HOLISTIC DESIGN FOR A SIMPLISTIC AUTONOMOUS VEHICLE

Jacob Coles

Bachelor of Engineering (Honours)
with a major in Mechatronics Engineering

Department of Engineering
Macquarie University

November 6, 2017

Supervisor: Dr. David Inglis

ACKNOWLEDGEMENTS

I would like to thank the teaching staff at Macquarie University for helping me in pursuing engineering. I would also like to thank the members of the 2016 Macquarie University National Instruments Autonomous Robotics Competition team #MQMechatronics for their hard work which was built upon this year. I would like to thank Jesus for giving me purpose in life, and my parents for all they have given for me.

STATEMENT OF CANDIDATE

I, Jacob Coles, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Mechatronics Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any academic institution.

Student's Name: Jacob Coles

Student's Signature: Jacob Coles (electronic)

Date: November 6, 2017

ABSTRACT

Autonomous vehicle control systems are becoming more and more reliable, efficient and safe than human drivers as time elapses. This is a consequence of improved, low power, and high speed embedded processors, advanced, high-level development platforms, open source maps and software, high quality, low cost sensors, and satellite navigation. This thesis features an investigation of some simplistic control methods used for a miniature vehicle needing to navigate a physical grid with walls in a known, mapped area. The purpose for this thesis is to demonstrate and explain some of the elementary concepts of autonomous control on the most basic level of logic and control. While the vehicle presented is simple, its design would suit many applications, such as factory floor transportation, or a mail delivery system in a building. Consequently, this thesis will be focused on a simplistic, low cost implementation of such a type of system. It will not, however, focus in detail on modern technologies and advanced control systems, for example, autonomous road driving. The vehicle developed for this project was designed as an entry in the 2017 NI ARC (National Instruments Autonomous Robotics Competition) competition.

Contents

Acknowledgements

Abstract

Table of Contents

1 Introduction
- **1.1 Current State of Driverless Technology**
- **1.2 Research Question**
- **1.3 Research and Design Goals**

2 Background and Research
- **2.1 LabView and LabView Programming**
  - **2.1.1 LabView Programming Fundamentals**
  - **2.1.2 Interfacing with Devices using the MyRio and FPGA**
- **2.2 Motors and Motor Control**
  - **2.2.1 Induction Motors**
  - **2.2.2 Permanent Magnet Brushless DC Motors (PMBLDC)**
  - **2.2.3 Permanent Magnet DC Motors**
- **2.3 Sensors**
  - **2.3.1 Ultrasonics**
  - **2.3.2 LIDAR**
- **2.4 Wheel Types and Control Methods**
  - **2.4.1 Tank Drive/Skid Steering**
  - **2.4.2 Ackermann Drive**
  - **2.4.3 Tricycle Drive**
  - **2.4.4 Mecanum Drive (and Localisation Characteristics)**

3 Design and Implementation
- **3.1 Hardware**
  - **3.1.1 Motors**

3.1.2 Wheel Type and Localisation
3.1.3 Encoders
3.1.4 Frame/Chassis
3.1.5 Batteries and Buck Converter
3.1.6 Ultrasonic Sensors
3.1.7 FPGA and Hardware Interface

3.2 Software
3.2.1 FPGA and Interfacing with the Real Time System (RTS)
3.2.2 Basic Control System
3.2.3 Rotary Encoders
3.2.4 Position Calculation
3.2.5 Going to a Set Position
3.2.6 Going to Set-Points Sequencing
3.2.7 Ultrasonic Sensors
3.2.8 Position Corrections
3.2.9 Angle Corrections
3.2.10 Path Selection Algorithm
3.2.11 WiFi Server Retrieval

4 Conclusions
5.1 Discussion
5.2 Conclusions
5.3 Future Work

References

Chapter 1

Introduction

The project detailed in this document, will include the process, design elements, challenges and progress made in the process of designing an autonomous robotics system which can navigate through a walled map in real time. The design limitations, and map were set by National Instruments in 2017 for the National Instruments Autonomous Robotics Competition (NI ARC), which was completed on the 5th September 2017. The competition requires the development of a small, wheeled robotic vehicle that can navigate a specific, walled area with obstacles, and then return to its starting point. The mapped area consists of an enclosed maze-like environment, with obstacles in certain areas, and the obstacles' position being broadcasted over WiFi to the vehicle. As for the hardware configuration, the MyRio platform, containing both an ARM based microprocessor and an FPGA (Field Programmable Gate Array), serviced the logic of the system, and was programmed using LabView. It was used to perform I/O logic for the systems motor drivers, rotary encoders and ultrasonic sensors, as well as logic and navigational algorithms.

The hardware build consisted of the aforementioned MyRio microcontroller, ultrasonic sensors, brushed DC motors, rotary encoders, mecanum wheels, Li-Ion cells as well as other construction materials, the majority of which was reused from a previous entry (2016) in the NI ARC competition.

The majority of design in this project is consisted of software. Localisation algorithms use both the rotary encoders and ultrasonic sensors, the latter of which detects walls. Since the system is situated within a known map, the system is able to calculate how far it has strayed from its expected position and rotation, with respect to the map, and in response may correct its position and rotation; this is an implementation of position correction. Navigation is consisted of a sequence of steps. Firstly a sequence of 'set-points' must be determined. A set-point is, here, an X-Y coordinate defined by a coordinate system within the aforementioned map, determining the location to where the system will drive to next. When the software determines a path for the system to travel along, it will store the sequence of these set-points in a set of arrays. Locomotion was performed by four mecanum wheels attached to brushed DC motors.

The implementation of hardware and software in this report, while effective, could be shown to have viable alternatives, however the intent of this report is to demonstrate an effective and simplistic method of control for a vehicle travelling within a well known and mapped environment, however in future, map generation may be possible.

This will be elaborated qualitatively in the future work section of this document. Research which was not used, but was considered, in the early stages of this project will be elaborated, as well as the reasons why some methods were not used, especially with respect to ease of implementation and cost.

1.2 Research Question

There are a wide variety of components, embedded processors and software packages, which would lend themselves useful to the application of designing an autonomous robotic system. These options need to be investigated, with respect to the requirements of the NI ARC competition, for which the vehicle is being designed. The requirements of the competition, are that the vehicle must begin at a specified place in a given map, proceed to move to various set-points (while being loaded with items), continue through a maze-like area with obstacles obscuring the robot's path, for which it may have to generate a new path, then stopping at other setpoints where the aforementioned loaded items would be removed, and then proceeding back to the initial starting point. Thus in summary, a simple, low cost and easy to design vehicle must be designed, to allow the transportation of items within a known map, with the ability to avoid obstacles.

1.3 Research and Design Goals

The goal is this project is to design a wheeled robotic vehicle which is able to navigate through a walled course without any direct human interaction, by measuring information about its environment using sensors, and use appropriate control techniques and algorithms, such that it is able to do it, and return to its starting place [1]. This requires designing a vehicle, which can utilise multiple sensors, and use a software model, to understand its environment and employ appropriate control techniques, to navigate to to a number of destinations. This vehicle was to be included in the NI ARC competition, so the design needed to use the National Instruments MyRio for onboard logic and control, as well as controlling motors, hardware and other external controllers. In addition, software needed to be developed, such that the system would be able to localise itself sufficiently well. Localising refers to being able to locate and identify the vehicle's placement it within a map, or some well-defined reference frame. This necessitated designing path prediction and planning algorithms, which will take into account the dynamic (changing) nature of the course. The challenge of path planning is simplified by the fact that the layout of the obstacles are given in advance over a wifi server, which can be read by the MyRio. The general layout of the course is given below in Fig. 1:

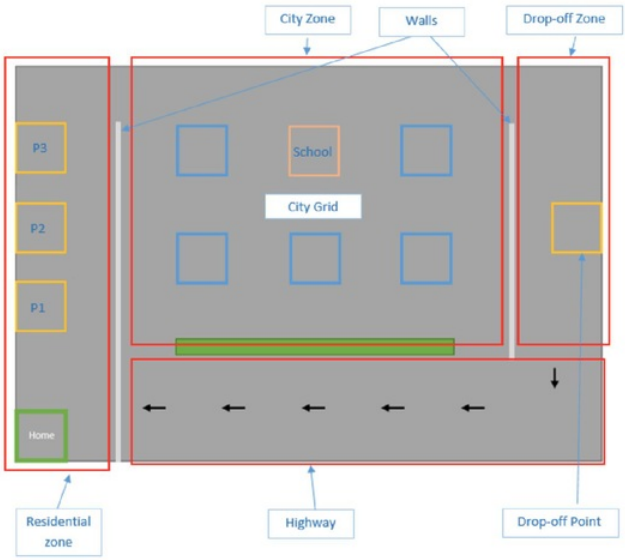


Fig. 1 [1]

The 'dynamic' nature of the course, refers to the fact that obstacles may be placed in the 'city grid region', so the vehicle needs to be able to alter its path to go around obstacles. The vehicle needs to be able to stop at the P1, P2 and P3 'parking' locations, continue through the city grid, stop at the 'drop-off point' at the right in yellow, continue through the highway, then return to 'home'. Fig. 2 demonstrates one possible path for the vehicle to travel in the map.

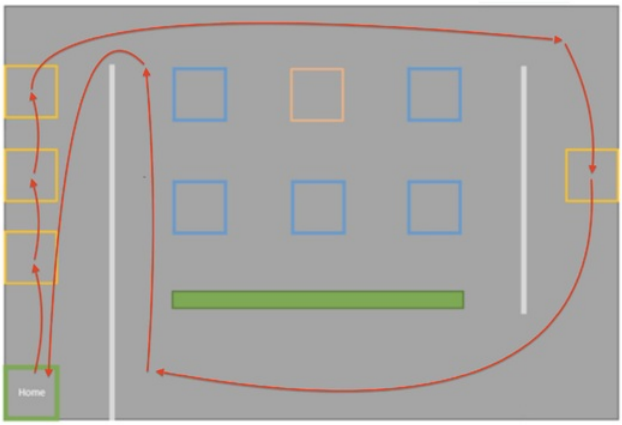


Fig. 2 [1]

Chapter 2

Background Research and Evaluation

As this project is with an extensive scope, there are a wide variety of hardware and software components which will need to be investigated. As such, research conducted in the following section will not be so detailed, but instead will primarily focus on the simple solutions to the design requirements. Various options for hardware components will be considered and evaluated in terms of their difficulty for implementation, especially with regard to control software. The software control methods themselves will also be evaluated in a limited capacity. Specific information on the design will be elaborated further in chapter 3 of this document.

2.1 Labview Programming

As the bulk of software and logic in this project was programmed in LabView, an understanding of the workings of this program is needed to fully grasp the design elements presented in chapter 3 of this document. This section may be skipped.

2.1.1 LabView Programming Fundamental

The programming language of LabView is the G programming language. Every program/routine written in LabView is known as a Virtual Instrument or 'VI'. These VIs consist of two 'layers' which are displayed in separate desktop windows. The first layer is the 'Front Panel', contains a display of all the variables being used in the program, as well as any other data or metrics that the programmer wishes to display there. This includes graphs or other I/O. The second layer is called the 'block diagram' and contains the actual code of the program. Here, every program/routine/VI's block diagram consists of function blocks, and 'wires' between them, which can can carry any specified data type between the blocks. An example of a simple labview program which adds two values is as follows:

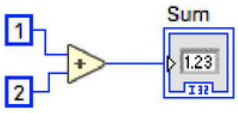


Fig. 3

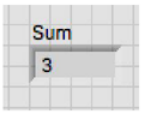


Fig. 4

The block diagram is shown on the left, while the front panel, (which displays results, outputs and user inputs is on the right).

LabView contains an extensive library of functions and blocks which can be used for a wide range of purposes.

Programs which need to run iteratively or continuously, can be done as such by using, what is known in LabView, as a 'structure'. These 'structures' provide similar functionality to while loops, for loops etc. A simple example of a program which uses a 'For Loop' follows:

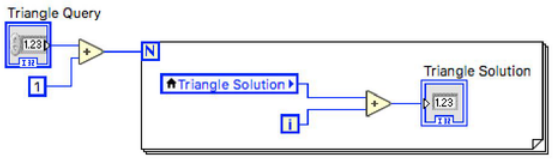


Fig. 5

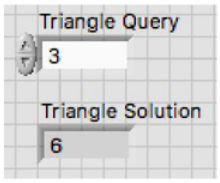
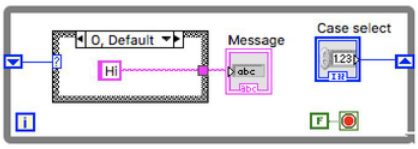


Fig. 6

Note that 'Triangle Query' and 'Triangle Solution' are arbitrary variable names. The loop structure is the large black box. The input to the 'N' terminal in the block diagram (left side of Fig. 5) is the number of times the loop will run. The two items with the label 'Triangle Solution' are actually the same variable, except one of them (the one with the home symbol), is a local variable, a duplicate variable. The small 'i' block in the for loop refers to the loop iteration number. When this program runs, it runs through the number of iterations wired into the 'N' terminal, performs the operations in the loop, and eventually completes. This particular program calculates triangle numbers.

