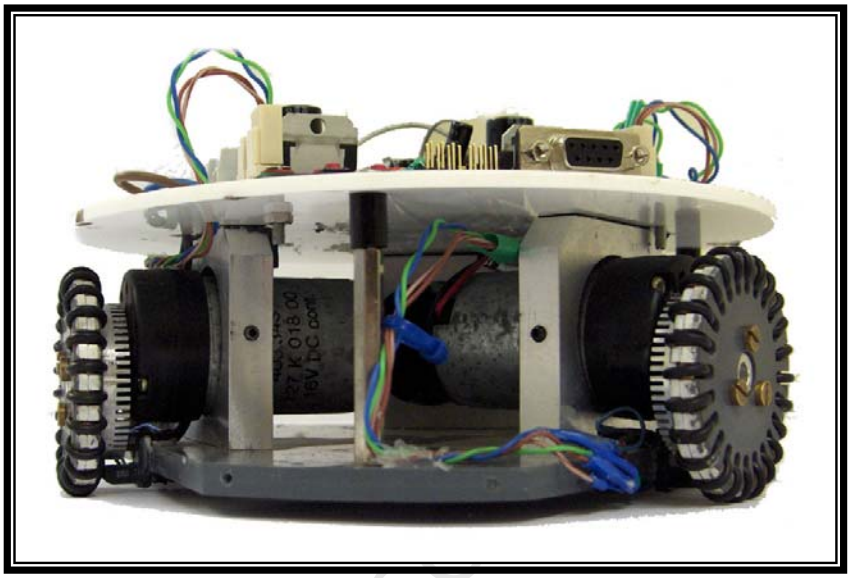
Omnidirectional Robotic Platform



The Control of an Omnidirectional Robotic Platform for use in Robot Soccer

Prepared By

Sally-Ann Levesque

Prepared for:
Graeme McPhillips, Stephen Marais and Brandon Reed
Robotics and Agents Research Lab
Department of Mechanical Engineering
University of Cape Town
February 2008

Declaration

This dissertation is submitted in complete fulfilment of an M.S.C (Eng) (Mechanical Engineering) at the University of Cape Town.

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. Each contribution to, and quotation in this dissertation from the work(s) of other people has been attributed, and has been cited and referenced.

I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.

Sally-Ann Levesque
Student number: LVSSAL001
18 February 2008

Acknowledgements

The author would like to thank the following parties:

The National Research Foundation (NRF) for financial assistance.

The Harry Crossley Foundation for financial assistance.

My supervisors, Graeme McPhillips, Stephen Marais and Brandon Reed
for their invaluable assistance.

Tristan Phillips

Kate M^{c}Williams

Tracy Booysen

Jo-ann and 'the group'

My family for all their support

Opinions expressed and conclusions arrived at in this dissertation are those of the author and are not necessarily to be attributed to the NRF or the Harry Crossley Foundation

Summary

The University of Cape Town competes in a national robot soccer competition. Teams of five small robots compete in the game of soccer without any human intervention. The robots are controlled by the artificial intelligence on a host computer connected to an overhead imaging system. The host computer controls the robots by sending them instructions via wireless communications. The robot soccer platform calls for the integration of electronic, mechanical and computer technologies and provides an exciting area for research.

UCT first competed in the robot soccer competition in 2003, using differential drive robots designed by Graeme McPhillips. Research has shown that in the international robot soccer competition, teams are replacing their differential drive robots with omnidirectional robots – robots which can move in any direction without first changing their orientation to face the direction of motion. These robots have proved to be highly manoeuvrable and the winning teams in the small robot league are consistently those that use omnidirectional robots. In 2004, Craig Inman-Bamber designed and implemented UCT's first omnidirectional robot platform. It is this platform that this dissertation is concerned with controlling. Electronic components were designed and implemented and software code written to control the robot in an omnidirectional manner. The final robot is shown in Figure i.

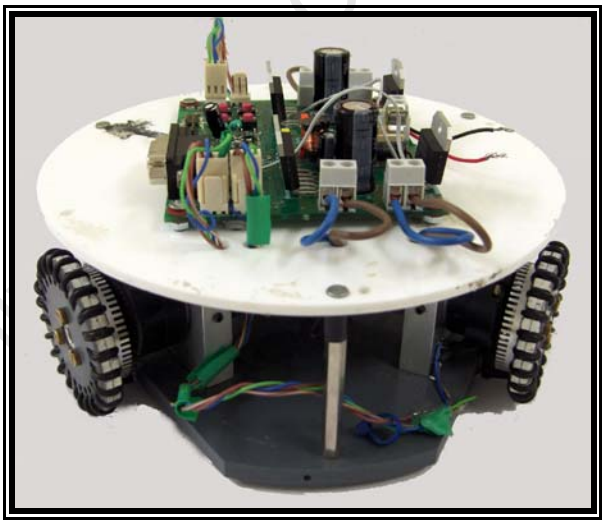


Figure i: Omnidirectional robot discussed in this dissertation

Central to all omnidirectional robots are the wheels. Omnidirectional wheels have rollers mounted perpendicular to the wheel hub which allow for motion in any direction rather than the single line of motion allowed by conventional wheels. Mr Bamber's platform consists of three omnidirectional wheels spaced 120° from each other. By driving the three wheels in the correct combination, the robot can be made to drive in any direction. Figure ii shows one of Mr. Bamber's omnidirectional wheels and Figure iii shows the robot's wheel base.

Figure ii: Omnidirectional wheel

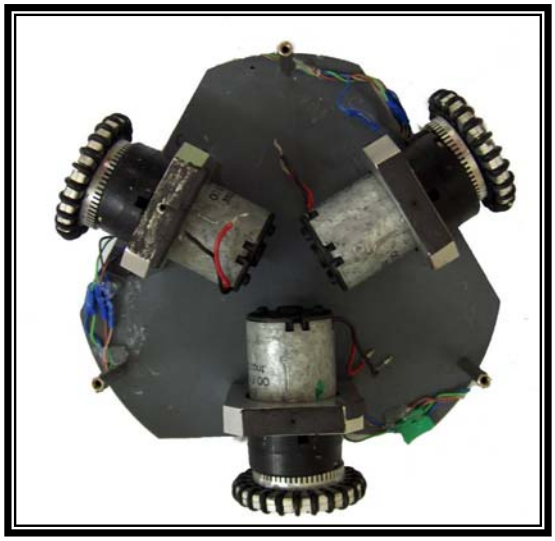


Figure iii: Omnidirectional wheel base

A Freescale DSP56F8323 DSP was used to control the robot. The DSP controlled three h-bridges to control the motor speeds and received signals from shaft encoder circuitry to determine the speed of the wheels. Circuitry to enable communication with the robot via serial communication was also added as was circuitry for power regulation.

Two versions of the robot were designed. One version made use of a DSP development board with peripheral circuits made on veraboard. The second version of the robot had the DSP and all peripheral electronics mounted on a single, purpose built, printed circuit board (PCB). Due to a malfunction of the PCB, all testing of the robot system was done on the first, development board based version of the robot.

To control the robot, kinematic equations were derived relating the speed and direction of the robot to the speed of the individual wheels. Wheel speed control was implemented on the robot before the kinematic equations were used to achieve basic robot platform control. At this level, the robot can be given a speed and direction in which to travel or a speed at which it should rotate. For small distances this allows the robot to move in a lateral direction while rotating. However over larger distances the path of the robot begins to curve.

The highest level of control allows the robot to move in a straight line while rotating. To achieve this, 'advanced', platform control, the motion of the robot was broken up into small iterations. For each iteration the robot moves a small distance, rotating while travelling in the direction of motion. The orientation of the robot is updated at the end of each iteration so that it will travel in the correct direction relative to the real world. The amount of correction needed is relative to the speed of rotation, the time of the interval, and a correction factor Q. The value of Q was determined experimentally, as was the length of the time interval. The implementation of advanced motion was successful at high lateral speeds. It was determined that the value of Q was dependent on the lateral speed. Further tests are recommended to investigate this relationship.

All control of the robot is done on the DSP. The host computer sends commands to the robot via serial communication. The robot can also be controlled using a joystick.

Both advanced and basic control was implemented on the robot platform. The robot can move in an omnidirectional manner. Testing was done to confirm this. The printed circuit board successfully controlled the robot but a malfunction prevented extensive testing of the design. It is the belief of this author that the omnidirectional platform will serve the UCT robot soccer team well and lead it to victory in future tournaments.

Table of Contents

List of Figures

Glossary of Terms

Differential drive	- vehicle system consisting of two independent drive mechanisms
Holonomic	- able to be controlled in all directions
Omnidirectional	- able to perform both lateral and rotational movements without reorienting
Veraboard	- prototyping board for electronics

List of Acronyms

ADC	-	analogue to digital converter
AI	-	artificial intelligence
DSP	-	digital signal processor
GT16	-	Freescale MC68HC908GT16 microprocessor
IC	-	integrated circuit
LED	-	light emitting diode
MR16	-	Freescale MC68HC908MR16 microprocessor
PCB	-	printed circuit board
PVC	-	polyvinyl chloride
PWM	-	pulse width modulation
RF	-	radio frequency
SCI	-	serial communications interface

1 Introduction

‘The ultimate goal of RoboCup is to develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer by 2050’.[1]

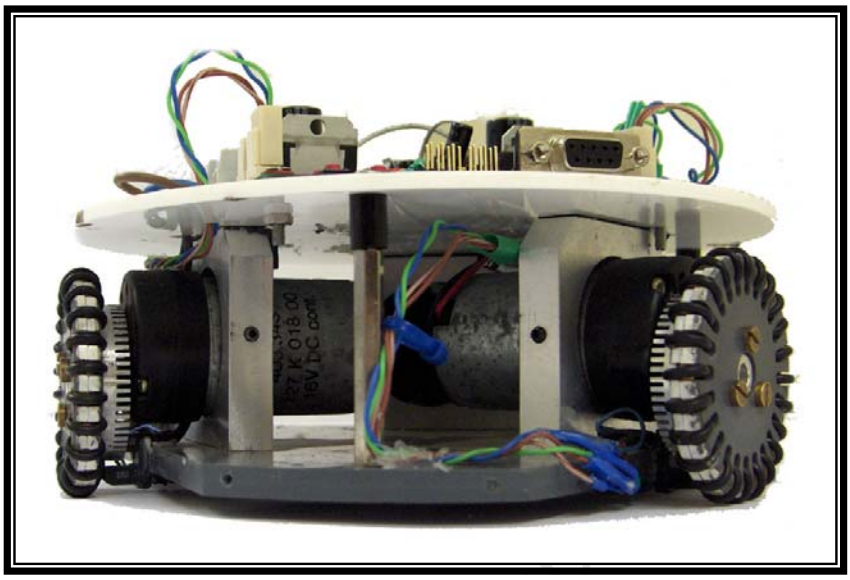


Figure 1: Omnidirectional robot

This report describes the control system of an omnidirectional robotic platform to be used in the University of Cape Town (UCT) robotic soccer team. An existing omnidirectional platform was integrated with a digital signal processor (DSP) and peripheral electronics to control the robot. The platform was controlled using a DSP development board and peripheral electronics. A dedicated printed circuit board was also designed consisting of the DSP and all peripheral electronics necessary to control the robot. The robot is omnidirectional, meaning that it can perform both lateral and rotational movements without reorienting itself. The robot receives commands via a serial link from a PC. Figure 1 shows the final robot.

1.1 Robot Soccer

RoboCup is an international organization which aims to ‘promote AI, robotics, and related fields’ [1]. To provide a platform for this research, the organisation holds tournaments in which robots compete autonomously in the game of soccer. Since each team of robots must be able to compete without any human interaction, the design of a robot soccer team requires the integration of a number of different fields of technology (robotics, sensors, mechanics, artificial intelligence, software development, image processing and communications). Competing in the league encourages researchers to look for ways to further develop each of these technologies as well as for improved means to fuse the technologies together. RoboCup has five soccer leagues - a small size and middle size leagues as well as a humanoid league, a standard platform league, and a simulation league.

Although the apparent focus of RoboCup is on the soccer, the real pay-offs are the technologies that are developed in order to create these robots and experience gained by researchers working on them (A similar parallel could be in the many technological spin-offs

that have resulted from man's exploration of space). UCT is interested in developing robots that can be used in industrial applications such as materials handling and automated inspection. Robot soccer provides a very good arena in which to test a robotic platform since in tournament conditions, it is crucial that robots are robust, easily repairable, and can be effectively controlled. These aspects of design are also essential in industry; ways in which a robot does not meet these requirements will quickly become apparent during competition.

Figure 2: UCT's 2004 Robot Soccer Team

The University of Cape Town competes in robot soccer at a national level at an annual tournament with the Universities of Natal and Pretoria. Figure 2 shows the robots that competed in 2004 tournament. RoboCup runs a number of different leagues, for robots of different sizes. The South African tournament follows the rules of the small robot league, also called the F180 league. According to the F180 rules teams can have up to 5 robots on the field at a time. The robots must fit within a cylinder of 180mm and have a maximum height of 150mm. The robots play on a 2.3m x 2.8m field with goals on either side. The 'soccer ball' is an orange golf ball. The rules of the game are adapted from the rules for the human game of soccer. Robots are controlled by AI software on an external PC which is connected to the overhead vision system as illustrated in Figure 3. The PC communicates with the robots using wireless communication.

UCT's 2003 robot soccer team was made up of robots designed by Graeme McPhillips[2]. These robots worked well and won the tournament in both 2003 and 2004. However it was hoped that improvements on the drive system and the incorporation of a dribbling mechanism might bring the robots closer to the level of those robots competing internationally. In 2005 two undergraduate projects in the department of Mechanical Engineering were concerned with these improvements: Craig Bamber designed an omnidirectional drive system for the robots[3] while Lindsey Upsher [4] worked on a kicking and dribbling system. The control of Mr. Bamber's omnidirectional platform is the focus of this report.

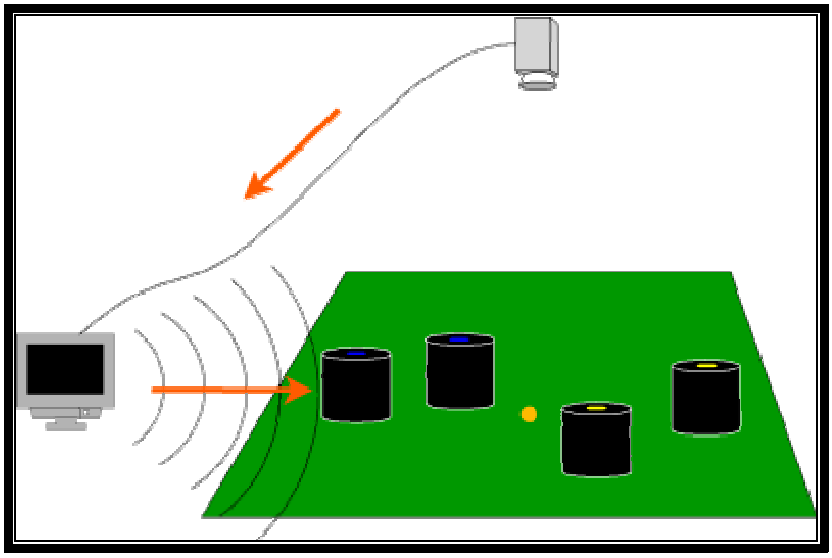


Figure 3: Interaction between overhead camera, host computer, and robots [5].

1.2 Omnidirectional Robots

Conventional rolling robots, and indeed conventional vehicles, use a drive system which allows only two degrees of freedom. To move in a specific direction, the robot must first turn to face that direction. Omnidirectional vehicles however are able to move in any direction without having to first reorient to face the desired direction of motion. This is particularly useful in the robot soccer environment as it allows for quick reactions and increased manoeuvrability.

The design of the vehicle's wheels is what gives it omnidirectional capabilities. The wheels are designed with rollers which roll either perpendicularly or at a 45° angle to the direction of rotation of the wheel hub (see Figure 4 and Figure 5). This means that although the wheel can only drive in a direction parallel to the hub, it can roll in a direction perpendicular to its hub. Since each wheel can roll in the direction of the hub or the direction of the rollers, any direction of movement is possible in a two dimensional plane.

Figure 4: A 90° omnidirectional wheel [3]

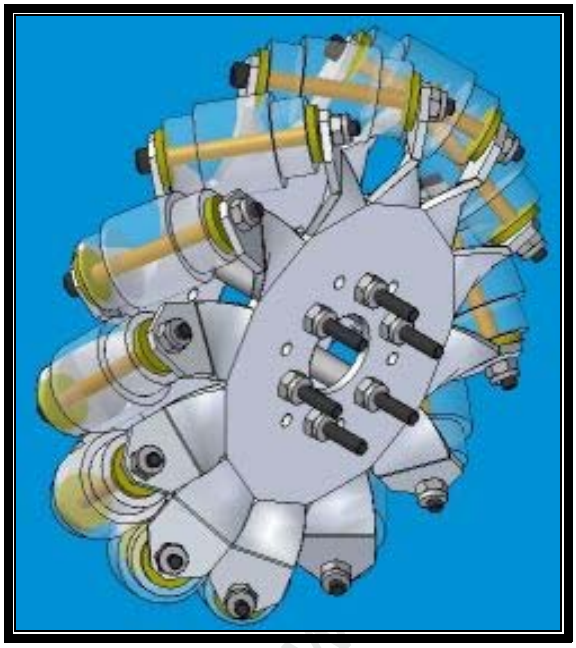


Figure 5: A 45° omnidirectional wheel [6]

A number of omnidirectional designs are possible (see Appendix A, A.2). The robot discussed in this document has three omnidirectional wheels. The rollers of the wheels are perpendicular to the wheel hub. The diagrams below (Figure 6 and Figure 7) show the base of this robot. Driving wheel B and wheel A at the same velocity results in a motion of the robot in the direction of wheel C (shown by the arrows in Figure 7). Varying the velocity ratio of the wheels results in motion in any lateral direction. Driving all three wheels at the same speed results in rotation.

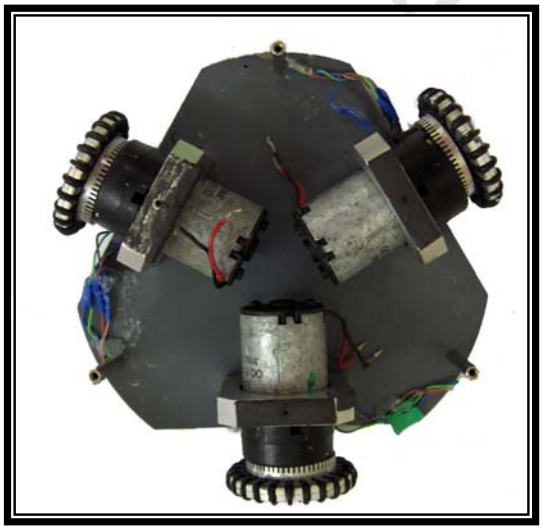


Figure 6: Omnidirectional wheel base

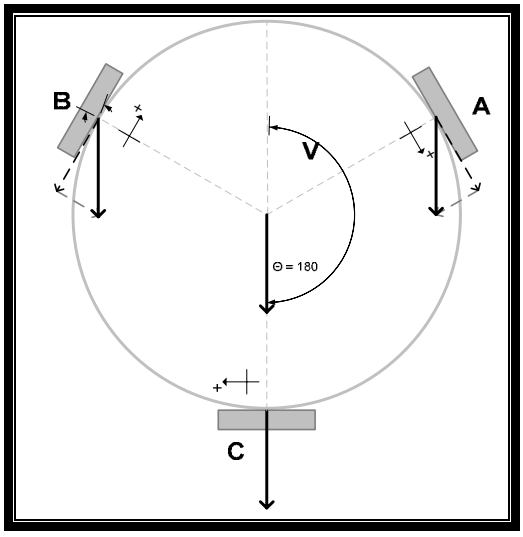


Figure 7: Diagram of wheel base

1.3 Previous Work

In 2005, Craig Bamber designed and built UCT robot soccer's first omnidirectional robot. He designed a three wheeled robot platform using 90° omnidirectional wheels which he also designed. This dissertation is concerned with the control of that robot platform. Mr. Bamber was able to implement limited control on the robot. The robot was controlled with a Freescale MR16 microprocessor, and could interpret commands to run the three wheels at specified speeds. The wheels were controlled in open loop - there was no speed feedback. A further limitation was that communications with the robot were from the controlling computer to the robot only; the robot could not acknowledge commands or send information to the host computer. A Matlab program running on the host computer calculated the wheel speeds needed to drive the robot in a particular direction and rotation.

1.4 Specifications

The aim of this project was to design, implement and test a new control system for the omnidirectional platform designed by Craig Bamber. The platform needed to be controlled so that it could perform lateral and rotational movements in any direction without first having to change its orientation. All kinematic equations needed to be performed on the robot, ie, given the speed and direction of a lateral movement and the orientation which it needed to face, the robot should be able to move as expected. Additionally a printed circuit board should be designed to accommodate all electronics needed for the control of the robot. The robot should be able to both send and receive information and interpret commands from the host computer.

2 Background

2.1 Introduction

UCT has been involved in robot soccer since 2002. In 2002/3 Graeme McPhillips developed the first UCT robots to enter the RoboCup League. In 2004 Craig Bamber developed UCT's first omnidirectional robot. It is this robotic platform that this dissertation is concerned with controlling. This chapter provides background information about the two types of robots previously designed at UCT. The information below is taken from Mr. McPhillips' Masters dissertation [2] and Mr. Bamber's undergraduate dissertation [3].

2.2 Drive System

The robots designed and built by Mr. McPhillips for the 2003 competition were based on a differential drive system; sometimes referred to as a 'turtle' robot design. The robot has two wheels, in line with the centre of gravity, and 2 skid pads for balance. To steer the robot, the two wheels are driven at different speeds. Driving the two wheels in opposite directions allows the robots to turn on the spot when necessary. Omni-directionality was not achieved with these robots since they could not move in any direction without changing their orientation. Figure 8 shows Mr. McPhillips' design.

In Mr. Bamber's undergraduate thesis, an omnidirectional robot was designed which could move in any direction without needing to turn and face the direction of motion. Such 'omnidirectional' robots have been used by a number of top teams in the RoboCup including four time winners Cornell University [7]. Mr. Bamber chose to use a three wheel design. He designed and fabricated the omnidirectional wheels, consisting of 24 rollers mounted

perpendicular to the hub. Figure 9 shows the robot made by Mr. Bamber. Figure 10 shows an exploded view of the omnidirectional wheels he designed.

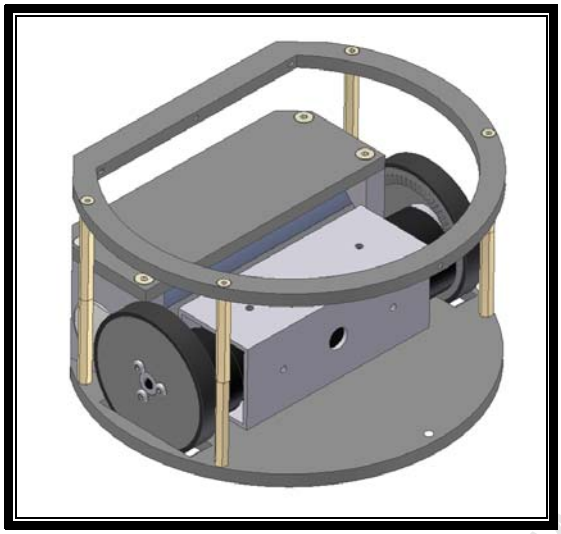


Figure 8: Robot designed by Mr. McPhillips

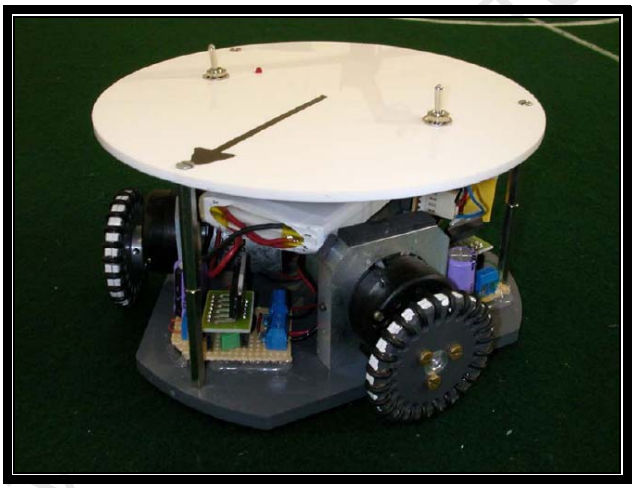


Figure 9: Omnidirectional platform designed by Mr. Bamber

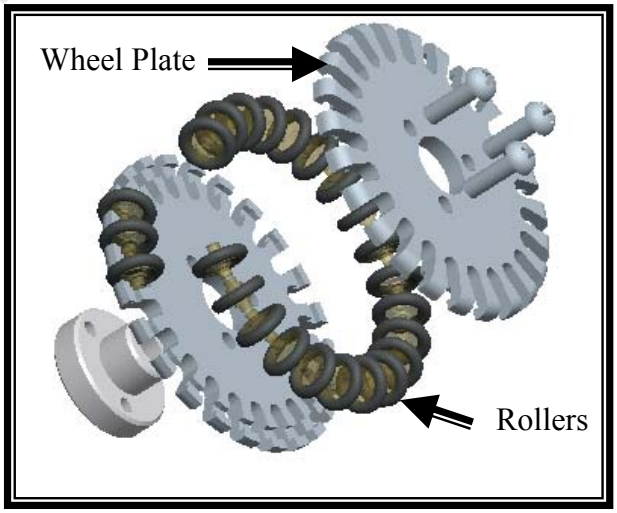


Figure 10: Exploded view of omnidirectional wheel designed by Mr. Bamber

2.3 Motors

Mr. McPhillips' and Mr. Bamber's designs used the same DC geared motors with built in gear boxes to drive the wheels. They achieved a speed of 425RPM at 16V, drawing 200mA at no load and approximately 350mA at full load.

2.4 Encoders and wheel speed control

Mr. McPhillips successfully used encoders to measure wheel speed in his robot design and it was hoped that a similar method could be used to integrate speed control into the omnidirectional design. To measure the speed of the motors, Mr. McPhillips attached a toothed encoder wheel to each drive wheel. A Fairchild H21A1 optical interrupter switch was attached across the toothed wheel to form an encoder with which to measure the wheel's rotation and thereby it's speed.

Mr. McPhillips implemented PID control to control the speed of the wheels of his robots. When he found that wheel slip was affecting the accuracy of the control he used a prefilter control algorithm to control the amount of acceleration and reduce the slip.

Mr. Bamber attempted to use the wheels themselves as encoder disks by using an optical interrupter switch to detect the spaces between rollers (see Figure 11). He had only moderate success and found that the resolution provided by this method was not sufficient for effective wheel speed control. He was not able to successfully implement a control algorithm partly due to the lack of resolution on his wheel measurement method and partly due to time constraints.

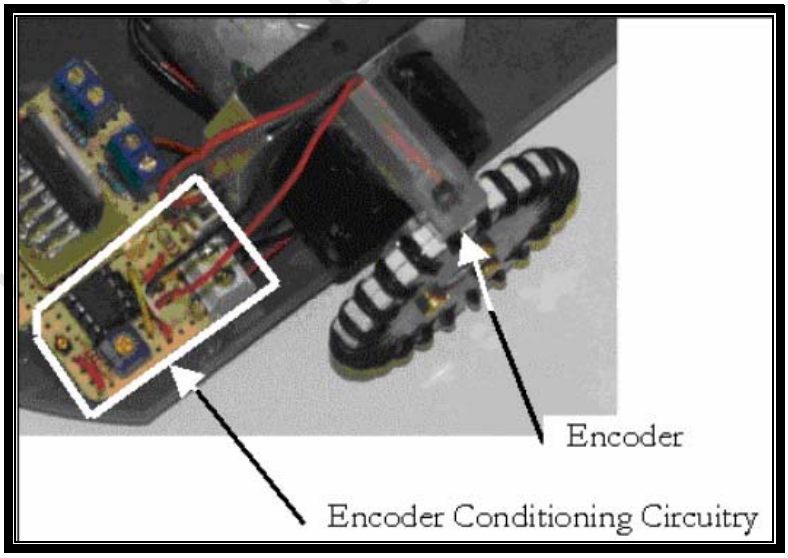


Figure 11: Mr. Bamber's use of the wheels themselves as encoder disks

2.5 Control

Mr. McPhillips used a Texas Instruments TI320F243PGEA DSP to control his robots. It is a 20Mhz processor with 36 IO pins. Its peripherals include a 16bit timer channel and a 10 bit, 8 channel ADC. It has an SCI interface and 16Kb of internal flash memory [8].

Mr. Bamber used a Motorola MC68HC908MR16 microcontroller to control his robot. The MR16 is an 8Mhz processor which has 36 IO pins of which 2 can be used for timer channels and 10 for the ADC. It has an SCI interface. It has 16Kb of internal flash memory.

2.6 Communications

Mr. Bamber used the same serial communications protocol as Mr. Phillips. The host computer, which controls the actions of the robots, directs the robots by sending them commands via a wireless communication system. The robots communicated using off the shelf radio technology (Linx HP RF modules). The controller computer had an RF (radio frequency) transmitter which it communicated with via a serial port. This sent out commands which were detected by RF units on the robots. A communications protocol was developed so that the computer could send commands to specify how it should travel. Some error checking was built into the protocol to make it more robust.

The communication system worked well most of the time but since the frequency spectrum it was using was close to that used for cellular telephones, interference from cellular phones in the vicinity was very noticeable when no error checking was implemented. Even with error checking, the interference could cause the robots to behave unexpectedly

Both Mr. McPhillips and Mr. Bamber implemented one way communication only. The robot could receive commands from the host computer but not send acknowledgements or data.

Both communications protocols consisted of commands from the host computer as to the speed at which each wheel should be driven. This means that, for Mr. Bamber's omnidirectional robot, calculations converting the robots speed, direction and rotation into wheel speeds were done using a Matlab program run on the host computer. This is not an ideal system as during a robot soccer match it uses up processing time of the host computer which is already doing image processing and AI calculations. This could potentially slow down the reaction time of the host computer and therefore the robot soccer team. A more suitable arrangement is for the host computer to send the robot speed, direction and final rotation commands and for the onboard controller to convert these into wheel speeds which it then implements. This is the arrangement implemented in this dissertation.

2.7 Power

Mr. McPhillips' robots used a 16V battery pack made up of 14, 1.2V, Nichol Cadmium batteries to supply the motors. The power for the electronics was tapped off this pack at 8V which meant that some of the batteries were discharging faster than the other resulting in some batteries having a shorter lifespan than others.

Mr. Bamber used an 11V Lithium polymer battery for powering his robot. Although used successfully, these batteries proved too expensive to implement on a team of 5 robots.

Mr. McPhillips suggested investigating alternative power solutions, including having separate battery packs for electronics and drive motors and having a single battery voltage which is either stepped up for the drive motors or stepped down for the electronics. He had found

alternative batteries (7.4V Lithium Ion) which were more suitable in terms of price, size and ease of charging and suggested that these could be integrated into the new system.

2.8 Concluding Remarks

Mr. McPhillips successfully designed and built UCT's first robot soccer team. The robots had differential drive systems which did not allow for the kind of maneuverability that omnidirectional drives do. Mr. Bamber designed and manufactured UCT's first omnidirectional soccer robot. The robot platform proved very effective but the control of the robot platform was based on the host computer rather than the robot itself. Also, effective wheel speed control was not implemented, hindering the accuracy of the robot. The battery packs used by both Mr. Mc Phillips and Mr. Bamber were not ideal, and an alternative battery was required. Both projects used the same wireless communications interface which proved very sensitive to interference. Mr Bamber's robot platform was a success and could be used as the platform for this project. However, the control system of the robot could not meet the objectives of this project and would have to be replaced.

The Chapter which follows gives an overview of the system which replaced Mr. Bamber's control system.

3 Final Design Solution

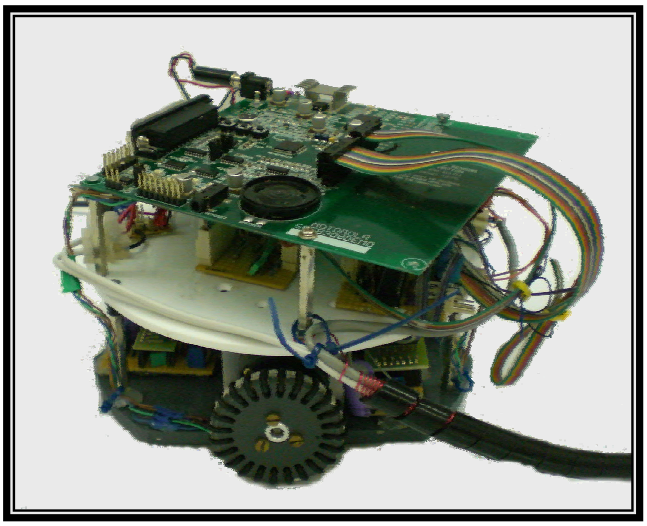


Figure 13: Development board based robot

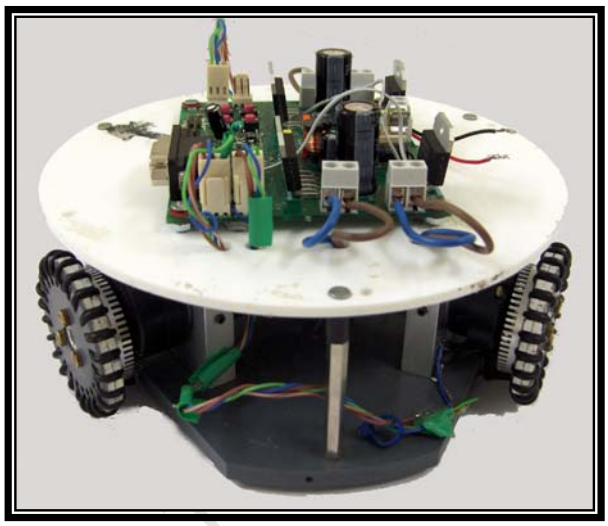


Figure 12: PCB based robot

3.1 Iterative Design

Two robot control systems were designed. The design on which all testing was done, made use of a development board which hosted the digital signal processor. All electronics peripherals to this board were made on veraboard. Additionally, a printed circuit board was designed with the DSP and all electronics needed for the robot control on a single board. This board was used for the robot step tests but malfunctioned during final integration with the robotic platform. Since the function of both designs was the same and the same code was used to control both designs the two are described here together. All explanations begin with a description of the PCB system. In areas where the development board solution differed from the PCB solution a description of the development board solution is included. Both robots and their controlling electronics are shown in Figure 14, Figure 15, Figure 16 and Figure 17 below.

3.2 General Description of Robot

The robot consists of the physical robotic platform, the electronic circuitry and software code elements. Figure 18 on page 13, shows a breakdown of the parts that make up the robot. The physical robot platform is that designed by Craig Bamber. To control this robot, a DSP board with controlling electronics was used. The code written to control the robot allows it to accept commands from a host computer via a serial link. A move command tells the robot what direction, and at what velocity, it should move laterally or at what velocity it should rotate. Additional move commands allow the robot to move both laterally in a specified direction while rotating. A system has also been implemented whereby the robot can be controlled via a joystick plugged into the host computer. Other commands instruct the robot to perform open or closed loop step tests.

Two robot designs exist. One makes use of a development board on which the DSP is hosted, the other makes use of a single, purpose designed, printed circuit board shown in Figure 16. The circuit board consists of the following:

- The DSP and peripherals needed to run and program it
- Power circuitry to supply 3,3V from the 14,8V supply
- Four h-bridge circuits to drive the wheel motors
- Four encoder circuits to receive wheel speed information
- Circuitry to enable serial communications

The robot can also send information via serial communications. The robot acknowledges commands when received, requests additional inputs when certain commands are entered, and returns information about wheels speeds when step tests are run.

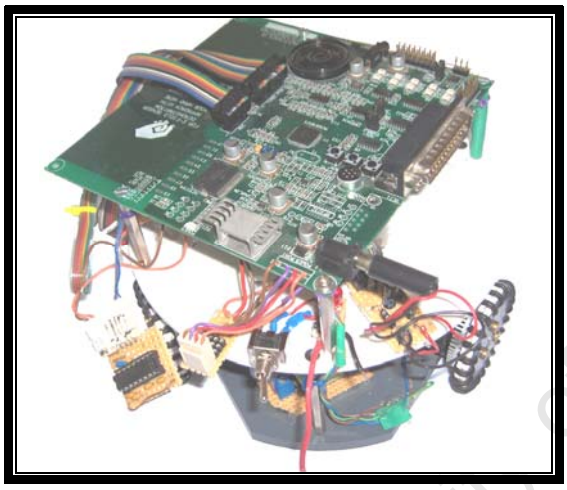


Figure 14: Development Board Robot

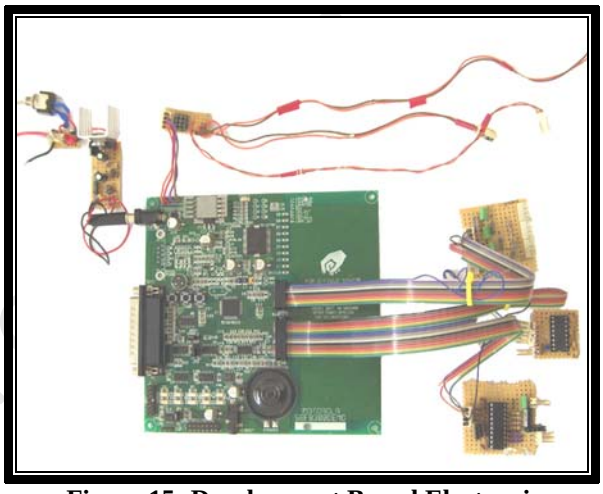


Figure 15: Development Board Electronics

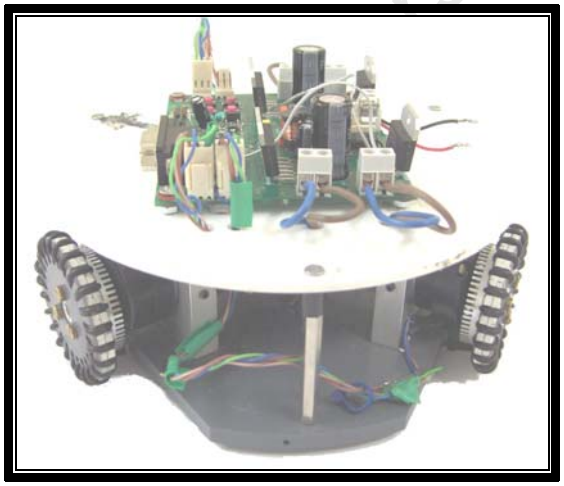


Figure 16: PCB Robot

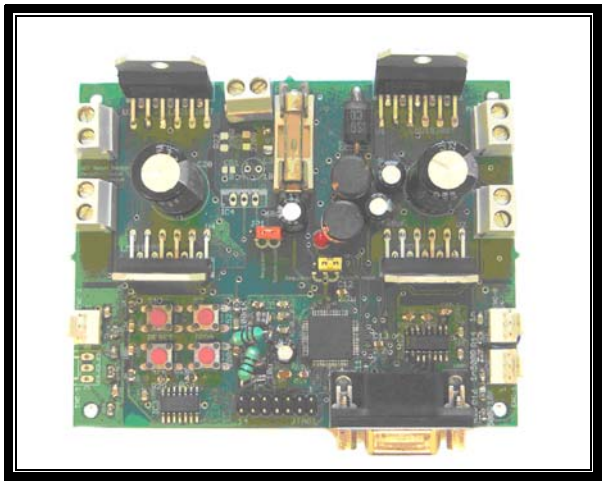


Figure 17: PCB Designed to control robot

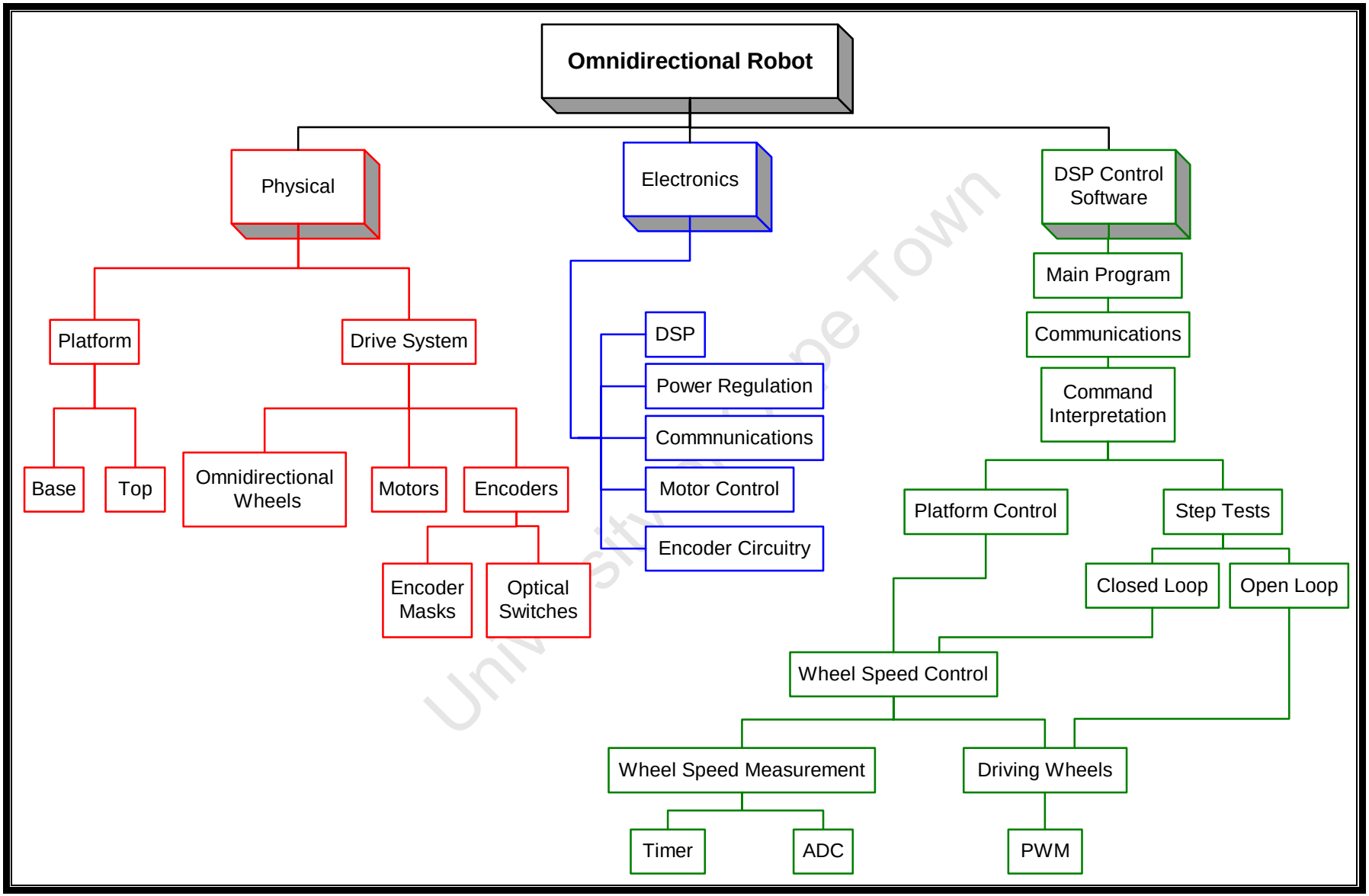


Figure 18: Top down diagram of showing the parts that make up the robot

3.3 Physical System

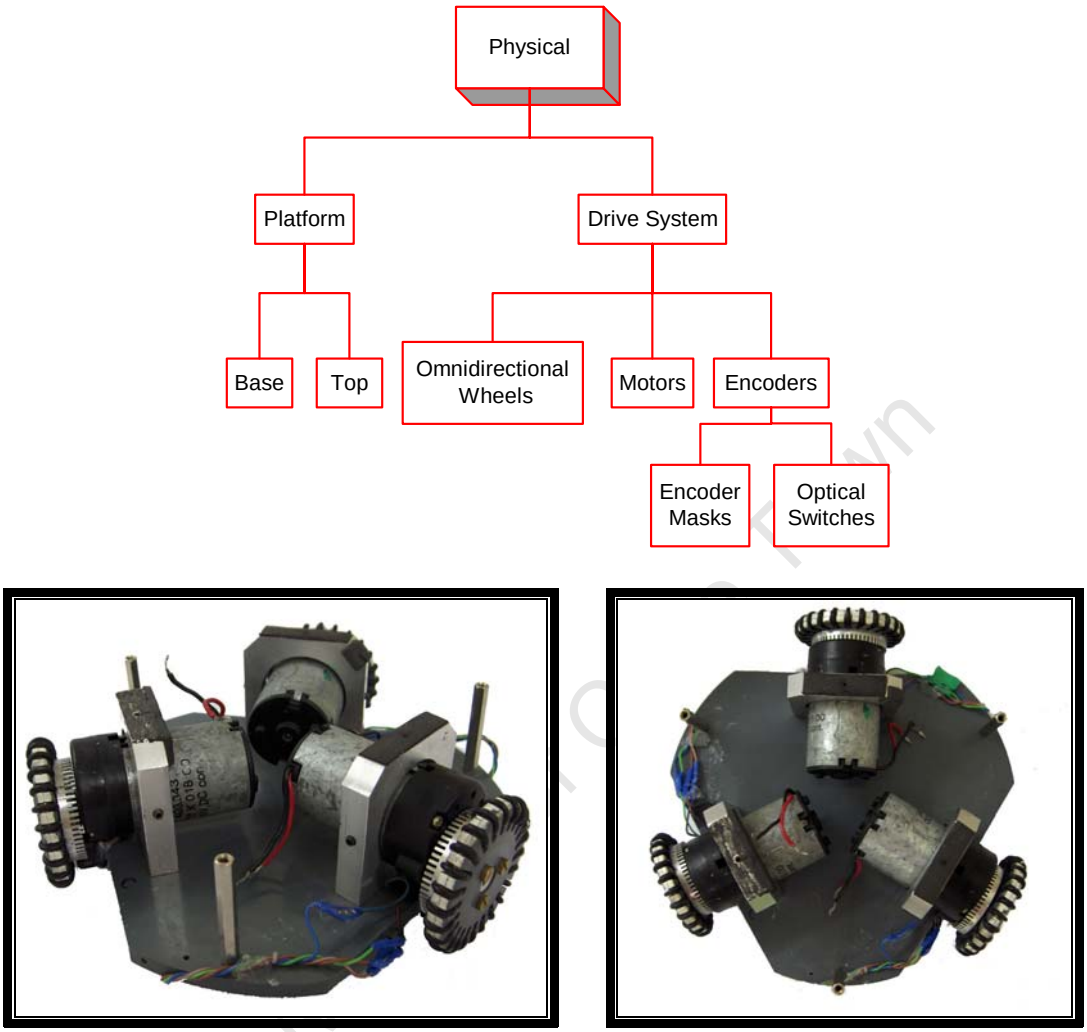


Figure 19: Robot platform designed by Mr. Bamber

Figure 19 shows the robot platform designed by Mr. Bamber. It consists of a PVC base with three motors attached to it. The motors are spaced 120° apart. Motor mountings attach the motors to the base. Attached to the motor shafts are the omnidirectional wheels and wheel encoder masks (the encoders were added to the system by the author). Metal standoffs are used to support a second PVC platform onto which the printed circuit board has been attached. The development board version of the robot had a third layer – the development board was held above the second layer using metal standoffs. The development board electronics on veraboard were all housed on the second layer except for the h-bridges which were placed on the lowest level, next to the motors. Figure 20 and Figure 21 show the PCB and development board robots.

Attached to the wheel hubs are encoder masks designed by Mr. McPhillips for wheel speed measurement. Figure 22 shows the encoder mask and Figure 23 shows the mask mounted on the wheel hub. Optical interrupter switches were mounted on the platform

in such a way that the mask interrupts the switch, providing a means to measure the rotation of the wheel.

Figure 20: Final robot with PCB

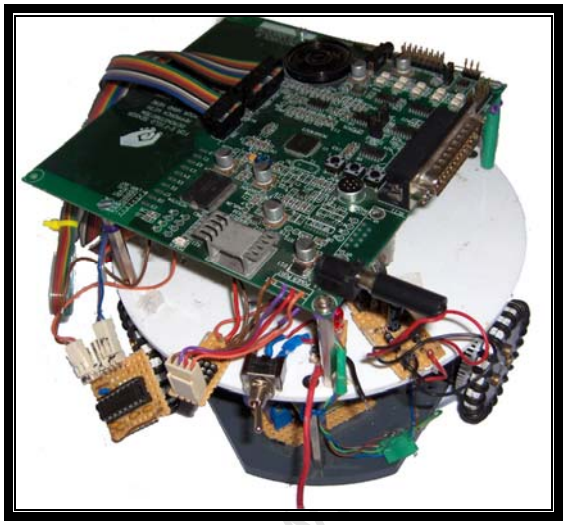


Figure 21: Development board robot with development board

Figure 22: Encoder mask

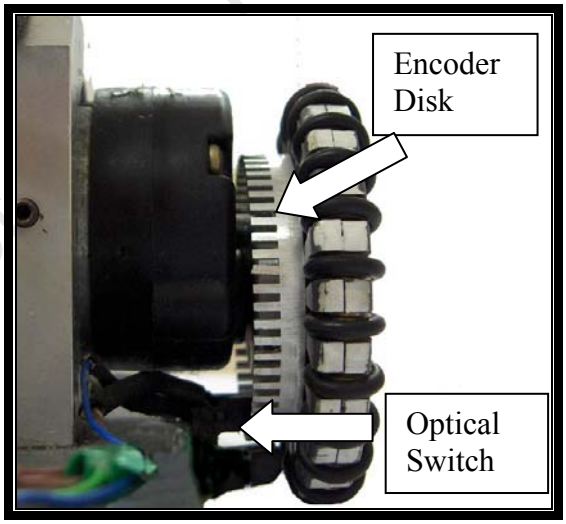
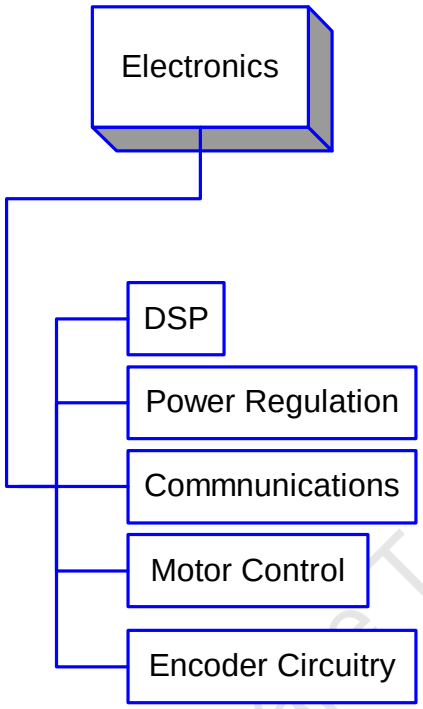


Figure 23: Encoder mask and optical switch mounted on wheel

3.4 Electronics



3.4.1 Introduction

The electronics of the robot consists of five main components:

The DSP
Power Regulation
Communications
Motor control
Encoder Circuitry

On the PCB robot, all these components are on a single, purpose designed PCB. Figure 24 shows a block diagram of the PCB based electronics. Figure 25 shows a block diagram of how the different components interact on the development board; note how all components are separate from each other. What follows is a brief description of the electronics used to control the robot. For a more detailed description see Appendix B.

