High Speed Autonomous Vehicle for Computer Vision Research and Teaching

Michael Mollison
20772677
School of Electrical, Electronic and Computer Engineering
University of Western Australia

Supervisor: Prof. Dr. rer. nat. habil. Thomas Bräunl
School of Electrical, Electronic and Computer Engineering
University of Western Australia

Final Year Project Thesis

School of Electrical, Electronic and Computer Engineering
University of Western Australia

Word Count: 6600

Submitted: 1st November, 2017

ABSTRACT

This thesis presents the development of a rugged, high-speed, low-cost, and highly adaptable autonomous ground vehicle, to serve as an educational tool for future Robotic Engineering students at The Universtity of Western Australia, and enabling future research into Automous Navigation and Computer Vision not possible with the university's current fleet of robots. The vehicle was developed with the goal of facilitating future research into Autonomous (car) Driving, and Self Localisation and Mapping (SLAM) techniques, utilising progressively lower cost sensors and hardware than traditionally employed for the purpose, thereby lowering the cost of entry for research into the field. The vehicle was developed around a highly configurably consumer R/C car to ensure stability at high speeds, over varied terrain, both indoors and out, ensuring adaptability to future test conditions, and autonomous and SLAM capabilities utilising a 2D Laser Distance Scanner as it's only sensor were developed as a proof of concept and baseline for future research.

The software was developed on top of UWA's RoBIOS software, which provides a touchscreen interface for users to run programs as well as providing a simple API to interact with the robot. As an additional part of this part of this project, improvements and alternatives to RoBIOS were investigated, and functionality required for the new vehicle was added to the RoBIOS API, providing a more feature rich environment for future Robotics students.

LETTER OF TRANSMITTAL

Michael R. Mollison
174 Little Marine Pde
Cottesloe, WA, 6011

30th October, 2017

Winthrop Professor John Dell
Dean
Faculty of Engineering, Computing and Mathematics
University of Western Australia
35 Stirling Highway
Crawley, WA, 6009

Dear Professor Dell,

I am pleased to submit this thesis, entitled “High Speed Autonomous Vehicle for Computer Vision Research and Teaching”, as part of the requirement for the degree of Bachelor of Engineering.

Yours Sincerely,

Michael R. Mollison
20772677

ACKNOWLEDGEMENTS

I would like to thank the following people for their support, contributions, and patience, without whom I would not have been able to complete this thesis:

- Prof. Thomas Bräunl for his supervision at the University of Western Australia, and for providing the opportunity to work on this project.
- Omar Anwar and Fangpeng Li for the countless hours and late nights spent rewriting and debugging and re-explaining code.
- Marcus Pham and Franco Hidalgo for their ideas and support with EyeBot and RobiOS.
- John Hodge for tech support
- FOSS developers
- My friends and family, especially over the few days prior to the due date

TABLE OF CONTENTS

1 INTRODUCTION

A robot requires an accurate understanding of it's environment if it is to be able to interact with it effectively. The degree of self and environmental awareness of a robot is often dependent on the quality of sensory information available. Self-driving vehicles may employ a wide variety of sensors, but it is common to see them rely heavily on Laser Scanners known as LIDAR, which can provide accurate distance measurements of the robots surroundings at high frequencies, and typically at high cost. The primary focus of this project was developing and testing an autonomous vehicle using LIDAR as its only sensory input, and therefore providing a baseline to compare results of future research into autonomous driving with lower cost sensors. [1] [2]

The project is perhaps best summarised by dividing it into 3 distinct stages:

- A study of software available for the Raspberry Pi 3, focussing on compatibility with UWAs existing robot interface RoBIOS
- Developing a self-driving algorithm to prove the ability to drive autonomously with a 2D Laser Scanner
- Demonstrating the ability of the Raspberry Pi 3 to perform a SLAM algorithm

Each of these stages contributes to the primary aims of the project, to enhance the existing educational resources available to UWAs robotics students, and to prove the capability of the platform to enable research into autonomy and SLAM using alternate sensor.

Raspberry Pi 3s running RoBIOS software are currently used by The University of Western Australia in the practical education of Electrical Engineering students. Work initiated with testing and troubleshooting of the RobiOS software through the development of an autonomous, maze navigating robot. As well as allowing familiarisation and refinement of RobiOS, this testing highlighted the difficulty of using calbrated, single dimension, IR-based Position Sensing Devices (PSDs) to navigate, even at low speed. As such, LIght-baser raDAR (LIDAR) sensors are frequently employed to assist in autonomous vehicle navigation.[3]

In order to test navigating with LIDAR, an alternate vehicle to the Eyebots normally used by had to be developed and configured. A 1:10 scale, Traxxas Stampede Radio Control vehicle had previously been used within the department and was available for use. This vehicle is a RWD electric car with servo controlled steering and capable of speeds up to 50km/h (scaled speed 500km/h).

The primary focus of the this project was configuring the car, the Raspberry Pi 3, and the Robios software to work in harmony enabling the vehicle to safely navigate autonously, at relatively high speed, through an office like environment using solely 2D LIDAR measurements (no odometry). The result is a stable autonomous platform which can easily be expanded to test more complex driving and mapping algorithms, or as a basis for comparing performance of alternate sensors. The RoBIOS software and it's underlying operating system Raspbian were also investigated, developed, and updated significantly to allow future Robotics students a simple way to access features used for in this vehicle.

2 HARDWARE

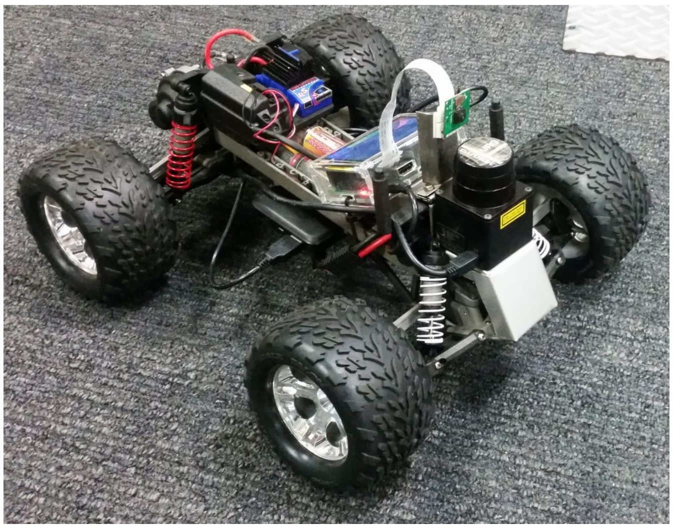


Figure 2.1: The Assembled Robot

As with any project, there were limitations on what we could and couldn't do to meet our objectives. Specifically, the robot was to use components (sensors, controllers, etc.) already owned by the Robotics department; a new Laser Scanner was not to be used for example. Compatibility with the departments existing teaching robots and platform, EyeBot and RoBIOS respectively, was to be maintained to the highest level possible, to ensure any new features implemented could be implemented across all the teaching robots, as well as providing a consistent user interface across the department's fleet.

These limitations dictated the physical components, and subsequently additional software, of the robot, which consists of a Traxxas Stampede R/C Car, Raspberry Pi 3 Single Board Computer with a 3.5" TouchScreen for UI, Hokuyo URG-04LX-UG01 Laser Scanner, and a Logitech Gamepad for remote control. These components would therefore be physically connected and virtually interfaced to become the autonomous platform in Figure 2.1.

