RoboND Project 2 : Robot Localization

Yoonyoung Cho

Abstract—A particle filter-based localization and navigation pipeline based on the ROS architecture is demonstrated and verified in gazebo-simulated environments against two distinct robot model configurations; in the proof-of-concept demonstration without exhaustive parameter tuning, the localization performance of each platform is reported to stay within a controlled boundary of 0.1m translational and 2 deg. rotational error throughout the mission.

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

1 Introduction

Localization is a foundational problem in robotic mobile platforms that dynamically interact with the nearly environments and introduces changes in position over time. While it is generally considered a solved problem, adaptation of the domain to specific platforms nonetheless poses a significant challenge.

In general indoor environments, a compelling sensor choice is a two-dimensional Lidar, which employs time-of-flight technology to obtain fast and accurate range information about the environment at high angular and spatial resolutions. As such, the formulation of the problem in this project focuses primarily on the fusion of the wheel encoder odometry and the laser-scan data to match against a prior map.

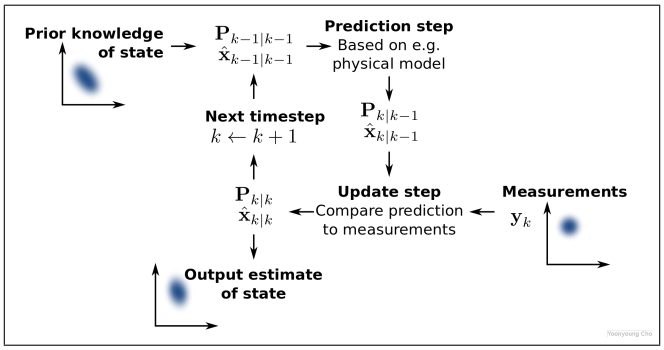


Fig. 1: Illustration of the Kalman Filter operation from Wikipedia.

2 Background

Algorithms employed in localization problems handle the integration of multiple sensors in order to compute the best estimate of the current position of the platform. As the sensors produce possibly conflicting measurements at different rates and quality, such a fusion requires a *filter*.

While there are many variants of specialized filters, such as *Mahony* or *Madgwick* filters for AHRS systems with Inertial Measurement Units, two of the representative filters that are applicable in the general domain are addressed: the *Kalman filter* and the *Particle filter*.

2.1 Kalman Filters

The Kalman filter is an efficient signal processing algorithm for sensor fusion known for being almost directly responsible for the success of the historic lunar landing. The basic premise is to have a statistically justified *optimal* method to fuse between current state estimation, state update and measurement data. The Kalman Filter achieves this by incorporating the *covariance* of the data, which represents the uncertainty of the information as high variance indicates lower confidence.

Traditional, most basic formulation of Kalman filters operate well for linear problems on state transition and observation models; this is often not the case in the real world that's composed of complex manifolds and cyclic data such as angular variates that are commonly seen in robotic mobile platforms. An EKF, or an *Extended Kalman Filter*, addresses this issue by linearizing the domain at the currently estimated state and applying the Kalman Filter accordingly. This is analogous to a first-order expansion of the modelling function, where the Jacobian is evaluated at each iteration for variance estimation and the proceeding state update logic.

There exist other variants of the Kalman filter that approaches approximation of the nonlinear function in different ways. A popular example is the UKF, or the *Unscented Kalman Filter*, which employs the unscented transform to represent the nonlinear mappings on the distributions through a sampling of sigma points from the current estimate and approximating the resultant distribution with another gaussian at the final stage.

2.2 Particle Filters

A particle filter is an application of the monte carlo method for representing the system state and how well each state corresponds to the measurement data. Much as how a histogram might represent a continuous statistical distribution, the particle filter approximates the underlying distribution with a sufficiently large population of discrete particles spread across the state-space.

