Localization Project Writeup

Gil Meiri

Abstract—In this paper we will present the result of the localization project that is part of Udacity's Robotics Nanodegree Program. The main purpose of this project was to learn about localization and path planning using ROS's AMCL and Move Base packages. These parameters were tuned in order to have a basic benchmark robot navigate to a target position. Next two different robots were challenging robots were created, one extremely wide and another extremely long. These were used to study how robust the the AMCL and Move Base packages are to challenging robot configurations. These models were also used to evaluate which parameters needed to be adjusted to get these new configurations to work.

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

1 Introduction

Robot localization and path planning in a stochastic environment have represented a challenge for robotics researchers and engineers. Noise in sensor and odometry can represent a real challenge even if the the robot is navigated through a pre-mapped environment. Errors and differences between maps and the real world, even if they are really small can also hinder a robots ability to identify its location. Finally a robot might need to operate in a dynamic environment where there could be unexpected obstacles some of them might even be moving obstacles such as a robot operating inside a museum with visitors walking about.

There have been many approaches to try to tackle this problem. One such approach is dead reckoning in which a robot tries to track its movement using only its odometry information. Unfortunately this approach is very limited since drift, wheel slippage and inaccurate map information can quickly cause a robot to loose its ability to reliably track its position. This approach also doesn't take into account an environment with unexpected obstacles.

Another approach is to use an external transmitters or sensors to aid with robot localization. These could include wall mounted cameras, or gps triangulation. In many real world situations there techniques may either not be accurate enough or not applicable in certain cases.

Lastly there exists an approach in which landmarks or beacons are placed, such as QR codes in specific areas of the map. Even though this is a promising technique, it is not always possible to alter the environment to aid a robots navigation mission. This approach is also unable to work in an ever changing, stochastic environment where there could be unknown obstacles.

In this paper we will explore a probabilistic approach to solve this challenging problem. In this approach in every given moment the robot doesn't hold a single deterministic belief of its absolute position. Instead the robot maintains belief model with several possible locations each with a certain probability in case a a discrete model or a probability distribution in case of a continuous model. As the robot navigates this model is updated using new sensory information as it arrives. These techniques are robust and can handle problems such as sensory noise and unknown obstacle obstruction.

2 Background

There are a few algorithms that implement the probablistic approach for localization. The two main algorithms are kalman filters and particle filters. Particle filters are sometimes called Monte Carlo Localization. Each of these has its advantages and drawbacks that make them a better fit for specific problem domains. In this exercise we chose the algorithm called "Adaptive Monte Carlo Localization". In section 2.3 we will discuss why this algorithm was chosen. [1]

2.1 Kalman Filters

Given a movement command a robot may not accurately to the desired position due to things like wheel slippage or errors. This is why the exact position of a robot cannot be tracked. Instead the position of the robot will be given using a probabilistic Gaussian distribution. The environment in which the robot operates effect the variance of this distribution. If a robot was to operate on a factory floor for instance the movement will be more precise and the variance will be smaller as compared to a robot operating in a forest. As robot gives more movement commands without external inputs such as sensory information this uncertainty of the robots absolute position stacks up over time and grows to a Gaussian with a larger variance. Sensory information also has errors usually modeled with a Gaussian distribution.

In the Kalman filter the robot maintains a Gaussian belief distribution to its current location. As the robot is given movement commands and take sensor measurement this belief distribution is updated in two steps. The first is the measurement step which increases the robots confidence in its location. The next is the state prediction caused by the robot movement. This decreases the robots confidence in its location.

During the measurement step a measurement is taken and the internal belief model is updated. The new belief's

mean and variance are calculated according to these equations:

$$\mu' = \frac{r^2\mu + \sigma^2 v}{r^2 + \sigma^2} \quad (1)$$

$$\sigma^{2'} = \frac{1}{\frac{1}{r^2} + \frac{1}{\sigma^2}} \quad (2)$$

Where:
μ': Mean of the new belief.
$\sigma^{2'}$: Variance of the new belief.
μ: Mean of the prior belief
σ^2: Variance of prior belief
v: Mean of the measurement
r^2: Variance of the measurement

During the state prediction step the robots new belief state mean and variance are updated according to these formulas:

$$\mu' = \mu_1 + \mu_2 \quad (3)$$

$$\sigma^{2'} = \sigma_1^2 + \sigma_2^2 \quad (4)$$

Where:
μ_1: Mean of the prior belief
μ_2: Mean of robot motion
σ^2: Variance of prior belief
σ^2: Variance of robot motion

These two steps are continuously cycled as control commands are given. The robot always holds an updated belief of its location.

