Where I am

ISRAILOV SARDOR

Abstract—Localisation is one of the most important aspects of Robotics. Although there exists many Localisation techniques, the two main techniques are Kalman and Particle Filters. Both of them have their advantages and disadvantages. In this work, two different robots were implemented, using simulation in GazeboRviz. The AMCL(Adaptive Monto Carlo Localisation) technique was applied with two different robot types, two wheels with differential drive, and 4 wheels. The main goal of the project is to correctly localise the robot, so that it could reach the desired goal position as well as orientation in a predefined map.

Index Terms—Robot, IEEEtran, AMCL, Kalman Filter, Udacity, ROS, Gazebo, Rviz LaTeX, Localization.

1 Introduction

Localisation is the process of determining the placement of the robot in its environment. There exists techniques to localize the robot in the unknown environment as well as already predefined maps. The main techniques for localisation in the known environment are the variations of Kalman Filters and different types of particle filters. Although extended kalman filter is compared to the AMCL, the main focus remains on the monto-carlo localisation technique in the known environment. The goal of this project is to create and tune the localisation stack(AMCL) for 2 robots to successfully navigate it to the goal.

1.1 Kalman Filters

Briefly describe Kalman filters. Explain how they work and why they are used for localization. Additionally, discuss the drawbacks of linear Kalman filters and how Extended Kalman Filters (EKFs) help resolve some of these issues.

1.2 Particle Filters

Briefly explain what a particle filter is, how it is used, and why it is useful. Adaptive Monte Carlo Localization (AMCL) dynamically adjusts the number of particles over a period of time, as the robot navigates around in a map. This adaptive process offers a significant computational advantage over MCL.

1.3 Comparison / Contrast AMCL vs. EKF

	Gaussian Distribution	Easiness	Computational power
AMCL	Not necessary	Yes	Big
EKF	Obligatory	No	Average

Easiness-implementation *Obligatory Gaussian distribution* is obtained by linearisation of non-linear function to obtain Gaussian distribution.

Adaptive Monte Carlo Localization was chosen because the sufficient computational power was available. At this stage, you should begin diving into the technical details of your approach by explaining to the reader what are the characteristics of the filters, what localization method was chosen, and the reason that it was selected (i.e. particle filters). This should be factual and authoritative, meaning you should not use language such as ”I think this will work” or ”Maybe Monte Carlo Localization with these parameters is better...”. Instead, focus on items similar to, ”Adaptive Monte Carlo Localization was chosen because...” Provide a sufficient background into the scope of the problem technologically while also identifying some of the current challenges in robot localization and why the problem domain is an important piece of robotics.

1.4 Introduction to steps to create viable ROS packages

The following steps were necessary to accomplish the results of the project

- create the udacity_bot package
- create 2 folders launch and worlds launch the package with ”roslaunch udacity_bot udacity_world.launch”
- Create the robot model: udacity_bot.xacro Create chassis(parent link), one wheel, using the 1st wheel as a template modify the position and the child link to create the 2nd link.
- add camera, laser range finder and add Gazebo plugins (like libgazebo_ros_diff_drive.so)
- create 2 nodes: joint_state_publisher robot_state_publisher, then integrate RVIZ by creating the node with name=”rviz”
- add the jackal_race.world to ”worlds” folder, and change world_name in the udacity_world.launch
- create amcl.launch file that would become the core for the particle filter that we implement. The package contains 2 nodes: map_server that would load a map, and localisation node(name=”amcl”).
- Implement navigation stack(move_base package) that use local and global costmaps to update a continuous way to the navigation goal. The local costmap is only displaying what the laser sensor captures during particular time period. Just like the amcl, or any other package, move_base also has its own set of required parameters that help it perform efficiently.

- add 4 configuration files for move_base package. In my case I also defined the amcl.yaml. PoseArray, in RViz helps to identify how certain is "amcl" algorithm in the robot pose(localisation).
- Tune THE PARAMETERS

In order to implement another robot, I needed to do the following:

- Define new xacro file, that would be used to generate URDF file.
- change the call in the robot_description.launch
- we change the number of wheels, the mechanics, so RETUNE THE PARAMETERS
- implement new techniques

