Where Am I?

Lucas Wohlhart

Abstract—The goal of the project at hand is the examination of the capabilities of a widely used localization technique known as Adaptive Monte Carlo Localization (AMCL). This method is based on a particle filter algorithm which will be conceptionally compared to another localization strategy using a Kalman Filter. For the evaluation of the localization method, two different robots models are placed in a ROS/Gazebo/RViz simulation environment in which they have to navigate through a maze world to reach a specified goal position. These two mobile robots both feature a differential drive, one of which is given by the Udacity project description and the creation of the second (custom) robot model will be discussed within this report.

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

1 Introduction

THE capability of accurately determining the pose of a mobile robot is a crucial necessity for being able to autonomously navigate any given environment. The problem of localization poses three challenges to tackle, which are known as: local localization, global localization and the kidnapped robot problem. The goal of local localization is to keep track of the pose changes due to robot movement given a known initial pose. In global localization the robots pose starts out as being unknown and the objective is to determine the pose by comparing observations to a ground truth map. The kidnapped robot problem is similar to global localization since the goal is also to find the robots pose without knowing it's initial location, but with the challenging additional constraint that the robot might be randomly placed at an entirely different spot in the world at any time. This project tackles the problem of global localization which, if implemented and tuned correctly, enables the robots to autonomously navigate the maze world depicted in Figure 1 to reach a specified goal pose.

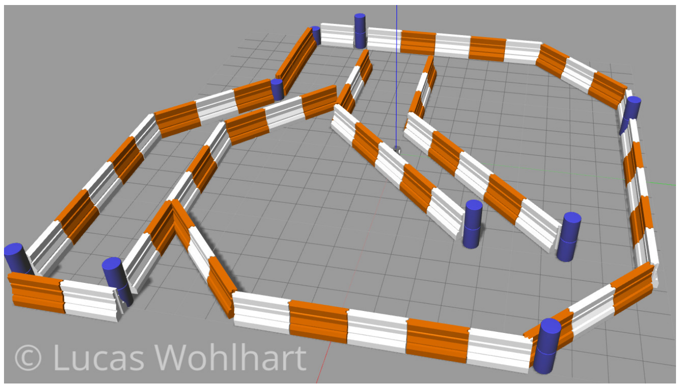


Fig. 1: Maze environment

2 Background

Solving a localization problem boils down to modelling the robots pose changes resulting from the motions executed by the actuators and interpreting the available sensor readings to gather evidence from the environment which can be compared to a known ground truth map. Both these processes are subject to noise since motions can't be performed with absolute accuracy and sensors don't capture the exact truth about the surrounding world. The objective of a localization algorithm is therefore to filter out the motion and measurement noise and yield a robust estimation of the robots pose.

The (Extended) Kalman Filter is a well established method for local localization and will discussed in comparison to the Adaptive Monte Carlo localization particle filter which is able to solve global localization problems.

Since in this project setup the initial pose of the robot is not known, the Adaptive Monte Carlo Localization (AMCL) method was chosen and implemented. Additionally this approach provides a simple way of modelling the nonlinear motions of the robot and is capable of solving the kidnapped robot problem as will be discussed in the following sections.

2.1 Kalman Filters

The Kalman Filter is a widely used state estimation algorithm which starts out with an initial state (e.g. a state vector holding the estimated x and y position and the angular orientation of a robot) as well as a covariance matrix expressing the certainty of the estimate and iteratively applies measurement updates and state predictions for each motion step of the robot.

The state prediction step takes the current motion commands of the robot, maps them to the state domain using the predefined state transition matrix and combines the result with the prior state and covariance plus additional gaussian noise to account for noisy motions. This yields the posterior belief about the robots state after a motion step.