The basic Kalman filter is used in linear systems in which the output is proportional to the input. In the real worold most systems are non-linear. A extension to the Kalman filter is the Extended Kalman filter which can be used in non linear systems.

The problem with non linear systems is that either the motion models, measurement models or both are non linear. This was an important assumption for the basic Kalman filter. In order to overcome this assumption, in each step the Extended Kalman filter approximates these models using the Taylor series around where we are operating. This is continuously done for each measurement and stare prediction step.

2.2 Particle Filters

One major disadvantages of the Kalman filter is that they only work in state spaces that can be represented by a unimodel Gaussian distribution. In many real world situations this is not the case. This is where particle filters can come in handy. In the particle filter algorithm the robots internal belief is comprised of many particles, each of which represents a possible robot location and orientation. If these particles are distributed in a large area the robot is not confident of its position. As the distribution shrinks the robot is more confident of its location.

The particle filter works in three steps for every movement and measurement. First step is the motion update. In this step every particle is updated with the movement of the robot. The movement given to each particle is drawn from the robot movement probability distribution.

The next step is the measurement step. In this step each particle is given a weight according to how likely the the robot is located in the particle location given the sensory measurement that was acquired.

Lastly we have the re-sampling step. In this step a new set of particles is drawn from the list of previous particles. The chance of drawing a particular particle is the is proportionate the the weight of that particle. The number of particles that are drawn is the same as the number of that were in the previous particle list.

A variant of the particle filter is the Adaptive Monte Carlo Localization. In this algorithm the number of particles that are re-sampled from one cycle to the next changes in accordance to how confident the robot thinks its location is. This is useful because the processing power needed to compute the particle filter algorithm for a large number of particle may be high and hinder performance, which is especially important in a real time system with limited resources. Using the particle filter with a small number of particles may also hinder performance in cases where the robot is unsure of its location.

2.3 Comparison / Contrast

There are some major differences for these two types of filters. The Extended Kalman filter is limited to cases in which the state space can be represented by a unimodal Gaussian distribution. Particle filters are not limited in this sense. The Extended Kalman filter is also limited that it needs the movement and measurement noise to be Gaussian which is not the case for the particle filter. Particle filters are also able to localize globally were the Kalman filters aren't.

The major advantages of the Extended Kalman filter as apposed to particle filters is the efficiency in computational time and space.

In general the particle filters are more robust but are more computationally intensive compared to the Kalman filters.

In this project we used the Adaptive Monte Carlo Localization algorithm. The reason we chose this algorithm is that we needed a robust algorithm that could handle global localization. We used the adaptive algorithm so it will be more efficient computationally.

3 SIMULATIONS

3.1 Achievements

In this exercise set out to check how robust the ROS's AMCL and move_base packages are. In order to do this we built three robots. One was a benchmark robot. The other two were robots with challenging dimensions. One robot is a extra long robot. Another robot was an extra wide robot. We were able to create three well tuned robots that where able to accurately localize and navigate to a given position. We where able to show that once the majority of the parameters were tuned well, a small number of parameters needed to

be changed in order to get robots with even challenging dimensions to navigate.

The most challenging part of the navigation path was to navigate around the corner. As you can be see in the pictures that once the parameters are tuned properly all the robots were able to accomplish this regardless of their dimensions.