2 Simulations

All robots consist of links and joints. The link has three basic elements: collision, visual and inertial. The role of "Joints" is to join links together. So, the good strategy is to define some parent element, for these robots-chassis, and to join other links to it. STRATEGY FOR TUNING Costmap is the package used to take sensor information, and build the occupancy grid mapping. The main parameters for costmap_common_params.yaml are obstacle_range(the range used to determine whether to add the obstacle to costmap),raytrace_range(used to clear and update the free space in the cost-map as the robot moves),inflation_radius. If the robots bumps into obstacles from time to time, it's reasonable to increase the inflation_radius. With other two parameters, it's good to find the compromise between the utility and the power. If the robot seems to turn right, left a lot, then it's reasonable to increase controller_frequency so that the robot corrects itself more rapidly. There are a lot of parameters the impact of which is hard to observe. I still have many parameters not optimised. Udacity_bot The friction of front/back caster was modeled with 2 coefficients for mu1, the friction coefficient in the movement direction and mu2, the friction coefficient in the perpendicular direction.

2.1 Achievements

The main componenets of robots created are 2/4 wheels with their joints, chassis(the main body), 1 camera and 1 hokuyo sensor(LiDAR). In order to implement the AMCL package in ROS, a lot of parameters have been tuned both for AMCL package and basic localisation and mapping as well as controller parameters.

2.2 Udacity_bot

2.2.1 Model design

The robot (URDF file) consisted of the following:

1) **robot_footprint**-parent link connected via
2) **robot_footprint_joint**(joint-fixed) with chassis
3) **chassis**-link
    - **chassis_visual**-visual box(*0.4x0.2x0.1*)
    - **back_caster_collision**-visual semi-sphere of(*r=0.5*)
    - **front_caster_collision**-visual semi-sphere of (*r=0.5*)
4) **left_wheel**-link with visual of shape *cylinder l=0.05; r=0.1*
5) **left_wheel_hinge**-continuous joint.
6) **right_wheel**-link with visual of shape *cylinder l=0.05; r=0.1*
7) **right_wheel_hinge**-continuous joint

2.2.2 Packages Used

rviz- advanced 3D visualisation tool for ROS, works with gazebo.

joint_state_publisher publishes joint_state values for given URDF

robot_state_publisher publishes robot_state values to tf.

tf keeps track of multiple coordinate systems over time. Makes the transformation from different systems to base system and vice versa.

map_server provides the map data as a ROS service.

robot_state_publisher publishes robot_state values to tf.

amcl - probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, a particle filter.

move_base provides an implementation of an action, that given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task.

In short, the move_base package will help navigate your robot to the goal position by creating or calculating a path from the initial position to the goal, and the amcl package will localize the robot.

Other navigation techniques included in move_base: **global_planner** provides an implementation of a fast, interpolated global planner for navigation. Different techniques can be used to find the optimal path such as A* , Dijkstra's etc.

base_local_planner acts as a trajectary planner on the local level. It implements of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane. Given a plan to follow and a costmap, the controller produces velocity commands to send to a mobile base. This package supports both holonomic and non-holonomic robots, any robot footprint that can be represented as a convex polygon or circle, and exposes its configuration as ROS parameters that can be set in a launch file.

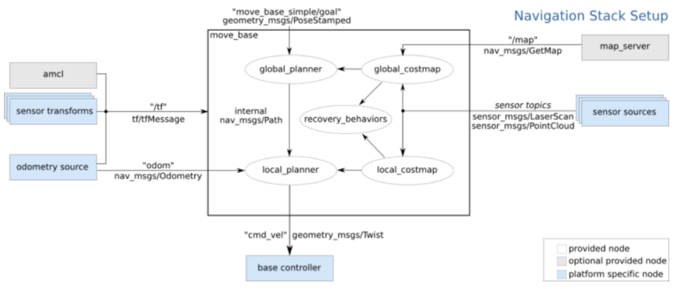


Fig. 1. navigation stack(credits: WIKI ROS)

2.2.3 Parameters

First of all, the transform_tolerance was increased to postdate the transform that is published. In order to eliminate the warning of the default parameters, the update and publish frequencies of the both **global_costmap** and **local_costmap** were tuned. The controller_frequency was too fast, so it was also tuned appropriately. afterwards, the following parameters in the *costmap_common_params.yaml* were tuned: **obstacle_range**-the range within which the obstacle added to the occupancy grid, **raytrace_range**-the distance to use to clear and update the free space while moving. **inflation_radius**-radius with which to inflate the obstacles in the global map. These parameters determined how the laserScan data is used.

