Robot Localization and Navigation

Forrest Edwards

Abstract—Two mobile robot designs are presented that successfully navigate a simulated virtual environment consisting of connected hallways and open spaces. Localization of the robots is performed using the ROS Adaptive Monte Carlo Localization package with input from a simulated Hokuyo laser range finder and wheel odometry data. Navigation is implemented using the ROS navigation stack utilizing Dijkstra's algorithm for global planning and the Dynamic Window Algorithm package as the local planner. Both robots use differential drive for their motion control solution. Navigation was demonstrated to be collision free and smooth with maximum travel times from point to point in the 40m x 40m all under 5 minutes. The specific project goal was successfully achieved in 60 seconds.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, ROS, AMCL, Localization, Navigation.

1 Introduction

Successful navigation of a mobile robot within a complex environment has been demonstrated widely in industrial and manufacturing applications. In these environments automated guided vehicles (AGV's) are used to move components and goods along predefined routes which are often dedicated solely to their use. Common solutions include magnetic tape (Figure 1), low power RF wire guidance systems and strategically placed laser targets. While these solutions have shown to work well in settings where the navigation is on fixed paths with little to no route changes, they are not easily adaptable to residential, office or other highly variable environments. In these applications the wide variability in map layouts coupled with both the budgetary, aesthetic and ongoing maintenance to adapt these solutions to layout/route changes makes them impractical for mass adoption.

A more practical solution for these dynamic environments would rely instead on a system that could collect data from the surrounding passive local features (i.e. furniture, structures, walls, equipment etc...) and extrapolate current location based on prior knowledge of these features within the global environment. Such a system would avoid the added cost and upkeep associated with purpose-specific external navigation equipment. Additionally, any fully realized implementation of such a solution would be capable of both locating itself on a map and improving the map by detecting changes in features as objects are added or removed from the robot's physical surroundings. This is analogous to how humans locate themselves in their environment; by observing the patterns in the current surroundings and matching those to a memory of that location. In addition, when a change is detected (e.g. a piece of furniture is removed), the human response is not typically to become lost, but to update their internal truth map to reflect the change.

The work presented here is intended to demonstrate a solution to the localization and navigation problem but leaves the challenge of iterative map updates (SLAM) for future investigation. The virtual robot design that follows demonstrates the feasibility of implementing the AMCL particle filter approach to successfully locate and navigate a mobile robot base through a simulated environment. This is shown to be possible with the existing ROS Navigation stack, an off the shelf scanning laser range finder, and a known map of the environment.

Fig. 1. AGV using magnetic tape. *Photo courtesy of Creform Corp. www.creform.com*

2 Background / Formulation

The task of robot localization and navigation can be broken down into 4 high level tasks.

1) **Localization:** Where Am I?
2) **Global Navigation:** What is the path to my destination?
3) **Local Navigation:** How do I stay on the path and avoid obstacles?
4) **Motion Control:** What motor/actuator settings will achieve local navigation goal?

The ROS framework breaks down these tasks into individual packages which, when combined, form the ROS navigation stack [1] shown in Figure 2.

2.1 Localization

The localization task for this work was implemented using the Adaptive Monte Carlo Localization (AMCL) package

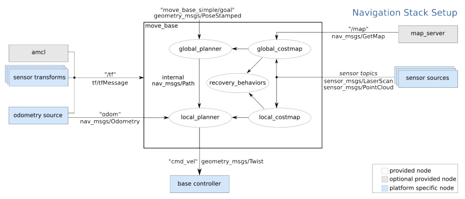


Fig. 2. ROS Navigation Stack. *Reproduced under Creative Commons 3.0. Attribution http://wiki.ros.org/navigation/Tutorials/RobotSetup*