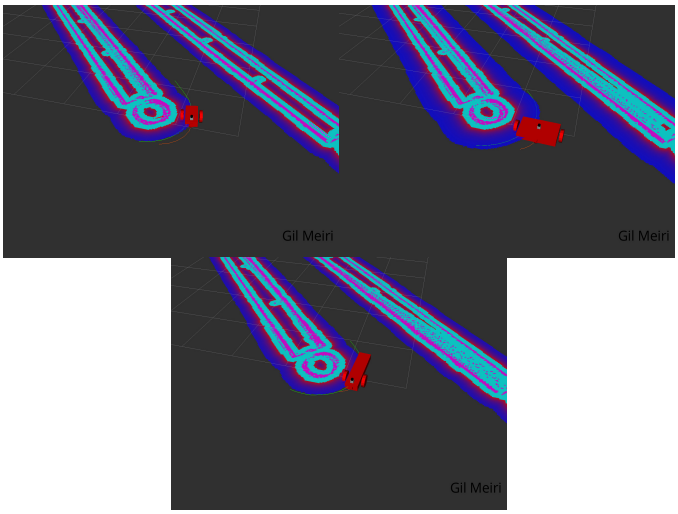

Fig. 1. Three robots turning the corner.

TABLE 1
Robot specs for benchmark robot

Benchmark Robot		
Chasis	Mass	15 kg
	Width	0.2 m
	Length	0.4 m
	Height	0.1 m
Footprint Coordinates	Left front	(0.1, -0.2)
	Left back	(-0.1, -0.2)
	right front	(0.1, 0.2)
	right back	(-0.1, 0.2)
Caster wheels	Mass	Part of the chasis
	Radius	5 cm
	Front caster	(0.15,0,-0.05)
	Back caster	(-0.15,0,-0.05)
Wheels	Mass	5 kg
	Radius	10 cm
	Width	5 cm
	Left wheel	(0, -0.15,0,0)
	Right wheel	(0,0.15,0)
Camera	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0.2, 0, 0)
Laser range finder	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0.15, 0, 0.1)

3.2 Benchmark Model

3.2.1 Model design

The benchmark robot we used is a square in shape. It has two wheels located on either side and two caste wheels for support. The center of rotation is in the center of the robot.

The robot also has two sensors. One sensor is a camera mounted in the front of the robot. Another is a Hokuyo laser range finder above the front of the robot.

All of the robots specs are summarized in Table 1. The zero coordinate of our robot is located around the robots center of rotation which is located in the center of the two wheels.

TABLE 2
ROS packages used

Package Name	Subscribed Topics	Published Topics	Services
joint_state_publisher		/joint_states	
robot_state_publisher	/joint_states	/tf /tf_static	
map_server		/map	static_map
amcl	/initial_pose /tf /tf_static /udacity_bot/laser/scan	/tf	global_localization request_nomotion_update
move_base	/map /tf /tf_static /move_base/global_costmap/footprint /udacity_bot/laser/scan /move_base_simple/goal	/move_base/global_costmap/costmap /move_base/global_costmap/costmap_updates /move_base/current_goal /move_base/global_costmap/footprint /cmd_vel	make_plan clear_unknown_space clear_costmaps

3.2.2 Packages Used

The packages that were used in this project are summarized in Table 2.

3.2.3 Parameters

The parameters that we changed were for the acml node, base_move node. We also modified the minimum range of the laser range finder in the udacity_bot.gazebo file, to 0.15 meters otherwise the robot would detect its wheels as obstacles.

These are the parameters we worked on for the AMCL node.

1) **min_particles**: This variable set the minimum number of particles used in the AMCL algorithm. The lower the value the fast the calculations but this comes at a price of accuracy.
2) **max_particles**: This variable set the maximum number of particles used in the AMCL algorithm. It is best to set relatively high but not too high since it comes at a computational price.
3) **update_min_d**: Minimum movement the robot needs to move forward for there to be an update step. For accuracy this value needs to be small, but if it is too small comes at a computational cost.
4) **update_min_a**: Minimum movement the robot needs to turn for there to be an update step. For accuracy this value needs to be small, but if it is too small comes at a computational cost.
5) **laser_max_range**: The maximum range of the laser. Need to set it to the real lasers maximum range.
6) **laser_min_range**: The minimum range of the laser. Need to set it to the real lasers minimum range.
7) **laser_max_beams**: How many beams in each scan to be used when updating the filter.
8) **laser_z_hit**: Mixture weight for the z_hit part of the model. This is the weight given to the probability that the measurement distance given by the sensor is a result of the beam hitting real object in real

world.

9) **laser_z_rand**: Mixture weight for the z_rand part of the model. This is the weight given to the probability that the measurement distance given by the sensor is a result of random noise.
10) **laser_likelihood_max_dist**: Maximum distance to do obstacle inflation on map, for use in likelihood_field model.