Being able to use these kinds of structures, effectively, is a core aspect of programming in LabView. 'Case Structures', for example, are structures which can run one of many different snippets of code, depending on which piece of code is selected, through a 'case selector' value, which is wired into the case selector terminal. These case selector values are arbitrary and of any data type, so the only requirement for a case to run is for the case selector input matches the name (or numeric value) of the name of the desired case. For example, a case structure could be an 'if' statement, if the data-type for the input to the case structure is boolean. However, it must be reiterated, that these case structures enable any data-type to be used as the case selector, not simply a boolean value, such that there can be theoretically infinite cases. Following is an example for a set of cases, where the input for the case structure is an integer. The case structure itself is encapsulated in a while loop, so that it runs continuously.



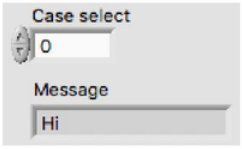


Fig. 7

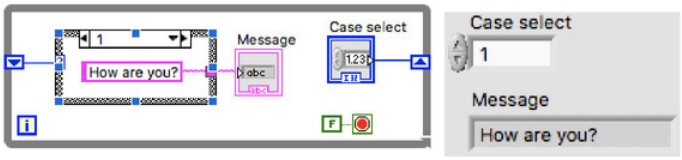


Fig. 8

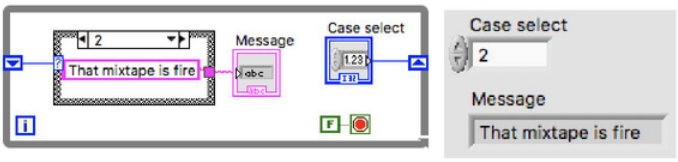


Fig. 9

For any of three different values of the 'Case select' 'control' variable, a different message is shown, by selecting one of the cases, each of which contains some string value, as demonstrated by the images above. Here, the 'Case Select' variable has not been directly wired into the case structure (although it could have been) but it has been wired into something called a 'shift register' (which is not technically the usual kind of shift register). The 'shift register' depicted, simply takes the value wired into the blue block on the right hand side of the while loop, and uses it in the next loop iteration, through the terminal on the left hand side of the while loop. This style of programming is useful in more complex situations, when logic residing within case structures themselves can determine the next case to be run, in the next iteration of the while loop. Note that the red coloured 'stop while-loop' control at the bottom right of the while loop has been wired with a false boolean constant, such that it will run forever.

2.1.2 Interfacing with Devices with the MyRio and FPGA

The MyRio designed by National Instruments is a microcontroller, which is able to run code written in LabView [2]. The MyRio contains both a real-time microprocessor (dual core ARM Cortex-A9 processor) as well as an NI Xilinx FPGA. The real-time ARM processor runs the 'main' code, and does the majority of data processing, just as a typical microcontroller or processor may. The FPGA is a co-processor which must be used as a passthrough for all input and output information (of the physical pins/connectors). This real-time system is able to interface with the FPGA through a special set of functions available in the RTS coding environment. The code which runs on the FPGA is also written in LabView using the same visual G language, however there are a simpler subset of functions available.

The purpose of the FPGA is to enhance interfacing with devices at a low-level, which is beneficial in cases where fast response times are important. Following is a simple block diagram written in the real time environment (RTS), which sends data to the FPGA (Fig. 10):

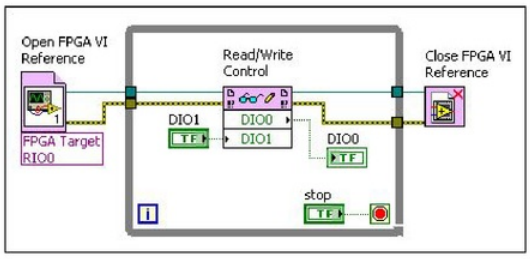


Fig. 10 [10]

In Fig. 10 above, there are two boolean variables in the FPGA accessed. DIO1 is being written to, and DIO0 is being read from. The actual values of these variables would be visible in the front panel (the block diagram is shown here). The black box is a while loop, indicating that these variables are continually accessed. The 'Open FPGA VI Reference' is configured to refer to a particular VI/code-file (built to run on the FPGA), telling the real-time system what code and variables it should be accessing. The 'Close FPGA VI Reference' is run in order to complete the read/write operation. This is the most common use case for the FPGA.

As previously mentioned, the FPGA runs separate code from the RTS. An important FPGA function is the read/write block, which allows a physical pin to be accessed:

Fig. 11 [3]

In the above example, the left block reads 'Digital Input 1' and outputs the value to 'Digital Output 1'. This program would pass on a value from one input, straight to an output.

A diagram demonstrating this code, in terms of the relationship between the physical DIO pins, and the code of both the FPGA and the RTS is shown in Fig. 12:

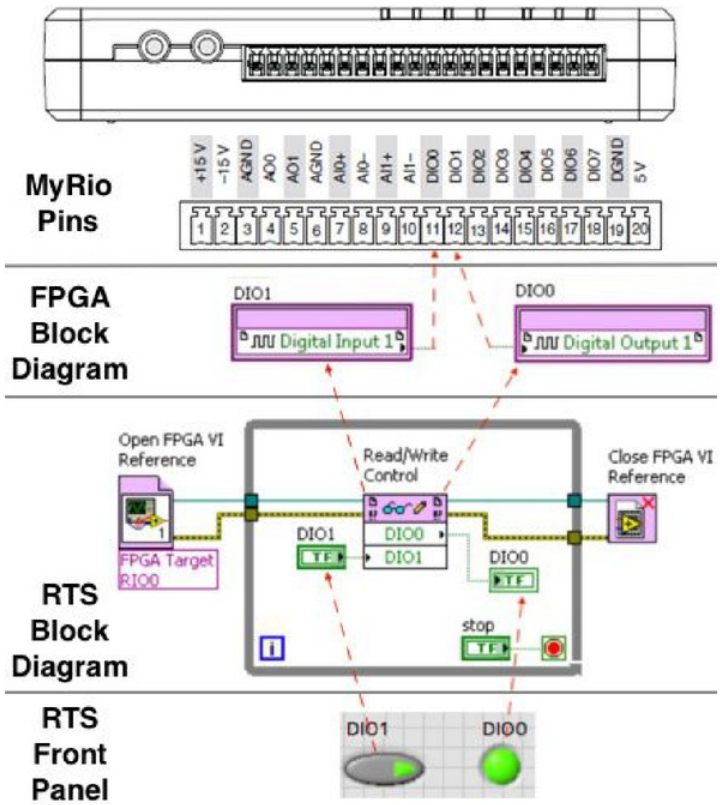


Fig. 12

2.2 Motors and Motor Control

There are a few available motor types suitable for this project. The two motor types determined to be suitable for this project were either the permanent magnet brushed DC motor, or the permanent magnet brushless DC motor (PMBLDC) [4]. While AC induction motors were considered, they were determined not suitable, as they have slow response at low speed, and only generate torque at zero speed once a magnetic field has been generated in the rotor. Both types of DC motors under consideration respond with approximately high torque at approximately zero speed [5]. Out of the varieties of DC motors, the brushed version was chosen as it was available on-hand, but other motor types were considered in the case that they proved to provide significant improvement. The comparison will be made in depth as follows.

2.2.1 AC Induction Motors

AC induction motors do not utilise permanent magnets, which makes them useful for large scale and high power applications, where the impracticality of permanent magnets outweigh the relatively high power density of DC motors [5]. They have the best torque characteristics at synchronous speed, when the rotor is highly magnetised, and this is when it runs at maximum efficiency. Position control is usually difficult to implement well, but speed control is possible. The

(relatively) high inertial mass of the rotor of an AC induction motor gives them naturally good stability characteristics at high speed [6], however at zero speed the initial lack of a magnetic field in the rotor means that there is a delay in the response of the motor. Additionally, the high inertial mass of the rotor causes relatively low acceleration [6]. It is assumed that the vehicle will be starting and stopping at regular intervals, so these drawbacks are sufficient to not consider their use when compared with DC motors.

2.2.2 Permanent Magnet Brushless DC Motors (PMBLDC)

The PMBLDC motors are multi-phase motors, where each phase (coil) must be controlled with respect to the rotor angle, and as such feedback control methods need to be implemented. Benefits of the PMBLDC motor include their high starting torque, almost constant maximum torque, as well as a high power to weight ratio, compared with other motor types (especially when compared with induction motors). They also have relatively high power efficiency and longevity, and they do not have brushes which wear and generate heat, like brushed DC motors do [7].

The drawbacks include a high purchase cost, and the requirement of a complex control system. It is most common to use a three phase system, however other phase numbers may be used. The power of these phases must be synchronised with the position of the rotor, thus the position of the rotor must be tracked using hall effect sensors (considered the simplest method to design) or by current sense feedback (on each of the phases), by utilising differential amplifiers, one across each phase [8]. The general structure of a PMBLDC motor is as follows, including winding design as well as discretised control inputs of each phase on a graph:

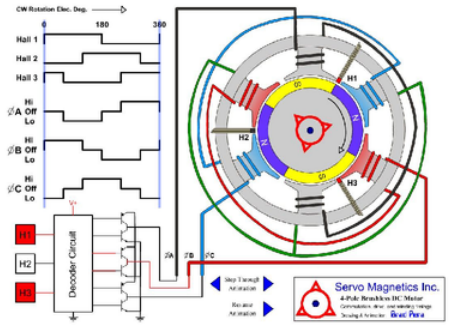


Fig. 13 [9]

A mathematical transformation which turns a (2 dimensional) phasor/tensor into a (3 dimensional) scalar vector called the direct-quadrature-zero transformation can also be used instead, for smoother control.

2.2.3 Brushed Permanent Magnet DC Motors

The primary benefit of the permanent magnet DC motors are their simplicity, control requires changing the voltage of the power source. Typically the speed of the motor can be controlled by a PWM (pulse width modulation) signal, while its direction can be varied by changing the direction of power (or current). The control logic for changing the direction of current will depend on the design of the motor controller used.

Motor control, is typically done by using a motor controller with a single H-bridge, utilising power transistors (either MOSFETs or IGBTs). Control input types may vary, depending on the type of motor controller, and could include one of two primary control methods [10]. The most common type uses two pins, one of which is a boolean pin for direction selection, with the other pin is the voltage control, a PWM pin that decides the proportional speed output of the motor.

The brushed DC motor is not suitable at high speed due to bad contact of the brushes at high speed, and excessive wear. However, in this project it is unlikely that the system will be driven at that kind of speed [11].

2.3 Sensors

Sensors which map a physical environment are essential when designing a robotic control system. It is insufficient to rely only on sensors which track a change in position (for example rotary encoders) as errors are likely to build up. An absolute measure of the robot with respect to a physical environment is necessary to stabilise a robotic system.

2.3.1 LIDAR

LIDAR (light detection and ranging), is a type of range finding sensor that uses pulses of light on or near the visual spectrum to determine physical distance along some physical axis where the sensor is orientated. It utilises multiple light pulses, phase shifts and logic to do this [12]. For the intents and purposes of this project, there were two suitable forms of LIDAR sensors available. The most simplistic type is the 'stationary' ultrasonic sensor, which can only detect a distance measurement on a single axis on which it is orientated. There is also a spinning LIDAR (the common kind), which uses the same type of sensor, but is attached to a motor, which can track its rotation, so that it can correlate a distance measurement to a particular angle. This allows sufficient data to generate 2D plane of distance measurements.

The simplistic (non spinning) variant would be easy to implement, however the cheapest available sensor was too costly, and with functionality similar to an ultrasonic sensor (detailed in the next section). A spinning LIDAR was available on hand, however after some research, it was determined that implementing a functional design would have been too time consuming and risky. Simultaneous Localisation and Mapping (SLAM) is a technique applied to using these sensors to generate a map in real time, and use it to localise the sensor within that map. These techniques are very complex, and not suitable for the purposes of this project in the given timeframe. Using some kind of data filtering, it may be possible to extrapolate useful data from LIDAR, however ultrasonics, can provide this kind of information much more easily.

2.3.1 Ultrasonics

Ultrasonic sensors come in a variety of forms, and are used for distance measurements. An ultrasonic transducer itself is able to both generate a high frequency sound and detect a sound of the same frequency. [13] Thus, a transducer can be used as both transmitter and receiver. The basic ultrasonic sensor design consists of an ultrasonic transducer and a resonator [14]. These transducers are essentially a speaker/microphone combination, and resonators generate a time varying (AC) voltage, by either using a quartz crystal, or simply a resonant RLC circuit. The basis of operation consists of three things. Firstly is the emission of a high frequency sound through the transducer, secondly is a measurement of the return of the sound through the same transducer, and finally the

time between transmission and retrieval must be measured. The time between transmission and retrieval corresponds to the time taken for the sound to reflect off some object and return, and if the speed of sound in the medium under measurement is known (~340.3m/s in air at sea level), an approximate distance measurement can be made.