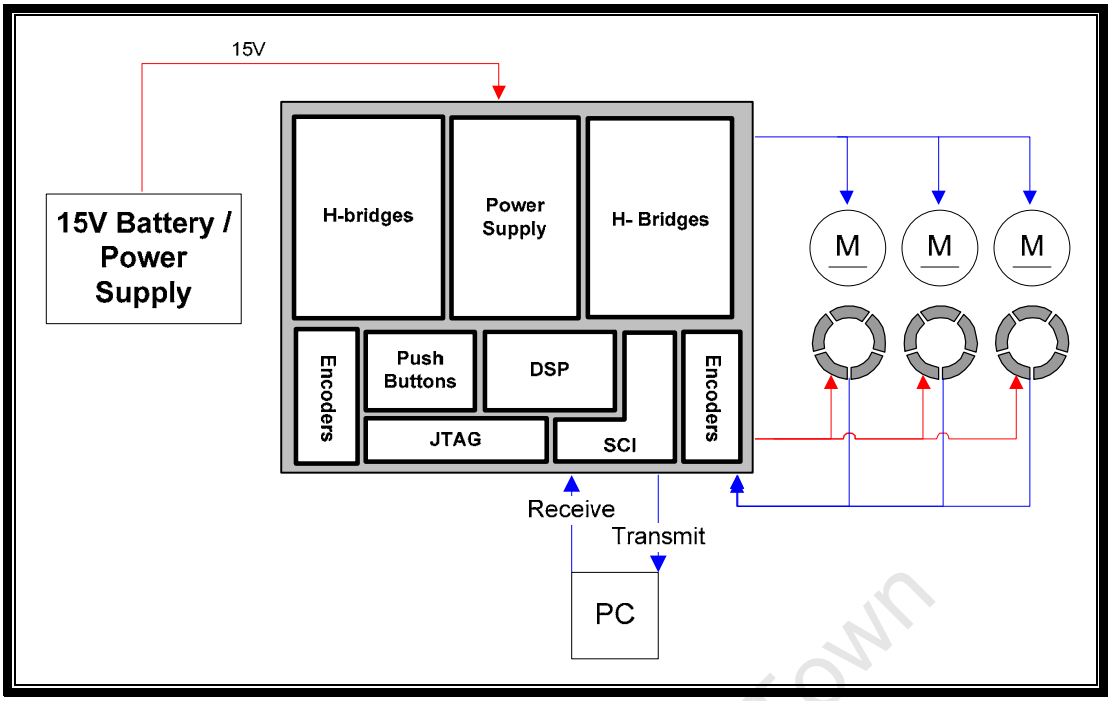


Figure 24: Block diagram of PCB based control system

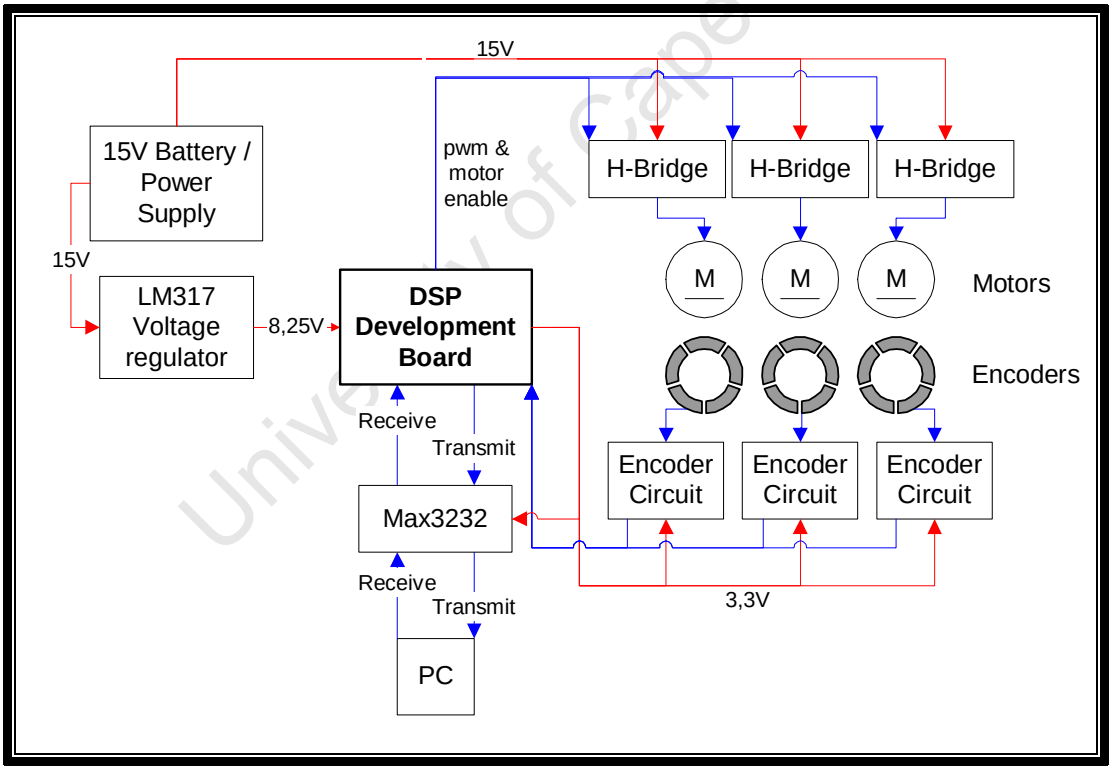


Figure 25: Block Diagram of development board based control system

3.4.2 Digital Signal Processor

The Freescale DSP56F8323 digital signal processor is the central electronic component. The choice of controller was based on availability and suitability. In order to reduce the lead time, a development board was used for the robot. A digital signal processor was used instead of a slower, less powerful microprocessor to ensure that

the controller had sufficient processing power to do the calculations needed to control the platform's movements.

3.4.3 Power

A design requirement was that the new robot be designed to operate off 7,4V lithium ion batteries. It was decided to use two of these batteries in series for the battery pack, giving a 14,8V supply. For the PCB, this needed to be regulated to provide 3,3V for the DSP and other electronics. A switch mode power supply chip, the max5035, was used to efficiently supply 3,3V from the 14,8V supply. This solution was preferred to using a more simple voltage regulator as a switch mode power supply does not waste as much power, resulting in a longer battery life.

The development board needed a 9V supply and it had onboard regulation to provide the required 3,3V and 5V supplies from this. An LM317 adjustable voltage regulator was used to provide the required voltage to the development board from the 14,8V supply.

Although the DSP's power was designed to be used with the lithium ion batteries, all tests were done using power provided via an 'umbilical cord' from a voltage regulator as a battery holder was not part of the design requirement.

3.4.4 Communications

Both the development board and the final PCB used a max3232 chip to enable communications with the serial port of the host computer. One of the DSP's two serial control interfaces (SCI) was used to allow the robot to both send and receive data. For debugging the program Docklight was used to send and receive commands to and from the robot. The joystick control program (discussed in Section 3.5.2 and Appendix E) used Python to interface with the robot. Other test programs were written in Matlab to send instructions to the robot. The physical layer of the communications was a serial cable. Given the problems experienced with the wireless communications of the previous robots, it is recommended that a more reliable serial to radio frequency converter be found for future versions of the robot.

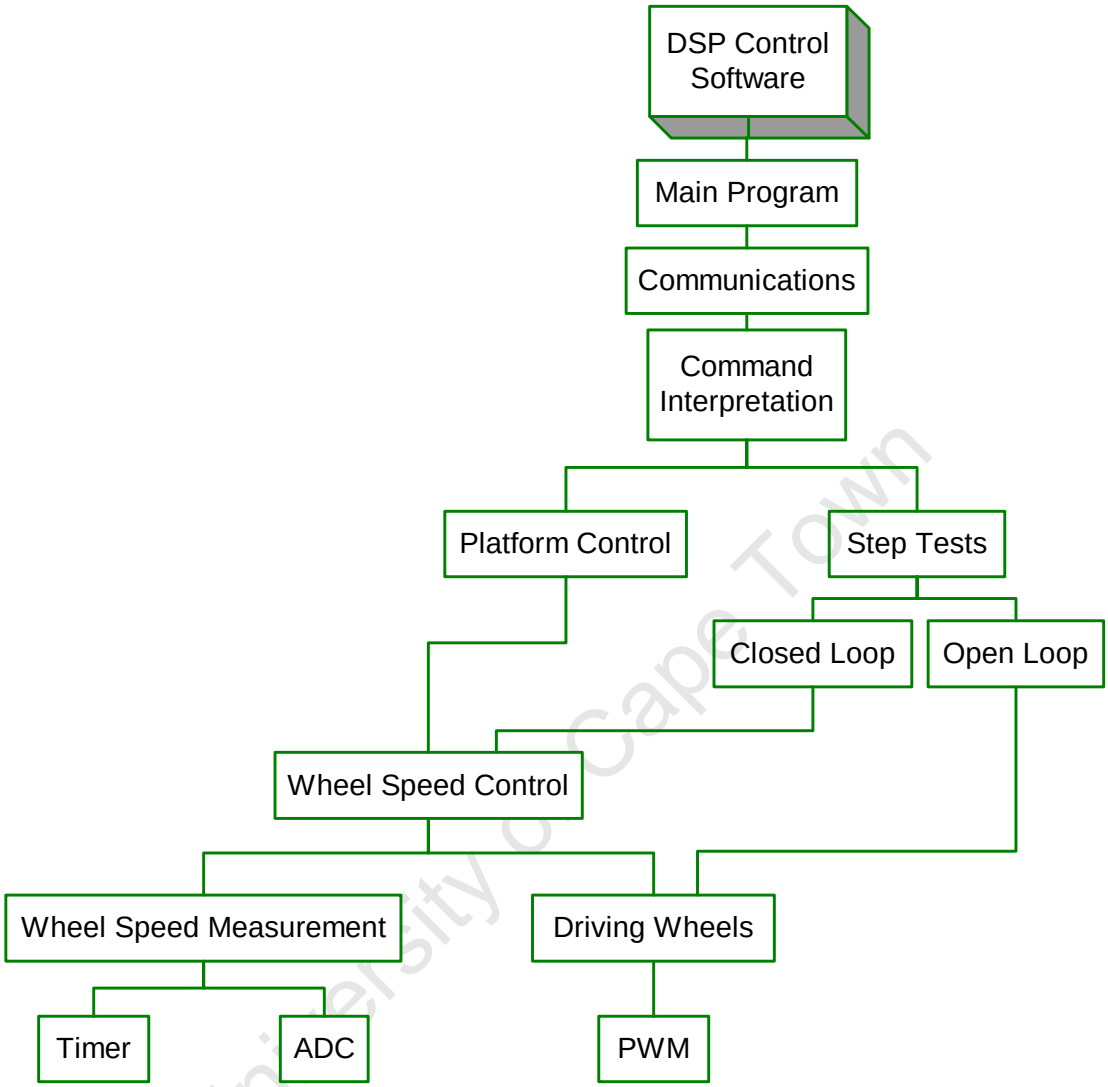
3.4.5 Motor Drivers

To control the speed and direction of the wheel motors, LMD18200T h-bridge chips were used. PWM signals are fed into the chips from the DSP, enabling control of the motors.

3.4.6 Encoders

To determine the speed at which the wheels were rotating, the wheel encoder masks described in Section 3.3 were attached to the robot. To detect the rotation of the encoder mask, an H21A1 optical interrupter switch is used. This consists of an LED and a phototransistor facing each other with gap in between. When a tooth from the encoder mask passes through the gap, the phototransistor no longer detects the light from the LED. As the wheel rotates, the signal from the encoder circuitry rises and falls, allowing the DSP to count the number of teeth which pass per time period and therefore the speed of the wheels. The unconditioned signals from the three wheel encoders are read using the DSP's ADC. The ADC's high and low level triggers are used to determine the rising and falling of each signal. This eliminates the need for comparator circuitry. The DSP's timers are used to measure the time between teeth.

3.5 DSP control software



3.5.1 Introduction

The robot is controlled through commands given on the host computer. The commands are received by the robot on its serial port. A serial cable connects the serial port of the robot to that of the host computer. A Python program was written to enable the host computer to translate joystick commands into commands to control the robot. Alternatively commands can be sent to the robot using a command terminal such as Hyperterminal or Docklight. In a robot soccer tournament the host computer will run the AI to direct the robot during play and will communicate via the SCI. The serial cable will be replaced by a wireless interface.

3.5.2 Joystick Control

The Python program *joystickControl.py* was written by the author to enable the robot to be controlled with a joystick. The program is run on the host computer, which has a

joystick plugged into its USB port (a Microsoft Sidewinder Precision2 joystick was used). The robot is connected to the host computer via a serial link.

To drive the robot laterally, the joystick is moved in the desired direction. The speed with which the robot moves is determined by how far the joystick is moved. To rotate the robot the joystick is twisted. The speed of rotation is determined by how much the joystick is twisted. This is illustrated in Figure 26 below.

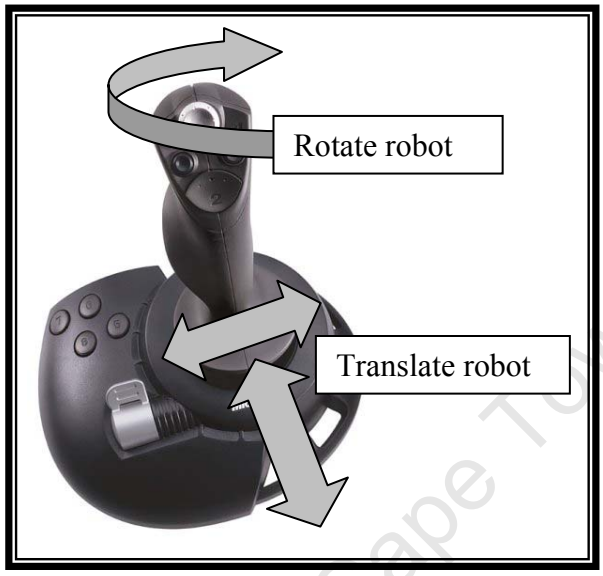


Figure 26: Joystick controls

The Python program receives commands from the joystick and converts them into commands for the robot. The commands structure used by the robot lets the robot receive information about the speed and direction of lateral movement and the speed and direction of rotation. The Python program uses the 'g' command which is explained in the next section. Videos 3a and 3b on the CD provided with this dissertation show the robot being controlled with the joystick.

3.5.3 Command Line Control

The robot can interpret a number of commands, each of which is described below. The commands are all four bytes in length. They take the format of a letter character which denotes the command to be executed followed by a number in either hex, decimal or byte format depending on the command. The number gives the parameters of the commands. Some commands, such as the move and step tests commands, ask for extra information to be entered before the command is executed

3.5.3a Run Wheel Commands

Commands 'A' through 'F' tell the robot to run the wheels in open loop, supplying a particular voltage to the motor, but not controlling the wheel speed. These commands were mainly used for initial debugging and test of concept. The command is of the form 'A-F' 'Hex number from 0 to A00'. The letter A-F denotes which wheel(s) should be run A-C run wheels A,B or C individually, D-F runs pairs of two wheels to move the robot laterally in a particular direction. The parameter which follows the letter determines the wheel speed by setting the duty cycle of the PWM controlling

the wheel. ‘000’ runs the wheel at maximum backward velocity, ‘500’ stops the wheel and ‘A00’ runs the wheel at maximum forward velocity. For example the command ‘B300’ runs the wheel B in the backwards direction at a voltage that is a 40% of the maximum voltage. The command ‘DA00’ runs both wheel A and wheel C in the forward direction at full speed. Figure 27 shows three examples of the ‘run wheel’ commands being used. Blue text is commands sent to the robot and the red text is the robot’s acknowledgements.

```
b300 Driving Wheel B
dA00 Driving In Direction 1
c700 Driving Wheel C
```

Figure 27: Examples of the run wheel commands being used

3.5.3b Wheel Control Commands

Commands ‘X’ through ‘Z’ tell the robot to control a particular wheel at a particular speed. ‘X’ controls wheel A, ‘Y’ controls wheel B, and ‘Z’ controls wheel ‘C’. A speed from -14 to 014 can be entered. The speed is measured in pulses of the encoder wheel per 30ms. For example to control the wheel speed of wheel A at a speed of 5 pulses per 30ms the command ‘X005’ is sent via the serial comms to the robot. To control wheel C in the backwards direction at a speed of 12, the command ‘Z-12’ is sent to the robot. Figure 28 shows three examples of the wheel control commands being used.

```
X005 Controlling Wheel C
Y012 Controlling Wheel B
Z-12 Controlling Wheel A
```

Figure 28: Examples of the wheel control commands

3.5.3c Move Robot Command

The ‘M’ command moves the robot in a specified direction at a specified speed for 2,4s. It is mainly used for debugging and demonstration purposes. The command has no initial parameters. Sending the command ‘M000’ puts the robot in move mode. The robot then requests what speed it should go at. This should be entered as a four digit number, the robot then requests a direction, which should also be entered as a four digit number. The speed is in encoder teeth per 30ms and should be between 0 and 14. The direction is an angle in degrees. Figure 29 shows the orientation of 0°. The angle is relative to the robot, not the field. Once the robot has finished moving, it returns a record of the three wheel speeds during the movement. Figure 30 shows an example of the ‘move robot’ command. For the sake of brevity only the first and last wheel speeds are shown.

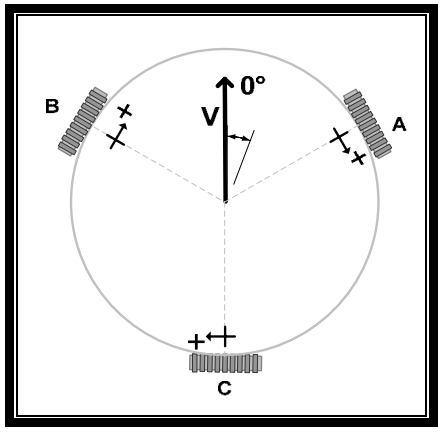


Figure 29: Direction of rotation

```
m000Enter speed to move at (2-20):
0012Enter direction to move in (0-360):
0075The rotation is 0.

Test complete
Speed Values Motor C
000000
000006
...
000012
000010
000012
Speed Values Motor B
000000
000001
000004
...
000008
000008
000008
Speed Values Motor A
000000
000001
000000
...
000002
000004
```

Figure 30: Example of the move robot command

3.5.3d Main movement command

The ‘g’ command is the command used by the joystick control program to drive the robot. It is also the command that the host computer will use to drive the robot. The command has speed, direction and speed of rotation parameters. The parameters are entered as single bytes. The packet structure is shown in Figure 31. The first byte is the character ‘g’ (which has ASCII value 103). The second byte is the speed, which is a value from 0 to 14. The following byte is the direction in which the robot should move. The direction is given as half the direction angle – since only 1 byte is available for the number, it has to be a number between 0 and 255. The last byte gives the robot the rotation it should move through. Since the robot can rotate in a negative

(anticlockwise) or positive (clockwise) direction, and the bytes are sent as unsigned bytes, the rotation is sent as 20+ the desired speed of rotation (20 was chosen arbitrarily). Thus, to rotate anticlockwise with a speed of 4 pulses per 30ms, the direction should be entered as -4 + 20 which is 16. Since the speed can vary between 0 and 14, the rotation+20 parameter can vary between 6 and 34. Figure 32 shows two example commands. The value 20 was chosen arbitrarily.

g	Speed	Direction/2	Rotation + 20

Figure 31: Packet structure of movement command

Command	Speed	Direction	Rotation
‘103’’3’’60’’0’	3	60 * 2 = 120	0
‘103’’0’’0’’22’	0	0	22 - 20 = 2

Figure 32: Example 'g' commands

3.5.3e Commands for Advanced Motion

The ‘g’ and ‘m’ commands control the robot either in a lateral direction or while rotating, but do not allow one to accurately control the robot so that it moves laterally in a specified direction while rotating. The ‘h’ ‘k’ and ‘l’ commands allow for this advanced motion control. They work by breaking up the motion into discrete time intervals and updating the orientation of the robot after each time interval so that it will continue to travel in the intended direction despite having rotated to face another direction. This motion is further decribed in Appendix F. The ‘h’ and ‘k’ commands are used mainly for debugging and demonstration purposes while the ‘l’ command will be the command used by the host computer to control the robot.

The ‘h’ and ‘k’ commands both control the robot for a specified amount of time while logging the wheel speeds. The time that the robot is controlled is determined by parameters set at compile time. The ‘h’ command has no inputs and the speed, direction and rotation of the robot are entered at compile time. The ‘k’ command allows the user to enter these parameters when the command is run. Sending the command ‘h000’ or ‘k000’ puts the robot into advanced motion mode. If the command is ‘k’, the robot then requests what speed it should move at. The robot then requests a direction and a speed of rotation. All entries should be four digit numbers. The speed is in encoder teeth per 30ms and should be between 0 and 14. The direction is an angle in degrees. The speed of rotation should be a number between 0 and 14. Once the robot has finished moving it returns a record of the three wheel speeds during the movement. Figure 33 shows an example of the ‘k’ command. For the sake of brevity only the first wheel speeds are shown.

```
k000Advanced Motion
Enter linear speed to move at (2-14):
0012
Enter direction to move in:
0120
Enter speed of rotation (2-14):
0004
finished
Speed Values Motor C
000000
000004
000006
000010
...
```

Figure 33: Example of the 'k' advanced motion command

The 'l' command is used by the host computer to control the robot so that it can move laterally while rotating has the same format as the 'g' command already described. This format is shown in Figure 34.

l	Speed	Direction/2	Rotation + 20

Figure 34: packet structure of 'l' command

Once the command is entered, the robot will run continuously in the direction of motion at the set speed until another 'l' command is entered or the stop command 'l''0''0''20' is entered.

3.5.3f Step Tests

The 'T' commands run step tests and were used during the tuning of the control algorithm. 'T100' runs an open loop step test at 100% of the motor voltage. 'T200' and 'T300' run the open loop tests with the voltage to the motors set to 75% and 50% of the full voltage. 'T500' runs a closed loop step test. The user is required to input the values for K_P, K_I and K_D of the PID control algorithm and the test is then executed with those values. After the tests are complete, the robot sends via the serial port the 80 consecutive speed readings for each wheel taken during the test. Figure 35 shows an example of the closed loop test. For more information on the step tests, see Appendix D.

```
T500
Control Test2: Basic Control,
Please enter value for setpoint:
0008
Control Test2: Basic Control.
Please enter value for 100Kp:
1800
Please enter value for 100Ki:
2000
Please enter value for 100Kd:
0038
Running Control Test. KP = 000018.000
Running Control Test. KI = 000020.000
Running Control Test. KD = 000038.000
Test complete
Speed Values Motor C
000000
000001
000004
000007
...
000008
000008
Speed Values Motor B
000000
000000
000000
...
000000
000000
000000
Speed Values Motor A
000000
000000
000001
...
000008
000008
000008
```

Figure 35: Example of the 'T500' closed loop step test

3.5.3g Calibration Tests

To calibrate the robot's measurement of distance to real world measurements or the measurements of the overhead vision system, two calibration commands were implemented. The 'i' command is for calibrating distance and runs the robot at a user specified speed for a user specified time. The 'j' command is for calibrating degrees of rotation and rotates the robot at a user specified speed for a user specified time.

3.6 Concluding Remarks

The final design consists of a physical robot platform with electronic components controlled by DSP software. The physical system is the omnidirectional platform. The electronic components provides a means to power the robot, drive the motors and receive feedback as to the speed of the wheels. The DSP software controls the physical system by interacting with the electronic system. The following two chapters discusses how the robot was controlled so that it could move in an omnidirectional manner.

4 Robotic Platform Control

4.1 Introduction

This section describes how the robot platform is controlled by running the wheels at appropriate speeds. The section also discusses how this method of control differs from the method of control previously used on this robot. Note that in the context of this dissertation robot platform control is concerned with determining the speed at which the wheels should move to achieve a certain translation and rotation, wheel speed control is concerned with controlling the speed of the wheels so that their rotational velocity remains constant. This chapter is concerned with robot platform control while Chapter 5 is concerned with wheel speed control.

4.2 The move from external to onboard platform control

When first designed by Mr. Bamber, the platform control of the robot happened externally, on a Matlab program on the host computer. This project has moved the platform control onto the robot itself, freeing up computing power on the host computer. Mr. Bamber's Matlab program calculated the speed at which each wheel needed to run in order to move in a particular direction and rotation. Commands would then be sent to the robot to give it those wheel speeds. Also to be noted is that the wheel speeds were not controlled. Rather the wheels were run in open loop with no speed feedback.

The new robot, while using the same wheel base and wheels, is now controlled more effectively. The wheel speeds are controlled in closed loop by a PID controller using

wheel speeds measured by shaft encoders. The robot also performs all platform control calculations on the DSP. The robot receives velocity, direction and rotation commands, translates these into wheel speed velocities and then controls the wheels to run at the appropriate speed. Advanced robot platform control code then used the lower level of platform control to be able to move the robot in a lateral direction while it is rotating.

4.3 Kinematics

Appendix A.6 describes documents reviewed in an attempt to find an inverse kinematic equation with which to control the robot platform. A suitable equation could not be found so the inverse kinematic equation was instead derived by the author as described in Appendix E. Below is a statement of the inverse kinematic equation. This set of equations relate the robot speed, direction and rotation to the speeds at which the robot is travelling.

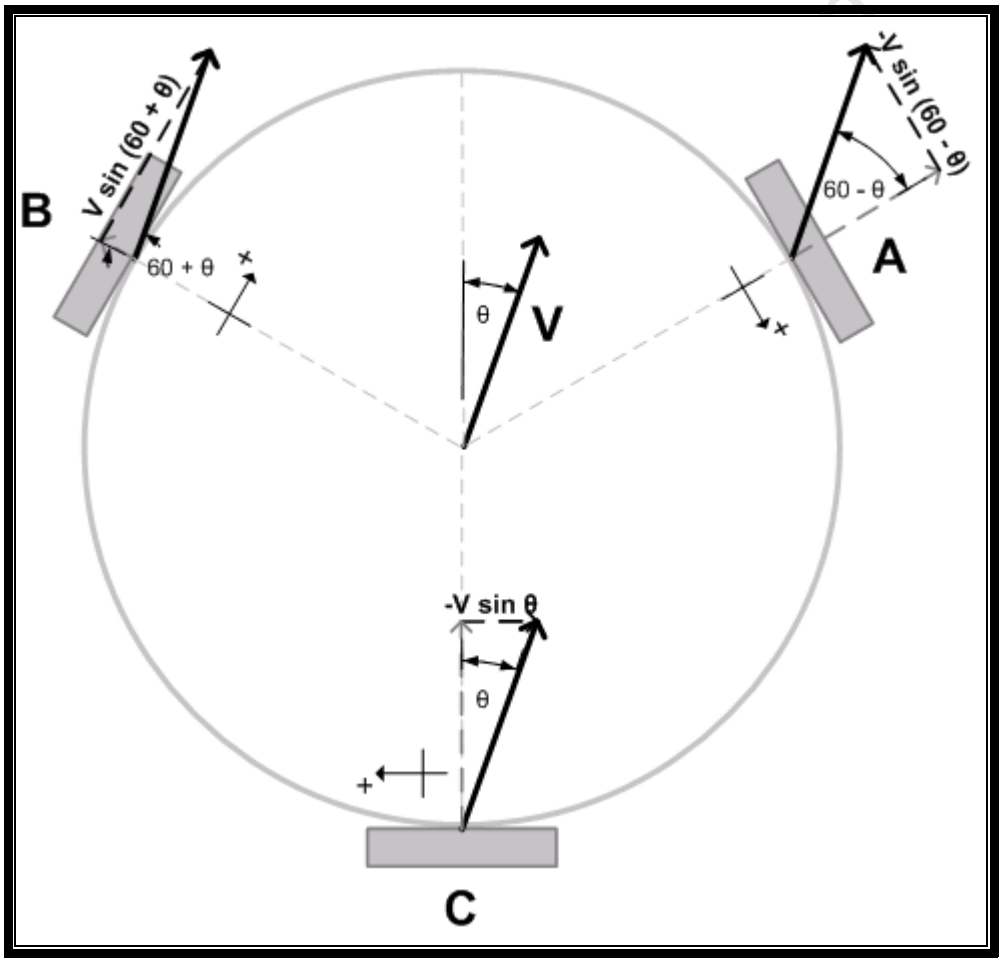


Figure 36: Diagram used to derive inverse kinematic equations

Referring to Figure 36, if one needs the robot to move at a speed V, in a direction θ, the wheels A, B and C should be run at the following speeds:

$$V_A = -V\sin(60-\theta) \quad [1]$$
$$V_B = V\sin(60+\theta) \quad [2]$$
$$V_C = -V\sin(\theta) \quad [3]$$

To rotate the robot, all three wheels need to be run at the same speed, $V_{rot,}$ in the same direction.

Therefore, for small distances, to rotate the robot at a speed V_{rot} and move it laterally at a speed V_{lat} in a direction θ, the wheels should be run at the wheel speeds below:

$$V_A = -V_{lat}\sin(60-\theta)+V_{rot} \quad [4]$$
$$V_A = V_{lat}\sin(60+\theta)+V_{rot} \quad [5]$$
$$V_A = -V_{lat}\sin(\theta)+V_{rot} \quad [6]$$

This set of equations was used as the basis for the 'g' main movement command. The code was successful in controlling the robot.

4.4 Advanced Control

Equations 4-6 are only valid for short distances because as the robot rotates, it's orientation changes. As the lateral movements are calculated relative to the robot's frame of reference, in the universal frame of reference, the robot will no longer be headed in the intended direction. The exciting thing about omnidirectional control is that the robot can be controlled to move laterally while rotating to face a new orientation. To solve this problem and allow the robot to be controlled while rotating so that the direction of lateral motion remains the same, a discrete method of controlling the robot is employed. The motion of the robot is broken up into discrete distances. The robot is given a lateral direction and a speed of rotation over a short distance. The orientation of the robot is then updated and the robot given a corrected direction in which to head. When the iterations are small enough this yields an effective method of smoothly controlling the robot in a lateral motion while it rotates to a new orientation.

Three videos on the accompanying CD illustrate the difference between the basic and advanced control. Video 4c shows the robot moving forward and then rotating. Video 4b shows the robot using basic control in which it moves laterally while rotating without updating its orientation, the robot follows a curved path. Video 4c on the CD shows the robot moving omnidirectionally – the robot remains on a straight path while it rotates to a new orientation.

4.4.1 Simulation

To better understand this method of control before implementing it on the robot, a Matlab simulation was written to show the robot moving omnidirectionally and to plot the changes in wheel speeds that would make this movement possible.

Figure 37 shows the simulation plotting the robot position as it moves along a 45° line, at a rotation of 1 pulse per 30ms and a lateral speed of 12 pulses per 30ms. The robot starts at the position (100,100) in the bottom left hand corner of the figure. The robot moves allong the 45º line for the time interval *ti*. During this time interval the robot would be rotating at a speed of 1 pulse per 30 ms while moving laterally at 12 pulses/30ms. The second plot in the figure shows the robot after the time interval. It has moved along the 45º line and has also rotated clockwise. Each subsequent plot shows the robot having moved further along the lateral line and having rotated further. After 5 time intervals the robot has moved 1,3m along a 45º line and rotated by 120º degrees.

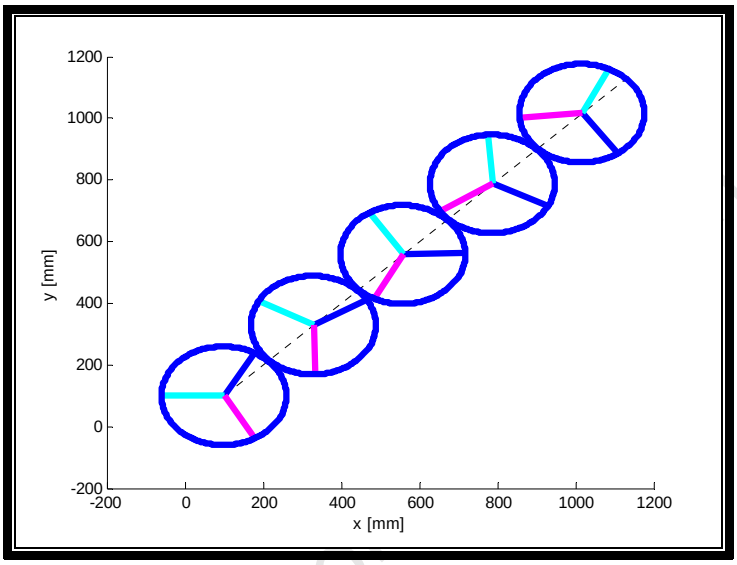


Figure 37: Simulation of robot rotating clockwise as it moves along a 45° line

Figure 38 shows a plot created by the simulation of what the wheel speeds would need to be for the robot to travel on the path shown in Figure 37. The speeds are determined based on the direction which the robot is facing at the time and the direction which it needs to move in (in this case 45º). The speeds are calculated using equations 4-6 from Section 4.3. The speeds follow a sinusoidal function with respect to time. This was expected since the direction of motion relative to the robot changes with time (as the robot rotates) and the speed of each wheel is a sinusoidal function of the robot's direction of motion.

To further investigate the wheel speeds, the simulation was repeated with varying lateral speed, speed of rotation and direction of rotation. Figure 39, Figure 40 and Figure 41 show the effect on wheel speeds of changing the direction of lateral motion, changing the speed of lateral motion, and changing the speed of rotation respectively.

In Figure 39 the effect of varying the direction of lateral motion by 5º is shown. The graph shifts with respect to time. This phase shift shows that the phase of the wheel speed functions is dependent on the intended direction of motion.

Figure 40 shows the effect of changing the speed of lateral motion. The amplitude of the wheel speed functions increases with increasing lateral speed. Therefore the amplitude of the wheel speed function is directly dependent on the lateral speed at which the robot needs to travel.

Figure 41 is a plot of the wheel speeds taken at three different rotational velocities. The frequency of the wheels speed functions increases as the rotational velocities of the wheels increase. This reflects the fact that, the faster the robot rotates, the more the direction of motion relative to the robot will need to be updated each time interval.

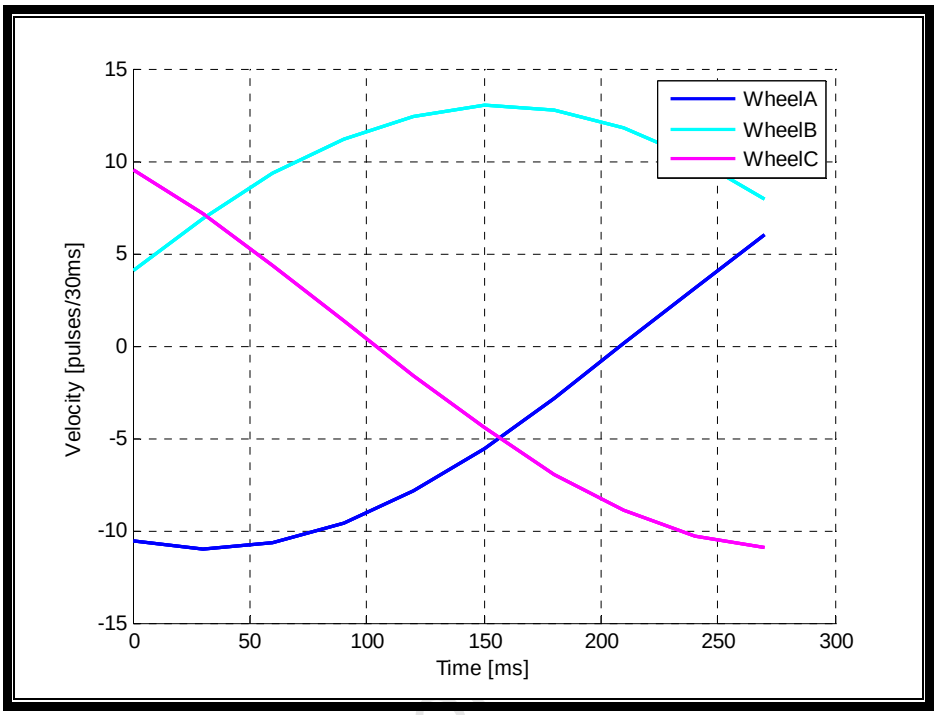


Figure 38: Changing wheel speeds produced by simulation

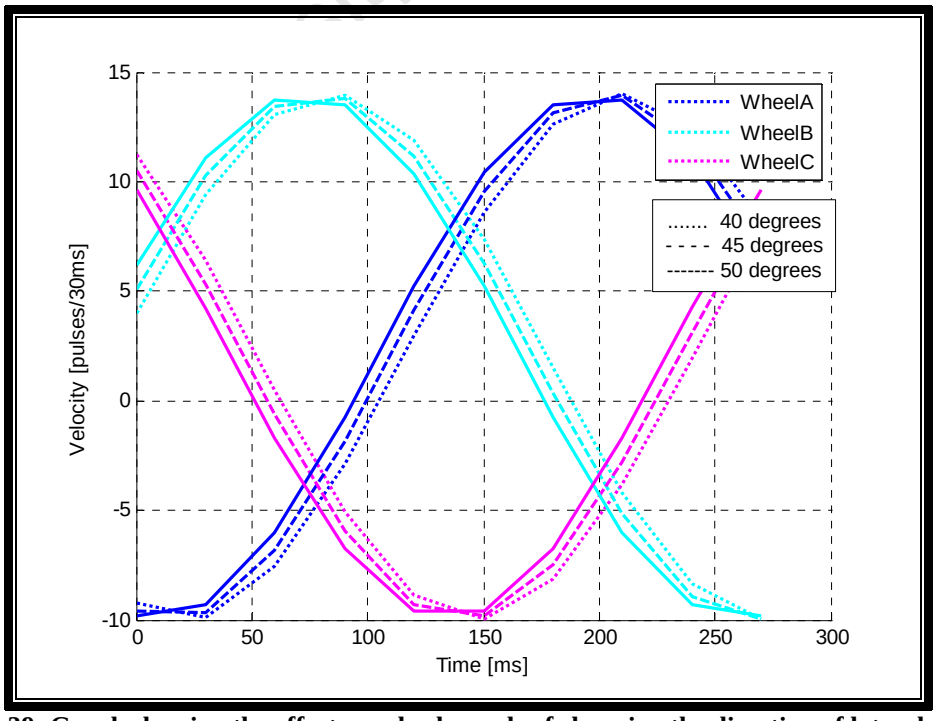


Figure 39: Graph showing the effect on wheel speeds of changing the direction of lateral motion

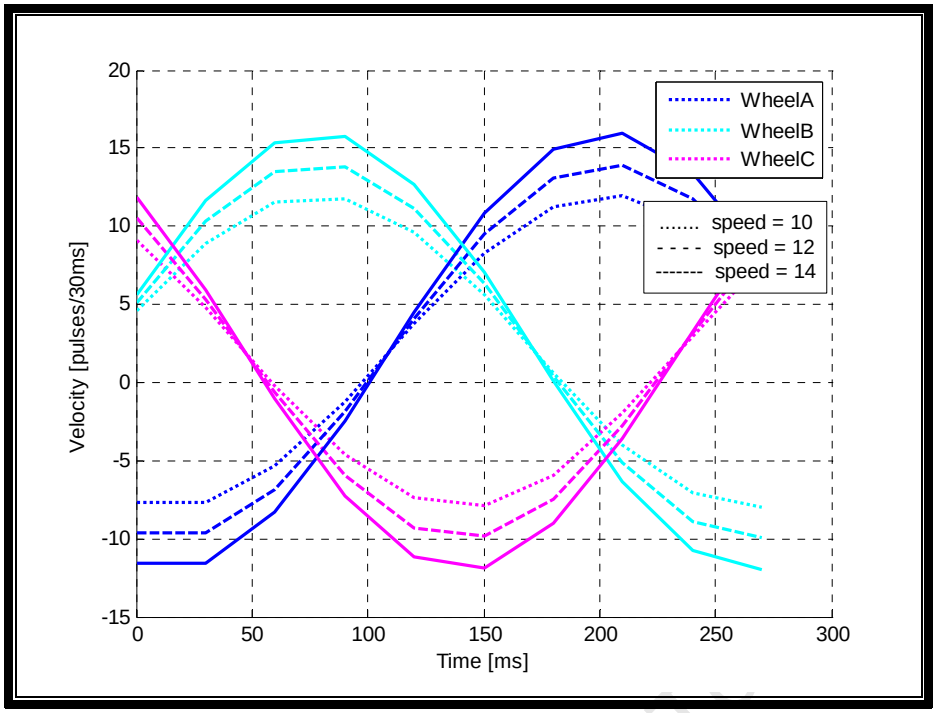


Figure 40: Graph showing the effect on wheel speeds of changing the speed of lateral motion

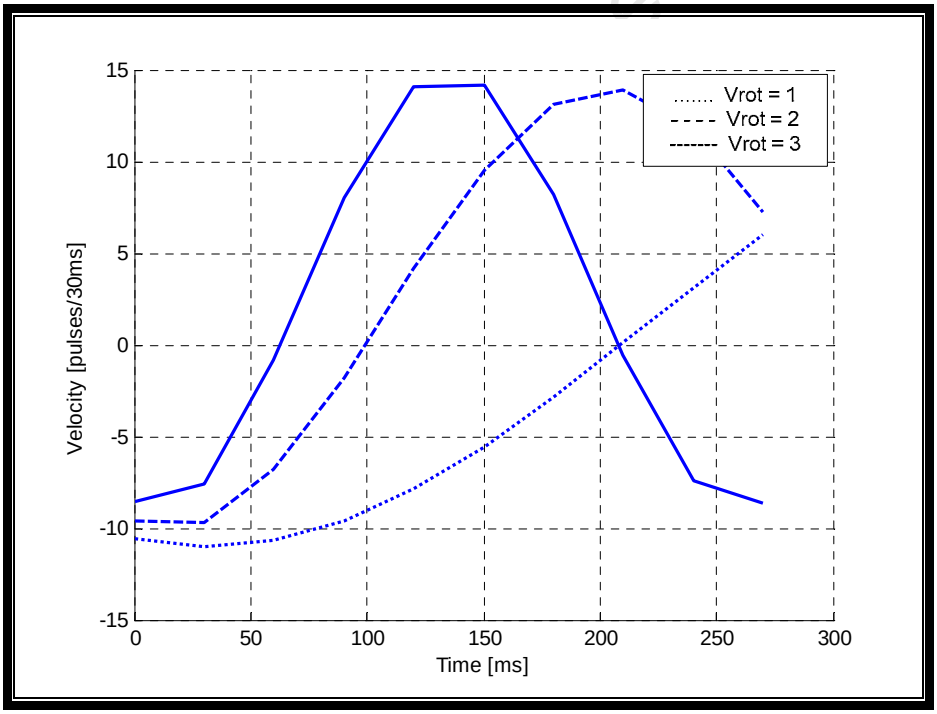


Figure 41: Graph showing the effect on wheel speeds of changing the speed of rotational motion

Investigating advanced motion through simulation determined the effects of lateral direction, lateral speed and speed of rotation on the wheels speed functions phase shift, amplitude and frequency respectively. Advanced motion was then implemented on the robot.

4.4.2 Implementation on the robot

To continue travelling in a specified direction while rotating, the robot's orientation needs to be continuously updated. The 'h' , 'k' and 'l' commands (see Section 3.5.3e) achieve this by calling the method *advance*. This method moves the robot at a specified speed in a specified direction at a specified speed of rotation for a small time interval, *ti*. The method then updates the direction of the robot to compensate for the fact that the robot has rotated to face a new orientation. The amount of rotation is equal to the product of the speed of rotation, the time of the interval, and a correction factor, *Q*. The optimal length of time for *ti* and value of *Q* were determined experimentally as documented in Appendix F.

4.5 Results

Basic platform control using equations 4-6 was successfully implemented on the DSP. When the 'g' command is sent by the host computer, the DSP runs the *move()* method which sets the wheel speeds according to equations 4-6. These equations have been found to work well over small distances. Of course, as the robot rotates, it's orientation changes. Thus the direction it is facing relative to the direction it needs to drive does not remain constant. Therefore equations 4-6 are only true over small distances. Equations 1-3 which are effectively the other equation set with V_{rot} set to 0, remain true for all distances. Therefore the robot, when asked to move only laterally does so with only small error (due to slip) if no rotation is needed. Similarly, if asked to rotate with no lateral translation, the robot responds with little error.

Extended platform control was implemented on the robot with some success. The control is only fully effective at high lateral speeds and low rotational speeds. These are the wheel speeds that are anticipated during soccer tournaments. The robot moves laterally while rotating at given speeds, the motion is smooth and the robot successfully continues its lateral movement while changing its orientation.

To successfully achieve the platform control just discussed, it was necessary to accurately control the speed of the robot's wheels. How this wheel speed control was implemented is discussed in Chapter 5. The testing of both types of platform control, as well as the wheel speed control, is fully documented in Chapter 6.

5 Wheel Speed Control

5.1 Introduction

The robot platform control has been discussed in Chapter 4. The DSP translates robot movement commands into wheel speeds. This algorithm can however only be effective if the wheels are controlled to maintain an intended speed. To control the speed, an effective method of wheel speed measurement needs to first be implemented. This chapter gives a brief overview of the wheel speed measurement (odometry) and the implementing and tuning of a PID controller to control the wheel speed. A more detailed account of these can be found in Appendix C.

5.2 Odometry

Previous work done on this robotics platform by Mr. Bamber was unsuccessful in implementing wheel speed measurement. For this project, Mr. McPhillips designed and manufactured shaft encoder masks to fit onto the hubs of the omnidirectional wheels. With these in place, H21A1 optical switches were fit onto the robot base at each wheel so that as the wheel rotated, the teeth of the encoder mask would interrupt the optical switch. Figure 42 shows the encoder mask and optical switch mounted on the robot.

The circuitry required to drive the interrupter switch was on a single veraboard for the development board. It was also included on the final PCB. The PCB has four encoder circuits although only three are needed, to allow an extra motor to be controlled in the future if required.

Once the encoder had been implemented, the method of speed measurement had to be finalised. The commonly used period and frequency wheel measurement methods

were considered and well as a novel, slope measurement method. The comparison between these methods appears in Appendix C. It was decided to implement the frequency or tooth counting method where the number of teeth to pass the interrupter switch in a given time period are measured. The chosen time period was 30ms. This time period is long enough to give 14 different wheel speeds but short enough that the robot can still be controlled at slow speeds.

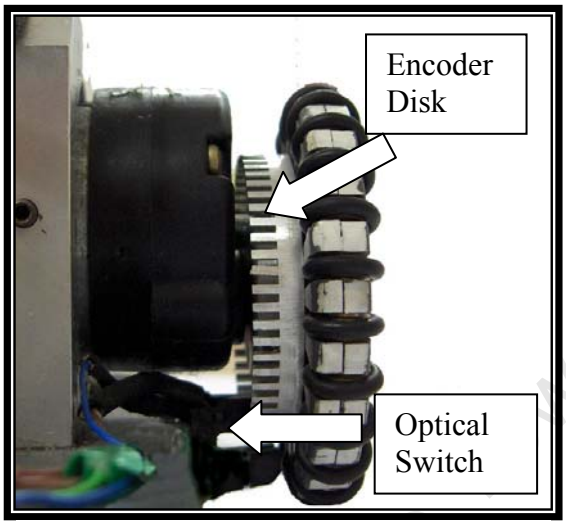


Figure 42: Shaft encoder system

5.3 Control Algorithm

The wheels were controlled using a PID control algorithm as shown in Figure 43. The format of the PID controller algorithm is :

$$V = P + I + D$$

where:

$$P = K_p e$$
$$I = I_{old} + K_i e$$
$$D = K_d (e - e_{old})$$

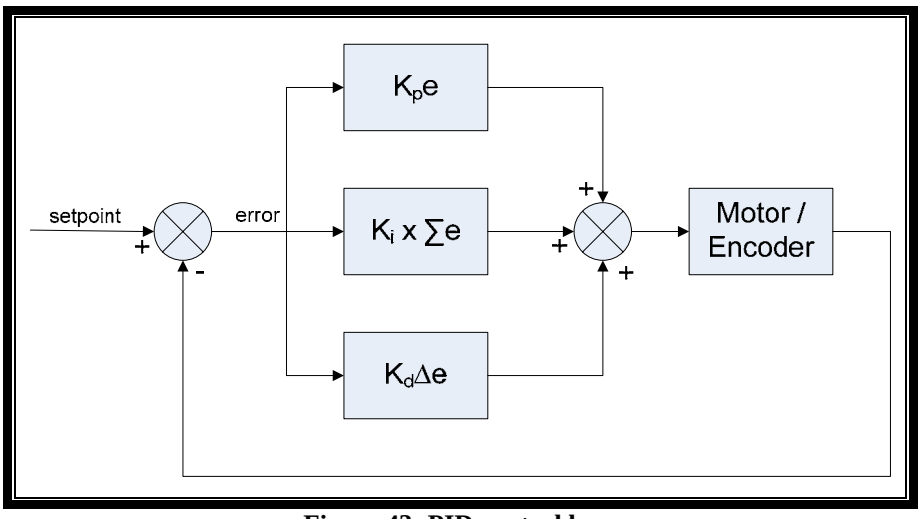


Figure 43: PID control loop

5.4 Loop Tuning

To select the PID parameters, a combination of the Ziegler-Nichols and experimental tuning were used. The Ziegler-Nichols method proved effective in finding the Kp value only. It is unclear whether the method proved ineffective because of the low resolution of the speed measurement or because of an error in how it was implemented or if the system does not match the method's intended use. Trial and error proved an effective method for tuning the Ki parameter of the control algorithm. An effective test could not be found to test the effectiveness of the Kd value suggested by the Ziegler-Nichols method.

5.5 Final Parameters Used

The following parameters were used in the final control algorithm:

$$K_p = 180$$
$$K_i = 20$$
$$K_d = 0.38$$

5.6 Concluding Remarks

The successfully implementation of PID control meant that the speed of the wheels could be kept constant. This enabled the robotic platform control discussed in Chapter 4 to be implemented which in turn enabled the advanced platform control (Section 4.4) to be implemented. The controller was designed to overcome the effects of the robot's inertia and the drag of the umbilical cord as effectively as possible. The effectiveness of the algorithm was however limited by the resolution on the encoders used for measuring the wheel speed. Chapter 6 presents the results of tests done to determine the effectiveness of all three levels of control.

6 Testing

6.1 Introduction

There are three levels of control to the control of the omnidirectional robotics platform. The wheel speed control is concerned with keeping the speed of the wheels constant. Basic platform control is concerned with controlling the robot to move in either a lateral or rotational manner. The advanced platform control is concerned with controlling the platform so that it moves laterally at the same time as rotating. This chapter describes the tests undertaken to determine the effectiveness of each of these methods of control. Visual tests were documented using photographs and videos. The resultant videos can be found on the accompanying CD.

6.2 Testing of Wheel Speed Control

6.2.1 Objectives

The wheel speed control algorithm needs to increase the wheel speed to, and hold it at, a desired speed. The behaviour of all wheels when controlled needs to be the same.

6.2.2 Method

The wheel speed control was tested both quantitively and qualitively. Quantitively, the wheel speeds were measured during closed loop step tests. The closed loop step tests were run at different speeds to test the effectiveness of the control algorithm. Qualititively, the robot was run in a straight line and the accuracy of that line was considered.

6.2.3 Results of closed loop step tests

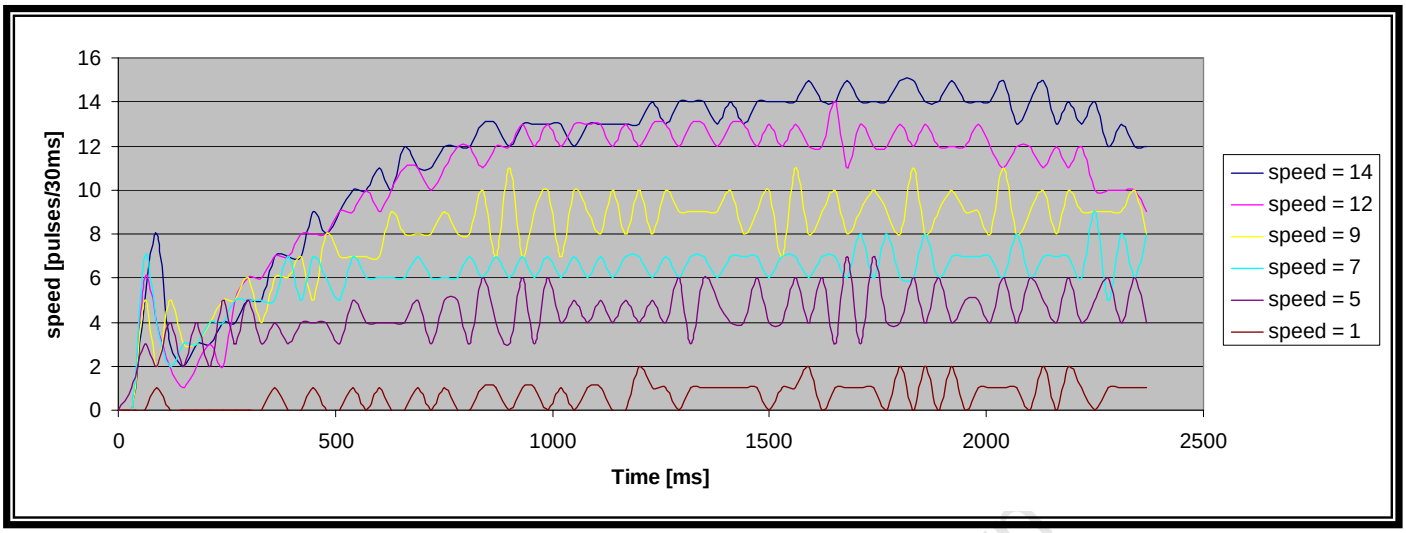


Figure 44: Step tests run at various speeds in closed loop

Figure 44 shows the results of closed loop step tests taken at 6 different speeds. The wheel speed settles after between 250 and 750 ms. The response spikes at 90ms before settling. Wheel speeds below 12 pulses/30ms settle well, at higher wheel speeds, the wheel speed is not as stable. This is because the PWM duty ratio applied is reaching a maximum, and the DSP cannot increase the voltage sufficiently to correct disturbances.

When the robot was operated on the soccer field it ran straight. Over distances of more than 1m, the line of motion would begin to curve, this is due to errors due to slip, which dead reckoning cannot correct for. The curve in the line of motion can also be accounted for by the robot having to drag its umbilical cord. Although step tests were done with the umbilical cord attached which means the control algorithm should account for the drag of the cord, if the cord is not directly behind the robot, it could cause the robot to veer to one side slightly. The low resolution of the encoders would also cause the control algorithm to behave less than optimally.

6.2.4 Concluding Remarks on Wheel Speed Control

The control algorithm was effective up to speeds of 12 pulses/30ms. The robot cannot be relied on to behave as reliably at speeds of 12pulses/30ms and above. Increasing the resolution of the encoders would increase the accuracy of the control algorithm. The robot was able to go straight but wheel slip and drag from the umbilical cord introduced some error, especially over long distances. The overhead vision system and host computer should be able to correct for these errors.

6.3 Testing of Basic Platform Control

6.3.1 Command Line Control

As discussed in Chapter 4, the lower level of platform control, here referred to as basic platform control consists of separate lateral and rotational movements of the

robot. At this level the robot can be given a speed and direction of travel or a speed of rotation. Commands can be sent to the robot ('g' and 'm' commands as described in Sections 3.5.3c and 3.5.3d respectively) to instruct it at what speed and in what direction to move laterally, or how fast to rotate. The robot can move laterally and rotate at the same time but as the robot rotates, the orientation is not updated, resulting in a curved line of motion.

6.3.2 Objectives

When testing the basic platform control, it is the robot's 'omnidirectionality' that needs to be considered. The robot needs to be able to move in any direction without first changing it's orientation to face the direction of movement. To achieve this the robot was tested driving in a number of directions from a stationery position. The maneuverability of the robot was also tested by driving the robot on set paths - a square path and a 'zigzagging' path.

6.3.3 Method

The 'm' command was used to direct the robot in a number of different directions from a stationery position at different speeds. To test maneuverability a Matlab program was written to drive the robot in a square pattern using the 'g' command. The program drove the robot straight, then at 90º, then at 180º then at 270º for the same amount of time at the same speed. The program to 'zig zag' the robot drove the robot alternatively in a 60º direction and then a 300º direction.

