Robotics localization

Antonio Cámara

Abstract – The robotic localization problem is studied in this work. Two different robot configurations are considered to study localization performance in a simulated environment. Localization parameters are optimized to allow robots to go to a goal position. Finally robot models are compared and differences in performance analyzed.

1 INTRODUCTION

Robotics localization consists of finding the location of the robot relative to the environment and its perception. Specifically, localization consist of finding x and y coordinates and orientation in a global coordinate system. Localization is an important feature in autonomous robot systems to decide what to do based on the robot's position.
The basic problem to solve in localization is to know the position while the robot is moving beginning with a known position, which is called position tracking. A more difficult problem is global localization where the robot starts at an unknown position and it has to localize itself. The hardest localization task to solve is the kidnapped robot problem where a robot with a known position is transferred to a different position and the robot should detect that is in other place and localize itself.
Localization has been implemented with different algorithms:

- Extended Kalman Filter (EKF)
- Markov Localization
- Grid Localization
- Monte Carlo Localization (MCL)

In this work global localization (using MCL) is done with two different robots in a static environment. In first place localization is optimized using a ROS MCL library to achieve localization and conducting the a benchmark robot to a goal position. After that, the algorithm is transferred to a different robot and adjusted to work with it.

2 BACKGROUND

Localization is crucial for a robot to know its position and what to do next. The two most common localization algorithms are Kalman Filters and Monte Carlo Localization. Adaptive Monte Carlo Localization (AMCL) was chosen because it presents many advantages over Kalman Filters.

2.1 Kalman Filters

A Kalman Filter is a data processing algorithm that estimates the state of a noisy linear dynamic system. It develops a quick accurate estimate of the true value of the variable being measured. As real systems are more often non-linear systems (like moving in a curve) a variant called Extended Kalman Filters is applied. For very short intervals non-linear functions can be approximated by a linear function and this allows to apply Kalman Filters to this non-linear system. To calculate a local linear approximation, the first two terms of a Taylor series centered around the mean is used. That approximation comes with an error but produces a simple and fast calculation.

2.2 Particle Filters

Monte Carlo Localization or Particle Filters solve global localization and kidnapped robot problems in a robust and efficient manner [1]. It can accommodate arbitrary noise distributions and non-liniarities. It represents the belief by a set of samples (particles) determined according the posterior distribution over robot poses. That is to say, it represents the posteriors by a random collection of weighted particles which approximates the desired distribution.

2.3 Comparison / Contrast

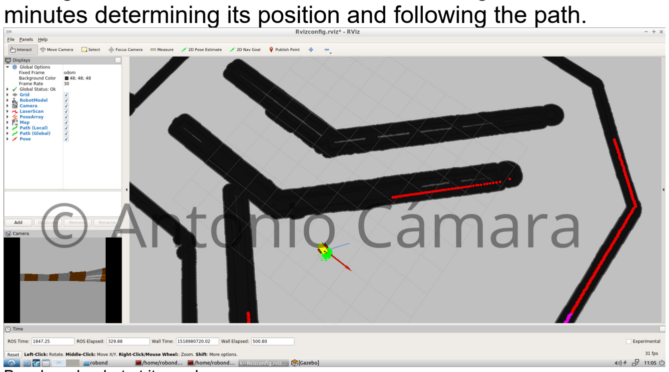
EKF is more efficient in terms of memory and time but MCL has several advantages over EKF. MCL is easier to program than EKF. EKF is restricted by a linear Gaussian states-based assumption while MCL can approximate any other practical important distribution. Moreover, MCL allows to control the memory and control resolutions by changing the number of particles distributed throughout the map. Due to those advantages MCL is used to solve localization problems in this work.

3 SIMULATIONS

In order to test MCL two simulations with two different robot models have been run. In first place a benchmark model where MCL parameters have been adjusted to obtain a good localization based navigation. Then, a second robot with different characteristics where some parameters should be adjusted.

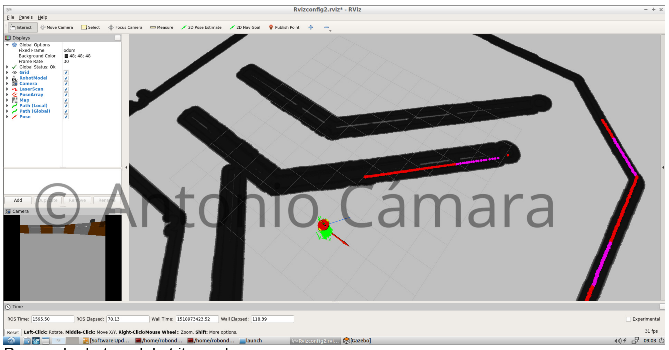
3.1 Achievements

The benchmark robot, after adjusting parameters reaches near goals soon and the more difficult test goal after several minutes determining its position and following the path.



Benchmark robot at its goal.

The personal model uses the same AMCL parameters except for the parameters and navigation parameters that describe its size and wheels. It is faster and does localization and navigation to the goal faster.