The measurement update step uses the measurement function to map the previously calculated posterior state of the prediction step to the measurement domain. This means calculating the expected values of all available sensors, assuming the robot were in this exact state. Computing the difference between this assumption and the actual sensor readings yields the measurement residual. The so called Kalman Gain is then computed by projecting the covariance to the measurement domain, adding potential measurement noise to all components and reprojecting this result to the state domain. This is a measure indicating the ratio of preserving the previous state estimate and taking in the new measurement evidence. It ranges from a 0-vector which keeps the previous state estimate entirely, to the pseudo inverse of the measurement function which overrides the estimate with the incoming measurement data.

Therefore the Kalman Gain is subsequently used to update the state and covariance estimate.

One drawback of the standard Kalman Filter is that it can't deal with non linear state transitions or measurement functions. Applying a non linear transformation on a Gaussian distribution yields a probability distribution which is non Gaussian and since robots usually perform non linear motions such as driving along a circle or a curve the standard Kalman Filter is not usable for most robotics applications. To overcome this problem the Extended Kalman Filter (EKF) was introduced which linearizes the non-linear transformations by approximating them with a first order Taylor series expansion. This linearization is only valid for a small region but if the update cycles are short enough the EKF has shown to be able to produce a good and stable state/pose estimation.

2.2 Particle Filters

A particle filter algorithm for robot localization starts by instantiating a number of uniformly distributed particles each representing one guess about the position and orientation of the robot. For every particle a weight can be calculated which corresponds to the likelihood of observing the current readings from the available range-finder sensors (e.g. lidar, RGB-D cameras or others). If a particle matches the observation compared to the ground thruth deduced from the known map better, it gets a bigger weight than others which are less likely. Every motion step of the robot is also applied to each particle with some added noise. After a specific resample interval (usually also every motion step) the particles are then randomly resampled based on their weights and new particles might be spawned in proximity to the previously higher weighted particles.

By iteratively applying the algorithm:

- motion update: apply robots motion with noise to each particle
- sensor update: assign weights to particles based likelihood of matching sensor readings
- resample: randomly sample current particle set and insert new particles based on weighted probabilities

the particles converge to the true pose of the robot because those which best fit the incoming evidence from the sensors have the highest chance of surviving the resample step.

If correctly configured, the Adaptive Monte Carlo Localization particle filter is even capable of solving the kidnapped robot problem by sparsely inserting particles in randomly chosen remote areas of the particle cloud which allows the algorithm to recover when losing track of the actual pose due to the kidnapping.

2.3 Comparison / Contrast

As outlined previously, a major advantage of the Adaptive Monte Carlo Localization method is that it isn't restricted to the assumption of a linear gaussian state space as is the case for a Kalman Filter approach. Any motion or measurement noise model can be used to compute the posterior particle estimates. Furthermore the AMCL is able to solve global localization and even the kidnapped robot problem since it doesn't require an initial state to converge. It is capable of recovering from initial or intermediate erroneous state estimates which makes filter much more robust.

While the EKF is in general superior in terms of memory and computational efficiency as well as achievable localization accuracy the AMCL allows the user to fully take control of said parameters. By tuning the number of particles used for the filter one can adjust the tradeoff between localization accuracy and resource efficiency to fit the needs of the application.

Table 1, taken from the Udacity lecture "Power of MCL", summarizes the advantages and disadvantages of the discussed localization techniques.

TABLE 1: MCL vs EKF comparison

	MCL	EKF
Measurements	Raw Measurements	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Memory Efficiency	+	+ +
Time Efficiency	+	+ +
Ease of Implementation	+ +	+
Resolution	+	+ +
Robustness	+ +	-
Memory/Resolution Control	+	-
Global Localization	+	-
State Space	Multimodal Disc.	Unimodal Cont.

3 Simulations

To test the capabilities of the Adaptive Monte Carlo Localization method two robots were simulated in a ROS environment using Gazebo and RViz.

3.1 Achievements

After tweaking all necessary parameters for the involved software packages, both robot models were able to accurately localize themselves in the environment and successfully navigate through the provided maze.