[2]. Monte Carlo Localization (MCL) [3] leverages a particle filter to approximate the robot's pose as the average pose of a set of particles. Each particle within the filter is assigned a statistical likelihood that it represents the true robot pose. This statistical likelihood, or weight, determines the probability that a particle will be replicated or eliminated when the particle filter is re-sampled. Weights are determined for each particle by comparing sensor data from the laser scanner with the hypothetical measurements that would be received if the robot's pose matched that of the particle. Resampling is completed by creating a new posterior particle set constructed of samples of the previous set. This new particle set will have the same dimension as the original but over-represent the particles with higher weights while under-representing the particles with lower weights. As this process is repeated, the particle set will converge to cluster around the true robot position, and thus the average of the particle poses will become increasingly representative of the actual robot pose indicated by the laser sensor. A simplified simulation of this process, as well as a practical depiction are shown Figures 3 and 4.

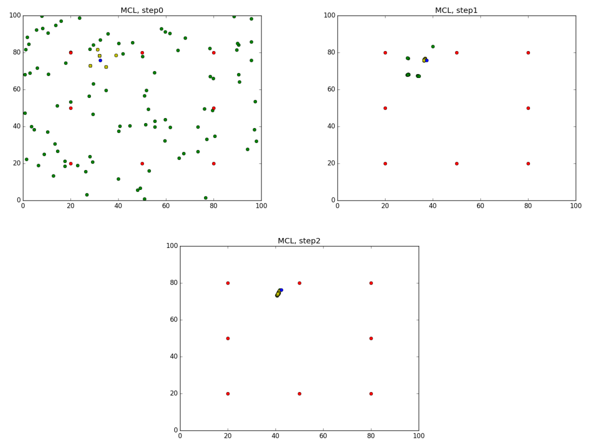


Fig. 3. Sample MCL particle filter iterations. n = 100 particles

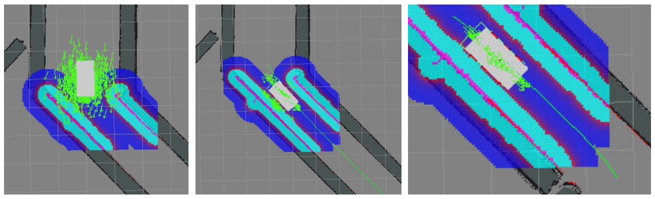


Fig. 4. AMCL pose array in practice surrounding Robot at t=0, 4, 8

The primary difference between the basic MCL algorithm and the adaptive (AMCL) variant, is that the adaptive version uses binning techniques to scale the particle set. Both approaches start out similarly, but where the dimension of the particle filter remains constant in MCL, AMCL leverages KLD sampling [4] to bin, and reduce the total number of particles as they become clustered. The augmented algorithm demonstrates performance gains since the number of operations per localization step is directly proportional to the size of the particle filter which shrinks as they converge.

A Kalman Filter [5] could also have been applied rather than utilizing AMCL. Instead of averaging the robots pose over a number of discrete particles, Kalman filters are model based and develop a continuous 2D Gaussian distribution representing the probability of robots location. Kalman filters are computationally efficient, making them ideal for small embedded systems [6]. However, for several reasons AMCL was chosen over a Kalman or Extended Kalman filter. In addition to being more difficult to implement Kalman filters are are unsuited to the global localization problem [3] and require that all process and sensor noise be modeled by guassian distributions or by Taylor series expansions for non linear distributions. In a simulated environment this is less of an issue but in actual practice the robot will encounter bumps, non-level surfaces, wheel slippage and other point disturbances to the odometry and range finder data that cannot be modeled by the Kalman filter.