Complex versions of ultrasonic sensors (also known as ultrasound sensors) may consist of many ultrasonic transducers (or separate transmitter/receivers), as well as complex measurement and interpolation systems. When multiple transducers or microphones are used, it becomes possible to interpret an area's spatial geometry, by measuring the time-varying echoed sound. The time-intensity profile of sound echos at different spatial locations (through the separated transducers or 'microphones') makes this possible. These are the type often used in medical imagery and are typically quite costly.

A basic ultrasonic sensor, for example the HC-SR04, which is widely available and mass produced, allows for distance measurement along an axis perpendicular to the sensor. The HC-SR04 sensor in particular contains an onboard circuit which performs the majority of signal conditioning. [15]

2.4 Wheel Types and Localisation

A relatively fast summary of the wheel types and control methods primarily under consideration will be detailed [16] [17]. The wheels ultimately used, the mecanum wheels, were chosen as they were on hand, and satisfied the criteria for use.

2.4.1 Tank Drive/Skid Steering

Control using tank drive consists of any number of fixed wheels, and steering performed by changing the rate of wheel turning on either side. In this case, a simple four wheel configuration would be considered. Benefits of this kind of control method (especially in a standard four wheel configuration) are that it provides good stability, as well as an easy to implement control method. A challenge with this method, is that an inconsistency in wheel grip can cause accidental rotation.

2.4.2 Ackermann Drive

There are two main (common) varieties of wheel/control system designs utilising front wheel turning. One, called the Ackermann drive utilises four wheels, of which the front two wheels are allowed to turn. This is the type of steering used by most cars. This design for turning inherently generates a characteristic which is the 'turning radius', which describes the point about which the system is turning, on the surface of movement. If both front turning wheels are turned by the same angle, both will experience slip or loss of traction, as they will not both be influencing the system to be turning about the same exact point/turning radius as one another. In order to compensate, a control method must be developed to enable the wheels to rotate in appropriate proportion to one-another, such that the turning point/radius is equivalent for both wheels. This is commonly done by using an appropriately designed turning rack [18], which can be seen in the following diagram:

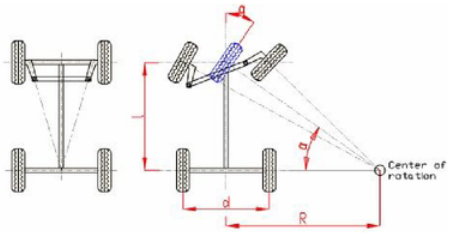


Fig. 14 [19]

Thus, this control method has minimal slip, when compared to the aforementioned tank drive. Additionally, benefits of this type of control method include stability at relatively high speeds, as well as stability in terms of falling over, especially when the center of gravity is kept low. Disadvantages include mechanical complexity, and control system complexity. If the front wheels are to be powered, complicated mechanical power transmission systems, which will freely allow the wheels to turn while transmitting power, become necessary. If such a system remained strictly rear wheel drive, there remains the mechanical complexity of designing a rack-turning system which would allow each wheel to turn at the appropriate angle in order to minimise slip. In addition, there is complexity in designing a control system, such that an autonomous vehicle would be able to navigate to a desired location, which is due to the fact that the only controls for the vehicle are forward/reverse movement and turning. These are the same complexities for the design of a control system, which has been previously mentioned in the previous sub-chapter about Tank Drive/Skid Steering.

2.4.3 Tricycle Drive

Tricycle drive is something of a three-wheeled variant of the Ackermann Drive. As there is only one front turning wheel, the design of a turning mechanism is greatly simplified, such that the turning angle is simply the angle at which the front wheel is placed. Otherwise, the other benefits, drawbacks and control characteristics parallel the Ackermann drive, with the exception of stability. A vehicle using tricycle drive has worse stability at speed compared with the Ackermann drive, and would need to have its weight distributed appropriately, otherwise there would be a greater likelihood of falling during a turn maneuver.

2.4.4 Mecanum Drive (and Localisation Characteristics)

A system utilising mecanum wheels, requires that each wheel can be driven independently. Mecanum drive wheels have small rollers attached to the main wheels, which are angled at $45\circ$ to the main wheels. Characteristically, this means that each wheel, when spun, generates a force vector at $45\circ$ to the direction of rotation of the main wheel [20].

The following equations outline how to drive each of the motors' voltages V_1, V_2, V_3 and V_4 (which are proportional voltages which are restricted on the domain [0,1]), given an overall throttle V and a travel direction of Θ, with V_Θ as the angular correction factor. This enables the vehicle to travel in any direction at any throttle up to maximum speed. [21]

$$V_1 = V_d \sin(\theta_d + \frac{\pi}{4}) + V_\theta$$
$$V_2 = V_d \cos(\theta_d + \frac{\pi}{4}) - V_\theta$$
$$V_3 = V_d \cos(\theta_d + \frac{\pi}{4}) + V_\theta$$
$$V_4 = V_d \sin(\theta_d + \frac{\pi}{4}) - V_0$$

Fig. 15 [21]

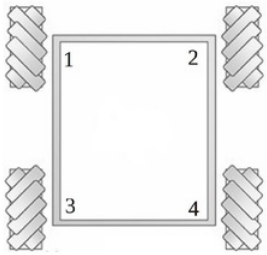


Fig. 16 [21]

An ability to localise the system, using encoders attached to the motors to which the mecanum wheels is additionally important. The following section gives a mathematical derivation to how this may be done.

Following are assumptions of this derivation:

- **Each wheel contribution is additive**
- **Wheel traction is perfect, i.e. there is no slip**
- **Rotary encoders give consistent, accurate data**

Firstly, the following data is needed about the physical characteristics of the system

- **Encoders output 1050 ticks per revolution**
- **Wheel diameter is 0.06m=6cm**
- **Force vectors generated by each wheel is at an angle of $\frac{\pi}{4}$ rad as shown by the thick red lines**

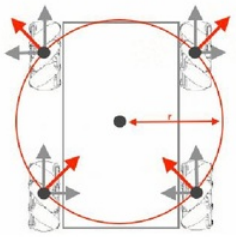


Fig. 17 [22]

For the first wheel (front left), the $F_{1,X}$ and $F_{1,Y}$ components can be extrapolated :

$$F_{1,X} = F_1 sin(\tfrac{\pi}{4}) = -\tfrac{F_1}{\sqrt{2}} \quad \text{and} \quad F_{1,Y} = -F_1 cos(\tfrac{\pi}{4}) = \tfrac{F_1}{\sqrt{2}}$$

Similarly :

$$F_{2,X} = F_2 sin(\tfrac{\pi}{4}) = \tfrac{F_2}{\sqrt{2}} \quad \text{and} \quad F_{2,Y} = -F_2 cos(\tfrac{\pi}{4}) = \tfrac{F_2}{\sqrt{2}}$$
$$F_{3,X} = F_3 sin(\tfrac{\pi}{4}) = \tfrac{F_3}{\sqrt{2}} \quad \text{and} \quad F_{3,Y} = -F_3 cos(\tfrac{\pi}{4}) = \tfrac{F_3}{\sqrt{2}}$$
$$F_{4,X} = F_4 sin(\tfrac{\pi}{4}) = -\tfrac{F_4}{\sqrt{2}} \quad \text{and} \quad F_{4,Y} = -F_4 cos(\tfrac{\pi}{4}) = \tfrac{F_4}{\sqrt{2}}$$

Now, taking R_1, R_2, R_3 *and* R_4 as the number of ticks on the respective enumerated encoders, we can draw correspondences between the force vectors and number of ticks.

we can **draw a correspondence between the force vectors and number of ticks.**

Abstracting this data yields:

$$R_{1,X} = R_1 sin(\tfrac{\pi}{4}) = -\tfrac{R_1}{\sqrt{2}} \quad \text{and} \quad R_{1,Y} = -R_1 cos(\tfrac{\pi}{4}) = \tfrac{R_1}{\sqrt{2}}$$

$R_{2,X} = R_2 sin(\frac{\pi}{4}) = \frac{R_2}{\sqrt{2}}$ *and* $R_{2,Y} = -R_2 cos(\frac{\pi}{4}) = \frac{R_2}{\sqrt{2}}$

$R_{3,X} = R sin(\frac{\pi}{4}) = \frac{R_3}{\sqrt{2}}$ *and* $R_{3,Y} = -R_3 cos(\frac{\pi}{4}) = \frac{R_3}{\sqrt{2}}$

$R_{4,X} = R sin(\frac{\pi}{4}) = -\frac{R_4}{\sqrt{2}}$ *and* $R_{4,Y} = -R_4 cos(\frac{\pi}{4}) = \frac{R_4}{\sqrt{2}}$

The distance per 'tick' of the encoders, needs to be calculated from the wheel geometry:

distance per 'tick' $= T = \frac{1}{(ticks\ per\ rot.)} \times (wheel\ circumference) = \frac{2\pi r}{(ticks\ per\ rot.)} = \frac{\pi \times 0.06}{1050}$

So now we can find the distance travel contribution of any wheel, in X or Y by multiplying by the distance per 'tick' by the 'number of ticks' in X or Y.

$D_{1,X} = R_{1,X} T = -\frac{R_1}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$ **and** $D_{1,Y} = R_{1,Y} T = \frac{R_1}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$

$D_{2,X} = R_{2,X} T = \frac{R_2}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$ **and** $D_{2,Y} = R_{2,Y} T = \frac{R_2}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$

$D_{3,X} = R_{3,X} T = \frac{R_3}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$ **and** $D_{3,Y} = R_{3,Y} T = \frac{R_3}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$

$D_{4,X} = R_{4,X} T = -\frac{R_4}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$ **and** $D_{4,Y} = R_{4,Y} T = \frac{R_4}{\sqrt{2}} \times \frac{2\pi r}{(ticks\ per\ rot.)}$

But the contributions in each dimension can be added and factored:

$D_X = \frac{1}{\sqrt{2}}(-R_1 + R_2 + R_3 - R_4)\frac{2\pi r}{(ticks\ per\ rot.)}$ **and** $D_Y = \frac{1}{\sqrt{2}}(R_1 + R_2 + R_3 + R_4)\frac{2\pi r}{(ticks\ per\ rot.)}$

Which can be implemented in software for calculating the position of the vehicle.

Additionally, it could be expected that rotational errors would build if not all the motors and wheels behaved in exactly the same manner. Thus an angle error calculation was derived for the sake of angular correction.

Considering the same diagram as above, it can be seen that if diagonally opposite wheels rotate in opposite directions, the system will be stationary. While this is partially true, the system would actually rotate in its' place. By subtracting diagonally opposite encoder 'ticks' from one another, we can determine a proportional rotation. This term doesn't need accurate units as it may be used to feed a proportional controller, to minimise angle error. This is a valid calculation as all the wheels lie on the path of a circle, with a common center, as shown in the diagram. (This is an assumption which could be proven).

$$\Theta_{error} \propto (R_1 + R_2 - R_3 - R_4)$$

Chapter 3

Final Design Elements

The following chapter will include detailed design of the vehicle, including final decisions made from prior research as well as the reasons for those decisions. Efforts will be made to fully explain how each software and hardware component performs and functions as part of the system as a whole, including how the system functions as a whole.

3.1 Hardware

The hardware electrical wiring is given in the following diagram Fig. 18. Components are wired into the MyRio using 3 connectors with connector pins. Components are given explanation in the following section.

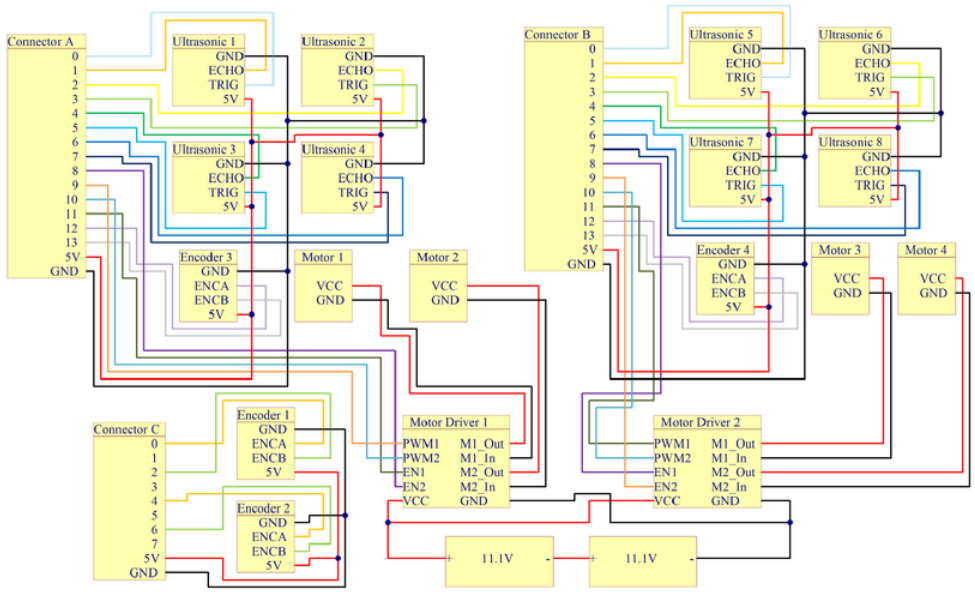


Fig. 18

3.1.1 Motors

For the reasons outlined in chapter 2 of this document, the brushed DC motor type was chosen for their simple control characteristics, and simple set-up. The motors used were the NeveRest 20 Gearmotor, which is a 1:20 gear ratio brushed DC motor. The official constraints as

defined in its' specifications sheet are that it is nominally rated for 12V at a stall current of 11.5A max. The motors were eventually run at the higher voltage of ~ 22V (originally ~12V during testing) to increase the motor speed above the rated 275RPM, however special caution was taken to ensure that the motors didn't overdraw current, if there wasn't sufficient torque to move the motors.

