Adaptive Monte Carlo Project: Where am I?

Gassó Loncan, J.C.

Abstract—This paper describes the creation and testing of two model robot simulations using ROS, Gazebo and RViz software. We discuss about two methods of localisation: Extended Kalman Filter (EKF) localisation algorithm will be compared to Adaptive Monte Carlo localisation (AMCL) algorithm then AMCL performance is tested in the two different simulated robot models. The conclusions are based on efficiency of the set of parameter chosen for every model robot in reaching the goal.

Index Terms—Robot, Udacity, localisation, Adaptive Monte Carlo.

1 Introduction

One of the most difficult tasks of robotics is to determine where the robot is in the world and how to reach a particular goal or objective in a particular pose. There are several methods that could be applied to perform localisation of the robots position and orientation within a known world. This project attempts to implement Adaptive Monte Carlo Localisation for a simulated environment. The world provided by Udacity is used here and ROS AMCL package is used to for the localisation.

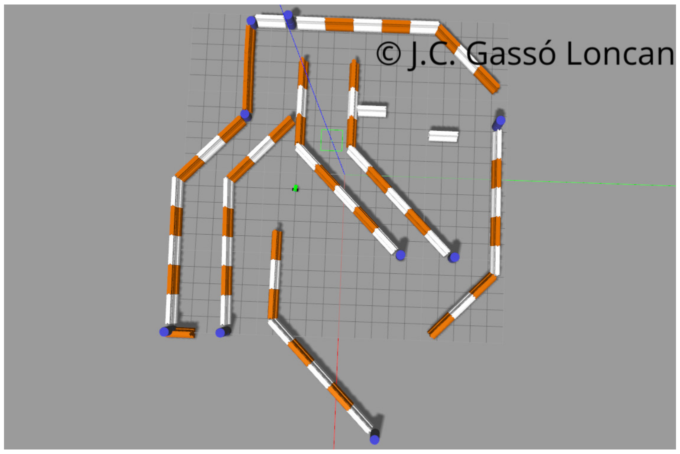

Fig. 1. Jackal Race map by clearpathrobotics

2 Background / Formulation

Localisation is the challenge of determinate where a robot is in the world. This task is important in mobile robot which used a several sensors with given uncertainties of each due to noise, such as a camera or LIDAR (Light detection and Ranging) or uncertainties due to imperfect actuators/encoder counts when moving the robot. There are three basics types of localisation problems: they are the position tracking problem, global localisation problem, and the kidnapped robot problem. In the first, the robot is initialised with its starting or home position well known and must act on additional odometry updates provided by its sensors as it moves to its goal location. In the second one, the robot has an uncertain of its initial position and requires the robot to estimate its position from scratch. Even more difficult is the kidnapped robot problem in which the robot successfully localises in its environment but then it is teleported to another location. The two most common approaches to Localisation are Kalman Filters and Monte Carlo localisation.

2.1 Kalman filters

Kalman filters are commonly used in control when it is necessary to deal with real time information. Kalman filter are effective in converging close to accurate value working with very noisy data. This value can represent the position or velocity of a robot. Kalman filter is a two-step iterative algorithm. Firstly, the sensors update the state of the robot in the ”Measurement Update” step and these information with the current state are used to predict the new state in the "State Prediction" step. Initially, an estimated guess of the state is used and iterates through the two steps to converge and predict the current state. Overall, the filter is based on the following two assumptions:

- State space can be represented by a unimodal Gaussian distribution.
- Motion and measurement models are linear.

These two assumptions provide practical limits to modern day robotic applications, due to the nature of non-linear readings and behaviours found in both the hardware and environment. Therefore, Extended Kalman Filters (EKF) were introduced to address these issues. While the EKF algorithm addresses the limitations of the Kalman Filter, the mathematics is relatively complex and the computation uses considerable CPU resources making it somewhat unsuitable to perform on a small embedded processor.

2.2 Particle filters

Monte Carlo Localisation algorithm (MCL), also known as Particle Filter, is the most popular localisation algorithm in robotics. It tackle the localisation problem discretising the environment the into individual "particles", where each one is basically one possible state of your model, and a collection of a sufficiently large number of particles allow to handle any kind of probability distribution. The particles are re-sampled each time robot moves and sense its environment. MCL is limited to Local and Global Localisation problem only.

