Realtime Localization using AMCL

Christian Rabus

Abstract—This report introduces a particle filter algorithm for localization (Adaptive Monte Carlo Localization) which can be used to estimate the real-time pose of a robot. Further in this project, the algorithm is applied to two different robot models. The first model (benchmark model) is predefined and the second model (student model) is freely selectable. Both models are described with URDF (Unified Robot Description Format). Besides several ROS packages and plugins are utilized for applying the models, implementing the AMCL algorithm and navigating the robots. A map is provided for Gazebo and RViz simulation environment. After tuning several parameters of the AMCL, both robots have to navigate to a preset target position within the given map. Finally, the localization results of both robot models are compared.

Index Terms—Robot, Adaptive Monte Carlo Localization, Kalman Filters, Kalman Filter, Extended Kalman Filter, Localization, Gazeboo, RViz, Particle Filters, ROS, URDF.

1 Introduction

This report focuses on localization of robots. By the term of **Localization**, it is meant the challenge of determining the pose of a robot in a mapped environment. For this purpose probabilistic algorithms are implemented to filter noisy sensor measurements and track the position and orientation of the robot [1]. In a known environment, but without knowing its exact location, the robot moves around. After a few time steps and noisy sensor measurements, it then locates itself with some uncertainty in the map using a probabilistic model.
In general, there are 3 different localization problems in robotics, ascending by the difficulty to be solved:

- **Position Tracking**: easiest problem, robot knows its start position
- **Global Localization**: initial position of robot is unknown
- **Kidnapped Robot Problem**: robot is teleported to a different location within the map, robot has to relocate itself

To solve these localization problems, there are 4 different known models (algorithms) available today:

- **(Extended) Kalman Filter**
- **Monte Carlo Localization**
- **Marko Localization**
- **Grid Localization**

The first two methods will be discussed in more detail in the next section. Another difficulty that can be added is if the known environment changes over time (e.g. items are added, moved or removed).
In this report two different robot models (**Benchmark Model** and **Student Model**) are compared, which are supposed to navigate to a defined target position in a known map using the same localization algorithm approach. The robots know their initial position within the environment, so in this case it is a **Position Tracking** problem. Fig. 1 shows the given map in which the robots should find their way.

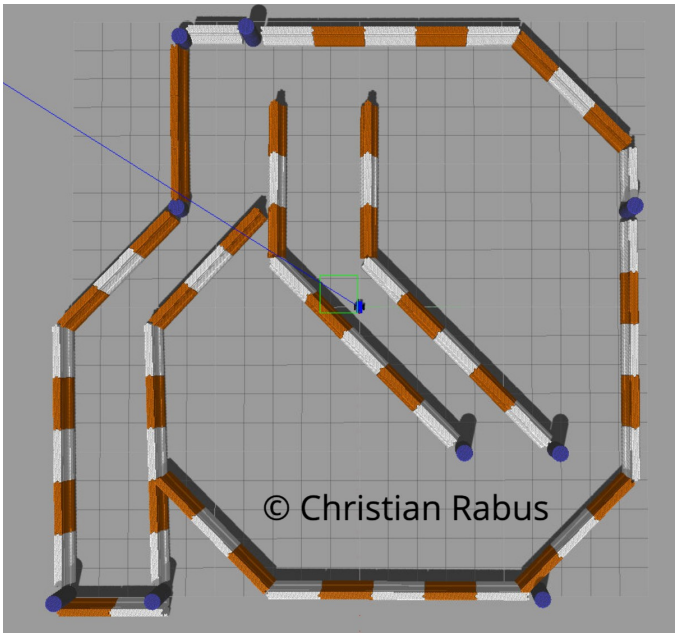

Fig. 1. Environment in Gazebo

2 Background

As already briefly explained in the introduction, this project focuses on navigating two different robot models in a mapped environment to a predefined position. Before a robot can drive to a defined target, it must know where it is on the map. So, the first step for a robot is to localize itself within the map by driving around. As soon as the robot knows where it is on the map, the same approach can be used to track the position of the robot and navigate to the destination. As already mentioned, a probabilistic model is used for this purpose, so called **Localization**. In this project, Adaptive Monte Carlo Localization (AMCL) is used. The advantages of this method are explained in section 2.3.
Before the robots can be navigated to a target using a localization algorithm, a development environment needs to be setup. In this project, a new ROS [2] package was created,