3.2 Benchmark Model

3.2.1 Model design

The baseline robot model called udacity_bot was designed using the URDF (Unified Robot Description Format) methodology, which is a XML based description standard allowing the creator to define the topology of a robot. Using the SDF (Simulation Description Format), an extension to URDF introduced for Gazebo simulations, one can declare virtual sensors and actuators for the robot which are then available to use in ROS. By creating basic geometric objects, including 3D mesh models and linking the individual parts with joint description one can easily create a simple robot which can be rendered and simulated.

The udacity_bot is a differential drive robot consisting of a simple box shaped rigid body with support caster spheres for balance, which has a front facing camera and a laser range sensor attached. The wheels are modeled as cylinders connected with continuous joints to the rigid body. The dimension of the individual parts is shown in detail in Table 2 and the resulting robot model in Gazebo simulation is depicted in Figure 2.

TABLE 2: udacity_bot model

Part	Geometry	Dimension
Chassis	Cube Origin	0.4 x 0.2 x 0.1 x: 0.0 y: 0.0 z: 0.0
Back/Front caster	Sphere Origin	radius: 0.0499 x: ± 0.15 y: 0.0 z: -0.05
Left/Right Wheel	Cylinder Origin	radius: 0.1, height: 0.05 x: 0.0 y: ± 0.15 z: 0.0
Camera Sensor	Cube Origin	0.05 x 0.05 x 0.05 x: 0.2 y: 0.0 z: 0.0
Laser Range Sensor (Hokuyo)	Mesh Origin	0.1 x 0.1 x 0.1 x: 0.15 y: 0.0 z: 0.1

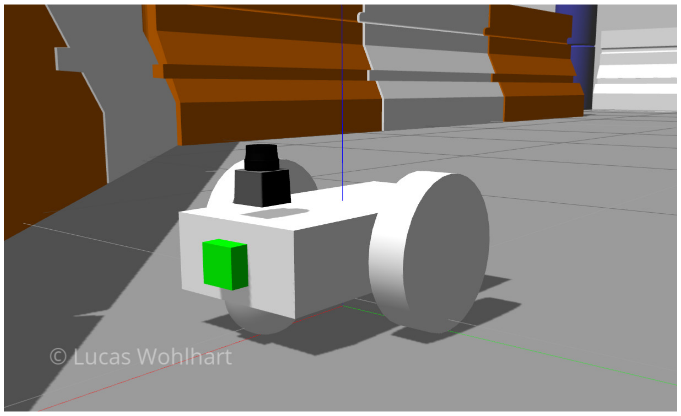


Fig. 2: Udacity Bot

3.2.2 Packages Used

The ROS package created for the simulation utilizes several components from the NavigationStack meta package as well as default ROS nodes. All necessary external packages, which are not present in the default ROS-Kinetic installation including Gazebo and RViz tooling, are declared as dependencies in the package.xml.

Required packages:

- rviz
- gazebo_ros
- joint_state_publisher
- robot_state_publisher
- map_server [1]
- amcl [2]
- move_base [3]

Launching the udacity_world.launch combines all required nodes to fire up the simulation environment. The gazebo_ros empty_world.launch starts a plain Gazebo physics simulation world and loads the world model declared in jackal_race.world. With the help of `gazebo_ros` spawn_model, `joint_state_publisher` and `robot_state_publisher` one can instantiate the URDF+SDF robot model defined in the robot_description.launch file.

The `map_server` package is used to load and publish the maze map defined in jackal_race.yaml and jackal_race.pgm.

Once the environment is set up, the `AMCL` node implementing the localization using a particle filter as described in section 2.2 starts, subscribing to the laser scan published from the hokuyo sensor and the map. The `move_base` package provides the move_base node which takes care of planning and executing the motions to navigate to a specified goal position. A custom C++ node is included in the package which simply uses the move_base action server to send this goal location and wait for the robot to arrive there.

3.2.3 Parameters

Tuning the parameters of the AMCL localization node as well as the path and motion planner node of the move_base package to the requirements of the application is the crucial aspect to solve the localization and navigation task. Initially following the guidelines of ROSNavigationGuide [4] and iteratively adapting the parameterization yielded the following configuration.