The motors themselves were powered by two MOSFET based motor drivers, each of which are capable of driving two DC motors. These motor controllers utilised two control pins per motor, one of which was a PWM proportional speed control pin and the other a directional select pin. Motor control logic was passed from the MyRio to the motor driver chips by two digital I/O pins (per motor) through the FPGA unit on the MyRio.

3.1.2 Wheel Type and Localisation

The comparative benefits between the control characteristics of the available wheel types were reviewed, and it was decided that for the purpose of a small scale vehicle travelling at low speeds, that mecanum wheels were ideal for the sake of their favourable maneuverability in tight spaces, especially considering that the vehicle would be capable of being translated in lateral position without having to be rotated, despite the fact that these may not have the best traction and speed characteristics (they can propel a vehicle at ~0.707 times the speed of a 'normal' wheel). Additionally, compared with a typical car's control method (front wheel turning), the control method for the mecanum wheels is very simple to implement from a software standpoint. Mecanum wheels of diameter 0.06m were used. The control characteristics are detailed in chapter 2.4.4 of this document.

3.1.3 Encoders

The stock encoders which come as part of the NeveRest 20 Gearmotor package were primarily used to track the positioning of the system, with respect to the surface upon which it travels. In order to achieve this, a mathematical transformation from the rotational frame of the motors into the frame of the surface was used (of which is explained in chapter 2.4.5 of this document).

$$D_X = \frac{1}{\sqrt{2}}(-R_1 + R_2 + R_3 - R_4) \times \frac{\pi \cdot 0.06}{1050} \quad \textbf{and} \quad D_Y = \frac{1}{\sqrt{2}}(R_1 + R_2 + R_3 + R_4) \times \frac{\pi \cdot 0.06}{1050}$$

3.1.4 Frame/Chassis

The chassis consisted of two transparent acrylic plastic (Polymethyl methacrylate) sheets, as well as polycarbonate 3D printed holders for the motors. The components are held together by bolts screwed through both the acrylic plastic sheets as well as the motor holders. Both the motor holders and all the electronic components are held between the two acrylic sheets, with the exception of the batteries, which are secured atop the vehicle. There are other non-structural acrylic plastic parts, which hold the ultrasonic sensors in place, and are screwed into the top acrylic pastic structural sheet.

3.1.5 Batteries and Buck Converter

Initially during early stages of testing, a ~12V NiMH battery pack was used, as it was available on hand. The capacity of the battery would run low quickly, and the voltage supply was lower than desired, so it was eventually replaced. The supply for the system ended up consisting of two LiPo battery packs, each of which are rated to run at 11.1V nominally, however upon testing, it

was discovered that these would each run at up to 12.2V at full charge. These battery packs were connected in series in order to achieve the intended ~22V supply for the motors. The cells were connected to the system using Anderson Powerpoles, such that each battery pack could be removed individually. A supply voltage in the range of 6V to 16V is required to supply the MyRio, so a generic step-down converter was connected to power it. The SKYRC iMAX LiPo battery charger was used to charge the batteries, external to the vehicle.

3.1.6 Ultrasonic Sensors

Multiple HC-SR04 ultrasonic distance sensors were used. On each side of the vehicle were two ultrasonic distance sensors. The one-dimensional axis along which the sensors measure distance is orthogonal to the side surface of the vehicle, and through the middle of the sensor. There were two sensors on each side for two main reasons. The first reason, is for localisation, and there are two sensors to verify that a distance reading is valid, such that 'bad' values may be discarded. Secondly, an angle error may be identified, by taking the difference between the distance values. If the system has rotated by some unintended amount, this error in rotation can be quantified and subsequently the control system can correct for this error. The layout of the ultrasonics are to be as follows:

Fig. 19

3.1.7 FPGA and Hardware Interface

The MyRio's FPGA coprocessor is what enables the Real Time System's (RTS's) microprocessor to interface with hardware devices at a very low level. Interfacing with devices through the FPGA is done through the DIO pins located on one of three ports of the MyRio (A, B and C). Further information on this is provided in chapter 2.1.2 of this document.

3.2 Software

The majority of time spent on the project was on the software design of this system. The intent was to build the majority of the code from scratch in LabView (minimising use of high-level functions or other code-bases).

3.2.1 FPGA and interfacing with RTS (Real Time System)

Interfacing with the FPGA within the RTS enables the MyRio to have fast response times when interfacing with low level devices. The FPGA was used for all I/O, which includes the reading of data from rotary encoders, writing control outputs to the motors (including PWM outputs), as well as reading and writing data to and from the ultrasonic sensors. The details of the relationship between FPGA and RTS are found in chapter 2.1.2 of this document.

3.2.2 Basic Control System

Within the FPGA exists the following code for controlling each motor, for each of which there are two control components:

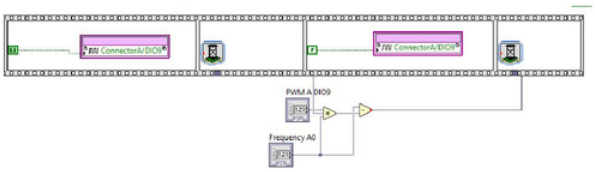


Fig. 20

Fig. 21

The left part of the code above (Fig. 20), generates a PWM signal, of duty cycle specified by a FLOAT input passed from the RTS, restricted on the interval [0,1] (in the specific case in Fig. 20, it is labelled PWM A DIO9). This is how the speed of the motors is controlled. The boolean input (Fig. 21 passes on a boolean input control from the RTS, telling the motor which direction to turn (where FALSE is forward rotation, TRUE is reverse rotation).

A control scheme was developed in the RTS, where the control inputs need to be passed through the FPGA. Since mecanum wheels were being used, the following control scheme derived from the document "Simplistic Control of Mecanum Drive" was used [21]. This is further explained in chapter 2.4.4.

Following (Fig. 22) is the code in the RTS which allowed the control inputs to be generated and then passed on to the FPGA. The blue blocks contain the control equations (described in chapter 2.4.4). Each block represents the output control for a different wheel. The inputs to the equations are passed into these blocks from the left, and outputs to the right.

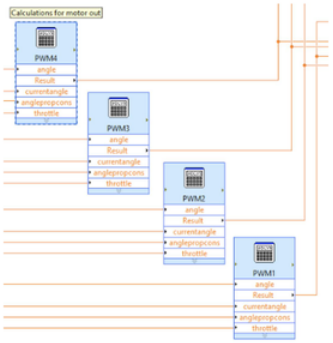


Fig. 22

The function in Fig. 23 is used to send and receive data to and from the FPGA. Note that the triangular blocks to the left are logical functions which split the throttle information into magnitude and directional components, so that it is in the format needed by the FPGA.

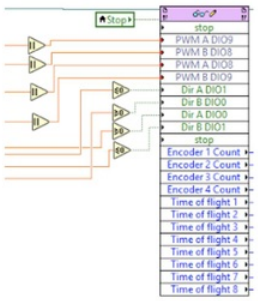


Fig. 23

The following code shows these pieces of code together in context. The parts to the left consist of the various (higher level) inputs for controlling the motors. There are many controls due to the fact that both autonomous control, and direct human control are optional. More explanation on this topic is not important.

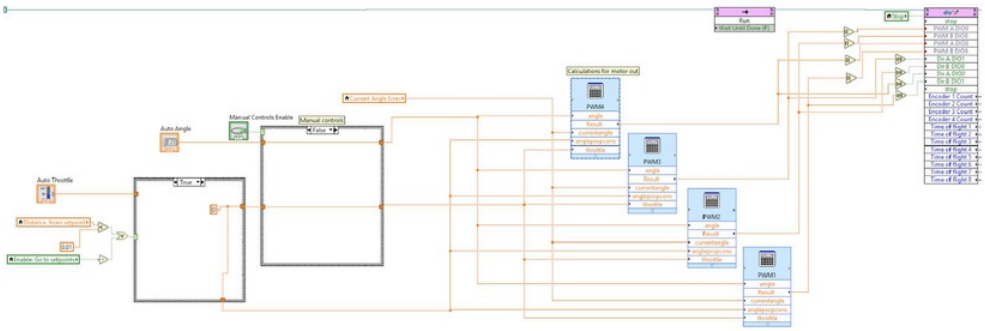


Fig. 24

3.2.3 Rotary Encoders

Rotary encoders track the sum of the changes in rotational position of the motors/wheels. The algorithm used to record these changes in position is as follows (for one rotary encoder) running on the FPGA (this particular piece of code was reused from the 2016 NI ARC MQMechatronics) shown in Fig. 25:

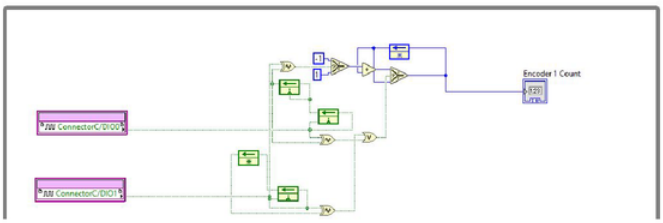


Fig. 25

The output of this algorithm is in a number of 'ticks'. For these encoders there are 1050 ticks per revolution. The value of 'ticks' are sent to the RTS, and used in localisation as explained in the following section.

3.2.4 Position Calculation and Localisation

Position calculation and localisation is based primarily on the rotary encoders. The following equations (derived in chapter 2.4.4) are used in the calculation blocks below.

$$D_X = \frac{1}{\sqrt{2}}(-R_1 + R_2 + R_3 - R_4)\frac{2\pi r}{(ticks\ per\ rot.)} \quad \textbf{and} \quad D_Y = \frac{1}{\sqrt{2}}(R_1 + R_2 + R_3 + R_4)\frac{2\pi r}{(ticks\ per\ rot.)}$$

At a later stage, an improvement to this method, i.e. using position correction, will be detailed. By performing a particular mathematical transformation to the rotary encoder values, a contribution of position on the surface of movement can be calculated for each wheel. Taking the sum of these, with consideration for their directional components, gives the sum of changes in position, which is the position of the vehicle on a flat two-dimensional surface. The derivation (and equations) of this transformation is given in the research section of this document (chapter 2.4.4). Following is the representation of this code in LabView, where the blue boxes have the equations written 'inside' them:

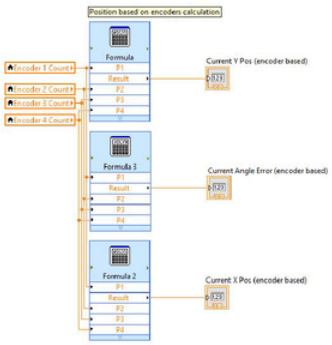


Fig. 26

The output 'Current Angle Error (encoder based)' works in a similar way, and calculates a value for how much the vehicle has rotated, with respect to its initial reference frame. This is to be used later in a proportional controller to help ensure that the vehicle is kept straight. As this simply needed to be of arbitrary units fed into a proportional controller, values do not have to be scaled to reflect true units (such as radians).

$$R_{ROT} = -R_1 + R_2 - R_3 + R_4$$

3.2.5 Going To A Set Position

Now that the system localises itself in a set of 2D coordinates, and also drive in any direction with throttle, it is possible to design a proportional controller, to force the vehicle to drive to a 'set-point'. It is assumed the vehicle travels on a flat surface of two coordinates: X and Y. A proportional controller can utilise this set of two coordinates.

Two sets of coordinates are needed by a controller to make the system drive to a set point. One is an input control; the coordinate set-point (that is, the location the vehicle wants to drive to). The other is the current position (where the vehicle is now with respect to the same coordinate frame). To make the vehicle to drive towards the set-point coordinate, an angle for travel must be calculated, and a proportional throttle also generated. The proportional throttle could be thought of like an 'error' value, which in this case is proportional to the distance between the two points (current position and set-position). The throttle for the control system could be generated by scaling this 'error' value by a proportional constant K_p and limiting it to 1 (the maximum throttle).

$$throttle = K_p.\sqrt{(x_{set} - x_{current})^2 + (y_{set} - y_{current})^2} = K_p.E$$

Where K_p is the proportional constant, and E is the distance from the set point or the 'error'.