which loads the two models into a Gazebo world [3] and utilizes packages like AMCL [4] and the Navigation Stack [5]. To visualize the two robots in Gazebo, URDF (Unified Robot Description Format) [6] is used to describe the models and Xacro (XML Macros) [7] is used to implement the physics and sensors. Further Gazebo plugins are used for the Differential Drive Controller, the Camera Sensor and the Laser Range Sensor. For data visualization, RViz [8] is used. It can visualize any type of sensor data being published over a ROS topic like camera images, point clouds, Lidar data, etc. The shown map in Fig. 1 was created by Clearpath Robotics [9]. For loading the map, the map_server package [10] is used. For navigating to a goal position, the move_base package [11] within the Navigation Stack is used. It calculates a local costmap for the moving robot (with obstacles, walls etc.) in relation with a global costmap, where the path to the target position is defined.
As mentioned, the selected localization method is the Adaptive Monte Carlo Localization. The next sections will explain more detail about other probabilistic modells, and why AMCL was selected for this project.

2.1 Kalman Filters

A Kalman Filter is an estimation algorithm. It estimates the value of a variable (e.g. the position of a robot in an environment) in real-time as the data is being collected. The advantage of that filter is, that it can process data with a lot of uncertainty (e.g. very noised sensor measurement) and calculate a very accurate estimation of the real value. The filter can only process data that is based on a Gaussian distribution. The iterative process of the Kalman filter is divided into two steps: **measurement update** and **state prediction**. During the **measurement update** the prior belief (data with Gaussian distribution) is taken and updated by the sensor measurement (data with Gaussian distribution) to calculate the posterior (Gaussian with new distribution). In the next step, the **state prediction**, the posterior is taken and calculated with the motion (data with Gaussian distribution) to a new prediction, which becomes the new prior belief for the next **measurement update**. There are 3 different Kalman Filters:

- **Kalman Filter (KF)**: only applicable for linear systems (output proportional to input)
- **Extended Kalman Filter (EKF)**: used for non-linear systems(no proportional relation between input and output, using taylor approx. for linearization)
- **Unscented Kalman Filter (EKF)**: highly unlinear systems (using statistical methods for linearization)

2.2 Particle Filters

Another and more powerful method for localizing the position of a robot is by using a Particle Filter. The most popular one is better known as Monte Carlo Localization. For localization of the position of the robot, particles are distributed on the entire map at the beginning. Particles are a kind of virtual element that each represents the position and orientation of the robot, as well as a weight (probability) that the robot might be there. Each time the robot moves to a new position and senses the environment, these particles are re-sampled (depending on movement and sensing). After the re-sampling, particles with higher weight are more likely to survive, while the others more likely disappear. After a few steps, the particles converge and estimate the position of the robot. The algorithm used here, AMCL (Adaptive Monte Carlo Localization), also has the special feature that the number of particles decreases over time (depending on the accuracy of the estimated position).

2.3 Comparison / Contrast

Table 1 compares the two most common algorithms for localization [12]: Although MCL covers many more scenarios

TABLE 1
Comparison MCL and EKF

Rational	MCL	EKF
Measurements	Raw	Landmarks
Measurements Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency (memory)	+	++
Efficiency (time)	+	++
Ease of Implementation	++	+
Resolution	+	++
Robustness	++	-
Memory & Resolution Control	yes	no
Global Localization	yes	no
State Space	Multimodel Discrete	Unimodal Continuous

and has significant advantages over EKF, whether EKF or MCL should be used for localization depends on the use case:

- EKF over MCL might be better in a scenario where memory space is limited and efficiency time is crucial. However, the prerequisite is, that the transition function has to be known and in the worst case nonlinear, and as well the measurement noise must be Gaussian. Besides the initial position of the robot has to be known. A good example for EKF is a space rocket.
- A typical scenario for a robot where MCL is better than EKF is e.g. a service robot or vacuum cleaner robot. In those cases, normally the robot does not know its initial position. Besides the measurement nose ist mostly not Gaussian and it should be robust enough against external influences.