2.3 Comparison

Kalman Filters have limiting assumptions of a unimodal Gaussian probability distribution and linear models of measurement and actuation. Extended Kalman filters relax these assumptions but at the cost of increased mathematical complexity and greater CPU resource requirements. Monte Carlo Localisation does not have the limiting assumptions of Kalman filters and is very efficient to implement. Monte Carlo Localisation is used in this project. The differences between the approaches are summarised in the following table:

Feature	MCL	EKF
Measurement	Raw Measurement	Landmarks
Measurement Noise	Any	Gaussian
Posterior	Particles	Gaussian
Efficiency(memory)	lower	higher
Efficiency(time)	lower	higher
Implementation	easier	more difficult
Resolution	lower	higher
Robustness	higher	lower
Memory and Resolution Control	yes	no
Global Localisation	yes	no
State Space	Discrete	Continuous

TABLE 1
MCL vs EKF

3 Simulation

The simulation was performed in the Gazebo Physics simulator environment. In order to study the implementation of the AMCL algorithm in ROS packages, a robotic model and a map were provided by Udacity ND staff. Also, a custom model was developed to test the simulation. Both models are described in .XACRO and .GAZEBO files. The simulation is aimed at reaching a predefined target by performing simultaneous localisation and motion until the robot reaches the final goal pose.

3.1 Udacity bot

The project has a base model provided for bench-marking and parameter setting for evaluation. Udacity bot is a simple model with two centred wheels and a box shape chassis. A camera and a LIDAR hokuyo sensor that publishes information to the topic udacity bot/laser/scan are embedded in the robot configuration.

3.2 UNER bot

The UNER bot is the custom robot developed to accomplish the requirements of this project. As the original model, UNER robot has two wheels but they are arranged in the front of the chassis. In addition there are a cylindrical shape over the main chassis where the hokuyo sensor take place. Colours and dimensions were also changed.

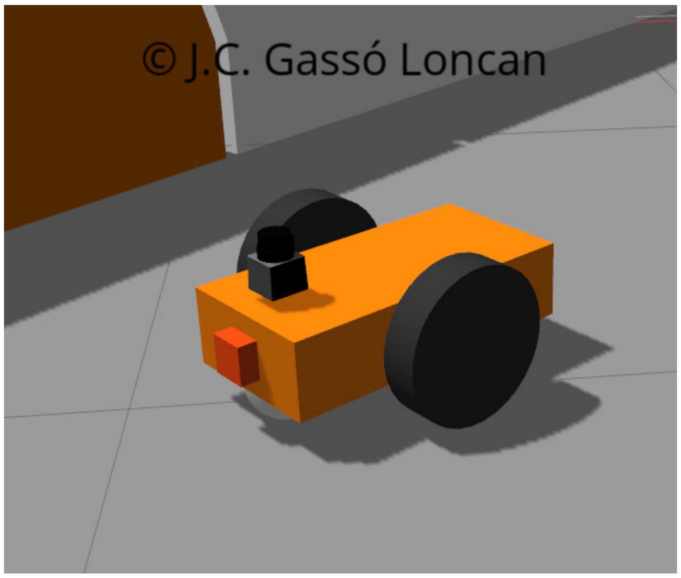

Fig. 2. Udacity bot

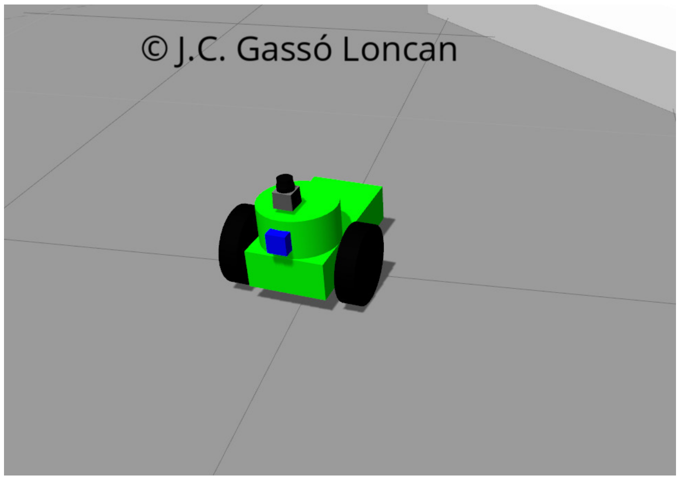

Fig. 3. UNER bot

3.3 ROS Packages

Three ROS packages were used: AMCL, move base and map server.

- AMCL is a probabilistic localisation system for a robot moving in 2D. It implements the adaptive Monte Carlo localisation approach.
- 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 maintains two costmaps, one for the global planner, and one for a local planner that are used to accomplish navigation tasks.
- map server is used for provide the global map of the jackal race world which has been provided.

