Capabilities and Challenges of SLAM and Map Localisation for Autonomous Vehicles

A Research Progress Report
Presented to
Queensland University of Technology
As Part of the Degree Bachelor of Mechatronic Engineering

Presented by
Grant Dare

Supervisors
Professor Michael Milford

James Mount

23 June 2019

Table of Contents

Table of Figures

1 Introduction

The term *autonomous vehicle* is a broad term used a lot recently. The term can include civilian cars, supply transport (trucks), and logistical vehicles. An important component of this idea is the ability for the vehicle to understand its location and able to remember where it has been. A study by the BMW group found that the public are not supportive of the automation of vehicles [1]. The study found that users were equally supportive as they were unsupportive of autonomous vehicles and predominately uncertain where the option was available.

Simultaneous Localisation and Mapping (SLAM) is the ability for a robot to understand where it has been. Using an odometry model of a robot's plethora of inertial sensors such as; tachometers, Inertial Measurement Unit (IMU), and Global Positioning Systems (GPS), a robot can understand its motion path. Leveraging on this, and most often using a Laser Range Finder (LIDAR), an occupancy grid can be generated to describe the surroundings of the robot.

SLAM and map localisation are pertinent to a robot's ability to navigate optimally. By being able to map and localise inside a map, the robot is given the ability to plan optimal navigational paths and avoid obstacles.

1.1 Purpose

This research is aimed at investigating the challenges and capabilities of autonomous vehicles to perform SLAM and their ability to localise inside a known environment. With the ability to map an environment and localise inside a known environment, navigation and obstacle avoidance becomes possible.

1.2 Scope

This project investigates the capabilities and challenges for autonomous vehicles to perform EKF SLAM and localise inside a known environment. This project will investigate the ability of a Remote Controlled (RC) car to perform SLAM. This project will look to localise a motion path using odometry and inertial sensors. Using a 2D LIDAR, this project will develop an occupancy grid to describe the robot's environment as a map. Given the mapping process, the project will localise the robot inside of the map using the developed motion path and occupancy grid.

Analysis will be performed based on visual analysis of components pertinent to the stage being analysed. To visually analyse the results, the ROS Visualisation (RVIZ) tool will be used in the appropriate reference frame. For localisation this will mean comparing the overall odometry path to the expected shape of motion. For mapping this will mean comparing the OGM to the floor plan of the environment. Finally, for AMCL this will mean assessing its ability to estimate the pose in the map by comparing the laser scans to the map.

2 NightRider Project

The team at the Australian Centre for Robotic Vision (ACRV) have already developed a small research robot to leverage for easy integration. This robot project is called *NightRider* and consists of an RC car designed for the purposes of research for real world application. This investigation will be carried out on the NightRider RC car to leverage existing hardware. The NightRider RC car conforms to the Ackermann kinematic steering model which is the same as a commercial vehicle.

Figure 2.1 - NightRider RC car

The car uses a *2-layer* approach to hardware, illustrated in Figure 2.2, where the bottom layer contains the low level control such as; motor control, power distribution, and the interface to the mid-level hardware. The interface between the mid-level and low-level is a Teensy used to communicate electrical control signals to the drive motors and steering servo from the Jetson TX2[1]. The Jetson TX2 makes up the bulk of the mid-level hardware as the primary processing module.

Figure 2.2 – Mid-level (Left) and 2-layer (Right) NightRider Hardware

The Jetson TX2 runs the Linux Operating System (OS) Ubuntu 16.04 LTS. This is as NightRider employs the Robot Operating System (ROS) Kinetic Kame[2] environment for processing and running the necessary algorithms. The Jetson TX2 includes an IMU, which makes up part of the odometry/inertial sensors on NightRider. The low-level includes a tachometer on the wheels to determine linear speed and uses the pulse time to the servo to measure the steering angle.

The mid-level contains a network switch and a USB hub used to interface with the top-level sensors, the LIDAR and camera. The project will not be using the camera; however, it will be using the A3 RPLIDAR[3] for map generation.

[1] https://developer.nvidia.com/embedded/jetson-tx2
[2] http://wiki.ros.org/kinetic
[3] https://www.slamtec.com/en/Lidar/A3

2.1 Known Issues

The steering servo used does not use an encoder to achieve absolute positioning. Instead, the steering angle used for the odometry model is calculated using the pulse time sent to the servo rather than the actual movement and assumes ideal is achieved. The IMU is not mounted over the rear axle and so accelerometer readings cannot be used. In the odometry model, the only measurement of linear translation for the EKF fusion is the wheel odometry.