2.1 Traxxas Stampede R/C Car

The commercially available *Traxxas Stampede XL-5®* is a 1/10 scale R/C (Radio Control) Car, and is the backbone of the robot and perhaps the primary component allowing the adaptability and flexibility of the end product. The car features:

- Speeds up to 50km/h or increased torque at lower speeds via adjustable gearing
- Fully adjustable suspension and steering similar to that found on a car
- Waterproof motor and electronics

- Rear wheels driven through a differential, brakes via the motor (through the rear wheels only)

The *Traxxas Stampede* was available to the department following previous research

Additional information can be found at the Traxxas website [4]

2.2 Hokuyo URG-04LX-UG01 Laser Scanner

As mentioned, Laser Scanners are widely used in Robotics to provide high quality nearest distance data to a controller for processing. Available to us was the Hokuyo URG-04LX-UG01. Important specifications include[5]:

- 10Hz scan rate
- 240° Field of Vision
- 0.352° Angular Resolution
- 20 – 5600mm range, ±3%
- USB power and communication

2.3 Logitech Gamepad F710

This is a wireless, gaming console style, handheld controller. It uses a USB dongle and operates at 2.4GHz.

The controller features 2 “triggers’ with 8 bit resolution, and two thumbsticks featuring 32 bit resolution in 2 axes.[6]

2.4 Raspberry Pi 3

The primary teaching robot currently used at the UWA is known as “EyeBot”, and is a small differential drive robot controlled via a Raspberry Pi 3 Single Board Computer (SBC) running a GNU/Linux distribution “Raspbian” and the university RobiOS robot interface. As we wished to build upon RobiOS to enable the new robots use as a learning aid, the Pi 3 was the obvious choice of controller. Additional Pi 3 features taken advantage of by include:

- USB powered with a relatively low power draw
- Onboard Wi-FI, Bluetooth, 100M Ethernet, 4xUSB2.0 ports
- General Purpose Input/Output (GPIO) pins, enabling the use of a 3.5” LCD touchscreen for a user interface with remaining pins for interfacing with external components
- Full Linux development environment with very active community ensuring high compatibility of software and hardware
- Small footprint with multiple enclosures and mounting options available

More information is available through the Raspberry Pi Foundation [2]

3 SOFTWARE

The use of the Raspberry Pi 3 SBC opens up a realm of software possibilities for robot control, development, and interfacing in comparison to more traditional embedded robotics microcontrollers such as the Motorolla 68k series. Different operating systems available for the Pi 3 are available with varying levels of Robotics software support. Several were investigated for stability, usability and compatibility with UWA's RoBIOS software [2]

3.1 SINGLE BOARD COMPUTERS

Traditional embedded microcontrollers, such as the M68322 used on the previous generation of the Eyebot teaching robots, limit power and resource usage by only running and storing what is needed at any one time. The original RoBIOS interface and the desired user programs would be close to the only software stored and run by the controller.

The Pi 3 is a Single Board Computer (SBC), which by definition is much more akin to a modern home computer than the traditional embedded controller. SBCs require an Operating System (OS) to provide access to the array of features integrated on the board (eg the CPU itself, network interfaces, storage). This is traditionally installed as part of a Linux distribution, which also provides repositories of compatible user applications and packages to customise and extend your system, as you might your PC.[7]

3.2 Pi 3 DISTRIBUTIONS

The Pi 3's large userbase and development community has led to a number of official OS releases from major Linux distributions (Raspbian, Ubuntu, openSUSE etc.), as well as a number of unofficial OS builds, each implementing a unique set of features or software. To make matters more complicated, the OS officially supported by The Raspberry Pi Foundation, *Raspbian,* has been through multiple iterations since it’s initial release.[8] [9]

Each of these Raspbian release was investigated for RoBIOS compatibility and features that may be desirable in our robot, as was an official Ubuntu for Pi release, configured by a third party to include a ready-to-run ROS installation. These were selected from the countless available distributions and releases due to their larger user bases and thus support facilities.

3.2.1 Raspbian Wheezy

Raspbian Wheezy was the first Raspbian (Debian for Raspberry Pi) officially released (in July 2012) and last seeing a major update in May 2015 prior to the release of Jessie. The software officially available for this release has not seen much development in several years and can feel a little dated. It is however extremely stable when configured correctly, and was the OS used by RoBIOS until mid-late-2017. ROS packages are not available nor is it recommended to build from source. The Mobile Robot Portable Toolkit is also unavailable. Due to it's age it should be avoided for new work.[10]

3.2.2 Rasbian Jessie

Rasbian Jessie was first released in September of 2015 and initially featured only minor improvements to stretch beyond compatibility with newer software and updated repositoiesy. Between its release and final major update in July of 2017 however, a lot of the underlying system, and front end, were updated to improve performance, appearance and security. Unfortunately these updates broke a lot of software, including RoBIOS when it was moved from Wheezy to Jessie in 2017.

RoBIOS has been rebuilt and is now mostly stable on Jessie, however it still exhibits problems that do not exist on Wheezy. The Jessie repositories provide limited access to the MRPT but a complete robotics development environment is not officially available or supported.

Jessie was superseded with the release of Stretch in August 2017[10]

3.2.3 Raspbian Stretch

Rasbian Stretch was released in August of 2017 and featured an updated kernel and firmware revision, even compared to a fully up to date Jessie install.

Stretch was the first release based solely on the updated init system and core packages and not incrementally upgraded like Jessie. This made compatibility with older software (eg RoBIOS) a potential issue.

The software repositories include prebuilt ROS packages to install, however at the time they were tested (not long after the August release), they did not seem to be configured correctly, and many packages referenced in the official ROS documentation and tutorials did not seem to have any equivalent in the Raspbian release.

That may be resolved in the future, however if ROS were required, it can be built successfully from source on Stretch with multiple reports of success online and ErleRobotics providing support to commercial customers.

ErleRobotics is also working with ROS 2 (currently still alpha) on Pi 3s running Stretch.[11]

The Mobile Robot Portable Toolkit (MRPT) installed successfully from the official Raspbian repositories and worked as documented out of the box. Unfortunately MRPT is not nearly as prominent in the online Robotics community and it was difficult to find support. Very limited testing had the icp-slam implementation working with the Hokuyo Laser within minutes of installation however no progress was made beyond that.[12]

RoBIOS was updated to work on a recent Jessie install only weeks before the official release of Stretch. Not seeing the logic in updating all the teaching robots to an already outdated OS, I spent many hours configuring Raspbian Stretch to successfully build RoBIOS from source, and many more hours making replacing code broken by the new OS (eg. ifconfig to find ip address) with their new counterparts (eg. ip a).

A number of issues were fixed, especially those responsible for crashes, but bugs remained and the team responsible for maintaining the EyeBots simply didn't have the resources to perform another upgrade. EyeBots and therefore the RoBIOS software are therefore running on the older Rasbian Jessie.