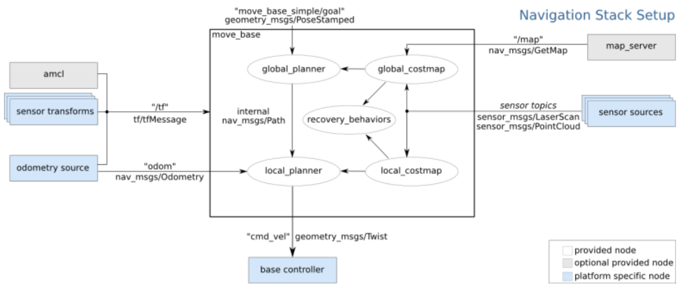

Fig. 4. ROS Navigation stack

To successfully solve the localisation problem and reach the goal position, the parameters for each of the used ROS packages were tuned. The parameters that were tuned in the amcl node, global costmap node, local costmap node, common costmap node, and local planner node was detailed later. [1]

The costmap nodes keep information about obstacles in the world. The local costmap node is used for local planning and obstacle avoidance while the global costmap node is used to create long-term plans over the entire environment [Cita]. The common costmap node contains parameters shared by both the local and global costmaps. The base local planner node is responsible for computing velocity and acceleration commands to send to the mobile base of the robot and gives weight and time to forming the forward trajectory of the robot's path. [2]

3.3.1 Parameters setting

For both, Udacity and UNER bot, has been a different setting of the most relevant parameters. [3]

3.3.1.1 AMCL: AMCL parameters have an direct impact in the computational load. One of the most important parameter is the amount of particles. The minimum and maximum particles were set to 10 and 500 respectively and were chosen to minimise computational burden on the development environment (the maximum has a default value of 1000). Increasing the max particles affect the amount of operations performed each cycle and results in slower overall performance. Similarly, the transform tolerance was increased up to 0.3 seconds to reduce computational cost. Also, some characteristics of the laser model were changed. [4] The table below resume the values of the file AMCL.launch:

Parameter	Value
Overall filter	
transform_tolerance	0.3
min_particles	10
max_particles	300
update_min_a	Pi/12
update_min_d	0.05
initial_pose_x/y/a	0
Laser	
laser_model_type	likelihood_field
laser_min_range	-1.0
laser_max_range	30.0
laser_max_beams	60
laser_z_hit	0.990
laser_z_rand	0.01

TABLE 2
AMCL parameter settings

3.3.1.2 Move base: Move base configuration includes the common parameters and local/global costmap parameters. Both robots had the same parameters values.

In the common costmap params file the most important change was the inclusion of a custom footprint for the robot. As both model has the same chassis, this is similar. Inflatious radius was increased to 0.6 in order to avoid possibles collisions with obstacles. The obstacle range was set to 2.5 meters and denotes that only objects detected within one meter are obstacles that need to be avoided. Increasing this parameter allows the robot to detect obstacles from a larger distance but may result in poor navigation. The raytrace range was set to 4.0 meters and signifies that the robot will attempt to clear out space in front of it up to three meters away when given a sensor reading.

In the local costmap params file, the publish and update frequency values was decreased to avoid errors due to missed cycles. In addition, the map size also is smaller than default configurations. Similar set of parameters is used for the global costmap.

Finally, base local planner params file contains a large number of ROS Parameters that can be set to customise the behaviour of the Trajectory Planner ROS. These parameters are grouped into several categories: robot configuration, goal tolerance, forward simulation, trajectory scoring, oscillation prevention, and global plan. Several changes in the velocity parameters (e.g. max vel x), simulations time(sim time, controller frequency) and accuracy to reach the goals was performed.

4 RESULTS

4.1 Udacity bot

Udacity bot had an acceptable result during the simulations. It was necessary 12.5 minutes to reach the goal. The path was not optimal but the robot always could reach the goal.