3 Localisation of Odometry Model

The localisation of the odometry model used two sensors on NightRider; the tachometer (in conjunction with the servo pulses) and the IMU. The localisation stage creates an odometry model to describe the motion path of the robot given sensor readings.

3.1 Design

The method of SLAM to be used is the Extended Kalman Filter (EKF) SLAM, which is defined by the localisation method used. The EKF approach falls under a theory of control called Kalman Filters (KF). A KF is primarily used for linear systems which are not well represented in the real world. The EKF differs from a KF in that it accounts for nonlinear systems. ROS is an open source initiative which was greatly beneficial for the basis of this project. The localisation algorithm used was a ROS package developed by ClearPath Robotics, *robot_localization*[4]. This package includes a library for an EKF implementation of localisation. The design component behind this stage was tuning covariances to achieve a more accurate odometry model.

The localisation package uses 15 degrees of freedom:

$$\begin{matrix} x & y & z \\ roll & pitch & yaw \\ \dot{x} & \dot{y} & \dot{z} \\ \dot{roll} & \dot{pitch} & \dot{yaw} \\ \ddot{x} & \ddot{y} & \ddot{z} \end{matrix}$$

Due to this being a ground based robot, the states $z,\ roll,\ pitch, \dot{y}, \dot{z}, \dot{roll}, \dot{pitch}, \ddot{y}, and\ \ddot{z}$ are not applicable and are never defined. The tachometer and servo sensors can provide an odometry measurement, after applying Ackermann's kinematics to the sensor readings, produces an $\dot{x}$ and $\dot{yaw}$ state measurement. The IMU provides a $\dot{yaw}$ measurement using the gyroscope, however, cannot provide an $\ddot{x}$ source due to the aforementioned issues in the IMU.

There can be no planning for the covariance measurements since the tuning comes down to the sensors and how they are calibrated and accurate. Given that the package was developed by ClearPath Robotics, one of their engineers leveraged this on their husky platform[5] to achieve outdoor waypoint navigation. In the interest of open source, the developer, Nick Charron, produced a tutorial[6] to his work with the package. This tutorial and source code provide a base set of covariance values to begin tuning from.

The structure for ROS reference frames is presented in Figure 3.1, where the *base_link* is the frame of the robots' motion. The localisation package collects inertial and odometry data in the *base_link* frame and provides the transform for *base_link* to *odom.* This is a necessary step to implementing the full

[4] http://wiki.ros.org/robot_localization
[5] https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/
[6] https://www.clearpathrobotics.com/assets/guides/husky/HuskyGPSWaypointNav.html

solution, as the goal is to achieve *map* localisation which as seen in Figure 3.1 requires an *odom* localisation to conform to proper ROS standards.

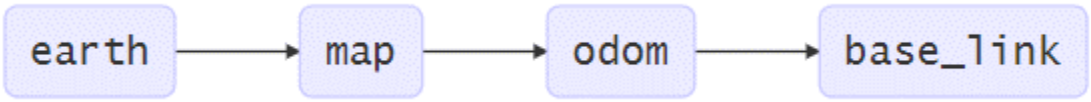


Figure 3.1 - ROS REP105 Reference Frame Order

3.2 Implementation

The NightRider project already implements nodes responsible for publishing an odometry reading from the tachometer and servo as a *geometry_msgs/Odometry,* and the IMU readings as type *sensor_msgs/IMU.* Therefore, the integration of the sensors into the EKF package is seamless in that no re-publisher is required. Instead the EKF node is implemented through a config file seen in Appendix 2: EKF Config File.

The config accepts various measurement inputs such as odometry, IMU, and pose. The node also requires the following variables important to the operation of the node. The variable *two_d_mode* is set to true as it zeros all 3D components given that this research is only looking to map and localise in a two-dimensional plane. The variable publish_*tf* ensures that the node publishes the transform between the *base_link* and *odom* which as described previously in 3.1 is important for the following stages.

The states $\dot{x}$ and $\dot{yaw}$ were appropriately used as the measurement states for the odometry reading from the tachometer. The IMU however did not report $\ddot{x}$ data as was expected, therefore only $\dot{yaw}$ measurements were used in the EKF. It is unclear why the IMU did not publish $\ddot{x}$ data as it had the capability to. It is possible that this was disabled in the base NightRider project for reasons undocumented.

The odometry source did not require a covariance state specified, however the IMU source does. The IMU covariance began with base values as per the tutorial by ClearPath Robotics which made marginal improvements over the default values. Given this, the numbers were tuned in the same *direction* as the change from the default to the tutorial. There is no *one-size-fits-all* approach to tuning the covariance since the values depends directly on the specific sensor being used and interference in the system. All tuning was carried out on *bag files* so that behaviour was constant across every test.