6.3.4 Results

The robot was successfully able to drive straight in a number of different directions from 0º to 360º. Figure 45, 46 and 47 show the robot driving in a 60º, 150º and 270º direction respectively, moving from a standstill without rotating to face the direction of motion. Videos 6a – 6g on the accompanying CD show the videos from which these snapshots were taken as well as the robot driving in other directions at various speeds. The robot responds as expected, driving in the desired directions without changing orientation. Occasionally errors due to slip cause the robot to go off course slightly but these are to be expected on a system based on dead reckoning. In the robot soccer competition the overhead vision system and controlling host computer are designed to be able to compensate for these errors. Further errors in the movements of the robot can be attributed to the robot having to drag the umbilical cord. Once the need for an umbilical cord is eliminated it would be expected that the accuracy of the robot's movements would improve.

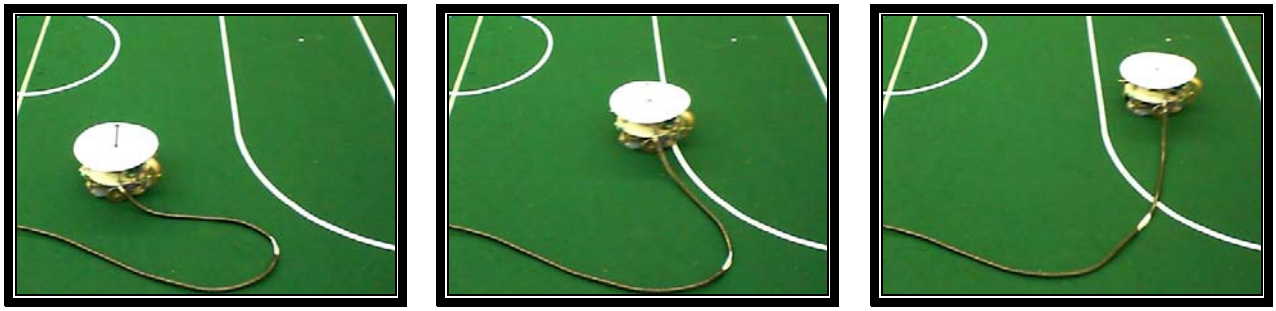


Figure 45: Robot moving in a 60° direction

Figure 46: Robot moving in 150° direction

Figure 47: Robot Moving in 270° direction

Figure 48 shows the robot as it moves in a square. The robot moved as expected and its final position and orientation were very close to its original orientation and position indicating that the movements of the robot were accurate. The video footage of this test is the file '6l – square.avi' on the accompanying CD.

Running the robot in a 'zig-zag' pattern effectively shows its maneuverability. The video footage '6k – Zig Zag' on the accompanying CD shows footage of the robot driving in a 'zig-zag' pattern.

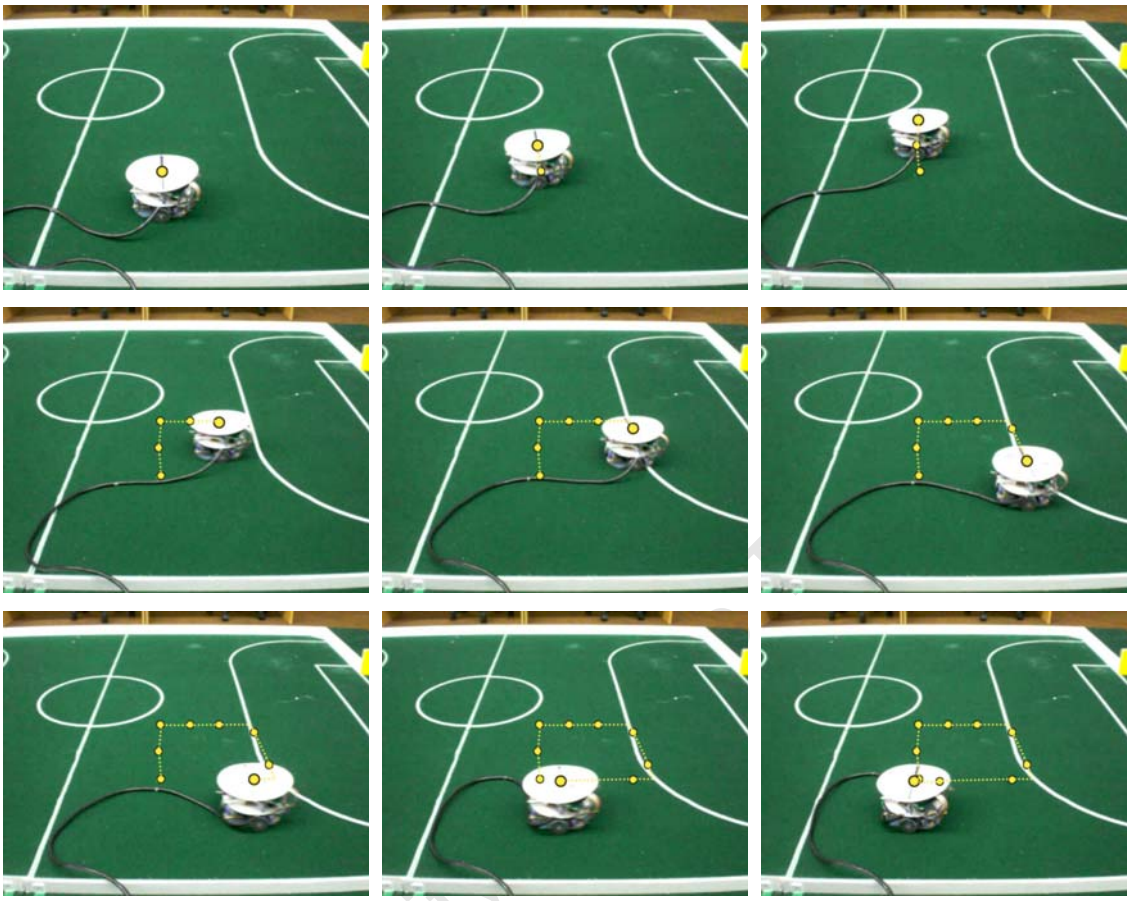


Figure 48: Sequence of snapshots taken as robot moves in a square

6.3.5 Concluding Remarks on Basic Control

The basic control of the robot was implemented effectively. The robot is omnidirectional; it is able to move in any specified direction without first reorientating itself to face the direction of motion. The robot is highly maneuverable. Errors due to slip are apparent periodically but in the robot soccer competition these will be compensated for by the host computer and the overhead visions system.

6.4 Joystick Control

6.4.1 Objectives

A joystick was used to control the robot manually. Although it would not be used in the final robot soccer system, the joystick program provides a platform for testing the controllability and maneuverability of the robot platform. The robot needs to respond as expected to the joystick commands, driving in the direction in which the joystick is moved, with an appropriate speed, and rotating when the joystick is twisted.

6.4.2 Method

To test the effectiveness of the joystick control the robot was driven around the robot soccer field in various directions. The robot was controlled at different speeds. The robot was rotated at different speeds and in different directions. Video footage was taken of these tests.

6.4.3 Results

The robot was effectively controlled by the joystick. The robot drives in the direction in which the joystick is pushed and rotates when the joystick is twisted. The robot moves very fast, making it difficult to control accurately with the joystick – the joystick must be operated very lightly to get the robot to drive slowly. The video footage of the robot being controlled by the joystick are the files 3a and 3b on the accompanying CD.

6.4.4 Concluding Remarks on Joystick Control

The joystick control is effective at showing the maneuverability of the robot and provides a good basis for testing the robot's movements. The controls are however very sensitive, making the robot difficult to control. Decreasing the sensitivity of the joystick control program so that the robot drives at slower speeds would make the robot easier to control with the joystick.

6.5 Testing of Advanced Platform Control

6.5.1 Objectives

The advanced platform control is concerned with rotating the robot while in moves laterally in a specified direction. The testing of the algorithm to control the robot in this manner is concerned with whether the robot does continue in a straight line along the lateral path while it rotates. The robot should behave as expected over a range of lateral speeds, and rotational speeds and in all directions. The size of these ranges needed to be determined.

6.5.2 Method

All testing of the advanced platform control was done visually. The calibration of the advanced motion parameters was performed at a lateral speed of 12 and a rotation of 2. With these speeds as a starting point the first test was whether the advanced motion control would work in any direction. The next tests varied the speeds of rotation and lateral motion while the motion of the robot was observed.

6.5.3 Results

The lateral speed can vary between 12 and 14 pulses/30ms. At speeds below this, the lateral motion is no longer straight. The rotational speed can vary between 1 and 6 pulses/30ms (the speed was not tested above 6 pulses/ms since at this speed the robot is rotating significantly faster than it is moving laterally and it is unlikely that such rotational speeds would be necessary). The robot behaves accurately in any direction. Videos 6h-6j show the robot moving in three different directions at a speed of 12 pulses/30ms while rotating clockwise. Figure 49 shows stills from video 6i. The robot is traveling in a 330º direction while rotating. Arrows are used to show the change in the robot's orientation over time.

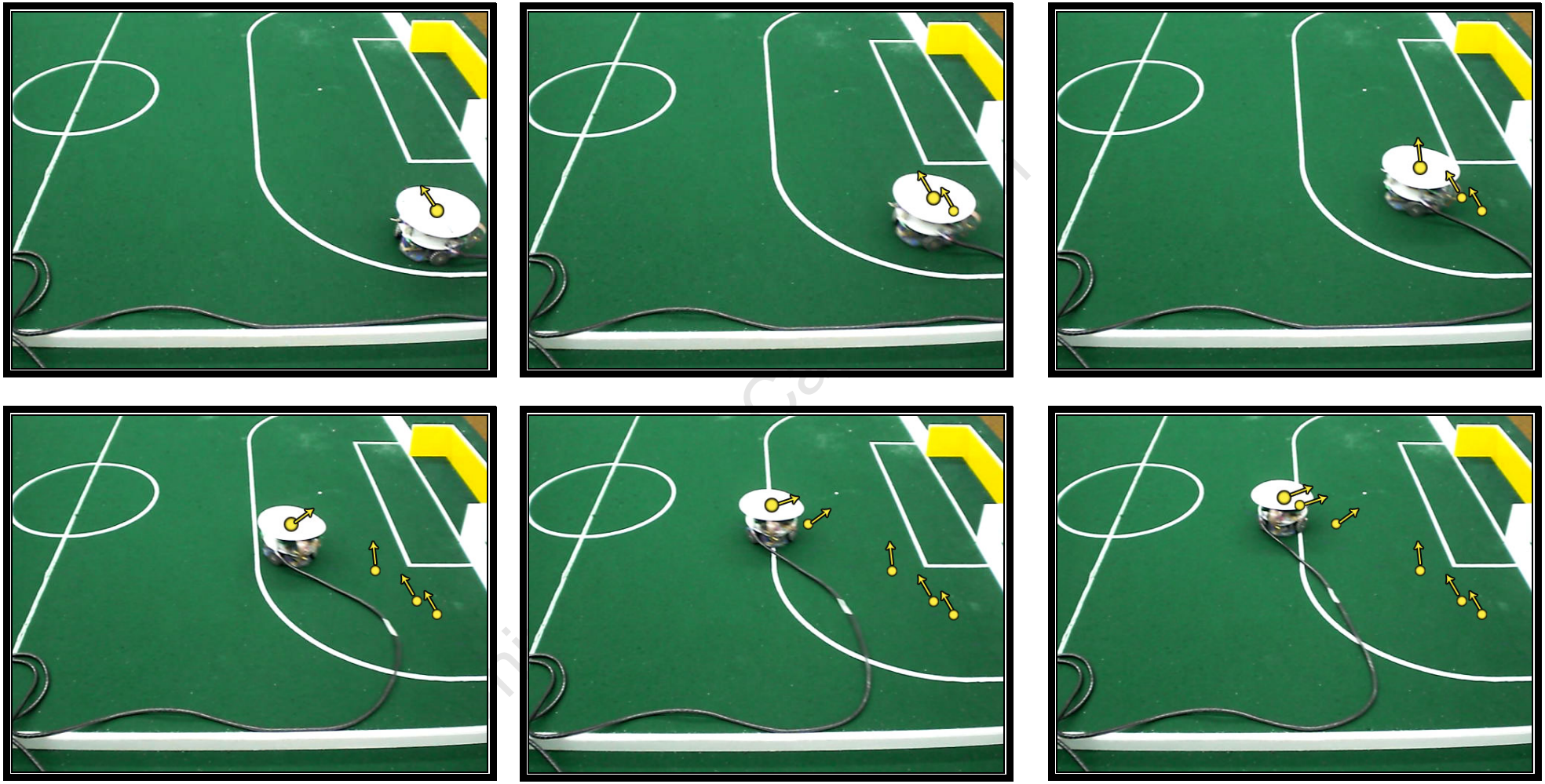


Figure 49: Robot travelling at a speed of 12 pulses / 30ms, in the direction of 330º, while rotating clockwise at a speed of 2 pulses / 30ms.

6.5.4 Concluding Remarks on Advanced Platform Control

The lateral motion is only straight when the lateral speed is between 12 and 14. This suggests that the assumption that the correction factor Q is not affected by the lateral speed of the robot is incorrect. Further tests would have to be done to determine the effect of the speed of lateral motion on Q and the control code would have to be altered to compensate. The advanced motion is effective over a large range of rotational speeds (1-6 pulses/30ms). The advanced motion behaves accurately in any direction of lateral motion.

7 Conclusions

7.1 Advanced Motion

The robot can be controlled to move laterally while rotating but only at higher speeds. The correction factor, *Q,* which was assumed to be independent of lateral speed, is related to the lateral velocity. This relationship was not found, though a suitable Q of 0.7 allows the robot to be controlled effectively at speeds 12-14pulses/30ms. Smoother control could also have been achieved had the value of the timing interval *ti* been able to be reduced further. However, the length of time of *ti* is limited by the length of time of the control algorithm time step which is in turn limited by the resolution of the wheel speed encoders.

7.2 Robot Platform Control

The kinematic equations derived to control the omnidirectional platform proved effective. Tests confirmed that the robot is able to move in any direction without first reorientating itself to face the direction of motion.

7.3 Wheel Speed Control

7.3.1 Control Loop

The wheel speeds were effectively controlled by the PID control algorithm. However, small errors in the robot's trajectory were noted periodically. These are due to wheel slip which a dead reckoning measurement system cannot correct for. The overhead vision system and host computer will correct for these errors in the final robot soccer system. Wheel slip can also be caused by the control algorithm wanting to accelerate

the wheels too fast. Increasing the resolution of the encoders would increase the accuracy of the control algorithm.

7.3.2 Loop Tuning

The Ziegler-Nichols method of loop tuning proved ineffective in finding suitable parameters for the PID controller. Manually tuning the controller proved effective for finding the K_p and K_i parameters but not the K_d parameter.

7.3.3 Wheel speed measurement

The wheel speed measurement system was effective. The frequency method of odometry proved effective on this robot. Errors in the trajectory were apparent periodically due to slip. Within the robot soccer environment the overhead vision system and host computer will be able to correct for these errors.

7.4 Physical System

Mr Bamber's base proved to be a very satisfactory omnidirectional platform. The robot base can be controlled successfully.

7.5 Onboard electronics

7.5.1 Digital Signal Processor

The Freescale DSP56F8323 proved an effective controller.

7.5.2 Power

Since a battery holder was not developed for the new lithium ion batteries, their effectiveness could not be tested. The robot was tested using power supplied by a tether. The switch mode power supply implemented on the PCB proved effective and should increase the life of the batteries between charging.

7.5.3 Communications

The communications over a serial cable were effective. Attempts to replace the serial cable with the wireless transmitters used previously for robot soccer, were unsuccessful.

7.5.4 Motor Drivers

The LMD18200T h-bridge chips were effective in driving the motors.

7.5.5 Encoders

The H21A1 based circuit used for detecting the shaft encoders were effective.

7.6 DSP Control

7.6.1 Code

C++ code had been written to control the robot. The code is modular, robust and effective. The code allows for communications between the host computer and the robot both during testing and debugging phases and for when the robot is controlled

in the final system. The communications protocol allows for only one robot to be controlled - it does not include an identifier to indicate which robot a command is intended for. The code has been documented and is written in a modular fashion so that it can be expanded as necessary.

7.6.2 Command Based Control

A command system has been designed and implemented that provides both a testing and debugging interface and a final system for controlling the robot from the host computer. The commands effectively control the motions of the robot in both basic and advanced platform control.

7.6.3 Joystick Control

The joystick control system was effective in demonstrating the manoeuvrability of the robot. The joystick system was however very sensitive, making it difficult to drive the robot slowly and control it accurately.

7.7 Printed Circuit Board

A printed circuit board was to ultimately replace the demonstration board and peripheral electronics. An undiscovered fault on the first two boards built meant that the board was not implemented on final robot testing though initial testing of the board proved that it could be used to effectively control the robot. The PCB provides a cost effective and much smaller replacement to the larger, more expensive demonstration board. Populating the circuit board is also quicker than creating the peripheral circuitry on veraboard. Faults with the board which were discovered during testing were corrected and are documented in Appendix B.

8 Recommendations

8.1 Advanced motion

Further investigation needs to be done to enable the robot to be controlled to move laterally while rotating at all possible lateral speeds. The dependency of the correction factor *Q* on the speed of lateral motion needs to be investigated. The time interval *ti* needs to be further reduced to result in smoother motion and in order to do this the resolution of the wheel speed encoders needs to be increased.

8.2 Robot Platform Control

The kinematic equations derived in Appendix E should be implemented on future omnidirectional robots.

8.3 Wheel Speed Control

8.3.1 Control Algorithm

To prevent wheel slip the robot's control algorithm could be adapted to include a pre-filter to control the amount of acceleration of the wheels.

8.3.2 Loop Tuning

The Ziegler-Nicols method of loop tuning is not recommended. Future loop tuning should be done using more traditional methods such as root locus diagrams.

8.3.3 Wheel Speed Measurement

Since the robot is being used in a system where additional control will be implemented by the image processing system and host computer, shaft encoders are a sufficient method of speed measurement. However, should the robot be used in an environment where it is expected to operate in isolation, it is recommended that an additional odometry system which does not rely on dead reckoning be implemented eg. The optical based system of measurement used on computer mouses.

As already mentioned in Section 8.1, the resolution of the wheel speed encoders should be increased if possible. In the latter half of 2007, an undergraduate project successfully used a novel approach to increase the resolution of the wheel speed encoders motors similar to those used in this thesis. The shaft encoder was mounted on the gearbox side of the motor, before the wheel speed is reduced. This provided a much greater resolution encoder without having to increase the number of teeth on the shaft encoder. The system is pictured in Figure 50. It is recommended that this system be used to increase the resolution of the omnidirectional robot's wheel speed measurements.

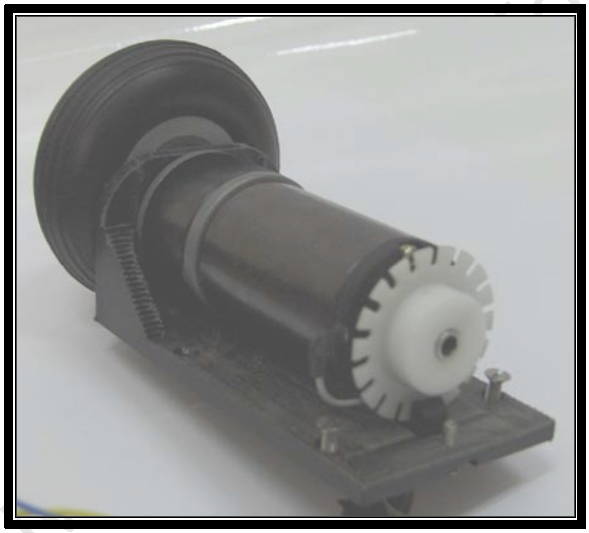


Figure 50: Shaft encoder with increased resolution

8.4 Physical System

It is recommended that the same omnidirectional robot base be used in future omnidirectional robots for UCT robot soccer.

8.5 Onboard Electronics

8.5.1 Digital Signal Processor

The Freescale DSP56F8323 should continue to be used as the controller for future UCT omnidirectional robots.

8.5.2 Power

A battery holder needs to be developed so that the robot need not be powered from a power supply.

8.5.3 Communications

A more reliable serial to wireless converter needs to be found so that the serial cable used for communications can be replaced with a wireless system free of interference.

8.5.4 Motor Drivers

The LMD 18200T h-bridge chips should continue to be used to control the motors.

8.5.5 Encoders

The H21a1 based encoder circuitry should continue to be used as the means of detecting the rotation of the shaft encoder.

8.6 DSP control

8.6.1 Code

The code written to control the robot should be implemented on the final soccer robots. The code can be added to or edited should this become necessary. Since in the final system the host computer will be controlling five different robots, the communications protocol needs to be edited so that commands include which robot commands are intended for.

8.6.2 Command Based Control

The command system implemented is effective and should be retained.

8.6.3 Joystick Control

The Python joystick control program should be reduced in sensitivity so that the robot can be controlled at slower speeds more easily and therefore more accurately.

8.7 Printed Circuit Board

The UCT robot soccer team should aim to use a printed circuit board for its final omnidirectional robot design. The board should be based on the omnidirectional board version1.2 described in Appendix B. Before designing a future board however, it is suggested that the fault on the current board (version1.0) be determined. This would be done by populating another version 1.0 board and investigating it according to the suggestions in Appendix B. It is also suggested that future versions of the PCB have separate boards for the H-bridge circuits and electrically isolate their signals from the DSP with optocouplers.

9 References

1. *RoboCup Website.* www.RoboCup.org, accessed Jan 2008.
2. McPhillips, G., The Control of Semi-Autonomous Robots, in Department of Electrical Engineering. 2004, University of Cape Town.
3. Craig Inman-Bamber, *Robot Soccer Platform*, in *Mechanical Engineering Department*. 2005, University of Cape Town.
4. Upsher, L., Soccer Robot Kicking and Dribbling System, in Mechanical Engineering Department. 2005, University of Cape Town.
5. *small robot league website.* http://small-size.informatik.uni-bremen.de, accessed: Jan 2008.
6. *Trossen Robotics - Mecanum Wheels.* http://blog.trossenrobotics.com/index.php/2006/12/11/mecanum-wheels/, accessed: Jan 2008.
7. CoralResearchGroup, *Robot Soccer, Carnegie Mellon University, Small Size Team.* http://www.cs.cmu.edu/~robosoccer/small/, accessed: Jan 2008.
8. *Texas Instruments homepage.* www.texasinstruments.com, accessed: Feb 2008.

Appendix A Literature Review

Table of Contents

Table of Figures

Appendix A. Literature Review

A.1 Introduction

Conventional rolling robots, like conventional vehicles can only move in a direction parallel to the wheel hub. These robots are termed non-holonomic robots. A holonomic or omnidirectional robot is designed in such a way that it is able to move in any direction in a two dimensional plane [1]. This Chapter discusses a number of omnidirectional platforms, with a specific focus on omnidirectional robots.

A.2 Types of Omnidirectional Wheels

Probably the first omnidirectional wheel to be patented is that shown in Figure A-1 below. The wheel was patented by J. Grabowiecki in 1919. It consisted of a hub with 8 rollers placed at a 90° angle [2] [3]. More well known is the patent taken out by inventor Bengt Ilon and the Swedish company he worked for , Mecanum AB, in 1973 [4]. For this reason omnidirectional wheels are often referred to as Mecanum wheels, and even Ilonator wheels [5].

Omnidirectional wheels can be orientated at 90° to the wheel hub as are those designed by Craig Bamber and used in this project (Figure A-3), or at 45° as shown in Figure A-2 below.

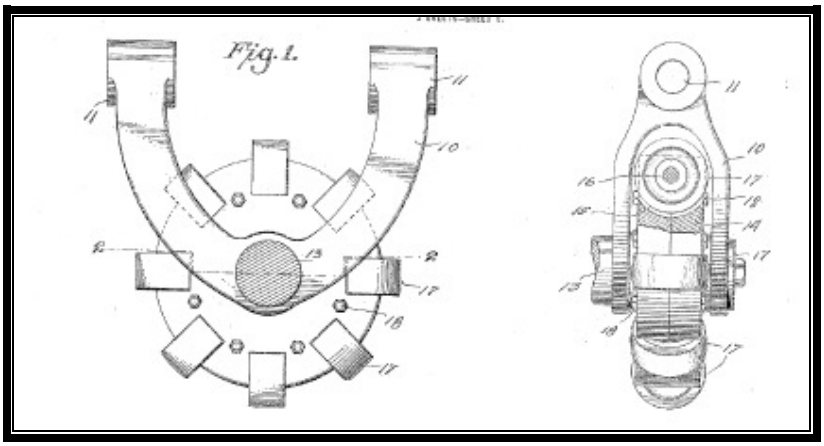


Figure A-1: Drawing from Grobowieki's 1919 patent for an omnidirectional wheel [3]

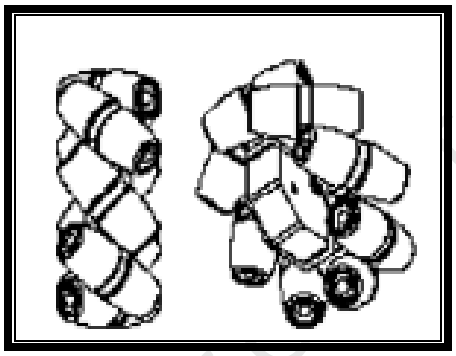


Figure A-2 Diagram of a 45º omnidirectional wheel [6]

A.2.1 90° Omnidirectional Wheels

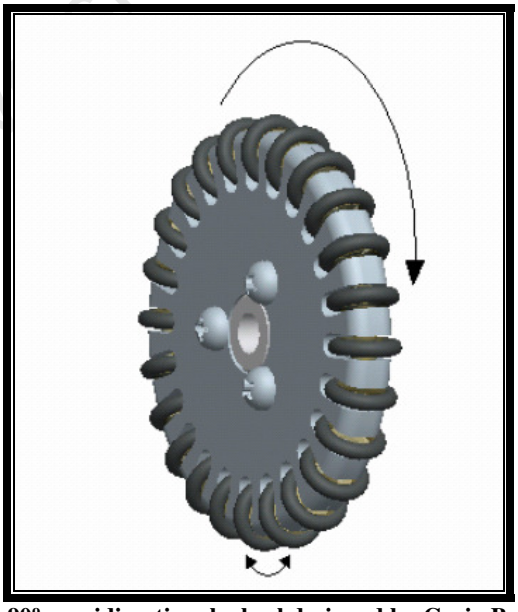


Figure A-3: 90º omnidirectional wheel designed by Craig Bamber[7]

Figure A-3 above is a diagram of the omnidirectional wheels designed and manufactured by Craig Bamber which are used in this thesis. Each wheel consists of a single hub with rollers which roll perpendicular to the hub. These wheels can be used in either three or four wheel configurations (see Figure A-4 and Figure A-5 below). The only literature that could be found comparing the three and four wheel configurations was Mr Bamber's Thesis. He pointed out that the four wheel configuration takes up more space than the three wheel configuration since each wheel needs its own motor. This leaves less space for the kicker and dribbling systems which later need to be added to this robot platform [7][pg8]. Also to be considered is that the four wheel configuration can give a greater acceleration than the three wheel configuration. However, given the volume restrictions of the small size robosoccer league the space saving of the three wheel configuration would be more advantageous than the added acceleration of the four wheel configurations. Therefore it was decided to re-use Mr Bamber's three wheel configuration platform.

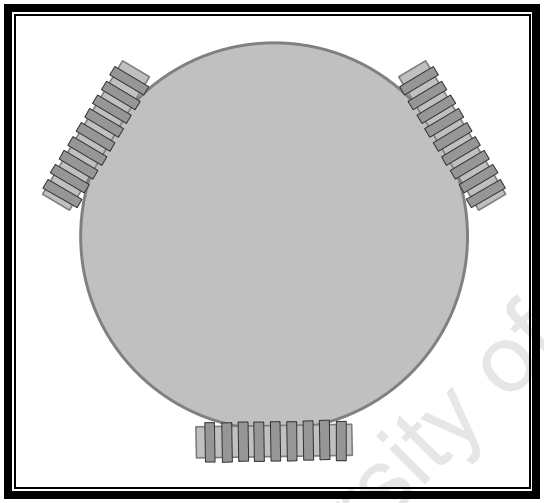


Figure A-4: Three wheel omnidirectional robot

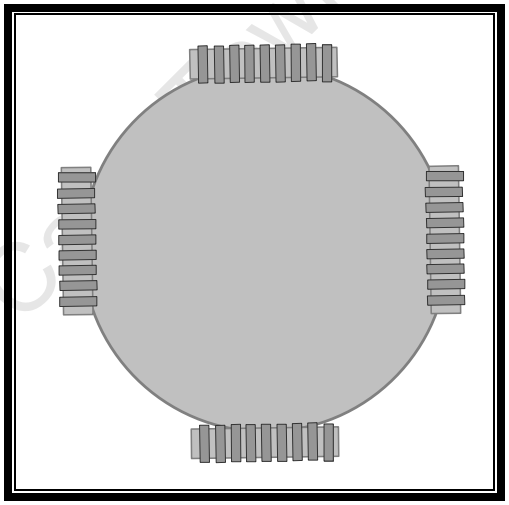


Figure A-5: Four wheel omnidirectional robot

A.2.2 45° Omnidirectional Wheels [1][pgs114-117]

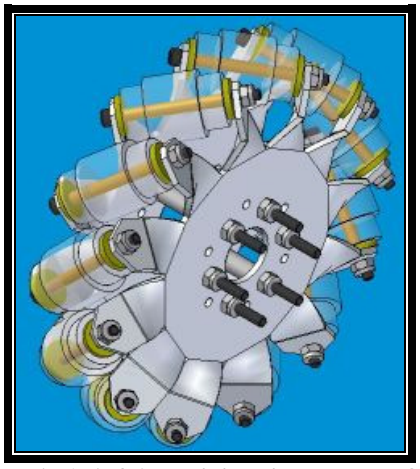


Figure A-6: A 45° omnidirectional wheel [8]

45° omnidirectional wheels are used on four wheeled robots. There are right handed and left handed versions of the 45° wheels and to drive a robot one needs two of each. The left handed wheels have rollers at +45° to the wheel axis while the right handed wheels have rollers at -45° to the wheel axis. The arrangement of the wheels is shown in Figure A-7.

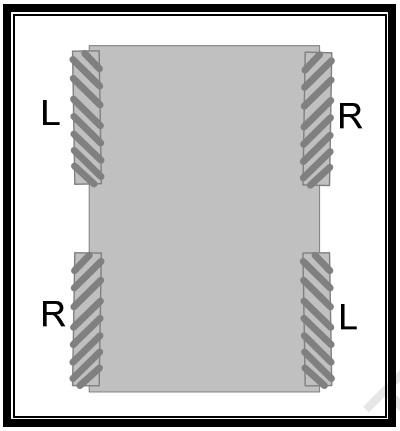


Figure A-7: Wheel Configuration of 45º omnidirectional robot

Figure A-8 below illustrates basic motions of such a robot. To drive the robot forward, all four wheels are driven forward. To drive left, wheels 1 and 4 are driven forward but wheels 2 and 3 are driven backwards. To rotate on the spot the wheels on one side are driven forward and the wheels on the other side driven backwards. Braunl gives the complete forward and inverse kinematics in his book Embedded Robotics [1][pgs117,118].

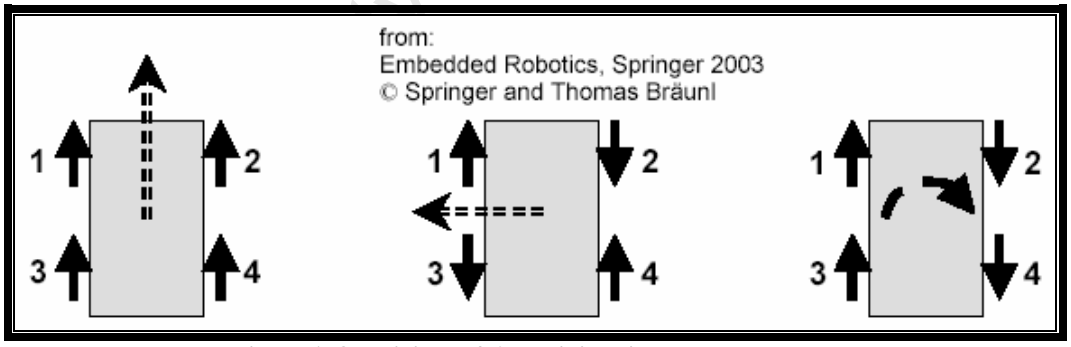


Figure A-8: Driving a 45º omnidirectional robot

A.3 Omnidirectional Vehicles

A.3.1 Airtrax

The company Airtrax has developed 3 different omnidirectional vehicles for commercial use. The sidewinder (Figure A-9) is an omnidirectional forklift. The cobra (Figure A-10) is an omnidirectional aerial work platform and the MP2 (Figure A-11) is a adaptable omnidirectional platform originally developed for the military. These vehicles allow access to small spaces and manoeuvrability unattainable by traditional industrial vehicles [9]. These vehicles are controlled by an operator using a joystick. The vehicle moves in the direction that the joystick is pushed. Twisting the joystick causes the vehicle to rotate.

Figure A-9: The Sidewinder [9]

Figure A-10: The Cobra [9]

Figure A-11: The MP2 [9]

A.3.2 Omnidirectional Wheelchairs

Being able to move in an omnidirectional fashion would make conventional wheelchairs far more manoeuvrable, especially within restricted spaces. Research into electric wheelchairs with omnidirectional capabilities has been conducted at The universities of Hagen and Western Australia. This literature search could not find any omnidirectional wheel chairs that are available commercially.

The Control Systems engineering group at the university of Hagen have developed the prototype omnidirectional electrics wheelchair shown in Figure A-12 [10, 11]. The Robotics and Automation Lab have designed the wheelchair shown in Figure A-13 [12]. Both designs use four 45º omnidirectional wheels. They also both use joysticks for the user to control the chair.

Figure A-12: Omnidirectional wheelchair developed by Hoyer et al at the University of Hagen [10]

Figure A-13: Omnidirectional wheelchair developed by the Robotics and Automation Lab at the University of Western Australia [12]

A.4 Omnidirectional Robots

In addition to the omnidirectional wheelchair, the University of Western Australia Robotics and Automation Lab has also developed two omnidirectional robots called Omni1 and Omni2 (pictured in Figure A-14 and Figure A-15 below). The rimmed design of the Omni1 wheels make it only successful on relatively smoother surfaces. The Omni2 improves on this system by adding suspension and redesigning the omnidirectional wheels so that they are rimless. This enables the Omni2 to drive over much rougher terrain. The controller for both robots is the EyeBot controller developed by Prof Thomas Braunl and based on a 32 bit microprocessor [13].

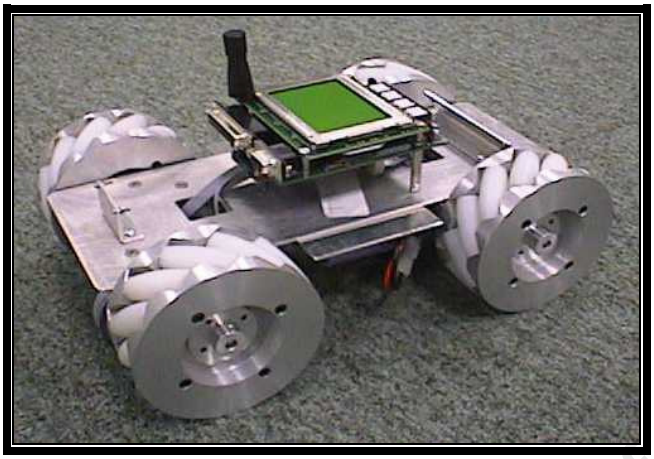


Figure A-14: Omni 1. First omnidirectional robot developed at UWA [14]

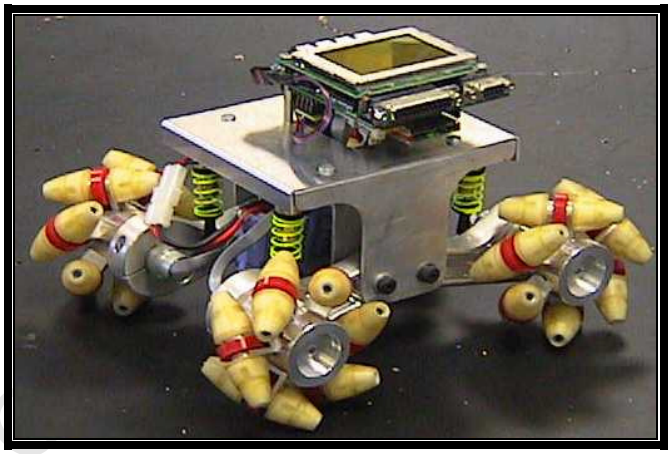


Figure A-15: Omni 2. An improved omnidirectional robot from UWA [15]

A.5 Omnidirectional Soccer Robots

A.5.1 Cornell University – Big Red

The first team to introduce omnidirectional robots into the robot cup competition was the Big Red team from Cornell University in 2000. The advantage gained is clear by the fact that they won the competition that year and the following year more teams entered the competition with omnidirectional robots - now most robots in the league are omnidirectional [16] [17].

Originally, Cornell's omnidirectional robot had three wheels but in 2001 Cornell developed their first four wheel omni-directional robot since, although more complicated to control, it allowed for greater acceleration [18] .

In 2004 Cornell moved away from centralised control - instead of the artificial intelligence being based on an external host computer which told the robots what to do, the robots ran Windows CE on PC104 computers which allowed them to act independently of a host computer, more like real soccer players.

The Cornell team retired after 2005.

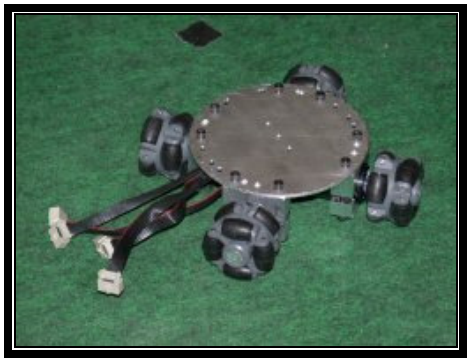


Figure A-16: Cornell's four wheel omnidirectional robot introduced in 2001 [18]

Figure A-17 : Cornell's 2001 Robocup team [18]

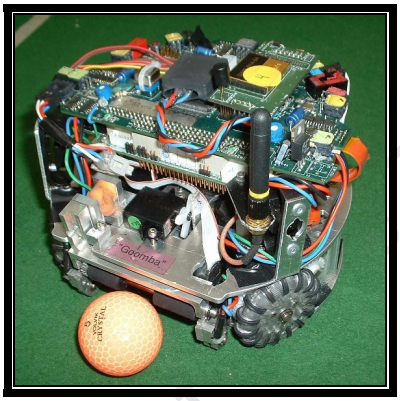


Figure A-18: 'April, 2004: Cornell completes Windows CE and PC104 based robots, which distribute intelligence to the players for the first time.' [18]

A.5.2 Carnegie Melon – CMDragons

The team that have come to the fore since Cornell's retirement is the CMDragons from Carnegie Melon University. They have competed since 1997 but started using omnidirectional robots in 2003 [19].

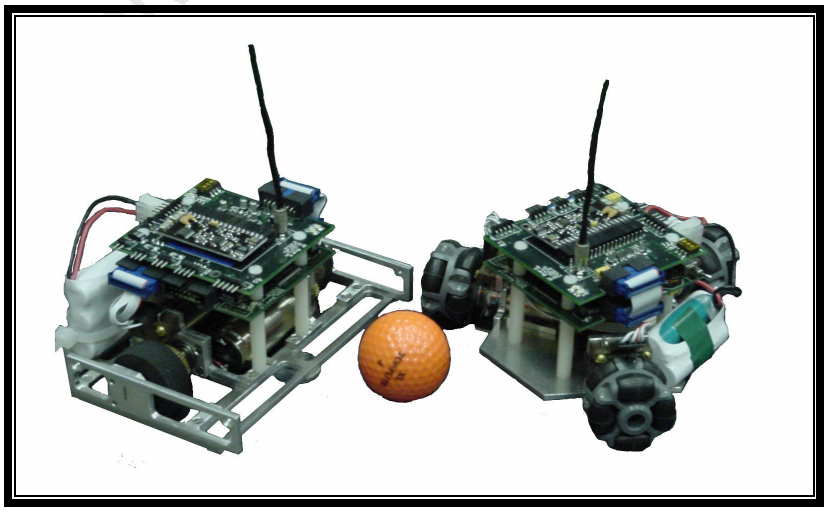


Figure A-19: Carnegie Melon's 2003 Differential drive robot (left) and omnidirectional robot (right) [19]

A.6 Omnidirectional Literature

A number of papers were read in an attempt to gain a greater understanding of omnidirectional robots and especially of their control. One question of interest was the difference between using 90º and 45º omnidirectional wheels as well as between three wheel and four wheel designs. The author was interested to find out why one design had been chosen over another. Also of interest were the kinematic equations used to control the robots, it was hoped that an easily implementable algorithm could be found with which to control the omnidirectional platforms. Below are summaries of three papers discussing omnidirectional platforms.

A.6.1 Buhler et al

In their paper published in Robotics and Autonomous Systems, Buhler et al [11] describe the technology available for advanced wheelchairs. They describe the use of an omnidirectional platform to make an electric wheelchair. Their prototype proved highly manoeuvrable and they believed it would provide wheelchair users with access to confined areas not accessible from a conventional electric wheelchair. The control loop used to control the individual wheels of the wheelchair is introduced. It consists of an inner wheel velocity loop and an outer wheel position control loop. They describe both a four wheel and three wheel design using 45° omnidirectional wheels. They give equations relating the wheel velocities needed to obtain specific velocities and rotation of the platform.

A.6.2 Saha et al

Saha et al [20] from McGill University's Centre for Intelligent Machines discuss the kinematic and inverse kinematic equations of omnidirectional vehicles. Unlike other papers which discuss omnidirectional wheels with rollers mounted either at 90° or 45° to the wheel hub, Saha et al attempt to define the kinematics for a robot with its wheels mounted at some arbitrary angle, α, to the wheel hub. They derive kinematic equations dependent on the angle α, the ratio of roller radius to hub radius and the number of rollers per hub. They also consider the number of wheels, the location of the wheels on the body of the robot, and how many wheels are actuated. Although they derive a number of equations relating these facts, no conclusions are drawn as to the effects of parameters such as the number of wheels, the placement of wheels on the robot body or the optimum angle α.

A.6.3 Feng et al

Feng et al [5] developed an omnidirectional wheel base at Carnegie Melon University's Robotics Institute. Their omnidirectional base uses four wheels with rollers at 45° to the wheel hub. Their robot can be controlled using a joystick or by downloading trajectory commands onto the onboard computer. The paper includes a statement of the inverse kinematics of the wheel base.

A.6.4 Conclusion

No recommendations could be found as to the whether 45° or 90° angled wheels are optimal or why. Nor could a comparison between 90º wheeled robots with three wheels and those with four wheels be found.

No clear algorithm for controlling an omnidirectional robot was found in the papers studied. It is the opinion of this author that the methods followed by the papers reviewed to derive kinematic equations with which to control omnidirectional robots may by more complex than necessary. This author successfully implemented a simpler derivation of the kinematic equation. This derivation is given in appendix E.

A.7 Conclusions

From the literature review it is clear than omnidirectional platforms are useful, not only in the field of robotics but in any instance where it is necessary for a moving platform to be highly maneuverable (eg wheelchairs and fork lifts). Roboticists have been researching omnidirectional robots for some time and have found them to be particularly successful in the robot soccer small size league.

Much information seems to be available on the more complex 45º omnidirectional wheels and the use of four 90º omnidirectional wheels than on the simpler, more compact three wheel 90º design. The literature tends to describe either 45º or 90º platforms but does not discuss the differences in the designs nor if one design has any advantage over the other. 90º omnidirectional designs can be in three or four wheel orientations but again, no literature could be found discussing the advantages of one over the other. In the interest of finding an elegant, simpler, smaller omnidirectional design, this project has continued the work begun by Mr Bamber in exploring the three wheel 90º design. Mr Bamber's investigation showed no reason not to pursue the three wheel design.

The literature suggests various options as to how to control an omnidirectional robot. Most consist of complex inverse kinematic equations. It was decided to try a simpler equation for controlling the robot platform, this system is explained in Appendix E.

It should be noted that the makers of both the industrial vehicles and wheelchairs researched chose to use joysticks to direct the robots. Moving the joystick laterally results in motion in that direction while twisting the joystick results in rotation.

References

1. Braunl, T., *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. 2nd ed. 2006: Springer.
2. Rojas, R., *A short history of omnidirectional wheels.* http://robocup.mi.fu-berlin.de/buch/shortomni.pdf, Accessed: November 2007.
3. Grabowiecki, J., *"Vehicle Wheel"*. June 3, 1919: US Patent 1 303 535.
4. Ilion, B.E., *Wheels for a Course Stable Selfpropelling Vehicle Movable in any Desired Direction on the Ground or some other Base*, U.S. Patent, Editor. 1973.
5. Feng, D., M.B. Friedman, and B.H. Krogh. *The servo-control system for an omnidirectional mobile robot*. in *International Conference on Robotics and Automation (ICRA '89)*. 1989.
6. Tlale, N.S., *On distributed mechatronics for omni-directional autonomous guided vehicles.* Industrial Robot, 2006. **33**(4): p. 278-284.
7. Craig Inman-Bamber, *Robot Soccer Platform*, in *Mechanical Engineering Department*. 2005, University of Cape Town.
8. *Trossen Robotics - Mecanum Wheels.* http://blog.trossenrobotics.com/index.php/2006/12/11/mecanum-wheels/, accessed: Jan 2008.
9. Airtrax, *Airtrax Vehicles.* www.airtrax.com/vehicles, accessed Jan 2008.
10. Hoyer, H., U. Borgolte, and A. Jochheim. *The OMNI-Wheelchair - State of the Art*. in *The Centre on Disabilities 14th annual international conference*. 1999. Los Angeles. Accessed via: http://www.dinf.ne.jp/doc/english/Us_Eu/conf/csun_99/session0274.html, Jan 2008.
11. Buhler, C., et al., *Autonomous robot technology for advanced wheelchair and robotic aids for people with disabilities.* Robotics and Autonomous Systems, 1995. **14**: p. 213-222.
12. Robotics and Automation Lab, U.o.W.A., *Omni-Directional Wheelchair with Driver-Assistance System.* http://robotics.ee.uwa.edu.au/omni/omni3.html, Accessed: Jan 2008.
13. Robotics and Automation Lab, U.o.W.A., http://robotics.ee.uwa.edu.au/omni/omni1.html, Accessed: Jan 2008.
14. Robotics and Automation Lab, U.o.W.A., *Omni-1.* http://robotics.ee.uwa.edu.au/omni/omni1.html, Accessed: Jan 2008.
15. Robotics and Automation Lab, U.o.W.A., *Omni-2.* http://robotics.ee.uwa.edu.au/omni/omni2.html, Accessed: Jan 2008.
16. *Cornell student RoboCup team wins world title for the fourth time*, in *CornellNews* *http://www.news.cornell.edu/releases/July03/RoboCup03.ws.html*. 22 July 2003, accessed Jan 2008.
17. Saulnier, B., *Robocup Retrospective*, in *Cornell Engineering Magazine*. http://www.engineering.cornell.edu/news/engineering-magazine/archives/cem-spring-2006/RoboCup-Retrospective.cfm, accessed Jan 2008.

18. *Cornell Robocup Website.* www.cis.cornell.edu/boom/2005/ProjectArchive/robocup/history.php, accessed Jan 2008.
19. CoralResearchGroup, *Robot Soccer, Carnegie Mellon University, Small Size Team.* http://www.cs.cmu.edu/~robosoccer/small/, accessed: Jan 2008.
20. Saha, S.K., J. Angeles, and J. Darcovich, *The design of kinematically isotropic rolling robots with omnidirectional wheels.* Mech. Mach. Theory, 1995. **30**(8): p. 1127-1137.

Appendix B. Robot Platform – On Board Electronics

Table of Contents

Table of Figures

Appendix B.Robot Platform - On Board Electronics

B.1 Introduction

This chapter discusses the electronics used to operate the robot, from the DSP used to control the robot, to the encoder circuitry used to measure wheel speeds. Two systems for the control of the robot platform were developed. One system uses a generic development board with peripheral electronics on veraboard. The second system consists of a single printed circuit board (PCB) designed to include all the necessary circuitry to control the robot, including the DSP.

Figure 20, below, is a block diagram of the development board system, consisting of the development board and the peripheral electronics. Figure 21 below, shows a block diagram of the PCB. Both contain the same or very similar components but the prototype consists of 9 separate boards connected by cables while the PCB has all components on a single board. The combined area of the prototype electronics is approximately 250cm^2 whereas the PCB has all the same functionality but has an area of only 80cm^2.

Central to the circuitry on the boards is the DSP, this controls the robot's movements by interacting with the peripheral circuits. H-bridge circuits drive the motors, allowing one to control the speed and direction of the motors using pulse width modulation. To determine what speed the wheels are going at, optical encoders are used. To tell the robot what it needs to do and where it needs to go, commands are sent from a PC to the robot via SCI. All of these circuits need power at specific voltages which requires voltage regulation. This appendix discusses these circuits. First the circuits are explained then the differences between the development board and PCB versions are

discussed. The appendix continues with a discussion of the layout of the PCB and concludes with a report on the final PCB, corrections that needed to be made to it, and suggestions for an updated version of the board as well as a report on debugging an error on the PCB. The circuit diagrams for each module are shown when they are discussed. For a complete circuit diagram please see the schematic in Appendix H.

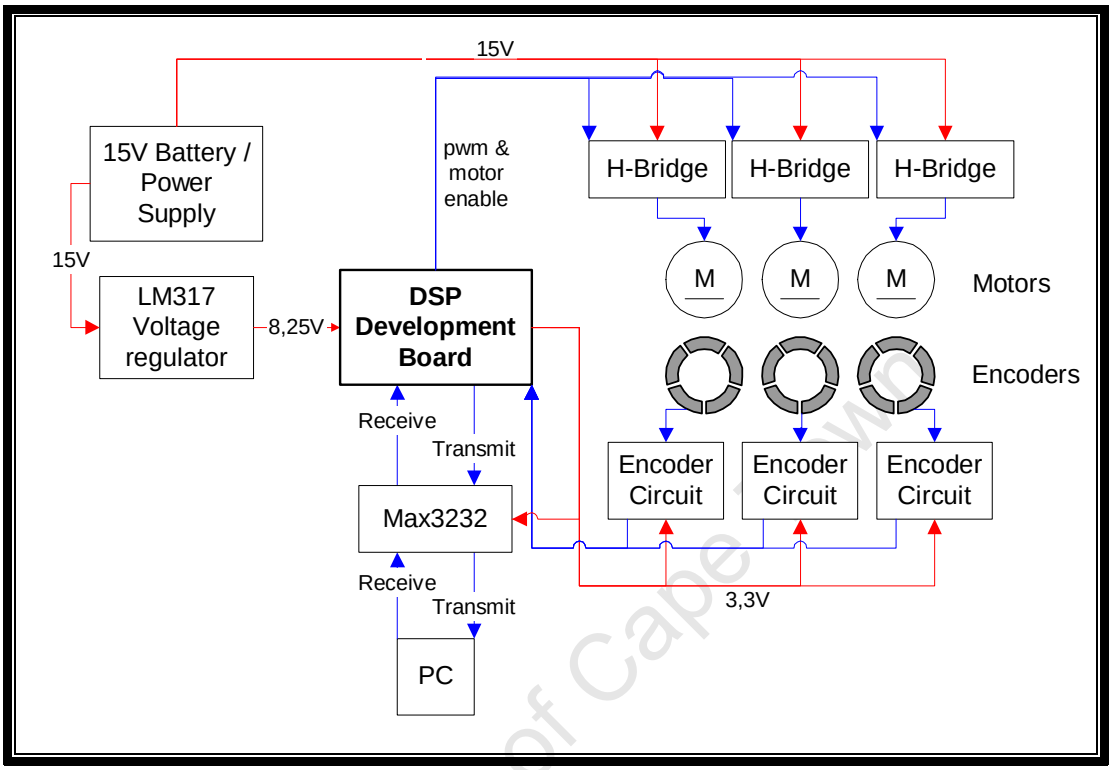


Figure B-1 Block Diagram of prototype electronics.

