Udacity Robotic Localisation

Patrick Mockridge

Abstract—Two robot designs were successfully navigated through a maze to a desired location in a simulated Gazebo environment. The ROS Navigation and Adaptive Monte Carlo Localisation packages were integrated in order to achieve this aim. The first robot design was provided by Udacity, the second was loosely modelled on R2D2 from the Star Wars franchise. It was found that raising the robot's centre of gravity made it more difficult to tune the base planning parameters necessary to give the bot enough acceleration and momentum to follow path. Therefore ballast was added to the bot to lower the centre of gravity in addition to 6 total casters, compared to just two for the Udacity Bot, in order to improve balance. This allowed the New Robot to achieve the acceleration and speed parameters necessary to reach the goal.

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

1 Introduction

Robotic localisation is the problem of determining a robot's pose in a mapped environment. Noisy sensor data is filtered using probabilistic models before being used to determine the robot's position, direction and orientation.

The challenge of robot localisation can be grouped into three categories of increasing complexity:

1) Local Localisation: The robot knows its initial location and pose. The localisation problem is thus defined relative to this initial information.
2) Global Localisation: The robot is placed into an environment with zero knowledge about its initial location.
3) The Kidnapped Robot: The robot is not only uncertain of its initial location, but it also may be kidnapped and placed anywhere on the map at any time. So the robot is also uncertain about the continuity of its own information.

This project ROS packages were utilised to solve a local localisation navigation problem involving a pre-built URDF defined robot provided by Udacity, and a second novel robot, created to compare and contrast with the performance of the original.

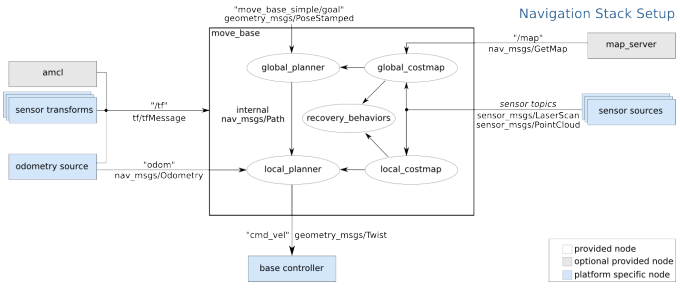

Fig. 1. ROS Navigation Stack

Custom ROS packages were created to launch the robot and the navigation stack in RVIz and Gazebo simulation environments. Costmap, speed and rotation parameters were tuned within each package to achieve the project goal of successfully creating a robot that can autonomously navigate the provided 'Jackal Race' map from the starting point to the finish.

2 Background

The four main techniques for robot navigation are thus:

1) Kalman Filters: the most common Gaussian filter for nonlinear navigation models.
2) Markov Localisation: maintains a probability distribution of all possible locations and orientations of the robot.
3) Grid Localisation: estimates the robot's pose by separating the probability space into a grid and therefore a histogram.
4) Monte Carlo Localisation: also known as a 'Particle Filter' creates a particle swarm representing possible position and orientation of the robot at each time step.

2.1 Kalman Filters

Kalman Filters excel at filtering noisy sensor data. An Extended Kalman Filter (EKF, a non-linear more general iteration of the Kalman filter can also fuse data from a camera, laser sensor and the motor differential drive using the ROS Extended Kalman Filter package.

2.2 Adaptive Monte Carlo Localisation/Particle Filters

Adaptive Monte Carlo Localisation (AMCL) maps robot location and orientation as a swarm of particles with mimetic properties, representing possible future positions of the robot. Particles with better prediction at each time step are allowed to procreate, poor predictors are deleted at each time step.

2.3 Comparison / Contrast

AMCL has many benefits relative to EKF. Firstly AMCL is easier to implement in practice, it is easier to conceptualise and visualise the functionality in the ROS RVIz/Gazebo

environment. AMCL can represent any possible probability distribution and is not limited to a Gaussian frame of reference, as is the case for Kalman Filters. Therefore AMCL has the potential to be more adaptive and more computationally robust to a wider range of situations.

For these reasons, AMCL was carried forward and used with the ROS Navigation Stack to navigate the robot through the 'Jackal Race' environment provided by Udacity.

3 SIMULATIONS

Simulations were carried out using the Gazebo 3D simulation environment supported by the RViz visualisation toolkit.

The ROS Vigation stack was used, a visualisation of which can be found in Figure X.