The size of the OS images increased to around 3.5GB for the GUI or Desktop versions of Jessie or Stretch compared to the 2GB or less for Wheezy. I chose to attempt to build a lighter system by using the Raspbian Stretch Lite (no gui) image and building only the packages required by RoBIOS, and MRPT for my own education. If anybody reads this and would like a copy, don't hesitate to contact me. [10]

3.2.4 Ubuntu + ROS

The developers at "German-Robot Humanoid Robots" have provided instructions to build ROS Kinetic on Ubuntu 14.04 for Pi 3, as well as providing a ready to run image. Unfortunately official support for 14.04 has now ended.

This image was not encountered until further into the project and as such RoBIOS compatibility was not checked. Instead the image was used to investigate the possibility of integrating ROS SLAM algithms into our robot, but after completing a number of ROS tutorials and realising the magnitude of the ROS project, the idea was forgotten.

Despite all this, for someone wanting to learn ROS, this image and a Pi 3 is a much nicer way to do so than installing on your PC and it is surprisingly responsive and smooth.[13]

3.3 OPERATING SYSTEM SUMMARY

After researching, downloading, flashing, updating, configuring, and repeating all the aforementioned Pi 3 Operating Systems, it shouldn't surprise the reader to read we are still using Wheezy, the original version of Raspbian from 2013. Officially EyeBots are now running RoBIOS on Jessie. In our experience this is still not as stable as the Wheezy installation, and there is no benefit for our project to upgrade at this time.

Additionally, the I/O expansion board used by EyeBot connects via USB to the Pi 3 at ttyACMx, as does the URG-04LX. [5]This has been solved in our RoBIOS by modifying the way USB devices are detected in the source code. The same correction will work in Jessie but due to restricted code access it has not yet been implemented.

The Stretch image with RoBIOS (created from the corrected but old source) worked with our program reliably however to eliminate the possibility of the new, relatively untested OS affecting behaving strangely we opted to revert to Wheezy. Our original desire to upgrade was to make use of ROS or MRPT in our autonomy and laser visualisation and processing and still remains a potential extension of this work. The complexity of ROS and lacking documentation of MRPT put this option out of our reach with the timeframe we had. Our recommendation for future OS and RoBIOS development is to use the latest version of Raspbian Stretch to take advantage of it's performance and security improvements and allow access and compatibility with the latest software repositories and robotics platforms.

3.4 ADDITIONAL SOFTWARE REQUIRED

With the Operating System decided, it was necessary to start interfacing the Pi 3 with the additional hardware. Fortunately an open-source software solution existed for all these devices enabling reliable communication between the Pi 3 and peripheral devices.

3.4.1 Traxxas ESC and Servo Interface

The Traxxas R/C car originally utilised a radio transmitter/receiver combination common to most radio control vehicles, indicating the Electronic Speed Controller was most likely controlled with the standard servo-style PWM signal; 1-2ms pulses every 20ms, or at a rate of 50Hz. This was confirmed by applying the appropriate signal to the control wire previously attached to the R/C receiver with the battery connected and ESC enabled. The steering servo is a standard model airplane type servo utilising the same protocol. We therefore required 2 servo-style pulse generators, and a reference for ground.[4]

The Raspberry Pi SBCs feature 2 rows of General Purpose, Input/Output (GPIO) pins, enabling digital input and output on most of the pins as specified by the user. The Pi 3 features 40 GPIO pins, 26 of which are utilised by the 3.5" LCD TouchScreen leaving 14 pins including the requiured ground free to use.

There is no obvious provision for accessing and controlling these pins in the standard Raspbian Distribution, beyond using an inefficient and inaccurate software defined output.

Fortunately, user "richardghirst" of Github has developed ServoBlaster, a user space daemon (background process), which sets any GPIO pin as defined by the user to the pulse width value written to the file /dev/servoblaster.

Simply put, he has provided a simple interface enabling PWM signals to be generated, in hardware, on any of the GPIO pins according to the timing value and pin specified by the user. This proved to be perfect for our requirements and required only a slight modification to the default configuration to ensure the pins used by the display were not being modified.[14]

3.4.2 Hokuyo URG-04LX-UG01 Interface

The developers of the Laser Scanner provide the Urg Library, an open source C library providing a very complete interface to the URG Laser Scanners. [15]This was the first and most obvious library encountered for this sensor, so is the one we worked with most.

The library is designed to work with all URG scanners and provides far more functionality than required. It is for the most part simple to use but documentation is lacking and difficult to find, and, as with commenting throughout the code, is often in Japanese Kanji. Regardless, it has proved to be a robust and reliable interface to the sensor.

BreezyLIDAR provides a much simpler interface for accessing the scanners measurements and is designed to work directly with BreezySLAM if desired.[16]

ROS, MRPT and others provide their own drivers and interfaces for accessing the Laser data.[17] [18]

It is important to note that you may only use one of these libraries at any one time. Multiple requests to open or access the device will result in errors or erraneous behaviour

The software URG Viewer, which provides a graphical display of the current distance measurements from the scanner, was compiled for the Pi 3. This provided a simple verification of scanner functionality and settings, and a convenient way to visualise the incoming data.[19]

Unfortunately URG Viewer makes use of the QT application framework and we experienced issues making this compatible with Remote Desktop Protocol (RDP). RDP is used heavily in the department to provide a virtual monitor for the Pi and we therefore discontinued use of URG Viewer.

3.4.3 Logitech Gamepad F710

The Logitech Gamepad utilises a wireless usb dongle to receive user input from the wireless gamepad.

The OS kernel automatically supports the device, which mounts at /dev/input/eventX where X depends on the order the kernel recognises the device.

Trying to display the incoming data directly results in unreadable characters, however the event tester (evtest) utility, installed via the Raspbian repositories, recognises the incoming data and prints out the appropriate "event" or keypress.[20]

3.5 PRELIMINARY TESTING

Prior to working on any more advanced development, the ability of the Pi 3 to assume control of the Traxxas, and it’s ability to autonomous code or building maps with Laser Scans, we needed to configure and test the basic controls of the vehicle; Acceleration, Braking and Steering.

Using evtest, we recorded the "Types" and "Codes" of various controller inputs and their associated values. We opted to utilise similar controls to those that might be found in a Driving game utilising a similar controller (Right Trigger = Accelerator, Left stick = Steering...)

Evtest is free software, allowing editing of the source to scan the incoming events for the previous identified Types and Codes. On identification, we are able to use ServoBlaster to control motor speed and steering position based on the values attached to the relevant input events.

The resulting program enabled manual operation of the car through the wireless controller and served as the basis for all future experiments.