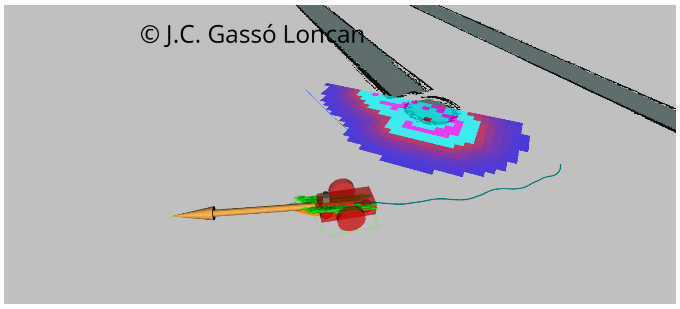

Fig. 5. Udacity bot driving to the goal

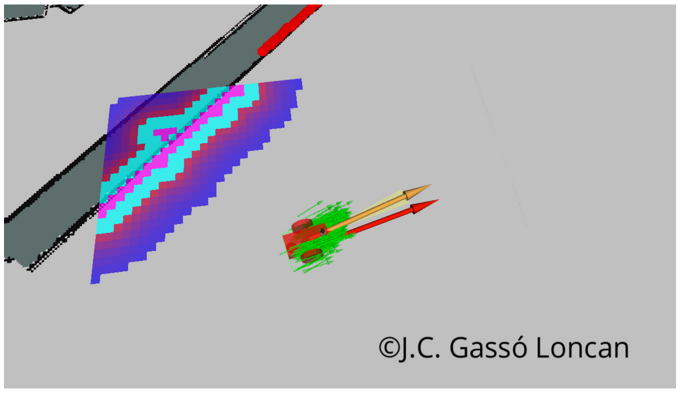

Fig. 6. Udacity bot after reach the goal

4.2 UNER bot

The custom model made for this project had a better performance than the Udacity bot. It took less than a 2 minutes to reach the goal. From start to finish, UNER bot follows a path closely trailing alongside the estimated global path to the goal and maintains a particle cloud with an average approximation close to its trajectory.

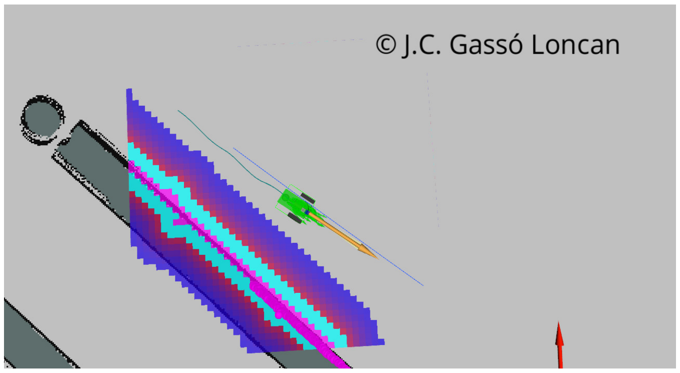

Fig. 7. UNER bot driving to the goal

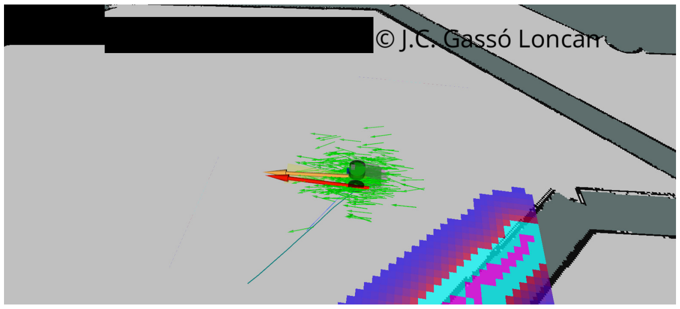

Fig. 8. UNER bot after reach the goal

5 Discussion

While design and simulate this model robots en Gazebo environment is a task relatively easy, real robot is a more complicated issue. Every design implies a new set of parameters which must be adjusted using a trial and error method. It could take many iteration until reach the best and optimal solution. Moreover, the set of parameters uses during the simulation are directly related to the hardware and the power calculation. The result could vary between computers.

6 Conclusion / Future work

Although the Udacity bot reaches the goal, the time that it takes to get there is extremely long compared to UNER bot. The reason may be related to the different robot configuration. For example, the wheels disposal is a key issue to improve the velocity to reach the goal. Including more sensors could also improve the performance of the AMCL and move base packages.

References

[1] K. Zheng, *Ros navigation tuning guide*. 2016.
[2] . ROS wiki: Open Source Robotics Foundation, "Roswiki: costmap 2d." "http://wiki.ros.org/costmap_2d", 2009 (accessed May 25, 2018.
[3] . ROS wiki: Open Source Robotics Foundation, "Ros navigation tuning guide." http://wiki.ros.org/navigation/Tutorials/NavigationTuningGuide. accessed May 25, 2018.
[4] . ROS wiki: Open Source Robotics Foundation, "amcl. package summary." http://wiki.ros.org/amcl. accessed May 25, 2018.