3.1 Achievements

3.2 Benchmark Model: UdacityBot

3.2.1 Model design

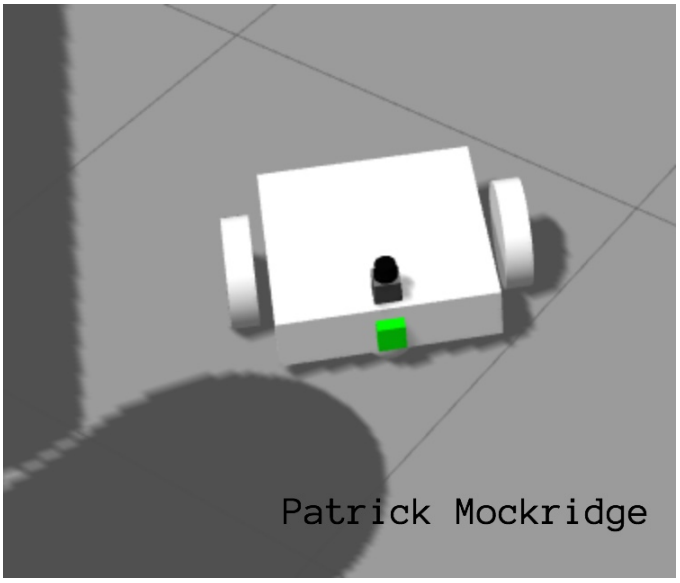

Fig. 2. Udacity Bot Close Up

- Maps: The Clearpath Jackal Race map was used
- Meshes: The simulated Hoyoku Scanner was rendered using the hoyoku.dae mesh.
- Launch: Three launch scripts launch the simulation:
    1) robot_description.launch: defines the joint_state_publisher, the robot_state_publisher and the robot_description which all together connect the URDF model to the TF and joint frames.
    2) amcl.launch: launches the map server, the odometry frame, the AMCL localisation server, the move_base server and the trajectory planning server.
    3) udacity_world.launch: launches the jackal_race world in Gazebo, spawns the robot in Gazebo and launches RVIz.
- : Worlds: two worlds are defined
    1) jackal_race: defines the maze
    2) udacity_world: defines the ground plane, the lighting and the world camera
- : URDF
    1) udacity_bot.gazebo: defines the differential drive controller, the camera and camera controller, the laser scanner and laser scanner controller for rendering in Gazebo
    2) udacity_bot.xacro: defines the robot in XML macro format.

TABLE 1
Benchmark Model Visual and Collision Parameters

Footprint	Joint	xyz: '0 0 0' rpy:'0 0 0'
Chassis	Pose	0 0 0.1 0 0 0
Chassis	Collision/Visual	Box: 0.4 0.2 0.1
Back Castor	Collision/Visual	xyz:-.15 0 -.05 radius='0.05'
Front Castor	Collision/Visual	xyz: .15 0 -.05 radius: .05
Left Wheel	Collision/Visual	radius= .1 length= .05
Left Wheel Hinge	Continuous	xyz: 0 0.15 0 axis xyz: 0 1 0
Right Wheel	Collision/Visual	radius: 0.1 length: .05
Right Wheel Hinge	Continuous	xyz: 0 -.15 0 axis xyz: 0 1 0
Camera	Collision/Visual	Box: .05 .05 .05
Camera	Joint	origin xyz: .2 0 0
Scanner	Collision/Visual	Box: .1 .1 .1
Scanner	Joint	origin xyz: .15 0 .1

3.2.2 Packages Used

The navigation_goal.cpp service provided by Udacity was used to instruct the robot to begin its journey and defined the successful parameters of completion.

3.2.3 Parameters

- Maximum Particles: 400 This was chosen because of the limitations of the development environment and was the maximum number of particles that could maintain an AMCL update frequency of 10Hz.
- Transform Tolerance: 0.2 Maximum latency in seconds between transforms. Default is 0.1, but was doubled to be more lenient on the available computational resources.
- Obstacle Range: 0.8 This parameter was just tweaked in order to constrain the robot to mostly straight paths towards the goal and reduce the time the robot spent in rotation. 0.8m was found to be successful by trial and error.
- Raytrace Range: 0.7 This parameter was just tweaked in order to constrain the robot to mostly straight paths towards the goal and reduce the time the robot spent in rotation. 0.7m was found to be successful by trial and error.
- Inflation Radius: 0.9 The inflation radius is the turning radius at which the cost scaling factor is applied. 0.9m was found to work by trial and error.
- Maximum Forward Velocity: 0.3 Was found to be adequate by trial and error.