AMCL filter parameters: AMCL dynamically adapts the number of particles to be used in its algorithm, however we can specify **min_particles and max_particles initial_pose**-the starting pose for the robot(x,y,yaw).

Laser parameters: laser_min_range was added for the minimum laser distance. in order not take account wheels for laser scan **laser_max_beams** is the number of maximum beam of Hokuyo sensor.

laser_z_hit the exactness of beams of LiDAR. The degree with which we trust the laser scan. **laser_z_rand** is the randomness coefficient for beams hitting.

odom_model_type is the odometry model type which is diffcorrected. **odom_alpha1..4**specifies the expected noise in optometrys rotation estimate. used for diff and omni odom model.

2.3 Sardor-Personal Model

2.3.1 Model design

The robot with a chassis, 4 wheels, and camera as well as LiDAR. The ”omni” type of optometry model was implemented for sardor robot. The minimum range of hokuyo sensor was increased in order not to treat its own wheels as obstacles.

2.3.2 Packages Used

All the packages used are the same as for the udacity_bot.

2.3.3 Parameters

For omni type of drive, odom_alpha5 was added, and a number of parameters have been tuned.

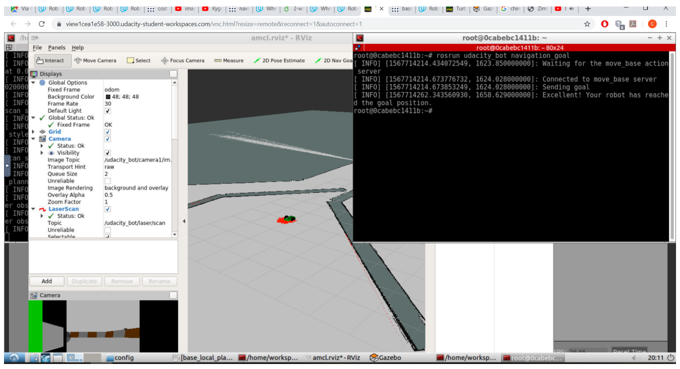
3 Results

3.1 Localization Results

The udacity_bot didn’t have any hardship to reach the goal. However, the ”sardor” took longer to achieve the goal due to some initial touch with the wall. Sardor had four wheel and omni model, the parameters of which weren’t tuned to the maximum. It took more time to take a turn for ”sardor” robot. A lot of trials and errors gave these results.

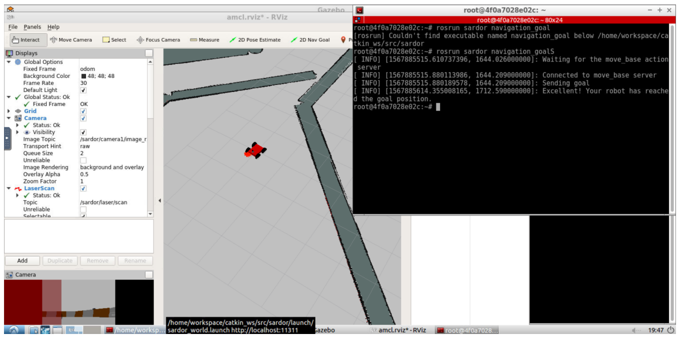
3.1.1 Benchmark

Both of robots were able to reach the specific goal during Udacity_bot at 47 seconds and sardor at 98 seconds.



```
root@0cabebc1411b:~# rosrun udacity_bot navigation_goal
[ INFO] [1567714214.434072549, 1623.850000000]: Waiting for the move_base action
 server
[ INFO] [1567714214.673776732, 1624.028000000]: Connected to move_base server
[ INFO] [1567714214.673853249, 1624.028000000]: Sending goal
[ INFO] [1567714262.343560930, 1658.629000000]: Excellent! Your robot has reache
d the goal position.
root@0cabebc1411b:~#
```

Fig. 2. Udacity_bot reached the goal



```
root@4f0a7028e02c:~# rosrun sardor navigation_goal
[rosrun] Couldn't find executable named navigation_goal below /home/workspace/ca
tkin_ws/src/sardor
root@4f0a7028e02c:~# rosrun sardor navigation_goalS
[ INFO] [1567885515.610737396, 1644.026000000]: Waiting for the move_base action
 server
[ INFO] [1567885515.880113986, 1644.209000000]: Connected to move_base server
[ INFO] [1567885515.880189578, 1644.209000000]: Sending goal
[ INFO] [1567885614.355008165, 1712.590000000]: Excellent! Your robot has reache
d the goal position.
root@4f0a7028e02c:~#
```

Fig. 3. Sardor sucessfully reached the goal

3.1.2 Student robot

3.2 Technical Comparison

The basic mechanical layout was different in the following ways: udacity_bot comprised 2 wheels and 2 caster_collisions versus 4 wheels for sardor. The differential drive was compared for omni odometry model. The position of hokuyo and camera were slightly different(-0.05 in z) for sardor. The alpha 1-4 were smaller for sardor and the controller_frequency was increased to perform faster turns.