3.3 Results

The EKF localisation stage of this project returned an accurate representation of the motion path of the robot. The covariance of the output was still rather large as per Figure 3.2 however, was still an improvement over the default model. It was later found that the covariance in the odometry model did not affect the performance of the other stages, only the actual estimation.

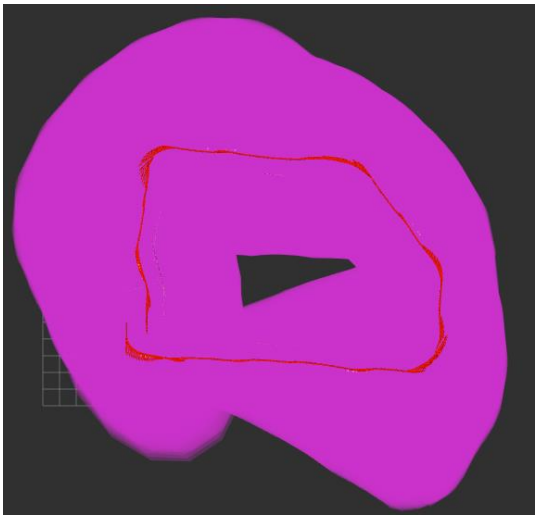


Figure 3.2 - Tuned Covariance

As per Figure 3.3 it is apparent why the odometry model requires a combination of both sensors where the dead reckoning (tachometer odometry) did not appropriately read its angular velocity. Given that the servo angle is based on pulse time rather than actual response, the steering angle is confirmed in these results to be inadequate due to this case.

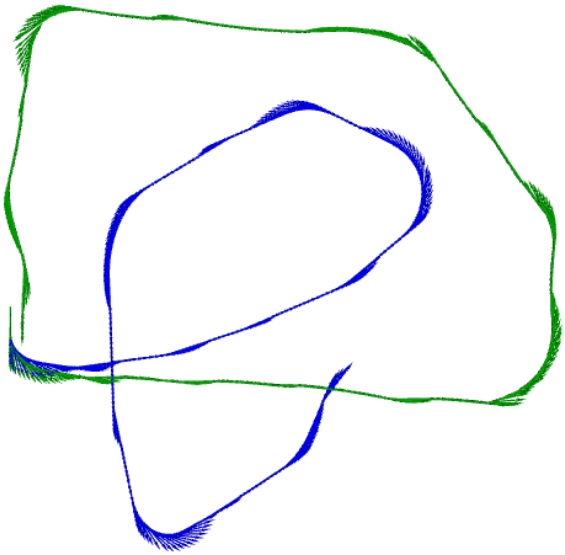


Figure 3.3 - Dead Reckoning (Blue) Comparison to EKF Odometry (Green)

It was observed that the odometry model appears to translate linearly longer than the generated map as per Figure 3.4. This causes problems in the map localisation stage as the odometry model reports a linear translation greater than the observed LIDAR scans. This design consideration is explored in 5.2.

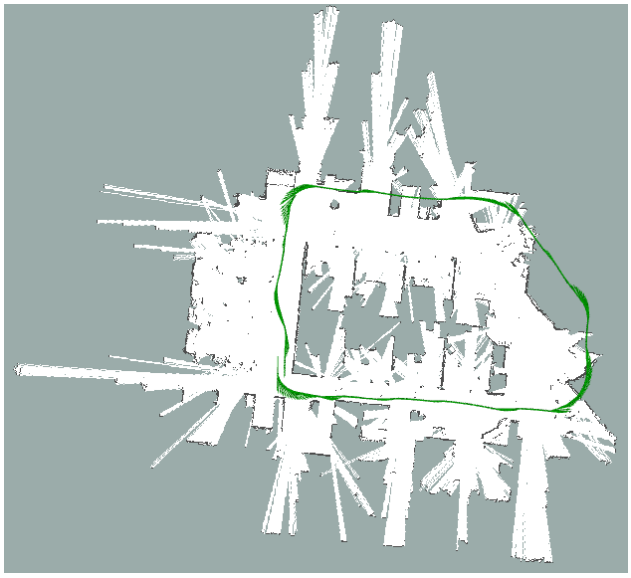


Figure 3.4 - Odometry Model Inaccuracy vs Map

4 2D LIDAR Mapping

The LIDAR Mapping solution requires the previous EKF odometry model and a *sensor_msgs/LaserScan* source. The NightRider platform produces a *sensor_msgs/LaserScan* from the A3 RPLIDAR in the *laser* reference frame.