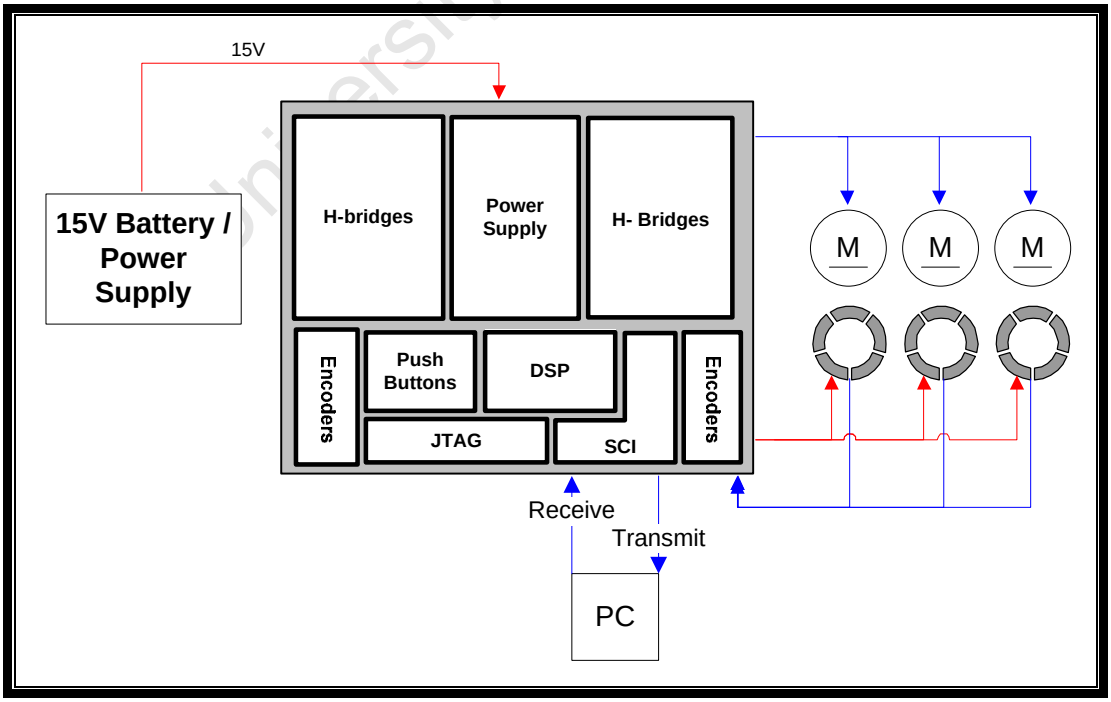


Figure B-2: Block diagram of PCB

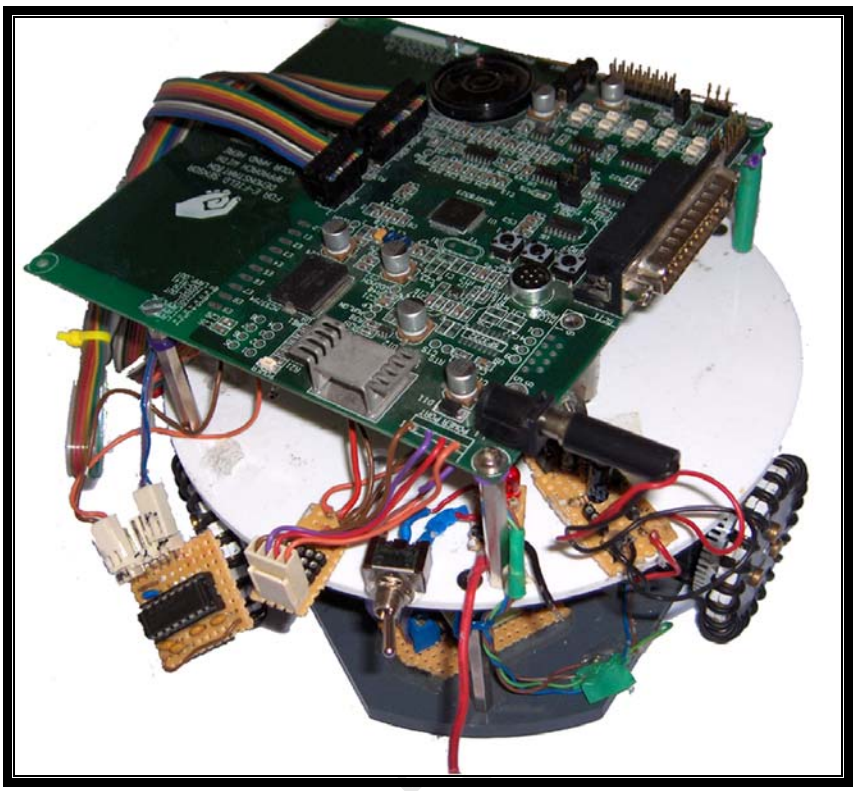


Figure B-3: Robot with development board and peripheral electronics connected

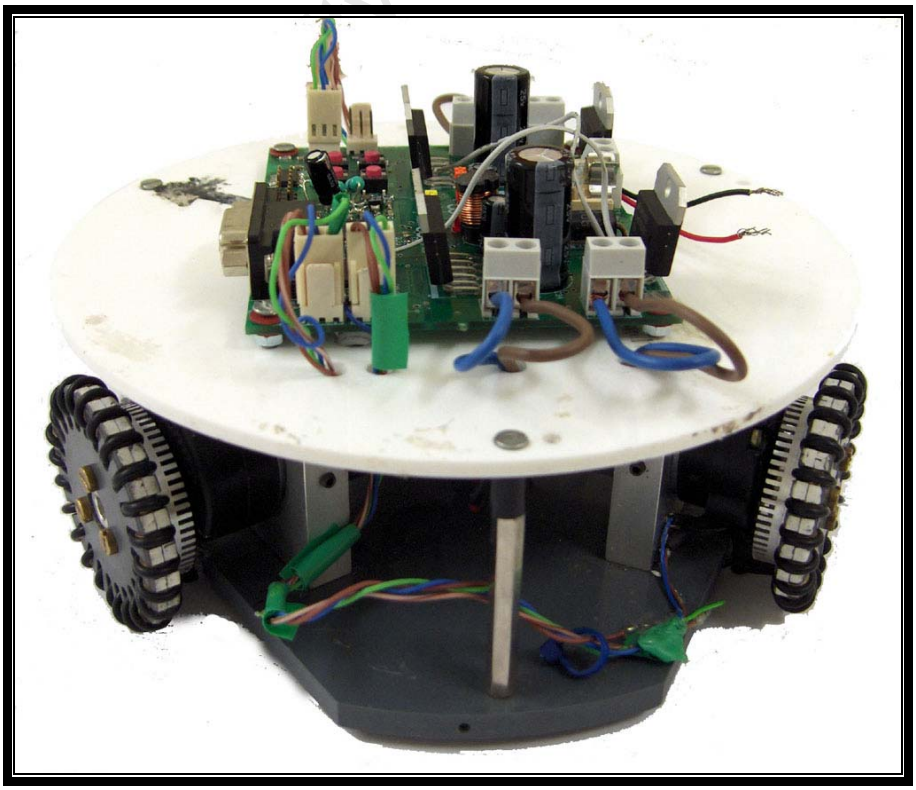


Figure B-4: Robot with PCB connected

B.2 Controller

B.2.1 Controller Choice

The controller was chosen according to suitability and availability. It was decided to use Freescale controller since a number of members of the Mechanical Engineering Department had experience using Freescale controllers – so help was available in the development stages. Using a development board consisting of a microcontroller or DSP on a printed circuit board with connection to the microcontroller's pins as well as useful peripheral circuits, can reduce development time significantly. The two development boards available in the department were for the MC68HC908GT16 and DSP56F8323 processors. The GT16 is a smaller, slower processor with an 8-MHz internal bus[1]. The 56F89323 is a Digital Signal Processor (DSP) which can operate at the much faster frequency of up to 60Mhz and has more memory [2].

Table 4 shows a comparison of some of the more important specifications of the two processors.

Table B-1 Comparison of DSP56F and GT16.

	MC68HC908GT16	DSP56F832
Speed	8MHz	40-60MHz
Internal Flash Memory	16kB	32kB
Timer Channels	2	8
IO Pins	36	27
ADCs	1 x 8 Channel, 8 bit	2 x 4 Channel, 12 bit

The DSP clearly has more processing power thanks to its faster operating speed and larger amount of onboard memory, allowing for the storage of larger programs. The DSP also has greater resolution analogue to digital converters (ADCs). The two main disadvantages of the DSP are price and complexity. Because of the extra processing power and number of peripherals the DSP is more expensive than the microcontroller, and since one controller would be needed each of the five robots in the Robocup team, it would be preferable to keep those costs low. The DSP is also more complex than the GT16, which will increase the development time involved.

It was decided that the increased functionality would make up for the increase in development time and that despite the desire to keep the costs down, the control algorithms what are needed for an omnidirectional robot are complicated and require a lot of processing power and complex algorithms that may take up a lot of RAM. It is possible that the GT16 would prove powerful enough for the functionality needed, but not being able to know in advance how much RAM

would be needed for the code or how fast the processor would be required to run to control the robot effectively, it was decided to use the more powerful, though more expensive DSP. It was decided that it would be preferable to have more processing power and memory than needed rather then too little. Also, since the DSP and GT16 are both Freescale devices, should the final code prove to be small and un-demanding enough, the code from the DSP could be adapted for the smaller microcontroller.

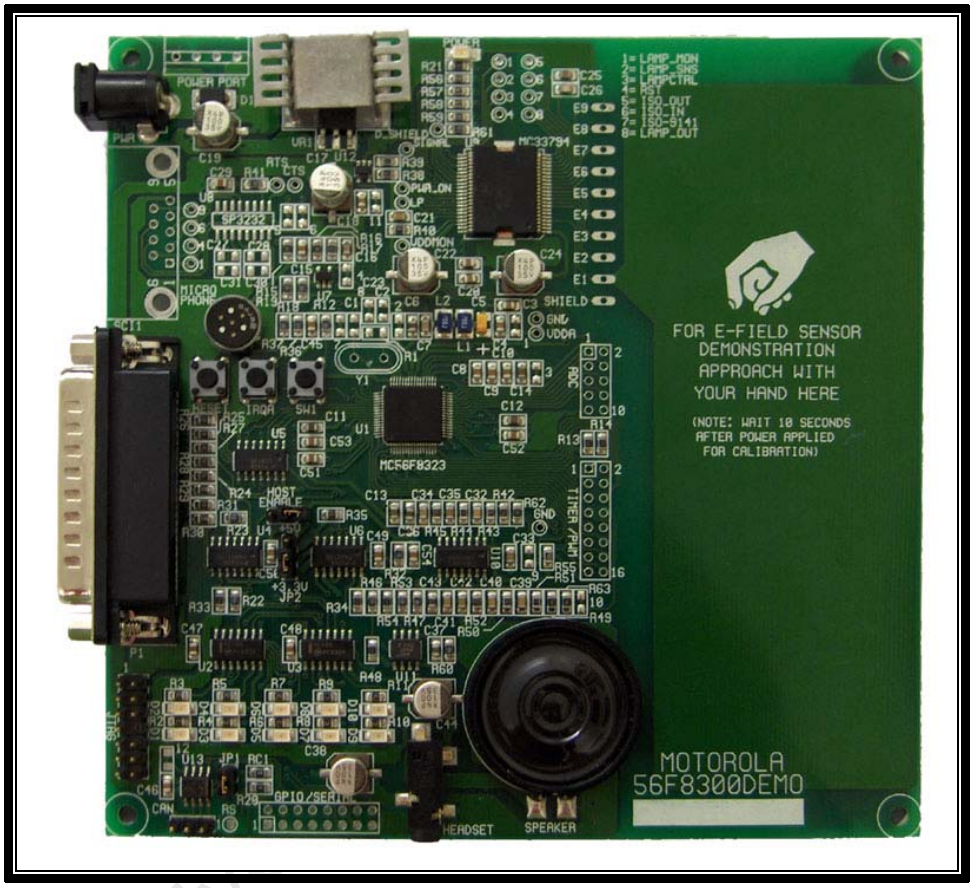


Figure B-5: The Freescake DSP56F8300 demonstration board

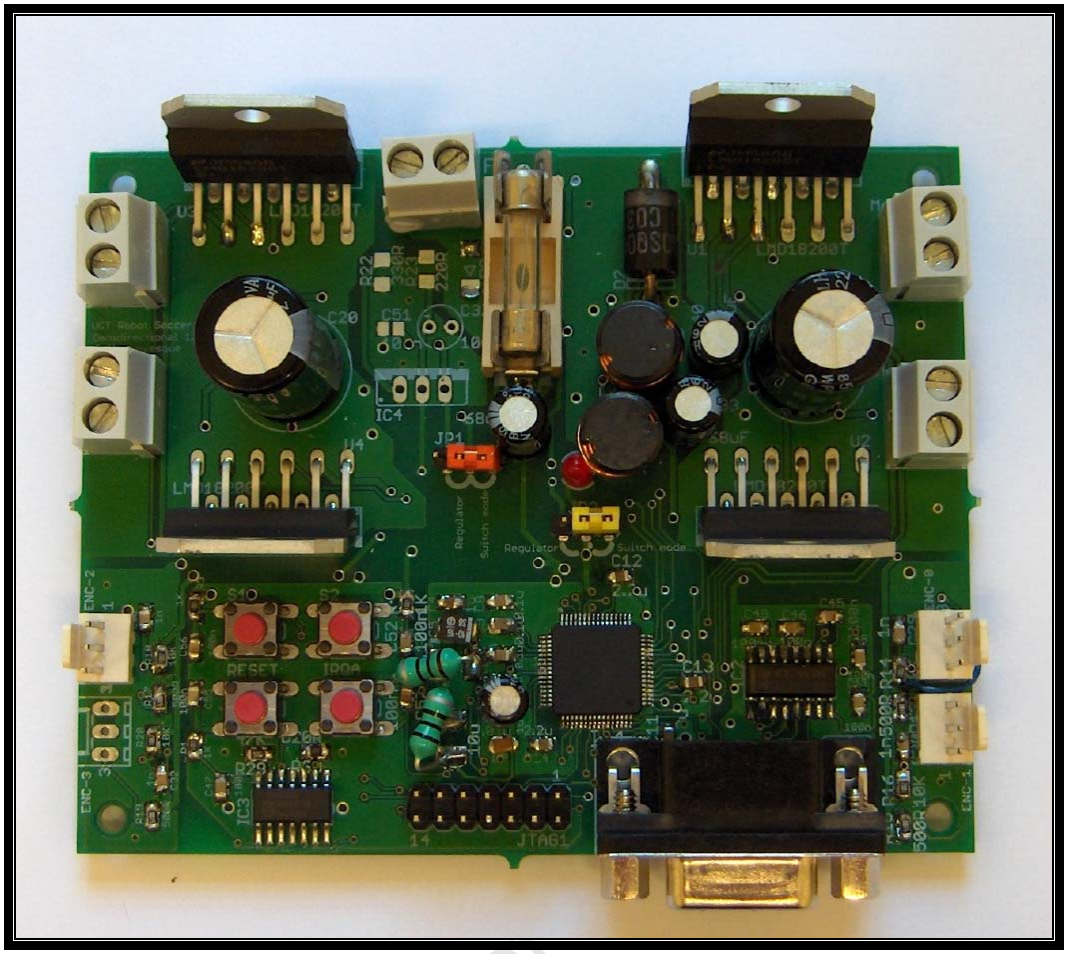


Figure B-6: Printed Circuit Board Designed to Control Omnidirectional Robotic Platform

The demonstration board available for the DSP56F8323 was the Motorola 56F8300DEMO shown in Figure B-5. This board has connections to the GPIO, PWM, SCI and ADC pins of the board as well as both parallel port and JTAG programming interfaces. The board also includes LEDS and switches for debugging purposes as well as an on board power supply. Using the development board meant that the development of peripheral circuitry and the code to interact with the peripherals could begin immediately, without first having to design the circuitry needed for the DSP to work. Once there was a working system using the development board, the dedicated printed circuit board was designed. The dedicated PCB had only those components needed to run the robot platform, making it much smaller than the development board. Figure B-6 above shows the PCB.

B.2.2 Placing the Controller on the PCB

The layout of the DSP and peripheral electronics needed for its operation were based on the Freescale 56F8300 Demonstration Board [3]. The following features of the demo board were not included on the Robot Board: speaker and associated audio features, parallel port, 5V supply. Table B-2 below documents the connections between the DSP and the other on board circuitry.

Table B-2: Output and Input Lines of DSP

DSP Pin	Connection	Function
Motors		
PWMA0	Motor 1 - Direction	PWM for Driving Motor 1
PA5	Motor 1 - PWM	Enables Motor 1
PWMA1	Motor 2 - Direction	PWM for Driving Motor 2
PA6	Motor 2 - PWM	Enables Motor 2
PWMA2	Motor 3 - Direction	PWM for Driving Motor 3
PA8	Motor 3 - PWM	Enables Motor 3
PWMA3	Motor 4 - Direction	PWM for Driving Motor 4
PA7	Motor 4 - PWM	Enables Motor 4
Encoders		
ANA0	Encoder 0	Signal from Wheel Encoder 0
ANA1	Encoder 1	Signal from Wheel Encoder 1
ANA2	Encoder 2	Signal from Wheel Encoder 2
ANA3	Encoder 3	Signal from Wheel Encoder 3
Serial Communications		
TXD0	Max3232 - T1IN	DSP transmit line
RXD0	Max3232 - R1Out	DSP receive line
JTAG		
TDI	JTAG port - 1	JTAG Test Data Input
TDO	JTAG port - 3	JTAG Test Data Output
TCK	JTAG port - 5	JTAG Clock
TMS	JTAG port - 10	JTAG Test Mode Select
TRST*	RESET Switch OR JTAG 14	Can be triggered by RESET switch or JTAG
Switches		
RESET*	RESET Switch AND JTAG	Can be triggered by RESET switch or JTAG
IRQA*	IRQA Switch	Switch can be used to trigger external interrupt
PB6	Switch 1	Multi purpose switch
PB7	Switch 2	Multi purpose switch

B.2.3 Programming Interface

When using the development board, the DSP was programmed using the parallel port interface on the development board. On the PCB it was decided to instead program using the JTAG interface. This was because the parallel port plug and peripheral electronics would take up a lot of space on the board, whereas the JTAG needs a much smaller connector and the electronics for programming are all available on an external module – the CodeWarrior USB TAP. Figure B-7 below is a schematic of the connections for the JTAG.

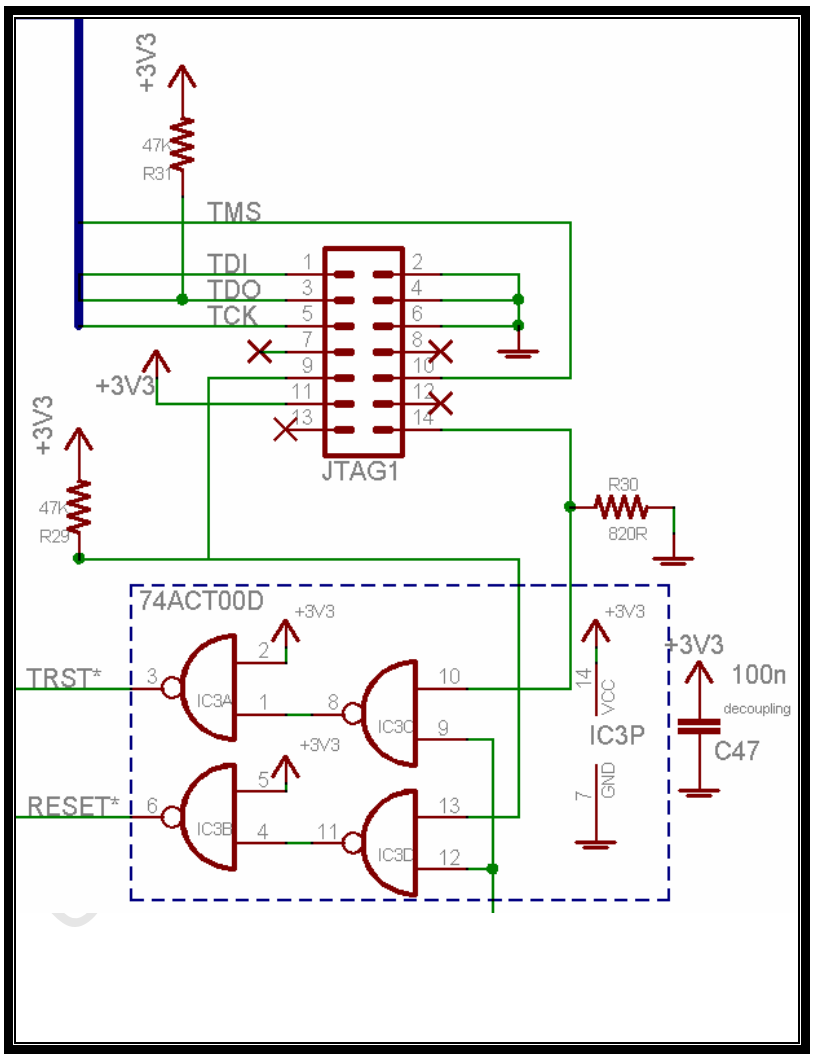


Figure B-7: Connections for the JTAG interface on the PCB

Once the board had been made and populated, programming of the board using the JTAG connection and the USB TAP module was attempted. This was unsuccessful, because of a fault with the USB TAP module used (the integrated development software could not establish a connection with the board using USB TAP module). The parallel port now needed to be used to program the board. However circuitry is needed to buffer between the 3,3V lines of the microcontroller and 5V used by the parallel port. This circuit was based on the schematic for the Freescale development board [4]. Instead of buffer ICs however, resistors were used to drop the voltage from the 5V to the 3,3V. Below is the

schematic of this circuit. Since the parallel port programming interfaces with the same lines as JTAG programming, the circuit could plug directly into the JTAG port of the PCB. A cable was made to connect a parallel port connector to the JTAG port on the PCB with the necessary circuit in between. This cable is shown in Figure B-9 below. The one modification required from the board was that resistor R30 be changed from a 820R resistor to a 3,3K resistor.

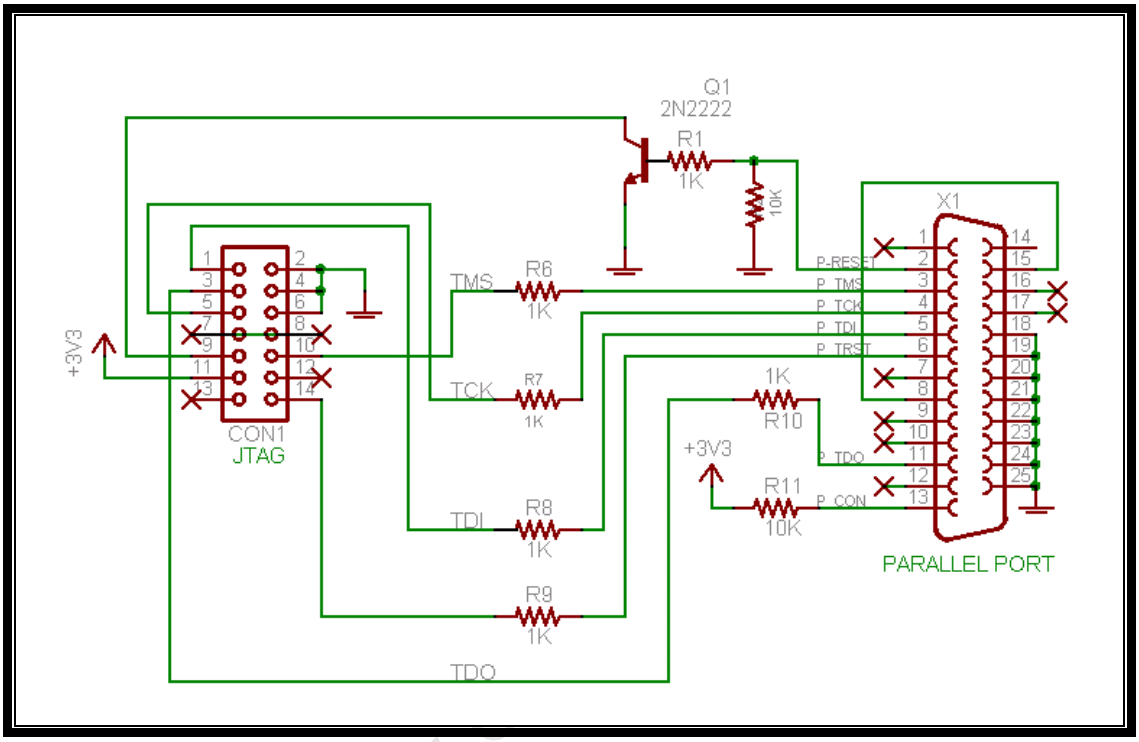


Figure B-8: Circuit for connecting parallel port interface to JTAG port of PCB

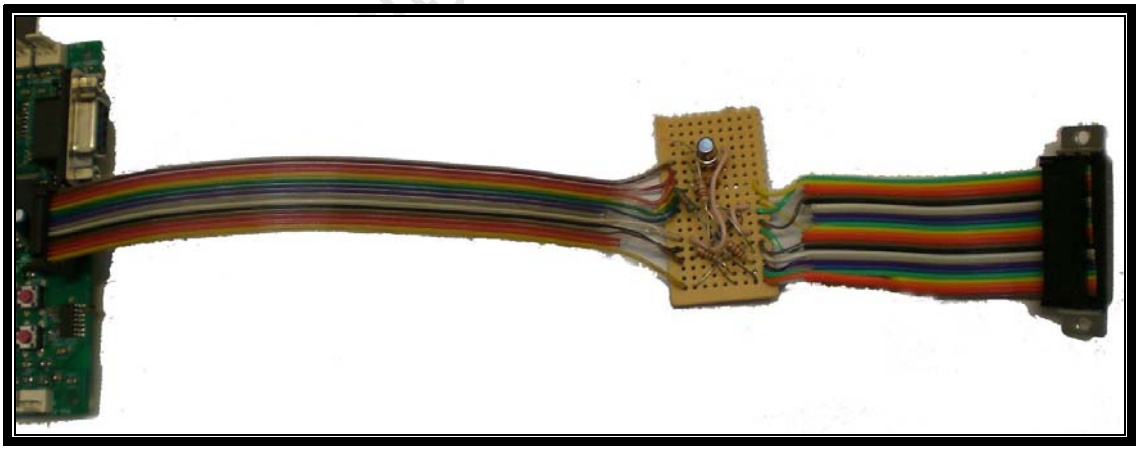


Figure B-9: Programming cable

B.3 Power

Power needed to be supplied to both the electronics that control the robot and the motors for the wheels. Since electronics generally operate on much lower voltages than motors, any power supply would have to include some voltage regulation (or two different power sources) to provide suitable voltages for each system.

Since the DSP operates on 3,3V it was decided to operate all the electronics at that voltage. On testing the motors it was found that they needed to operate on at least 10V, but closer to 15V, to reach any appreciable speed.

The previous robot soccer robot used a 16V battery pack made up of 14 1.2V Nichol Cadmium batteries to supply the motors. The power for the electronics was tapped off this pack at 8V which meant that some of the batteries were discharging faster than others - as a result some batteries would have a shorter lifespan than others. Mr McPhillips asked that the new robot use a more suitable power system. Mr McPhillips suggested investigating alternative power solutions, including having separate battery packs for electronics and drive motors and having a single battery voltage which is either stepped up for the drive motors or stepped down for the electronics. Mr McPhillips had found alternative batteries which were more suitable in terms of price, size and ease of charging and he asked that these be integrated into the new system. Also it was asked that the layout of the robot be such that the batteries would be easily accessible, so that the robot needn't be disassembled in anyway to be charged.

The new batteries were 7.4V lithium ion batteries. Measurements showed the voltage range from these batteries to be between 7.4V and 8.0V. Connecting two of these batteries in series would give a voltage of between 14.8V and 16V - a voltage large enough to run the DC motors successfully. To avoid having to recharge the batteries often during testing, a power supply was used during most to provide 15V to the robot.

The peripheral electronics needed to operate the robot needed a 3,3V supply. What follows is a description of how that 3,3V was generated from the 15V battery supply in both the prototype and final configurations.

B.3.1 Development Board

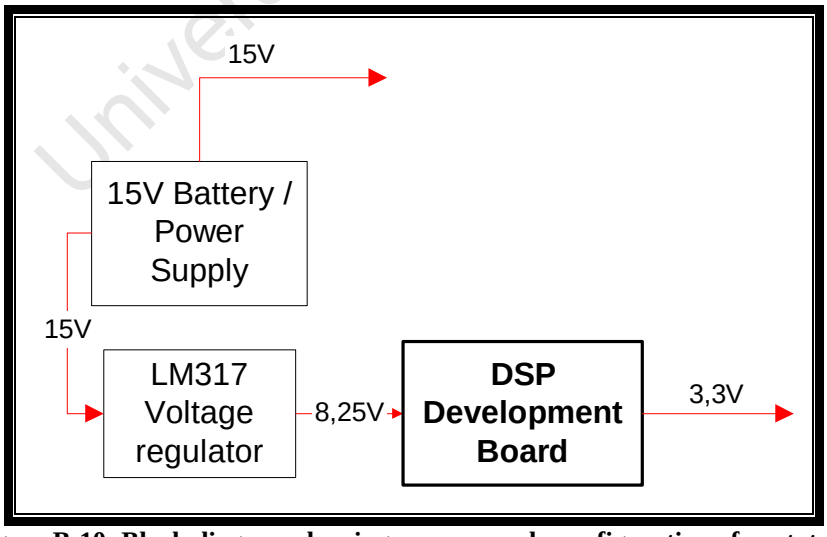


Figure B-10: Block diagram showing power supply configuration of prototype

The Freescale development board has an onboard power supply regulating 3,3V and 5V from a 9V input. This 9V is usually supplied by an ac/dc adapter plugged into the mains. To run the development board and all peripheral electronics off a single power supply, the 15V from the batteries was regulated to provide 9V. This

9V supply was plugged into the development board which had pins from which the 3,3V could be drawn. The 9V regulation was done using an LM317T. The LM317T is an adjustable voltage regulator, with the voltage being set using two resistors [5]. The simplest resistor selection was one that provided 8,3V rather than 9V. The development board was found to still be able to operate and supply 3,3V with an 8,3V input voltage.

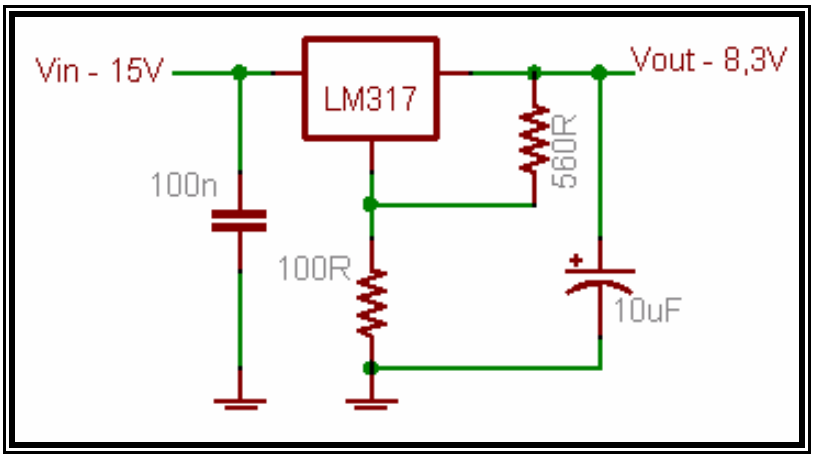


Figure B-10: Circuit Diagram of LM317

B.3.2 Printed Circuit Board

The power source for the robot is two 7.4V lithium ion batteries. When connected in series these give a supply voltage of 14.8V. This voltage can be used directly by the H-bridges to drive the motors. The other electronics however need 3,3V. On the prototype, this voltage was supplied by the on board power supply of the Freescale development board. The robot's PCB needed to include circuitry to do this voltage conversion itself.

The previous robot soccer robots used voltage regulators to supply 5v. This is the simplest and least expensive solution, but the large voltage drop results in a lot of wasted power, draining the batteries and reducing the time the robot can operate between charges. A more elegant solution is to use a switch mode power supply which can convert the voltage more efficiently than a simple voltage regulator.

Ideally, one would use a DC to DC converter such as TMH series of converters, which have 76% to 83% efficiency and require no external components [6]. Unfortunately these only work for input voltages of 5V, 12V and 24V, so they cannot be used with the 14.8V input voltage. An alternative option was found in the max5035 step down DC-Dc converter. The max 5053A, the 3,3V version of the IC, needs external components to operate but can supply 3,3V from an input voltage between 7.5V and 76V. Its efficiency is 86% [7].

The disadvantage of switched mode power supplies is that they are notoriously noisy. Not having used this power supply IC before, it was decided to structure the board with two power supplies, one, the max5035A switch mode power supply,

and the other a standard voltage regulator power supply. The user could choose which power supply to use using jumpers. With this design, should the switch mode power supply prove too noisy to provide power to the DSP, the voltage regulator could be used. When the board was completed and tested the switch mode power supply provided a clean enough supply that the voltage regulator did not need to be used.

The preferred choice of voltage regulator would have been a 3,3V voltage regulator such as the National Semiconductor LM3940 [8]. Unfortunately, no 3,3V regulator could be found which could operate off the high 15V input. For this reason the LM317 adjustable regulator was used. The circuit was the same as when it was used to give 8,3V for the prototype, but the resistor values were changed so that it would supply 3,3V.

Figure B-11 shows the schematic of the PCB's power supply. Due to the noisy nature of the switch mode power supply, it had to be electrically and physically separate from the other circuits, how this was done is explained in section G.8.

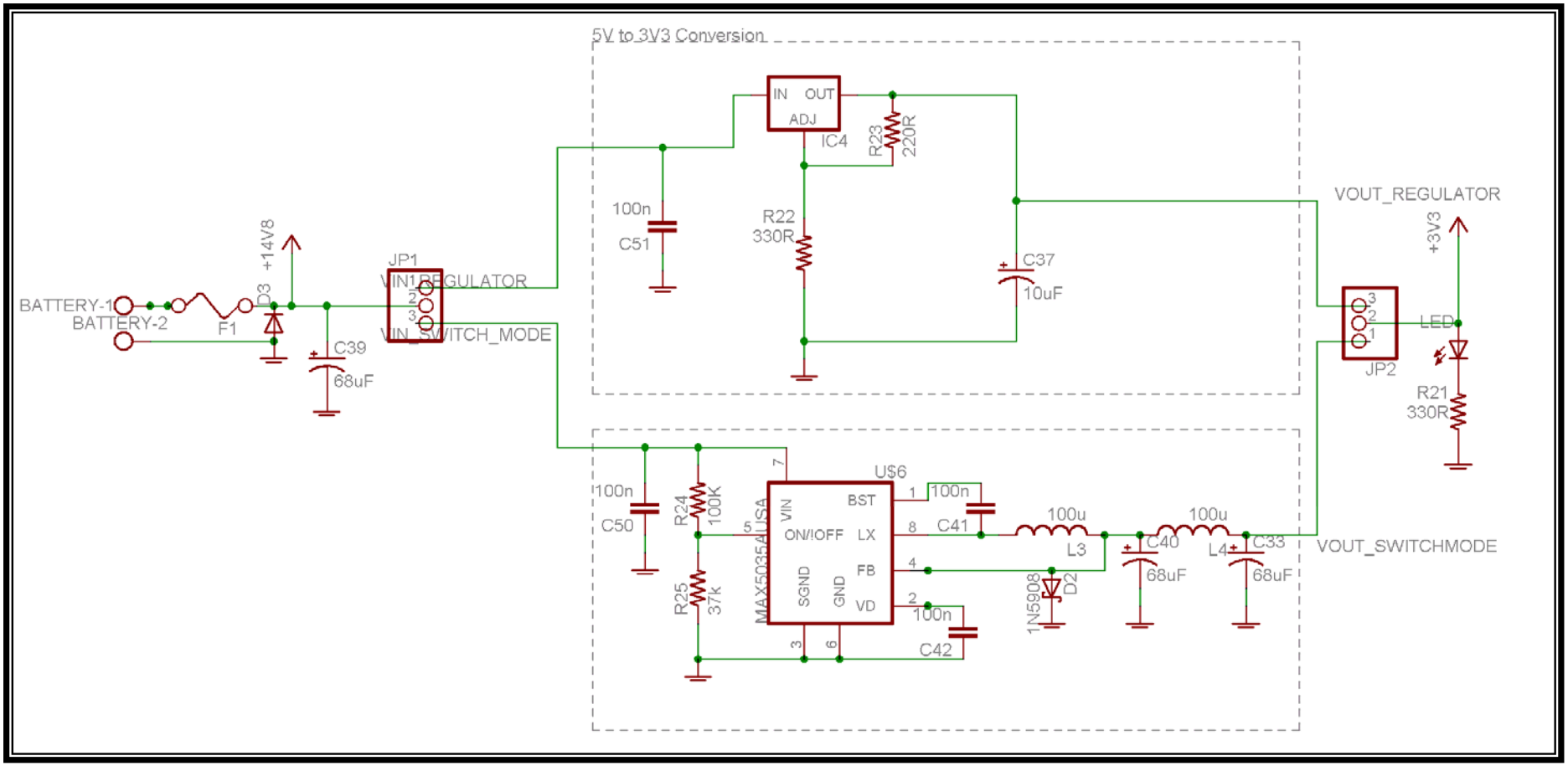


Figure B-11 Power Supply. The battery terminals are on the left. Jumpers allow the user to select between using the potentially noisy, but more efficient switch mode power supply (bottom) and the cleaner, less efficient voltage regulator (top).

B.4 Motor Drivers

The 3 wheels of the robot platform are driven by 16V DC motors. To enable the motors to drive in either direction they are driven using LMD18200T H0-bridge ICs. The speed of the motors was controlled by using a pulse width modulated signal to drive the h-bridges.

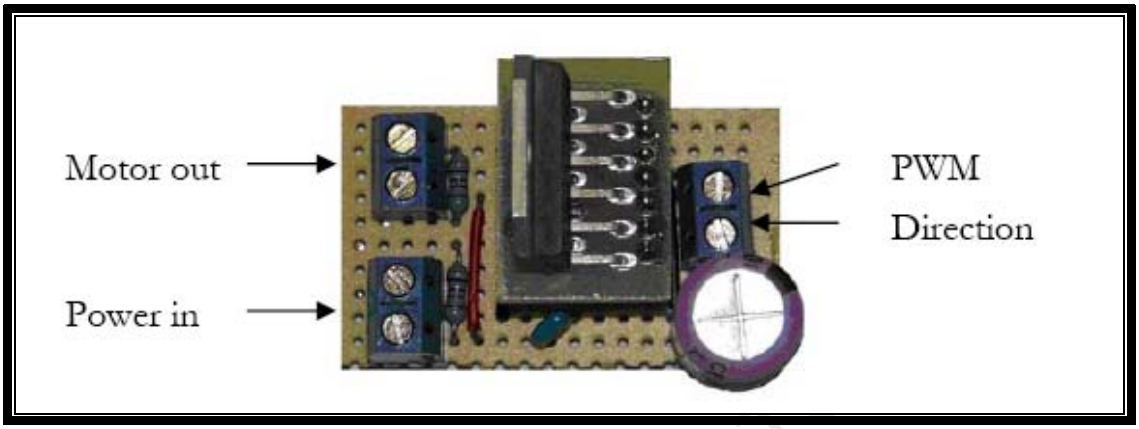


Figure B-12: H-bridge chips and enabling circuitry used for Prototype [9].

Figure B-14 below shows the connections and external components of the h-bridge chips. The configuration was based on that suggested in the IC's datasheet. There are two common ways to connect the LMD18200T to drive a motor, sign/magnitude PWM and locked antiphase PWM [10]. The IC has two control lines, the direction line, the PWM line and the brake line. The brake line allows one to quickly stop a motor by effectively short circuiting its two inputs, but was not used for our purposes (the weight of the robot means the robot brakes quickly as soon as the motors are no longer driven). Table B-3 below shows the logic table for the input lines, 1 and 2 refer to outputs 1 and 2 across the motor. The *dir* line controls the direction of current between the two motor outputs and therefore the direction of the motor.

Table B-3: Truth Table for LMD18200T [10]

PWM	Dir	Brake	Active Output Drivers
H	H	L	Source 1, Sink 2
H	L	L	Sink 1, Source 2
L	X	L	Source 1, Source 2
H	H	H	Source 1, Source 2
H	L	H	Sink 1, Sink 2
L	X	H	NONE

In sign/magnitude PWM, *dir* is used to determine the direction of the motor and the duty cycle of the pulse width modulated signal applied to *PWM* determines the speed of the motor. In locked antiphase *PWM* the pwm line is used as an enable line, and a pulse width modulated signal is applied to the *dir* line. In this arrangement a signal

with 50% duty cycle stops the motor, less than 50% drives it backwards and greater than 50% drives it forward. Figure B-13 is a diagram illustrating antiphase PWM.

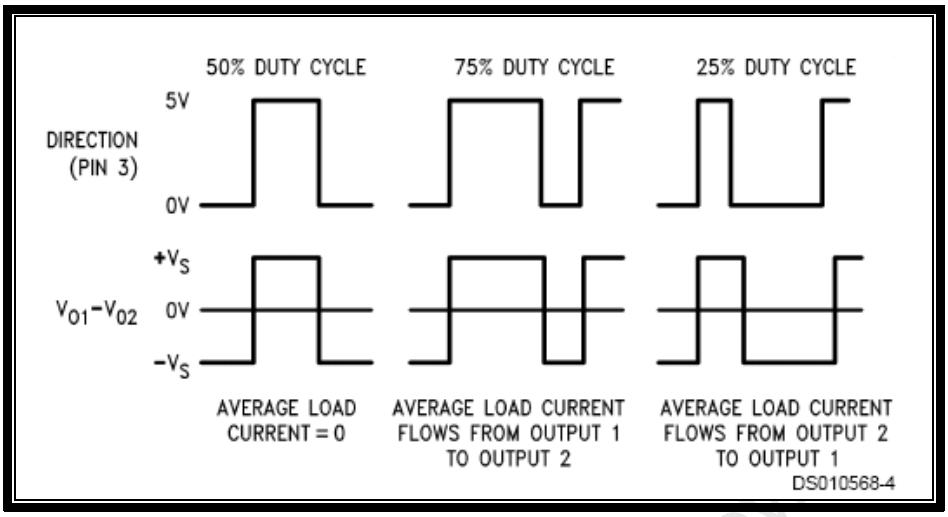


Figure B-13: Logic Level for LMD18200T

Antiphase PWM was chosen for this project as it simplifies the driving of the motors. It means that just one line can be used to control both the speed and the direction of the motor which simplifies the code needed to control the motor.

B.4.1 Prototype

The development board system used H-bridge boards that had been built by Craig Bamber to control the platform before. They were connected as described above.

B.4.2 Printed Circuit Board

The PCB was made to be able to control 4 motors so that the fourth motor could be used for a kicker or dribbler or the board could be used to drive an omnidirectional robot using the 4 wheel configuration.

As described above, National Semiconductor LMD18200T H-bridges are used to drive the robot motors. The circuit for the PCB remained largely the same as the one used for the prototype. Wherever appropriate, through hole components were replaced with surface mount components. Since the PCB was to be placed in the centre of the robot, with wheels spaced evenly around it, the ideal design would be to have each H-bridge and its connections on a different side of the PCB, however, the more space efficient and easier to route placement would be having the four h-bridges next to each other. As a compromise, the H-bridges were grouped into two groups of two, placed on the left and right sides of the board. The largest component in the circuit is the 1200uF decoupling capacitor. To save space, each group of two h-bridges shares a 2200uF capacitor, which is the same size as a single 1200uF capacitor.

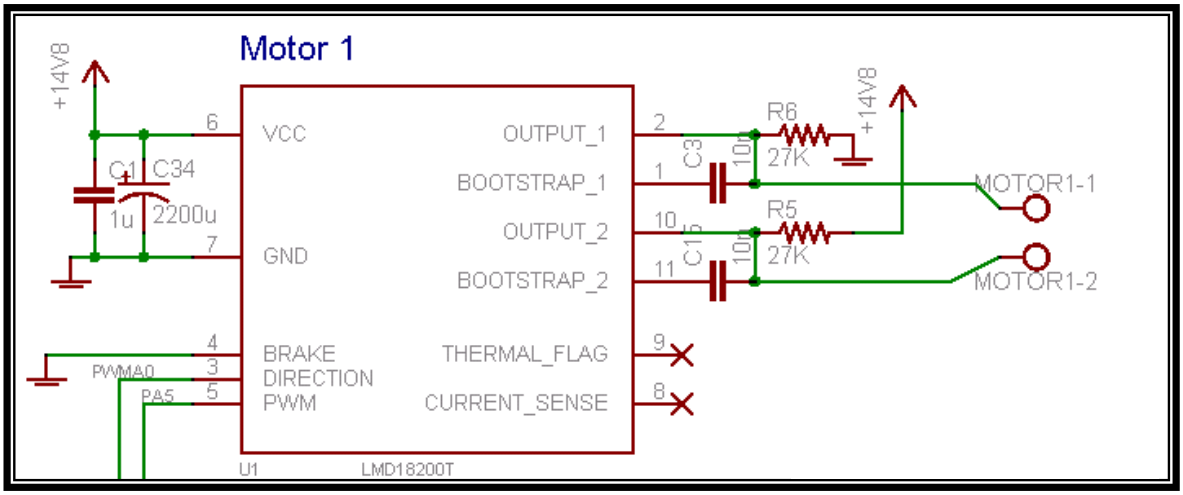


Figure B-14 H-bridge circuit.

The DC motors add a lot of noise to the h-bridge circuits and for this reason the h-bridge circuits needed to be electrically and physically separate from components such as the DSP. This separation is described in section H.8.

B.5 Wheel Encoder Circuitry

To determine the speed of the wheels, shaft encoders were manufactured. These are further described in Appendix H. Fairchild H21A1 optical interrupter switches were used to determine the passing of the teeth of the encoders. The interrupter switch consists of an infrared LED and a phototransistor, placed opposite one another. The teeth of the encoder alternatively block and allow the light from the LED to fall on the phototransistor. The prototype and PCB both use the circuit shown in Figure B-15. The resistor values were chosen to draw the least possible amount of current while providing a large voltage swing as the teeth of the encoder passed the switch. Molex connectors were placed on the PCB into which cables running to the optical switches could be plugged. The encoder circuit used on the prototype is shown in Figure B-16.

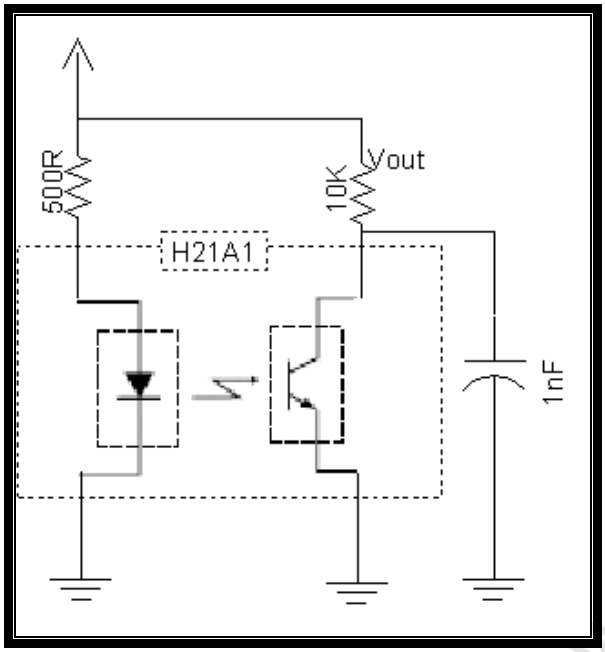


Figure B-15 Schematic of Encoder Circuit

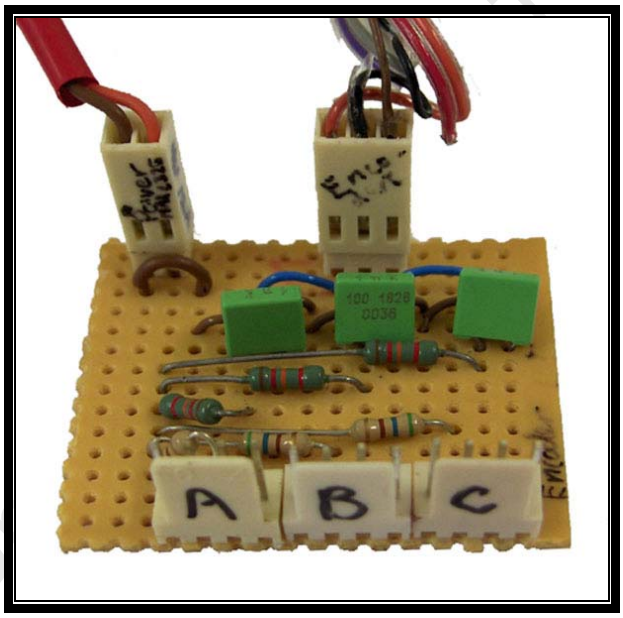


Figure B-16: Prototype encoder circuit

B.6 Serial Communications

To communicate between the DSP and a computer, so that the computer can send commands to the robot and information can be sent back from the DSP to the computer, a serial communications interface was used. Separate lines were used for transmitting and receiving from the DSP. The DSP operates on 3,3V while a PC uses ±9V on its serial interfaces. The max 3232 chip was used to convert between these two voltages. The max3232 DIP package was used for the prototype on veraboard and the surface mount version of that IC was used for the PCB.

The development board system uses a custom made cable to connect the DSP to the PC, using a Molex connector to connect on the DSP side. The PCB instead uses

a standard serial cable connector so that an off the shelf cable can be used to connect to the robot. Figure B-17 show the schematic of the connections to the MAX323 on the PCB

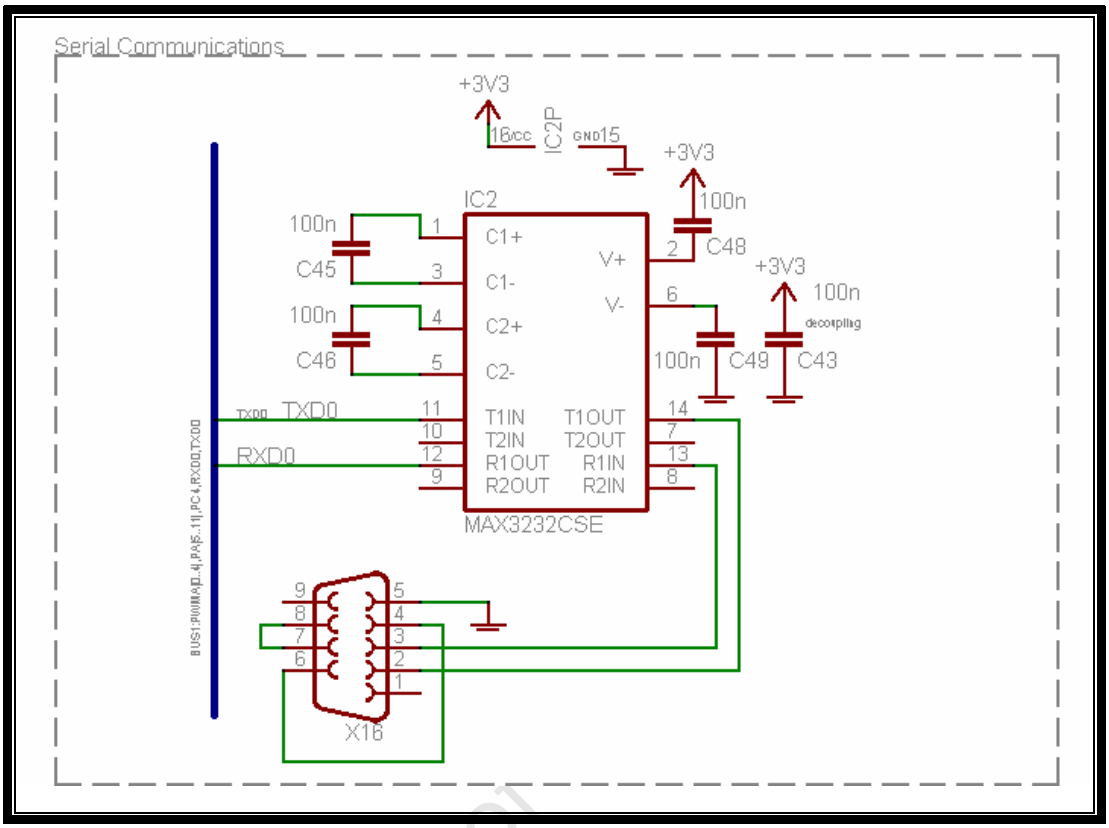


Figure B-17 : Schematic of Max323 connecting DSP to serial port for communicating with PC

B.7 Switches

The development board includes 3 push button switches; one RESET button, one IRQ interrupt button and one button connected to an I/O pin. These switches proved useful for debugging in the development stages so it was decided to include similar switches on the final PCB. The PCB therefore has a RESET and IRQ button as well as 2 buttons connected to pins 6 and 7 of port B.

B.8 Printed Circuit Board Layout

B.8.1 Introduction

Once the generic development board and electronics mounted on veraboard had been successfully integrated with the robot platform, a printed circuit board was designed so that the DSP and the control electronics could be incorporated into one board. This section describes the design of the PCB.

The layout editor Eagle 4.16r1 was used to draw the schematics and design the PCB layout. Since the Light Edition of Eagle was being used, the maximum board

size that could be made was 100mm x 80 mm. This size was large enough to accommodate the circuits needed to run the robot but left no space for diagnostic tools such as indicator LEDs or extra connections to the DSP's pins.

B.8.2 Noise Protection and Board Layout

The controller board includes two electrically noisy circuits – the switch mode power supply and the h-bridge motor drivers. The other circuits on the board, especially the DSP and its peripherals, need to be protected from this noise. To protect these circuits from electrical noise, the different circuits are separated both physically and electrically.

The noisy H-bridge and switch mode power supply circuits are placed at the top of the board, away from the more sensitive circuits. Each circuit has a separate supply and separate ground plains to further isolate the circuits from one another. The board is laid out in a star configuration - the ground and supply tracks for each circuit spread out from a single point, preventing large current fluctuations in one circuit from radiating noise into other circuits.