In many ways, it is also an extension of genetic algorithms: in short, various candidates are maintained for

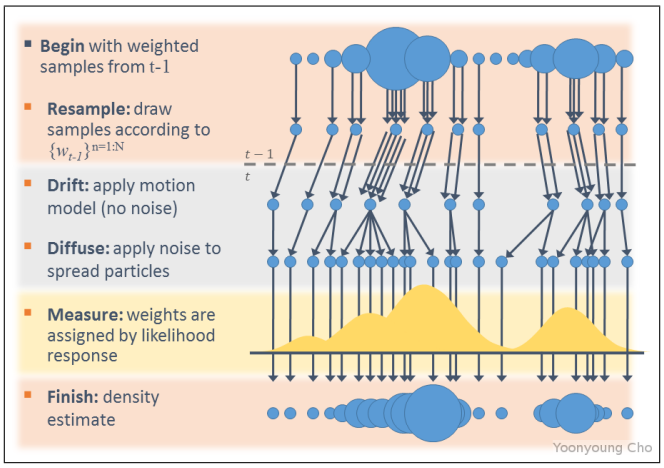


Fig. 2: Illustration of the Particle Filter operation from CodeProject.

ground truth estimation; at each step, each particle is updated according to the defined state transition and matched against measurement data, which yields respective weights on the particles. This weight is then applied to filter for the best-matching, most likely estimate of the current state among the representative particles; based on the metric, better particles are re-sampled at a higher probability, and mutated with noise to prevent convergence.

While incredibly effective across a wide range of problem domains, the key feature of the particle filter is that it is easy to intuitively understand its operations and implement the individual logical units.

2.3 Comparison / Contrast

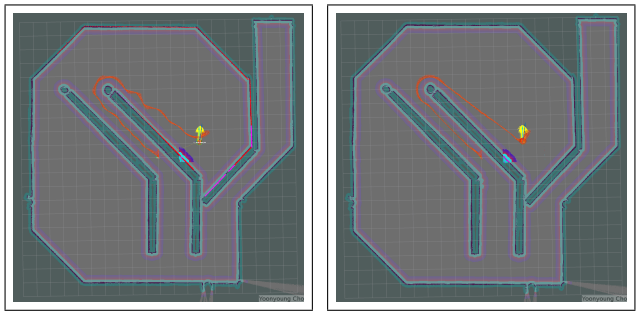
One key assumption on the Kalman-flavored filters is the assumption of the underlying distribution to be uni-modal gaussian. While this is a reasonable *default* assumption considering the central limit theorem and commonly observed noise models, such a naive approach does not always represent the reality accurately. Although historical data show that EKF *can* operate in conditions where several of its key assumptions are broken, a particle filter can be a more enticing candidate where no rigorous statistical justification for the motion and measurement models can be given, and in the domains where the underlying models can be quite complex. It is worth noting that, in general, the particle filter is more computationally intensive than the Kalman Filter, and often allows for more fluid parameter tuning processes.

Note that in this project, the localization method of choice is the particle filter. While the types of feature used for each filter may vary, the sensor data used for the particular localization problem favors the particle filter in that the observation function for the Kalman Filter with the laser scan data is more complex and does not easily produce a direct, non-ambiguous conversion to state residuals.

3 Simulations

The whole of the project had been executed on the Gazebo simulation, a development by the OSRF foundation to provide a virtual world in which mobile platforms may operate without significant risk or cost. The simulation interfaces directly with ROS, in that the provided data can be directly channeled to ROS message streams to apply the same general code architecture in both simulated and real-world environments, provided that appropriate parameters had been specified.

3.1 Achievements



(a) Default Model (b) Custom Model

Fig. 3: Execution trajectory visualization from each model; note that both models have successfully reach the goal location.

While specific comparisons will be addressed at later sections, Both models, at a high level, complete the localization and navigation tasks at a compelling performance.

3.2 Benchmark Model

3.2.1 Model design

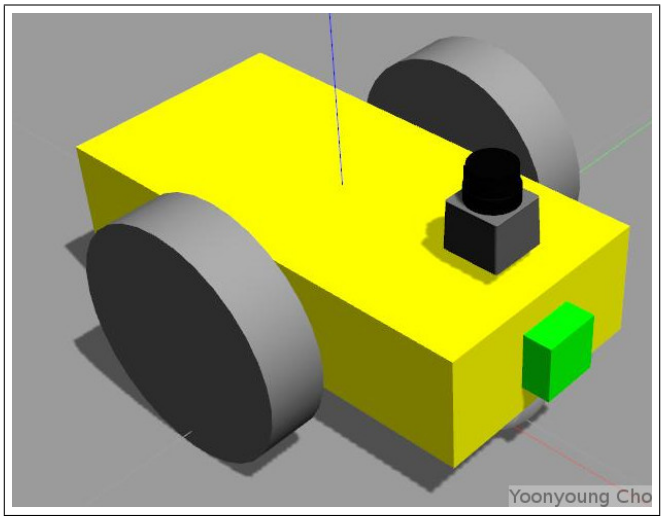


Fig. 4: Default Udacity model as constructed through the project specifications, decorated with custom color palette.