3.6 ADDITIONAL SOFTWARE SUMMARY

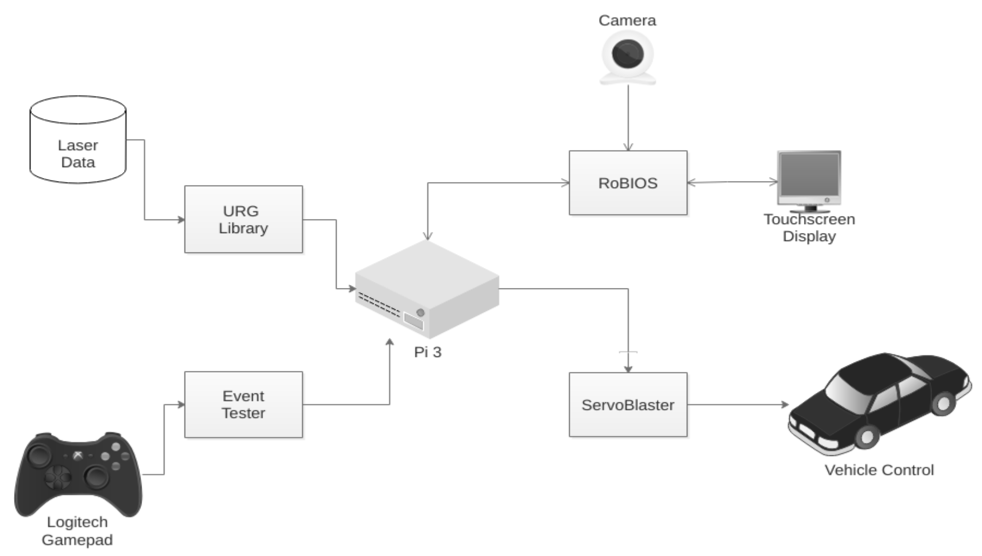


Figure 3.1: Simplified data flow diagram. The arrows indicate the direction of the relevant sensor or control data, which is interpreted correctly by the software indicated in the intermediary blocks.

The project required a way for the Raspberry Pi 3 to interact with devices not before used with the Raspberry Pi 3 within the department. Figure 3.1 shows the devices and the software we chose to use. We made sure to test the new software for compatability with RoBIOS to ensure anything we made use of could be incorporated into future

RoBIOS development. ServoBlaster to allow PWM control of any GPIO pin is already in use, and the BreezyLIDAR package for laser scan access is being developed. The additional features these packagegs provide will soon be accessible through the Eyebot API for use by students studying robotics, hopefully providing a richer and more valuable learning experience.

4 DRIVING PROGRAM

A fairly rudimentary autonomous driving program has been developed for the vehicle and parameters optimised for the Electical Engineering hallways enabling speeds of up to and perhaps beyond 20km/h, with average speeds while performing “laps” of the 4th floor around 12km/h.

Autonomous Navigation for wheeled vehicles is a well developed and researched field of mobile robotics; the increasing number of autonomous passenger vehicles on the road is evidence of this. Nonetheless the development of a seemingly simple autonomous driving routine from scratch is not an easy task and is quickly complicated by the reality of uncertainty and limitation of sensor data availability in the real world. The following sections provide an overview of the final program, some of the complications we encountered developing this autonomous routine, and solutions to these issues.

The program is still under development and is in no way an optimal or revolutionary method of navigation. Nonetheless it has proved effective and serves as a great demonstration of the capabilities of a robot using a 2D Laser scanner. It also may provide a benchmark for comparison should alternative sensor research proceed.

4.1 USER INTERFACE

The program makes use of the RoBIOS API to provide a graphical user interface (GUI) to the user.

Upon running the program or returning from other screens, Figure 4.1 is displayed to the user. On the left, a map showing the location of the most recent laser scan data (in red) is displayed, to the right, the current camera view is displayed.

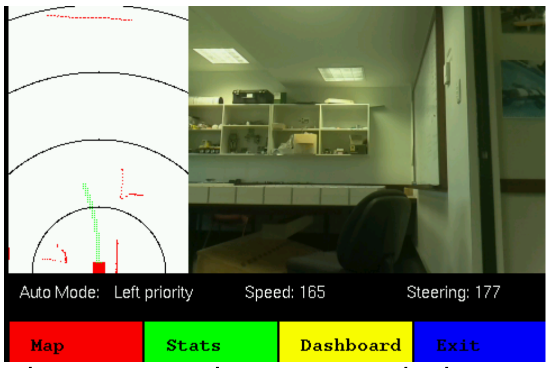

Auto Mode: Left priority Speed: 165 Steering: 177

Figure 4.1: Main Program Display

Below these, the current drive mode (see 4.2) is displayed, alongside the current motor and steering values (in Pulse Width Duration)

If the program is running in autonomous mode, the map displays an approximate vehicle trajectory as seen in Figure 4.1. The colour of this trajectory indicates whether the vehicle is accelerating (green) or decelerating (red).

The colourful row along the bottom of the display provides access to alternate displays:

Stats provides information mainly for debugging and tuning driving parameters,

Dashboard allows changing program options via the touchscreen, specifically whether text is displayed correctly for the LCD or for a remote desktop session, although other options could be configured.

4.2 DRIVE MODE

A number of “Drive Modes” are available to the user depending on the current use case.

4.2.1 Manual Mode

The default mode, used when the program is first started, is Manual Mode. This provides complete manual operation of the car via the Logitech Controller, with Controller inputs mapping directly to Steering and Speed outputs (Speed output is still limited to hard-coded value).

Implementing some form of collision avoidance was considered but decided against due to the common Manual Mode use cases, car retrieval when the laser scanner fails.

4.2.2 Auto Mode

Auto Mode is entered by pressing the Left Bumper of the Logitech Controller and has “sub-modes” which can be selected with the D-pad of the controller.
Auto Mode lets the program take over control of the robot. As a safety measure, the Right Trigger is used as a dead-mans-switch and must be pulled to enable autonomous driving, but otherwise the robot is in full control of steering and speed and will attempt to negotiate the path it decides is best at the maximum speed it decides is possible.

The sub-modes, Left Priority, Right Priority, and Balanced, prioritise turning in the specified direction when the robot reaches a junction. The turning sub-modes are selected by pressing the preferred direction on the D-pad.

The sub-mode U-turn can be used to automatically perform a U-turn if it is possible in the robots current position. This feature is still experimental and enabled by pressing Down on the D-pad.

4.3 AUTONOMOUS DRIVING ALGORITHM

The algorithm developed for the autonomous driving mode is quite simple in it’s logic and implementation and its usage can be described by Figure 4.2.

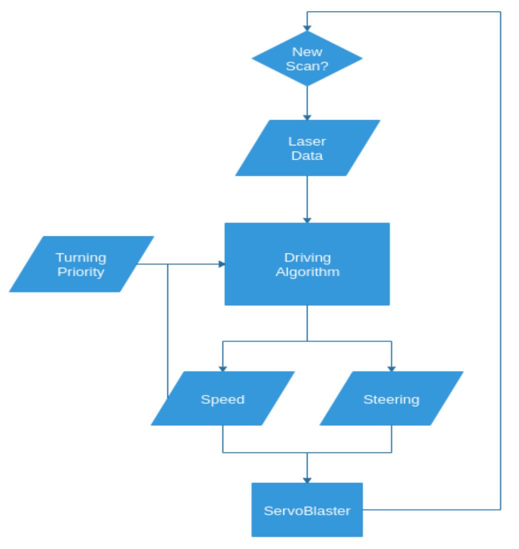


Figure 4.2: Driving Algorithm Usage

If a new scan is available, the algorithm uses the new scans data, the current turning priority sub-mode and the speed set by the algorithms previous iteration as inputs to determine the optimum settings at the current time for the steering servo and speed controller. These are applied by Servoblaster and the cycle repeats when the next scan is received.

It is important to note that the current value of “Speed” is not an accurate representation of the actual robot speed and is only the speed we have requested the vehicle to accelerate or decelerate to. In fact Speed is an inaccurate name, as its value is more closely related to the proportion of the available power that we wish the motor to see. Therefore factors like battery voltage and capacity, and physical inclines may result in considerable variance of actual speeds at the same setting.