TABLE 3: Costmap parameterization

Costmap common	
obstacle_range	2.5
raytrace_range	3.0
transform_tolerance	0.2
robot_radius	0.25
inflation_radius	0.5
Global costmap	
update_frequency	10.0
publish_frequency	5.0
width	10.0
height	10.0
resolution	0.1
static_map	true
rolling_window	false
Local costmap	
update_frequency	10.0
publish_frequency	5.0
width	3.0
height	3.0
resolution	0.1
static_map	false
rolling_window	true

The parameters `obstacle_range` and `raytrace_range` define the distance of the laser range sensor readings to consider while updating the costmap with incoming evidence.

Since the measurement process and the update of the transformation tree is not time syncronised one has to allow for a small time delay by adjusting the `transform_tolerance`. Though, this tolerance shouldn't be excessive because updating the map based on an outdated pose estimate deteriorates the result.

The `inflation_radius` defines the extent to which the cost of a detected obstacle is affecting the unoccupied terrain around it. Together with the `robot_radius` it is used to define how strongly the navigation algorithm avoids coming close to obstacles.

Both maps are regulary updated and published with their respective frequencies. The global costmap is a static (10x10) map covering the world while the local costmap is not static but uses the rolling window approach to model a small section of the local surroundings of the robot. This way the local map can be relatively small (3x3) making it computationally less expensive to derive a local navigation policy. The resolution of both maps can be adjusted to fit the necessary granularity for collision free navigation while keeping memory and processing requirements minimal.

TABLE 4: NavFN local trajectory planner parameterization

Robot Configuration

acc_lim_x	2.0
acc_lim_theta	2.0
max_vel_x	0.4
min_vel_x	0.1
max_vel_theta	1.0
min_vel_theta	-1.0
min_in_place_vel_theta	0.4
escape_vel	-0.1
holonomic_robot	false

Forward Simulation

sim_time	1.5
sim_granularity	0.025
angular_sim_granularity	0.025
vx_samples	10
vtheta_samples	20
controller_frequency	10.0

Trajectory Scoring

meter_scoring	true
pdist_scale	1.0
gdist_scale	0.8
occdist_scale	0.01
dwa	true

The trajectory planner uses the Dynamic Window Approach (Fox et al. [5]) to derive a local navigation policy at each control cycle (`controller_frequency`) for the command velocities of the actuators, resulting in a piecewise approximation of the path with circular arcs. This is done by sampling directly from the subspace of rotational and translational velocities that are reachable within the next timeframe, given the current state and capabilities of the robot defined by acceleration and velocity limits.

The number of translational and rotational velocity pairs to consider can be configured with `vx_samples` and `vtheta_samples`. Each velocity proposal is forward simulated for a short time interval defined by `sim_time` with a step size of `sim_granularity` and `angular_sim_granularity`. Then it is rated with regards to the objective cost function balancing the tradeoff between sticking to the proposed path (`pdist`), reaching the next local goal (`gdist`) and staying away from obstacles (`occdist`).

The particle filter implemented in the AMCL package is configured to yield satisfactory localization performance while keeping the computational load manageable.

The main aspect to control is the amount of particles used for the filtering process. To solve the global localization problem at hand it suffices to operate with 50 to 200 particles. The `resample_interval` of 1 ensures a resampling of the particle set after every motion step while `update_min_d` and `update_min_a` declare the minimum translational and angular change between steps to update.

TABLE 5: AMCL parameterization

min_particles	50
max_particles	200
update_min_d	0.1
update_min_a	0.26
resample_interval	1
recovery_alpha_slow	0.0
recovery_alpha_fast	0.0
initial_pose_x	0.0
initial_pose_y	0.0
initial_cov_xx	20.0
initial_cov_yy	20.0
laser_max_beams	60
odom_alpha1	0.02
odom_alpha2	0.02
odom_alpha3	0.02
odom_alpha4	0.02

Setting the `initial_pose_(x|y)` to (0.0, 0.0) with an `initial_cov_(xx|yy)` of (20.0, 20.0) expresses very high uncertainty about the initial state since this a global localization problem. The resulting initial distribution of particles is shown in Figure 3.