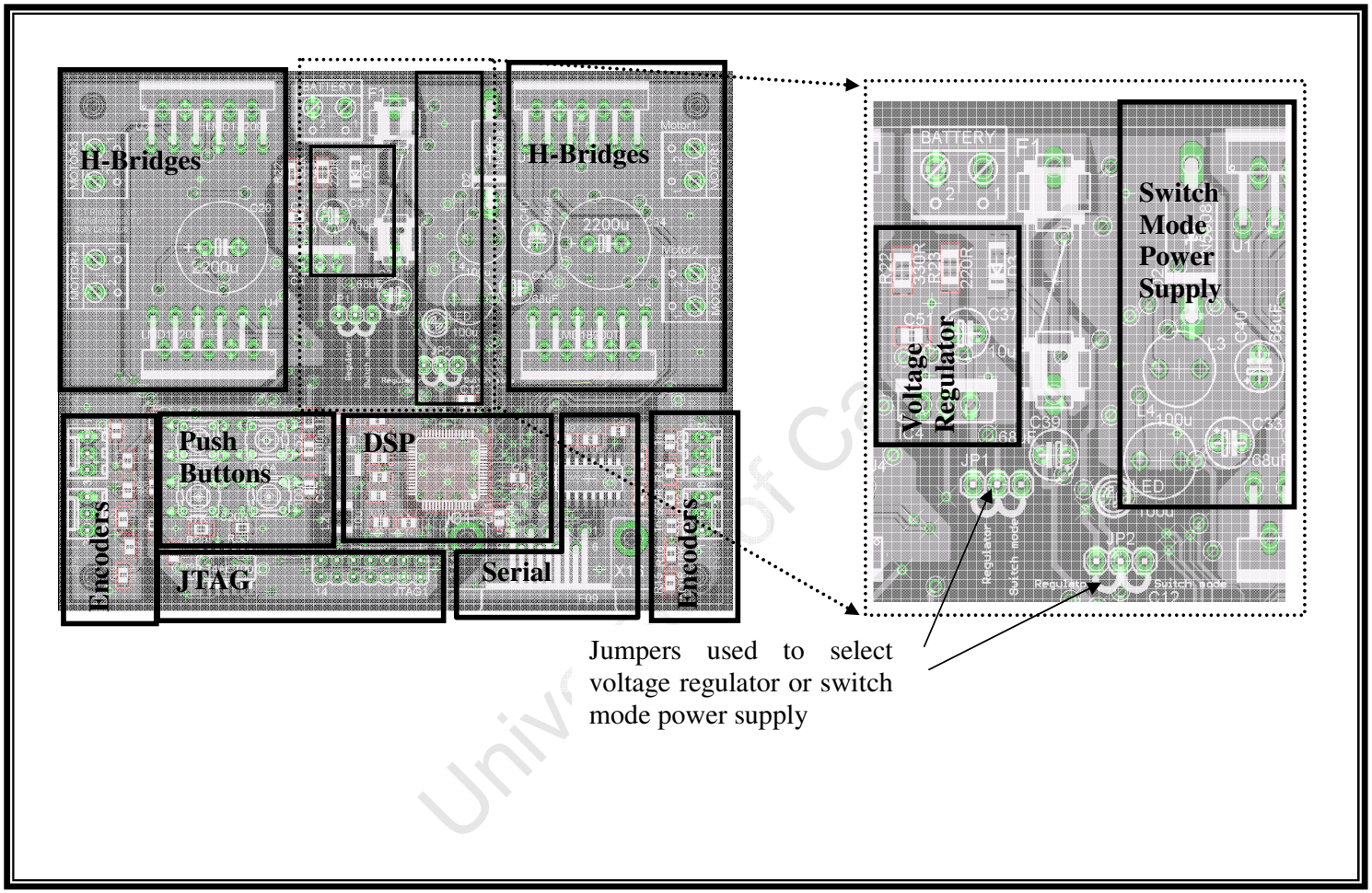


Figure B-18 Layout of PCB.

B.9 Initial Testing and Modifications

When the final board was printed and populated, a number of errors were spotted. These errors are listed below as well as the modifications needed to be made to the board to correct the errors.

B.9.1 Missing track

The track connecting the LED to resistor R21 was never laid, for this reason the negative leg for the LED needs to be soldered directly to the one track of R21, as shown in the photograph below.

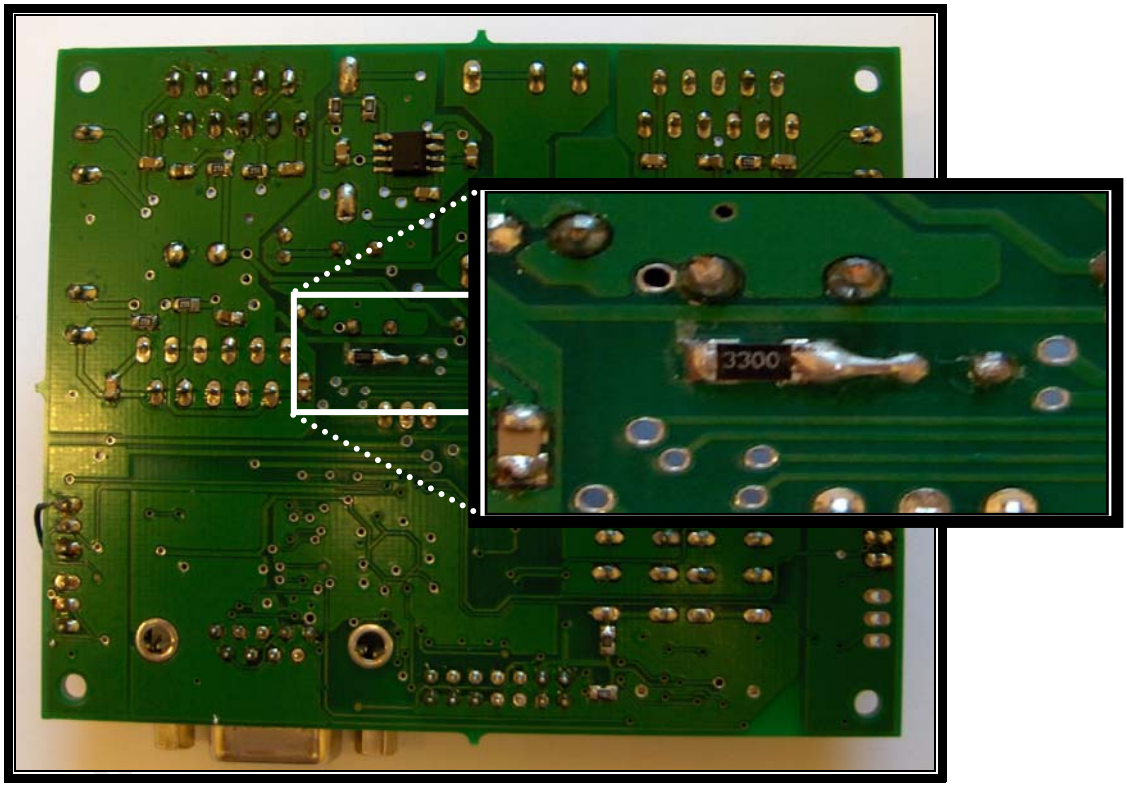


Figure B-19: Modification connecting LED to R21

B.9.2 Via connects GNDA to +3V3z

A via underneath switch S1, intended to provide an additional connection between the top and bottom ground planes, was placed too close to the +3v3 track, connecting the GNDA and +3v3 signals. To correct this, a 1,1mm drill bit was used to bore the via from the top, disconnecting the via from the +3v3 track.

B.9.3 Short Circuit on encoder 0 connection

The track connecting pin 3 of encoder 0 to the 510R resistor, R13, crosses the pad of pin 2, short circuiting these two signals. To correct this, the track was cut either side of pin 2. Wire was then used to connect pin3 to R13.

B.9.4 Short Circuit on Motor 1 Circuit

On initial testing of the populated PCB, the H-bridge circuits for motors 2, 3 and 4 worked as expected, but the circuit for motor 1 did not work. On inspection it was found that the track connected to the PWM line had a via through it, connecting it to the ground plane on the other side of the board. Since in the configuration used the H-bridge PWM line serves as a motor enable line, the motor was permanently disabled. To correct this, the via was drilled on the bottom of the board so that it no longer connected to the ground plane on that side. The via also connected the track to the ground plane on the top of the board so this part of the ground plane was cut so that it was no longer grounded.

B.9.5 Short Circuit on Reset* Line

A via from the ground plane goes through the Reset* line near the SCI connector. To correct this, the via was drilled out.

B.10 Further Testing

Once the above modifications had been implemented the PCB was found to behave as expected. The DSP was able to be programmed with a program which allowed the wheels to be controlled based on feedback from the encoders. The serial communications also worked. The PCB was then attached to the robot base as shown in Figure B-20 below. The PCB based robot behaved effectively and was used to conduct step tests. However, after further testing the board began to show errors which later made it unusable for further testing. These errors are documented below.

Figure B-20: PCB attached to robot base

During testing it was noticed that motor 2 was behaving erratically from time to time – it was driving at full speed when it should have been stopped, suggesting that the pwm line driving it was remaining high when it should have been outputting a pwm signal. On concluding that the pwm channel A1 which drives motor two had been damaged, it was decided to run motor 2 from pwm channel A4 of the DSP. To do this, the leg of the h-bridge chip connected to the pwm channel was cut so that a thin

ribbon cable wire could be soldered from it to the pin pwmA4. Motor 2 continued to behave erratically suggesting that the h-bridge may have been damaged in some way. While this was being investigated motor 1 also began to behave erratically.

It was concluded that the h-bridge chips were in some way damaging the pwm lines of the DSP chip. It was therefore decided to buffer the pwm lines from the h-bridges with 1k resistors in series with the pwm lines. Since a number of pwm lines had now been damaged, the DSP chip on the board was also replaced.

The new DSP chip initially behaved as expected and was able to be programmed. On the fourth time the chip was to be programmed, the Metrowerks Codewarrior program could not detect the board to program it. Several attempts to program the board were unsuccessful. The programming cable was found to still be operational as it was able to be used to program a demonstration board. In an attempt to find the reason why the board could no longer be programmed, an oscilloscope was used to trace all the signals of the DSP which are used for programming. These were compared to the same signals on the development board and no inconsistencies could be found between the two. Particular attention was paid to the RESET* and TRESET* lines as these indicate to the DSP and PC that the DSP has been plugged in for programming.

When no error could be found it was assumed that the DSP chip was broken. Rather than risk damage to the new DSP chip by replacing the one on the current board, it was decided to populate a new PCB. The resistor buffers between the h-bridges and the DSP were included on this new board. The new board programmed successfully at first. Motor 2 again began to behave erratically so it was decided to not use that motor output and rather connect the second motor to motor output 4. However motor 4 then began to behave unpredictably. When the pwm to the motor should have been 50% and the motor stopped, the motor was turning at full speed. The motor would stop and start at seemingly random intervals. Before the reason for this could be more fully investigated, the new DSP chip failed to program and again attempts at debugging this fault were unsuccessful.

Time constraints and the unavailability of a third DSP chip prevented another board being make to be able to further investigate the motor fault. It is suspected that the noise from the H-bridges damaged the sensitive DSP chip and that the electrical isolation of the H-bridges from the other components was insufficient. Future redesigns of this board should have the H-bridges on a separate board with the pwm and enable signals electrically isolated by an optical isolation chip such as the Fairchild 6N135/6 [11].

B.11 Concluding Remarks

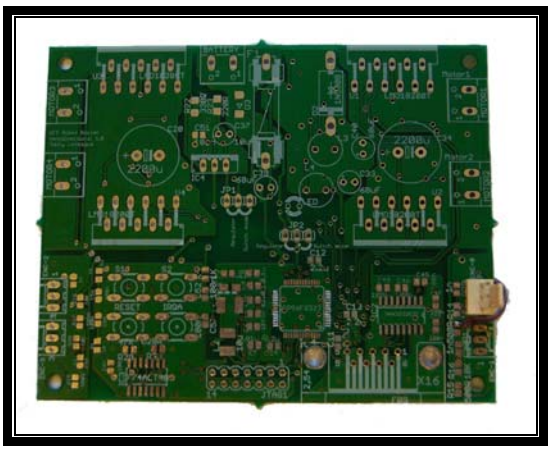


Figure B-21: Unpopulated PCB

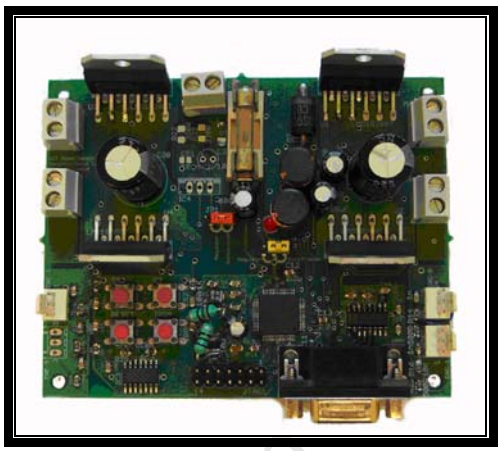


Figure B-22: Populated PCB

Figure B-21 and Figure B-22 show the PCB before and after population. The printed circuit board designed to control the omnidirectional robot was successful in combining the following elements on a single PCB:

- A Freescale DSP56F8323 DSP
- Power electronics consisting of a switch mode power supply and a voltage regulator either one of which can be selected to power the board
- 4 motor drivers
- 4 encoder circuits
- Serial communications
- Programming circuitry

Before the PCB can be used to replace the development board and peripheral electronics, the fault on the board will have to be further investigated and corrected for.

B.12 Recommendations for Future Boards

The following section documents changes made to the PCB board based on modifications documented in section B.9. Additionally it is suggested that changes be made to the power supply and programming circuits.

B.12.1 Power supply

The PCB was designed with both a switch mode power supply and a voltage regulator. The voltage regulator was included to be used if the switch mode power supply proved too noisy. The switch mode power supply was found to provide a

clean enough signal for the operation of the board so the voltage regulator circuit is unnecessary and can be removed from future versions of the board.

B.12.2 Programming Interface

Since the board was not successfully programmed using the JTAG interface and JTAG module, the parallel port interface described in section B.2.3 was used. Future versions of this board should either include this additional circuitry and a parallel port plug on the board or to save space, a small printed circuit board should be designed to plug onto the board, consisting of the circuit in Figure B-8 on page B-2.

B.13 Improvements – Robot Soccer Omnidirectional version 1.1

Based on the testing done on the PCB and the errors found on the board (described in Section B.9), the schematic and board design of the PCB were updated. The board with these changes made is version 1.1 of the omnidirectional board. Due to time implications, and the fact that the changes made were not enough to warrant reprinting the board, this board has not been printed or tested. It is suggested that all future versions of the omnidirectional robot be based on this board.

Below, each of the changes made to the board are described. Some of the changes are based on the errors pointed out in section H.9 while others make the board more readable when it is being populated.

B.13.1 Laying of missing track from LED to R21

The missing track between LED and R21 was laid.

B.13.2 Correction of Short Circuit on encoder 0 connection

The track between R13 and pin 3 of encoder 0 was re routed so it no longer crossed track

B.13.3 Via moved to prevent short circuit

The via mentioned in H.9.2, which inadvertently connects the GNDA and +3v3 signals, was moved so that it no longer crosses pin 2.

B.13.4 Via on motor 1 circuit moved

The ground plane via which was holding the PWM line of the motor 1 circuit low, was moved so that it no longer crossed the PWM track.

B.13.5 Via moved to prevent short circuit of *RESET

The ground via going through the *RESET line was moved so that the were no longer connected.

B.13.6 Labelling positive battery terminal

A ‘+’ sign was added to the silkscreen layer (Tplace) to mark the positive terminal of the battery.

B.13.7 Labelling Negative Terminal of LED

In the Eagle board design the symbol for the red indicator Led had a flat edge to indicate which way the LED should be placed. However on the PCB, the symbol no longer indicates the placement. For this reason the next version of the board includes a small ‘-‘symbol on the silkscreen layer to indicate the cathode.

B.13.8 Moving of Labels on Silkscreen Layer

A number of labels which were obscured by the copper layer have been moved and others have been made bigger to make them more readable

References

1. Freescale, *68HC908GT16: Microcontroller - Overview*. 2007.
2. Freescale, *Preliminary Technical Data 56F8323 16-bit Hybrid Controller*. 2003: www.freescale.com.
3. Freescale, *68HC908GT16: Microcontroller - Overview*, *http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=68HC908GT16&webpageId=1100709370386709808055&nodeId=016246844976638055&fromPage=tax*. 2007.
4. Freescale, *Targeting 56F8300 Demonstration Board - User Manual*. MC56F8300TUM Rev. 4. 08/2005.
5. STMicroelectronics, *Data Sheet - LM317*.
6. TracoPower, *Datasheet - TMH series.*
7. Maxim, *Datasheet - Max5035.*
8. NationalSemiconductor, *Datasheet - LM3940.*
9. Craig Inman-Bamber, *Robot Soccer Platform*, in *Mechanical Engineering Department*. 2005, University of Cape Town.
10. NationalSemiconductor, *Datasheet - LMD18200T.*
11. FairchildSemiconductor, *6N135 Datasheet.*

Appendix C Odometry

Table of Contents

Table of Figures

Appendix C. Odometry

C.1 Introduction

To be able to control the robot accurately, the speed at which it was traveling needed to be measured in some way. This chapter discusses the use of an optical encoder to measure the speeds of the three wheels. Also discussed is the choice of algorithm used to determine the speed from the encoder signal.

C.2 Choice of Speed Measurement Transducer

When choosing a method of speed measurement for digital control one has to consider its accuracy and resolution and the frequency with which the speed measurements can be updated. The three speed measurement systems considered for the robot platform were optical encoders, light speed measurement (such as in optical computer mouses) and rolling transducers (similar to those in traditional mouses).

C.2.1 Optical Shaft Encoders

Optical encoders are a simple method of measuring wheel speed. A disc is placed directly onto the shaft of the motor, or onto a shaft driven by the motor. This disc is divided into segments, either by slots cut into it, or by some segments being black and others white. Figure C-1 shows one of these discs. An optical sensor consisting of a transmitting LED and a receiving phototransistor detects the segments as they

pass. The speed at which the segments pass is used to determine the speed of the motors.

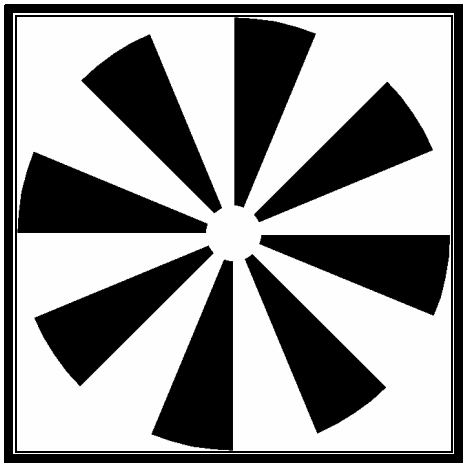


Figure C-1: Traditional shaft encoder disc [1]

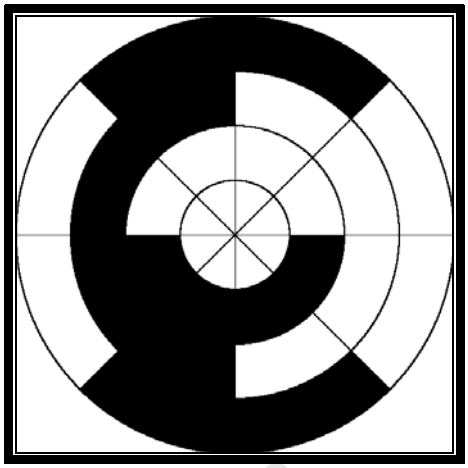


Figure C-2: Gray code encoder disc [2]

This method is incremental in that one can count the number of segments to have passed from a point but cannot determine the absolute position of the wheel. Should it be necessary to determine the absolute position of a motor, an encoder using gray code can be used [3]. Figure C-2 shows a gray code disc. Using one sensor means one has no way of determining the direction in which the wheel is moving, one must infer the direction of the wheel from the voltages used to drive the motor. This can be overcome by using two sensors on the wheel, positioned so that they are out of phase, depending on the direction in which the wheel is moving, one or the other sensor will see the segment transition first. [3]

Shaft Encoders are relatively simple and inexpensive to make. They have the disadvantage of having limited resolution since there is always a limit to how small the segments can be made.

Another disadvantage of shaft encoders is that they measure the speed of the wheels not the robot. The speed and position of the robot has to be inferred rather than directly measured. This is referred to as 'dead reckoning'[3]. In dead reckoning the accuracy of each move is affected by the accuracies of all previous moves. The accuracy of dead reckoning is limited since errors due to slippage, sensor errors or differences between the kinematic model and the actual system, add up over time. This is less of a concern for a robot soccer platform since localization is performed by the overhead cameras, the AI system can correct small errors in position and orientation.

C.2.2 Rollers

To measure the position and orientation of the robot directly one could use a system similar to that used for a traditional computer mouse. A ball or pair of rollers would be mounted underneath the robot; shaft encoders would measure the movements of the rollers and infer from these the movements of the robot. There would need to be two of these underneath the robot to be able to measure changes

in the robot's orientation. This type of sensor could be created by adapting a computer mouse. One is then limited by the resolution of the sensors in the mouse.

C.2.3 Optical mouse

Recently traditional computer mouses have been largely replaced by optical mouses. These use an LED and photo sensors to measure movements of the mouse. Adapting two of these and mounting them on the robot could provide a means of localizing a robot. Sekimori and Miyazaki [4] were able to successfully use optical mouses for dead reckoning on an omnidirectinal robot. To reduce errors, they used four optical mouse sensors. Palacin et al [5] also tested the use of an optical mouse for dead reckoning on a robot with some success, however they reported having trouble getting the system to work on carpeted surfaces. Since robot described in this thesis needs to work on the felt surface of the robot soccer we would not want to use a system that does not work well on carpet. On reading Sekimori and Palacins's research it became apparent that although the use of optical mouses can prove very effective in robot odometry, a lot of work is required to get it to work effectively. Since the odometry is not the main focus of this research, it was decided that a simpler, more easily implemented method would be more suitable.

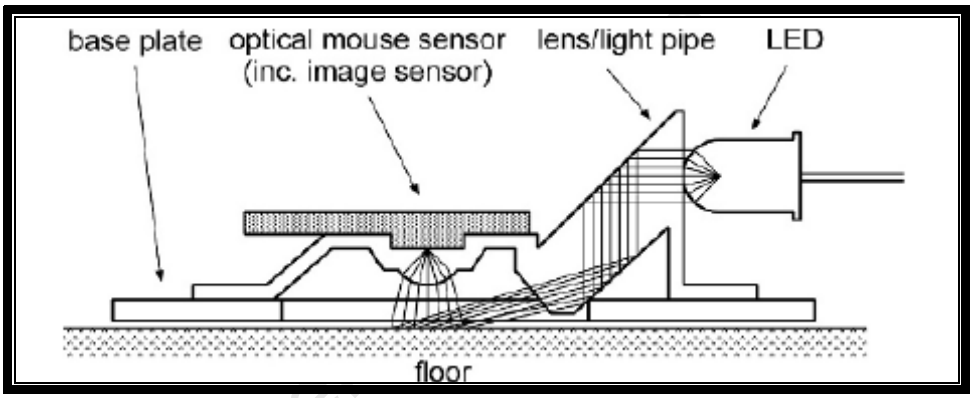


Figure C-3: Structure of an optical mouse sensor [4]

C.2.4 Conclusion

It was decided to use shaft encoders to measure the wheel speeds of the robot. This is the simplest and most cost effective option.

C.3 Optical Encoders

The prototype robot made by Craig Bamber used the wheels themselves as encoders. The rollers are black and the portions between wheels were painted white and a sensor was mounted above the wheel to sense the transitions [6]. This system was never successfully implemented and it was suggested that the resolution provided by the system would not be sufficient for accurate control. Hence this potentially elegant solution was abandoned in favour of a more traditional shaft encoder.

Graeme McPhillips designed a new wheel encoder mask for the robot. This encoder consisted of a PVC wheel cut on a CNC mill with 50 teeth cut into it,

designed to sit on the hub of an omnidirectional wheel. Figure C-4 below shows the encoder disc, Figure C-5 shows the disc mounted on the wheel hub.
A Fairchild Semiconductor H21A1 Phototransistor Optical Interrupter Switchi was used to detect the rotation of the teeth attached to the wheel. A discussion of the design of the electronics used to detect the wheel rotations and the interfacing of those electronics with the DSP can be found in Appendix G.

C.3.1 Physical Design

To mount the sensors the side parts of the sensor were cut off in order to make them fit. The way they were mounted sideway also meant that they could not be screwed in as would usually be done. So the sensors were glued on. This is not a robust method to put them on as they can fall off quite easily. Although good enough for a prototype, it is hoped that future designs for the sensors designs will be mounted more securely, possibly by using differently packaged sensors, or by placing the sensors where they could be screwed on.

Figure C-4: Wheel Encoder Disc

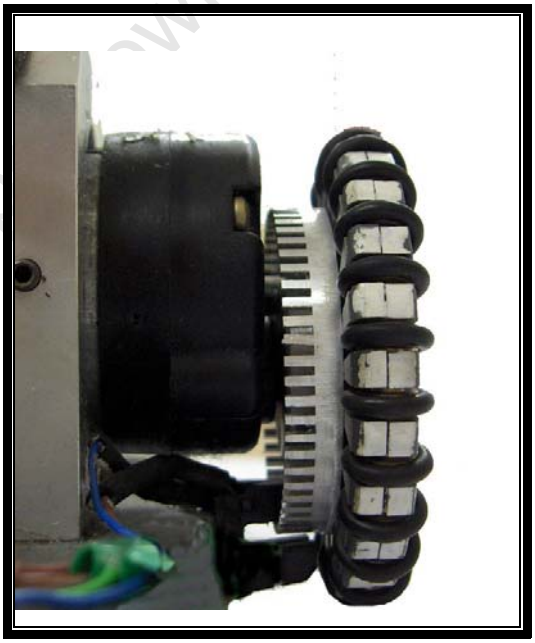


Figure C-5: Wheel Encoder Disc attached to robot wheel

C.4 Odometry Methods

Having attached an optical encoder to the robot wheel, an algorithm had to be chosen to use on the DSP to infer the speed of the wheels from the encoder signal. Different algorithms offer different accuracy, update intervals and ease of implementation. The following is a discussion of the different speed measurement techniques considered.

The advantages and disadvantages of the algorithms often only become apparent when one considers how the algorithm behaves over a range of speeds. For this reason oscilloscope traces of the encoder signal at different speeds are used to illustrate the effectiveness of each method. Figure C-6 is a trace of the encoder signal when a

wheel is run at a very low speed (the speed at which the robot just begins to move). Figure C-7 is the encoder signal when a wheel running at full speed. The oscilloscope was used to measure the frequency and maximum amplitude of the signals (show at the bottom of each figure). The robot was run on carpet similar to the felt of the robot playing field.

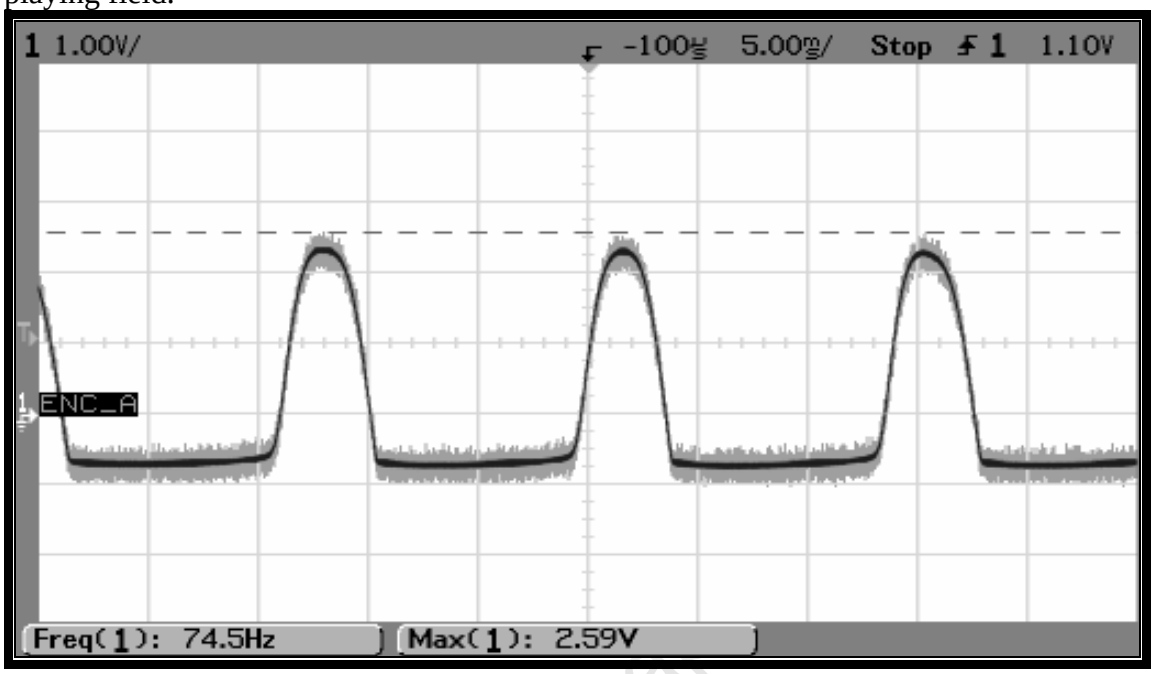


Figure C-6: Trace of encoder signal with wheel run at low speed. The Frequency of the encoder teeth when the robot just starts to move is about 75Hz.

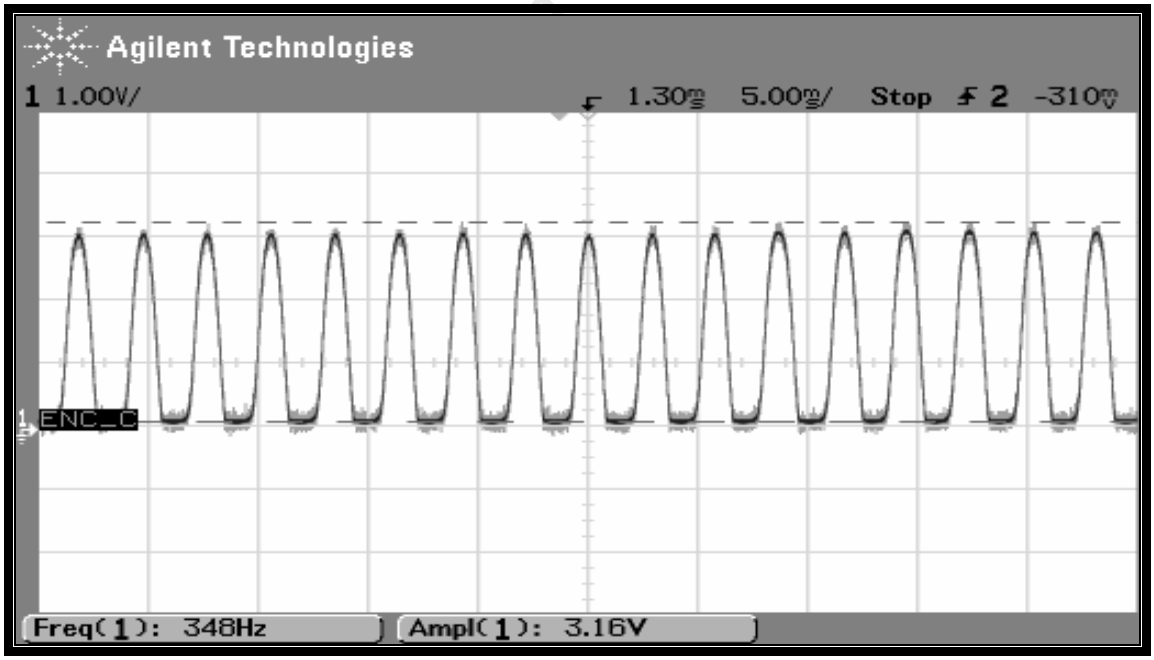


Figure C-7: Trace of encoder signal with wheel run at full speed. The frequency of the encoder teeth when the robot is running at top speed is about 350Hz.

C.5 Tooth Counting

Since speed is a measurement of distance covered in a certain amount of time, an obvious way to measure the speed of a wheel using an encoder is to count the number

of teeth which pass the tooth sensor in a fixed time interval. This is sometimes referred to as the frequency method[7]. The frequency method is simple to implement but has the disadvantage that the resolution is limited at fast update rates.

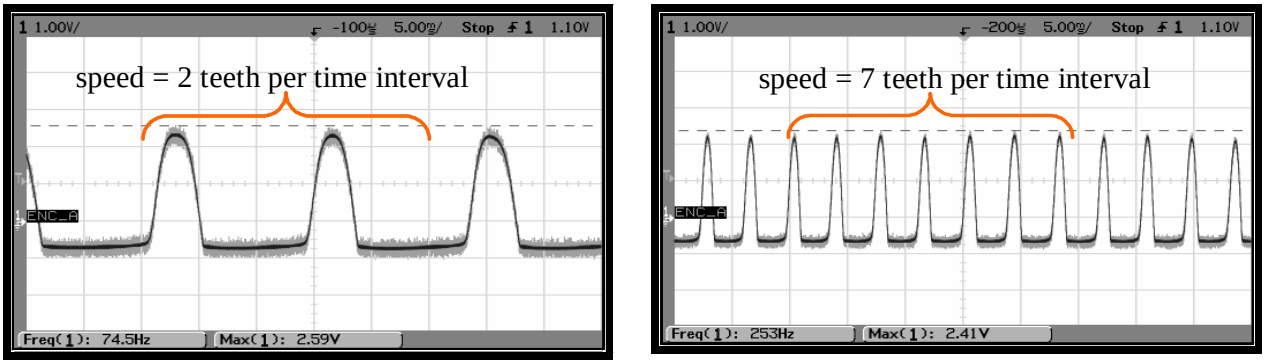


Figure C-8: Tooth Counting. The speed is determined by counting the number of encoder teeth to pass the sensor is a certain time period.

The effectiveness of a speed control algorithm is affected by the rate at which speed readings can be updated. To achieve a faster update rate when using the frequency method, the time interval over which teeth are counted must be decreased. The shorter the time interval, the less the number of teeth counted at the maximum speed. If, at maximum speed, 10 teeth pass the counter, then the odometer will be able to read only 10 different speed levels. A lower speed resolution would make the control algorithm inefficient. Thus, a compromise has to be reached between the rate at which the speeds are updated and the resolution of those speeds.

C.6 Measuring Time between Teeth

An alternative measurement method is to measure the time it takes for a single tooth to pass. This gives you the inverse of speed and is sometimes referred to as the period method [7]. This method works well at high speeds, but the update interval is proportional to the speed. At low speeds - since the speed only updates when a tooth passes – the wheel speed will update relatively slowly. The non constant update rate will cause problems with the control algorithm which relies on the speed being at specific time intervals.

Mr McPhillips, tried this method of speed measurement for his robot but abandoned this method because of inconsistencies. Inaccuracies in the manufacture of the encoder, and small misalignments in how the encoders were mounted on the wheels meant that the pulses were not exactly the same distance apart from each other. Mr McPhillips tried to use a digital filtering method to account for the inaccuracies, but was unsuccessful, possibly because of a lack of computing power on the DSP he was using [8].

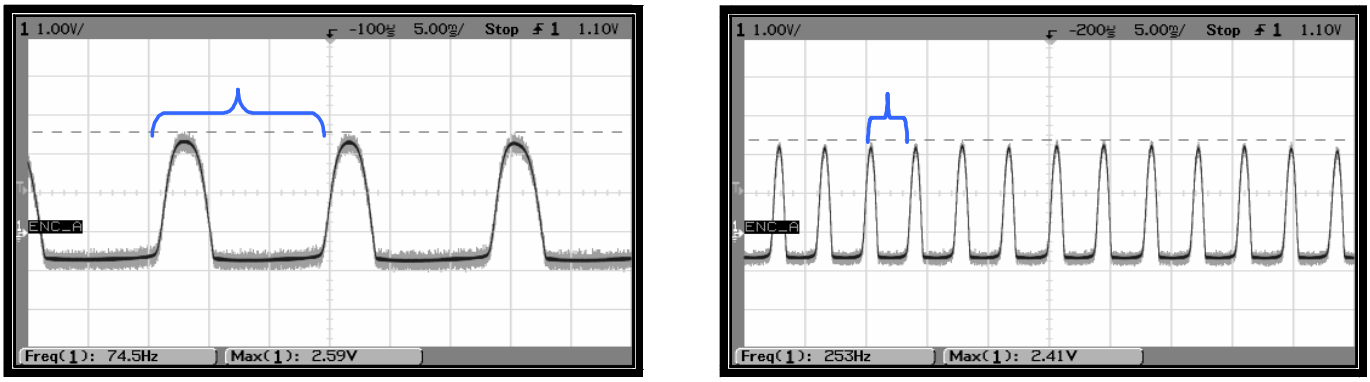


Figure C-9: Measuring time between teeth. The speed is measured by measuring the time it takes for a single tooth to pass.

C.7 Measure slope of encoder Signal

Mr McPhillips suggested another measurement method, based on measuring the rate of change of the encoder signal.

Figure C-10 below is an oscilloscope trace of the encoder signal. The voltage increases and decreases as the encoder teeth block the light to the phototransistor of the H21A1. As the speed increases, the frequency of this signal increases, as does the gradient of the slope of the rising and falling edges of the signal. By measuring the slope of the rising (or falling) edges, one could get an instantaneous speed reading which is updated every time a tooth passes, increasing the sample rate and improving its accuracy.

Like measuring the time between teeth, this method updates the speed measurement every time a tooth passes, meaning the update intervals are speed dependent. This method will also be subject to inaccuracies in the manufacture of the encoder. It was thought that inconsistencies in the encoders might have less of an effect on the rise time of the encoder signals than on the distance between high measurements and this method would therefore prove to be more accurate than measuring the time interval between consecutive teeth passing. What follows is a report on the implementation of this speed measurement system and an investigation into its effectiveness.

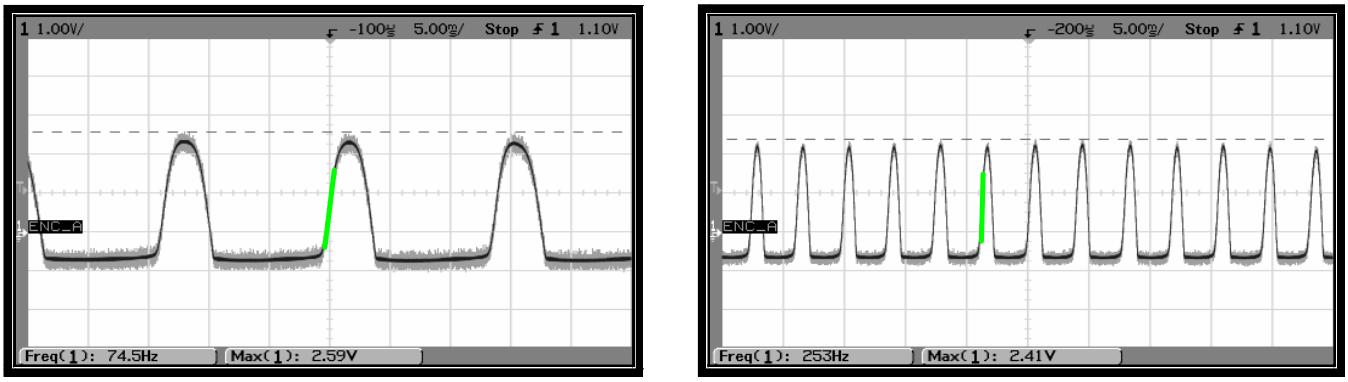


Figure C-10: Measuring speed by measuring the slope of the encoder signal

C.8 Investigation into Alternative Speed Measurement system

As mentioned above, traditional methods for measuring wheel speed using an encoder are the frequency and period methods. They both give an indication of how many fast encoder teeth pass the encoder beam but differ in whether it is the amount of teeth, or the time period that is constant. The frequency method determines the speed from the number of teeth to pass in a specific time. The period method measures the time interval between successive teeth. Rataj et al [7] point out that to increase the resolution of the frequency method one has to limit the update rate, while the period method is prone to noise. The method implemented here was an attempt to measure the wheel speed with a greater update rate than the frequency method but less noise than the period method.

Figure C-11 below, shows the signal recorded from an encoder attached to a wheel of the robot platform when run at low and high speed. The frequency of the signal increases as the wheel speed increases. So too does so does the gradient of the slope of the pulses as each tooth passes. This section describes how the ADC of the Freescale 56F8323 DSP was used to measure the slope of each pulse as it passed. The slope measurements were used as the basis for an alternative speed measurement system. Since the method looks at individual pulses, it is not prone to the limited update rate of the frequency method. It was hoped that this new method (referred to as the slope method) would have less inaccuracies between successive measurements than the period method. This chapter first describes how this slope measurement system was implemented on the DSP. The results of a speed tests done using this system are then reported before conclusions are drawn as to the effectiveness of the method. This method was attempted in the hope that it would give more accurate, quickly updated measurements than the traditional methods, it proved to be less accurate and require more signal conditioning methods and was therefore abandoned in favour of the more traditional tooth counting method.

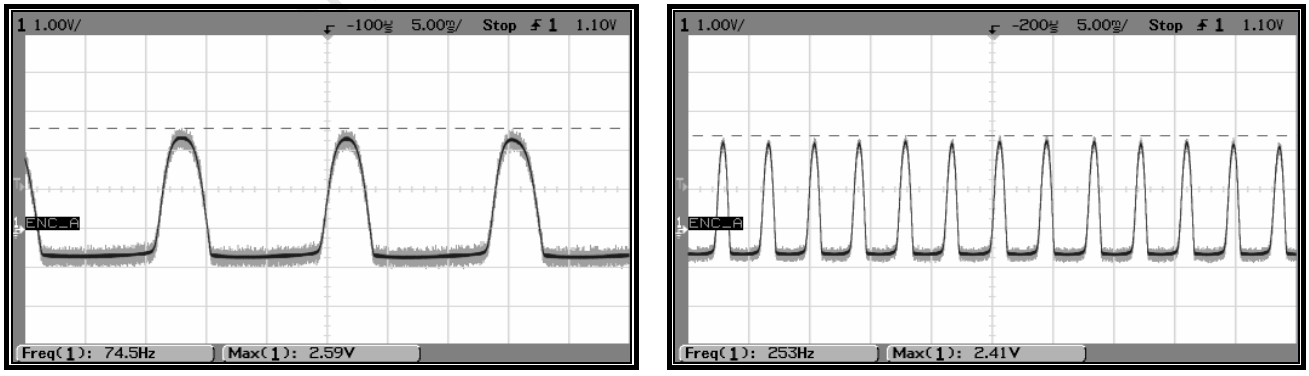


Figure C-11: The signal from an encoder when the wheel is run at low and high speed. Since the slope of each pulse changes as the signal frequency increases, by measuring the gradient of the slopes, one could get a measure of the wheel speed.

C.8.1 Implementation

To measure the gradient of the rising edge of each pulse as it passed, the change in the voltage level of the signal over a set amount of time was measured using the ADC. Since the ADC is already being used to measure the voltage of the signal, and the ADC takes a specific amount of time per conversion cycle, it was used for timing the interval rather than using the timer module. This frees up the timer modules so they can be used for other processes, such as control.

The encoder signals were fed to three ADC channels of the DSP. The ADC has high level and low level triggers on each channel. When enabled, the high level trigger calls an interrupt if the voltage to the ADC rises above a certain level, similarly, when the low level trigger is enabled, it will call an interrupt if the voltage drops below a certain level. The trigger levels are set by registers which can be written to at run time to change trigger levels or disable the triggers. The triggers for the different channels can be set up independently of each other. To measure the slope of the encode signal the high level trigger was used to indicate the beginning of a pulse, and the low level trigger to indicate when the pulse had ended.

Figure C-12 below illustrates the algorithm used to measure the slope of each pulse. Below that Figure C-13 shows the flow chart of the code used on the DSP. Once the high level trigger has determined the beginning of a pulse (when the voltage is higher than 390mV), the DSP waits for 10 ADC cycles which indicates that 500µs have passed. Then, the voltage level is recorded using the ADC. This voltage minus the threshold voltage of 390mV, divided by the time period of 500µs gives the gradient of the slope, which should be proportional to the speed of the wheel. When the voltage dips again as the tooth passes the low level interrupt triggers, resetting the state of the DSP so that it is waiting for the next pulse. A state variable is used to keep track of whether the code is waiting for the beginning of a pulse, or counting the ADC cycles after the start of the pulse.

The speeds are output directly from the DSP. Each time a tooth passes and a speed is updated, a flag variable is set. The main program of the DSP continually checks if this flag has been set and if it has, transmits the value of the three wheel speeds to the PC across the serial link. The code for these tests can be found in Appendix K.

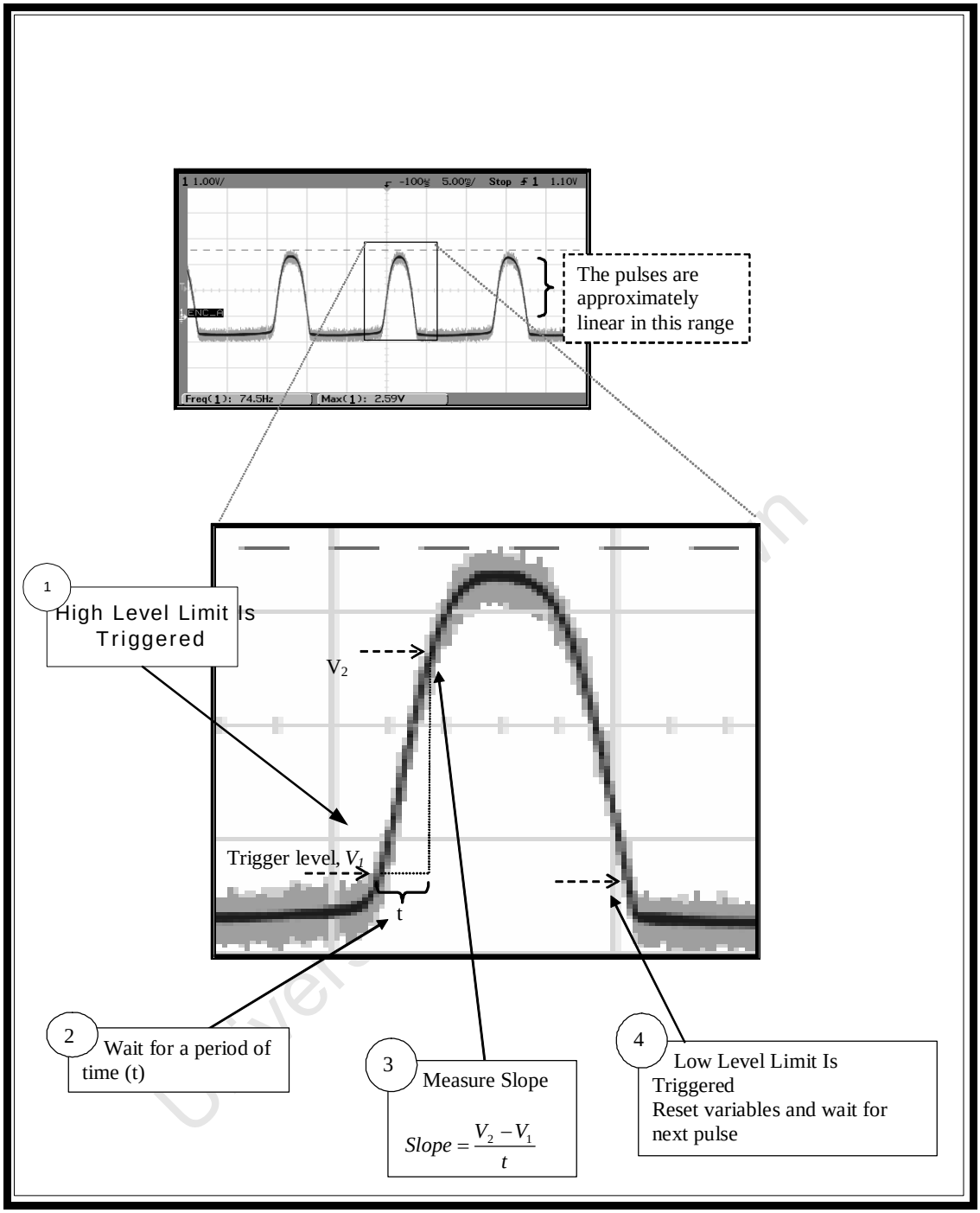


Figure C-12: Diagram illustrating how DSP determines slope of encoder pulses

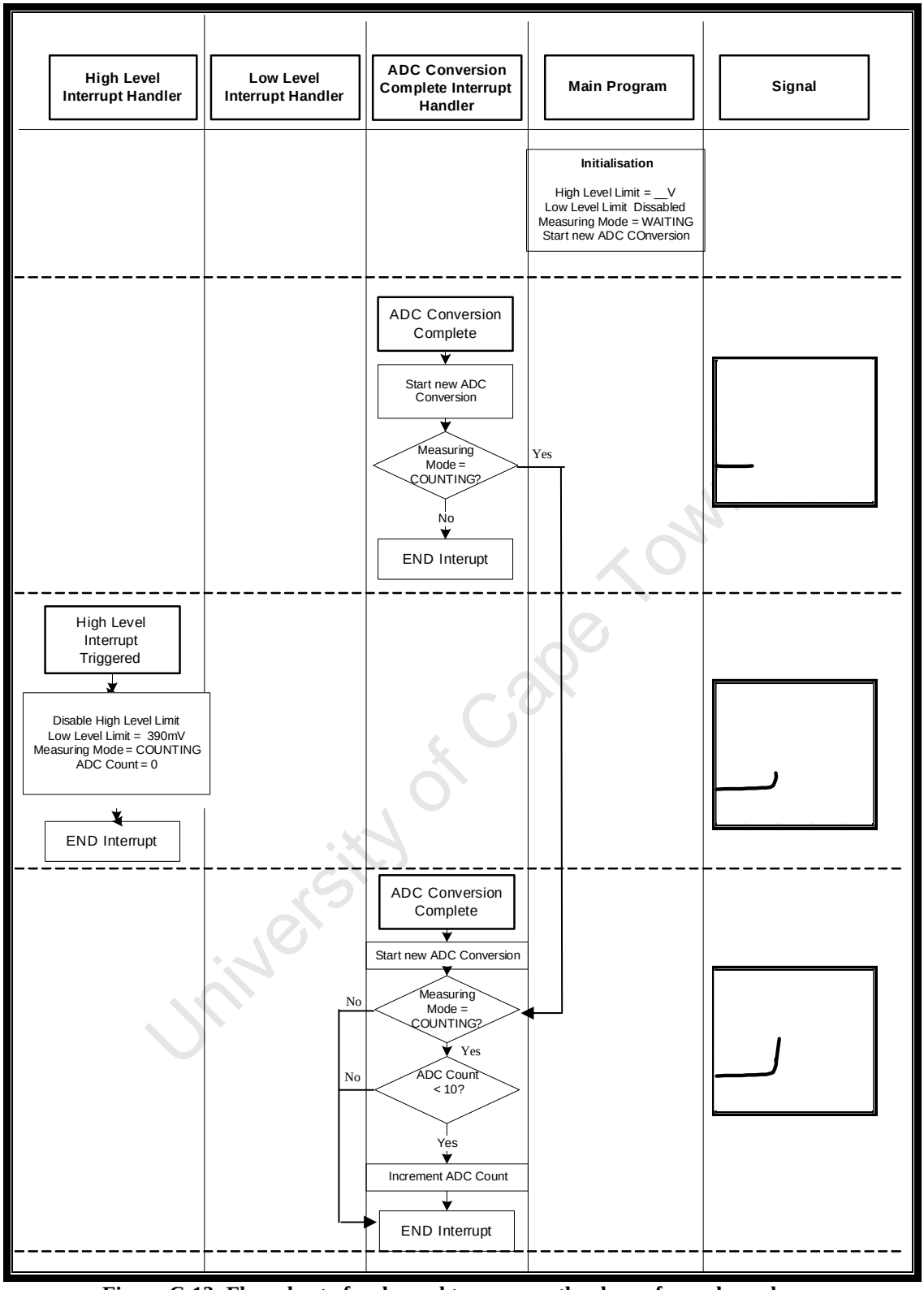


Figure C-13: Flow chart of code used to measure the slope of encoder pulses

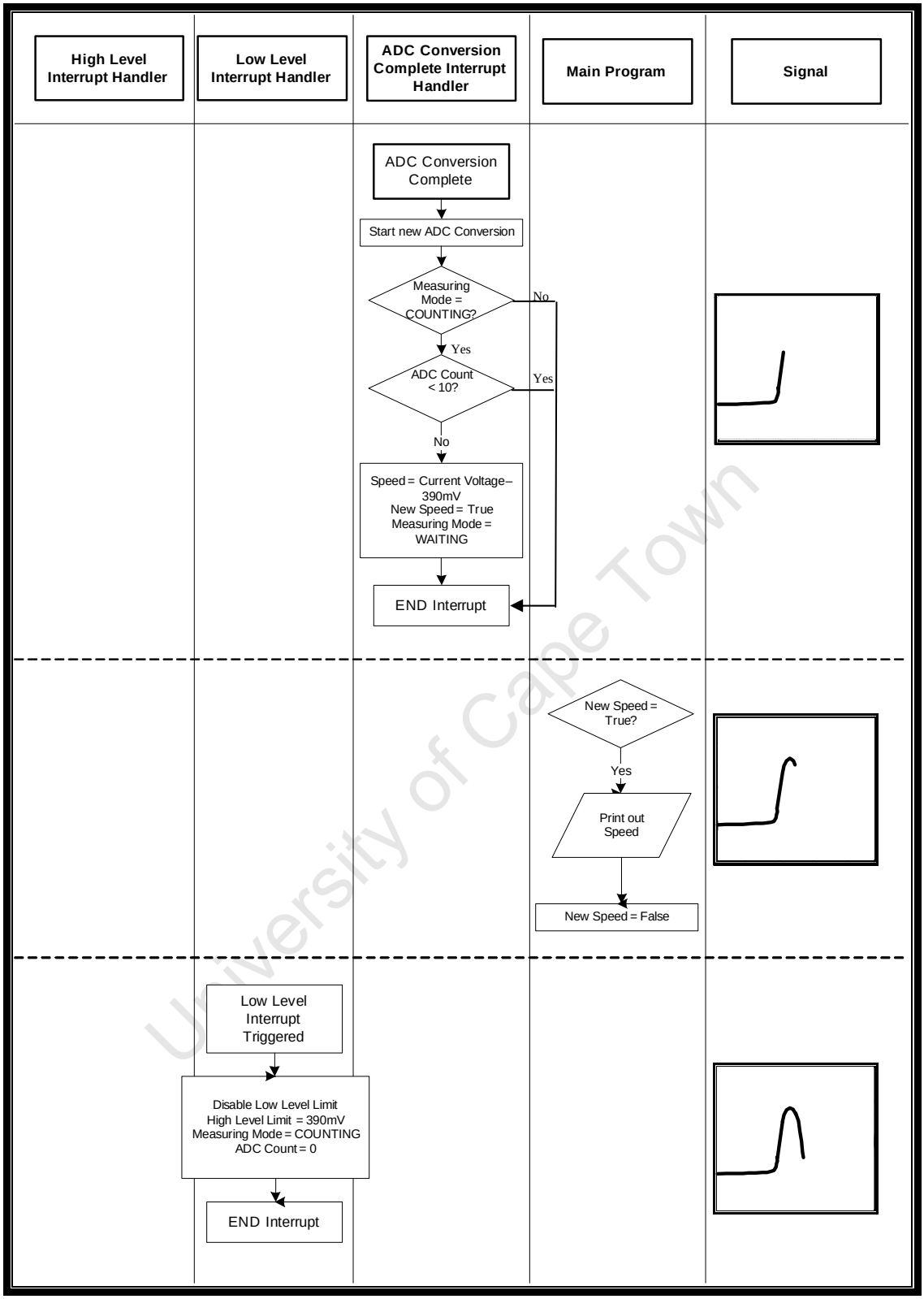


Figure C-13 continued: Flow chart of code used to measure the slope of encoder pulses

the wheels were varied in order to determine how the speed outputs changes as the wheel speeds changed and what the range of values output by the system were. To determine the linearity of the speed measurement system, the frequency of the encoder signals was compared to the speed output at different speed values. The speed output versus frequency graphs of the three wheels were compared to see if the speed outputs from each wheel correlated.