Personal robot model at its goal.

3.2 Benchmark model

3.2.1 Benchmark model

The benchmark model has the following characteristics:

Element	Shape & Size	Position	Weight
Chassis	Box 0.4 0.2 0.1	0 0.1 0	15
Back caster	Sphere 0.05	-0.15 0 -0.05	
Front caster	Sphere 0.05	0.15 0 -0.05	
Left wheel	Cylinder radius: 0.1 length: 0.05	0 0.15 0	5
Right wheel	Cylinder radius: 0.1 length: 0.05	0 -0.15 0	5
Camera	Box .05 .05 .05	0.2 0 0	0.1
Hokuyo	Box 0.1 0.1 0.1	0.15 0 0.1	0.1

3.2.2 Packages used

Benchmark robot is defined in the udacity_bot package.
This robot model uses 3 libraries for sensors and driving:
- libgazebo_ros_diff_drive.so: differential drive controller. Uses *odom* topic and publishes *cmd_vel* topic.
- libgazebo_ros_camera.so: camera plugin. Publishes image_raw topic.
- libgazebo_ros_laser.so: hokuyo laser sensor. Publishes /udacity_bot/laser/scan topic.

To start simulation with this package the following command should be used: *roslaunch udacity_bot udacity_world.launch*
After that, Rviz configuration file should be load from the package launch directory: *udacity_bot/launch/Rvizconfig.rviz*
Finally, AMCL is launched with the command: *roslaunch udacity_bot amcl.launch*

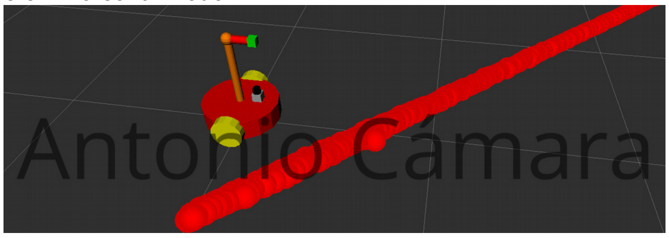
3.2.3 Parameters

AMCL parameters [2] used are:
- min_particles and max_particles: they should be enough to locate the robot properly but not many that makes the simulation and the localization slow. Lowering these parameters (from default 100 – 5000 values) in tests they are set to values 50 and 200.
- transform_tolerance: Having a maximum delay of 0.021s seen with “tf monitor” and adding a security margin, this value is set to 0.04.
- initial_pose_a: it should be 0 to avoid map rotation.
- initial_pose_x and initial_pose_y: they should be 0 to localize the particle filter pose array in the robot initial position.
- odom_alpha_x: these parameters have a default value of 0.2, but with diff-corrected odom type, the AMCL documentation recommended lower values have been used. They are: 0.005, 0.005, 0.01, 0.005, 0.003.
- laser_likelihood_max_dist and laser_max_range: use default values because no difference have been appreciated varying them.

Move_base [3] parameters used are:
- update_frequency and publish_frequency: they get low values to avoid a slow simulation. Values 5 and 2 are used.
- Other local and global parameters are left with default values.
- obstacle_range and raytrace_range: they have default values of 2.5 and 3 which are good values for our simulation.
- transform_tolerance: As commented before should be greater than the maximum delay. It is used a value of 0.1.
- robot_radius and inflation_radius: robot size and navigation margin. The minimum value to allow the robot navigate. Their values are 0.2 and 0.25.
- laser_scan_sensor: laser topic (/udacity_bot/laser/scan)
- base local parameters are left with default values except for max_vel_x which is set to 0.6 to allow the robot go a bit faster.

3.3 Personal Model

3.3.1 Personal Model



The personal model has the following characteristics:

Element	Shape & Size	Position	Weight
Chassis	Cylinder radius: 0.2 length: 0.1	0 0.1 0	12
Back caster	Sphere 0.05	-0.15 0 -0.03	
Front caster	Sphere 0.05	0.15 0 -0.03	
Left wheel	Cylinder radius: 0.08 len: 0.05	0 0.2 0	3
Right wheel	Cylinder radius: 0.08 len: 0.05	0 -0.2 0	3
Arm1	Cylinder radius: 0.02 len: 0.4	0 0 0.2	0.5
Elbow	Sphere 0.03	0 0 0.2 to Arm1	0.05
Arm2	Cylinder radius: 0.02 len: 0.1	0.075 0 0 to elbow	0.1
Camera	Box .04 .04 .04	0.05 0 0 to Arm2	0.1
Hokuyo	Box 0.1 0.1 0.1	0.1 0 0.1	0.1

3.3.2 Packages used

Personal robot is defined in the second_bot package.
This model uses three libraries for sensors and driving:
- libgazebo_ros_diff_drive.so: differential drive controller. Uses *odom* topic and publishes *cmd_vel* topic.
- libgazebo_ros_camera.so: camera plugin. Publishes image_raw topic.
- libgazebo_ros_laser.so: hokuyo laser sensor. Publishes /second_bot/laser/scan topic.

