that is highly uncertain and contains a large error.

Without local planner parameters the robot had hard time navigating down the hallway, zigzagging back and forth down the hall, getting stuck against walls and having to reverse out of them. During testing the local planner was throwing collision errors when the robot was getting too close to walls, however eventually made it to its destination.

Adding AMCL filter parameters, laser model parameters, and odometery model parameters provided drastic increases to the performance of the localization node, and

TABLE 4
AMCL parameters for the personal model.

Parameters	Global	Local
global frame	map	odom
robot base frame	robot_footprint	robot_footprint
update frequency	10	10
publish frequency	10	10
width	5	4
height	5	4
resolution	0.05	0.07
obstacle range	6	6
raytrace range	8	8
transform tolerance	0.3	0.3
robot radius	0.5	0.5
inflation radius	0.5	0.5

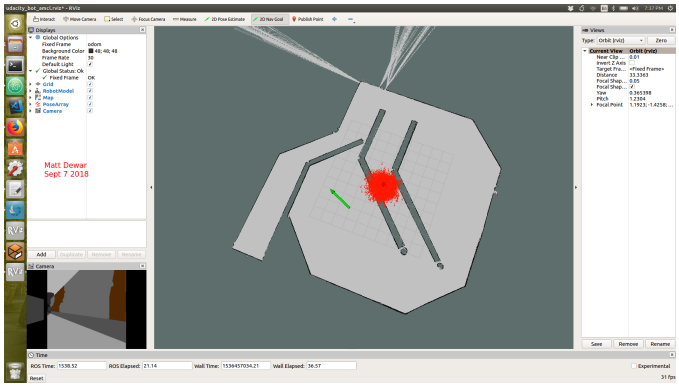

Fig. 9. Running 2D Nav Goal in RViz

the local planner node. Decreasing the amount of particles available for monte carlo localization aided in performance, and helped decrease uncertainty, however also decreased the accuracy of the localization. The KLD sampling were shown to be tuned appropriately where after a certain amount of time sets number of mone carlo particles was minimized, and were seen to be tightly centered around the robots actual pose. Figure. 10 Shows the improvement after adding the AMCL parameters.

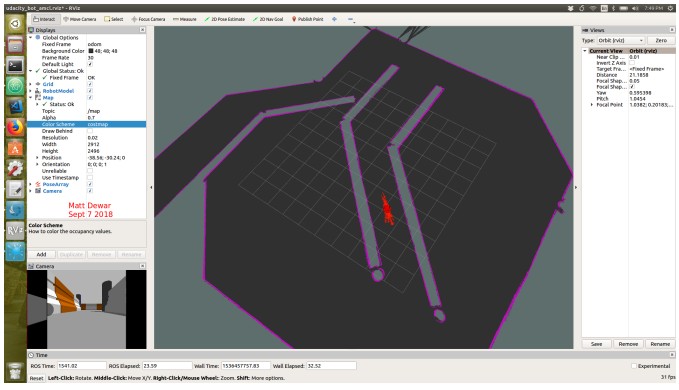

Fig. 10. Localization of benchmark robot after adding AMCL node parameters.

Upon running the *navigationgoal.cpp/* unit test on the optimization configuration parameters the robot was seen to navigate quickly and effectively through the narrow corridor, and was able to turn around the bottom corner obstacle in one smooth motion. However once it reached the proper position the robot had difficulty assuming the correct orientation of the goal pose, rotating around a few times before completing the script. Figure. 11 shows the benchmark robot reaching the navigation goal within 1 minute and 47 seconds.

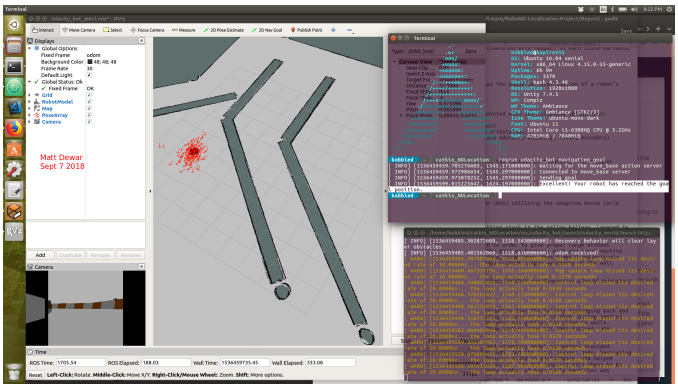

Fig. 11. Benchmark robot reaching the navigation objective set for this project.