Therefore, in this underlying project, an Adaptive Monte Carlo Localization Algorithm is implemented.

3 SIMULATIONS

In section **2 Background**, the development environment has already been briefly described. Here, the two robot model designs, used packages and the chosen parameters are explained in more detail.

3.1 Achievements

The both designed robot models have reached with their specific settings and parameters the target position.

3.2 Benchmark Model

3.2.1 Model design

Fig. 2 shows the Benchmark Model. It consists of a chassis, two wheels, a camera and a laser.

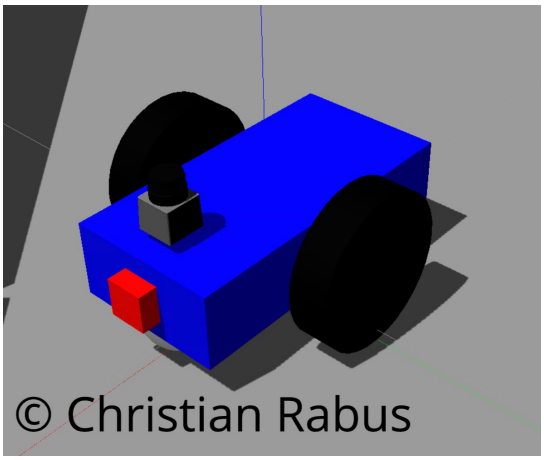

Fig. 2. Benchmark Model named udacity_bot

The following lists describes the model design of the benchmark robot [Fig. 2].

Robot Footprint

- parent link, no visual, acts as a base for the robot
- ”robot_footprint_joint” joint connects base with the chassis

Chassis

- blue cuboid (box), 0.4x0.2x0.1m
- 2 casters (spherical), radius 0.05m, white
- mass of 15, inertia of 0.1 for x, y and z axis
- collision geometry of casters smaller (radius 0.0499m) to avoid friction of casters

Wheels

- 2 cylinder (left and right), radius 0.1m, thickness (height) 0.05m, black
- mass of 5 each, inertia of 0.1 for x, y and z axis
- distance between wheels 0.3m (origin)
- 2 continuous joints linked to chassis (parent), damping 1.0, friction 1.0
- limits: velocity 1000, effort 10000

Camera

- box, 0.05x0.05x0.05m, red
- mass of 0.1, inertia of 1e-6 for x, y and z axis
- fixed joint to chassis, with 0.2m to wheel axis (y-axis)

Laser Rangefinder

- visual is mesh (grey/black in Fig)
- mass of 0.1, inertia of 1e-6 for x, y and z
- simplified collision geometry, box 0.1x0.1x0.1m
- fixed joint to chassis, with 0.15m x-axis and 0.1 to z-axis

In addition to the visual design (links, joints), plugins were also implemented so that the robot can drive and sensor data for camera and laser are available.

Differential Drive Plugin

- plugin: libgazebo_ros_diff_drive.so
  - always on (true) with update Rate 10
  - hinges of left and right wheel used for joints
  - wheel distance 0.4m, wheel diameter 0.2, torque 10
  - topics: cmd_vel (velocity), odom (Odometry)
  - robot footprint selected as base frame

Camera Plugin and Sensor Parameter

- sensor type: camera, name: camera1
- sensor settings: update rate 30, FOV 1.3962634, R8G8B8 800x800 image
- clip range from 0.02 to 300
- plugin: libgazebo_ros_camera.so
  - always on (true) with update Rate 0.0
  - topics: image_raw, camera_info
  - camera name: udacity_bot/camera1
  - frame name: camera
  - all distortions set to 0.0
  - hack baseline: 0.07

Laser Rangefinder Plugin and Sensor Parameter

- sensor type: ray, name: head_hokuyo_sensor
- pose: 0 0 0 0 0 0
- visualize: false
- update rate: 40
- ray (scan): horizontal, samples 720, resolution 1, min angle -PI/2, max angle PI/2
- ray (range): min 0.10, max 30.0, resolution 0.01
- ray (noise): gaussian, mean 0.0, stddev 0.01
- plugin: libgazebo_ros_laser.so
  - topics: /udacity_bot/laser/scan
  - frame name: hokuyo

3.2.2 Packages Used

The following ROS packages are used for the Benchmark Model:

- **gazebo_gui**, **gazebo** and **urdf_spawner**: for visualization of the robot in Gazebo
- **rviz**: for visualization of the sensor data (Laser, Camera, etc.)
- **joint_state_publisher**: publishes the joint state values of the robot (here, the angles of the two wheels)
- **robot_state_publisher**: publishes the state of the robot to transform tree, here the 3D poses
- **map_server**: loads the map
- **amcl**: implements the localization approach, here the Adaptive Monte Carlo Localization
- **move_base**: does actually the navigation of the robot to a goal position. It’s the ”heart” of the navigation stack where all information comes together.