2.2 Global Navigation

Global navigation was implemented using the ROS global_planner configured for Dijkstra's algorithm [7]. The global_planner was chosen over navfn for two reasons. Firstly, it is intended to supersede navfn as a more flexible planner [8] [9]. Secondly since it includes the alternate path planning algorithm A* and the ability to reproduce navfn behavior it allowed for comparisons of alternate algorithms and planners. The global_planner is dependent on the costmap_2d [10] module to plan collision free paths to the navigation goal. The costmap provides several key pieces of functionality required by the global_planner. Firstly, it maintains a map of obstacles based as published in the jackal_race map provided by the map server. Secondly, the costmap_2d package creates an inflation layer surrounding each obstacles that assigns high costs (253-255) to grid locations nearest the obstacles and lower costs to grid points further way. The costs at the further grid points decay exponentially as the distance from the obstacle increases out to the inflation_radius beyond which costs are set to zero. The inflation radius is user configurable but needs to be tailored to the environment such that collisions are avoided but plans through navigable narrow openings are still permitted. Figure 5 shows a cutaway view of what this inflation layer looks like from the side. The inflation layers purpose is to creates a buffer zone around the obstacles on the map. Since these buffer zones are navigable, but have a cost assigned, the global_planner optimizes routes around objects and through narrow corridors by choosing routes that minimizes the cost of route. This can be seen in Figure 6 which shows the minimum cost route for navigating

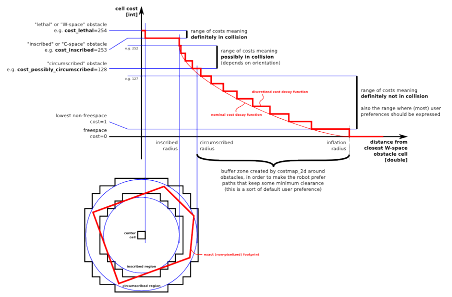

cell cost [int]
"lethal" or "W-space" obstacle e.g. cost_lethal=254
"inscribed" or "C-space" obstacle e.g. cost_inscribed=253
"circumscribed" obstacle e.g. cost_possibly_circumscribed=128
lowest non-freespace cost=1
freespace cost=0
e.g. 252
e.g. 127
range of costs meaning definitely in collision
range of costs meaning possibly in collision (depends on orientation)
range of costs meaning definitely not in collision
also the range where (most) user preferences should be expressed
discretized cost decay function
nominal cost decay function
inscribed radius
circumscribed radius
inflation radius
distance from closest W-space obstacle cell [double]
buffer zone created by costmap_2d around obstacles, in order to make the robot prefer paths that keep some minimum clearance (this is a sort of default user preference)
center cell
exact (non-pixelized) footprint
inscribed region
circumscribed region

Fig. 5. Cutaway view of costmap inflation layer.*Image reproduced under Creative commons 3.0. Attribution: http://wiki.ros.org/costmap_2d*

between the two walls of the hallway. This same costmap approach is also used for the local navigation task in the following discussion.

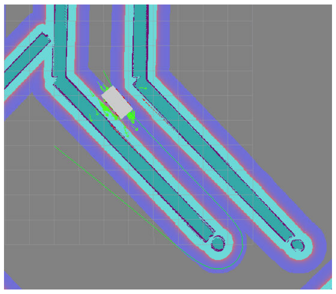


Fig. 6. Global planner route (green line) indicating minimum cost path through hallway.

2.3 Local Navigation

For the task of local navigation, the Dynamic Window Approach (DWA) algorithm [11] was used. The DWA algorithm is dynamic in that it constrains which local plans to optimize based on the robots current state(velocity, pose), kinematic capabilities, and local obstacles. This has several benefits the first of which is that only trajectories that are attainable by the robot are considered. Secondly, the algorithm discards trajectories that are unsafe or would place the robot in a dynamic configuration from which it can not recover before colliding with an obstacle. By restraining the possible trajectories to only those that are possible and safe, the third benefit of DWA is increased computational efficiency, since only a subset of the candidate trajectories are actually optimized. The optimization utilizes an objective function to determine the best path based on the following criteria:

- **Heading:** How close is the candidate trajectory's heading to a heading that would take the robot straight to the local goal?
- **Clearance:** How close does the candidate trajectory take the robot to obstacles?
- **Velocity:** Does the candidate trajectory allow for quicker movement (higher velocity)?