4 Discussion

The localisation results were good for both robots as the pose array shrinked with the movement.The omni model robot performed slightly worse and hardship at the beginning because it comprised 4 wheels and slightly bigger physical footprint. The localisation is an indispensable part in the robot motion. Knowing where the robot is crucial in any robot application. Among many techniques that exist for localisation, AMCL is easily-implemented and permits to correctly localise non-linear system and motion.

The MCL/AMCL would be a good fit for any mobile robotics application that demands transportation or to go from one place to another.

The ’Kidnapped robot’ problem happens when the robot is teleport-ed to some other place. The regular AMCL is not the best option for this problem, the solution would be to increase the number of particles or to used other more advanced variations of MCL.

5 Conclusion / Future work

This section is intended to summarize your report. Your summary should include a recap of the results, did this project achieve what you attempted, how would you deploy it on hardware and how could this project be applied to commercial products? For Future Work, address areas of work that you may not have addressed in your report as possible next steps. This could be due to time constraints, lack of currently developed methods / technology, and areas of application outside of your current implementation. Again, avoid the use of the first-person.

In conclusion, both robots could self-localise and reach the specific goal. There are many improvements/experimentation to have been done before implementing on real hardware:

- Imitate skid-steering for omni-drive
- Make the base more realistic by using .mesh files like it has been done for hokuyo sensor.
- **sensor Layout**. Having 360degree LiDAR would improve the localisation results. Depending on the height/shape of the obstacles, the location can be further improved.
- **Sensor Amount.** Localise with the RGBD camera that will communicate depth as well which might potentially reduce costs.

5.1 Hardware Deployment

Both differential-drive and omni robot has been simulated in Gazebo. So, an actual industry/commercial robot can be used with this navigation stack. It contains a 360degrees LiDAR, differential wheels and a raspberry PI for computing.

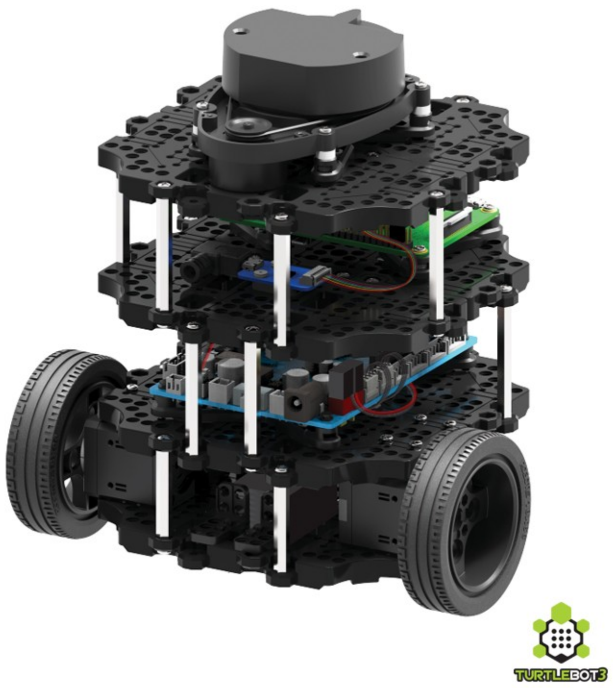


Fig. 4. turtle-bot burger

The new version of turtlebot3+ contains a robotic arm, so the basic use of localisation would be to use it for the home-service robot.

As for the computation resources, it is also possible to use cloud-based computing resources and send the data via the on-board wi-fi or Raspberry.

6 References

1) Retrieved september 7, 2019, from http://wiki.ros.org
2) Robotis, Robotis turtlebot 3, 2017.
3) Retrieved september 7, 2019, from https://link.springer.com/chapter/10.1007/978-3-030-20131-9_279