These packages (or nodes) communicate together via the so-called **rostopics**. With the command *rostopic list* all rostopics can be displayed. Alternatively the nodes and their connections can be visualized with the command *rosrun*

rqt_graph rqt_graph. The following list shows the most important topics:

- **/joint_states**: joint state for each non-fixed joint in the robot
- **/tf**: information about the relationships between coordinate frames
- **/udacity_bot/laser/scan**: published sensor data
- **/move_base/goal** and **/move_base/current_goal**: target position where robot has to drive
- **/map**: the known map
- **/odom**: odometry information of robot
- **/cmd_vel**: velocity information of robot
- **/particlecloud**: displays the pose estimations (particles) of the amcl

3.2.3 Parameters

For a successful localization the parameters of the **amcl** and **move_base** node must be optimized. The following lists show the set values. The default settings apply to parameters that are not listed.

AMCL Parameter

- Param #01: **min_particles**: 20
- Param #02: **max_particles**: 150
- Param #03: **update_min_d**: 0.1
- Param #04: **update_min_a**: 0.3
- Param #05: **resample_interval**: 1
- Param #06: **transform_tolerance**: 0.2
- Param #07: **recovery_alpha_slow**: 0.001
- Param #08: **recovery_alpha_fast**: 0.1
- Param #09: **laser_z_hit**: 0.99
- Param #10: **laser_z_rand**: 0.01
- Param #11: **laser_sigma_hit**: 0.1
- Param #12: **odom_model_type**: diff-corrected
- Param #13: **base_frame_id**: robot_footprint
- Param #14: **odom_alpha1**: 0.005
- Param #15: **odom_alpha2**: 0.005
- Param #16: **odom_alpha3**: 0.010
- Param #17: **odom_alpha4**: 0.005

The particle range (#01,#02) has been reduced to save computational resources. Required translational (#03) and rotational (#04) movements have been reduced, so that filter update is performed earlier. Resample interval (#05) was set to 1, so that every filter update there is a resampling. Transform tolerance (#06) was set to 0.2 seconds, so that transform is longer valid. Recovery (#07,#08) was enabled, so that particles can be added, if required. The three parameters #09, #10 and #11 have been adjusted experimentally to better align the laser beams with the actual map. The 4 alpha parameters (#14-#17) have been tuned according to [13].

move_base Parameter

- **costmap_common_params.yaml**
  - Param #18: **obstacle_range**: 2.0
  - Param #19: **raytrace_range**: 3.0
  - Param #20: **transform_tolerance**: 0.2
  - Param #21: **inflation_radius**: 0.25
- **local_costmap_params.yaml**
  - Param #22: **update_frequency**: 10.0
  - Param #23: **publish_frequency**: 10.0
  - Param #24: **width**: 4
  - Param #25: **height**: 4
  - Param #26: **resolution**: 0.05
- **global_costmap_params.yaml**
  - Param #27: **update_frequency**: 10.0
  - Param #28: **publish_frequency**: 10.0
  - Param #29: **resolution**: 0.05
- **base_local_planner_params.yaml**
  - Param #30: **controller_frequency**: 5.0
  - Param #31: **meter_scoring**: true

The transform tolerance (#20) was set to avoid transform timeout. The inflation radius (#21) was set to 0.25m, so that the robot doesn't bump into the wall or obstacles. With the obstacle range (#18) of 2.0m all osbstacles within this range will be insterted into the cost map. Setting the raytrace range (#19) to 3.0m means, that all obstacles that are further away will be cleared from the cost map. The update frequency (#22) and publish frequency (#23) of the local costmap were set to 10Hz, due to a slow system. The size (#24,#25) of the map was reduced to 4x4m and a resolution (#26) of 0.05m to save system resources. Also the update frequency (#27) and publish frequency (#28) of the global costmap were set to 10Hz, due to a slow system. Besides, the resolution (#29) was set the same as for the local costmap. The controller frequency (#30) was reduced to 5Hz due to a slow system and the meter scoring (#31) was set to avoid other warnings.