The values given for the AMCL node for the benchmark robot are summarized in Table 3.

TABLE 3
AMCL parameters for benchmark robot

Parameter	Value
min_particles	40
max_particles	100
update_min_d	0.02
update_min_a	0.1
laser_max_range	30.0
laser_min_range	0.1
laser_max_beams	100
laser_z_hit	0.9
laser_z_rand	0.1
laser_likelihood_max_dist	30.0

The move_base node has different parameters to calculate the global costmap, local costmap and the local path planner. These are the parameters we worked on for the move_base node.

Global and local costmaps:

1) **map_type**: What map type to use. "voxel" or "costmap"
2) **obstacle_range**: The default maximum distance from the robot at which an obstacle will be inserted into the cost map in meters.
3) **raytrace_range**: The default range in meters at which to raytrace out obstacles from the map using sensor data.
4) **transform_tolerance**: Specifies the delay in transform (tf) data that is tolerable in seconds. This value is dependant on system performance. The value must not be set too low otherwise there will be error messages.
5) **footprint**: The footprint of the robot specified. If the footprint isn't given the inflation cannot be calculated.
6) **observation_sources**: A list of observation source names. Includes things like the sensor_frame, data_type and topic.
7) **global_frame**: The global frame for the costmap to operate in.
8) **robot_base_frame**: The name of the frame for the base link of the robot.
9) **update_frequency**: The frequency the map is updated. This parameter is system dependent and was set due to performance.
10) **publish_frequency**: The frequency the map is to be published.
11) **width**: The width of the map.
12) **height**: The height of the map.
13) **resolution**: The resolution of the map. As the resolution increase such the computational demand increase.
14) **static_map**: Is this map static or rolling window.
15) **rolling_window**: Whether or not to use a rolling window version of the costmap. If the static_map parameter is set to true, this parameter must be set to false.
16) **inflation_radius**: The distance from the obstacles to inflate the. This makes sure that when planning the path, the robot stays away from the wall.

The global costmap values given for base_move node in the benchmark robot are summarized in Table 4. Those for the local costmap are summarized in Table 5. There are two things we want to note. First is that for local costmap the size of the map is really small. We have found if the map is large than the robot may be incline to ignore the global path and go straight for the goal position if the target is located in the local costmap. The other thing is that we split the inflation for the global costmap and the local costmap. We found that this worked better.

TABLE 4
move_base parameters for global costmap for benchmark robot

Parameter	Value
map_type	costmap
obstacle_range	30.0
raytrace_range	30.0
transform_tolerance	0.3
footprint	[[0.2, 0.1], [-0.2, 0.1], [-0.2, -0.1], [0.2, -0.1]]
observation_sources	laser_scan_sensor
laser_scan_sensor	{sensor_frame: hokuyo, data_type : LaserScan, topic: /udacity_bot/laser/scan, marking: true, clearing: true}
global_frame	map
robot_base_frame	robot_footprint
update_frequency	15.0
publish_frequency	15.0
width	40.0
height	40.0
resolution	0.05
static_map	true
rolling_window	false
inflation_radius	0.35

Lastly the parameters for the base_local_planner which is part of the move_base node.

1) **holonomic_robot**: Determines whether velocity commands are generated for a holonomic or non-holonomic robot. In our case the robot is non-holomic
2) **sim_time**: The amount of time to forward-simulate trajectories in seconds. If this value is too low then there is a chance the robot will get stuck
3) **sim_granularity**: The step size, in meters, to take between points on a given trajectory. A value that is too small cause the robot to take very small steps and the robot could get stuck in the sense that it tries to optimize the the path and moves in place left and

TABLE 5
move_base parameters for local costmap for benchmark robot

Parameter	Value
map_type	costmap
obstacle_range	30.0
raytrace_range	30.0
transform_tolerance	0.3
footprint	[[0.2, 0.1], [-0.2, 0.1], [-0.2, -0.1], [0.2, -0.1]]
observation_sources	laser_scan_sensor
laser_scan_sensor	{sensor_frame: hokuyo, data_type: LaserScan, topic: /udacity_bot/laser/scan, marking: true, clearing: true}
global_frame	odom
robot_base_frame	robot_footprint
update_frequency	15.0
publish_frequency	15.0
width	2.0
height	2.0
resolution	0.1
static_map	false
rolling_window	true
inflation_radius	0.2

right around its center of rotation. If the value is too high than the trajectories that are chosen are also large and make it difficult for the robot to follow the global path.