The parameters `odom_alpha_(1,2,3,4)` specify the amount of noise added to the particles after a motion step to account for inaccurately executed motions.

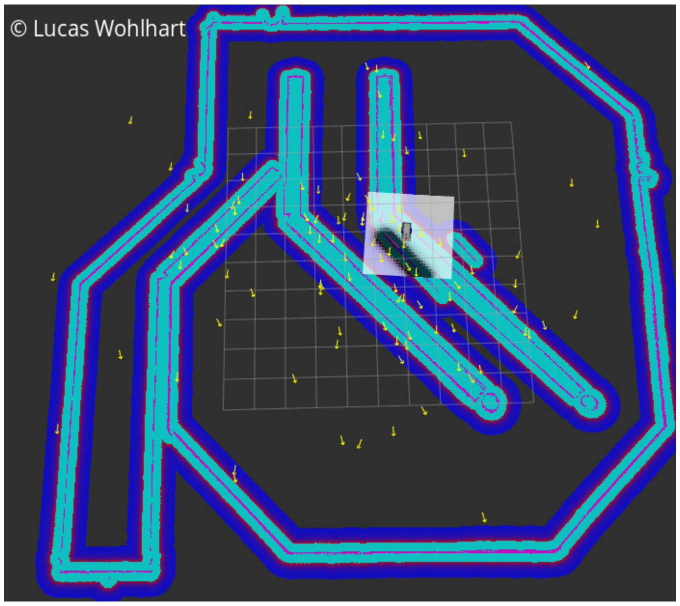


Fig. 3: Initial particle distribution

One could also cope with the possibility of having to solve the kidnapped robot problem by adjust the following parameters:

The parameters `recover_alpha_(slow|fast)` control the thresholds of average particle weights needed to randomly spawn new particles in unrelated locations to recover from erroneous pose estimates.

TABLE 6: AMCL parameterization for kidnapped robots

kld_err	0.005
kld_z	0.95
recovery_alpha_slow	0.01
recovery_alpha_fast	0.5

3.3 Personal Model

The custom robot model is inspired by the MSE-6-series repair droid [6] used by the Galactic Empire.

3.3.1 *Model design*

The chassis of the MSE-6 repair droid (shown in Figure 4) consists of a COLLADA mesh, modelled using the open source 3D modelling softwar blender [7]. The format is an XML based description for digital assets exchange, which can be included in a URDF file.

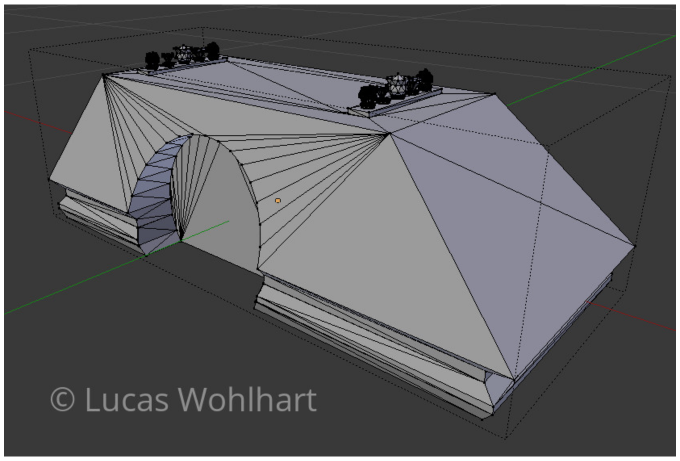


Fig. 4: Blender: MSE6 chassis design

TABLE 7: lw_mse6_bot model

Part	Geometry	Dimension
Chassis	Mesh Outer-Dimensions Origin	 0.675 x 0.3 x 0.236 x: 0.0375 y: 0.0 z: 0.062
Back/Front caster	Sphere Origin	radius: 0.0499 x: ± 0.15 y: 0.0 z: -0.05
Left/Right Wheel	Cylinder Origin	radius: 0.1, height: 0.05 x: 0.0 y: ± 0.15 z: 0.0
Camera Sensor	Cube Origin	0.05 x 0.05 x 0.05 x: 0.3 y: 0.0 z: 0.08
Laser Range Sensor (Hokuyo)	Mesh Origin	0.1 x 0.1 x 0.1 x: 0.2 y: 0.0 z: 0.15