3.3 Student Model

3.3.1 Model design

Fig. 3 shows the Student Model and shall represent a vacuum cleaner. It consists of a chassis, two wheels, a camera and laser.

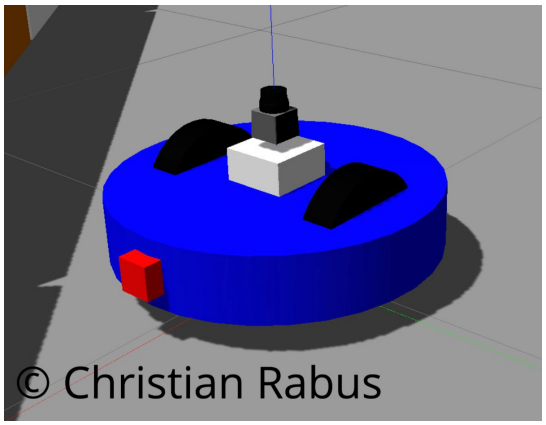

Fig. 3. Student Model named vacuum_cleaner

The following lists describes the model design of the benchmark robot [Fig. 3].

Robot Footprint

- parent link, no visual, acts as a base for the robot
- "robot_footprint_joint" joint connects base with the chassis

Chassis

- blue cylinder (box), radius 0.25m and height (length) 0.1m
- 2 casters (spherical), radius 0.05m, white
- laser socket (box), size 0.1x0.1x0.1m, white
- mass of 15, inertia of 0.1 for x, y and z axis
- collision geometry of casters smaller (radius 0.0499m) to avoid friction of casters

Wheels

- 2 cylinder (left and right), radius 0.1m, thickness (height) 0.05m, black
- mass of 5 each, inertia of 0.1 for x, y and z axis
- distance between wheels 0.3m (origin)
- 2 continuous joints linked to chassis (parent), damping 1.0, friction 1.0
- limits: velocity 1000, effort 10000

Camera

- box, 0.05x0.05x0.05m, red
- mass of 0.1, inertia of 1e-6 for x, y and z axis
- fixed joint to chassis, with 0.25m to wheel axis (y-axis)

Laser Rangefinder

- visual is mesh (grey/black in Fig)
- mass of 0.1, inertia of 1e-6 for x, y and z
- simplified collision geometry, box 0.1x0.1x0.1m
- fixed joint to chassis, centered between wheels and 0.15m off in z-axis

In addition to the visual design (links, joints), the same plugins (**Differential Drive Plugin**, **Camera Plugin**, **Laser Rangefinder Plugin**) as used as for the Benchmark Model had been implemented. The same values were used for the parameters.

3.3.2 Packages Used

For the Student Model, the same ROS packages were used as for the Benchmark Model. Consider section 3.2.2 for further details.

3.3.3 Parameters

For a successful localization the parameters of the **amcl** and **move_base** node have to be optimized and adjusted in comparison to the Benchmark Model.
Testing the **amcl** parameters of the benchmark model with the vacuum cleaner model, it can be noticed that these settings also seem suitable. Only a few changes of the **move_base** node had to be changed.

move_base Parameter

- **costmap_common_params.yaml**
  - Param #18: **obstacle_range**: 2.5
  - Param #19: **raytrace_range**: 3.5
  - Param #21: **inflation_radius**: 0.20

The thre parameters obstacle range (#18), raytrace range (#19) and inflation radius (#21) had to change, so that the vacuum cleaner successfully reaches the target position.

4 Results

After implementation of the two models and optimization of the parameters, both robots were sent to the target position. In Fig. 4 the start position of the Benchmark Model is shown. It also shows the randomly distributed particles (green arrows) and the laser beams on on the walls in the map. Fig. 6 and Fig. 8 show the benchmark model at the target position, where in Fig. 8 the alpha value of the robot was set to 0.1 to display it transparent. This makes it easier to view the orientation and distribution of the particles.
The same situations have also been captured for the student model: Fig. 4 represents the start position, Fig. 6 shows the goal position and Fig. 8 shows the goal position with alpha value set to 0.1.