This is especially important to consider when the robot wishes to reduce its speed but does not need to apply brakes. In such situations the vehicle “coasts” until it reaches its new set speed, meaning actual speed can remain much higher than the current Speed setting might suggest.

At a slightly lower level, Figure 4.3 shows the individual branches responisible for the robots path, one branch for deciding the vehicles speed, and one for deciding the steering angle.

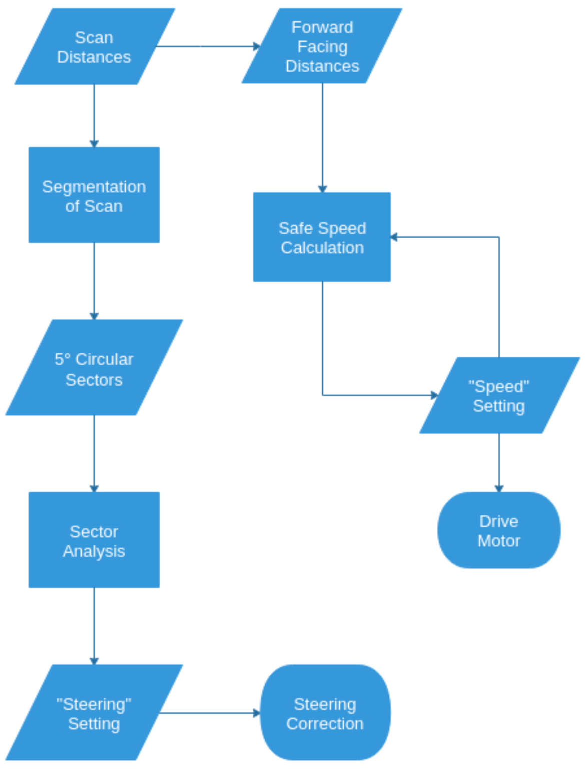


Figure 4.3: High Level Algorithm Overview

4.3.1 Speed Control Branch

The speed control branch of the algorithm is quite simple. A forward facing range of scans equivalent to the width of the car (plus a safety factor) is checked for obstacles, and a distance to the obstacle on the forward facing axis is calculated. The distance to this object and the vehicles previously set speed are used to determine the vehicles new speed and its behaviour is summarised in Table 1. It should be mentioned that there is no definition of close or distant objects in the code, the set speed is instead a function of previous speed and the forward distance to the obstacle.

Previous Speed	**Obstacle Location**	**Set Speed**
Low	None	High
Low	Distant	Increase or Maintain
Low	Close	Maintain or Decrease
Low	Danger Zone	Full Brakes
High	None	Maintain
High	Distant	Maintain or Decrease
High	Close	Decrease
High	Danger Zone	Full Brakes

Table 1: Speed Control Behaviour

The “Danger Zone” is similarly defined and represents the latest point the vehicle can safely stop without hitting the obstacle it has detected. At a high speed, the vehicle is going to take longer to brake to a complete stop and therefore the Danger Zone is larger. At low speed it is smaller, and if the vehicle has stopped, the braking distance and therefore Danger Zone will be 0.

As mentioned in 4.3, the Previous Speed variable does not give an accurate measure of actual vehicle speed, especially under deceleration and braking. Therefore as a safety measure, if the vehicle detects an object in it's Danger Zone, it will apply full braking force until either a period of time guaranteed to bring it to a complete stop has passed (time varies depnding on the previous speed at time of invocation), or the steering logic has repositioned the vehicle while braking to a position where it is safe to once again accelerate.

4.3.2 Steering Control Branch

The branch responsible for controlling the steering servomoter, shown in Figure 4.3, works in parallel with the speed control branch and performs a number of functions internally, aiming to ensure a clear forward path for maximum speed.

For simplicity, the scan data is separated into equal circular sectors, currently of 5° per sector, and the minimum horizontal and forward distances of each sector is calculated as defined in Figure 4.4.

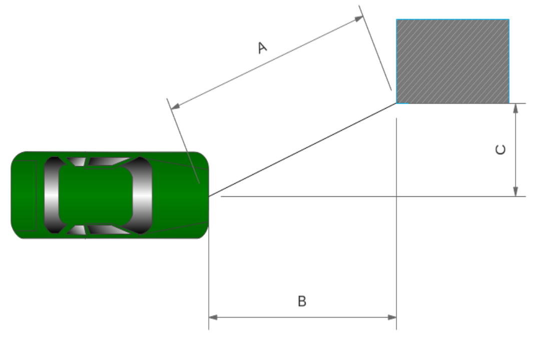


Figure 4.4: Distance Definitions
A: Actual Distance
B: Forward Distance (x-dimension)
C: Horizontal Distance (y-dimension)

These minimum distances are used in three distinct operation, visible in Figure 4.5.

The first operation uses the minimum distances across sectors to identify any objects it may need to avoid and calculates the angle required to do so.

The second operation attempts to ensure the vehicle is centered and aligned with the walls if it is in an environment like a straight corridor. The average minimum horizontal distance of each sector is calculated for the left and right side of the robot and used to calculate a steering angle for correction if required

The third operation looks for corners the vehicle may wish to perform a major turn at. In order to do so it uses both horizontal and forward distances to detect “holes” wider than the user defined minimum (eg ignore doors but turn at corridors). The steering angle required to enter any hole detected can then be calculated.

We can compare the angles generated by the 3 processes above and by using the largest angle we ensure the vehicle, combined with a weighting factor from the turn-preference sub-mode (Section 4.2.2) we ensure the vehicle will turn at any intersections it approaches as required to drive laps around the department.

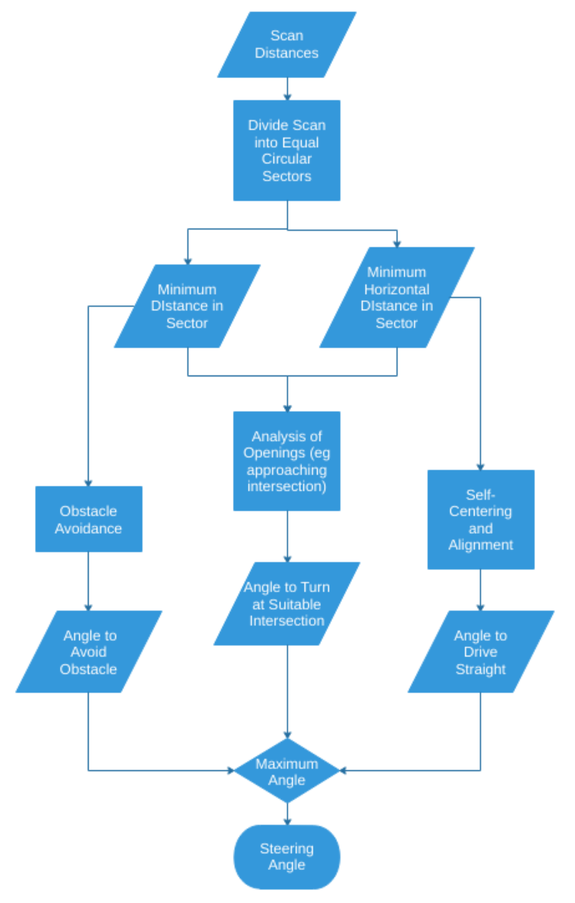

Scan Distances
Divide Scan into Equal Circular Sectors
Minimum DIstance in Sector
Minimum Horizontal DIstance in Sector
Analysis of Openings (eg approaching intersection)
Obstacle Avoidance
Self-Centering and Alignment
Angle to Turn at Suitable Intersection
Angle to Avoid Obstacle
Angle to Drive Straight
Maximum Angle
Steering Angle

Figure 4.5: Calculating Steering Angle

4.4 PERFORMANCE

The algorithm in its current state is able to consistently drive around the environment it was designed for at speeds far surpassing than UWAs other wheeled robots are capable of, despite using very little sensory information and lacking any feedback mechanism. Timing the robot during autonomous experiments around the department showed an average speed of 15km/h was possible. This converts to a scale speed of 150km/h.