- Minimum Forward Velocity: 0.1 Was found to be adequate by trial and error.
- Maximum Rotational Velocity: 1 Was found to be adequate by trial and error.
- Minimum Rotational Velocity: 0.4 Was found to be adequate by trial and error.
- Maximum Forward Acceleration: 1.5 Was found to be adequate by trial and error.
- Maximum Rotational Acceleration: 2 Was found to be adequate by trial and error.
- Yaw Goal Tolerance: 0.05 This was found to be adequate. Tightening burdened computational resources, relaxing caused unacceptable reduction in performance.
- XY Goal Tolerance: 0.1 This was found to be adequate. Tightening burdened computational resources, relaxing caused unacceptable reduction in performance.
- Update Frequency: 10 This was found to be the roughly maximum rate that a Lenovo IdeaPad 510s could update given all the other parameters and still successfully achieve the goal.
- Publish Frequency: 5 This was found to be the roughly maximum rate that a Lenovo IdeaPad 510s could publish given all the other parameters and still successfully achieve the goal.
- Odometry Model Type: diff-corrected

3.3 Personal Model: RiniD2

3.3.1 Model design

The same launch scripts were used as for the original robot.

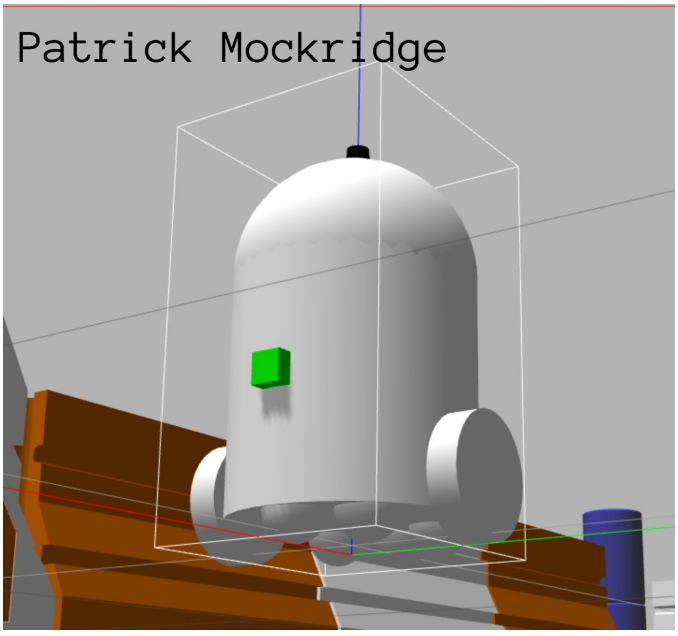

Fig. 3. Rini2D2 Close Up

3.3.2 Packages Used

The navigation_goal service provided by Udacity was used to instruct the robot to begin its journey and defined the successful parameters of completion.

TABLE 2
Personal Model Visual and Collision Parameters

Footprint	Joint	xyz:0 0 0 rpy:0 0 0
Chassis	Pose	0 0 0.1 0 0 0
Chassis	Collision/Visual	radius: 0.2 length: 0.3
Head	Collision/Visual	radius:0.2
All Castors (6)	Collision/Visual	xyz: -.15 0 -.05 radius: .05
Left Wheel	Collision/Visual	radius: .1 length:.05
Left Wheel Hinge	Continuous	xyz:0 .15 0 axis:0 1 0
Right Wheel	Collision/Visual	radius:0.1 length:0.05
Right Wheel Hinge	Continuous	xyz:0 -0.15 0 axis:0 1 0
Camera	Collision/Visual	Box: .05 .05 .05
Camera	Joint	origin xyz: .2 0 0
Scanner	Collision/Visual	Box: .1 .1 .1
Scanner	Joint	origin xyz:0.15 0 0.1