To start simulation with this package the following command should be used: *roslaunch second_bot udacity_world.launch*

After that, Rviz configuration file should be load from the package launch directory: second_*bot/launch/Rvizconfig2.rviz* Finally, AMCL is launched with the command: *roslaunch second_bot amcl.launch*

3.2.3 Parameters

AMCL parameters used are the same to the ones used in previous robot. Move_base parameters are similar changing only the laser_scan_sensor that uses the topic: /second_bot/laser/scan.

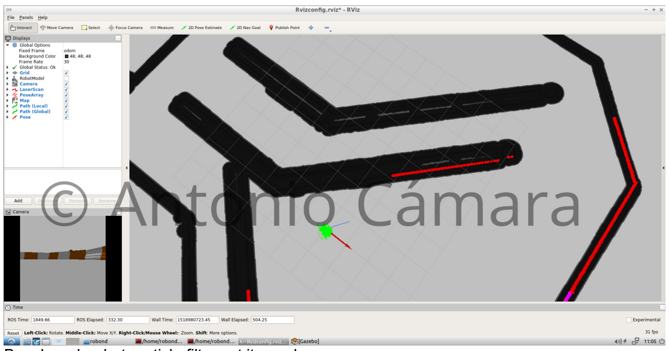
4 RESULTS

Similar AMCL and movements parameters are used in two different robots with different results. Both robots end achieving the test goal but one performs frankly better.

4.1 Localization results

4.1.1 Benchmark model

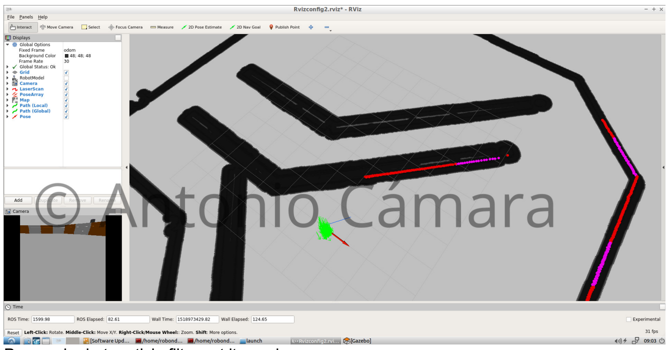
Benchmark robot goes faster to a near goal, but needs several minutes to get to a difficult one. It needs near a minute to make the particle filters to converge and more to start following the way to the goal. Its path to the goal is not totally smooth and needs 4 minutes or more to get to the test goal. It has an unexpected behavior after particle filters convergence going round several times before following the path to the goal.



Benchmark robot particle filters at its goal.

4.1.2 Personal model.

Personal model performs better than the benchmark one. It moves faster. Its particle filters converge in 1/3 of the time. In half a minute starts following the path very smoothly and it goes to the test goal in around a minute.



Personal robot particle filters at its goal.

4.2 Technical comparison

Both robots are similar in size and have the same sensors and two wheel drive. Benchmark robot is longer and personal robot is wider. Personal robot has an arm to manipulate the camera position but it is lighter. Personal robot chassis weights 3 kg less and its smaller wheels weight 2 kg less each.

5 DISCUSSION

As seen in results personal robot performs remarkably better than benchmark robot. Main difference between both is weight. Personal robot is lighter and that allow it to move faster and perform localization in 1/3 of the time. Therefore, physical properties of the robot can vary the final performance of the robot going to a goal position.
The 'Kidnapped Robot' problem can be simulated modifying parameters initial_pose_x and initial_pose_y getting an initial particle filters center different from the robot. Some tests have been done with the personal robot obtaining good results but needing a lot more time to get to the goal position.
The work done here can be used in industry or other known scenarios (museums, universities, airports, etc) to localize and move robots. With AMCL a robot can start its work localizing itself and after that tracking its position to the different places where it has to do any work. Having a known and static environment this algorithm can be used. In a variable environment a robot would need more capabilities to adapt its knowledge to it.

6 CONCLUSION / FUTURE WORK

Robotics localization problems has been studied in this work. Two different robot models have been used to test AMCL in a simulation environment with ROS, Gazebo and RViz. Localization parameters have been studied and configured to achieve a good localization. Finally, the two robots different properties have been analyzed to find that the lighter robot is faster and performs better.
To obtain more precision in localization AMCL can be configured with more particles and sensors but it will need more processing time. As a future work, new robot models with more sensors (3D camera, ultrasound sensors, etc) and different physical characteristics can be tested to obtain a more precise localization and navigation.

REFERENCES

[1] Sebastian Thurn, Dieter Fox, Wolfram Burgard and Frank Dellaert. Robust Monte Carlo Localization for Mobile Robots. http://robots.stanford.edu/papers/thrun.robust-mcl.pdf

[2] AMCL Ros package. http://wiki.ros.org/action/fullsearch/amcl

[3] Setup and Configuration of the Navigation Stack on a Robot. http://wiki.ros.org/action/fullsearch/navigation/Tutorials/RobotSetup