Below, in Figure C-14, is an example of the speed readings output from the DSP as the three wheels are run, each at a different speed. The robot was supported so that the wheels could run under no load, ensuring that the wheels would run at constant speed.

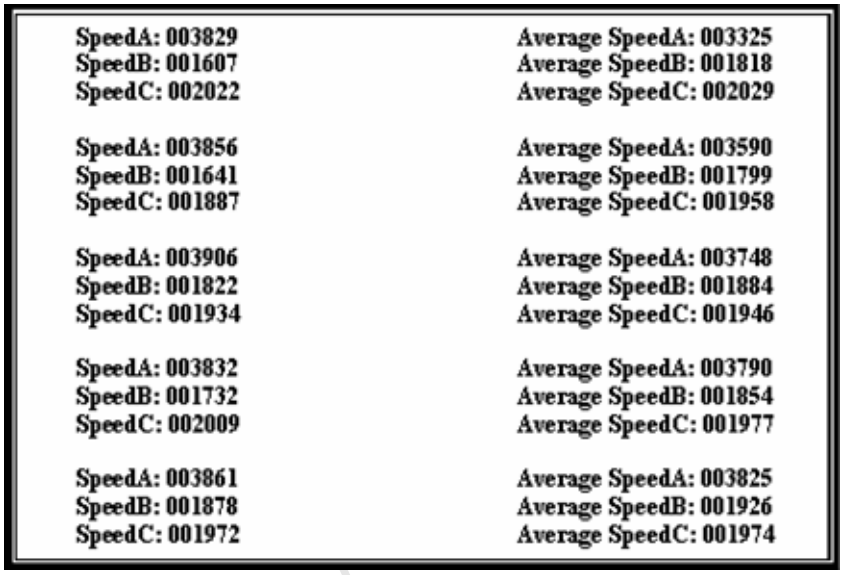


Figure C-14: Example of slope measurements out put from DSP when all three wheels of the robot are running

To test the effectiveness of the measurement method, each wheel was run at 8 different voltage levels to measure the slope at 8 different speeds. The voltage levels were varied by varying the duty cycle of the PWM to the motors. To get an indication of the speed of the wheel, the frequency of the encoder signal was measured on an oscilloscope. The results are shown in Table C-1 over the page and plotted in Figure C-15 on the following page.

Since consecutive speed readings differed dramatically, an averaging algorithm was employed that used a sliding window filter. Consecutive averaging readings differed from each other as much as the non averaged readings. Rather than try another filtering system without being sure that the slope method would be an effective measuring system, to get readings, maximum and minimum values within a range of ten consecutive values were used in the tables below. The motors were run under no load conditions.

Table C-1: Variations of encoder signal amplitude and the slope measured by the DSP code, with changes in speed of wheels.

Wheel A

Pwm Duty Cycle	Frequency	Speed Output		Max Amplitude
	[Hz]	Min	Max	[V]
63%	46.51	771	980	2.53
70%	108.10	1754	1848	2.50
75%	142.90	2174	2293	2.50
80%	185.20	2506	2619	2.47
85%	227.30	2471	2850	2.50
90%	278.00	2307	2965	2.44
95%	313.00	2948	3011	2.40
100%	357.00	2913	3028	2.28

Wheel B

Pwm Duty Cycle	Frequency	Speed Output		Max Amplitude
	[Hz]	Min	Max	[V]
63%	50.76	901	1037	2.56
70%	113.60	1286	1573	2.56
75%	156.30	1489	1654	2.53
80%	192.30	1657	1774	2.53
85%	238.10	1765	1834	2.56
90%	263.20	1845	1892	2.44
95%	313.00	1892	1923	2.37
100%	333.00	1895	1963	2.28

Wheel C

Pwm Duty Cycle	Frequency	Speed Output		Max Amplitude
	[Hz]	Min	Max	[V]
63%	39.68	163	528	2.12
70%	105.30	1087	1195	2.09
75%	149.30	1519	1688	2.12
80%	185.20	1986	2175	2.09
85%	232.60	2310	2402	2.12
90%	278.00	2498	2647	2.12
95%	333.00	2734	2789	2.12
100%	357.00	2853	2968	2.03

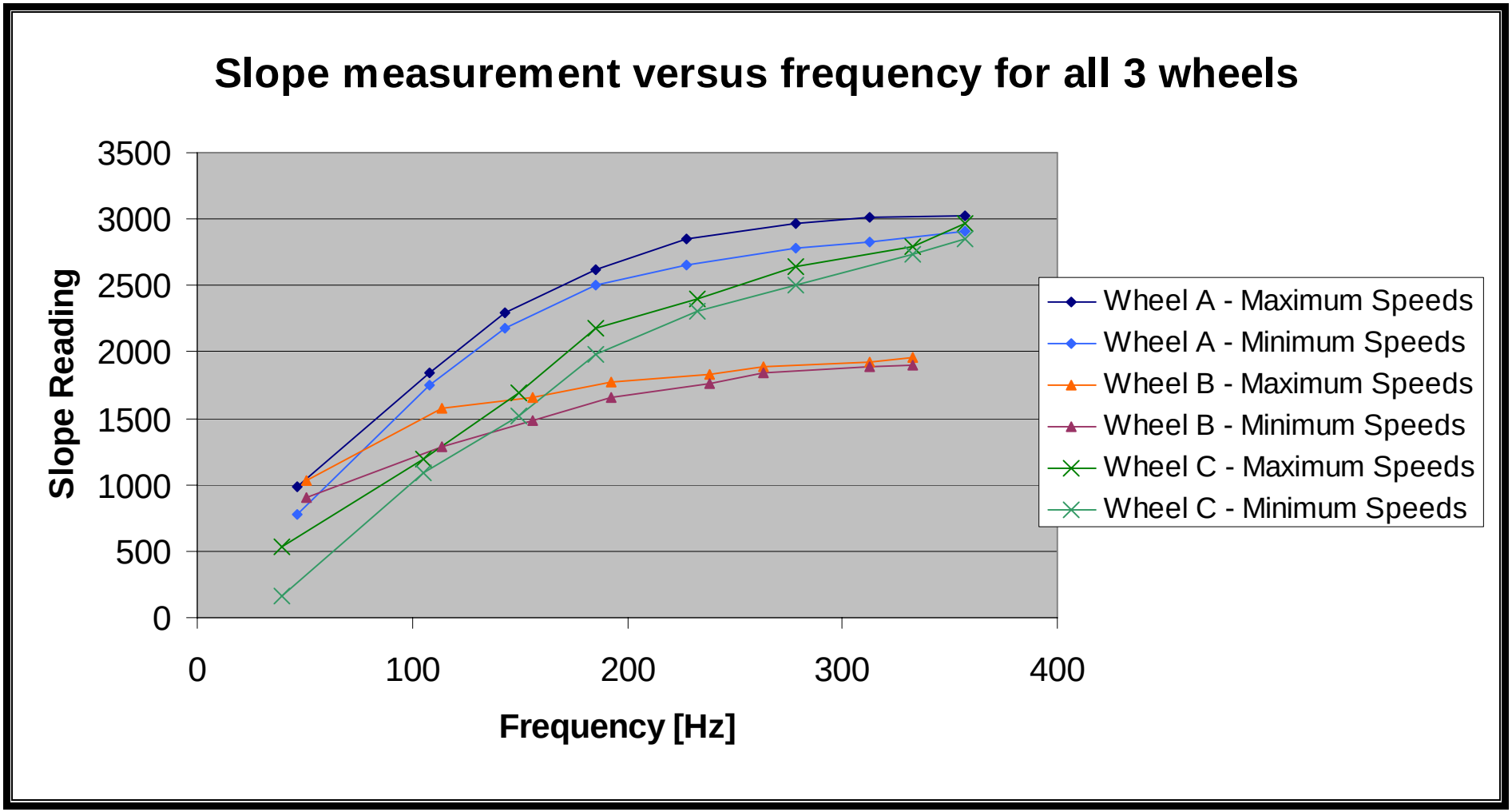


Figure C-15: Graph of Slope Measurement versus Frequency for the 3 wheels of the robot platform: Since slope readings varied greatly at any constant wheel speed, the maximum and minimum speeds noted at each frequency were used to give an indication of range for the slope readings.

Jitter

As can be seen from Figure C-16: Scatter plot of 10 consecutive slope readings with wheel run at constant 48Hz Figure C-14, consecutive values differ largely from each other, despite the wheel speeds being constant. Figure C-16 below is a scatter plot of ten slope readings taken from a wheel run at a constant 48Hz. The values range from 489 to 458. Attempts to use a sliding window filter to average the values and get a more consistent reading were unsuccessful.

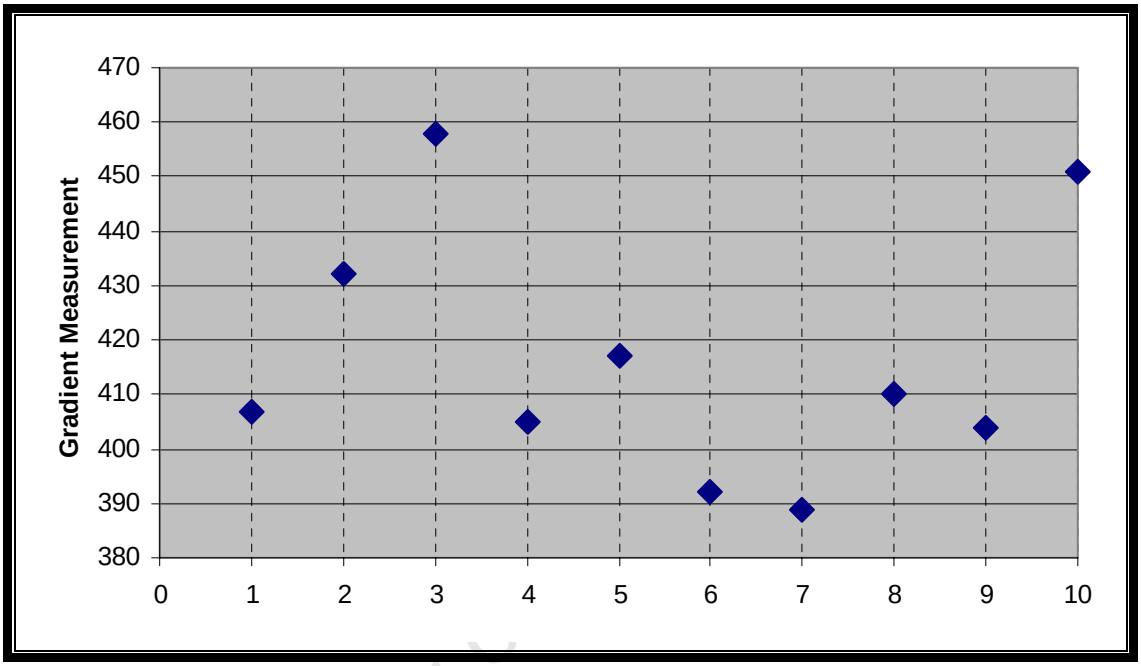


Figure C-16: Scatter plot of 10 consecutive slope readings with wheel run at constant 48Hz

Linearity of Algorithm

The graph in Figure C-15 shows that the relationship between the measured speed (slope) and the actual speed (indicated by frequency) is non linear. This indicates that to use the slope measurement to indicate speed, the slope measurement would have to be normalised so that the final speed output is directly proportional to the speed of the wheels.

Range of Readings

The slope readings for speeds of about 50Hz to speeds of 360Hz, ranged from 163 to 3028. This would provide sufficient speed levels for control of the robot. In Figure C-15 it can be seen that the range of minimum and maximum values for each wheel follow similar curves. The maximum and minimum slope values measured at any speed fall within a specific range; the differences between maximum and minimum values range from 30 – 365. This suggests that despite, the large amount of noise, with filtering, one should be able to get more constant readings at constant speeds.

Measurements on different Wheels

The graph also shows that the three sets of wheels do not follow the same slope versus frequency curve. This suggests that the differences in the physical and electronic setups for each wheel does have a significant effect on the slopes of the pulses from the encoders.

C.8.3 Conclusions and Recommendations

Noise

This method is very susceptible to noise. It had been hoped that irregularities in the encoder would have less effect on the slope method that the frequency method, but this turned out not to be the case.

Linearity

This slope method of measurement is non linear. Although a control system could handle a non linear speed measurement system, the non linearity would slow down the response of the system. A normalising function would ideally have to be used to improve the response of a speed control using the slope method.

Correlation of 3 Wheels

The speed readings from the different wheels are significantly different. A lot of calibration would have to be done to ensure that running two wheels at the same speed would result in the same speed reading. Presumably, small differences in how each sensor is placed in relation to the encoder are responsible for the very different readings. A differently placed sensor would have different amounts of light fall on it as a tooth passes, meaning the gradient of the slopes would differ. Also, slight differences in the electronic components would make differences in how the circuit responds, particularly to rise times. To overcome this one would have to be careful to place the sensors in exactly the same places (these sensors were glued in place by hand), precisely manufactured guides would have to be used. The components would have to be matched to check they have the same resistance, capacitance etc. That would probably improve the uniformity of the measurements from the wheels but extra calibration would probably still need to be done in software.

C.8.4 Overall Effectiveness of Slope Speed Measurement Algorithm

It was hoped that this system would prove less sensitive to inconsistencies in the system than the period method but it has become apparent that the gradient of the slopes are probably more sensitive to inconsistencies than the distance between teeth. When the method was implemented, it was found that the inconsistencies in the encoders have a considerable effect on slope measurements. Consecutive slope measurements at a constant speed varied over a large range. The slope measurements for different wheels run at the same speed also varied. The slope measurement was also not linear with respect to wheel speed. Signal conditioning

of the encoder signal and calibration of the measurements for each wheel may be able to overcome these problems. This would probably prove more difficult that the filtering techniques needed to get constant readings from the more traditional methods. Overall this system shows to have no advantages over traditional speed measuring systems. The method was therefore abandoned in favour of the more traditional tooth counting method

C.9 Tooth Counting

Since the slope measuring method of speed measuring proved to be unsatisfactory it was decided to use the more traditional tooth counting or frequency method. Below is a discussion of the successful implementation of this method using the DSP.

C.9.1 Implementation

A timer on the DSP was set up to trigger an interrupt every 30ms. This time interval was chosen using trial and error to find a time interval that yielded a large enough range of speeds with a reasonable update time. Each time the timer interrupt triggers it resets a tooth counting variable. The high and low level interrupts are used as explained in section C.8.1 to determine the beginning and end of pulses. Each time a pulse passes, the tooth counting variable is updated. When the timer times out, identifying that 30ms have passed, the speed of the wheel is updated to equal the tooth counting variable before the variable is again reset.

Figure C-17 and Figure C-18 below are timing diagrams taken using the oscilloscope to show how the teeth are counted as they pass. The top line is the signal from the encoder, the line below shows the teeth being identified as they pass, and the bottom line shows the beginning of each timing interval. The timing and high and low level interrupts triggered output pins of the DSP which the oscilloscope then read to create the timing diagram. In the first diagram, the wheel is running at 79Hz and only three teeth are passing per interval, so the speed is 3 pulses per 30ms. In the second trace, the wheel is going at the speed of 370 Hz, and 11 teeth pass per interval so the DSP measures a speed of 11 pulses per 30ms.

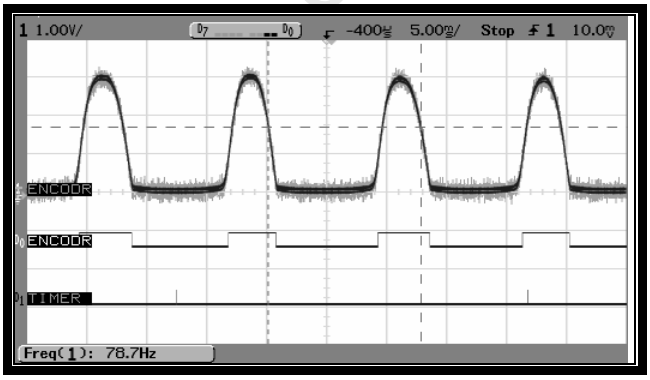


Figure C-17: Timing diagram of tooth counting method at low speed

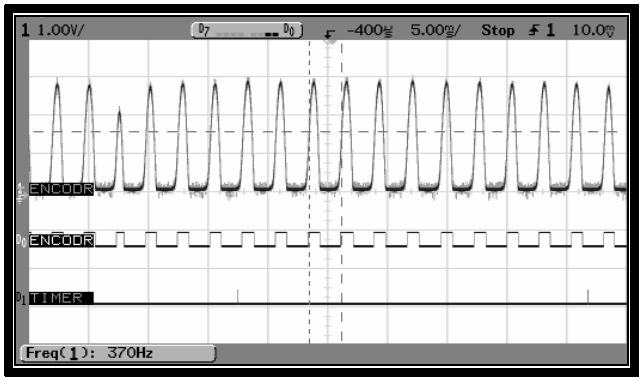


Figure C-18: Timing diagram of tooth counting method at high speed

C.9.2 Results

Figure C-20 on page C-20 shows the speed in pulses per 30ms as measured by the DSP. All three wheels were run at 7 different speeds. The frequency and speed measured by the DSP were noted for each speed and plotted against each other. The frequency ranged from 25 to 296Hz and the speeds from 2 to 18.

Jitter

The noise for this method was much less than for the slope method. Figure C-19 below shows a scatter plot of ten successive readings with the wheel running at constant speed under no load. The speed ranges over three values as would be expected - since the number of teeth to pass the sensor per interval will rarely be a whole number, every few readings can be expected to be one less or more than the most common reading.

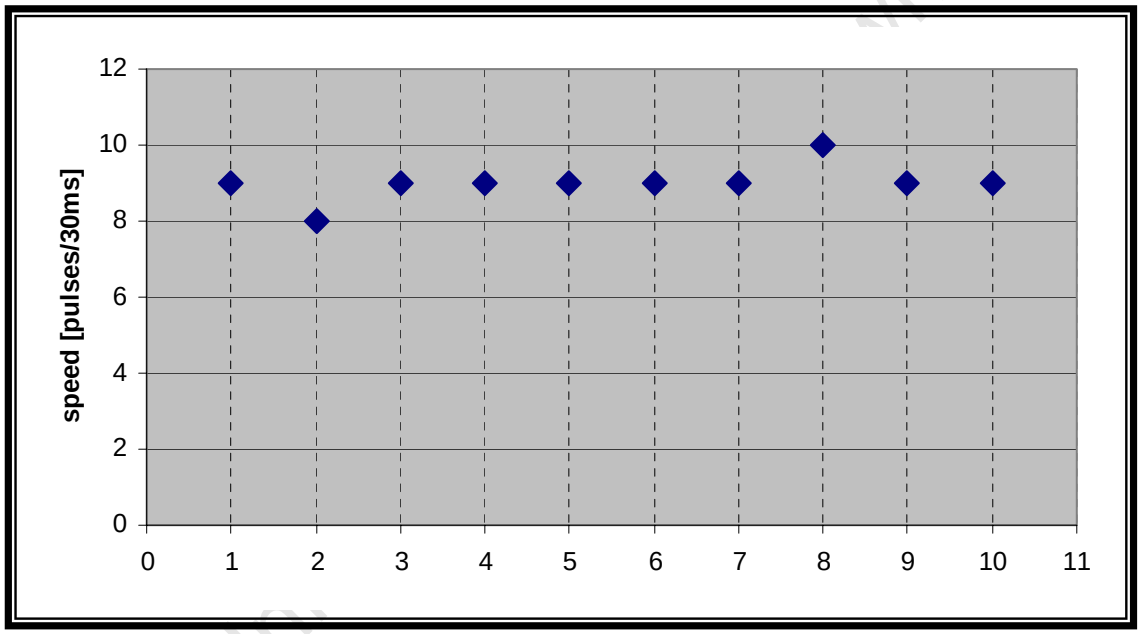


Figure C-19: Scatter plot of ten consecutive readings when measuring using the tooth counting method

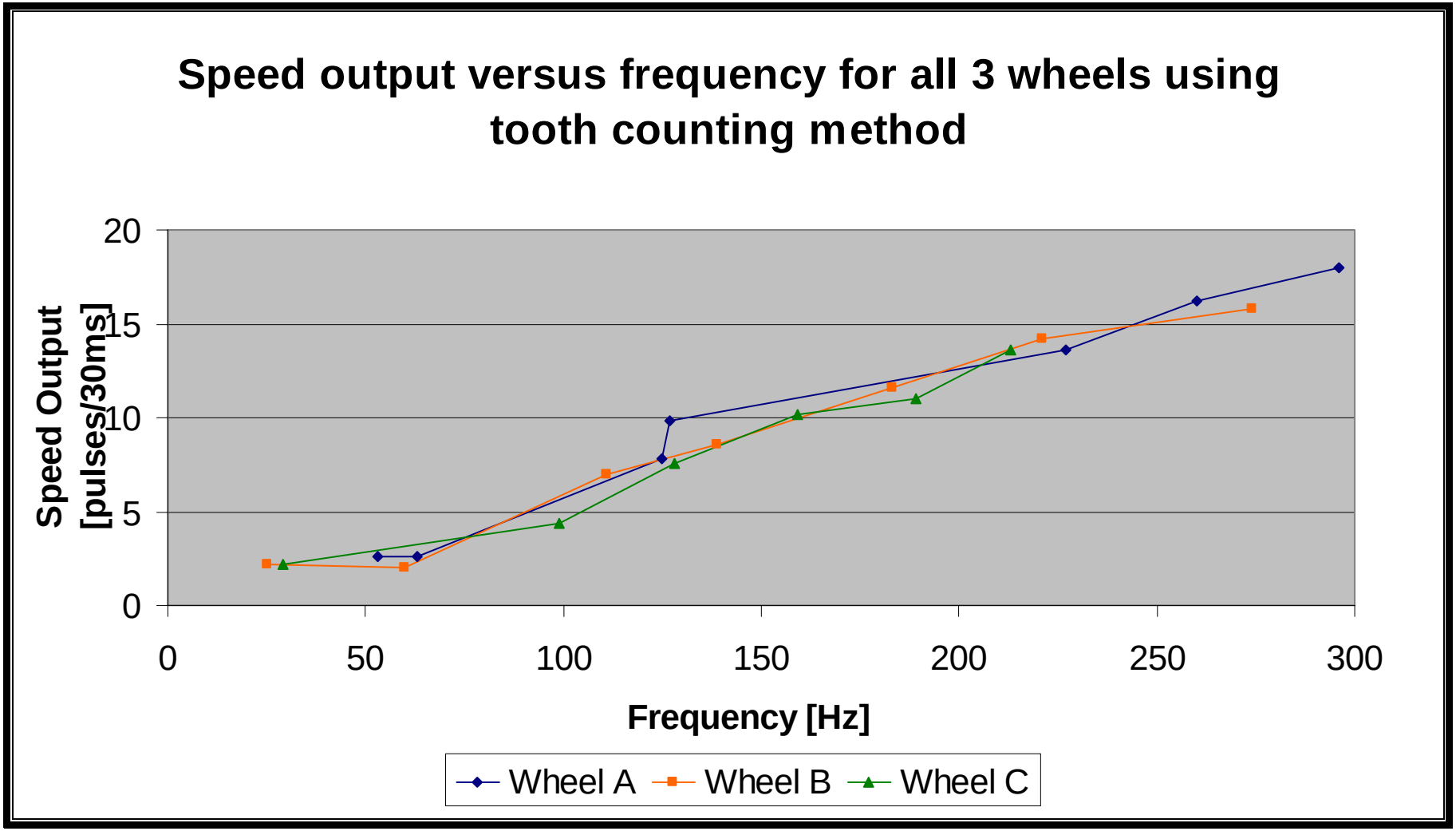


Figure C-20: Speed output versus frequency for all three wheels using the tooth counting method

Linearity of Algorithm

Figure C-20 above shows that the speed as measured by tooth counting has a linear relationship to the frequency of the pulses passing.

Correlation of 3 wheels

From Figure C-20 it can be seen that all three wheels follow the same speed versus frequency curve, indicating that the when two wheels are rotating at the same speed, the tooth counting measurement method reflects this.

C.10 Conclusion

The tooth counting speed measurement method proved accurate, linear and relatively free from noise and was therefore chosen as the speed measurement method on the robot.

References

1. James Kermode's Home Page: *http://www.srcf.ucam.org/~jrk33/ShaftEncoder.gif*. accessed: 3 Jan 2008.
2. Wikipedia, Graycode. *http://en.wikipedia.org/wiki/Gray_code*. accessed: 29 Dec 2007.
3. Braunl, T., Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems. 2nd ed. 2006: Springer.
4. Sekimori, D. and F. Miyazaki. Precise Dead-Reckoning for Mobile Robots using Multiple Optical Mouse Sensors. in Informatics in Control, Automation and Robotics II. 2007: Springer Netherlands.
5. Palacin, J., I. Valaganon, and R. Pernia, The optical mouse for indoor mobile robot odometry measurement. Sensors and Actuators A., 2006. 126: p. 141-147. Accessed via: www.sciencedirect.com.
6. Craig Inman-Bamber, Robot Soccer Platform, in Mechanical Engineering Department. 2005, University of Cape Town.
7. Rataj, D.H., et al., US Patent Number 6289294, 2001: Method for determining rotational data using an encoder device. accessed on URL: http://www.freepatentsonline.com/6289294.html; Accessed 28 June 2007.
8. McPhillips, G., The Control of Semi-Autonomous Robots, in Department of Electrical Engineering. 2004, University of Cape Town.

Appendix D. Wheel Speed Control

Table of Contents

Table of Figures

Appendix D. Wheel Speed Control

D.1 Introduction

When controlling an omnidirectional robot, one has to consider two levels of control; the high level control of the rotational and lateral motion of the robot platform, and the lower level control of the wheels themselves. Appendix H discussed the choice of odometry system. In this appendix, the chosen odometry method, tooth counting, is used to provide feedback for a PI controller. Once this lower level control had been implemented, the control of the robot as a whole could be implemented. This higher level control is discussed in Appendix J.

D.2 PID Control

It was decided to use a PID control algorithm to control the speed of the motors as this is a commonly used method of motor control. The format of the PID controller algorithm is

$$V = P + I + D$$

where:

$$P = K_p e$$

$$I = I_{old} + K_i e$$

$$D = K_d (e - e_{old})$$

V is the voltage to be applied to the motor to be controlled. K_p, K_i and K_d are the proportional, integral and derivative control constants. e is the error, that is the

difference between the speed achieved and the setpoint, e_{old} is the previous value of *e* and I_{old} is the previous value of *I*. Figure D-1 is an illustration of a PID control loop.

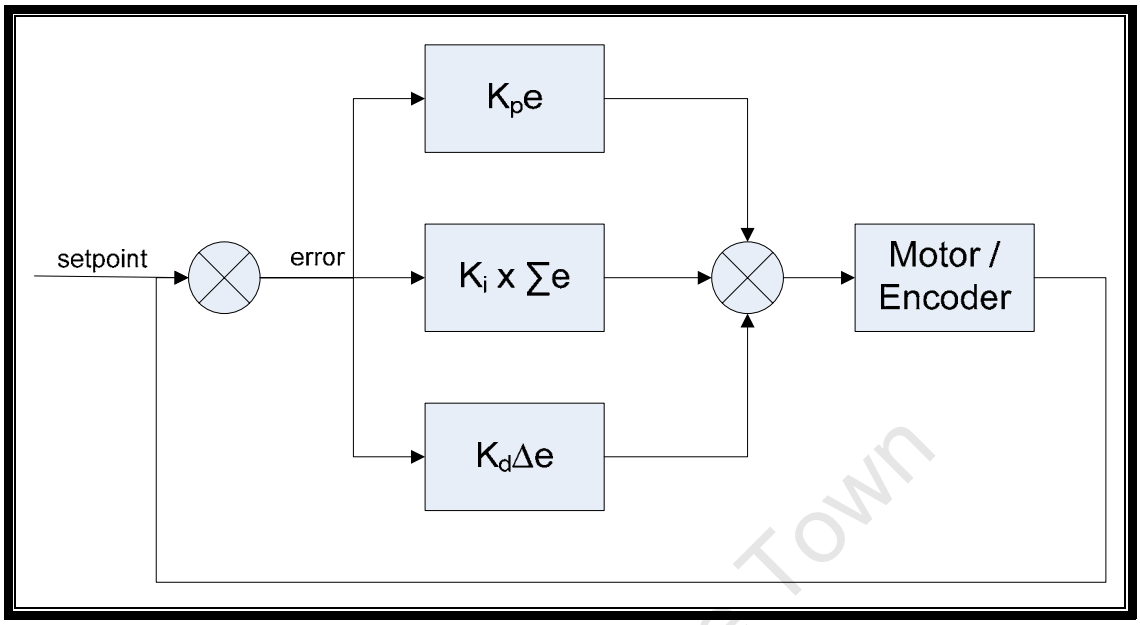


Figure D-1: Closed loop PID control

The proportional term *P*, adjusts the voltage applied to the motor proportional to the error. Increasing the value of *Kp* increases the speed at which the system approaches the setpoint, but increasing it too much results in an oscillatory system. Since an error of 0 would result in an output voltage of 0 which would result in the motor stopping, the system will never quite reach the setpoint, there is always a 'steady state error'[1].

The *I* term seeks to eliminate the steady state error. By integrating the error of the system, the *I* term keeps track of how long the error has been around for. The larger the error over time, the larger the value of I. In this way the I term moves the speed closer to the setpoint by increasing the voltage applied to the motor as long as an error remains. This also means that the *I* term makes the system able to handle disturbances. The P term is tuned for an ideal system, with no disturbances, as soon as a disturbance is introduced, the steady state error will increase, the P term is unable to respond to this change in the system, but since the *I* term responds to the amount of error over time, the *I* term will be able to compensate for disturbances in the system.

A PI controller will reach the setpoint but increasing the K value to get a fast response in the system can mean that the system will overshoot the setpoint before settling. To reduce this overshoot, the D term is introduced. The D term is proportional to the difference between the last two errors. This means it tracks how fast the error is changing and adjusts itself accordingly. As the rate of change in error increases, the D term decreases, making the system approach the setpoint more slowly as it approaches the setpoint, thereby avoiding overshoot.

D.3 Step Tests

D.3.1 Introduction

To test the effectiveness of the control algorithm, a method of recording the changes in speed readings was needed. For this purpose, code was added to the robot controller code to record the speed of the wheels of the robot at each time interval of the control algorithm. First this code was used to record open loop step tests with various voltages applied to the wheels. Later, the code was used to log the speeds once the speed algorithm had been used. The step tests were performed on the felt covered, wooden surface of the robot soccer field.

D.3.2 Time step

The step tests would record the speed of the wheels every few milliseconds, logging this information and outputting it across the serial interface once the step test was complete. One of the first decisions to be made was how often to update the speed measurements. To allow the step test code to be easily upgraded to become the control loop code, it was decided to update the step tests as often as the control loop would update itself. For discrete control one usually wants the sample time to be at lease ten times less than the time constant of the system(T_m). T_m is a measure of the time is takes for the system to reach 63% of its final value when in open loop. Since no open loop step tests had yet been performed on the motors used, it was decided that since the motors used on Mr McPhillips's robots had been similar, it would be assumed that the time constant was the same. The time constant of Mr McPhillips's motors was 150ms [2], so it was originally decided to make the sample time for the step tests around 15ms.

However, in digital control, the time constant for the control algorithm is limited by the time interval over which the speed is measured. The control algorithm cannot be updated more often than the speed interval is updated. As described in Appendix H, the speed was measured by counting the number of encoder teeth to pass in a 30ms interval. This was greater than the 15ms suggested, but it was decided to test the effectiveness of this 30ms time interval. If this time step had proved too large for fine motor control, a method of updating the speeds more often would have had to be found.

D.3.3 Implementation

The robot is connected to a computer via a serial link. The operator can give the robot commands via this link. There are 4 commands for step tests. Three commands tell the robot to perform an open loop step test, either at 50%, 75% or 100% PWM duty cycle. The forth command tells the robot to perform a closed loop step test, the parameters of which the robot prompts the operator to enter. Table D-1 below lists the four commands for the step tests.

Table D-1 : Command structure for step tests

Command	Step Test
T100	Open Loop Test at 100% PWM
T200	Open Loop Test at 75% PWM

T300	Open Loop Test at 50% PWM
T500	Closed Loop Test

Figure D-2 below, is a flow chart of the code to run the step tests. The user enters a command. The robot then acknowledges the command and runs the test. If the test is an open loop test, two motors are run at the appropriate PWM duty cycle by calling the *drivemotor()* function. If the test is a closed loop test, the user is asked to input the *setpoint*, K_p, K_i and K_d values and the *controlmotor()* function is called which implements the control algorithm. The motors now run while the DSP logs 80 successive speed readings. Each time the timer overflows, the current speed reading is added to the buffer of speed readings. After the 80 readings have been logged (3,84 seconds when the timing interval is 30ms) the robot stops the wheels and sends the speed readings to the computer via the serial port. These readings were copied into Excel and used to plot the step responses of the robot.

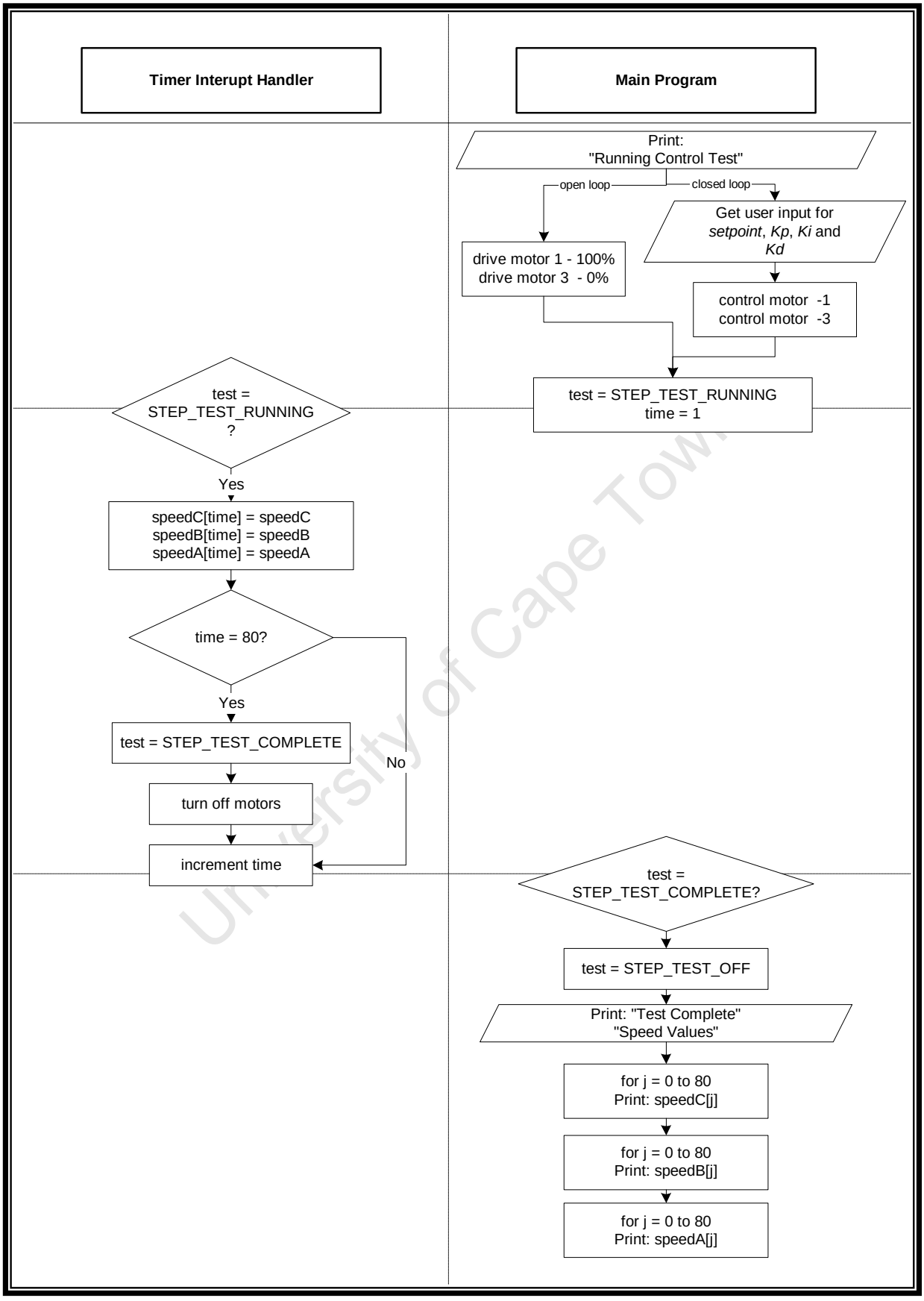


Figure D-2: Flow chart of open loop step test conducted at full power

Figure D-3 below shows a screen shot of the serial interface when a 100% power, open loop test is run. The user input is in blue and the output from the robot is in red. The user enters the commands ‘T100’. The robot drives two wheels to go forward, by applying the appropriate PWM signal to the wheels. The figure shows the speed readings as outputted to the serial port (for the sake of brevity only the first and last speed readings on each wheel are shown).

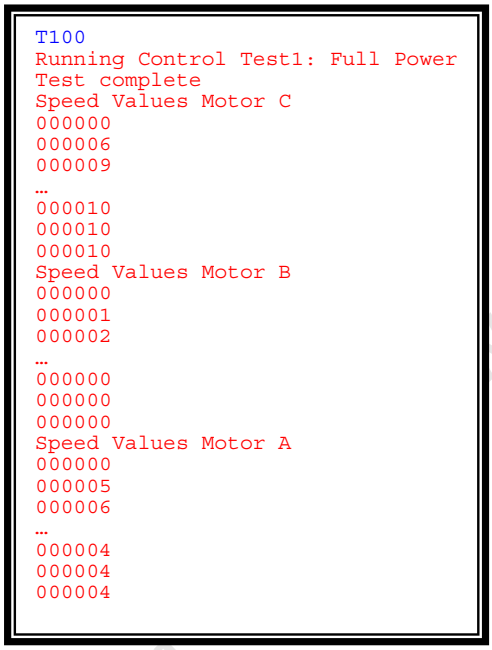


Figure D-3: Serial interface when running open loop step test.

D.3.4 Open Loop Tests

To determine the behaviour of the motors in open loop the robot was run in an approximately straight line by running two wheels at a time. To do this one motor is run in its forward direction and another in its backward direction. The step inputs to the motors were 100%, 75% and 50% of the maximum. Locked antiphase PWM is used to drive the motors. Therefore to drive the robot at full speed in one direction one motor was run at 100% duty cycle and the other at 0%, thereby applying full voltage to both wheels but in opposite directions. Table D-2 below shows the duty cycles that were applied to the wheels to apply 100%, 75% and 50% of the maximum voltage (Vmax) to the wheels. The lowest speed tested was 50% of Vmax since below this the torque applied to the motors tended not to be enough to move the robot.

Table D-2: Duty cycles applied to motors to drive at various speeds

Voltage applied to motor	PWM duty cycle Forward driving Motor	PWM duty cycle Backward driving motor
Vmax	100%	0%
0.75 Vmax	75%	25%
0.5 Vmax	87.50%	12.50%

The assumption that a wheel run at some percentage of the forward voltage would run similarly to a wheel run at the same percentage of the backward voltage, was proven to be correct when the step tests for both wheels were compared. Below, Figure D-4 shows the open loop step tests done at 50%, 75% and 100% of the forward voltage. The speeds were recorded for the two wheels moving the robot. Since the step responses of these two motors were almost identical, it was assumed that all three motors would respond the same when controlled so the tests were not repeated for the third wheel. The results below are those for one wheel only. Vmax was 14.8V. Power and serial communications were supplied to the robot using a tether.

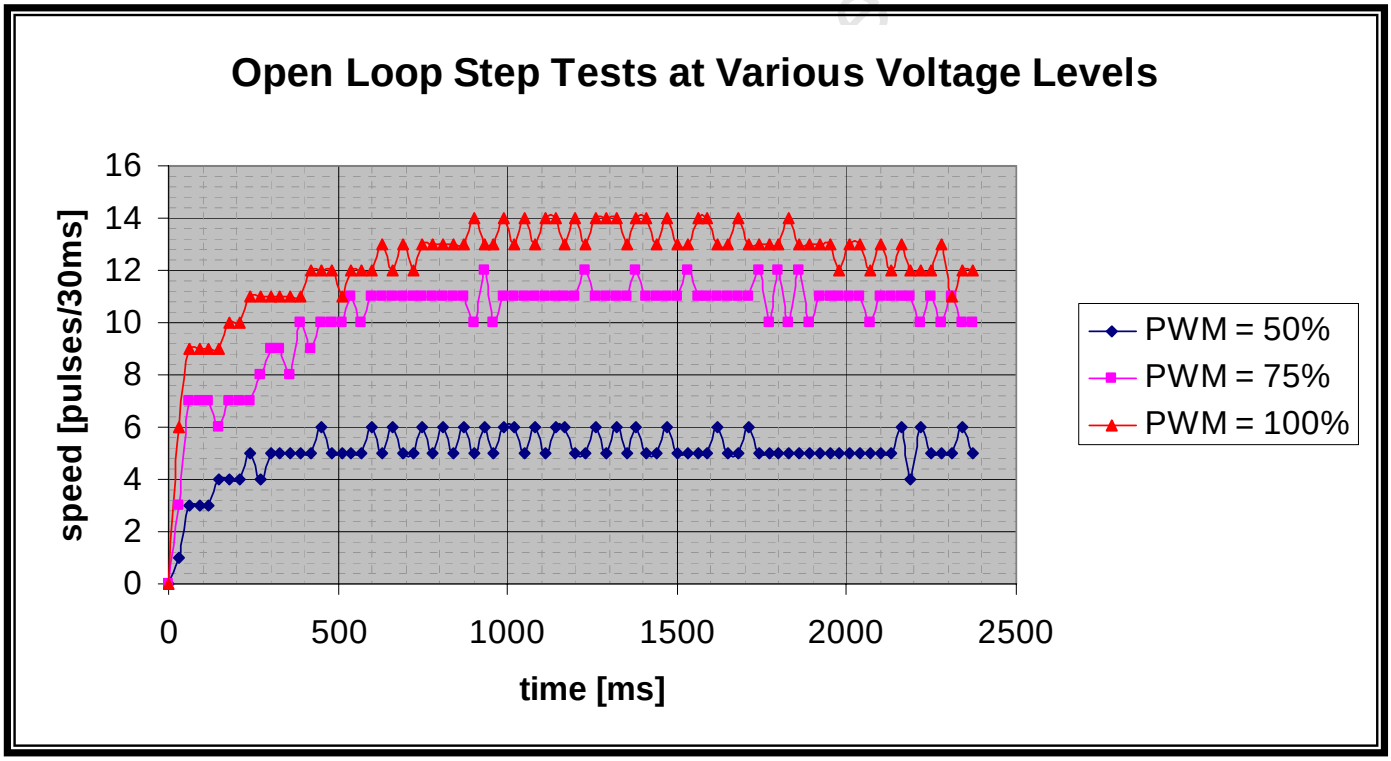


Figure D-4: Open Loop Step Tests

From the 100% step test it was determined that at full speed the motors can reach a speed of 14 pulses per 30ms. From the step tests it was determined that the motors had a time constant of between 160 and 260ms with an average of 243ms. A time constant of 150ms had been expected i.e. the response of these motors was slightly slower than that of those used by Mr Mc Phillips. This would mean than ideally the

time step of the control algorithm should be about 24ms, the time step of 30ms was therefore expected to give an adequate response.

D.3.5 Closed Loop Control Algorithm

As already shown in the flow chart in Figure D-2, when the user chooses to conduct a closed loop test, they are first asked the enter the parameters for the control loop and then the method *controlMotor()* is called.

A flow chart of *controlMotor()* is shown below in Figure D-6. For brevity only the logic to control one wheel (Wheel C) is shown. The code first checks that the speed the user wants to go is between -20 and 20. If the speed is too large, an error message is outputted, if the speed is 0, the motor is stopped and the state variable *controlC* is set to FALSE. Otherwise the speed becomes the setpoint of the control algorithm, the state variable is set to TRUE and the value of the PWM to the motor is set to either BACKWARD_ZERO or FORWARD_ZERO depending on the direction of the speed.

The timer interrupt is set to trigger every 30ms. If, when the interrupt is triggered, the control flag *controlC* is TRUE, then the function controlWheelC() is called. This function is illustrated by the flow chart in Figure D-7 below. The error and PWM offset are calculated as explained in section D.2. Depending on if the speed is in the forward or backwards direction, the PWMoffset is added to either FORWARD_ZERO or BACKWARD_ZERO respectively. If the new PWM value would be outside the range of the PWM code, the PWM value is set to be at the maximum or minimum of that range. The motor is then driven at this new PWM value. Lastly the current error becomes the old error and the current K_i becomes the old K_i.

D.3.6 Overcoming the dead band

When the value of the pwm to be sent to the motor is calculated, the value *pwmoffset* is not added or subtracted from the value for a 50% duty cycle, but from the values BACKWARD_ZERO or FORWARD_ZERO. This is done to overcome the motor's dead band. Low voltages applied to the motors do not make them move at all, it is only once a high enough voltage is applied to the motors that they start to move. If the control algorithm starts at 50% duty cycle, the system takes time to react, for this reason the control algorithm is structured to work outside of the deadband. The values for BACKWARD_ZERO and FORWARD_ZERO were found through trial and error. Figure D-5 below illustrates the concept.

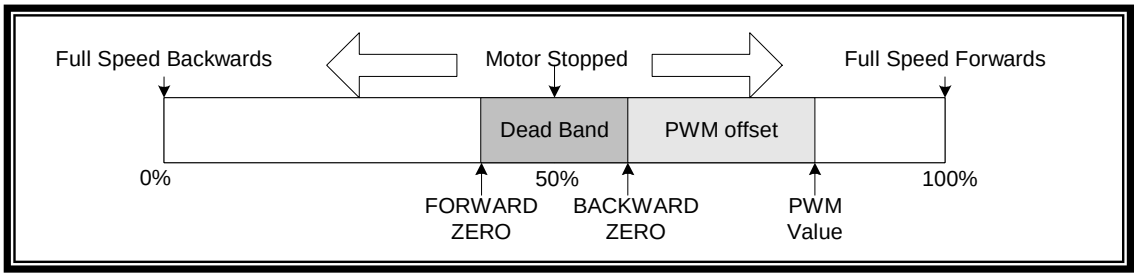


Figure D-5 : format of PWM used to overcome the deadband

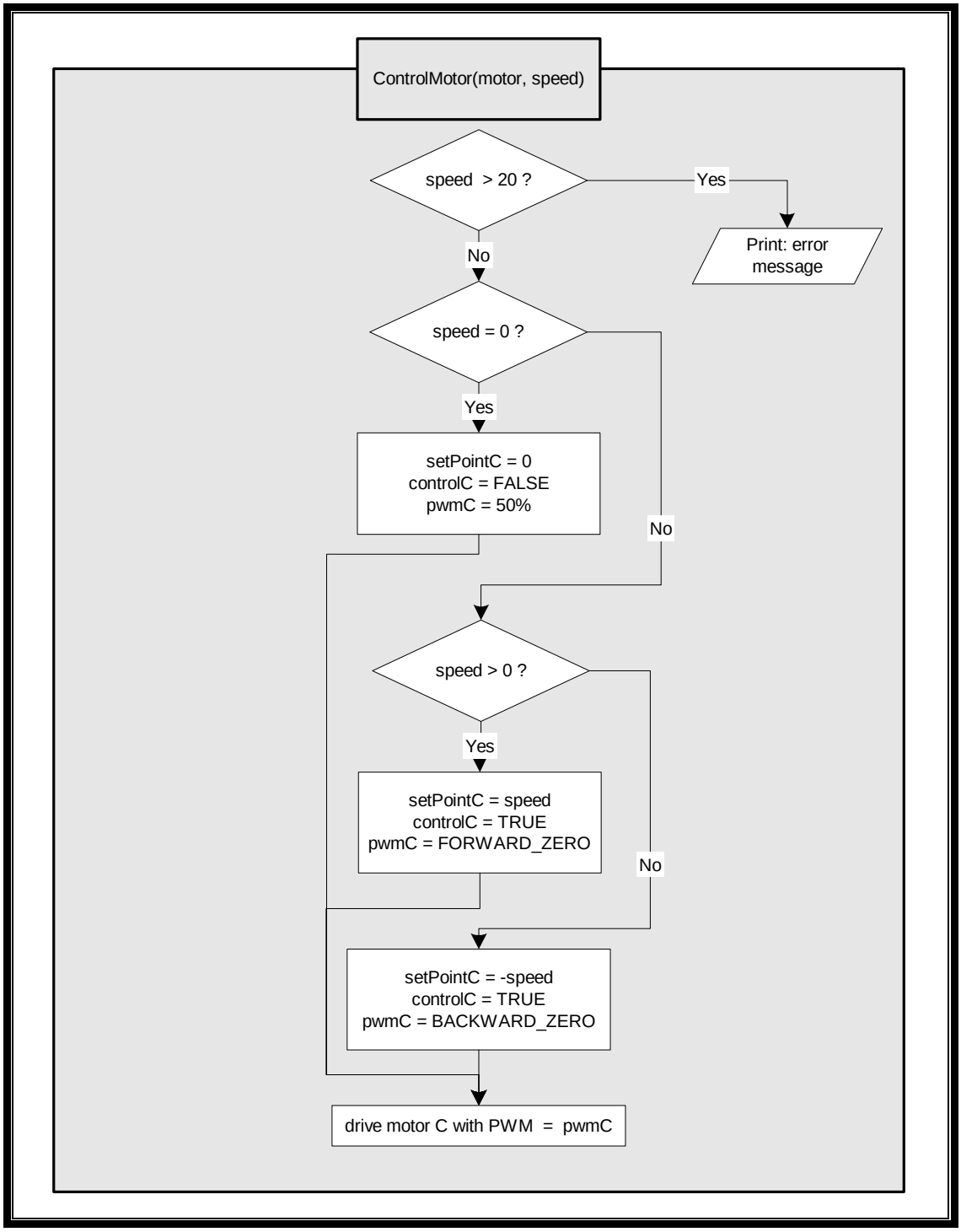


Figure D-6: Flow Chart of code which initiates control algorithm

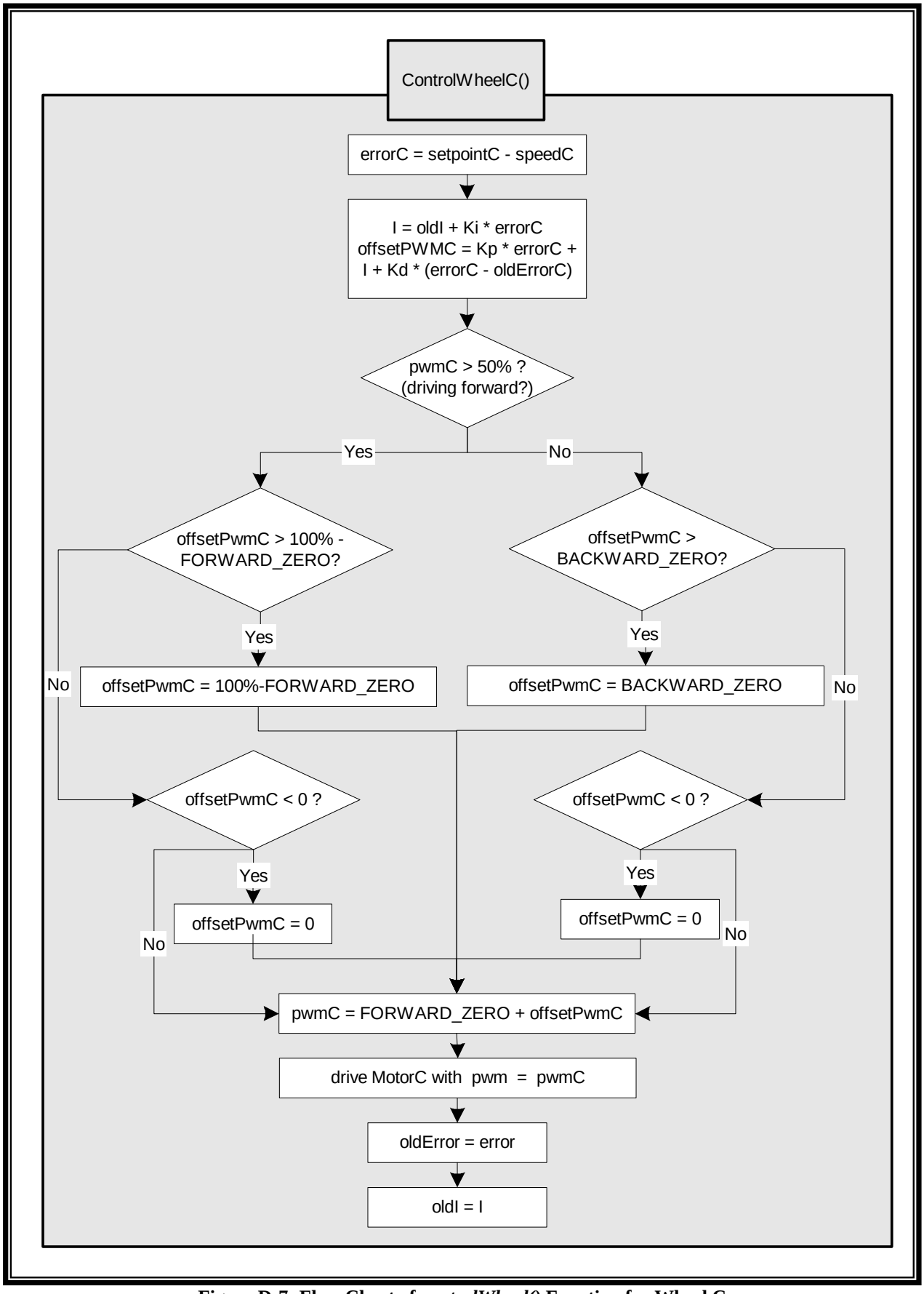


Figure D-7: Flow Chart of *controlWheel()* Function for Wheel C

D.4 Loop Tuning – Ziegler Nichols Method

D.4.1 Introduction

To find the optimum values of P, I and D for the control loop, the Ziegler-Nichols ultimate cycle loop tuning method was used [3]. In this method, also known as the ultimate sensitivity method [4], one first implements a P controller, slowly increasing the value of K_p until the system starts to oscillate. The value of K_p at which the system starts to oscillate is known as the critical gain [5], here referred to as P_c. The period of oscillation (T_c) is also noted. Optimum values for K_p, K_i and K_d for a P, PI or PID controller are derived from the values of P_c and T_c according to Table D-3 below.