The chassis is placed on a a back- and front-caster for stability and linked to two cylindrical wheels resulting in a differential drive robot. The laser range sensor is mounted on top of the chassis and the camera sensor sits on front of the robot. The entire robot is depicted in Figure 5 and the dimensions are declared in Table 7.

3.3.2 *Packages Used*

The MSE6 bot uses the same ROS packages as the udacity_bot described in section 3.2.2

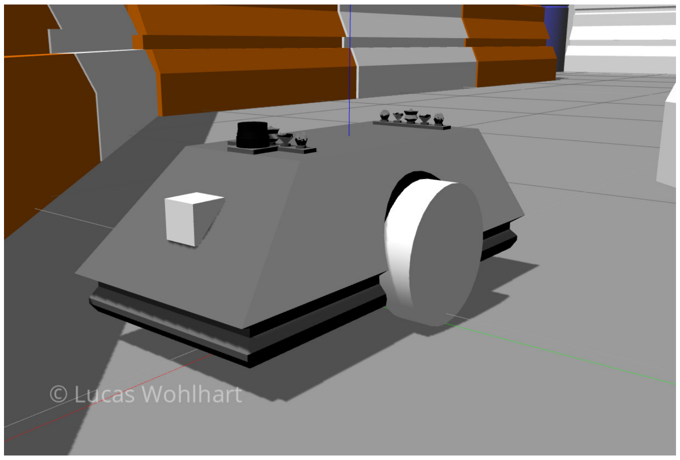


Fig. 5: MSE6 Bot

3.3.3 *Parameters*

Most of the parameterization used for the udacity_bot works for the MSE6 bot, but the newly introduced geometry of the chassis requires adaptation of the costmap common parameters. Since the robots hitbox can't be approximated by a circle anymore it's necessary to declare its `footprint` instead of the `robot_radius`. Additionally it has proven to be beneficial to increase the `inflation_radius` to 0.7 to prevent obstacle collisions.

costmap common parameters

footprint	[[0.375, 0.25], [0.375, -0.25], [-0.3, -0.25], [-0.3, 0.25]]
inflation_radius	0.7

In order to stop the hokuyo laser range sensor from hitting parts of the chassis the minimum range of gazebo configuration for the hokuyo sensor had to be increased to 0.15.

4 Results

Both robots were able to accurately localize themselves in the maze and navigate successfully to the target goal location. The average time required to finish the task is stated in Table 8.