4) **meter_scoring**: Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expressed in units of meters or cells.
5) **min_vel_x**: The minimum forward velocity allowed for the base in meters/sec. This value makes sure the robot keeps moving forward. A value that is too small may cause the robot to get stuck in the sense that it turns left and right around center of rotation.
6) **vx_samples**: The number of samples to use when exploring the x velocity space. There needs to be enough values so the robot can choose a good path. The value doesn't need to be too high
7) **vth_samples**: The number of samples to use when exploring the theta velocity space. There needs to be enough values so the robot can choose a good path. The value doesn't need to be too high
8) **pdist_scale**: The weighting for how much the controller should stay close to the path it was given.
9) **gdist_scale**: The weighting for how much the controller should attempt to reach its local goal.
10) **occdist_scale**: The weighting for how much the controller should attempt to avoid obstacles.
11) **heading_lookahead**: Whether to score based on the robot's heading to the path or its distance from the path. This is important that this value is set to true since otherwise the robot tries to find the fastest root to the global path and can start to go in the opposite direction.
12) **heading_scoring**: How far to look ahead in time in seconds along the simulated trajectory when using heading scoring.
13) **publish_cost_grid_pc**: This is useful visualization of the costmap for debugging

The base_local_planner values given for base_move node in the benchmark robot are summarized in Table 6.

TABLE 6
move_base parameters for local costmap for benchmark robot

Parameter	Value
holonomic_robot	false
sim_time	2.0
sim_granularity	0.3
meter_scoring	true
min_vel_x	0.1
vx_samples	10
vth_samples	20
pdist_scale	1.2
gdist_scale	0.8
occdist_scale	0.5
heading_lookahead	0.8
heading_scoring	true
publish_cost_grid_pc	true

3.3 Personal Model - Long robot

3.3.1 Model design

The Long robot as the name suggest is similar to the benchmark robot but is longer. There are a few changes. The robot wheels are in the front and so is its center of rotation. This robot has only one caster wheel located in the back. This wheel has a mass, otherwise the robot will tilt forward. The specs of this robot are summarized in Table 7.

TABLE 7
Robot specs for Long Robot

Long Robot		
Chasis	Mass	15 kg
	Width	0.2 m
	Length	0.8 m
	Height	0.1 m
Footprint Coordinates	Left front	(0.2, -0.1)
	Left back	(-0.6, -0.1)
	right front	(0.2, 0.1)
	right back	(-0.6, 0.1)
Caster wheels	Mass	2 kg
	Radius	5 cm
	Back caster	(-0.55,0,-0.05)
Wheels	Mass	5 kg
	Radius	10 cm
	Width	5 cm
	Left wheel	(0, -0.15,0,0)
	Right wheel	(0,0.15,0)
Camera	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0.2, 0, 0)
Laser range finder	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0.15, 0, 0.1)

3.3.2 Packages Used

The packages used are the same packages as the benchmark robot.

3.3.3 Parameters

The parameters used are almost identical to the benchmark robot. The only parameters that needed to be changed were part of the move_base node. These parameters were the footprint and the inflation values for both the local and global costmaps. The new values are summarized in Table 8. All the rest of the values are the same as they appear in Tables 3-6.

TABLE 8
Differences in parameters between Benchmark and Long Robots

Global costmap	
footprint	[[0.2, 0.1], [-0.6, 0.1], [-0.6, -0.1], [0.2, -0.1]]
inflation_radius	0.35
local costmap	
footprint	[[0.2, 0.1], [-0.6, 0.1], [-0.6, -0.1], [0.2, -0.1]]
inflation_radius	0.2

3.4 Personal Model - Wide robot

3.4.1 Model design

The Wide robot very similar to the benchmark robot but wider. Apart from being wider the laser range finder was moved back so it is above the robots center of rotation. The reason for this was that the minimum value that was set for the laser range was set to 0.4 m due to the fact that otherwise the robot would sense its wheels ans obstacles. In order for the "dead zone" in front of the robot to be as small as possible, the sensor was moved back.The specs of this robot are summarized in Table 9.