Initially, I started with a simple estimation of the rectangular bounding box around the robot as the footprint to be used in navigation. However, this entailed a more conservative estimate of where the robot could navigate to, on top of the already existing inflation layers on the costmap. As this caused issues where the navigational stack would enter into recovery mode too often and cause failures in the pipeline, I adjusted the footprint to be a tighter convex hull around the robot.

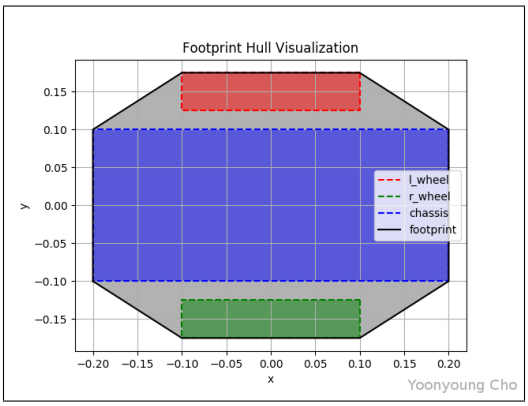


Fig. 5: Footprint hull visualization; note that wrapping tightly around the corner region reduces the effective radius significantly, thus serving a more accurate representation.

3.2.2 Packages Used

Overall, there are overall only a handful of processing nodes in the project; for a rough description of each package, the *amcl* package is the main localization package, which stands for *Adaptive Monte Carlo Localization*. the *map_server* package provides the static map server for the pre-built map, while the *robot_state_publisher* package publishes the current state of the robot based on the URDF description to the *tf* tree.

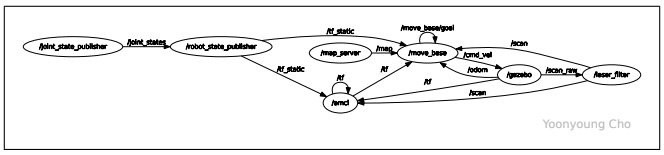


Fig. 6: ROS Graph representation of the computational nodes and interactions in the current configuration.

The *move_base* package is the main navigation stack that utilizes plugins for the local and global costmaps and planners to move the mobile platform to a target destination[1].

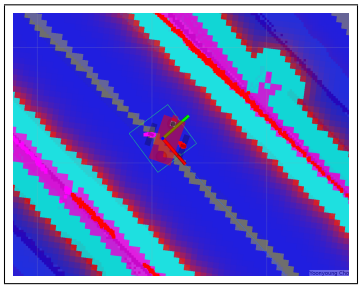


Fig. 7: Demonstration of scan self-intersection; note the scan data reported on the wheels, causing wall scans to be omitted at the side of the robot.

One interesting aspect of the default supplied model is that the size of the wheel is large enough that the visual field of the laser rangefinder can detect the wheels as well. As this could cause perplexing artifacts in robot mapping processes, as well as in the obstacle-avoidance behavior of the navigation stack, I had to incorporate a range-based filter from the *laser_filters* ROS package to remove such self-intersecting regions from the incoming scans.

1. note that some of the less relevant packages, such as the sensor plugins, have been omitted from the enumeration.

3.2.3 Parameters

TABLE 1: AMCL Odometry Model parameters, as formulated in *Probabilistic Robotics*.

Parameter	Value
α_1	0.01
α_2	0.005
α_3	0.01
α_4	0.02

A recent bug in the AMCL package was discovered, introducing a new model type named *diff-corrected*. This rendered the default parameters prone to error, so the parameters were adjusted accordingly, as seen in table 1.

$$\begin{pmatrix}\hat{v}\\ \hat{\omega}\end{pmatrix} = \begin{pmatrix}v\\ \omega\end{pmatrix} + \begin{pmatrix}\mathcal{E}(\alpha_1|v| + \alpha_2|\omega|)\\ \mathcal{E}(\alpha_3|v| + \alpha_4|\omega|)\end{pmatrix} \quad (1)$$

Equation 1, copied here from *Probabilistic Robotics*, is essentially where the coefficients originate; In a cursory description, these parameters specify the mapping between the true and commanded velocities, or the estimated scale of the motion error in the transition function between rotational and translational components; the parameters were configured such that the variance resulting from corresponding components was generally larger, with the rotational error estimated to be higher than translational error.