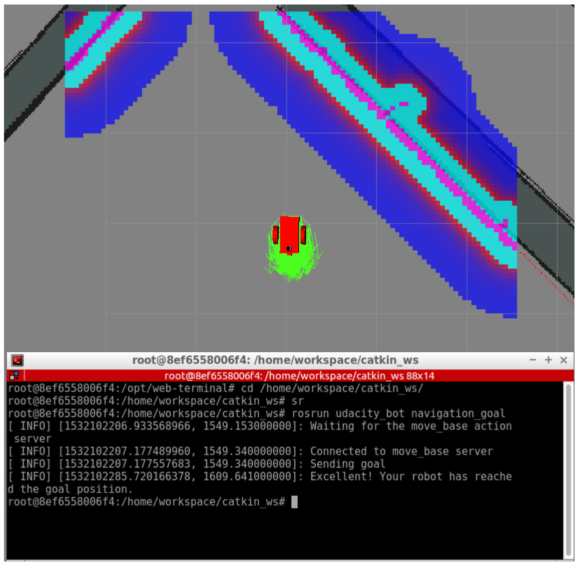
The obstacle data needed above by DWA planner is provided by the local costmap which is conceptually identical to the global costmap discussed previously but constrained to a small area around the robot. This local costmap is continuously updated with obstacle distance information from the laser range finder. Kinematic inputs to the DWA planner are received on the odometry topic and maximum velocity and acceleration limits are set as part of the robot's characteristics in the DWA planner robot configuration.

2.4 Motion Control

Motion control for the robot was implemented though the move_base package which supervises the ROS costmap and planner components then publishes the robots translational and angular velocity targets on the cmd_vel twist topic. The cmd_vel topic is subscribed to by the Gazebo libgazebo_ros_diff_drive plugin [12] that converts the messages into torque commands for the left and right wheels of the robot in the Gazebo environment. The libgazebo_ros_diff_drive plugin also monitors the motion of the robot's drive wheels and publishes the feedback to the odom topic for consumption by the location and navigation modules previously mentioned.

3 Results

With respect to reaching the target pose required for the project, both the reference robot and the modified robot were successful as can be seen in Figures 7 and 8.



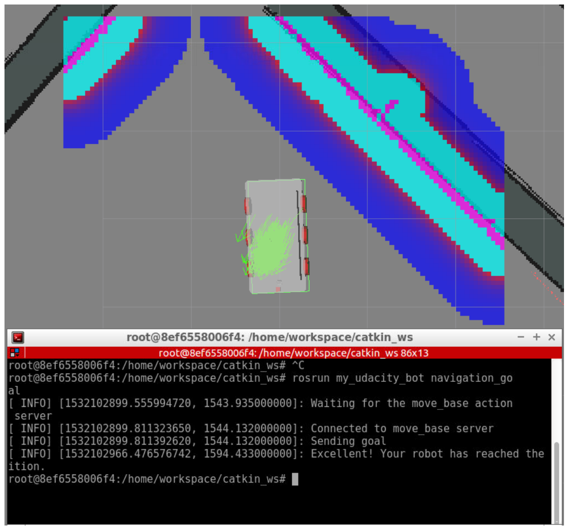
```
root@8ef6558006f4: /home/workspace/catkin_ws
root@8ef6558006f4: /home/workspace/catkin_ws 88x14
root@8ef6558006f4:/opt/web-terminal# cd /home/workspace/catkin_ws/
root@8ef6558006f4:/home/workspace/catkin_ws# sr
root@8ef6558006f4:/home/workspace/catkin_ws# rosrun udacity_bot navigation_goal
[ INFO] [1532102206.933568966, 1549.153000000]: Waiting for the move_base action
 server
[ INFO] [1532102207.177489960, 1549.340000000]: Connected to move_base server
[ INFO] [1532102207.177557683, 1549.340000000]: Sending goal
[ INFO] [1532102285.720166378, 1609.641000000]: Excellent! Your robot has reache
d the goal position.
root@8ef6558006f4:/home/workspace/catkin_ws#
```

Fig. 7. Reference robot at goal with AMCL particle pose array indicated by green arrows.