4.1 Design

GMapping is a ROS package by OpenSLAM that employs a Rao-Blackwellised Particle Filter (RBPF), whose purpose in SLAM is to estimate the joint posterior about the map and the trajectory of the robot [1]. The ROS wrapper for the algorithm is known as an *out-of-the-box* solution where the wrapper requires minimal modifications to work.

This was the case with this project where only the parameters *particles, linearUpdate,* and *angularUpdate.* The first parameter is to increase the total particles available to the filter for sampling, this has been increased from the default of 30 to 100 to increase density and improve performance. The other two parameters are to force a filter update every 0.1 meters of linear translation and 0.03 radians of rotational translation. The effect of these parameters causes the filters quality to increase and the state to update more often in translation.

4.2 Implementation

The A3 RPLIDAR used by the NightRider platform interfaces via an Original Equipment Manufacturer (OEM) ROS package, *rplidar_ros*[7]*,* which publishes a topic of type *sensor_msgs/LaserScan,* of which GMapping subscribes to. The LIDAR boasts a 360° Field of View (FOV), 25-meter maximum range, and a 16000 sample per scan resolution.

The odometry model generated by the EKF produces the robot trajectory of which the GMapping algorithm uses in its Particle Filter (PF) estimation. The odometry model does contain a high level of covariance, however it's found that the covariance doesn't affect the performance of the RBPF.

[7] http://wiki.ros.org/rplidar_ros

4.3 Result

The results of the mapping stage can be summarised by comparing the OGM to the floor plan as per Figure 4.1. The 3 OGMs are each generated from different data sets where each generate relatively the same map. In comparison to the floor plan the general shape of the OGMs match with clearly defined features such as walls and office spaces.