TABLE 9
Robot specs for Wide Robot

Wide Robot		
Chasis	Mass	15 kg
	Width	0.6 m
	Length	0.4 m
	Height	0.1 m
Footprint Coordinates	Left front	(0.2, -0.4)
	Left back	(-0.2, -0.4)
	right front	(0.2, 0.4)
	right back	(-0.2, 0.4)
Caster wheels	Mass	Part of the chasis
	Radius	5 cm
	Front caster	(0.15,0,-0.05)
	Back caster	(-0.15,0,-0.05)
Wheels	Mass	5 kg
	Radius	10 cm
	Width	5 cm
	Left wheel	(0, -0.35,0,0)
	Right wheel	(0,0.35,0)
Camera	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0.2, 0, 0)
Laser range finder	Mass	0.1 kg
	Width	5 cm
	Length	5 cm
	Height	5 cm
	Location	(0, 0, 0.1)

3.4.2 Packages Used

The packages used are the same packages as the benchmark robot.

TABLE 10
Differences in parameters between Benchmark and Wide Robots

Global costmap	
footprint	[[0.4, 0.2], [-0.4, 0.2], [-0.4, -0.2], [0.4, -0.2]]
inflation_radius	0.5
local costmap	
footprint	[[0.4, 0.2], [-0.4, 0.2], [-0.4, -0.2], [0.4, -0.2]]
inflation_radius	0.4

3.4.3 Parameters

The parameters used are almost identical to the benchmark robot and long robot. As with the long robot there where changes to parameters that are part of the move_base node. These parameters were the footprint and the inflation values for both the local and global costmaps. The new values are summarized in Table 10. All the rest of the values are the same as they appear in Tables 3-6.

Apart from those values the minimum laser range for the laser range finder was increased in the udacity_bot.gazebo file to 0.4 m. This was done in so that the wheels are not identified as obstacles.

4 Results

During our experimentations we saw the where the AMCL excelled and where it suffers. The major strength of the AMCL is that once the parameters are turned well the results of the localization are pretty much the same for robots with different dimensions. AMCL preforms well and converges fast when it has nearby walls. When the robots navigated through the corridor there was a good convergence in the x axis (sides) for the particles but the particles were spread in the robots y axis (forward-backward). The particles only converged in the y axis when they reached the end of the corridor. The particle filter also had a rough time when the robot preformed a spin around its center of rotation. If the particle filter didn't update frequently the particles would spread out.

When it came to navigation the robot did follow a smooth path that matched the global plan. This is true for all three robots. That being said it did take a long time to reach the target. With more parameter tuning better results could probably be achieved. The move_base package had a difficult time to respond if the local costmap was too large. This was due to the fact that if the target appeared in the local costmap, move_base would ignore the global path, not take into account the obstacles such as walls in the way, and go straight for the target.

4.1 Localization Results

4.1.1 Benchmark

As can be seen in figure 2 the robot was able to localize well at the target for the benchmark robot.