4.5 SUMMARY

The autonomous ability of the robot with the limited sensory data surprised us. Maximum speed is currently restricted by the Laser Scanner refresh rate of 10Hz and reliable range of ~4.5m. The distance travelled between scans and the braking distance both increase with speed and increasing it further would limit our ability to safely avoid obstacles.

The program as a whole is still very much a work in progress and the code needs significant refinement, if for no reason other than readability. The self-driving algorithm is very specific to office-like environments and could be expanded further if desired. That was beyond the scope of this project, which successfully demonstrated the self-navigating ability of the robot using only a Laser Scanner and can be used for comparison in future research.

5 SLAM

One of the objectives of this project was to develop a platform enabling future research into autonomous and SLAM vehicles. Specifically, a platform for testing the self-navigating capabilities with sensors alternate to LIDAR (ie cameras) was desired. This is out of the scope of this research, but we expect Visual(camera)-based SLAM algorithms will be investigated as a solution.

To ensure the platform is capable of the expected use-case, or at least providing a baseline for comparison, we wanted to ensure it was at least capable of performing more traditional laser-distance based SLAM iterations at a rate enabling an implementation in real-time.[2]

5.1 SLAM ALGORITHM

The SLAM algorithm we tested is commonly known as TinySLAM or tinySLAM and was developed Bruno Steux and Oussama El Hamzaoui of the MINES ParisTech Center for Robotics.[21]

Despite being designed to be less computationally heavy than some of the more full featured algorithms and implementations available, it has been demonstrated at least once that this is not the case despite producing significantly less accurate results. [22]

The focus of this research was not optimal algorithms or optimising SLAM performance, we simply wanted to confirm the systems ability to generate a reasonable map of its environment at a rate above the incoming data rate, something TinySLAM would be more than capable of. TinySLAM has also been implemented in an Open-Source and stand-alone repository named “BreezySLAM” courtesy of Simom D. Levy of Washington & Lee University. Ease of use and the algorithms ability to function without Odometry data were key in its selecton for testing[23]

5.2 INSTALLATION AND INTEGRATION

Thanks to Levy’s BreezySLAM package, installation was quite straight forward and required only a minor addition to the provided instructions to ensure the newly installed library was properly configured for execution through the use of ldconfig [24]

After installing and running the sample program however, the documentation fell a bit short and did not provide much guidance on extending the program for alternate use cases. The source code was however well designed and commented, enabling us to quickly adapt the sample program to work with our generated data without needing to fully understand or rewrite the backend. The changes are summarised below.

5.2.1 Scan Data

The original sample made use of data collected by the TinySLAM authors, which was formatted to include their robot’s wheel encoder data (Odometry). Our robot does not provide odometry data, and due to slightly ambiguous code in the function parsing the logged data, meant we rewrote the code for parsing the incoming log to match perfectly with our logging function in the navigation code, without looking for Odometry data.

5.2.2 Derived Classes

The package employs base and derivative classes, the ones of interest shown in Figure 5.1. While this is useful when the software is likely to be used across multiple platforms, for our brief testing it was excessive and we chose to remove a layer from the Laser class to enable greater parameter customisation as it was required.

The “WheeledRobot” Class and derivative only provided a convenient way to generate velocities from the measured Odometry data and as such were not required for our use and were removed.

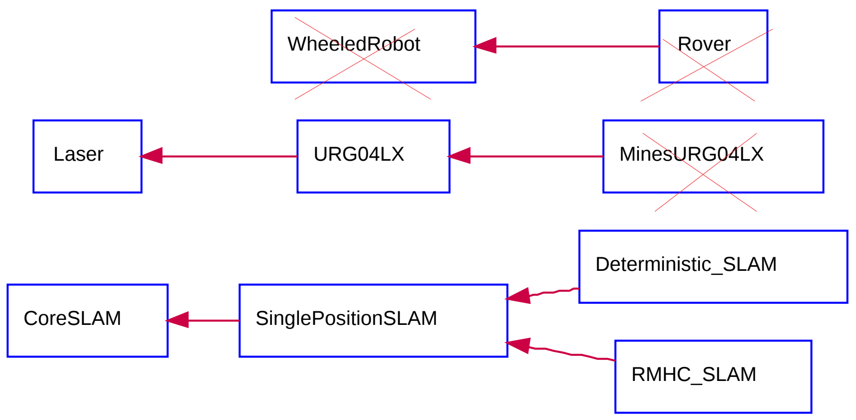


Figure 5.1: BreezySLAM class diagram showing the derived classes (right) pointing towards their more general base classes, and unused classes with a cross

5.2.3 Useful Parameters

The remaining base classes and derivatives in BreezySLAM contain the SLAM algorithm parameters, which influence the accuracy of the generated map, as well as what we've named environment parameters; the dimensions of the environment being stored as well as parameters defining the Laser Scanner data (which must match the scanner parameters when the log was taken).

5.3 EXPERIMENTAL METHOD

The robot was driven manually around the EECE 3rd and 4th floors whilst logging the incoming LIDAR. Due to some hard-coded parameters in our autonomous routine, we were unfortunately unable to log an autonous drive, however the difference in results would likely be minimal.

BreezySLAM was then configured for the 2 environments; the 3rd floor using a map size of 16×16m and the 4th floor 32×32m (based on the area covered by the robot), before executing the SLAM algorithm with a timer attached to the process.

5.4 SLAM TESTING RESULTS

5.4.1 Map Accuracy

The initial tests were not exactly a success, with the algorithm generating twisted and warped maps (Figure 5.2).

As configuring the algorithm for high accuracy was never priority, it received little time or attention and parameters were adjusted almost randomly rather than due to any real understanding of their impact. Regardless, BreezySLAM produced a relatively accurate map of the EECE 3rd floor.