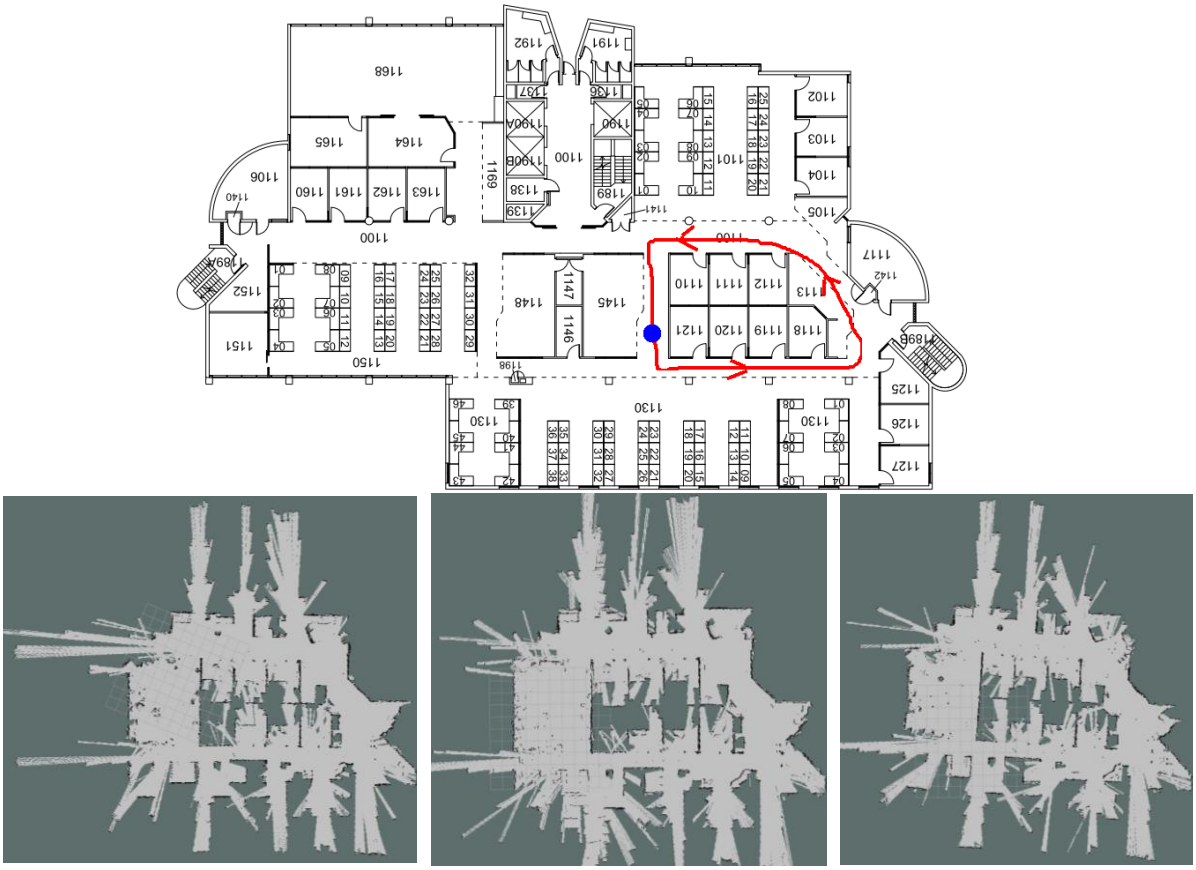


Figure 4.1 - RBPF Mapping Results

The 360° FOV of the LIDAR appears problematic as the scan *sees* the legs of the user walking behind NightRider, this is necessary to control the platform. The effect of this is isolated clusters along the free space of the map. These clusters do not seem to create issues in the operation of the later stages; however they are not ideal to the accuracy of the OGM. To solve this, an Open Computer Vision[8] (OpenCV) script could be employed to filter the OGM if the cluster area is not greater than some value.

5 Map Localisation

The map localisation solution implements the Adaptive Monte Carlo Localisation algorithm which uses a probabilistic approach to compare 2D LIDAR scans to a known map. The AMCL algorithm uses the odometry model, or rather the transform of base_link->odom, to verify the pose estimation in its comparison.

[8] https://opencv.org/

5.1 Design

The AMCL ROS wrapper requires an OGM of type *nav_msgs/OccupancyGrid* of which describes the environment. The OGM used is loaded using the *map_server*[9] package in ROS from one of the OGMs generated in the previous mapping stage. For this stage of AMCL, global localisation is not implemented and so requires an initial pose. Therefore, the data set used must match the OGM generated. The node also requires the transform of *base_link -> odom,* of which the robot's motion is extrapolated from and the 2D laser scans (*sensor_msgs/LaserScan).*

AMCL produces a *geometry_msgs/PoseWithCovarianceStamped* of which is of the reference frame *map*. The pose estimation is a probabilistic estimation based on its *geometry_msgs/PoseArray* which is the estimate of each sample.

5.2 Implementation

AMCL is also known for being an *out-of-the-box* solution which was relatively the case if not for the issues as per 3.3 Results that the odometry model is inflated compared to the map in linear translation. AMCL has parameters for noise in the odometry model which is set to a default of 0.2. The noise of the translation estimate from the translational component of the robot's motion requires tuning so that the linear translation of the odometry model is trusted less and the AMCL linear translation estimate.

The map generation was successful, and thus none of the parameters for the LIDAR estimation required tuning. These parameters only require tuning if the LIDAR is misaligned from the map when stationary. This was not the case for NightRider, the A3 RPLIDAR is a very accomplished and was expected to perform very well.

5.3 Results

The AMCL analysis is not easy to assess through still images except for single cases. The still images used to analyse the results has been uploaded to YouTube[10]. The analysis cases explore the performance of the effects of the linear translation tuning, estimation misalignment, and estimation correction.

As per the design, considerations pertinent to the odometry inflation to the map, the linear covariance of the pose estimation was rather large compared to lateral covariance. This effect is seen in Figure 5.1 shows that the pose estimation was still able to relatively align itself to the map appropriately.