TABLE 8: Time to reach goal

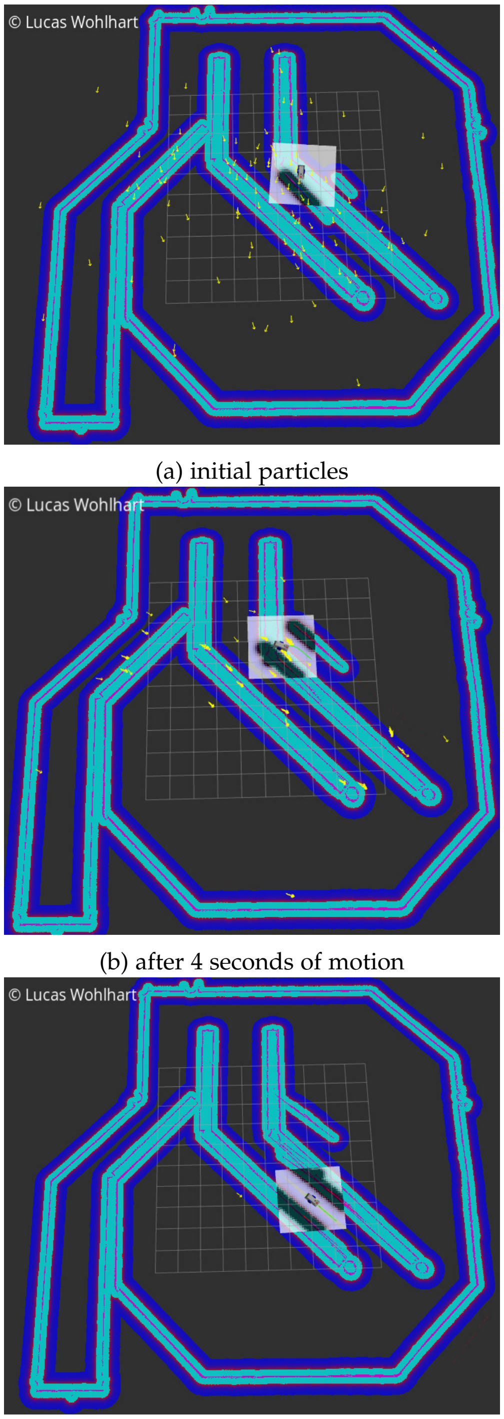
robot	avg. time to goal [seconds]	standard deviation
udacity_bot	78.50	4.28
lw_mse6_bot	81.04	6.69

TODO Present an unbiased view of your robot's performance and justify your stance with facts. Do the localization results look reasonable? What is the duration for the particle filters to converge? How long does it take for the robot to reach the goal? Does it follow a smooth path to the goal? Does it have unexpected behavior in the process?
For demonstrating your results, it is incredibly useful to have some watermarked charts, tables, and/or graphs for the reader to review. This makes ingesting the information quicker and easier.

4.1 Localization Results

4.1.1 Benchmark

The particle filter localization quickly converges to a satisfactory pose estimate after approximately 8 seconds of motion which can be observed in Figure 6. The initial distribution of particles covers the majority of the map which can yield a very inaccurate state assumption, but after a couple of motion steps and resampling intervals the estimate localizes the robot well.



(a) initial particles

(b) after 4 seconds of motion

(c) after 8 seconds of motion

Fig. 6: Convergence of AMCL particle filter

When the udacity_bot reaches the goal position the particles match its location rather accurately.

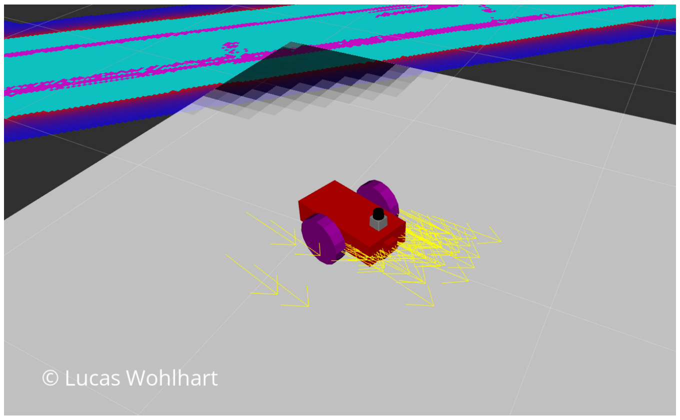


Fig. 7: Udacity Bot at goal location

4.1.2 Student

The MSE-6 bot is also able to localize itself perfectly and manages to navigate the maze to the goal position.