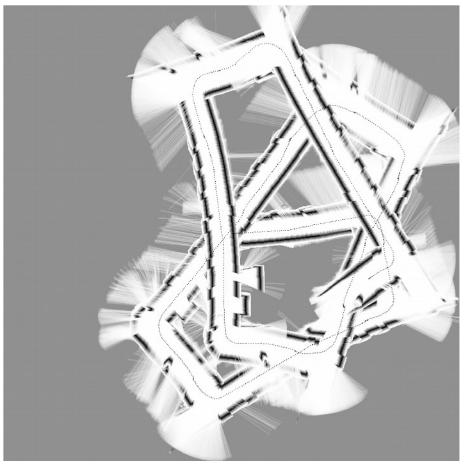


Figure 5.2: Initial SLAM Test

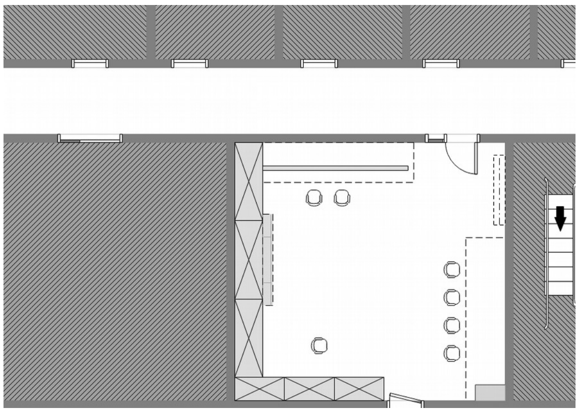


Figure 5.3: Floor Plan, 3rd Floor EECE

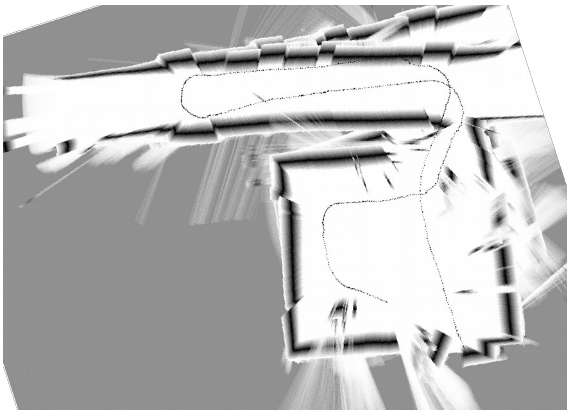


Figure 5.4: SLAM Generated Map, 3rd Floor EECE

Figure 5.3 above shows the actual floor plan of the area of the EECE 3rd floor that the roboit drove to generate the map shown in Figure 5.4. The Floor Plan includes various object the Laser Scanner sees that are evident in the generated map such as chair legs and cupboards. The dotted line through the generated map shows the position the SLAM algorithm believed the robot was at when each Laser Scan was recorded.

We also attempted to map the 4th floor of the EECE building, the same area we tested the self-driving algorithm. This a larger area, with few unique features for the algorithm to make use of.

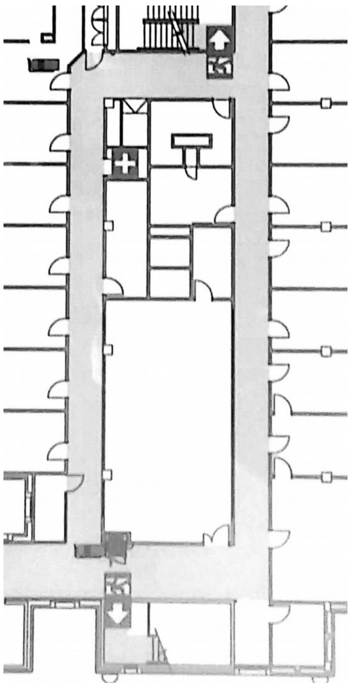


Figure 5.5: Floor Plan, 4th Floor EECE

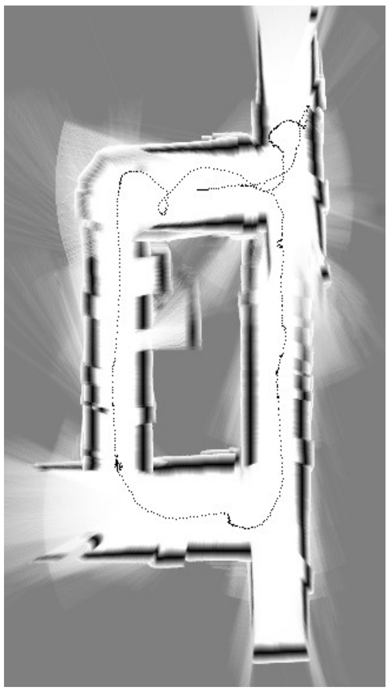
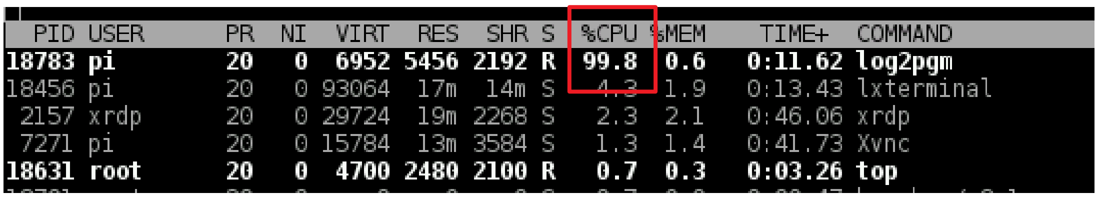


Figure 5.6: SLAM Generated Map, 4th Floor EECE

Figure 5.5 shows the actual floor plan the generated map in Figure 5.6 represents. The algorithm tracked the path of the robot quite accurately. A random path was manually driven after a loop was driven and is represented in the top right of 5.6. A pot plant in the lower left is also clearly visible.

The algorithm did not accurately determine the length of the hallways and produced a much smaller map than its actual size

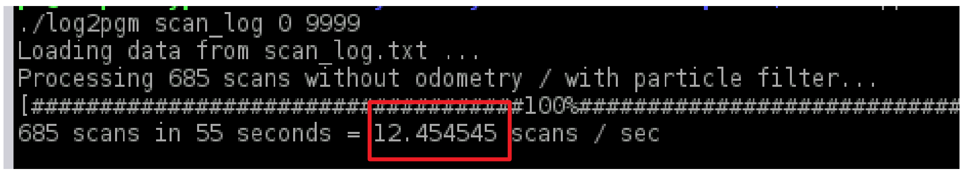
5.4.2 CPU Load and Processing Time



```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
18783 pi        20   0  6952 5456 2192 R 99.8  0.6   0:11.62 log2pgm
18456 pi        20   0 93064  17m  14m S  4.3  1.9   0:13.43 lxterminal
 2157 xrdp      20   0 29724  19m 2268 S  2.3  2.1   0:46.06 xrdp
 7271 pi        20   0 15784  13m 3584 S  1.3  1.4   0:41.73 Xvnc
18631 root      20   0  4700 2480 2100 R  0.7  0.3   0:03.26 top
```

Figure 5.7: CPU Usage During SLAM (PID 18783)

The SLAM algorithm utilised 100% of 1 of the 4 CPU cores on the Pi 3 (Figure 5.7) and achieved processing rates above the 10Hz that the URG04LX operates at when generating the previous maps (Figure 5.8).



```
./log2pgm scan_log 0 9999
Loading data from scan_log.txt ...
Processing 685 scans without odometry / with particle filter...
[##################################################100%################################
685 scans in 55 seconds = 12.454545 scans / sec
```

Figure 5.8: SLAM Processing Rate

5.5 DISCUSSION

An extensive analysis of results was not performed, however visually analysing the SLAM generated maps does indicate the vehicle in its current configuration is capable of SLAM in real-time and could therefore be used for comparison with visual SLAM algorithms, although a different or modified SLAM algorithm would be ideal.