4.1 Localization Results

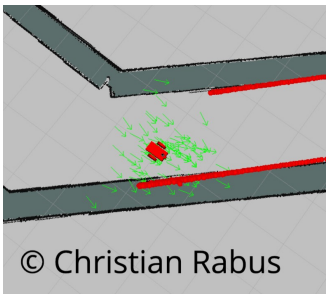

Fig. 4. Start position of the Benchmark Model

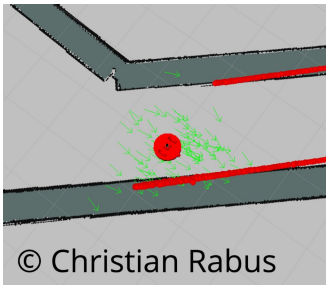

Fig. 5. Start position of the Student Model

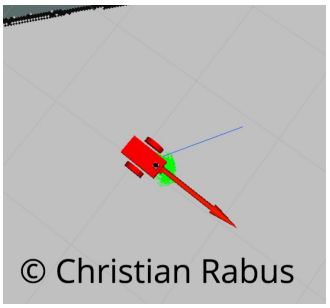

Fig. 6. Goal position of the Benchmark Model

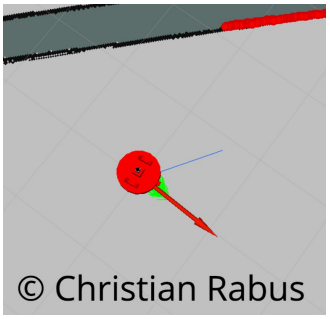

Fig. 7. Goal position of the Student Model

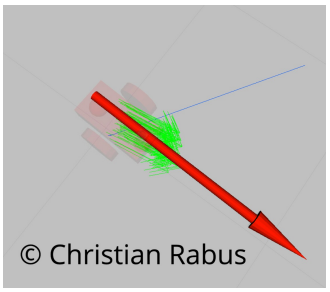

Fig. 8. Goal position of the Benchmark Model with alpha=0.1 of the robot model

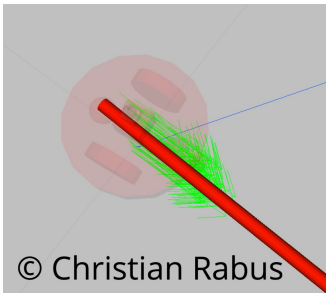

Fig. 9. Goal position of the Student Model with alpha=0.1 of the robot model

4.2 Technical Comparison

The two robot models differ primarily in the design and position of the sensors on the robot. The benchmark model has a cuboid as chassis, while the student model has a cylinder as chassis. In addition, the student model is larger in external dimensions. The Student Model also has a centered socket above the wheel axle on which the Laser Rangefinder sensor sits. The Benchmark Model, on the other hand, has a Laser Rangefinder sensor at the front. Both models have the camera at the front, while the student model has a larger distance to the center. Comparing Fig. 8 and Fig. 9, it seems that the particles of the student model are better aligned at the given target position.
In order to compare the performance of the two models, each model was sent 5 times to the target position and the required times were recorded.

TABLE 2
Recorded run times of robots

Run	Benchmark Model	Student Model
1	46.602s	48.601s
2	46.401s	70.800s
3	46.601s	47.600s
4	47.401s	47.001s
5	47.202s	48.001s
average w/ outliers	46.841s	52.401s
average w/o outliers	46.841s	47.801s

Table 2 shows the recorded times. Both robots successfully reached the target position for each run smoothly. Only the 2nd run of the Student Model shows a larger deviation and is therefore evaluated as an outlier.