[9] http://wiki.ros.org/map_server
[10] https://youtu.be/edF884Wh4tA

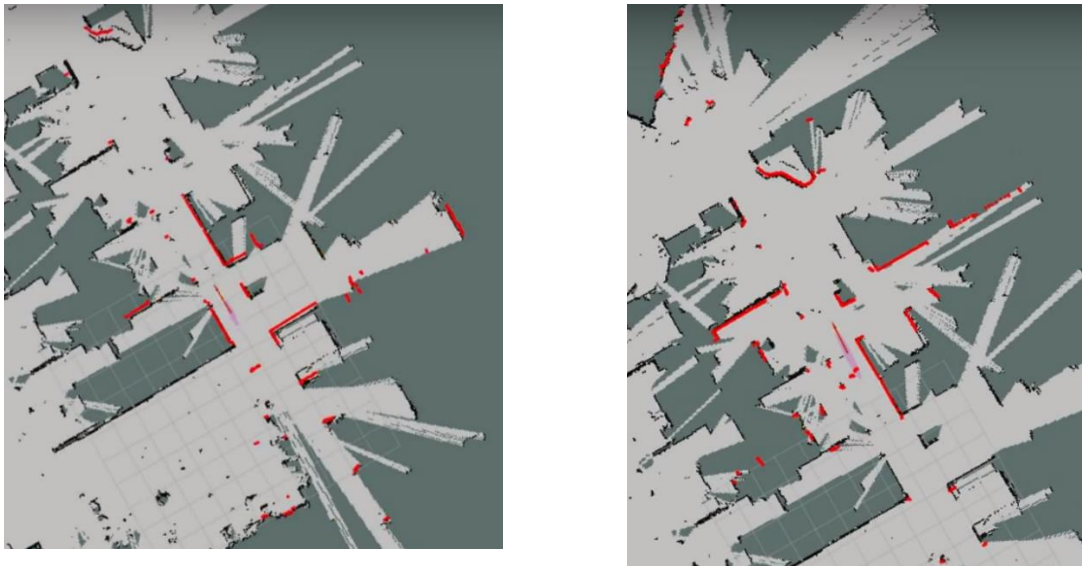


Figure 5.1 - AMCL Linear Covariance Example

As seen in Figure 5.2 there are cases in the AMCL estimation where the pose estimation shifts of which the estimation to the map is misaligned. The purple circle encircling the pose estimation describes the uncertainty in the estimation. The misalignment in the estimation can be seen in the laser scan source consistently being offset.

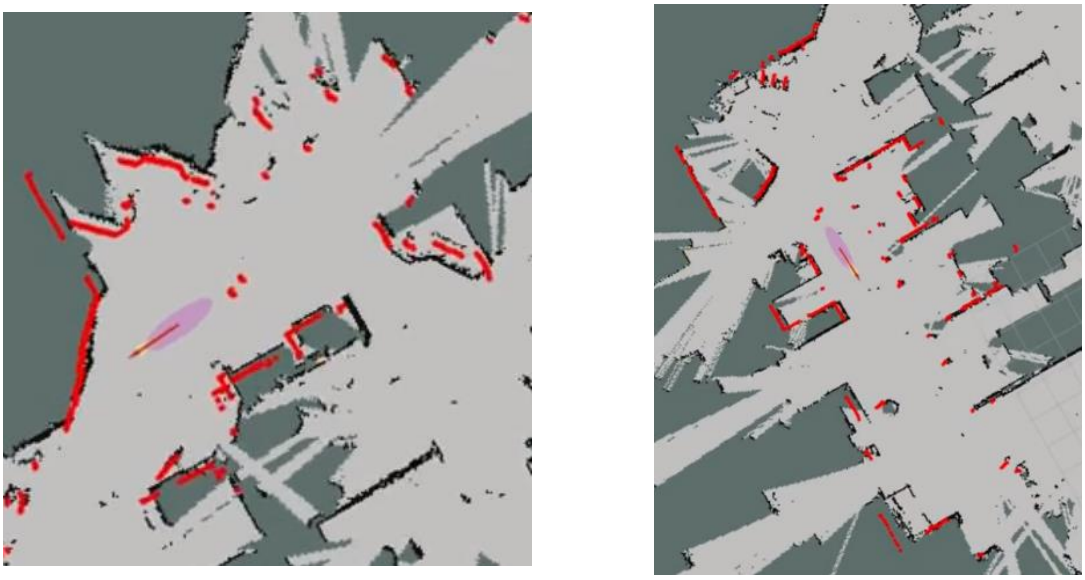


Figure 5.2 - Misalignment in AMCL Estimation (Left) and Corrected Pose (Right)

It was found that, like in Figure 5.2, once corners (right angled features) become present the pose estimate is able to correct its position. This issue shouldn't be as great of an issue when the linear translation estimate is solved as AMCL will be able to use the linear translation to correct such a misalignment.

6 Future Work

In the next stage of research, the odometry model will need to be solved. There are 3 cases suggested to identify the odometry issue; odometry model distance travelled against actual distance travelled, map sizing, and LIDAR distances.

The first of these theories should look to explore the distance the robot travels against the distance the odometry model estimates. This will determine if the EKF odometry model is overestimating lateral translation or if the problem exists in one of the other areas. The second of the theories should look to measure clearly identifiable landmarks in the OGM using the RVIZ measure tool against the actual size of these landmarks. This will determine if the GMapping algorithm might be shrinking the map in its optimisation step which might require tuning of the GMapping parameters. The final theory is that the 2D LIDAR might report distances incorrectly. If this is the case, then the GMapping optimisation step would cause the map to *shrink.*

With time, this allows NightRider to implement the basis for navigation incorporating obstacle detection and avoidance. The ROS navigation stack employs a package, *move_base*[11], which implements an algorithm used in *skid_steer kinematic* [2] navigation. The package *teb_local_planner*[12] by Christoph Rösmann uses the same planner as move_base however, has integration for vehicles using an Ackermann kinematic model. Leveraging the work to date, these navigation and path planning techniques could be integrated to assist the NightRider platform.