Using basic trigonometry, the angle between the current position and set position is generated, which could be used as the input Θ (the direction of travel) for the control system (which was given in 3.2.2). An equation for calculating the angle between the current position and the set-point is as follows:

$$\textbf{If } x_{set} - x_{current} > 0: \qquad \Theta = arctan(\frac{y_{set} - y_{current}}{x_{set} - x_{current}})$$

$$\textbf{If } x_{set} - x_{current} > 0: \qquad \Theta = \pi + arctan(\frac{y_{set} - y_{current}}{x_{set} - x_{current}})$$

(This solution is segmented due to the symmetry of the tan function, so would yield duplicate results for angles greater than π).

The code performing these calculations follows. Note that the black box is the case statement which splits the solutions for solving angle, and the orange lines exiting the image connect to other code not shown here.

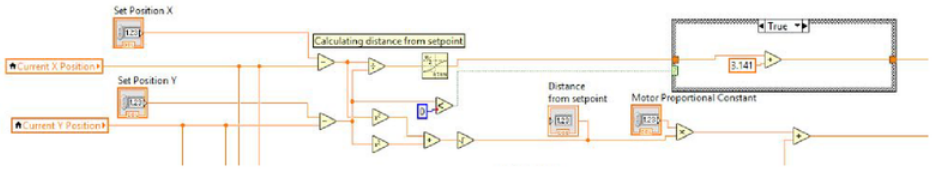


Fig. 27

An almost identical system was implemented for speed ramp-up, where the speed is proportional to the distance from a 'previous set-point', plus a small amount (to overcome friction torque of the motors and to start moving the vehicle). The 'previous set-point' is a set of coordinate variables which are the last destination achieved (which will be explained further later on, when set-points are sequenced). In the following figure, part of the code for this is shown. Note primarily that there are variables which hold the previous values of set-position, which are compared against (not shown in image) the current set-position.

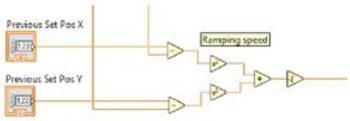


Fig. 28

A speed (limited) zone existed in the course, so this needed to be implemented. The following code checks if the system is in the speed zone, then uses a case statement to either limit the throttle or not.

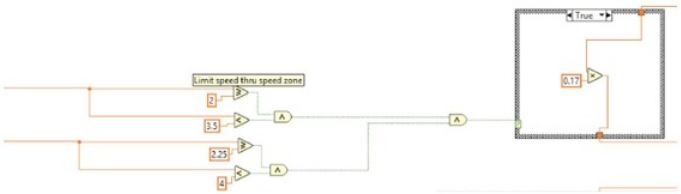


Fig. 29

Following is the full control system code, to show context of these pieces of code.

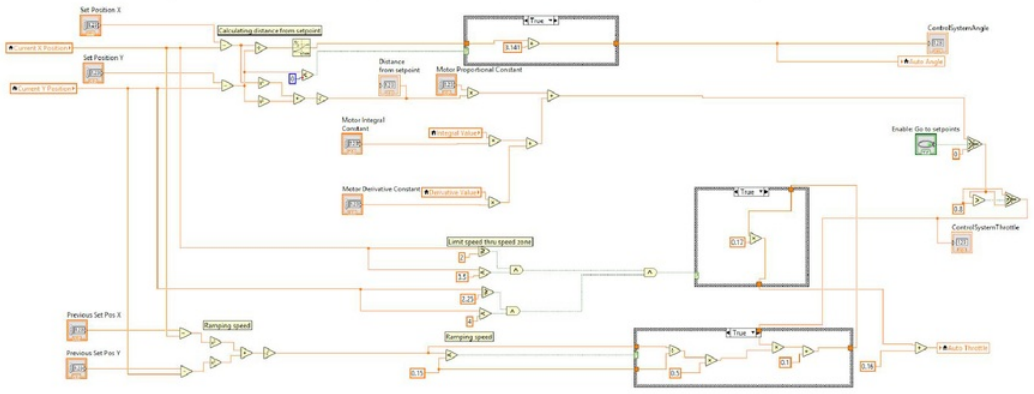


Fig. 30

In summary, this code takes control inputs in the form of a set position and current position, and turns it into control outputs to the motors. More functionality than this is shown here, however, the majority of the functionality will be elaborated further in this chapter.

A PID was developed, for the control of the motors, to improve acceleration and deceleration, while minimising overshoot. This PID was simple and built from scratch. In order to calculate derivatives and integrals of the 'error' signal, the change in error signal was calculated for

small time slices. For a discrete time interval, change in error would be calculated (over the previous time interval, since the last time it was calculated). This process would repeat constantly. Three of these past values of change in error values were stored at one time, stored as three variables, all of which would shift down when a new value was inserted. These were used to calculate integral and derivative approximations over a short past time interval. This took the assumption that changes in the signal were linear. The integral and derivatives calculated, were each multiplied by K_I and K_D scaling constants respectively. Following are example parts of the code for the PID (doesn't include code for the entire PID):

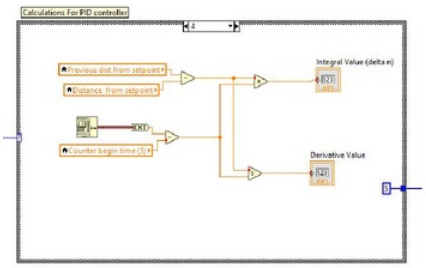


Fig. 31

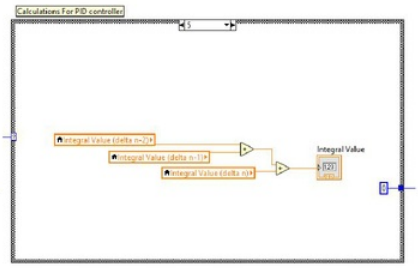


Fig.32

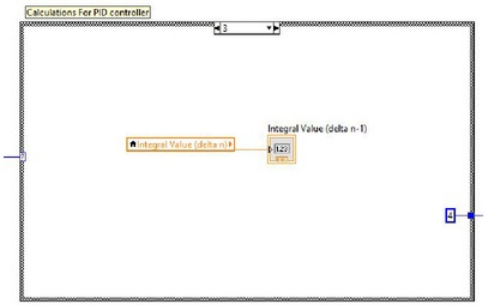


Fig. 33

Above, the top left figure (Fig. 31) demonstrates how the integral and derivative constants are generated for the most recent time slice. The top right figure (Fig. 32) shows the addition of three past integral values of the three most recent time slices. The bottom figure (Fig. 33) demonstrates the change in error values, shifting.

Additionally, the motors were designed to turn off once within very close proximity to the set-point. This consisted of a set of comparators, comparing the current position against the set position, and disabling the motors if the distance was less than 0.001m. Shown in the following Fig. 34.

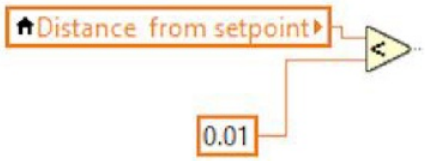


Fig. 34

3.2.6 Going to Set-Points Sequencing

The means by which the vehicle is able to navigate some path, is by travelling, in order, to a number of different setpoints. At any one time, there is only a single setpoint, in memory, which the vehicle travels towards, in a straight line. By storing a sequence of X and Y coordinates in two one-dimensional arrays; one corresponding to a sequence of X-ordinates, and the other corresponding to a sequence of Y-ordinates, the system can execute travelling to these setpoints one-at-a-time. The system loads up the first X and Y coordinates from the arrays, and inserts them as the setpoint. Once the system determines that it is in close proximity to the setpoint, it stores the next set of coordinates as the setpoint, and continues to that setpoint. This happens until the final item in the array has been loaded, then it terminates the program. An integer labelled the 'set-point index' would indicate which setpoint element/index was being accessed at any time, so the current X-Y coordinate could be traced. Additionally, when the next setpoint would be loaded, the previous setpoint would be stored in a variable called 'previous set-point'. This value was used in the controller for speed ramp up exclusively.

3.2.7 Ultrasonic Sensors

Data retrieved from the ultrasonic sensors is in the form of a time measurement, which is the duration for a sound pulse to be emitted, reflect off an object, and be received again. Since the speed of sound is known, distance measurements could be determined. The code below (Fig. 35) shows the timing measurements being read in the RTS from the FPGA, and the calculations turning these into distance measurements:

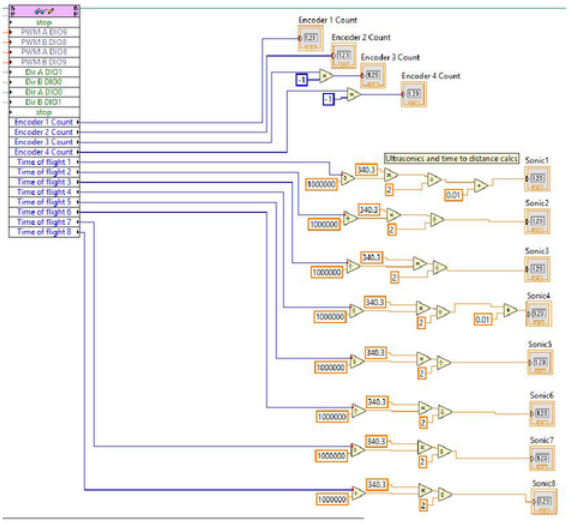


Fig. 35

While the data from ultrasonics could be used directly, there is a relatively high variability in data received. To solve this, the following strategy was used. The code below accumulates values from the ultrasonic sensors, over every 30 data points received, and averages them to ensure data

accuracy. A high refresh rate is not as important as having accurate data, since the rotary encoders exist for tracking small changes in position at a high rate, while the ultrasonics need to be used to correct the position. The inputs to the left are the data from the ultrasonics, while the output variables to the right are the averaged, accumulated data.

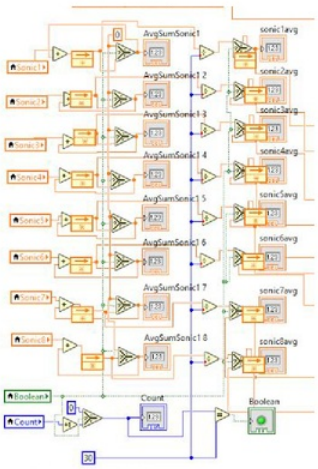


Fig. 36

3.2.8 Position Correction Algorithm

Position correction in this context, is the process of designing an algorithm which is capable of using sensor readings (of its environment) to recentre a system's coordinates to its' 'actual' coordinates. This becomes necessary as error builds up over time when rotary encoders are used. 'Errors' in this context refer to inaccuracy of sensor readings, which are partly due to the fact that, in this instance, rotary encoders track many small changes in position, which causes small inaccuracy errors to accumulate.

The system primarily uses the rotary encoders (with methods previously described), for tracking the system's spatial movement. Additionally, it uses ultrasonic sensors to make measurements of the environment, to correct accumulated error of position tracking. On each side of the vehicle are two ultrasonic sensors (the layout of which is shown in Fig. 19 in 3.1.6), which provide useful data for correcting both position, and (error) rotation of the system (which is explained in the next section 3.2.8).

Pairs of sensors on each side of the vehicle (front, back and both sides) are always used together in the following ways. If the vehicle isn't physically rotated in the frame (which can be determined), and a flat wall is beside the vehicle, the sensor pair can be used for accurate distance measurement. This position correction is implemented in code which runs and updates continuously, but whether it updates (and how) are dependant on the readings from the ultrasonic sensors.

For each pair of sensors, comparators test the difference between the sensor readings. If the sensor values very close to each other, the system will be told to correct the position of the vehicle, by using the average of the two sensor values, then resetting the vehicle's position using this sensor data. Similarly, if the difference between a pair of ultrasonic sensor readings are greater than some amount, but not too large (in case faulty values are received), the system is assumed to have rotated (which, again, will be explained in the next section 3.2.8).

Firstly, the system tests if the measurements from the ultrasonics are within 0.3m (as measurements given by the ultrasonic sensors become more inaccurate with distance). Then, the system tests if they are near each other (in this case within 0.01m of each other).

Below (Fig. 37) is an example part of the code which uses comparators to test whether both sensors outputs are below 0.3m. The green output to the right is a boolean output.