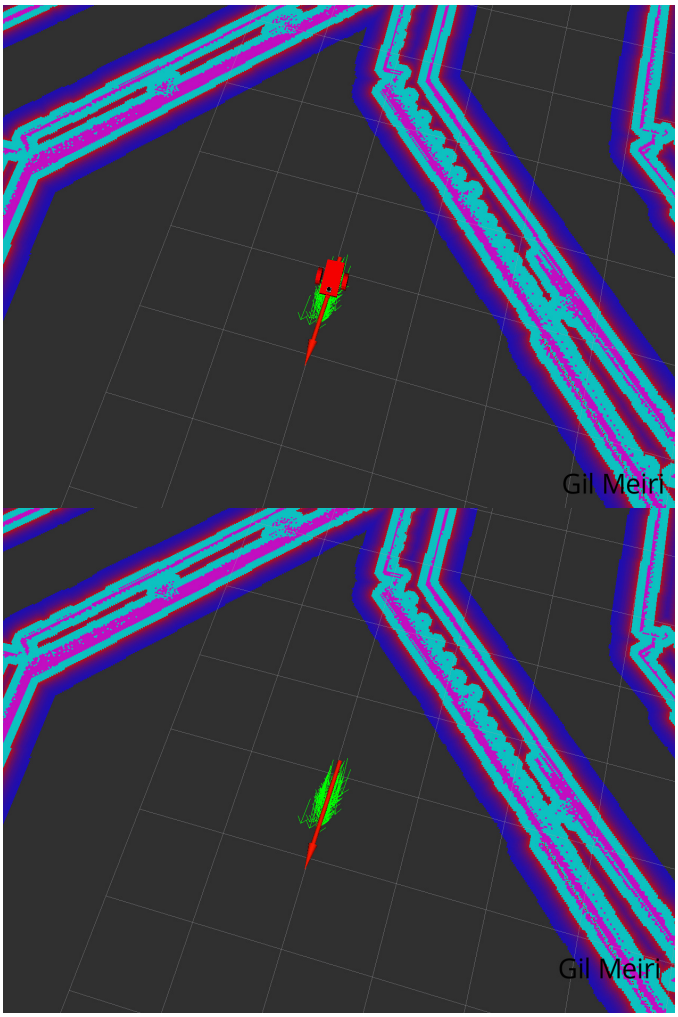

Fig. 2. Benchmark Robot at destination.

4.1.2 Student

The robot localization at the target for the long robot is given in figure 3, and for the wide robot in figure 4.

4.2 Technical Comparison

As stated before the main differences between the robots are the dimensions. The localization worked pretty much the same for all the robots. When it came to the navigation there weren't too much differences with performance.

One problem that arises every couple of runs, is that the robot reaches the target location and when trying to adjust the robots heading it gets stuck turning left and right on the spot. This happens more with the long and the wide robots and not so much with the benchmark robot. This may be due to the challenging dimensions of these robots and their difficulty to navigate to close locations.

All the robots manage to turn around the corner which is the most challenging part of the path.

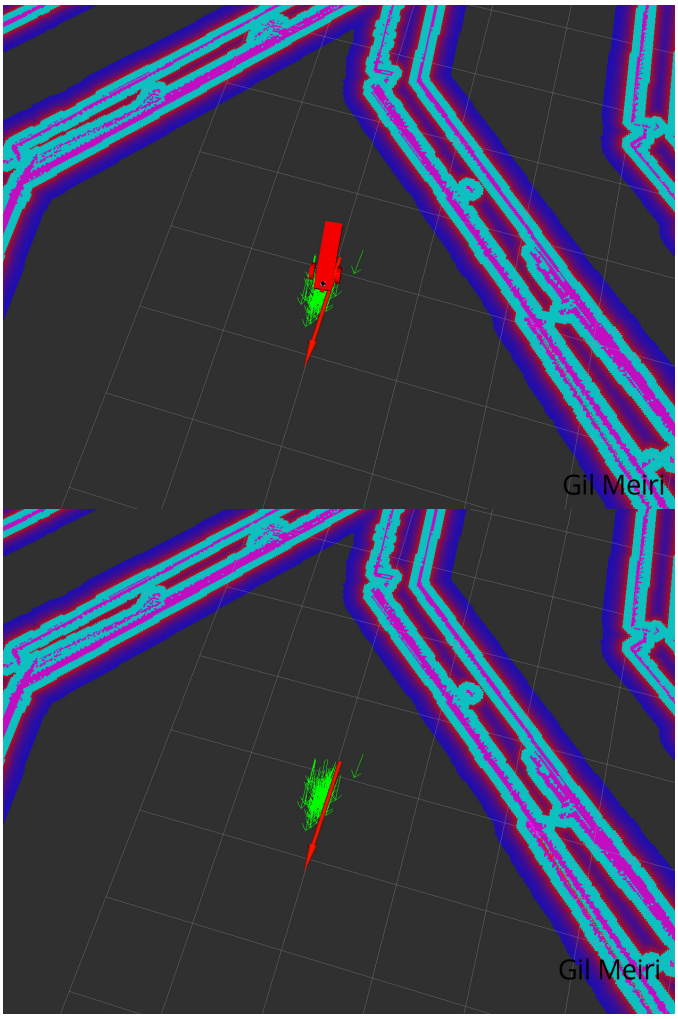

Fig. 3. Long Robot at destination.