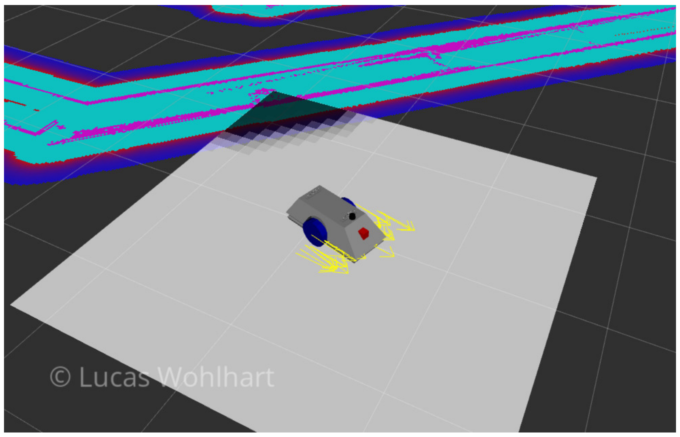


Fig. 8: MSE6 Bot at goal location

4.2 Technical Comparison

The udacity_bot performed the navigation task slightly faster as stated in Table 8. The marginally heavier and unevenly distributed chassis body of the MSE-6 causes the robot to be less agile, especially considering the turn rate. The positioning of the laser range sensor on the MSE6-bot blocks some of the rays at the edges of the scanner, but this doesn't affect the localization quality dramatically.

5 Discussion

In general the localization result of both robots are very satisfactory. Even with a relatively small amount of particles the AMCL algorithm is capable of solving the global localization problem.

One might assume that the superior performance of the udacity_bot model is due to the advantageous geometry of the robot. The MSE-6 bot has to operate with a bigger footprint as well as an increased inflation_radius of the costmap to guarantee collision free navigation. Additionally the simulation of the comparably complex chassis mesh imposes a higher computational load, especially considering that it is also used for the collision model of the robot, which might also contribute to slowing down the MSE-6.

The experiments concerning the 'Kidnapped Robot' problem, using the `recover_alpha_(slow|fast)` parameters of the AMCL node as stated in Table 6, didn't yield overly impressive results. Eventhough the robot was eventually capable of recovering from a kidnapping, simulated by manually injecting a 2D Pose Estimate in RViz, the strategies the robot executes in order to regain orientation in its environment don't always gather helpful insights. The insertion of new random particles works nicely, but the robot obviously can't distinguish between ambiguous-looking locations and sometimes these particles even cause the robot to wrongly think it's stuck in non-navigable terrain.

Under the assumption that one is able to provide an accurate ground-truth map of a relatively static and bounded environment, the AMCL algorithm is definitely able to solve global localization (and with proper parameterization and recovery strategies even kidnapped robot) problems. For example a warehouse scenario in which one robot operates alone in a confined space would be a very good fit.

6 Conclusion / Future work

To put it in a nutshell, the AMCL particle filter appears to be a very powerful tool for solving localization tasks. Eventhough it requires a finetuned parameterization to meet the necessary accuracy and still be computationally feasible, which can be a very tedious process, the implementation itself is relatively easy and produces good pose estimates.

Modelling a robot with URDF and Gazebo has shown to be a good approach to quickly develop and simulate custom robot models. With the help of 3D modelling tools one can design intricate geometries.

In the future, implementing the localization method on a real world robot would definitely require adjustments with regards to the noise assumptions for the motion and the sensor models. Additionally one would have to adapt the parameterization of the algorithm to meet the possibly limited computing capabilities of a mobile robot, in order to guarantee the operation of the actuator controller at the necessary frequency.

Furthermore the use of additional localization sensors, be it a second laser range sensor an RGB-D camera or other range finder technologies, would be an interesting and most likely beneficial addition.

References

[1] "ROS Navigation map_server." https://github.com/ros-planning/navigation/tree/kinetic-devel/map_server.
[2] "ROS Navigation amcl." https://github.com/ros-planning/navigation/tree/kinetic-devel/amcl.
[3] "ROS Navigation move_base." https://github.com/ros-planning/navigation/tree/kinetic-devel/move_base.
[4] "ROSNavigationGuide." https://github.com/zkytony/ROSNavigationGuide/blob/master/main.pdf.
[5] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance.," *IEEE Robotics and Automation*, vol. 4, January 1997.
[6] "MSE-6-series repair droid." http://starwars.wikia.com/wiki/MSE-6-series_repair_droid.
[7] "Blender." https://www.blender.org/.