7 Conclusion

Conclusively, the final solution presented in 5.3 performed rather well with the robot being able to localise itself inside of the known environment after variable tuning. The mapping stage using the RBPF algorithm appropriately generates an OGM of the environment irrespective of the data set used. The EKF localisation stage exhibits an issue in the estimated odometry model. The motion model resembles the correct shape of motion, however, is of an incorrect scale to the map. As per 6 Future Work it is unclear why this discrepancy occurs, however, 3 approaches to address this have been suggested for the next stage of research. With this issue corrected the AMCL stage should become more accurate and reliable for navigation.

In conclusion, the research is contributing to an understanding of the importance of the localisation step to all other stages in the final solution. The research so far has demonstrated capabilities of autonomous vehicles to localise and map whilst also presenting the challenges, especially in hardware choice and the impact this has on the solution. It has become apparent how important an accurate odometry model is to any advanced work in this area.

[11] http://wiki.ros.org/move_base
[12] http://wiki.ros.org/teb_local_planner

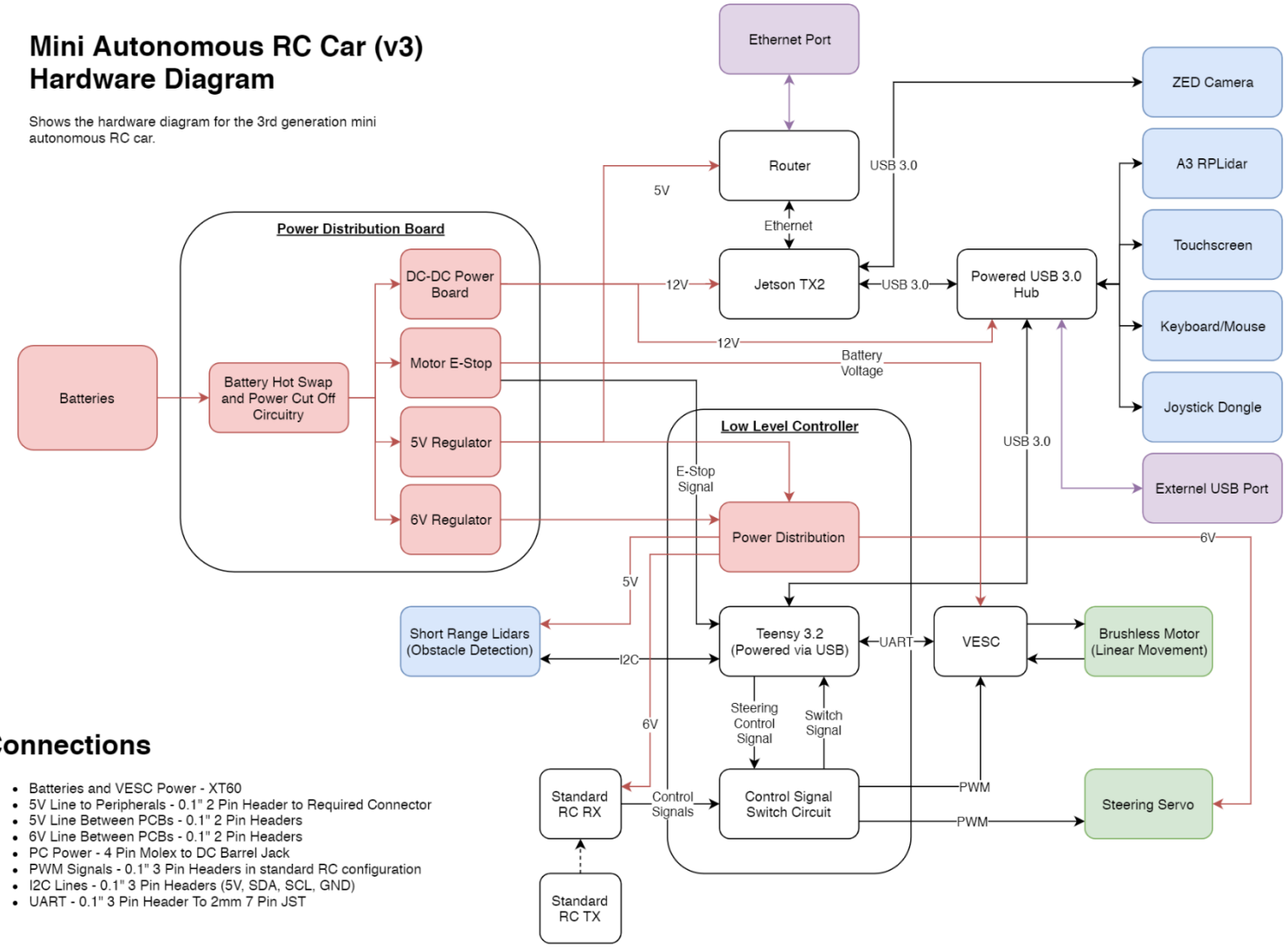
8 References

[1] S. Parida, M. Franz, S. Abanteriba and S. Mallavarapu, “Prospects, Obstacles, User Acceptance and Public Opinion,” in *Advances in Intelligent Systems and Computing*, 2018.

[2] G. Grisetti, C. Stachniss and W. Burgard, “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters,” in *IEEE Transactions on Robotics*, 2007.

[3] T. Wang, Y. Wu, C. Han, J. Chen and Q. Zhao, “Analysis and Experimental Kinematics of a Skid-Steering,” in *Sensors*, Beijing, 2015.

Appendix 1: Hardware flowchart by James Mount



Connections

- Batteries and VESC Power - XT60
- 5V Line to Peripherals - 0.1" 2 Pin Header to Required Connector
- 5V Line Between PCBs - 0.1" 2 Pin Headers
- 6V Line Between PCBs - 0.1" 2 Pin Headers
- PC Power - 4 Pin Molex to DC Barrel Jack
- PWM Signals - 0.1" 3 Pin Headers in standard RC configuration
- I2C Lines - 0.1" 3 Pin Headers (5V, SDA, SCL, GND)
- UART - 0.1" 3 Pin Header To 2mm 7 Pin JST

Appendix 2: EKF Config File

```
frequency: 10
sensor_timeout: 0.5
two_d_mode: true
transform_time_offset: 0.0
transform_timeout: 0.0
print_diagnostics: true
debug: false
debug_out_file: /path/to/debug/file.txt
publish_tf: true
publish_acceleration: false

map_frame: map              # Defaults to "map" if unspecified
odom_frame: odom            # Defaults to "odom" if unspecified
base_link_frame: base_link  # Defaults to "base_link" if unspecified
world_frame: odom           # Defaults to the value of odom_frame if
unspecified

odom0: /nightrider/odom
odom0_config: [false,  false,  false,
               false, false, false,
               true, true, false,
               false, false, true,