3.3.3 Parameters

- Maximum Particles: 400 This was chosen because of the limitations of the development environment and was the maximum number of particles that could maintain an AMCL update frequency of 10Hz.
- Transform Tolerance: 0.2 Maximum latency in seconds between transforms. Default is 0.1, but was doubled to be more lenient on the available computational resources.
- Obstacle Range: 1.2 This parameter was just tweaked in order to constrain the robot to mostly straight paths towards the goal and reduce the time the robot spent in rotation. 1.2m was found to be successful by trial and error.
- Raytrace Range: 1 This parameter was just tweaked in order to constrain the robot to mostly straight paths towards the goal and reduce the time the robot spent in rotation. 1m was found to be successful by trial and error.
- Inflation Radius: 2 The inflation radius is the turning radius at which the cost scaling factor is applied. 2m was found to work by trial and error.
- Maximum Forward Velocity: 0.7 After fitting 6 castors to RiniD2, it was found that a higher maximum speed could be set. The higher maximum speed encouraged the robot to cover greater distances in straight lines towards the goal.
- Minimum Forward Velocity: 0.3 Same as above, encouraging RiniD2 to make headway towards the goal when moving forward.
- Maximum Rotational Velocity: 1 Any higher than 1rad/s was found to encourage the robot to rotate in circles, before the costmap and AMCL could update properly, thus causing RiniD2 to become disorientated.
- Minimum Rotational Velocity: 0.4
- Maximum Forward Acceleration: 2 With the balance of RiniD2 optimised with ballast, a high maximum forward acceleration was possible, encouraging greater momentum towards the end goal.
- Maximum Rotational Acceleration: 1.5 Any higher than 1rad/s was found to encourage the robot to rotate in circles, before the costmap and AMCL could

update properly, thus causing RiniD2 to become disorientated.

- Yaw Goal Tolerance: 0.05 This was found to be adequate. Tightening burdened computational resources, relaxing caused unacceptable reduction in performance.
- XY Goal Tolerance: 0.1 This was found to be adequate. Tightening burdened computational resources, relaxing caused unacceptable reduction in performance.
- Update Frequency: 10 This was found to be the roughly maximum rate that a Lenovo IdeaPad 510s could update given all the other parameters and still successfully achieve the goal.
- Publish Frequency: 5 This was found to be the roughly maximum rate that a Lenovo IdeaPad 510s could publish given all the other parameters and still successfully achieve the goal.
- Odometry Model Type: diff-corrected

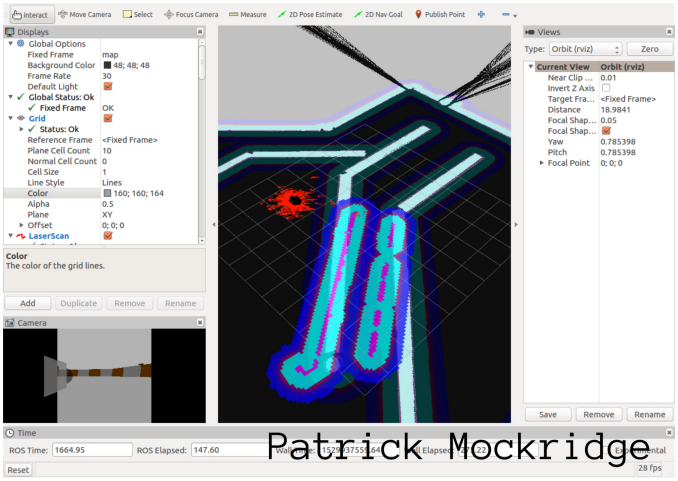

Fig. 5. Udacity Bot reaches the goal in RViz

4 Results

Each robot was tuned until it was able to reach the goal. RiniD2 was found to initially have a very high centre of gravity and be unstable only operating with two castors. Four additional castors were added, but still to robot was found to be unstable in acceleration and deceleration phases.

Finally a dense ballast module was added to the centre of the robot at the foot of the base. This improved performance over the Udacity Bot and allowed for RiniD2 to attain much higher speed and acceleration, and thereofore momentum towards the goal.

It was found that momentum was the key factor of performance for this simple navigation task. If the robot could make a 'beeline' for the corner turn, the rest of the navigation was not a problem. RiniD2's performance was therefore found to be twice as fast as Udacity Bot, although Udacity Bot was only optimised in so much as to achieve the goal and further optimisation is possible beyond the scope of this project.