4.1.2 *Student*

The Tank-bot had a much more difficult time navigating than the benchmark bot, where numerous iteration on configuration parameters were experimented with, however none yielded successful results. Figure. 12 illustrates some problems with the Tank-bot, particularly with the orientation that the robot starts in, attempting a three point turn within the corridor in order to reorient itself. This namely results in the Tank-bot eventually erroring out, and not completing the navigation goal. Teleop was used to try an identify if there were any intrinsic locomotion problems with the model. While an issues with the robot wheels at different heights causing instability was addressed, handling multiple wheels with the differential drive controller was unable to be solved for this report. Responsiveness of the robot appeared to be hindered by the extra wheel on the side either not stopping on rotation, or not moving appropriately with the given teleop command. Differential drive on all three wheels was investigated with the best response coming from a rear wheel drive configuration. Attempts were made to chain the wheels together in a controller configuration YAML file, however integrating that with the other topics and packages in this software proved challenging and were unsolved at the time of writing this.

5 Discussion

With great success of parameter optimization on the benchmark robot, the assumption that the navigation stack could be easily ported over to other robot platform proved false. The personal robot model was almost completely unresponsive to the parameters of the benchmark model that were able to solve the navigation goal test. At the time of writing this report the personal robot was unable to reach the goal destination. As mentioned in section 4.1.2, problems with

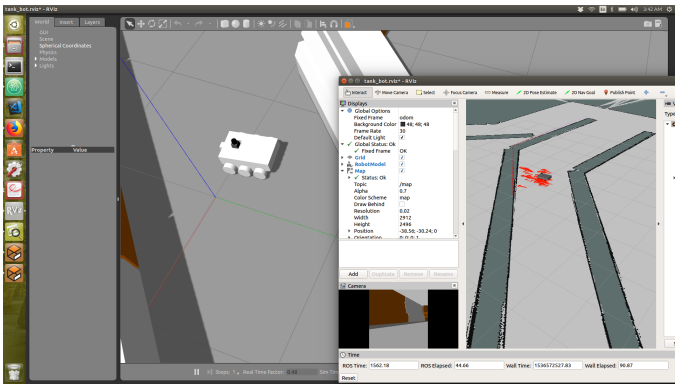

Fig. 12. Issues with the Tank-bot resulted in a solution for the Tank-bot to not be found.

the differential controller on a multi wheel robot were discovered and unsolved. If the diff_drive_controller package can be configured in such a way as to provide control of all the wheels, or if independent control of multiple wheels with a skid steer controller can be used, the navigation problems of the Tank-bot may be able to be distilled down to a locomotion issue.

Turning around in the narrow corridor seems to be the edge case for the navigation stack presented in this project. While turning in the corridor is the Achilles heal for the Tank-bot, the benchmark robot was observed to struggle in the same case, but eventually be able to recover out of it. Parameter testing shows that the inflation radius, obstacle range, and robot radius all need to be minimized in order to be able to rotate within a tight corridor. However after dozens of tests on the Tank-bot appropriate parameters could not be found.

In the kidnapped robot problem the robot is removed the environment, and randomly placed at a new location within the environment. From observations made in this project it can be stated that the AMCL would work well in solving the kidnapped robot problem as it is not completely dependant on the initial position of the robot, and after a few time steps should be able to start localizing.

In an outdoor environment AMCL would be useful in locating objects within a mapped terrain. For example locating boreholes for geological surveying operations is a tedious task that could be automated by way of surveying robots. Moreover, finding missing persons or equipment in a forest or other sheltered terrain could be done with a robot equipt with AMCL.

6 Conclusion / Future work

In conclusion a navigation stack with AMCL localization was successfully implemented on the benchmark robot provided for this project, completing the navigation goal unit test in 1 minute and 47 seconds. A personal robot was developed that was drastically different than the benchmark robot, however the increased complexity of the wheel design proved to be outside of the scope of this project. Tuning of costmap and AMCL parameters greatly increase the ability, and efficiency of the navigation and localization of the robots, however a solution was unable to be found for the personal robot.

Moving to a machine with higher computational resources, particle sizes should be increased to improve accuracy of the localization.

For the personal robot a restructured differential drive controller or skid steer controller should be developed. Once the robot is able to navigate and localize, redesigning the sensors in order to adapt to outdoor environments will be crucial. It is recommended to try modelling two laser scanner at different height (one on the bottom, and one on top), to be able to identify obstacles into categories that the robot can, and cannot traverse. From here an outdoor simulated environment can be tested to evaluate the proficiency of this model for such a case.

References

[1] Udacity, "Lesson 9: Introduction to localization," August 2018.
[2] "Ros wiki." http://wiki.ros.org/. Accessed: 2018-08-28.
[3] "Ros amcl package." http://wiki.ros.org/amcl. Accessed: 2018-08-20.
[4] "Ros move-base package." http://wiki.ros.org/move_base. Accessed: 2018-08-20.
[5] "Kalman filter wiki." https://en.wikipedia.org/wiki/Kalman_filter. Accessed: 2018-08-28.
[6] Udacity, "Lesson 10: Kalman filters," August 2018.
[7] H. Peel, S. Luo, A. Cohn, and R. Fuentes, "Localisation of a mobile robot for bridge bearing inspection," *Automation in Construction*, vol. 94, pp. 244 – 256, 2018.
[8] "Phobos github." https://github.com/dfki-ric/phobos. Accessed: 2018-09-02.