Table D-3: Tuning table for Ziegler-Nichols' Ultimate Cycle tuning method [3]

	Controller Settings		
Controller Type	K_p	K_i	K_d
P	$0.5P_c$		
PI	$0.45P_c$	$\frac{1.2}{T_c}$	
PID	$0.6P_c$	$\frac{2}{T_c}$	$\frac{T_c}{8}$

D.4.2 Results

Since the maximum speed that the motors can reach is 14, it was decided to run the step tests at a setpoint that the motors could reach comfortably. The chosen setpoint was 8. By running control tests with different values of K_p, varying from 1 to 1000, and with K_i and K_d set to 0, it was found that oscillation starts when K_p is at about 300. Figure D-8 shows the results of step tests for K_p = 150, before oscillations start, K_p = 300, the critical gain at which oscillations start and K_p = 500 where oscillations are clearer.

To find the period of oscillations, Tc, the period was measured over areas where the oscillations were clear on a number of step tests. The average period was found to be 90ms. Since the units of the control algorithm are time steps of 30ms, this means that Tc = 3. Table D-4 show the values of Ki, Kp and Kd as deduced from these values of Kc and Tc using the Ziegler Nichols Method.

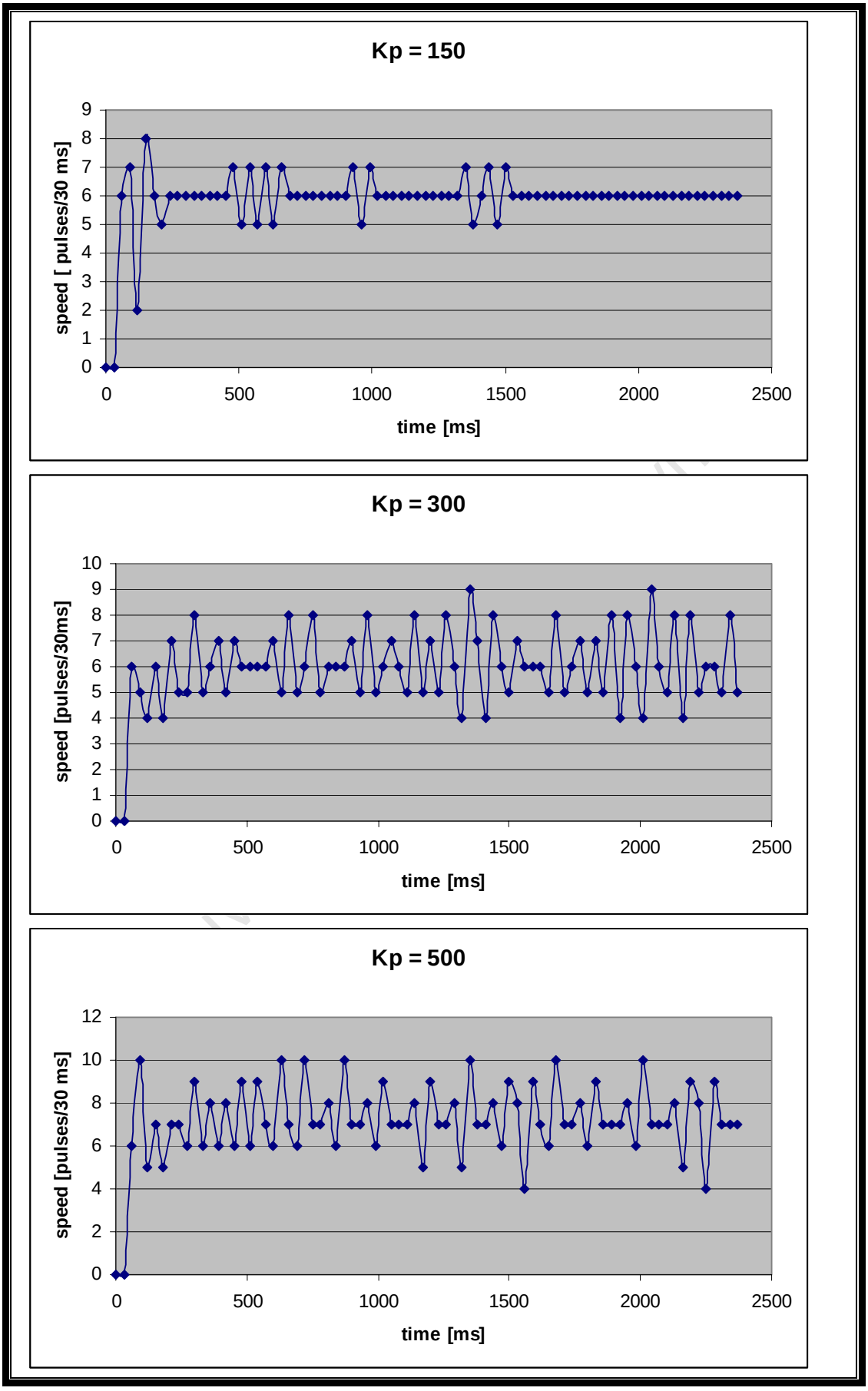


Figure D-8: Step tests to determine the critical gain Kc

Table D-4: Tuning values determined for the motors using the Ziegler Nichols Method

	Controller Settings		
Controller Type	K_p	K_i	K_d
P	150		
PI	135	0.4	
PID	180	0.66	0.38

D.4.3 Effectiveness of Method

Figure D-9 below shows a step test of the PID control algorithm run with the tuning values suggested by the Ziegler Nichols Method and shown in Table D-4 above. It is evident from the graph that the system does not reach the setpoint of 8. This suggests that the Ki value of the PID controller needs to be increased. Through trial and error a suitable value of Ki was found at Ki = 20. Figure D-10 below shows the effect of varying Ki from the suggested 0.66 to 20.

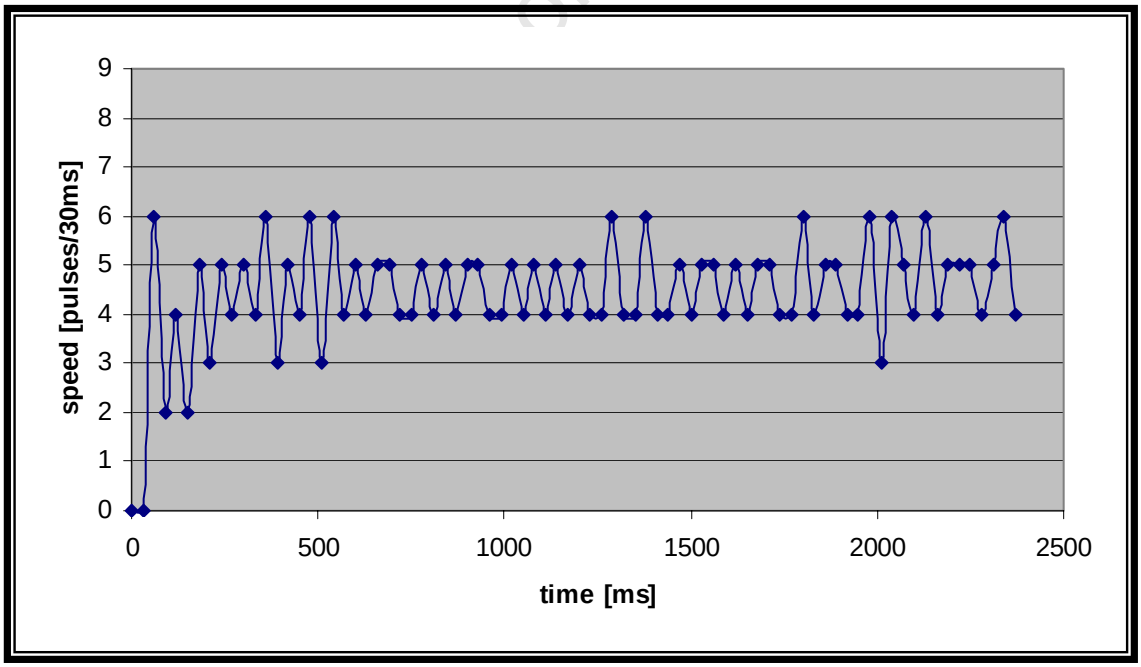


Figure D-9: Step tests with K_p, K_i and K_d values suggested by Ziegler-Nichols method. The setpoint is 8.

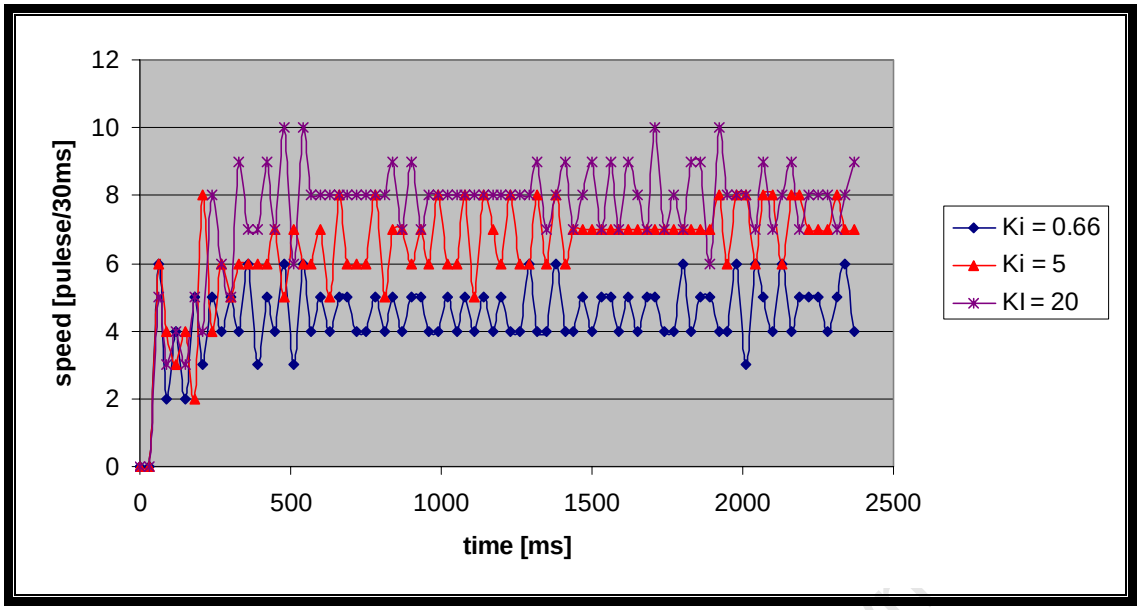


Figure D-10: Step tests using K_p and K_d suggested by Ziegler-Nichols, but varying K_i

Tests were run with the new K_i value while keeping the values of K_p and K_d suggested by Ziegler-Nichols. The graphed step tests shown below in Figure D-11 show that the motors are controlled at various speeds. It was found that when running the robot with control on both wheels the robot ran in a straight line, showing that the control algorithm and chosen values were effective.

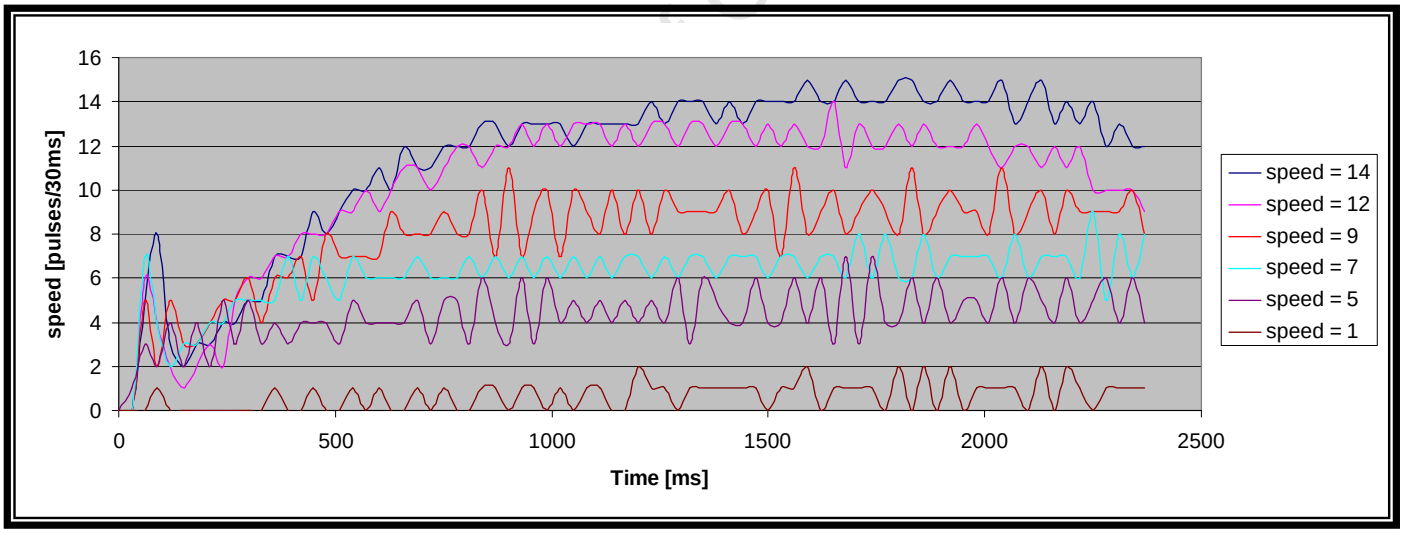


Figure D-11: Step tests for varying speeds using the chosen values of Kp, Ki and Kd

D.5 Conclusions

The Ziegler Nichols Method was used to tune the parameters of a control algorithm to control the speed of the robot's motors. The method proved effective in finding a value for K_p. The value for K_i had to be found through trial and error. The value of K_d was determined using Ziegler Nichols was used. The following values were chosen:

$$K_p = 180$$
$$K_i = 20$$
$$K_d = 0.38$$

The time period of 30ms was used for the time step of the control algorithm and proved effective.

References

1. Braunl, T., *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. 2nd ed. 2006: Springer.
2. McPhillips, G., *The Control of Semi-Autonomous Robots*, in *Department of Electrical Engineering*. 2004, University of Cape Town.
3. Bolton, W., *Mechatronics, Electronic Control Systems in Mechanical Engineering*. pgs 238 -240.
4. Astrom, K.J. and B. Wittenmark, *Computer Controlled Systems - Theory and Design*. pgs 186 -187.
5. *Implementation of Self Tuning Algorithms - IEEE Control Engineering Series*, ed. P.D.P. Atherton and D.K. Warwick.

Appendix E. Robot Platform Control

Table of Contents

Table of Figures

Appendix E.Robot Platform Control

E.1 Introduction

In order to drive the robotic platform, one has to understand the kinematics of the platform. One needs to be able to know what speed to set the different wheels to run at to achieve a desired movement in a desired direction. The kinematics of conventional vehicles are fairly well understood, and simple to derive. The derivation of models to predict the movements of omnidirectional vehicles are more complicated, and less well understood. It should be noted that the level of complexity with which one can model an omnidirectional platform differ greatly. The simplest method is to consider the platform to be a point and for all wheel to be acting directly from that point. More complicated models take into account slippage and the fact that the wheels are spaced further out. One obviously wants to use a model that is as simple as possible, but without sacrificing accuracy.

For this project it was decided to use the simplest possible route when modelling the kinematics. Testing these kinematics would then let us know if we needed to replace the model with a more complex one, or could let the controlling PC make up for any inaccuracies in the model. What follows is the derivation of this simple model, an explanation of how it was implemented on the robot, and the results of testing done on the robot once this control had been implemented.

More complex analyses of the kinematics of omnidirectional robots can be found in Kalmar-Nagy[1] et al and Saha et al[2].

E.2 Kinematic Model

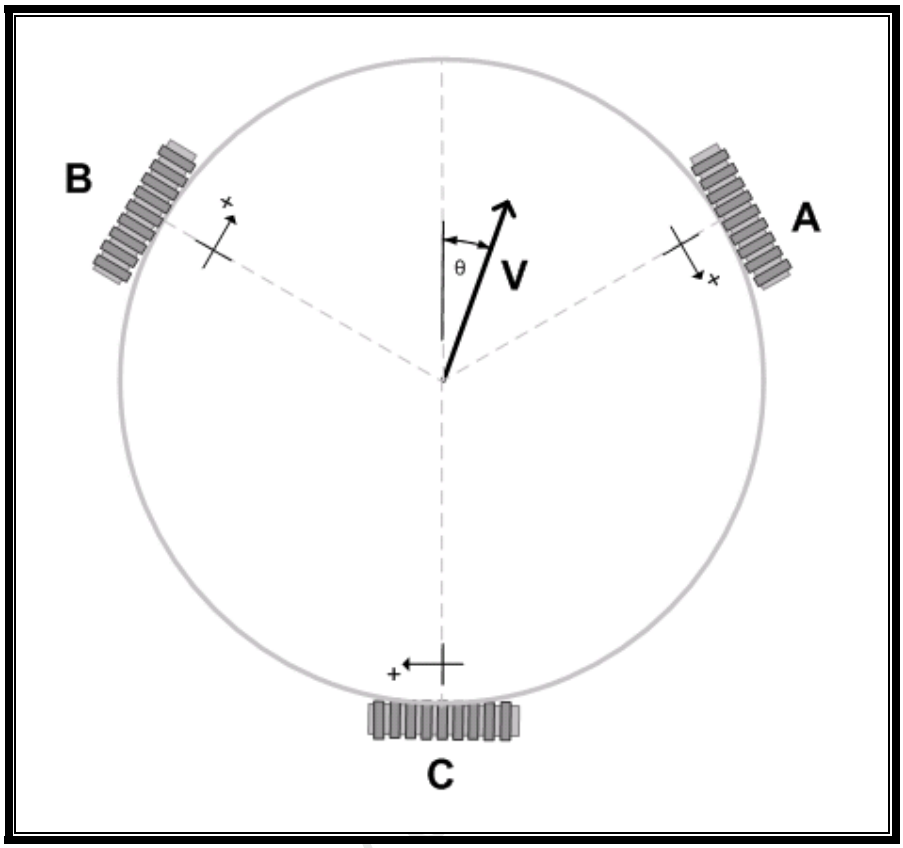


Figure E-1: Diagram of Robot as seen from above

Figure E-1 above shows a diagram of the robot from above. The three wheels are spaced 120º from each other and have been labeled A,B and C. The positive direction of motion for each wheel is taken as the clockwise direction when looking at the robot from above.

The robot can perform either linear or rotational motion, giving it three degrees of freedom. To rotate in either direction, the wheels are all run at the same speed in the same direction. Running the wheels in the positive direction results in a clockwise revolution of the robot.

The simplest lateral movement of the robot is in the direction of any of the wheels A,B or C. For example in Figure E-2 below the robot is moving in the direction of wheel C (bearing $\theta = 180°$). To do this both Wheels A and B are driven: A in the positive direction and B in the negative direction. Wheel C will only experience motion in the direction of $\theta = 180°$ i.e. its hub will remain stationary while its rollers roll. Wheels A and B will experience motion of both the hubs and the rollers, as can be seen in the more detailed view of wheel B.

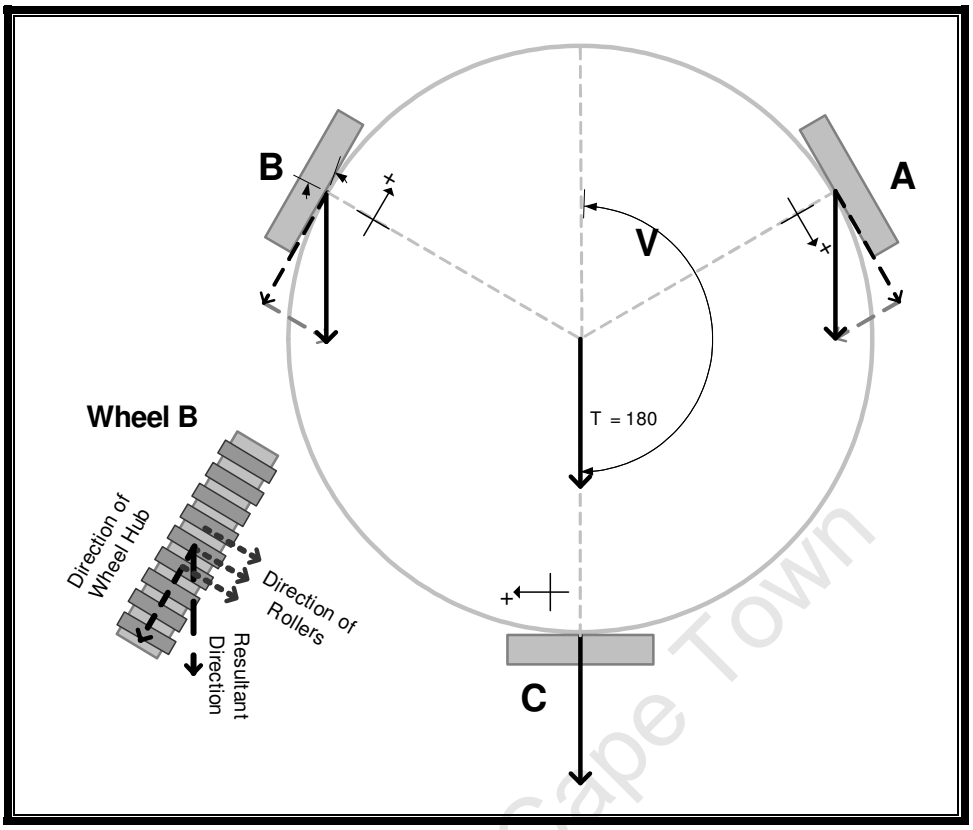


Figure E-2: Robot Driving in the direction of wheel C.

More complex linear motion can be derived using the vector diagram in Figure E-3 below. Here, the desired direction is θ and the desired speed is V. Each wheel can be considered to have two types of motion, the driven motion of the hub, which follows a line parallel to the edge of the robot, and the driven motion of the rollers, which roll perpendicular to the edge of the robot. The speed at which any wheel must be driven can be determined as the projection of the robot's velocity vector in the direction of the wheel's driven vector. Projecting the desired vector onto the driven direction of each wheel, results in the speed at which that wheel should be driven.

In the diagram below, to drive in a direction θ, with a speed V, the wheels A, B and C must be driven at the following speeds:

$$V_A = -V\sin(60-\theta) \quad [1]$$

$$V_B = V\sin(60+\theta) \quad [2]$$

$$V_C = -V\sin(\theta) \quad [3]$$

This is an equation for linear motion only. As already mentioned, rotation is achieved by turning all three wheels at the same speed, in the same direction. So to rotate the robot with an angular velocity of ω,

$$\omega = \frac{V_{A,B,C}}{r} \quad [4]$$

Where $V_{A,B,C}$ is the velocity of wheels A,B and C and r is the radius from the centre of the robot to the centre of the wheel hubs.

Additionally, for small distances, to drive in a direction θ, with a speed V, and an angular velocity ω*r*, the wheels A, B and C must be driven at the following speeds:

$$V_A = -V_{lat}\sin(60-\theta)+V_{rot} \quad [5]$$

$$V_B = V_{lat}\sin(60+\theta)+V_{rot} \quad [6]$$

$$V_A = -V_{lat}\sin(\theta)+V_{rot} \quad [7]$$

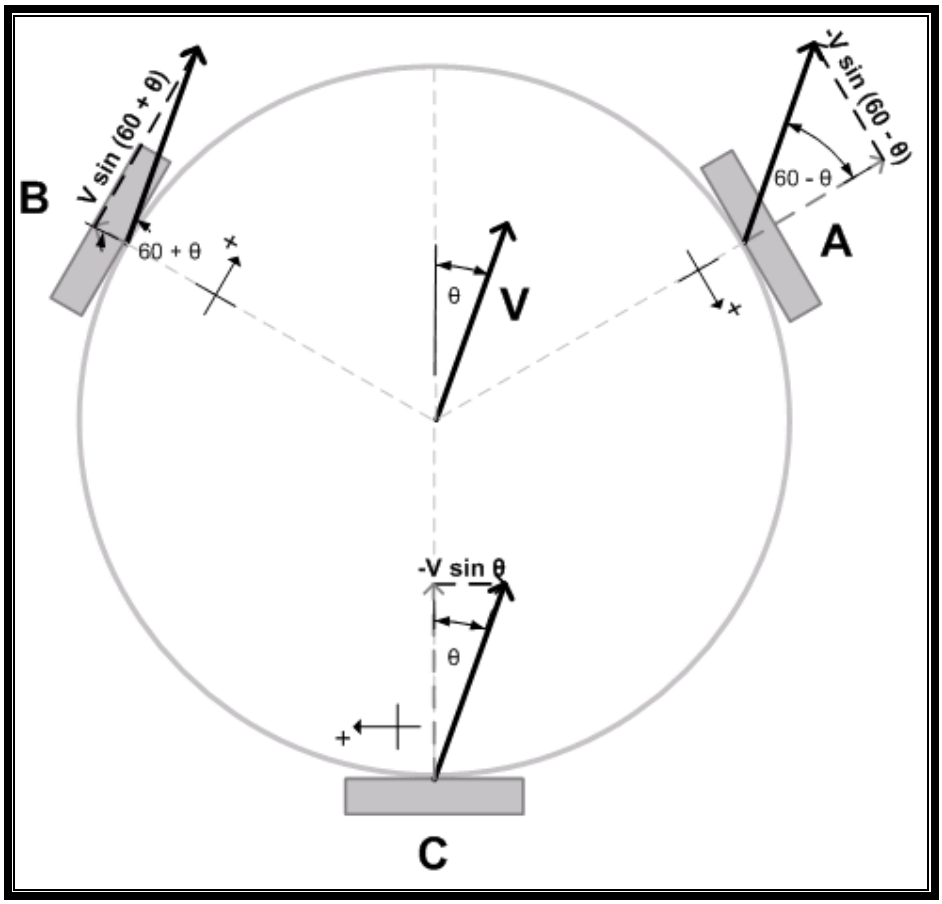


Figure E-3: Vector Diagram of Individual Wheel Speeds

E.3 Implementation

Once the kinematics had been derived, they could be implemented in code so that the robot could follow commands to drive or rotate in a certain direction at a certain speed.

The code had already been written to control the wheels at a particular speed (Appendix D). Now the robot had to be given a speed and direction and using the kinematic equations derived above, decide what speed to drive each wheel at. Figure E-5 is a flow chart of the code used to implement this functionality. First the user

enters a command using the serial port. The command is four bytes long. Figure E-4 shows the command structure. The first byte is the character ‘g’ which tells the robot which mode to be in. The subsequent bytes are speed, direction and rotation. The speed can vary between 0 and 20. Since the maximum value of each byte can be 255, and the direction needs to vary between 0º and 359º, the direction is divided by two, this reduces the resolution of the direction, but not significantly enough to be noticed. The rotation can vary between
-20 and 20, but since non signed integers are being used to transmit the bytes, 20 is added to the rotation so that -20 to 20 is represented by 0 to 40. The sign denotes the direction of rotation and the number the speed at which it should rotate.

g	speed	Direction/2	Rotation + 20

Figure E-4: packet structure of commands to control robot platform.

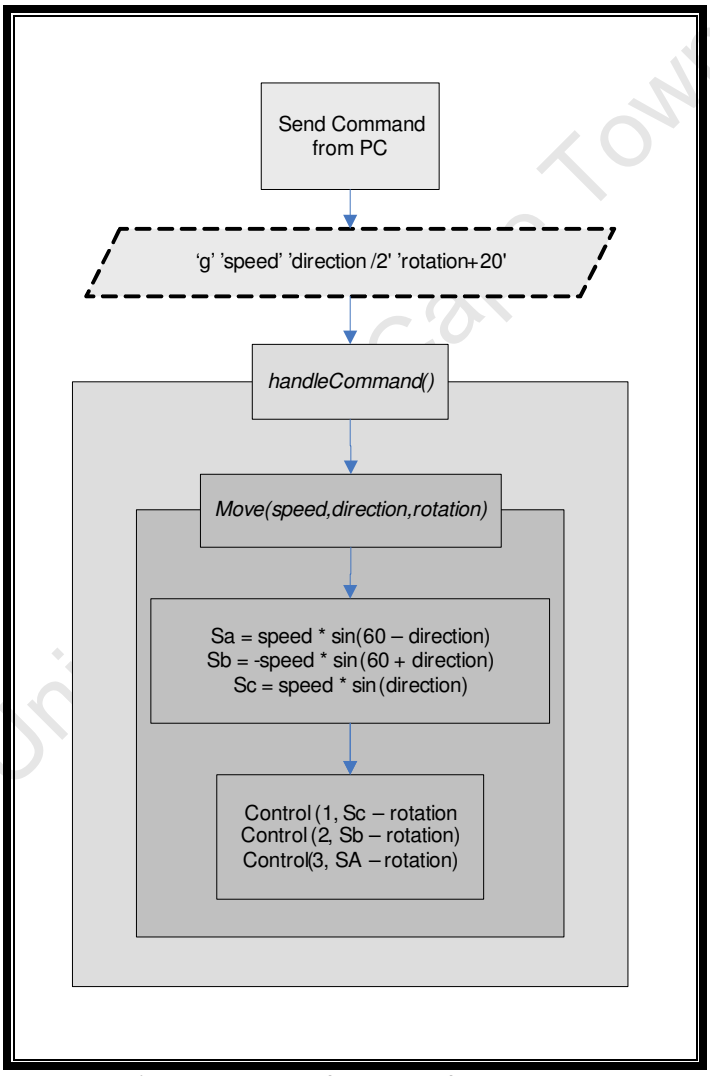


Figure E-5: Flow of robot platform control.

E.4 Joystick operation

To be able to control the robot platform in real time it was decided to implement joystick control. A Python program was written to interpret the commands from the joystick and convert the position and rotation of the joystick to serial commands which were sent to the robot via SCI. The Python program was adapted from code written by Eugene Dreyer[3]. Changes in the joystick's x and y position result in lateral movements of the robot, the direction in which the joystick moves denotes the robot's direction and how far the joystick moves from its central position determines the robot's speed. The maximum position of the joystick from the centre position corresponds with the robot's maximum speed.

E.5 Results

Testing showed that the robot responded as expected. The robot headed in the direction specified and the speeds varied as expected. From time to time the robot was seen to change direction from the expected direction, presumably due to slippage, and over longer distances (more than about 0.5m) the path was generally slightly curved rather than completely straight. Using dead reckoning, which cannot detect or correct for slippage or errors due to misalignment, these small errors are to be expected.

E.6 Conclusions

The simplified kinematic model seems to be sufficient to control this robot. The robot suffers from the small inaccuracies in movements that can be expected from any robot using dead reckoning for navigation. The robot is successfully able to move in any direction and is able to rotate on the spot and is therefore omnidirectional.

E.7 Recommendations

The robot was found to drive slightly off course from time to time. When driving longer distances tended to follow a slightly curved path. These errors were probably due to slippage and misalignment of the wheels. The following may be done to improve the accuracy of the robot.

E.1.1 Increase resolution of encoders

By increasing the resolution of the encoders, the control algorithm could more accurately measure and therefore control the speed of the wheels, making the robot follow the intended path more closely.

E.1.2 Decrease rate of acceleration

A prefilter could be included in the control algorithm, to decrease the rate of acceleration of the wheels and therefore decrease slippage. This was successfully done by Mr Mc Phillips on his soccer robots [4].

E.1.3 Improve wheel alignments

The robot kinematics are sensitive to the position of the wheels in relation to each other. Any improvements in the wheels aligning exactly at 120° to each other would result in more accurate movements by the robot.

E.1.4 Add additional movements sensors

Further improvements could be made by not only relying on dead reckoning for the navigation but by adding an element such as the computer mouse navigational system discussed in Section C.2.2 and C.2.3 or by feeding the robot its position and orientation as measured from the overhead camera in the robot soccer system. The robot could then change its direction and rotation dynamically in order to make up for errors in its path.

E.8 References

1. Kalmar-Nagy, T., R. D'Andrea, and P. Ganguly, *Near-Optimal Dynamic Trajectory Generation and Control of an Omnidirectional Vehicle.* Robotics and Autonomous Systems, 2004. **46**: p. 47-64.
2. Saha, S.K., J. Angeles, and J. Darcovich, *The Design of Kinematically Isotropic Rolling Robots with Omnidirectional Wheels.* Mech. Mach. Theory, 1995. **30**(8): p. 1127-1137.
3. Dreyer, E., Explorer Robot Platform, in Department of Mechanical Engineering. 2007, University of Cape Town.
4. McPhillips, G., The Control of Semi-Autonomous Robots, in Department of Electrical Engineering. 2004, University of Cape Town.

Appendix F. Advanced Robot Platform Control

Table of Contents

Table of Figures

Appendix F. Advanced Robot Platform Control

F.1 Introduction

The robot control already discussed is the basic platform control, that is, control which allows the robot to move in any direction or to rotate at a certain speed. This level of control does not allow the robot to accurately rotate while moving in a lateral direction. What follows is a description of how this higher level of control was implemented on the robot. This 'advanced control' allows the robot to accurately move from one position to the next while rotating to a given orientation.

F.2 Intended Movements

The intention of the advanced platform control is for the robot to move in a specific lateral direction while at the same time rotating at a specified speed. The basic platform controls only allow the robot to move laterally or rotate accurately, not do the two at the same time. Equations 5-6 from Section E-2 are specified as being accurate over short distances because, as the robot rotates, the new orientation means the lateral movement is in a new direction. Over long distances, if the robot travels at a constant lateral speed with some rotational speed, the robot will follow a curved path.

F.3 Breaking up the Movements

It was noted that over small distances, the basic control was effective in controlling the robot to drive laterally while turning. It was therefore decided to implement advanced control by breaking up the path of the robot into a number of small paths of

basic control, with the direction of lateral movement updated as the robot's orientation changed.

F.4 Simulation

To better understand the advanced robot motion, a simulation was done using Matlab. The simulation consists of a graphic plot showing the robot's movements and an analysis of what speeds would need to be applied to the wheels to achieve those movements.

Figure F-1 below shows the plot of the robot positions as the robot moves in a 45º direction, rotating clockwise at a speed of 1pulse/30ms. The robot starts at the bottom left of the figure. Each time interval, a new plot of the robot is shown. The robot is shown to have moved further along the 45º line of lateral motion and have rotated clockwise by a small amount. The figure shows the robot over 6 time intervals.

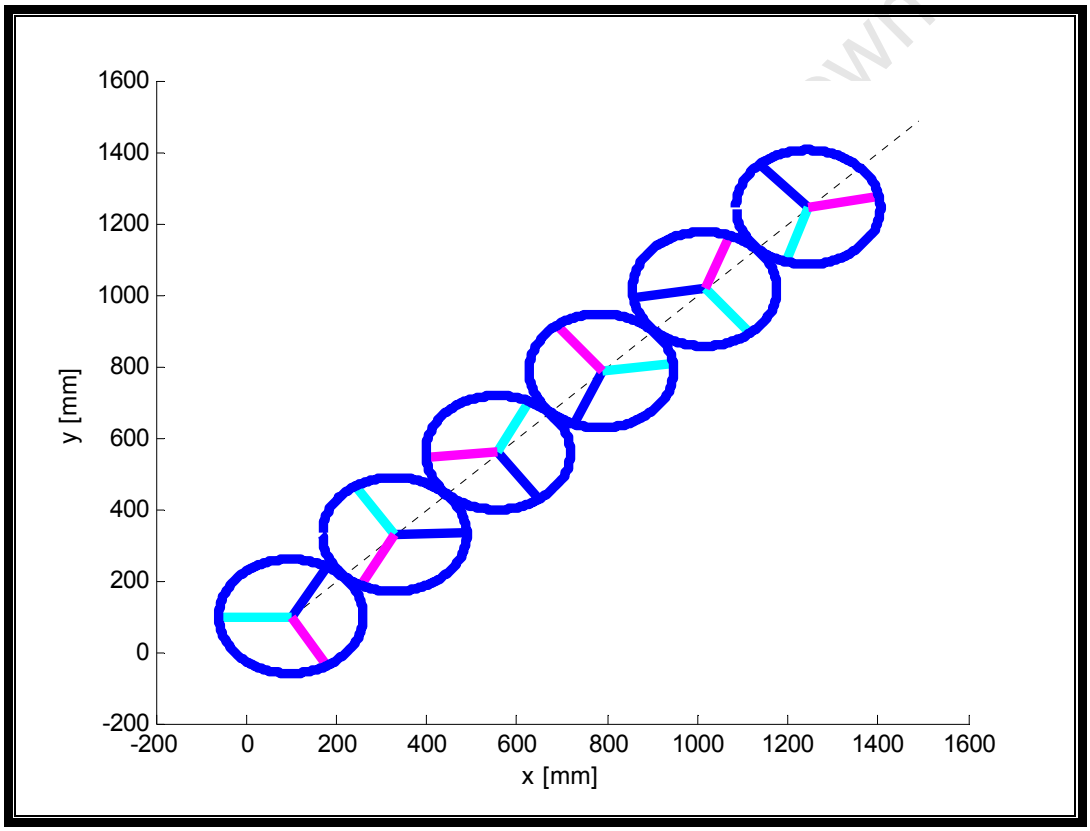


Figure F-1: Simulation of the robot moving laterally while rotating Lateral Velocity = 12, direction = 45°, speed of rotation = 2.

While the graphical simulation is run, the wheel speeds of the robot are calculated. Figure F-2, is a plot of the wheel speeds which would be needed to move the robot along the path in Figure F-1. A function was written to calculate the speed of each wheel needed to drive the robot in a given direction at some speed and to rotate it at a given speed. This function uses the basic platform control equations 5-6 from Section E-2. Since the robot is rotating while travelling, the robot's orientation changes each time interval. To plot the graphs of wheel speed, the wheel speeds are recalculated each time interval, using the new orientation of the robot. As can be seen from Figure F-2, the wheel speeds follow sinusoidal patterns, spaced 120º apart. The plot repeats

itself every time the robot rotates by 360º. The sinusoidal nature of the wheel speed function can be attributed to the equations of wheel speed being sinusoidal functions of the direction in which the robot is travelling. This direction (which is relative to the robot), changes with time as the robot rotates.

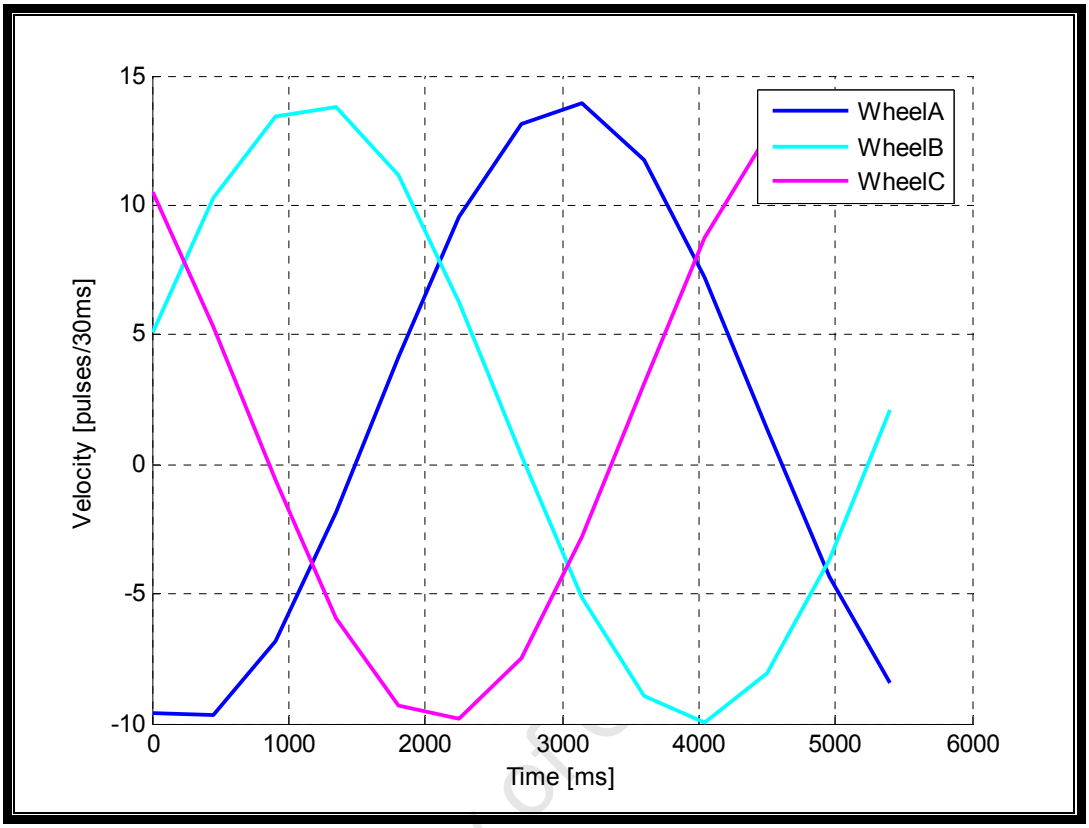


Figure F-2: Plots of changing wheels speeds plotted by the simulation. Lateral Velocity = 12, direction = 45°, speed of rotation = 2.

To further investigate the wheel speed functions, the simulation was repeated with varying directions, speeds of lateral motion and speeds of rotational motion. Figure F-3 on page F-4 investigates the effect of changing the direction of lateral movement of the robot. The graphs of the wheel speeds shift with respect to time. This is because the direction, θ, from equations 5-6 is now dependent on time. This can be expressed by replacing the θ term in the equations with (θ - Φ*t*).

Equations 5-6 become:

$$V_A = -V_{lat}\sin(60 - \theta - \phi t) + V_{rot} \quad [8]$$

$$V_B = V_{lat}\sin(60 + \theta - \phi t) + V_{rot} \quad [9]$$

$$V_C = -V_{lat}\sin(\theta - \phi t) + V_{rot} \quad [10]$$

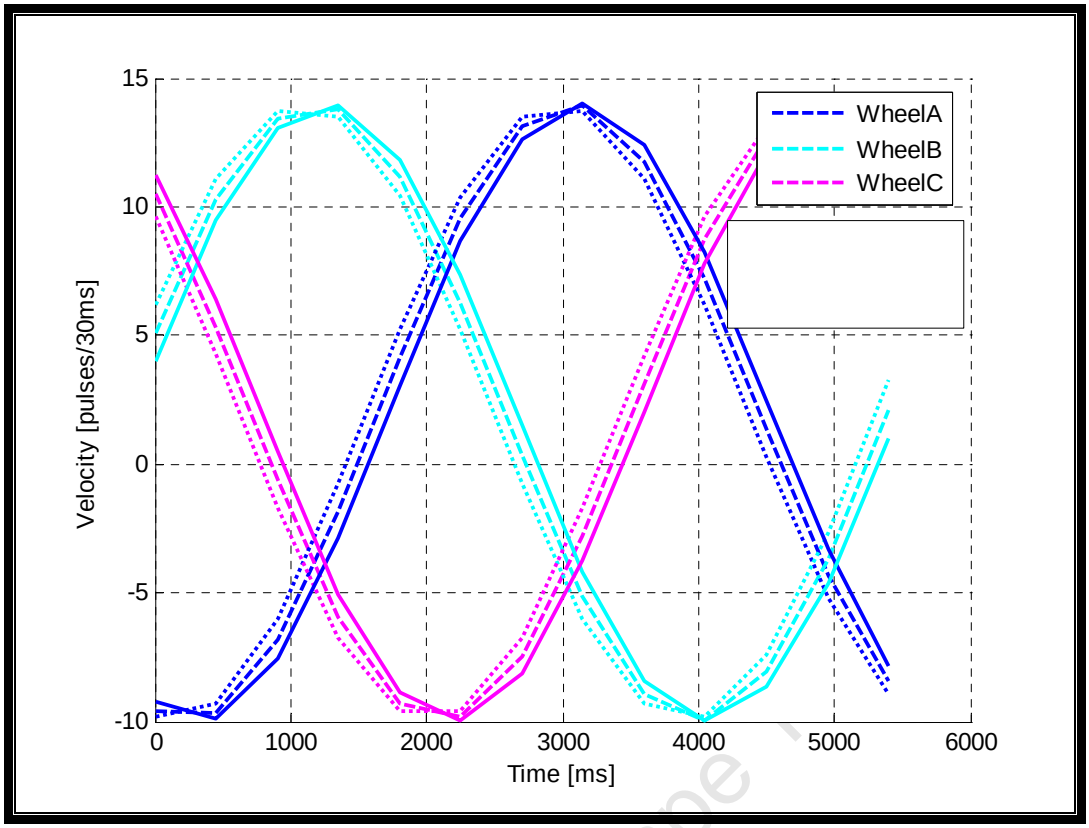


Figure F-3: Graph showing the effect on wheel speeds of changing the direction of lateral motion

Figure F-4 shows the effects on wheel speeds of changing the speed of the lateral motion of the robot. As the speed reduces, the amplitude of the graphs of wheel speeds reduces. This result corresponds with equations 8-9 since changing the term V_{lat} would adjust the amplitude of the functions.

Figure F-5 shows the effect of changing the rotational speed of the robot. The graph is for wheel A only. As predicted by equations 8-9, an increase in the rotational speed, shifts the graph up slightly. This is particularly apparent at t = 0. What is not predicted by equation 8-9, is that increasing the speed of rotation of the robot has the effect of increasing the frequency of the graphs of wheel speed. This identifies the fact that the faster the robot rotates, the sooner it will turn a full 360º, returning the robot to its original orientation. Equations 8-9 now needed to be updated to show the effect that V_{rot} has on the frequency of the graphs:

$$V_A = -V_{lat} \sin\left(kV_{rot}\left(60 - \theta - \phi t\right)\right) + V_{rot} \quad [11]$$

$$V_B = V_{lat} \sin\left(kV_{rot}\left(60 + \theta - \phi t\right)\right) + V_{rot} \quad [12]$$

$$V_C = -V_{lat} \sin\left(kV_{rot}\left(\theta - \phi t\right)\right) + V_{rot} \quad [13]$$

Where k and ϕ are arbitrary constants.

Section F.7 compares the simulated graphs to the wheel speeds recorded from the robot.

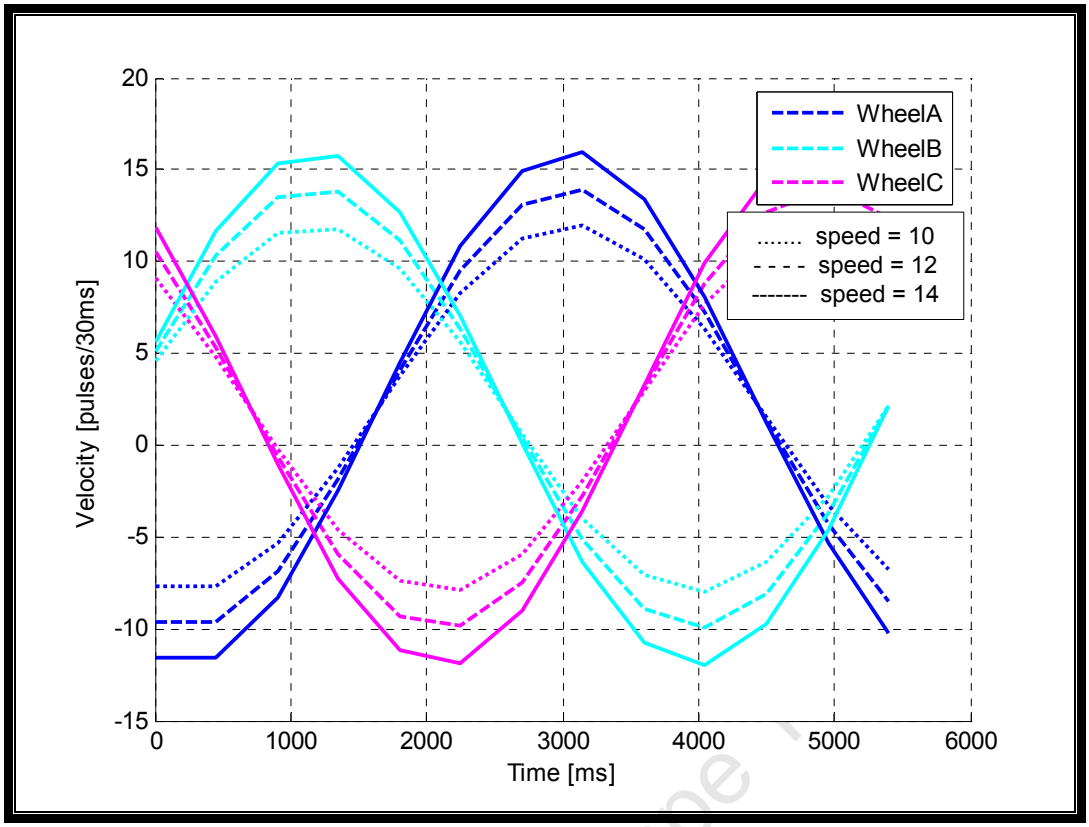


Figure F-4: Graph showing the effect on wheel speeds of changing the speed of lateral motion

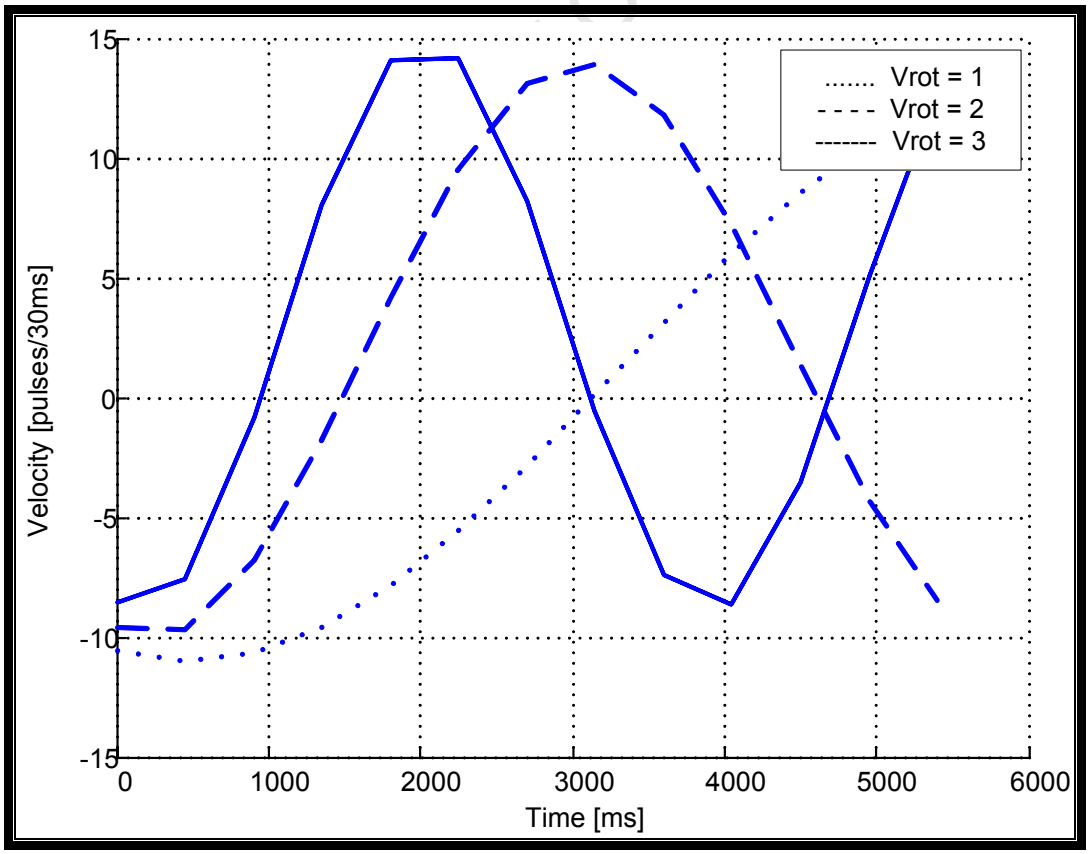


Figure F-5: Graph showing the effect on wheel speeds of changing the speed of rotational motion

F.5 Implementation on the Robot

The 'move' method had already been implemented on the robot. This allows the robot to move in a lateral position while rotating but over long distances results in curved motion. To implement lateral and rotational movement on the robot at the same time, the 'advance' function was written. This broke up the motion into a number of smaller motions. The robot moves a small distance, at a set velocity, while rotating. As the robot rotates, the intended direction of lateral movement changes relative to the robot. For example, if the robot is directed to move at a bearing of 0° while rotating clockwise, after a small movement the robot will be facing a bearing of 30°. For the robot to continue to move at a bearing of 0°, the direction of lateral motion must now be 330° relative to the robot. The 'advanced' command updates the direction of lateral motion every interval.