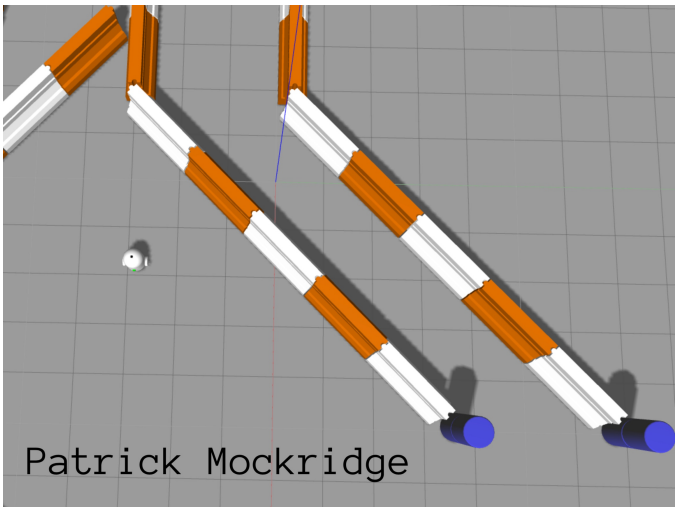

Fig. 6. Rini2D2 reaches the goal in Gazebo

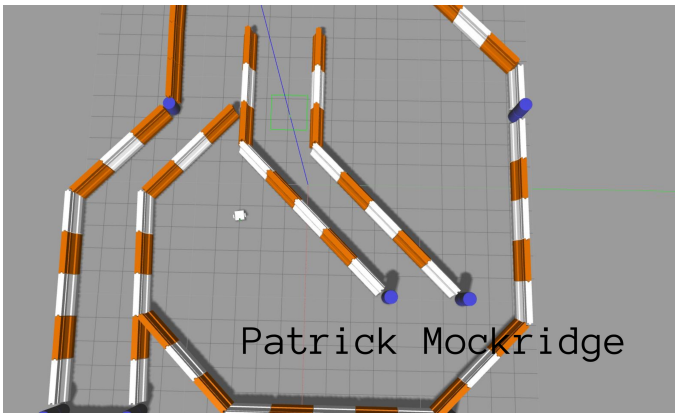

Fig. 4. Udacity Bot reaches the goal in Gazebo

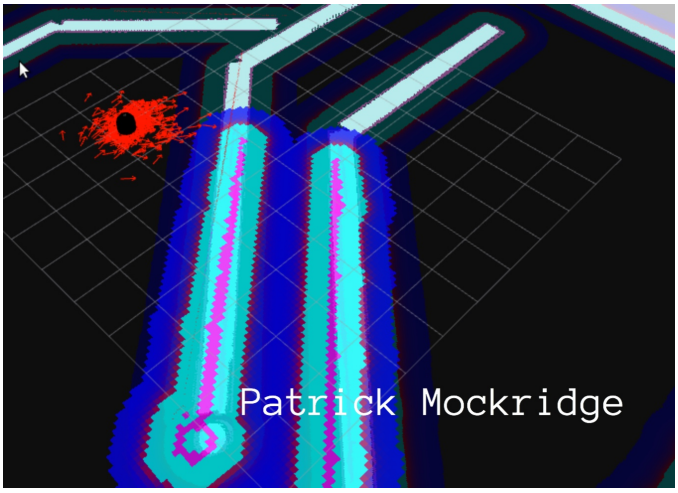

Fig. 7. Rini2D2 reaches the goal in RViz

4.1 Localization Results

Both robots achieved the navigation goal.

4.2 Technical Comparison

RiniD2 was faster and less circuitous than Udacity Bot, with higher acceleration and velocity parameters. udacity Bot was only tuned enough to reach the goal reliably according to the rubric. Further optimisation is possible.

5 Discussion

AMCL has proven to be a reasonably good means of navigating a robot around an undetermined environment. However key parameters for the success of navigation often had little to do with amcl parameters themselves and had more to do with the general momentum of the robot towards the goal as well as its balance.

Localisation could be useful for delivery or home service robots. AMCL could readily be used in search and rescue situations, disaster recovery, nuclear fallout et cetera

AMCL is not well suited to solving a kidnapped robot problem on its own. Such a problem would require additional machine learning and mapping functionality. If a robot can calibrate its surroundings to a prexisting map and extract features from, perhaps, an Support Vector Machine or Convolutional Neural Network then solving the kidnapped robot problem would be more possible. In a warehouse closed environment robot could localise themselves relative to fixed WiFi points, in a global environment, for self driving vehicles for example, accurate satellite GPS would probably be required.

6 Conclusion / Future work

RiniD2 outperformed the Udacity Bot but there is scope for further optimisation of Udacity Bot, in terms of inflation radius, velocity and acceleration parameters.

6.1 Modifications for Improvement

RiniD2 could be further improved with 360 degree laser scanning and camera, or perhaps integrated 360 LiDar. Also a greater number of points for the AMCL with quicker updates would allow both robots to move even more quickly. This was contrained by the hardware used (Lenovo IdeaPad 510s).

6.2 Hardware Deployment

This project could be deployed with a Jetson TX1, camera, Hoyoku laser scanner, mobile power source and engine with appropriate hardware drivers installed.

The TX1 is contains a bespoke machine learning module and therefore more AMCL particles and a faster update frequency could probably be accomodated.

In a real environment the inputs and outputs of the packages would be altered from the simulated Gazebo environment to those of the ports on the development kit.