The one notable difference between the two results being that the modifed robot required its starting position placed further from the goal than the reference design due to its larger physical size which, when placed at the origin, was in collision with a barrier at initialization. Results of time to complete goal for each robot are summarized in Table 1.

TABLE 1
Navigation Requirement Results

Robot	Goal Achieved	Total Time
Reference Design	Y	59s
Modified Design	Y	51s


root@8ef6558006f4: /home/workspace/catkin_ws
root@8ef6558006f4: /home/workspace/catkin_ws 86x13

```
root@8ef6558006f4:/home/workspace/catkin_ws# ^C
root@8ef6558006f4:/home/workspace/catkin_ws# rosrun my_udacity_bot navigation_go
al
[ INFO] [1532102899.555994720, 1543.935000000]: Waiting for the move_base action
 server
[ INFO] [1532102899.811323650, 1544.132000000]: Connected to move_base server
[ INFO] [1532102899.811392620, 1544.132000000]: Sending goal
[ INFO] [1532102966.476576742, 1594.433000000]: Excellent! Your robot has reached the
ition.
root@8ef6558006f4:/home/workspace/catkin_ws#
```

Fig. 8. Modified robot at goal with AMCL particle pose array indicated by green arrows.

In addition to the required goal, both robots were able to successfully navigate from/to any valid points on the map without issues in under 5 minutes.

The AMCL filter did not responded well to kidnapped robot events where the robot was spawned in a location that the AMCL filter was not expecting. The robot would move briefly in a kidnapped robot scenario, then panic and stop navigating entirely.

4 Model Configurations

As part of this work, two virtual robots were created, tuned and tested. The first was largely provided by Udacity as a reference design and only required tuning and adjustment to achieve the desired navigation goal. The second robot represents a modification to the first with significant changes to the dimensions, base frame, sensor location, and wheel setup. A rendering both designs can be seen in Figure 9.

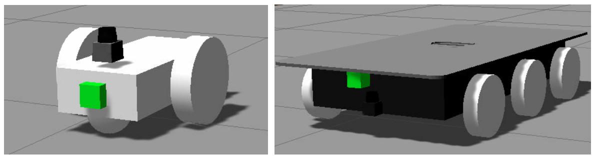


Fig. 9. Udacity reference design robot (left) and modified design (right)

4.1 Udacity Reference Robot

4.1.1 Physical Design

As can be seein in Figure 9 the physical design of the Udacity robot is comprised of 7 components:

1) Robot chassis
2) 2 drive wheels located on the left and right of the chassis
3) Forward facing camera (green box)
4) Hokuyo rotating laser range finder (mounted top front)
5) 2 casters, fore and aft (spheres attached to undercarriage)

4.1.2 Localization Parameters

Several parameters were adjusted to optimize the AMCL particle filter. The greatest improvement in performance was seen after updating the parameters used to estimate noise in the odometry data. By default, these parameters are several orders of magnitude greater than the correct values for the diff-corrected model utilized [2]. At their default values, the settings imply extremely noisy odometry data which manifests as AMCL particle filters that do not converge on the robot frame (due to the high level of uncertainty).

Since it is known that both memory and computational time scale linearly with particle filter size, the maximum and minimum number of particles used by the AMCL module were both reduced to reduce memory and CPU usage. While this still yeilded adequate performance in this environment, reducing the particle set excessively can cause the robot to become lost due to having no particles representative of its actual position.