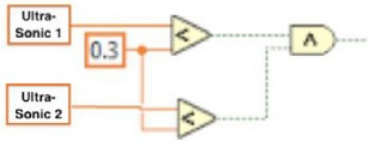


Fig. 37

In Fig. 38 is this part of the code in context:

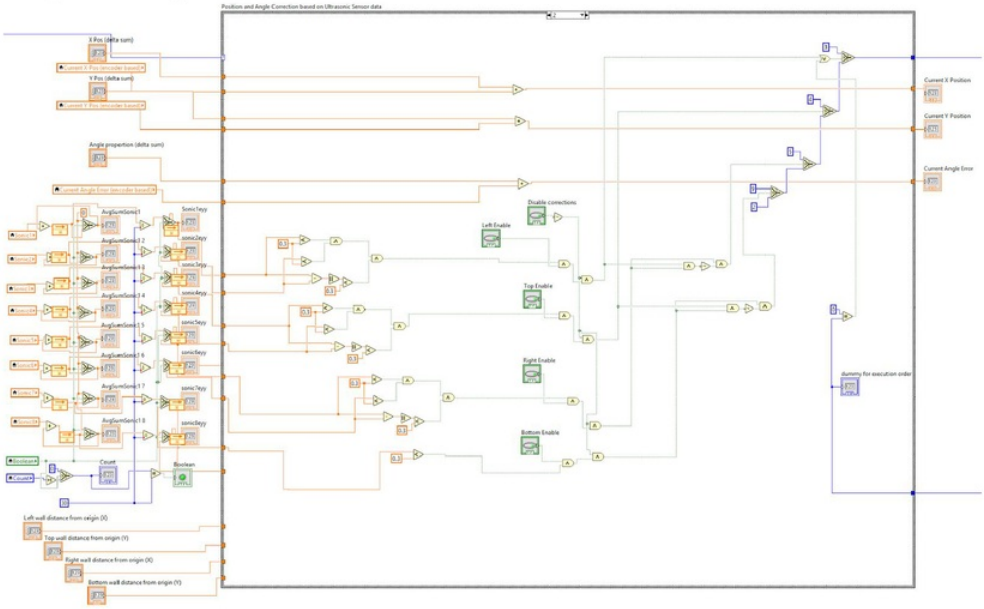


Fig. 38

If the comparators' conditions are satisfied, system uses the result of these distance measurements to correct its position.

The system's position, is determined by both rotary encoders and ultrasonic sensors. These need to be combined in some way without conflict to output a single set of position values. The way this was done, was by combining utilising two sets variables. One of these variable sets were labelled 'encoder based' variables, while the others were 'delta sum' variables. The position as calculated from the rotary encoder values, provided the primary contribution. This was done, due to the fact that the time divisions of measurement of position from the rotary encoders are very small, so provide a smooth output for position, while data from the ultrasonics are relatively slow and error prone.

The second contribution is from a set of 'delta sum' variables, calculated from the ultrasonic

sensors. Every time a new measurement was taken with the ultrasonic sensors, the physical position of the wall being observed had to be known (how this was done, will be explained later in this section). Knowing both the physical position of the wall, as well as the system's distance from the wall (from the ultrasonics) provides enough information to work out the vehicle's actual current position in one dimension. Thus, using these pieces of information, the 'delta sum' variables needed to be generated, such that when added to the contribution of 'encoder based' variables, would give the correct position of the vehicle. The variables can be seen at the top right and top left of the Fig. 38 above. An equation describing this relationship follows:

$$P_{total} = P_{encoder\ based} + (P_{wall} + P_{sonic} - P_{encoder\ based}) = P_{encoder\ based} + P_{delta\ sum}$$

Where P_{total} is the position of the system in either X or Y, P_{wall} is the known position of the wall and P_{sonic} is the position value given by a ultrasonic sensor.

Now, how the system knows the position of the walls around it, will be explained. As previously mentioned, it is important for the system to know the position of a wall under observation by an ultrasonic sensor, if needing to localise the vehicle. The assumption is made that the current X and Y coordinates for system localisation are approximately accurate. A long series of logical (comparator) cases are used to test which region of the map the vehicle is in, and depending on these (where the vehicle is), the location of the nearest walls surrounding the vehicle are output. Under the software architecture given here, this means that a new set of logical cases needs to be generated for each particular map. Clearly this is not an ideal solution, but is suitable in a small map without a high level of detail. Automating this process, using a map as an input for the software, is an ideal goal in continuation of this project, and was intended goal, but there was insufficient time to fully implement this functionality. The following code is the implementation used. Observe the majority of code (covering the left of the figure) consists of a large number of logical cases, checking where the vehicle is, so that it knows which wall positions to output.

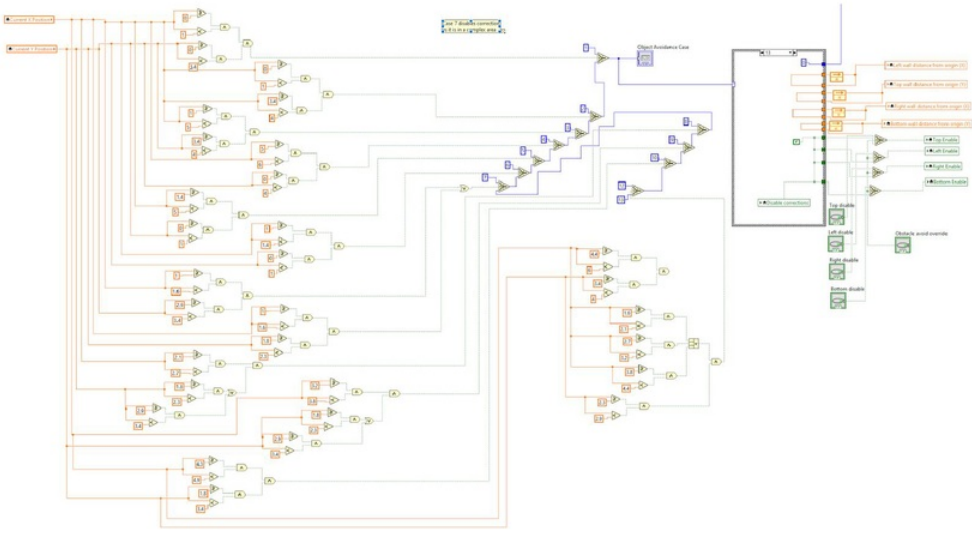


Fig. 39

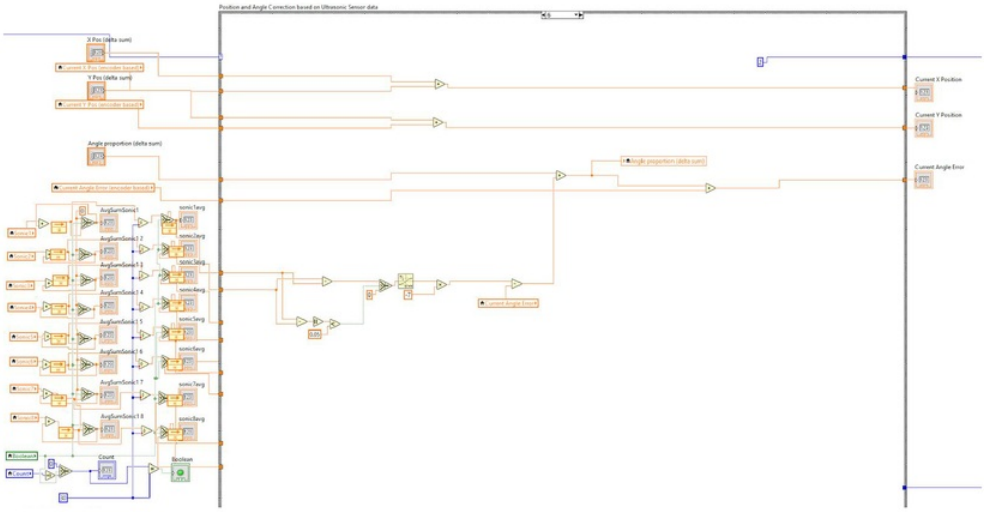


Fig. 40

3.2.9 Angle Correction

The vehicle wasn't designed to be able to rotate to any set angles. As the vehicle can travel in any direction and correct its position, it was unnecessary to intentionally rotate the vehicle. However, due to small uncontrollable factors such as unequal performance of the motors, unequal traction for the wheels and accumulated errors on rotary encoders, the system naturally (albeit slowly) can rotate unintentionally, so it became necessary to correct this (ensure the set angle was always zero). Calculation of 'error' in terms of angle was performed using both encoder values and ultrasonics. The angle of rotation, in terms of encoder values is given by:

$$\Theta_1 = k(R_1 + R_2 - R_3 - R_4)$$

Which is found in section 2.4.4, k is a scaling factor, the R's are encoder values and Θ_2 is a solution of angle rotation.

Angle error found in terms of ultrasonic sensors is:

$$\Theta_2 = \pi - arctan(\frac{l_2 - l_1}{d})$$

Where l_1 and l_2 are the distance measurements from the ultrasonic sensors, d is the distance between the sensors (which was 0.01m), the R's are encoder values and Θ_2 is a solution of angle rotation, from zero angle. The following diagram (Fig. 41) shows geometry:

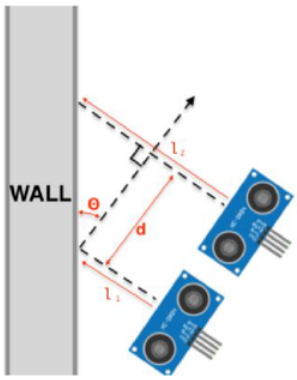


Fig. 41

The angle correction, consisted of combining both encoder and ultrasonic data. These were combined in a the same way which was described for position correction in 3.2.8. That is, the encoder values served as the basis for calculating an error angle, and the ultrasonic sensors were used to improve accuracy of this data. Combining this data allowed a variable called 'Current Angle Error' to be calculated.

Since the amount of angle of rotation could be quantified, a simple proportional controller was implemented within the main wheel control system code, using this 'Current Angle Error' variable, described in control system section 3.2.2.

It must be reiterated at this point that the system isn't designed to intentionally rotate in the frame, but every instance of angle calculations are for the purpose of rotating the system back to the same zero angle.

3.2.10 Path Selection Algorithm

Part of the requirement of the project is the ability to navigate the vehicle, autonomously, through a grid of obstacles. This means that the system needed to be able to calculate an optimal path around the obstacles, using a map as an input. The path selection algorithm needed to be capable of navigating a 'city grid'. This grid in the centre of the map could have an obstacle in any of the locations showing a labelled box in Fig. 42. A number of paths are possible.

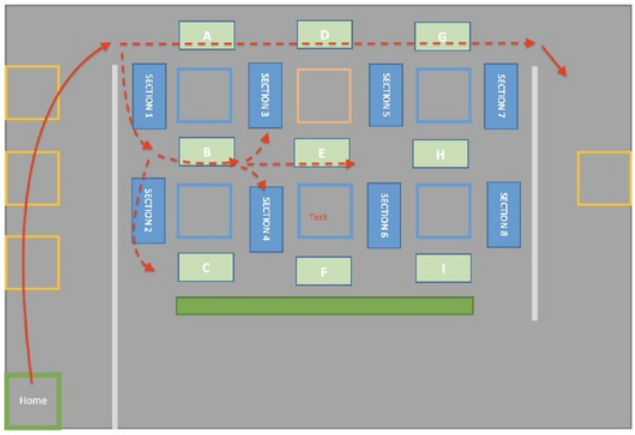


Fig. 42

The map itself, in this city grid region, being in the shape of a simple grid, was represented by a discrete map of 5×7 resolution. Each value on this map had an associated physical location. Within this region, each grid point could be represented as either having an obstacle or not, so the map was populated with boolean values. This map data was effectively represented as a two-dimensional array, where each dimension index would represent an X or Y spatial dimension, and the boolean data elements, represent whether there was an obstacle at each grid point index (either TRUE or FALSE). This grid was extended to have a boundary of TRUE values to ensure a closed boundary for the map, such that the final array 'map' had a size of 7×9. In labview, the representation of this map is shown in Fig. 43:

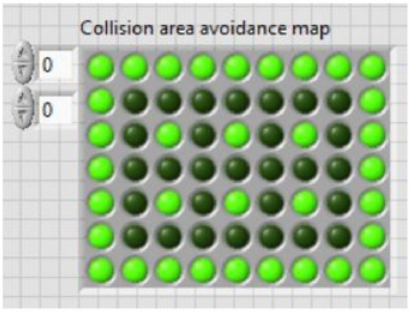


Fig. 43

When the system starts up, the boolean map is populated with information about the location of the obstacles, which is possible since it is able to retrieve the locations of obstacles in advance from the WiFi server (which will be detailed in section 3.2.11). This was done as it was specified in the competition rules. The following code (Fig. 44) compares the string data, retrieved over WiFi, with known names of locations, and generates the index values (coordinates) for insertion into the above map Fig. 43.

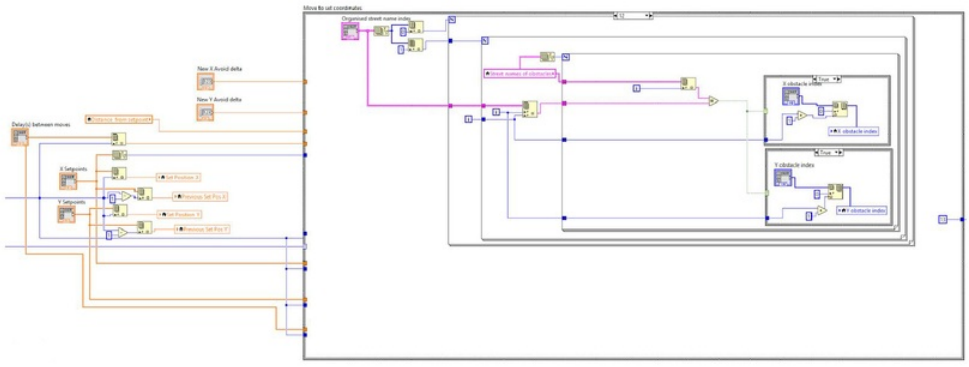


Fig. 44

The obstacles locations are inserted into the map above (Fig. 43). Now, the system can perform the path selection algorithm, and determine the best path to travel, even before the vehicle begins moving. The algorithm itself simply takes steps through the map, checking if there are obstacles at each step. The algorithm prioritises stepping up, then right, then down then left, but will not step in a direction in which it has visited before. Once the destination (upper right hand of the map) is arrived, the algorithm will terminate, and will insert the path found, into the sequence of setpoints.