As for the *move_base* parameters, a few of the critical parameters were as follows:

TABLE 2: Navigation stack parameters

Parameter	Value
inflation	0.3m
local costmap dimension	3m
local planner	TEB
goal rotation tolerance	0.1 rad
goal position tolerance	0.2 m

Other parameters were either kept at default values, or specified the dynamic limitations of the robot.

Note that the *base_local_planner* and the *dwa_local_planner* configurations underwent significantly more rigorous tuning procedures with methods such as *dynamic_reconfigure*, but was ultimately discarded due to apparent bugs in the implementation that failed to follow the global trajectory correctly.

3.3 Personal Model

3.3.1 Model design

The size of the custom model remained at a similar radius (0.25m) as the largest radial distance of the supplied model (0.22m), in order to balance between posing a significant navigational challenge in the simulated arena by not being too small, while avoiding to introduce unnecessarily harsh conditions by being too large.

One important design decision on the custom robot model is the choice of odometry source: whereas the provided model sourced the information directly from the 3d

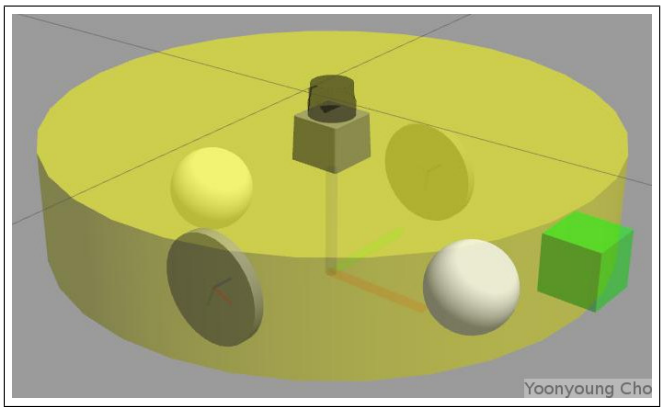


Fig. 8: Custom-Bot model visualization; note the semi-transparent render of the model, intended to show the embedded wheels at the bottom.

pose of the robot in the simulation (hence the term *world*), the custom model computed the odometry from the rotation of the wheel joints. This reflected the physical process of obtaining wheel encoder odometry from robot motion more accurately, and exposed bugs in the original model such as incorrectly specified wheel separation (0.4m, as opposed to 0.3m).

In order to simplify the collision computation and to ensure that the robot could not get *stuck*, the base frame was designed to be circular, with the wheel embedded below the body. In order to allow for the wheels to be beneath the body, the radius of the wheels had to be reduced as well. Overall, the form factor of the robot was heavily inspired by the roomba-like platforms.

Finally, the laser rangefinder sensor was relocated to the center of the base platform in order to simplify the overall obstacle distance model; applying transformations on the lidar sensor necessarily introduces slight errors in the produced rays, such that the footprint origin must coincide with the lidar location in order to satisfy the assumption exactly.

3.3.2 Parameters

One issue with the personal model was that AMCL performance appeared to degrade significantly with the encoder-based odometry model, which was in fact directly due to the errors due to wheel slip in the simulation. As it turns out, the model was not behaving properly with the modified model configuration due to a perplexing bug.

TABLE 3: Gazebo simulation contact coefficients, provided here for reference.

Parameter	Value
μ_1	10.0
μ_2	5.0
k_p	$1.0 \cdot 10^9$
k_d	$2.0 \cdot 10^5$
min_d	0.01

Tuning the parameters for the AMCL node was fairly straightforward after the contact coefficients directly contributing to the physics computation in Gazebo simulation were tuned correctly, reducing the amount of wheel-slip and thus ending with a better odometry information from the encoders.

Thus, the AMCL parameters used for the custom model was identical to the one used for the previous model.

4 Results

4.1 Localization Results

In order to assess the performance of each model, the localization error of the mobile platforms at each configuration were computed by comparing against the ground truth pose data from the *gazebo_ros_p3d* plugin that streams the model positions from the simulation to a *nav_msgs/Odometry* topic.