TABLE 2
AMCL Parameters

Parameter	Default Value	Design Value
odom_alpha1	0.2	0.005
odom_alpha2	0.2	0.005
odom_alpha3	0.2	0.010
odom_alpha4	0.2	0.005
transform_tolerance	0.1	0.3
min_particles	100	50
max_particles	5000	500
laser_min_range	-1.0	0.010
laser_max_range	-1.0	30.0
laser_max_beams	30.0	45.0

4.1.3 Global Navigation Parameters

By default the global planner runs once for each navigation goal. However, it was found that by setting the global planner frequency to periodically re-plan yielded positive results. In the situation where the robot moves off of the existing plan or gets stuck, re-planning the path ensures that the global plan is always near optimal from the robots current position. This is preferable to attempting a sharp turn to follow the initial plan which may no longer be the optimal path. In this environment, a replan frequency of 0.20hz worked well.

Based on the work by Zheng [9], three additional planner parameters were configured as well which provided smooth, collision free global paths: cost_factor, neutral_cost, and lethal_cost. The three parameters are used to modify the raw cost score set by the inflation layer before being optimized by the planner. neutral_cost is an offset that allows the the entire costmap to be adjusted up or down by a fixed value. cost_factor is used to scale the raw cost value linearly. lethal_cost allows for adjusting the threshold indicating a robot is in collision with an object.

The common costmap settings were also updated, based primarily on the physical dimensions of the surrounding obstacles and the robot itself. Firstly, since the global map was static, the update and publish frequency were reduced from a frequency of 50Hz to 1Hz. This is less than the default of 5Hz but showed no degradation in navigation planning performance. obstacle_range and raytrace_range were also adjusted to limit the effects of ray divergence in detecting and clearing obstacles. Inflation radius was adjusted to cover the narrowest hallways and ensure continuous costmap coverage. By doing this, it ensured that plans would tend toward the centerline of narrow spaces.

Leveraging the footprint setting of the costmap yielded the best improvement in terms of smooth robot motion and eliminating periodic stops. Since the inflation layer is defined (Figure 5) in terms of the robot's geometry (cost_lethal, cost_inscribed, cost_possibly_circumscribed) using the default robot footprint resulted in overly cautious plans and situations where the robot believed it was in collision or could not find a plan when neither case was actually true. Figure 10 shows the difference between the default footprint, and a polygon that actually mimics the footprint. With this modified footprint, the robot had more room to maneuver and select smoother plans.

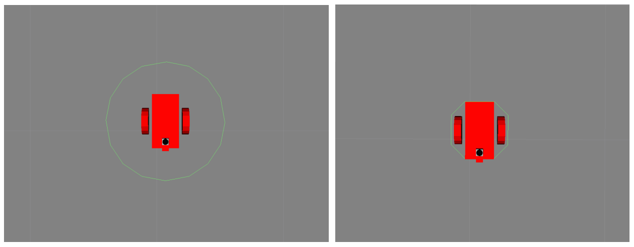


Fig. 10. Green line indicates default robot footprint (left) and modified (right)

TABLE 3
Global Planner and Costmatp Parameters

Planner Parameters	Default Value	Design Value
planner_frequency	0.0	0.20
cost_factor	3.0	0.55
neutral_cost	50	66
lethal_cost	253	253
Costmap Parameters		
obstacle_range	2.5	7.5
raytrace_range	3.0	10.0
footprint	1m dia circle	polygon
transform_tolerance	0.2	0.3
inflation_radius	0.55	1.0
cost_scaling_factor	10.0	10.0
update_frequency	5Hz	1Hz
publish_frequency	0Hz	1Hz

4.1.4 *Local Navigation Parameters*

The initial performance of the local planner was poor. The robot frequently diverged from the global path. It would then oscillate around the global path, occasionally making progress, but more commonly ending up getting stuck near a wall with no valid path found to extract itself. The footprint modification mentioned above provided some improvement with respect to getting stuck as more path options became available. However, several parameters were key to developing stable, repeatable local navigation.