Parts of the code described above will be shown following. The following code (Fig. 45) checks the values surrounding the 'current location' in priority order specified above:

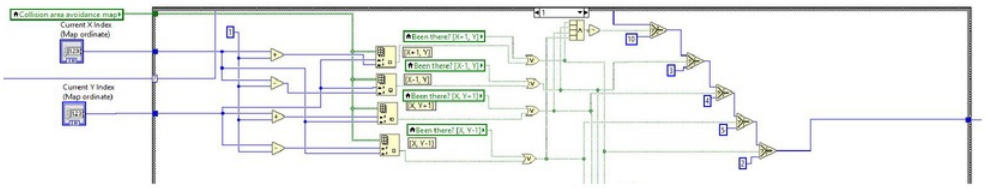


Fig. 45

The following code stores the next location in the path, if determined suitable:

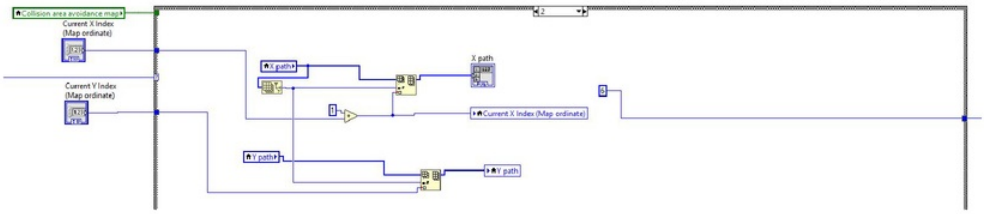


Fig. 46

The following code generates a spatial path, in the frame of the vehicle, from the path locations on the map:

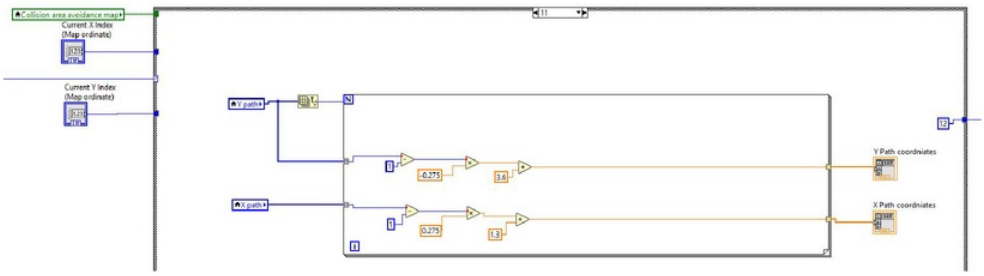


Fig. 47

These locations are then inserted into the main path (setpoint arrays) which the vehicle will take:

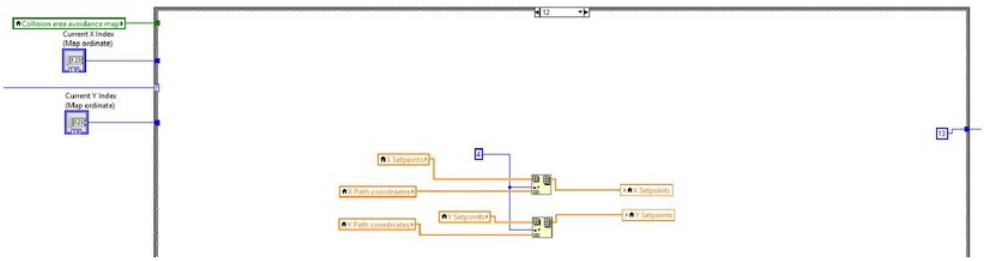


Fig. 48

3.2.11 WiFi Server Retrieval

The vehicle, once connected to a WiFi network, is able to retrieve a string value over the UDP protocol, from a server designed by the NI. The following code retrieves the string, and separates it by a specified separator character, which is the comma (,). This code was derived from examples available in the LabView suite, and was not written for this project (Fig. 49):

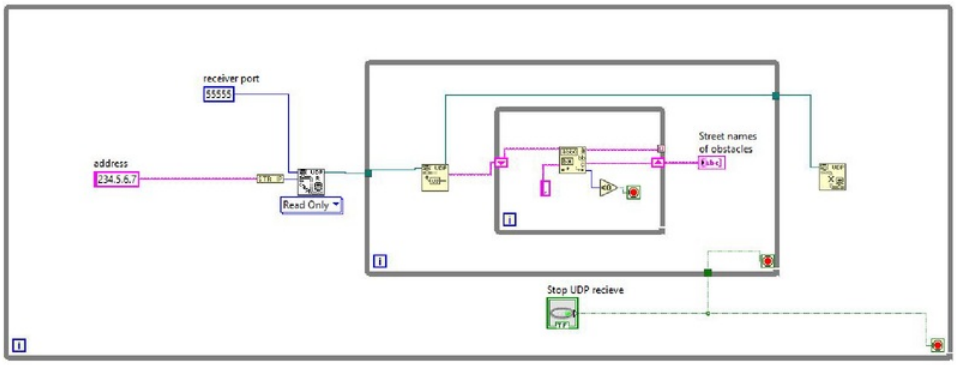


Fig. 49

Chapter 5

Conclusions

5.1 Discussion

The hardware construction was simplistic and consisted primarily of parts which were easily available on hand, which didn't require a significant amount of work to make functional. Geared brushed DC motors were used, which, have generally worse characteristics than the PMBLDC motors which were investigated and researched extensively (further detailed in the research chapter 2 of this document). Despite the fact that the control characteristics of the PMBLDC were heavily investigated, as it was known that the most difficult aspect of the project was designing the autonomous control system, so it was preferred to remain with something simple. As a result, the quality of the design wasn't at the desired standard. This would be addressed in future revisions of the hardware, and it must be reiterated that this project was intended to be a prototype, for the demonstration of basic functionality and testing. As motor control is a well established field, and there are a variety of controllers available, it is assumed that other motors and controllers could be used. In addition, cost was a significant consideration at this time, and PMBLDC motors cost significantly more than their brushed counterparts, on a retail level.

The types of wheels used, the mecanum type wheels, had incredibly simple control characteristics (especially in the four wheel case used, due to symmetry). Translation of the system on a surface could be done reasonably accurately (in the direction of some set angle) at an arbitrary rate, without needing to know more physical information about the system. Further, modelling the system, with feedback, eventually enabled particular values of physical position and angle rotation to be obtained. This system performed satisfactorily. Being able to rotate the vehicle wasn't implemented, but it wasn't a limitation in this case. In future, it may be useful to implement, especially since it wouldn't be too difficult to implement.

The motors used, each contained a rotary encoder. This was satisfactory as a basis for feedback for motor rotation, as estimating it without feedback is grossly inaccurate. Using external, potentially more accurate rotary encoders was considered, however was decided against as it was thought to be unnecessary, as it was assumed that even highly accurate rotary encoders wouldn't improve localisation, since position correction has been implemented. This was perhaps an incorrect assumption, which should be investigated in the future, as the proportionality between these is not known. More research on this would be useful.

Modelling the localisation with rotary encoders involved using the dimensions of the wheels, as well as knowing the number of ticks per rotation generated by the rotary encoders, in order to work out distance travelled by the vehicle. Unfortunately, this modelling seemed to be insufficient, as there were small discrepancies between the ticks per rotation of the wheels, among

other unknown factors. Therefore, scaling parameters had to be inserted and tuned for the calculation of position, by trial and error. Better rotary encoders could have provided a partial solution to this issue.

The low level control system for the motors takes a throttle and a direction of travel (angle). These inputs are to be transformed into control inputs for individual motors using a specially designed function, and then sent to the FPGA containing logic for PWM motor control.

The system initially utilised a proportional controller to drive the vehicle to set-points. Inputs to this controller were the current position of the vehicle as well as the desired position. The outputs were control outputs for the motors, which are generated as a throttle and direction of travel, before being transformed into the control inputs for the motors. This system functioned sufficiently well as intended. A PID was developed, as the proportional controller was insufficient, and there was significant overshoot. While this functioned, the design of the PID itself, including number of past time samples, and sampling duration could be improved.

There were constants for the PID which had to be tuned in order for the system to attain its set position without instability, but this was a simple solution found through trial and error. This could be improved by employing mathematical techniques to solve optimal parameters.

Early in the project, when the voltage on the motors were ~12V, there was never any overshoot, as the motors automatically had a relatively strong braking effect. As such a proportional controller was sufficient to drive the vehicle to a set-point. When the voltage was raised to ~24V, the vehicle could travel significantly faster, such that the vehicle would grossly overshoot, and if the proportional constant was dropped sufficiently to prevent overshoot. Many attempts were made to tune the constants, after which a suitable set of constants were found. However, there were issues with the stability of the control system, when the battery capacity ran low.

Unusual, unexpected behaviour of the system's driving stability arose when the vehicle's battery capacity ran low. The system showed signs of instability, and displayed slow acceleration at times, and significant overshoot otherwise. It is assumed at this stage that the fact that the motors are driven by the batteries directly, the drop in voltage and current supply changed the torque and acceleration characteristics of the vehicle, which in turn, with the tuned parameters of the PID not designed for a system of that specification. Power regulators would be needed in future, and it was a flaw in design to not consider this. In future power supply will need to be kept consistent.

Ultrasonic sensors were chosen for position correction and localisation, due to their low cost, and were assumed to be sufficient for this project. While this was a correct assumption, that they were sufficient, ideally, better and more accurate hardware would preferably be used. Data provided by these sensors have a relatively large variance in their output, and occasionally incorrect data points are received, for example if a ping is not detected after returning from bouncing off an object. Some of this could be rectified in software. Data with an unexpectedly large or small value could be discarded, and average values over the last few received data points could be taken to improve data consistency. There are many improvements which could be done in future, without using new sensors. Improving data consistency could be improved by not simply taking the last few measurements and averaging them, but instead projecting an expected current value from the last

few measurements, by reducing the system to a linear least squares approximation, for the last few values received, and solving the expected solution at the current time. The maximum frequency at which data measurements could be made could be optimised. In addition, attaching some kind of physical sound polariser to the ultrasonic sensors might ensure more consistent results, limiting sound from bouncing off objects off-axis to the intended axis of measurement for the sensor. Preferably, LIDAR rangefinders could be used, which provide extremely accurate data at a high frequency, however they come with high purchase cost.

Interfacing the hardware with the FPGA component on the MyRio wasn't initially without challenge. It was discovered that while the RTS had access to hardware pins, if the FPGA was accessing any of the pins, the RTS was refused access. Unfortunately this wasn't known for some time. Once this was discovered, all interfacing with hardware was routed through the FPGA, however fortunately, this was seen to be beneficial, as instead of relying on inbuilt functions for controlling low level hardware, the code for interfacing with hardware could be written directly at a very low level, due to the fast response times possible with the FPGA. For example, code for controlling motor speed through PWM outputs, was coded directly on the FPGA module, by designing code to quickly turn on and off an output, proportional to some FLOAT input from the RTS. Another use was receiving pulses from the rotary encoders. Using the FPGA improves data reliability, since unlike running a program solely in the RTS, there is no interruption in either data processing or data retrieval, and significantly reduces the amount of potentially missed data, errors or process delay. An FPGA co-processor proved extremely useful.

In order to use data from ultrasonic sensors to generate accurate localisation data, the system needs to be able to know which walls and objects are within its immediate vicinity, otherwise, the system won't know, for example, which wall it is under observation that is immediately in front of the vehicle. Thus, if the system is trying to correct and update its current position, it will need to know which object or wall corresponds to measurements from ultrasonic sensors. It is the lack of detailed data from ultrasonic sensors that necessitates this. An issue arising from this, during the design and testing phase of this project, was that if there was a mistake in the code that allowed a measurement to be miscorellated to the wrong wall, it would 'think' it was somewhere other than where it really was. Early on in the project, the plan was to design a map, from where the system would be able to calculate where the nearest walls were, and then correct its position by combining this information with that from the ultrasonic sensors. Amidst other design elements, there was insufficient time to complete this, as it would require either generating a high resolution map, or parameterising the entire system (including the map) into mathematical representations.