4.1.1 Benchmark

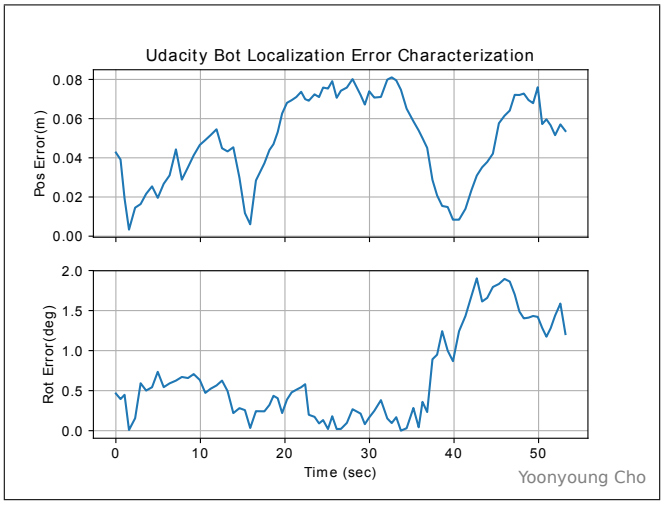


Fig. 9: Udacity-Bot error visualization during execution.

Overall, the robot remained in a relatively controlled region of below 0.1m and 2.0 degree deviation from the ground truth throughout the mission. The error profile can be seen in Fig. 9.

4.1.2 Student

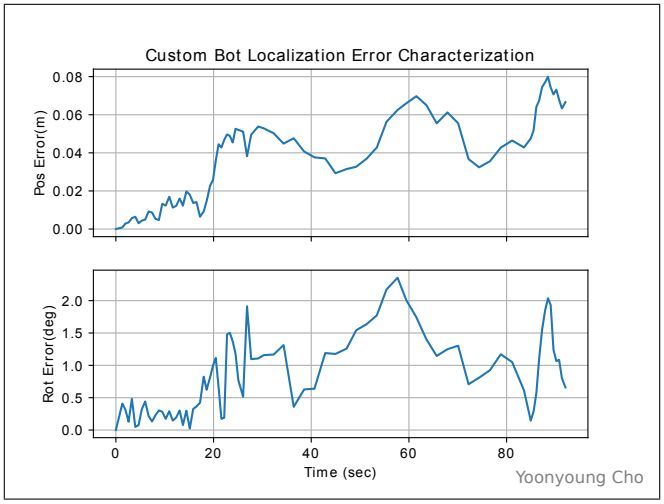


Fig. 10: Custom-Bot error visualization during execution.

Overall, the robot remained in a similar region of below 0.1m and 2.0 degree deviation from the ground truth throughout the mission. The error profile can be seen in Fig. 10.

4.2 Technical Comparison

While both robots achieved similar error profiles during the mission, the latter achieved a smoother trajectory, which can be advantageous in rougher, more unpredictable environments by reducing the effective collision profile throughout the navigation period.

However, the execution time for the latter was significantly greater, at approximately 93 seconds versus the 53 seconds from the default model. This is approximately 75% increase, which poses significant risk on time-critical missions.

5 Discussion

5.1 Comparison

Based on the navigation trajectory, which looks nearly identical, the cause of the increased time consumption on the custom model can only be that the platform was travelling at a slower speed. Considering the amount of time and effort that was required to engineer the contact coefficient parameters, it is quite likely that the dynamic properties of the robot such as maximum effective speed against motor torque was adversely affected by the simulation.

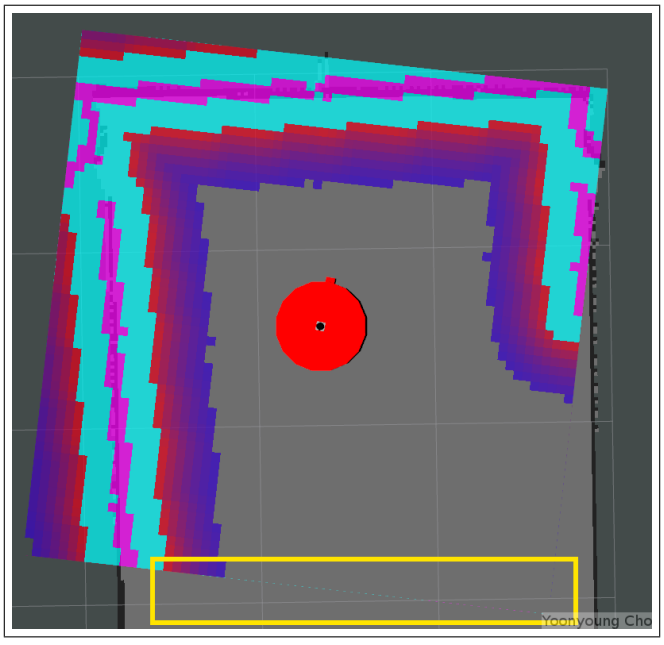


Fig. 11: Local costmap artifacts, highlighted in yellow. As the traces are quite small, it is recommended to zoom into the image on an electronic copy.