Proper setting of the sim_time parameter provided the greatest increase in stability, primarily with respect to oscillatory behavior. The local DWA planner uses the sim_time parameter to determine how far into the future potential navigation plans extend. In the current implementation the planner only considers the end point of the trajectory when deciding on the best plan. For this work, with sim times greater than 2s the DWA planner would often choose routes that had their endpoints near the global path, but were represented by arced paths that initially headed away from the global path. The robot would proceed down these paths and either get stuck, replan another poor local path, or reach its goal only to select another arced path moving away from the global path. By choosing a relatively small sim_time of 1s the local planner was forced to consider only short arc's which begin to approximate straight lines.

Although the bulk of local planner issues were resolved with the updates to sim_time the robot still had a tendency to track near but not follow the global path. To address this, the path_distance_bias and goal_distance_bias parameters were modified to produce local plans that stayed closer to the global path. path_distance_bias is a linear scaling factor that increases or decreases the planner's affinity for the local path. goal_distance_bias is similar but influences the local planners affinity for the local goal.

The local costmap was also modified to improve computational performance. The initial size of the local costmap was larger than needed for the local navigation task. A setting of 5m x 5m (reducing the memory required by 75%) was selected with no penalty observed in robot performance as a result.

TABLE 4
Local Planner and Costmap Parameters

Planner Parameters	Default Value	Design Value
path_distance_bias	32.0	7.5
goal_distance_bias	24.0	2.0
min_vel_x	0.0	0.0
Costmap Parameters		
obstacle_range	2.5	7.5
raytrace_range	3.0	10.0
footprint	1m dia circle	polygon
transform_tolerance	0.2	0.3
inflation_radius	0.55	1.0
cost_scaling_factor	10.0	10.0
update_frequency	5Hz	5Hz
publish_frequency	0Hz	5Hz
width	10.0	5.0
height	10.0	5.0

4.2 Modified Robot

4.2.1 *Physical Design*

Referring back to Figure 9 one can see that the physical layout and configuration of the modified robot are significantly different than the reference design. The modified design contains 8 major components/links as follows.

1) Robot chassis
2) 2 drive wheels located on the left and right of the chassis center

3) 4 caster wheels 2 fore and 2 aft of the center drive wheels
4) Forward facing camera (green box)
5) Hokuyo rotating laser range finder (mounted top front)

The design concept grew from a desire to create a robot that could be expanded on to fulfill a useful purpose such as providing a platform for additional equipment, or load carrying capability. In support of this concept it was advantageous to have a robot with a wide footprint for stability. This also provided an additional challenge to see if a wider robot would be capable of navigating some of the narrower features in the map.

The bulk of the design was carried out through modification of the xacro files to both add components and change locations and dimensions of existing components. The custom mesh file that is visible as the base however was created using parametric modeling software (FreeCAD). In addition to creating the custom mesh file, shown in Figure 11, physical dimensions for other components were also mocked up and checked here in preparation for creating the xacro file.

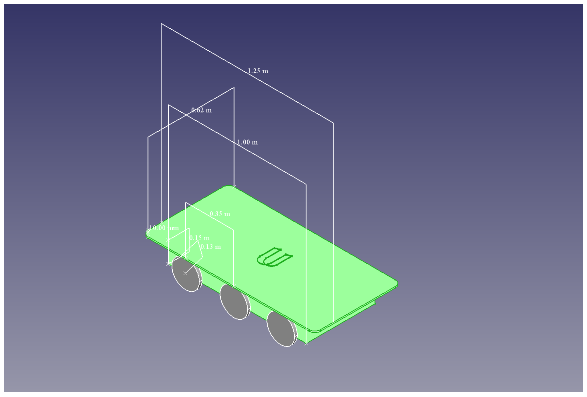


Fig. 11. Modified bot mock up in FreeCAD. Custom mesh is shown in green.

Sensor location was driven by the use concept as well. If the top of the robot is intended as a working surface, the camera and laser range finder should not be mounted there. Instead the choice was made to mount them below the work surface where they would be out of the way, still able to provide the necessary sensor inputs, and be afforded a measure of damage protection provided by the platform overhang.