5 Discussion

The purpose of this exercise was to learn how to tune parameters for the AMCL and move_base packages. The question that arose was how can these packages be stretched to the limit. We used robots with challenging dimension to try to answer this question.

From the results of this exercise it could be that the localization with AMCL works well without the need to change any parameters. As for move_base only the footprint and inflation needed to be changed in order for the robots to work. This show the robustness of these packages.

One major drawback for these robots what we have designed is that they would not survive the 'Kidnapped Robot' problem. One way to solve this is to change the recovery_alpha_slow and recovery_alpha_fast parameters of the AMCL. This adds a number of random particles every iteration that help solve the 'kidnapped robot' problem.

The MCL/AMCL algorithms are limited since they need prior map. This means that for an industry setting, these algorithms could be used in places where there are accurate maps. This could include things like a factory floor, warehouse etc.

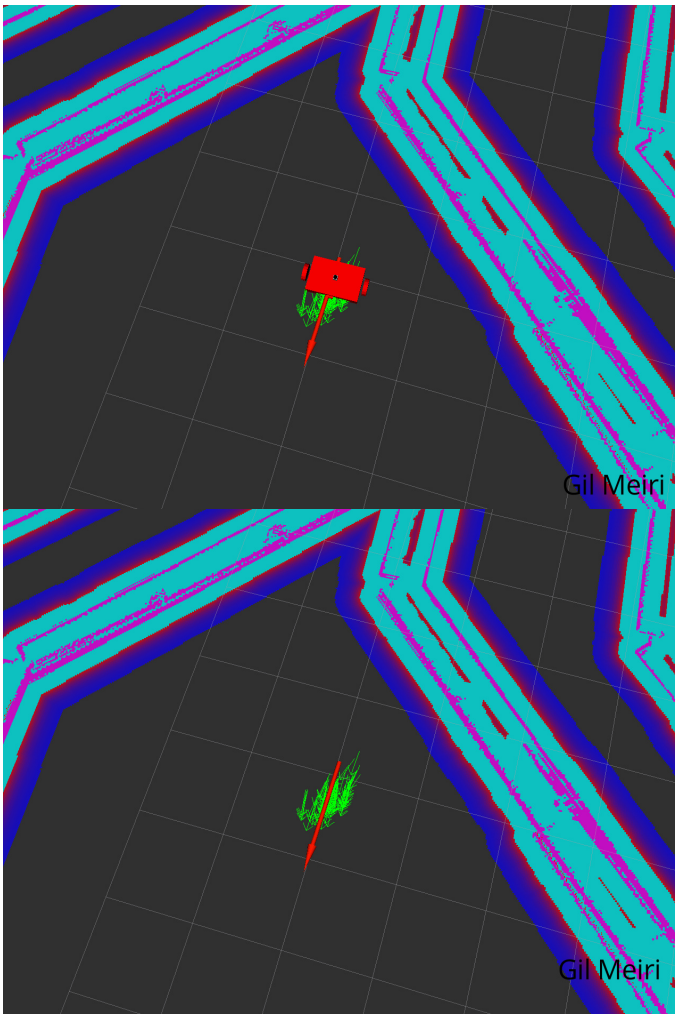

Fig. 4. Wide Robot at destination.

6 Conclusion / Future work

In this exercise we tuned the parameters for AMCL and move_base for a particular robot. We saw that once these parameters were properly tuned we could easily change the dimensions of the robot and the robot would still be able to reach their target.

Different parameters had trade offs between the accuracy and the computational performance. This means that certain parameters could be tuned to increase performance but at the price of the use of computational resources. When tuning the parameters we had to take this into account and do allot of iterations to find the right balance.

When it came to localization the robots were good at converging on their actual locations. Additional sensors might help, especially more accurate long distance sensors that could help the robot localize when the walls are far away. Probably the more computational power could help improve results allot.

For future work the parameters for move_base could be improved. The time to reach the goal position could probably be lowered with rightly tuned parameters.

References

[1] L. Lamport, *LATEX: a document preparation system: user's guide and reference manual*. Addison-wesley, 1994.