As another possible explanation, it is possible that artifacts from the previous scans failed to be cleared due to numerical precision and remained in the local costmap boundaries, keeping the closest obstacle location at a persistent 1.5m from the robot, which would have influenced the robot's navigational strategies to lean on the conservative side to avoid collision. From the execution for the default model, such an artifact was not discovered.

5.2 Algorithm

One particular extension that was added to the supplied *navigation_goal* script was the ability to optionally reset the particle filter initialization pose before navigation task was executed. This was done to ease the testing process and to be able to focus the development efforts on navigation tasks when deemed necessary.

One critical insight from this is that the AMCL algorithm cannot rigorously handle the *Kidnapped Robot* problem; the particle filter requires at least a few valid candidates around the ground truth region, or within a close enough boundary such that the noise introduced during the evolution of the particles could account for such small deviations. Otherwise, the particle filter will fail entirely, converge to a completely wrong solution at low confidence, or require re-initialization, based on the variant of the filter used.

In industry, the AMCL algorithm will be most appropriate for scenarios in which the environment is relatively static, and a fairly clean pre-built map either exists or can be generated. While the algorithm can handle some dynamic obstacles during the matching process, it works best if the obstacle were known prior, or exists in a transient enough period not to adversely affect the particle samples entirely.

One possible application case could be in a warehouse, where the movements of the internal robots are deterministic, the environment is relatively static, the distribution of dynamic variables are few and sparse, and the locations of the operating agents within the environment are known or well-characterized.

6 Conclusion / Future Work

In the project, two variants of a differential drive robot equipped with a 2d lidar was modelled and tested against standard ROS localization and navigation stacks. Both models demonstrated compelling performance in a field-size enclosed arena constructed with straight geometries.

6.1 Modifications for Improvement

6.1.1 Localization

The performance of the AMCL algorithm is quite notoriously insufficient for rectifying against large rotational errors, which are also the most frequent sources of errors in the domain of localization. Often, the alternative is to run a full SLAM(Simultaneous Localization and Mapping) algorithms, which may be prohibitively expensive for running for every normal operations.

A more tractable solution would be a feature-based matching algorithm between the map and the laser scan. For instance, a map may be represented as a series of lines and curves that define the boundary between navigable regions and obstacles; in the proposed architecture, a feature-based matching algorithm for the particle filter could simultaneously report the confidence values for the particle and optionally propose a modification to the represented state of the particle which would increase the likelihood for the particle to match the true position, such as introducing a

small rotation to align with the discovered wall in the scan data, or a translation to locate a high-confidence feature at the correct location.

6.1.2 Navigation

Independent of the issue mentioned in the discussions section, the local costmap can keep a large region of persistent cache of previous scan data in conditions where the wheel encoder odometry does not provide sufficiently accurate transforms between the scans. While the particle filter can rectify the transforms afterwards, the cache remains as a virtual obstacle within the costmap which may completely disable the robot from following the global trajectory until an extremely fragile recovery behavior of the navigation stack is executed.

A possible improvement on this would be to keep a dynamic copy of the local costmap, maintaining a queue on the scan data limited by the number of scans in the queue and how old each data is. In this way, each rectified transform on the scan data can be registered accordingly without artifacts in the long term. One caveat to such a dynamic approach, however, is the increased memory requirements and a continuous re-computation of the local costmap whenever a new transform is registered.

6.2 Conclusion

While some of the proposed improvements may dramatically enhance the performance of localization and navigation for general robotic platforms, the implementation was deemed fairly ambitious and out of scope given the timeframe of the project. As such, the points above are addressed mainly as a potential direction in which the project could be extended in the future, should it be desired.

Although the performance demonstrated by the pipeline during the project was overall satisfactory, it is critical to beware of the limitations. The arena was composed of relatively large, coarse geometries with little defects in the field, and the platform was mostly operating in an isolated environment with ample computational resources and a well-characterized sensors.

As such, a future iteration of the project should aim to improve the overall robustness of the pipeline against corrupted input data, sensor/motor faults, compute-throttled scenarios, as well as environmental hazards such as reflective or transparent geometry and non-planar operating terrain.