4.2.2 Localization Parameters

The localization parameters from the reference design worked well on the modified design and are identical to the ones shown in Table 2 with one exception. The base of the modified robot is significantly larger than the reference robot. Without modification, this lead to the the front of the robot spawning on the barriers in the map. To avoid this the initial starting point of the robot had to be moved in the gazebo launch file, and along with that change the AMCL settings needed to be updated to indicate the correct starting position for the robot. Another option would have been to modify the initial yaw to align the robot down the hallway but it was decided this would give the robot an unfair advantage in completing the navigation challenge. Table 5 shows the additional localization parameters added to the modified design to achieve this.

TABLE 5
Starting Pose Parameters for Modified Robot

Parameter	Default Value	Design Value
initial_pose_x	0.0	-2.0
initial_pose_y	0.0	-1.0

4.2.3 Global Navigation Parameters

Global planner parameters from Table 3 required no modification to work with the modified robot. Despite being larger, since the inflation layer also scales as a function of robot footprint the route is procedurally adjusted to compensate. For this to function properly the costmap footprint parameter did require updating to reflect the larger footprint of this robot.

4.2.4 Local Navigation Parameters

Similar findings were realized with the local planner and local costmap parameters. No adjustments were needed other than the footprint change mentioned previously.

5 Discussion

The adaptability of the AMCL filter and navigation stack to the different robot frames with nearly identical parameters was an unexpected result. With only minimal changes to the navigation parameters the modified robot was able to navigate successfully to the goal. With respect to localization the AMCL filter performed well, but in it's current state does not respond well to the kidnapped robot problem. This was a direct result of having no particles located in the robot's actual position and the current inability of the robot to detect that it has been kidnapped. One way to overcome this would be to increase the size of the particle filter and increase the variance of it's distribution. A more intelligent approach could include detection of a kidnapping and then dynamically adjust the AMCL filter for the needs of that situation.

From this work it was demonstrated that the AMCL particle filter approach could supersede the guidance technologies mentioned in the introduction. With proper tuning, AMCL can produce accurate collision free paths that are adaptable to local disturbances. Augmented with a low fidelity localization system such as GPS or WIFI, the kidnapped robot problem could also be solved for these environments.

6 Future Work

Multiple opportunities exist for expanding on this work to improve the results. With respect to the kidnapped robot problem, while AMCL could solve this problem for a home or small office that solution alone would scale poorly to a 2.0M square foot warehouse. Instead it would be compelling to examine ways in which AMCL could be combined with a global localization solution that scales better. While the scanning range finder provided outstanding results, they are expensive devices, with moving components. Cost and reliability are always a concern for commercial products so another potential area for improvement would be to replace the Hokuyo sensor with a solid state LIDAR or computer vision solution. Additionally there is an opportunity to increase the robustness of this solution by incorporating a SLAM component to the navigation stack. In real world applications robots are not notified by the map server when an object has been place in their path so the ability to detect, map, and then reroute around unexpected objects is a key challenge in making these solutions robust replacements for their predecessors.

References

[1] "move_base - ROS Wiki."

[2] "amcl - ROS Wiki."

[3] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99–141, May 2001.

[4] "Monte Carlo localization," Jan. 2018. Page Version ID: 822240294.

[5] R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME Journal of Basic Engineering*, no. 82 (Series D), pp. 35–45, 1960.

[6] "Kalman filter," June 2018. Page Version ID: 846649030.

[7] "Dijkstra's algorithm," July 2018. Page Version ID: 849730156.

[8] "global_planner - ROS Wiki."

[9] K. Zheng, "ROS Navigation Tuning Guide," p. 19.

[10] "costmap_2d/flat - ROS Wiki."

[11] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, pp. 23–33, Mar. 1997.

[12] P. Khandelwal, "gazebo_plugins: gazebo_ros_diff_drive.cpp Source File," June 2013.