The results also confirmed shortcomings of the tinySLAM algorithm identified in other SLAM research, particularly that tinySLAM suffers from poor loop closure logic, which can be seen in the upper right of both Figures 5.4 and 5.6. [22]

The original tinySLAM creators are among a few researchers who have proposed solutions that could potentially replace the algorithm used by BreezySLAM. [25], [26]

The result of the test on the 4th floor show the limitation of SLAM without odometry data in featureless environments. The hallways are many times longer than the maximum range of the Laser Scanner, and the SLAM algorithm struggles to find reference points to determine relative motion. This can be corrected for by tuning the algorithm, to a degree.

The CPU usage recorded during the SLAM algorithm lead us to believe the algorithm could be implemented in real time. While the process did require 100% of one of the cores, the remaining 3 cores should be sufficient to enable control of the vehicle

6 CONCLUSION AND FUTURE WORK

6.1 FUTURE WORK

The platform was developed to enable research into alternatives to the expensive 2D-laser scanner and that remains a potential for future work. Attempting to achieve similar autonomous driving capabilities using only a camera is one such topic of research.

The platform would also serve as a good tool for developing SLAM algorithms.

The autonomous driving currently implemented is not a very intelligent design and could be significantly improved. An implementation to enable quickly adapting to new environments either automatically or through manual operation like teach-and-repeat would be particularly useful. Integration with SLAM for autonomous exploration would be a good use of the platform.

The robot is well suited to outdoors and harsh terrain, so future researchers should not limit themselves to office environments.

An accurate model of the car and it’s behaviour would be useful for any future work.

6.2 CONCLUSION

Experiments conducted using the robot platform developed in this paper have succeeded in demonstrating the capabilities of the new vehicle utilising only a 2D Laser Scanner for information about it’s environment and relative position. An autonomous driving routine has been developed capable navigating an office environment at speeds far greater than capable by anything in the departments current fleet of wheeled robots. We have also shown that SLAM is possible on the new platform and the results indicate SLAM in real-time is also viable.

The new platform and it’s components have been implemented to be compatible with the RoBIOS interface used in the practical components of robotics coursework. Some of these features are being integrated into the RoBIOS interface to provide new functionality, such as a simple API for accessing Laser Scanner Data and PWM control with the GPIO pins. The additional features can therefore be integrated into the coursework and equipment enabling hands on experience with a broader range of equipment. The cars high speed, unique appearance, and simple graphical interface would, in the authors opinion, be a great tool for generating interest among potential students.

The work described in this paper has resulted in a platform that is ready for future research into the use of low-cost sensors as a replacement for expensive laser-ranging technologies, and improving autonomous robotic navigation. It has also contributed to enriching the learning experience of Robotics students at the University of Western Australia through the development and improvement of tools used in their education.

[1]. O. El Hamzaoui, ‘Simultaneous Localization and Mapping for a mobile robot with a laser scanner: CoreSLAM’, Sep. 2012.

[2].T. Bräunl, ‘Introduction’, in *Embedded Robotics*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[3]................‘Localize with a Hokuyo laser range finder - Génération Robots’. [Online]. Available: https://www.generationrobots.com/en/content/52-localize-with-a-hokuyo-laser-range-finder. [Accessed: 26-Oct-2017].

[4]…Traxxas, ‘Stampede: 1/10 Scale Monster Truck with TQ 2.4GHz radio system’, 23-Oct-2014. [Online]. Available: https://traxxas.com/products/models/electric/36054-1stampede. [Accessed: 28-Oct-2017].

[5].....‘Scanning Rangefinder Distance Data Output/URG-04LX-UG01 Product Details | HOKUYO AUTOMATIC CO., LTD.’ [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=166. [Accessed: 26-Oct-2017].

[6].............................‘Wireless Gamepad F710 - Logitech Support’. [Online]. Available: http://support.logitech.com/en_us/product/wireless-gamepad-f710/specs. [Accessed: 01-Nov-2017].

[7]...‘Raspberry Pi Documentation’. [Online]. Available: https://www.raspberrypi.org/documentation/. [Accessed: 29-Oct-2017].

[8]................‘Raspberry Pi Downloads - Software for the Raspberry Pi’, *Raspberry Pi*. .

[9]..‘RPi Distributions - eLinux.org’. [Online]. Available: https://elinux.org/RPi_Distributions. [Accessed: 31-Oct-2017].

[10].................................Raspberry Pi Foundation, ‘Release Notes’. [Online]. Available: http://downloads.raspberrypi.org/raspbian/release_notes.txt. [Accessed: 31-Oct-2017].

[11].........................‘Robot Operating System | Erle Robotics Docs’. [Online]. Available: http://docs.erlerobotics.com/robot_operating_system. [Accessed: 31-Oct-2017].

[12]..‘Application: icp-slam – MRPT’. .

[13]‘Raspberry Pi SD-Card Image| Open Source Humanoid Robot’, *German-Robot.com | Open Source Humanoid Robot*, 26-May-2016. .

[14]. .richardghirst, *Contribute to PiBits development by creating an account on GitHub*. 2017.

[15]..‘URG Network’, *SourceForge*. [Online]. Available: https://sourceforge.net/projects/urgnetwork/. [Accessed: 31-Oct-2017].

[16]. .S. D. Levy, *BreezyLidar: Simple, efficient, Lidar access in Python and C++*. 2017.

[17]................‘urg_node - ROS Wiki’. [Online]. Available: http://wiki.ros.org/urg_node. [Accessed: 31-Oct-2017].

[18]..................‘mrpt::hwdrivers::CHokuyoURG Class Reference’. [Online]. Available: http://mrpt.ual.es/reference/devel/classmrpt_1_1hwdrivers_1_1_c_hokuyo_u_r_g.html. [Accessed: 31-Oct-2017].

[19]....................................‘URG Network / Wiki / urg_viewer_en’. [Online]. Available: https://sourceforge.net/p/urgnetwork/wiki/urg_viewer_en/. [Accessed: 01-Nov-2017].

[20]......................‘evtest - Input device event monitor - Linux Man Pages (1)’. [Online]. Available: //www.systutorials.com/docs/linux/man/docs/linux/man/1-evtest/. [Accessed: 31-Oct-2017].

[21].......B. Steux and O. E. Hamzaoui, ‘tinySLAM: A SLAM algorithm in less than 200 lines C-language program’, in *2010 11th International Conference on Control Automation Robotics Vision*, 2010, pp. 1975–1979.

[22].J. M. Santos, D. Portugal, and R. P. Rocha, ‘An evaluation of 2D SLAM techniques available in robot operating system’, in *Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*, 2013, pp. 1–6.

[23]. .S. D. Levy, *BreezySLAM: Simple, efficient, open-source package for Simultaneous Localization and Mapping in Python, Matlab, Java, and C++*. 2017.

[24]..‘ldconfig(8) - Linux man page’. [Online]. Available: https://linux.die.net/man/8/ldconfig. [Accessed: 29-Oct-2017].

[25].....O. E. Hamzaoui and B. Steux, ‘SLAM algorithm with parallel localization loops: TinySLAM 1.1’, in *2011 IEEE International Conference on Automation and Logistics (ICAL)*, 2011, pp. 137–142.

[26]….A. Huletski, D. Kartashov, and K. Krinkin, ‘TinySLAM improvements for indoor navigation’, in *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2016 IEEE International Conference on*, 2016, pp. 493–498.