5 Discussion

On the basis of the results (Table 2), it can easily be seen that the Benchmark Model is slightly faster than the Student Model. Comparing the runs (paths) of the Student Model with those of the Benchmark Model, it can be seen that the student model makes a larger arc around the lower corner pillar. Thereby the Student Model probably loses 1 second to the Benchmark Model. Watching the robots navigating, both the student model and the benchmark model do not choose the most cost-optimized path. With regard to path planning, both models can therefore still be optimized.
Observing the particles of the AMCL (parameters in both models equivalent) shows that within a few steps they converge to an accurate localization of the robot.
If the robot with implemented AMCL approach would be exposed to the "kidnapped robot problem" (removed from its known environment and replaced elsewhere), the algorithm would be able to **adapt** its particles accordingly. Due to its software architecture, the AMCL would initially increase the number of particles at the new unknown position, and once the robot has located itself after a few steps, the number of particles would decrease again. In addition, AMCL does not rely on landmarks, but on laser-based maps, laser scans and transform messages to estimate a position.
Localization can be applied in a variety of scenarios. AMCL has advantages where there are no known landmarks or the map does not exist yet and is created during navigation. It could also be environments that are highly dynamic and constantly exposed to changes. A good example would be delivery robots in city centres, at airports and in shopping centres with a lot of human traffic. Autonomous robots are already being used in large logistics centres (e.g. Amazon) to process goods.

6 Conclusion / Future work

In this report, the basic principles of localization have been sufficiently explained. The 3 most important localization problems were introduced and various solution algorithms (KF, EKF, MCL, AMCL) were presented. Kalman filters and Particle Filters were discussed and their advantages and disadvantages were presented.
In order to get to know and understand the functionality of the AMCL better, two robot models (Benchmark Model and Student Model) were implemented with the help of ROS and the corresponding packages. The goals was to first define the two different designs with the help of URDF, then to select the right sensors (camera, laser rangefinder) and finally to implement the localization algorithm (AMCL) with the help of the ROS navigation stack. After fine-tuning the AMCL and move_base parameter both models had to navigate to a given target position. With the help of the results, the two robots were compared and the Benchmark Model turned out to be slightly faster to reach the target.

6.1 Modifications for Improvement

The following list contains possible modifications for improvement that can be made to the robots:

- The robot models are very simple designs. For a more realistic representation, robot models should become more complex and detailed. However, this would have a negative effect on the process time.
- Due to time constraint it was not possible to examine if the different sensor positions between the two models have an impact on localization accuracy. This should be more investigated.
- The sensor parameters, both for the Camera and the Laser Rangerfinder were not adjusted. Here, too, it would have to be investigated how modifications can contribute to an improvement of the navigation accuracy.
- In addition to the two existing sensors, further sensors (e.g. IMU) could also be integrated, or existing sensors could be implemented several times. With more sensor data a higher accuracy would be expected, but also means a longer processing time.
- The parameters of AMCL and move_base have already been optimized and their influences on performance have been mostly investigated. Nevertheless, improvements can still be achieved here by investing more time to better investigate the influence of individual parameters.

6.2 Hardware Deployment

The idea of the Student Model is that of a robotic vacuum cleaner for home use. The following steps need to be done for hardware deployment:

1) More realistic design of the robot in the simulator and adjustment of the parameters
2) Selection of the right hardware components depending on availability, costs, power consumption, integration effort, etc.
3) Assembling of components and function testing
4) Deployment of software on the system
5) Testing of robot in real environment

REFERENCES

[1] Lesson 9, Introduction to Localization, Udacity Robotics Software Engineer Nanodegree Program.
[2] ROS, Robot Operating System, http://wiki.ros.org/.
[3] Gazebo, physical simulator, http://gazebosim.org/.
[4] AMCL, AMCL package for ROS, http://wiki.ros.org/amcl.
[5] Navgiation Stack, 2D navigation package for ROS, http://wiki.ros.org/navigation.
[6] URDF, http://wiki.ros.org/urdf.
[7] Xacro, XML macro language, http://wiki.ros.org/xacro.
[8] RViz, http://wiki.ros.org/rviz.
[9] Clearpath Robotics, https://www.clearpathrobotics.com/.
[10] map_server, http://wiki.ros.org/map_server.
[11] move_base, http://wiki.ros.org/move_base.
[12] Lesson 12, Monte Carlo Localization, Udacity Robotics Software Engineer Nanodegree Program.
[13] Tuning AMCL's diff-corrected and omni-corrected odom models, https://answers.ros.org/question/227811/tuning-amcls-diff-corrected-and-omni-corrected-odom-models/.