Either of these solutions would have been suitable, but the amount of work involved in either of these strategies would have been extensive, or required the use of other software, which may be difficult to compile on the MyRio under LabView. Instead, a 'brute force' method, was used. This method consisted of a number of comparators, and depending on the localisation of the system, numerical values for the locations of the surrounding walls would be output directly. It must be noted that for each region in the map with different possible solutions, another case had to be added, with different output values. This means that a lot of time had to be invested to design case statements for a particular map, and this is not a general purpose solution. In addition, obstacles placed in particular regions of the map could be seen to affect this system and again allow

miscorrelation of an object as a wall, so additional cases needed to be added to disable particular corrections if an object existed at particular regions within the map. This is clearly not an ideal solution, and had more time been allocated, a generalised software solution would be implemented.

There are many shortcomings in the design of the software, but a lot was learned in the process. In future, many design elements could be changed or improved.

5.2 Conclusions

Overall, the system, through design and testing has shown to be functional, and fulfils the design requirements. All of the design procedures and the functionality of the system have been outlined and evaluated, and limitations in the design of the system have been outlined. There are a number of improvements which could be made to the existing design, the majority of which have been outlined, in section 5.2. Improvements to be made in some of these areas are due to shortcomings of the architecture of the system itself, while others are due to limited time, so short-term, half functional structures were implemented.

Despite these shortcomings, the system is still functional, and due to the work established in this project, there are a number of improvements and extensions to this project which could be made, in future. Some of these improvements will be outlined in section 5.3.

5.3 Future Work

There are many potential improvements for this vehicle. In many areas, there are incremental improvements to be made, but there are many instances where it may be preferable to overhaul the methods which were used. These will be outlined.

Ultrasonic sensors served as the basis for measuring information about the environment around the vehicle. An issue with the sensors used, were that the sensors did not allow low level control, and as such only gave simple distance measurements along one axis. These sensors had inbuilt signal conditioning that limited their functionality. It may be possible to use sensors with custom signal conditioning, such that (something close to) the source signal received by the ultrasonic sensors can be read (directly) by a microcontroller. By knowing the relative placement of ultrasonic sensors, being able to read the waveform of reflected sound waves from the ultrasonic sensors, would give sufficient information to generate a 2D boundary of objects around the vehicle, in a plane (albeit to a low resolution), limited to measurements in close proximity to the vehicle. It is the interference of sound waves, with respect to time, that is the mechanism which allows this.

In addition, or perhaps alternatively, LIDAR may be used. The stationary type LIDAR, detailed further in the research chapter 2 of this document would provide minimal functionality improvement over existing ultrasonic sensors, and given their excessive cost per sensor, they should not be used. Additionally, interference cannot be used to generate a boundary around the vehicle at this (relatively large) scale for light.

LIDAR, the kind attached to a spinning motor with angular tracking, would allow generation of a boundary around the vehicle, using discrete data values output by the LIDAR. This would perhaps be the most suitable sensor for environment mapping in a plane. A large variability in software architecture is available in terms of design, and data analysis. LIDAR would allow simple features of the environment to be known, such as the direction of the nearest object. For

example, in pseudocode, the system could query “Are there any objects within 2m, between angles 40° and 50°?”. Using queries such as this, the system should be capable of localising itself within a map, with a method similar to that used in this project with ultrasonic sensors.

If more complex algorithms are used, the system may be capable of mapping out the environment without a preexisting map, and additionally, could localise itself within the map, using surface features of the environment, and not necessarily relying on past known locations. These methods should be considered a long term solution, and would be implemented after a simple implementation of spinning LIDAR was already implemented. This is due to the fact that the scope and complexity of data analysis, mathematics and statistical analysis methods involved in this kind of implementation is very extensive and time consuming.

The use of a different microcontroller may be useful. The only primary benefit of the used MyRio, over other microcontrollers, is its FPGA co-processor. Setting this aside, other reasons for using a different microcontroller will be herein outlined.

There are other microcontroller platforms which have more processing power than the MyRio, which may allow the foundation for more complex data analysis techniques that may not be possible on the MyRio. Research investigating complex LIDAR analysis commonly utilises multi-core desktop processors or GPUs, so it is unlikely that the MyRio would be sufficient for these methods. Additionally, other platforms, such as the Raspberry Pi, are intermittently updated, and will offer better long-term support and upgradability in future.

The difficulty in implementing codebases other than that which is written for LabView on the MyRio, severely limits what software can be written implemented on the vehicle currently. For example, there is a lot of open-source code written specifically for aiding development of robotics platforms. The amount of time spent developing software for the vehicle may be significantly reduced if this kind of code could be used, and more complex designs could be designed more quickly. Additionally, an FPGA could still be implemented with another microcontroller, but initial system design, with respect to this feature would consume additional time and resources to implement.

For the drivetrain design of the system there are a number of possible improvements to be made to be more general purpose. For nonuniform or bumpy surfaces, it would be useful to consider using ackermann steering, especially if the system needs to be used at higher speeds, or if it is to be used in a more spacious environment, as the main reason mecanum wheels were used, were for their maneuverability. The mecanum wheels have simple control, however they do not perform well on nonuniform surfaces, but this can be compensated for, to some extent, if the position and angle correction of the system is sufficient. Modification of this aspect of the system is necessary only if the system needs to be able to drive in different conditions.

If the system is to be driven at higher speeds, it would be useful to change the motor type being used. The brushed motors used are not suitable for being driven at high speed. If low speed control and position control are needed, the permanent magnet brushless DC motor (PMBLDC motor) would be ideal, but brushless AC (induction) drives are sufficient if this isn’t the case.

The current basic control system, while effective, simply allows the vehicle to travel in straight lines to a destination, where it must stop, before continuing. There are two primary improvements which could be made to this system. Firstly, to design the system to be able to travel through a destination, without stopping, and maintaining a particular speed through a setpoint. Secondly, to allow paths which aren’t straight. The first could be solved by specifying final conditions for a move to a setpoint, such as a final velocity. The second could be solved by having a

set of preset paths with tuneable parameters for curvature. There are likely many solutions to improving the control system, so this could be investigated and tested further.

The path selection algorithm could be improved, by utilising a much higher resolution map, which doesn't try to simplify the map to a discrete set of locations in which the vehicle can exist, but considers like a continuous region with a boundary. There is a wide body of research on various algorithms which find suitable paths through complicated, mapped regions. One of these could be used to do this. It may be possible to integrate this kind of path planning system with the control system, such that the planned path is the exact path which the vehicle follows, using information about the curvature of the path to determine the speed of the vehicle.

Other software improvements and architectural changes could be made, especially given more extensive research in the future. For example, digital neural networks show promise to provide novel means of improving system performance, while minimising the need to build increasingly complex algorithms.

References

[1]National Instruments. (2017). NIARC 2017 Task and Rules Documentation. [online] Available at: http://australia.ni.com/sites/default/files/Task%20and%20Rules%20NI%20ARC%202017.pdf [Accessed 13 Apr. 2017].

[2]Ni.com. (2016). Getting Started With LabVIEW FPGA. [online] Available at: http://www.ni.com/tutorial/14532/en/ [Accessed 29 Jul. 2017].

[3]Zone.ni.com. (2014). Synchronizing I/O in Single-Cycle Timed Loops (FPGA Module) - LabVIEW 2014 FPGA Module Help - National Instruments. [online] Available at: http://zone.ni.com/reference/en-XX/help/371599K-01/lvfpgaconcepts/fpga_sctl_and_synchro/ [Accessed 29 Jul. 2017].

[4]Kim, S.-H., 2017. Electric Motor Control: DC, AC, and BLDC Motors, Elsevier Science Ltd.

[5]Bianchi, N., Bolognani, S. and Luise, F., 2004. Potentials and limits of high-speed PM motors. IEEE Transactions on Industry Applications, 40(6), pp.1570-1578.

[6]Singh, G.K., 2002. Multi-phase induction machine drive research—A survey. Electric Power Systems Research, 61(2), pp.139-147.

[7]Parekh, R., 2003. AC Induction Motor Fundamentals. Microchip Technology Inc.

[8]Keeping, S. (2013). An Introduction to Brushless DC Motor Control. [online] Digikey.com.au. Available at: https://www.digikey.com.au/en/articles/techzone/2013/mar/an-introduction-to-brushless-dc-motor-control [Accessed 20 Aug. 2017].

[9]Brown, W. (2002). Brushless DC Motor Control Made Easy. [online] Microchip. Available at: http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf [Accessed 29 Jul. 2017].

[10]User, S. (2016). Motor tuning. [online] Avdweb.nl. Available at: http://www.avdweb.nl/solar-bike/hub-motor/permanent-magnet-dc-hub-motor-tuning.html [Accessed 21 Sep. 2017].

[11]Mouton, J. (2006). Brushed DC Motor Basics. [online] Microchip. Available at: http://www.microchip.com/stellent/groups/SiteComm_sg/documents/DeviceDoc/en543041.pdf [Accessed 28 Jul. 2017].

[12]SlamTec. (2016). RPLIDAR A1. [online] Available at: http://bucket.download.slamtec.com/e9e096e9d9f30205d665260abe2cfb0c2dd62efa/LD108_SLAMTEC_rplidar_datasheet_A1M8_v1.0_en.pdf [Accessed 12 Sep. 2017].

[13]Wang, J. (2013). How Ultrasonic Transducers Work - Beijing Ultrasonic. [online] Beijing Ultrasonic. Available at: https://www.bjultrasonic.com/ultrasonic-technical-info/how-ultrasonic-transducers-work/ [Accessed 1 Aug. 2017].

[14]Berg, R. (2015). Ultrasonics. [online] Encyclopedia Britannica. Available at: https://www.britannica.com/science/ultrasonics [Accessed 3 Aug. 2017].

[15]Sparkfun.com. (2017). Ultrasonic Ranging Module HC - SR04. [online] Available at: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf [Accessed 16 Aug. 2017].

[16]Lowellkinetic.com. (2015). Steering. [online] Available at: http://lowellkinetic.com/wp-content/uploads/2016/02/Steering-2.pdf [Accessed 17 Aug. 2017].

[17]Choset, H. (2015). Mobile Robot Mechanism. [online] Cs.cmu.edu. Available at: https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/16311/www/s17/syllabus/ppp/Lec14-Locomotion.pdf [Accessed 17 Aug. 2017].

[18]What-when-how.com. (2017). The Ackermann Principle as Applied to Steering. [online] Available at: http://what-when-how.com/automobile/the-ackermann-principle-as-applied-to-steering-automobile/ [Accessed 30 Sep. 2017].

[19]Ripel, T. and Kresja, J. (2017). Ackermann Mobile Robot Chassis with Independent Rear Wheel Drives. Power Electronics and Motion Control Conference 2010, pp.2,3.

[20]Roboteq.com. (2015). Driving Mecanum Wheels Omnidirectional Robots. [online] Available at: https://www.roboteq.com/index.php/component/easyblog/entry/driving-mecanum-wheels-omnidirectional-robots?Itemid=1208 [Accessed 14 Aug. 2017].

[21]McInerney, I. (n.d.). Controlling Mecanum Drive. [online] Thinktank. Available at: http://thinktank.wpi.edu/resources/346/ControllingMecanumDrive.pdf [Accessed 4 Aug. 2017].

[22]Tran, L. and Hien, L. (2017). Robot controller using Omni wheels running straight direction. 7th Vietnam Conference on Mechatronics, p.451.

[23]Hess, W., Kohler, D., Rapp, H. and Andor, D. (2017). Real-Time Loop Closure in 2D LIDAR SLAM. [online] Available at: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45466.pdf [Accessed 30 Aug. 2017].

[24]Hunt, G. (2017). How To Select a Brushless Motor. [online] Micromo.com. Available at: https://www.micromo.com/media/wysiwyg/Technical-library/Brushless/Brushless_How%20to%20select%20a%20DC%20Motor_WP.pdf [Accessed 18 Aug. 2017].

[25]Zone.ni.com. (2012). Using a Host VI to Communicate with the FPGA Target (FPGA Module). [online] Available at: http://zone.ni.com/reference/en-XX/help/371599H-01/lvfpgaconcepts/pfi_comm/ [Accessed 18 Aug. 2017].

[26]Zhao, S. and Farrell, J. (2017). 2D LIDAR Aided INS for Vehicle Positioning in Urban Environments. BCOE Research, pp.1-4.

[27]Bianchi, N., Bolognani, S. and Luise, F., 2004. Potentials and limits of high-speed PM motors. IEEE Transactions on Industry Applications, 40(6), pp.1570-1578.

[28]Singh, G.K., 2002. Multi-phase induction machine drive research—A survey. Electric Power Systems Research, 61(2), pp.139-147.

[29]Muetze, A. and Tan, Y.C., 2007. Electric bicycles-A performance evaluation. IEEE Industry Applications Magazine, 13(4), pp.12-21.

[30]Parekh, R., 2003. AC Induction Motor Fundamentals. Microchip Technology Inc.

[31]Sathyan, A., Milivojevic, N., Lee, Y.J., Krishnamurthy, M. and Emadi, A., 2009. An FPGA-based novel digital PWM control scheme for BLDC motor drives. IEEE transactions on industrial electronics, 56(8), pp.3040-3049.

[32]Kim, S.-H., 2017. Electric Motor Control: DC, AC, and BLDC Motors, Elsevier Science Ltd.