               false, false, false]

odom0_queue_size: 5
odom0_nodelay: false
odom0_differential: false
odom0_relative: false
odom0_pose_rejection_threshold: 5
odom0_twist_rejection_threshold: 1

imu0: /tx2/imu
imu0_config: [false, false, false,
              false,  false,  false,
              false, false, false,
              false,  false,  true,
              true,  false,  false]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
imu0_remove_gravitational_acceleration: true


use_control: false
```

```
process_noise_covariance: [1e-3, 0,    0,    0,    0,    0,    0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    1e-3, 0,    0,    0,    0,    0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    1e-3, 0,    0,    0,    0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0.3,  0,    0,    0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0.3,  0,    0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0.01, 0,     0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0.5,   0,    0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0.5,  0,    0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0.1,  0,    0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0.3,  0,    0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0,    0.3,  0,    0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0,    0,    0.3,  0,    0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0,    0,    0,    0.3,  0,    0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0,    0,    0,    0,    0.3,  0,
                           0,    0,    0,    0,    0,    0,    0,     0,    0,    0,    0,    0,    0,    0,    0.3]


initial_estimate_covariance: [1e-9, 0,    0,    0,    0,    0,    0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    1e-9, 0,    0,    0,    0,    0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    1e-9, 0,    0,    0,    0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    1.0,  0,    0,    0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    1.0,  0,    0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    1e-9, 0,    0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    1.0,  0,    0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    1.0,  0,    0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    1.0,  0,     0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    1.0,   0,    0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,     1.0,  0,     0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,     0,    1.0,   0,    0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,     0,    0,     1.0,  0,    0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,     0,    0,     0,    1.0,  0,
                              0,    0,    0,    0,    0,    0,    0,    0,    0,    0,     0,    0,     0,    0,    1.0]
```

Appendix 3: AMCL Parameters

```
odom_alpha1: 0.8
odom_alpha2: 0.4
odom_alpha3: 7.95
odom_alpha4: 0.6
```