The amount that the robot must update its direction of lateral motion by after each interval, is relative to the speed of rotation of the robot. The faster the robot is rotating, the more its orientation will change by, and the greater the amount of correction needed. This means that every interval, the direction of lateral motion, θ, must be changed by some correction factor multiplied by the speed of rotation and the time of the interval, *ti*. This correction factor is here referred to as *Q*. The correction factor was found using experimentation.

Another factor to be determined is the length of time of the intervals into which the motion is divided. This time was also found using experimentation.

To drive the robot at a lateral speed L and a rotational speed R in a direction D, the following pseudocode algorithm would be followed:

Instruction is received
1. Drive robot (Lateral speed = L, Rotational speed =R, Direction = D)
2. Wait *ti* seconds
3. $D = D - Q * R * ti$

Repeat 1-3 until new instruction received

The length of the time intervals into which the motion is broken affects the accuracy of the motion. A longer time interval makes the motion less smooth, approaching a 'zig-zagging' pattern rather than straight line motion. However, decreasing the time interval, decreases the time which the control algorithm has to reach the specified speed. At the extreme, should the length of the time interval be as small as the length of the control algorithm's time step, the control algorithm would no longer have time to react to the changes in intended motion and the system would become ineffective. The optimal time interval is a compromise between these two factors.

The time interval *ti* was measured in 30ms units (here referred to as time steps) since the DSP code measures time in units of 30ms (the time step of the control algorithm). Initial testing was done at a speed of 10 pulses per 30ms. At this speed a distance of about 300mm is covered in 30 time steps. 30 time steps was chosen as the first time interval length. The code implemented 8 of these intervals at a speed of 10 pulses/30ms and a rotational velocity of 5 pulses/30ms in a clockwise direction. The direction of motion was chosen as 0°. The original correction factor was chosen as 1.

The robot drove and rotated, but the overall line of motion was a curve to the left. The speed of rotation was reduced to 1 pulse per 30 ms. The robot still curved to the left considerably. This curve to the left indicated that the robot was overcompensating for the clockwise rotation so *Q* was reduced to 0.8. This yielded far more satisfactory results - the line of linear motion was much straighter. The robot was then tested in a number of directions and at a number of speeds of rotation. The overall effect was good, but it was found that the robot was still overcompensating for the rotations so the *Q* was again reduced, this time to 0.7. This resulted in satisfactory motion with the robot moving in a straight line while rotating. The motion was not smooth however so the next value to be adjusted was the length of the time interval *ti*. Video footage of the robot run at various *ti* values can be found on the accompanying CD, in the Appendix E folder

Code was written to track the speeds of the wheels as the robot moved. Figure F-6 below shows the changing wheel speeds when the robot is moving at a speed of 12 pulses/30ms, rotating at a speed of 1 pulse/30ms when the time interval *ti* is 30. As predicted by the simulations, the wheel speeds follow sin wave patterns. The intervals when the wheels are at constant speed show as flat sections with small variations. The intervals of constant speed were also very clear when the robot was run.

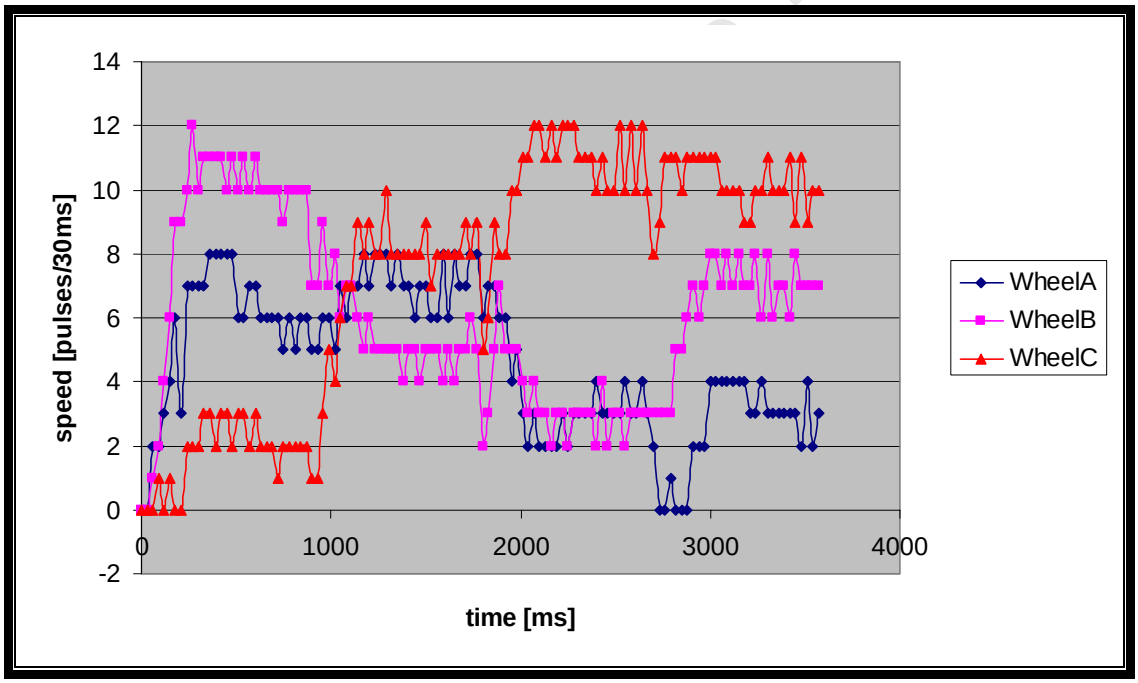


Figure F-6: Wheel speeds as robot rotates and moves laterally. Time interval = 30

The time interval was then halved, to 15. The robot ran in a much smoother motion and this can be seen from the plot of the wheel speeds in Figure F-7 as well as from the video footage. In the figure, the sections of constant speed are smaller and the plots resemble a sin wave more closely. Figure F-8, Figure F-9 and Figure F-10 show the wheel speeds and the value of ti was reduced to 10, 5 and 2. The motion became increasingly oscillatory as can be seen from large spikes in the plots. This instability could be heard as a 'ticking' sound from the motors and the robot became slower and less accurate as the value of ti was reduced. This oscillatory behaviour is because the rate at which the setpoint changes becomes faster than the control algorithm can track accurately. It was therefore decided that the optimal value of the time interval *ti* was

15. This value was used for all future testing. To be able to reduce the value of *ti* further, the control algorithm would have to be faster. The speed of the control algorithm is determined by it's time step which relies on how often the wheel speeds can be updates. The update speed of the wheel speeds is determined by the resolution of the speed encoders. Therefor to decrease *ti* to get smaller motion, the resolution of the wheel speed encoders would have to be increased.

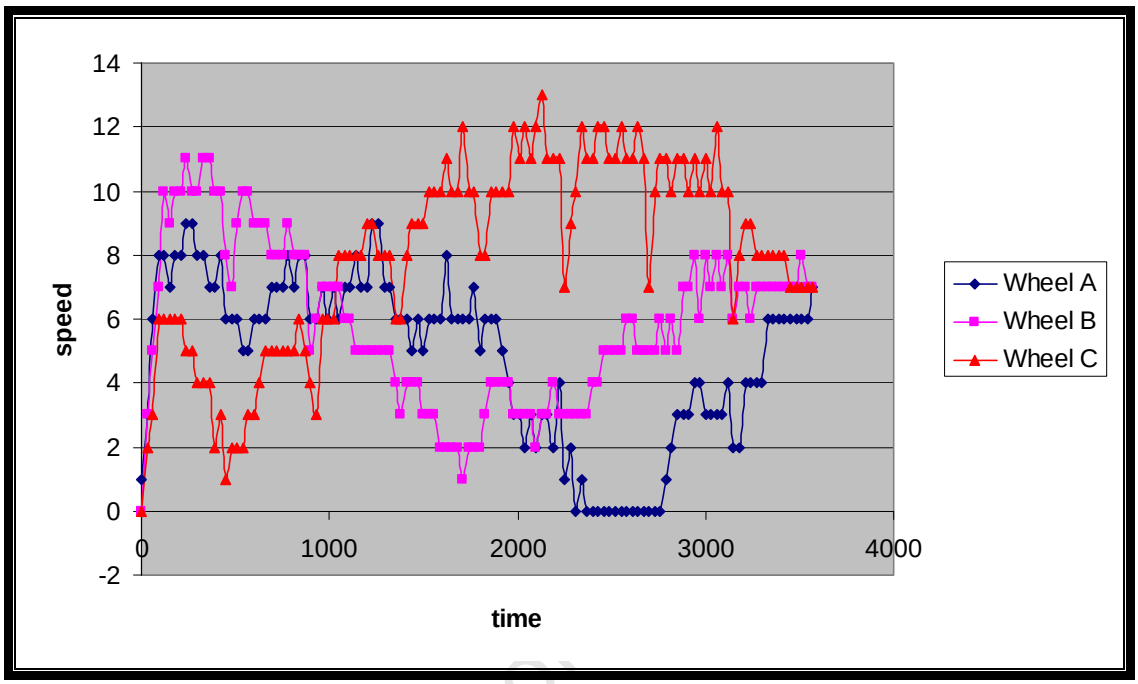


Figure F-7: Wheel speeds as robot rotates and moves laterally. Time interval = 15

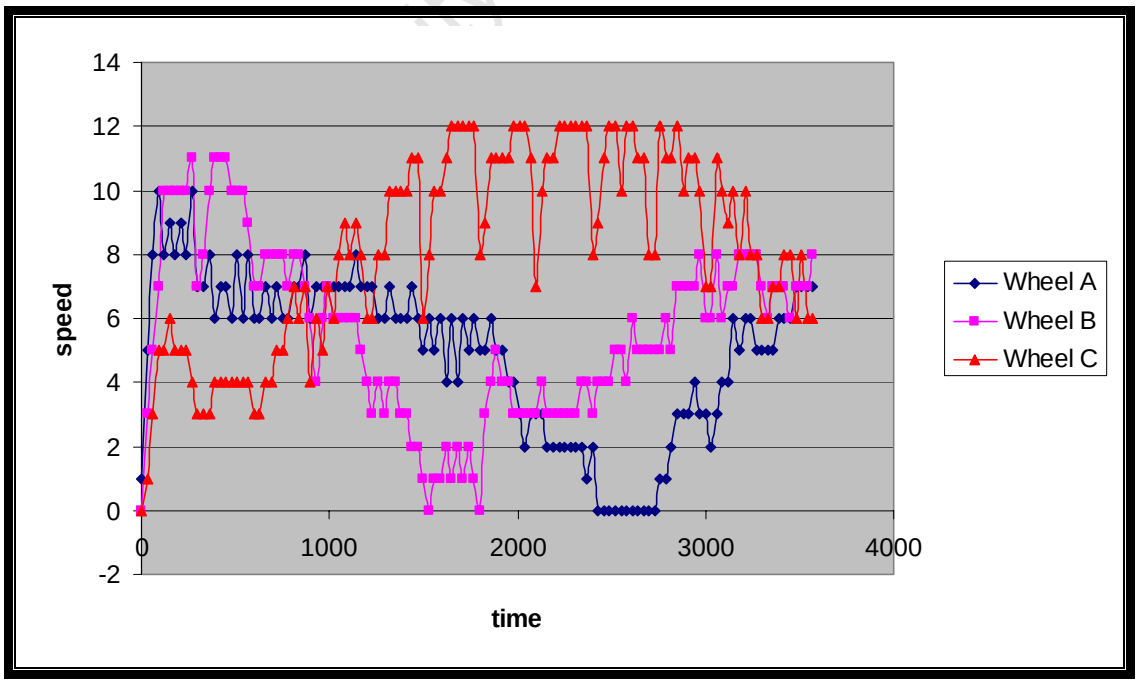


Figure F-8: Wheel speeds as robot rotates and moves laterally. Time interval = 10

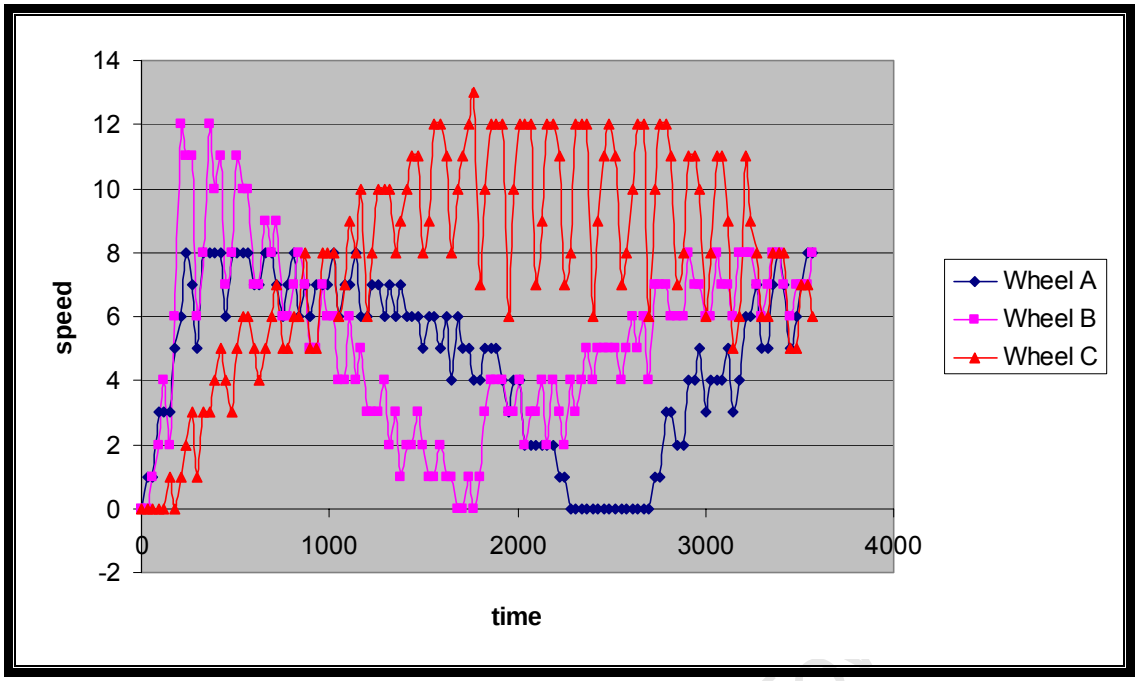


Figure F-9: Wheel speeds as robot rotates and moves laterally. Time interval = 5

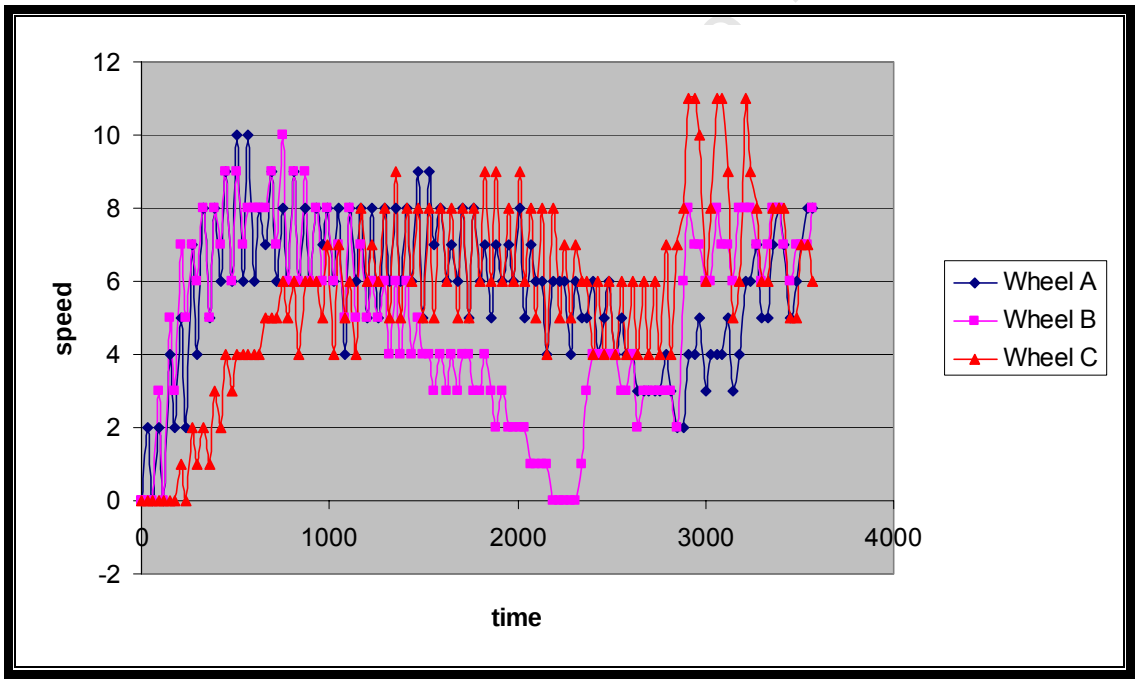


Figure F-10: Figure E 9: Wheel speeds as robot rotates and moves laterally. Time interval = 2

F.6 Testing

Tests were done to verify that the robot traveled straight during advanced motion. Lateral velocity, direction of motion and speed of rotation were varied to determine the range in which the control was accurate. The robot was found to behave satisfactorily in all directions, at rotational speeds below 6 pulses/30ms and lateral speeds above 12 pulses/30ms. It is unlikely that rotational speeds above 6 pulses/30ms will be needed in robot soccer competition. The lateral speed is limited to the robot's top speed range of 12-14pulses/30ms. Since the robot will be needed to operate at top speed during tournaments, this is an acceptable range for the robot to be able to perform advanced motion in. At lower lateral speeds, the compensation from *Q* is too large and the robot begins to drive in a curve. Should it be necessary to have advanced motion at lower speeds, further tests would have to be done to determine the relationship between *Q* and the lateral speed.

F.7 Comparison between simulation and actual system

To verify the results of the simulations documented in Section F.4, similar tests were conducted on the robot. The 'k' command was used to run the robot at specified speeds with a specified rotation and return a record of the wheel speeds during the motion.

Figure F-11 shows wheel speed data for wheel A with the robot run in three different directions. The wheel speed increases from t=0 to about t=2000 and then starts to decrease. As the direction of lateral motion increases by 10 degrees, the graphs shift to the right. Figure F-12 shows simulated data for all three wheels under the same conditions. The plot of wheel A increases with time (longer simulations prove this to follow a sinusoidal function) in a similar manner to the test data. As already noted, the simulated graphs shift with respect to time as the direction of lateral motion increases. The test data therefore verifies the earlier conclusion that changing the direction of motion affects the phase shift of the graphs of motion.

Figure F-13 on page F-12 shows the results of tests in which the robot was run at different lateral speeds. The amplitude of the graphs increase as the lateral speed increases. Figure F-14 shows the simulated data for all three wheels under similar conditions. Wheel speed A again increases with time. The amplitude of the graphs increases as the lateral speed increases. The test data therefore verifies the earlier conclusion that changing the direction speed of lateral motion affects the amplitude of the graphs of motion.

Figure F-15 on page F-13 shows the results of test in which the robot was run at different rotational speeds. When the rotational speed is 3, the graph drops at t = 2000, showing that the frequency of the graph has increased. Figure F-16 shows the simulated data for wheel A under similar conditions, here it is clearer that the frequency increases with increasing rotational velocity. Figure F-15 does not show a shift in the graph with respect to speed, further tests would need to be done to confirm this prediction from the simulation. Figure F-15 is inconclusive but does suggest that

the periods of the graphs are decreasing as the rotational velocity increases as predicted by the simulated data.

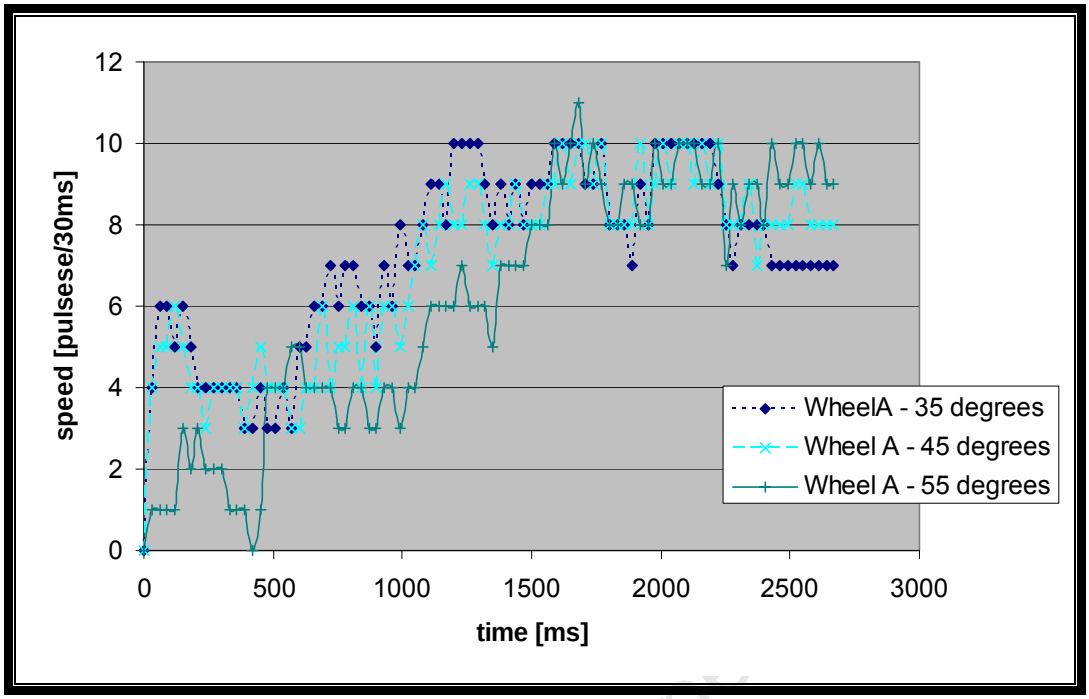


Figure F-11: Wheel speed data from the robot run in 3 different directions at a lateral speed of 12 and a rotational speed of 2

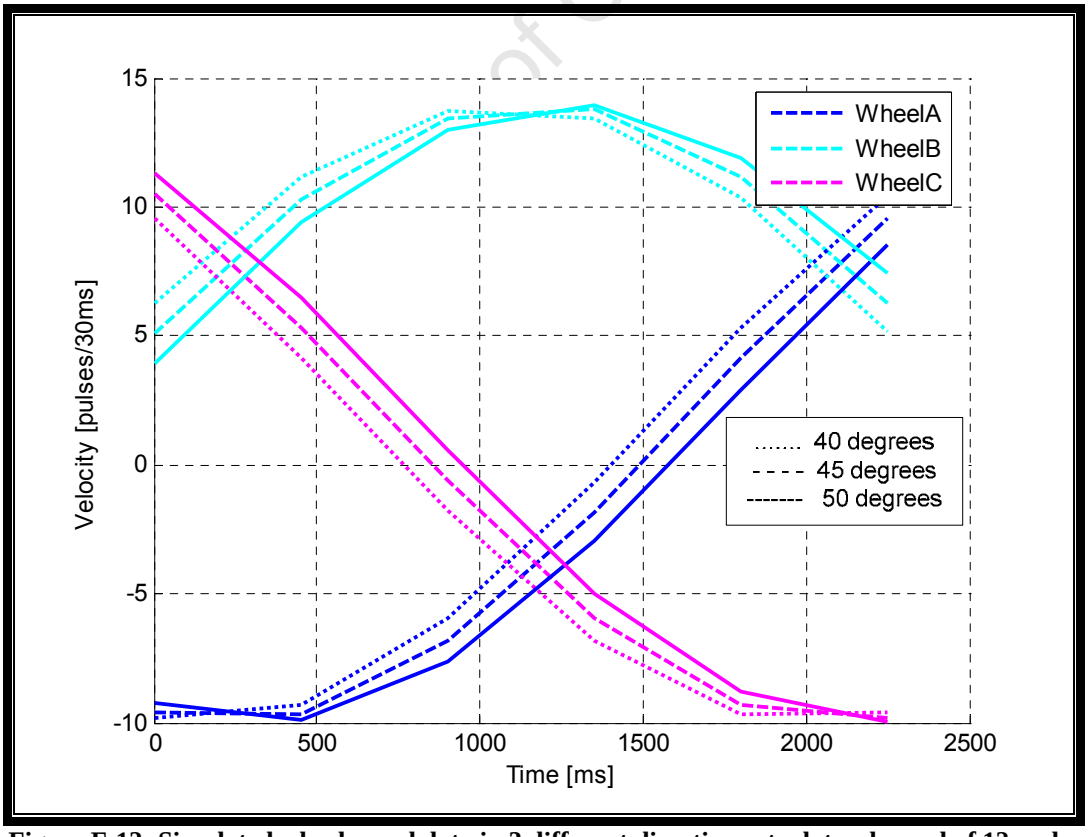


Figure F-12: Simulated wheel speed data in 3 different directions at a lateral speed of 12 and a rotational speed of 2

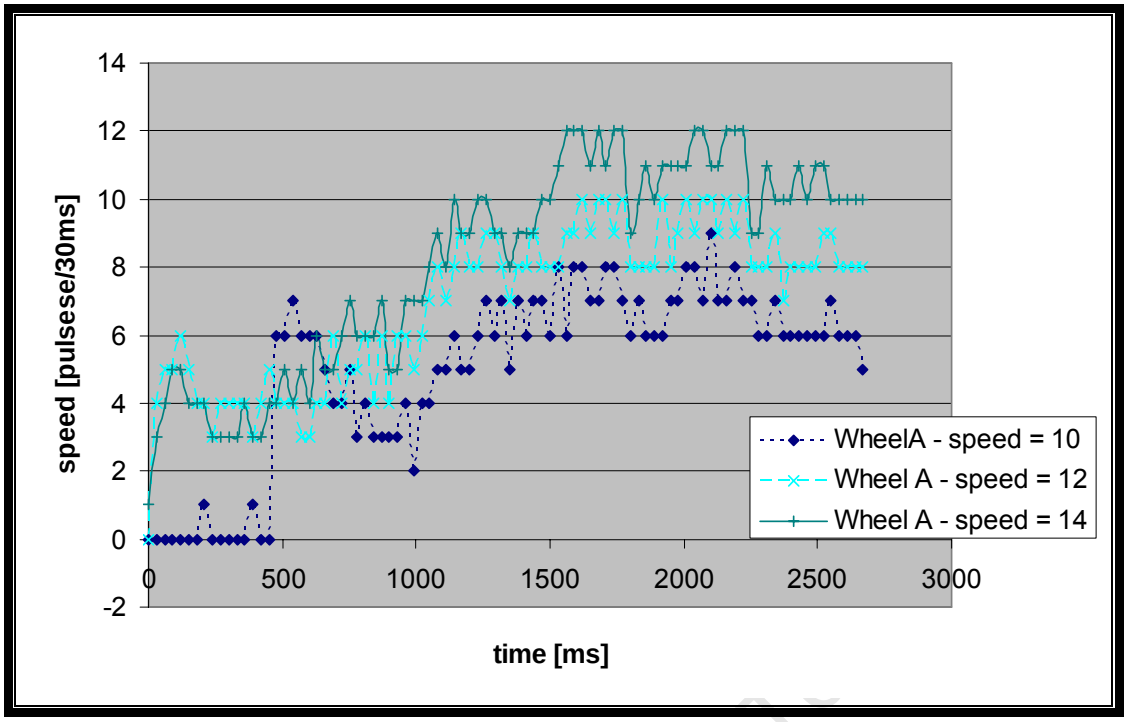


Figure F-13: Wheel speed data from the robot run at 3 different lateral speeds in the direction 45º with a rotational speed of 2

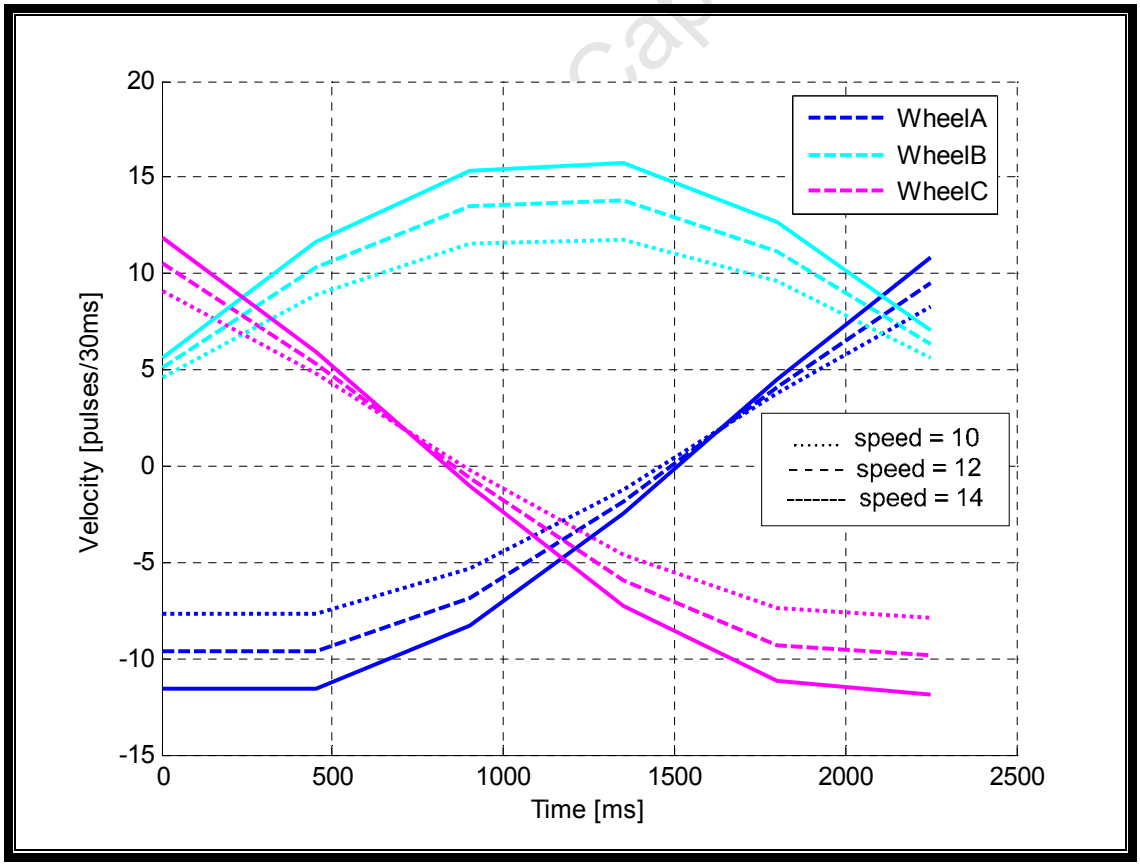


Figure F-14: Simulated speed data from the robot run at 3 different lateral speeds in the direction 45º with a rotational speed of 2

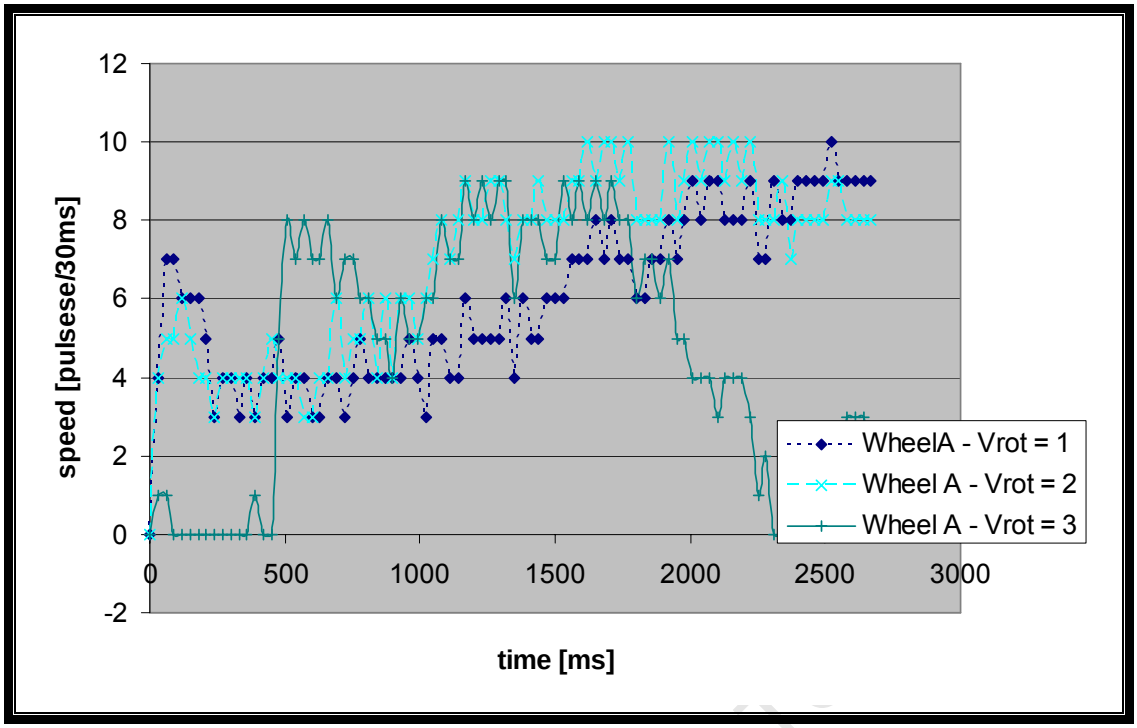


Figure F-15: Wheel speed data from the robot run at 3 different rotational speeds in the direction 45º with a lateral speed of 12

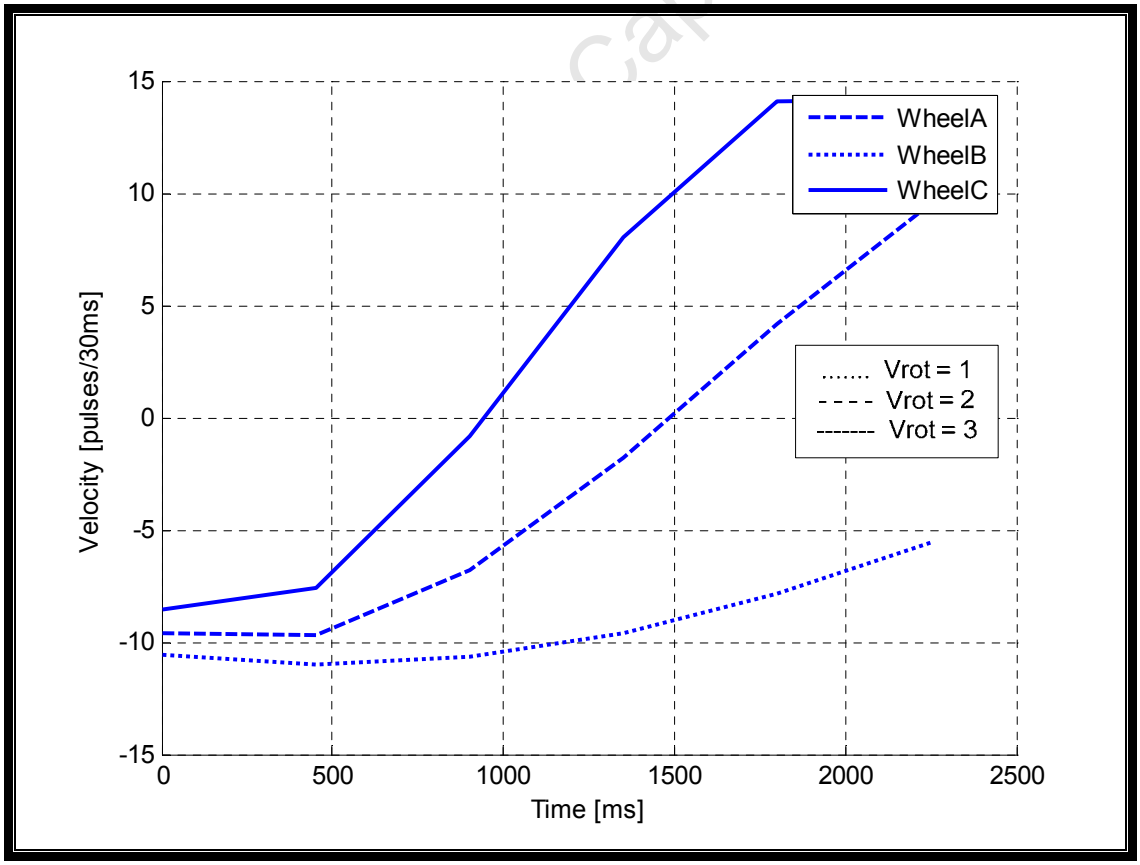


Figure F-16: Simulated speed data from the robot run at 3 different rotational speeds in the direction 45º with a lateral speed of 12

F.8 Conclusions

Advanced motion was successfully implemented on the robot at speeds above 12 pulses/30ms. The robot is able to move in a straight line in any lateral direction while rotating.

Simulation of the robot as well as verification through testing on the robot system determined the effects of changing lateral speed, direction of motion and rotational speed. The simulations determined that when rotating while driving laterally the wheel speeds of the robot follows a sinusoidal pattern. The phase shift of the graph is determined by the lateral speed of the robot. The amplitude of the graph is determined by the speed of the lateral motion. The frequency of the graph is dependent on the speed of rotational motion.

The compensation factor *Q* determines the relationship between the rotational speed of the robot and the amount by which the angle of motion is updated each time interval. This value was determined for large lateral speeds by trial and error. Should advanced motion at lower speeds prove necessary, the *Q* for lower speeds will have to be determined.

The length of the time interval *ti* determines how smooth the motion of the robot is. An optimal value of ti was found. Should smoother motion be desired, the value of the time constant of the control algorithm would need to be reduced. To do this the resolution of the robot's shaft encoders would have to be increased.

Note that the maximum speed that the robot can move at in advanced motion is determined by the sum of the lateral and rotational speeds. If for example the robot is moving at some lateral speed *L* and a rotational speed *R*, the sum of *L* and *R* needs to be less than the maximum speed of the robot wheels. This is because at some point in the robot's motion, the wheel speed due to lateral movement of one of the wheels will need to be equal to *L*. The speed of rotation, *R*, will be added to this value, resulting in a wheel speed of *L*+*R*. To avoid inaccuracies this value should not exceed the maximum allowable wheel speed.

Appendix G. Code Description

Table of Contents

Table of Figures

Tables

Appendix G. Code Description

G.1 Introduction

This appendix describes the C code written for the Freescale DSP56F8323 to control and interact with the robot which has not been discussed in other sections of this thesis.

The main method and general structure of the code are described here. Also discussed is the command structure written to allow commands to be sent to the robot via serial communication (SCI). The code also enables the robot to respond to these commands and carry out their instructions.

The code is interrupt based. The DSP performs all functions from a main loop which checks a number of flags. These flags are set when relevant interrupts are called. When a flag is set, the main loop calls the relevant procedure. Below is a description of the main loop, the interrupts and each procedure that can be called by the main loop.

G.2 Compiler

The code was written in Metrowerks Code Warrior for DSP56F8300 version 6.1 [1]. Supporting code for the DSP was generated using the Processor Expert application within the Code Warrior environment. Code Warrior was also used for programming the DSP.

G.3 Files

The following files hold the code needed to run the robot:

DriveRobot.C Contains the *main()* method

DriveRobot.H H file to support *DriveRobot.c*

RobotUtils.C Contains all auxiliary methods

RobotUtils.H H file to support *RobotUtils.C*

Events.C Contains methods executed when interrupts are called

Init.C Contains initialisation methods for all DSP peripherals

Inits.h H file to support *Inits.C*

PCB1.h Contains definitions to be used when code is compiled for prototype board

DevBoard.h Contains definitions to be used when code is compiled for PCB

The files in the folder *generated files* were generated by Metroworks Processor Expert and provide background functionality.

G.4 Variable Types

In the h file *robotUtils.h* the variable types are declared as u8, u16, u32, s8, s16 and s32. The 'u' or 's' denotes if the variable type is unsigned or signed. The number denotes the size of the variable in bits. Since the DSP56F8323 is a 16 bit processor most variables used were 16 bits long. The same variable names are used within this document. 32 bit floating point numbers are denoted by the variable type float.

G.5 Main Loop

The main loop is the procedure main() in the file driveRobot.C. This procedure first initialises all global variables. Then it calls the following procedures to initialise peripherals:

GPIOInit()	– Initialises IO ports A and B as outputs and PWM lines
ADCInit()	- Initialises analogue to digital converter to run at 20Khz using 3 channels all with high level trigger interrupts enabled
PWMInit()	– Sets up PWM to use 3 channels at about 16Khz
SCIInit()	– Sets up SCI module 0. Baud rate is 19200
TimerInit()	– Initialises timer to overflow every 30ms

Next, the motors are enabled (by pulling high their enable lines) and the first analogue to digital conversion is started.
The program then enters an endless loop which checks the flags *commandBufferFull* (for handling commands via SCI) and *test* (for step tests).

To look at the code overall, one can look at the code functionally. The code can be broken into the following functions:

- Communications – discussed below
- Speed measurement – discussed in Appendix C

- Speed control – discussed in Appendix D
- Robot platform control – discussed in Appendix E
- Step Tests – discussed below and in Appendix D

G.6 Communications

Communication with the robot happens via the DSP's SCI peripheral. The robot communicates with the PC via serial cable. A serial port terminal program on the PC provides the user interface. Attempts to replace the serial cable with a radio communications module were unsuccessful.

G.6.1 Transmit

Code was written so that the robot can send characters, strings, Hexadecimal digits, decimal digits and floating point numbers. The relevant functions are:

- send(u16)
- SCIprintf(u8*)
- SCIprintfDec(s32)
- SCIprintfHex(u16)
- SCIprintfFloat(float)

SCIprintfDec and SCIprintfFloat can print out both positive and negative numbers, SCIprintfHex only handles positive numbers. These functions were used for debugging during development to send to the console information about what state the DSP was in as well as to return the results of calculations. In the final version of the robot, the transmit functions are used in the code for performing step tests.

G.6.2 Receive

Code was written to enable the robot to receive commands. This code centres on the method *handleCommand()* in the file robotUtils.c. When a character is sent to the robot from the PC, the receive interrupt (*INT_SCI0_RxFull()*) is called. All commands and inputs must be in the form of 4 characters. When a new character is received, the character is added to a buffer. Should the buffer be full, i.e. there are 4 characters in the buffer, the flag *commandBufferFull* is set to true. The code then returns from the interrupt. The main loop, seeing that this flag has been set, calls the method *handleCommand()*. The robot can accept the commands shown in

Table G-1 on page G-5. *handleCommand()* consists of a case statement which executes the relevant code depending on the contents of the command buffer.

The move commands (M000) and (K000) as well as the closed loop step test command (T500), ask for extra user input. This is done using the flag *waitingForInput*. While this variable remains true the code remains in an empty while loop. Now, in the receive interrupt, once the commandbuffer is full, because *waitingForInput* is true, instead of setting the flag *commandBufferFull*, the interrupt moves the contents of the buffer to the variable *inputValue* and sets *waitingForInput* to false. This flags the main code to exit the empty while loop so that the *inputValue* can be used. Figure G-1 below shows the m00 command being used and the robot then asking for user input. The blue text is commands and input from the user, the red text is the prompts from the robot.

```
m000
Enter speed to move at (2-20):
0012
Enter direction to move in (0-360):
0075
The rotation is 0.

Test complete
                    Speed Values Motor C
000000
000006
...
000012
000010
000012
Speed Values Motor B
000000
000001
000004
...
000008
000008
000008
Speed Values Motor A
000000
000001
000000
```

Figure G-1: Screen shot of 'm' command receiving user input

For further information on the execution of the step tests see Appendix D.

Table G-1: Robot Commands

Command	Function	Constraints
A*** or a***	Drive wheel A with PWM = ***	*** is a three digit hexadecimal number from 000 to A00
B*** or b***	Drive wheel B with PWM = ***	*** is a three digit hexadecimal number from 000 to A00
C*** or c***	Drive wheel C with PWM = ***	*** is a three digit hexadecimal number from 000 to A00
D*** or d***	Drive wheel C with PWM = *** Drive wheel A with PWM = A00 - *** Robot drives in direction 1	*** is a three digit hexadecimal number from 000 to A00
E*** or e***	Drive wheel A with PWM = *** Drive wheel C with PWM = A00 - *** Robot drives in direction 2	*** is a three digit hexadecimal number from 000 to A00
F*** or f***	Drive wheel C with PWM = *** Drive wheel B with PWM = A00 - *** Robot drives in direction 3	*** is a three digit hexadecimal number from 000 to A00
M000 or m000	Move robot in desired direction at desired speed	Asks for further user input
‘g’’speed’’direction’’rotation’	Move in desired speed, direction and rotation.	See Appendix D.3
H000 or h000	Move at lateral speed 12, in direction 0º while rotating at speed 2. Wheel speeds are logged	
K000 or k000	Move in desired direction at desired speed while rotating Wheel speeds are logged	Asks for further user input
‘l’’speed’’direction’’rotation’	Move in desired direction at desired speed while rotating	See Appendix D.3
T100	Run open loop step test – 100% Vmax	
T200	Run open loop step test – 75% Vmax	
T300	Run open loop step test – 50% Vmax	
T500	Closed loop step test	Asks for further user input
X*** or x***	Control wheel A at speed ***	*** is a three digit decimal number from 000 to 014
Y*** or y***	Control wheel B at speed ***	*** is a three digit decimal number from 000 to 014
Z*** or z***	Control wheel C at speed ***	*** is a three digit decimal number from 000 to 014

G.7 Compiling for Different Boards

The same code is written to be able to be compiled for either the development board prototype robot or the final PCB robot. However the two board make use of different pwm and analogue input lines. To enable the same code to be used for both boards two H files were written, *devBoard.h* and *PCB1.h.* These two files define all parameters that are unique to each board. For example, in the pwm initialisation, the development board used pwm channels 1, 3 and 5 while the development board used channels 1,3 and 4. This means that the pwm control register PWMA_PMCCR which determines which pwm channels are enabled must be set up differently depending on which board the code is being compiled for. Defining either DEVBOARD or PCB1 in the file *RobotUtils.h* determines which h file is included when the code is compiled.

G.8 References

1. Freescale, *Metrowerks homepage.* www.metrowerks.com accessed Feb 2008.

Appendix H. Schematics

Table of Contents

Table of Figures

Appendix H. Schematics

H.1 Introduction

The figures which follow are the schematics of the PCB designed to control the robot platform. The Eagle files of the schematics, as well as the printed circuit board files are on the CD which accompanies this dissertation. Two versions of the boards exist; version 1.0 and version 1.1. They share the same schematics and differ only on their board layout. Version 1.0 is the board that was printed and populated and is described in Appendix B. Version 1.1 is an updated version of the board with necessary modifications made to it. It is recommended that future boards manufactured use board layout 1.1.

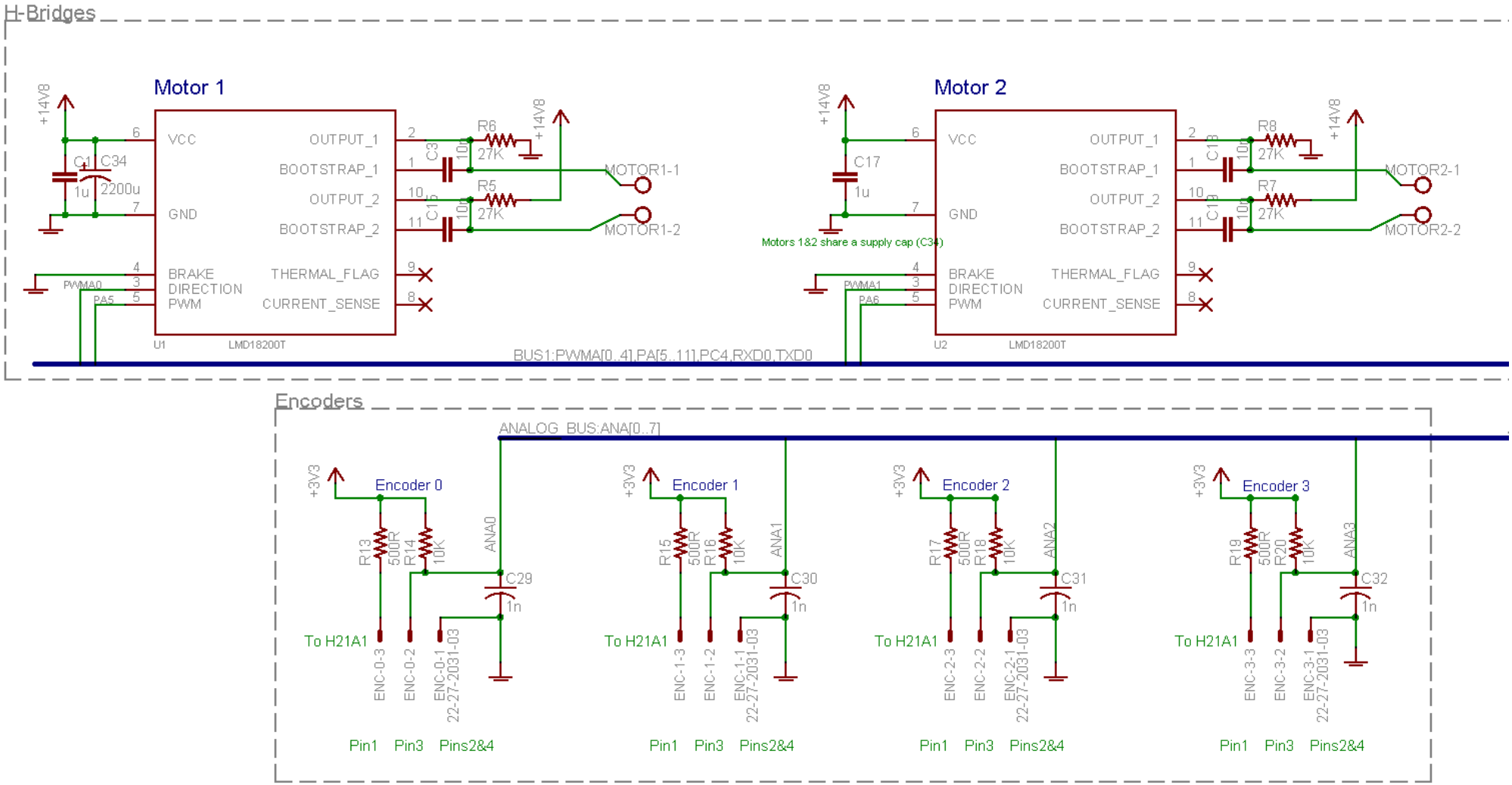


Figure H-1: H-bridge and encoder circuits

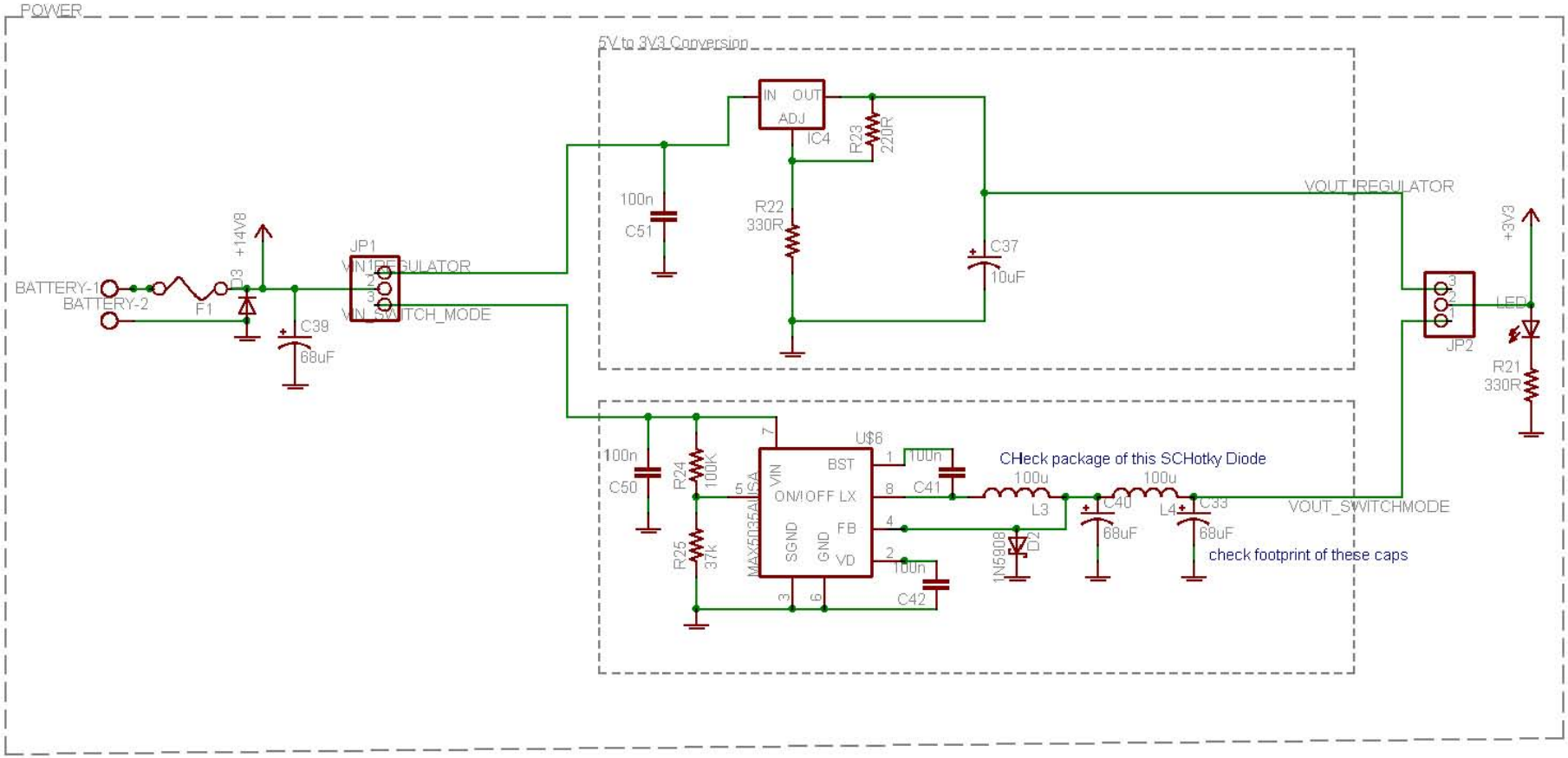


Figure H-2: Power regulation circuitry

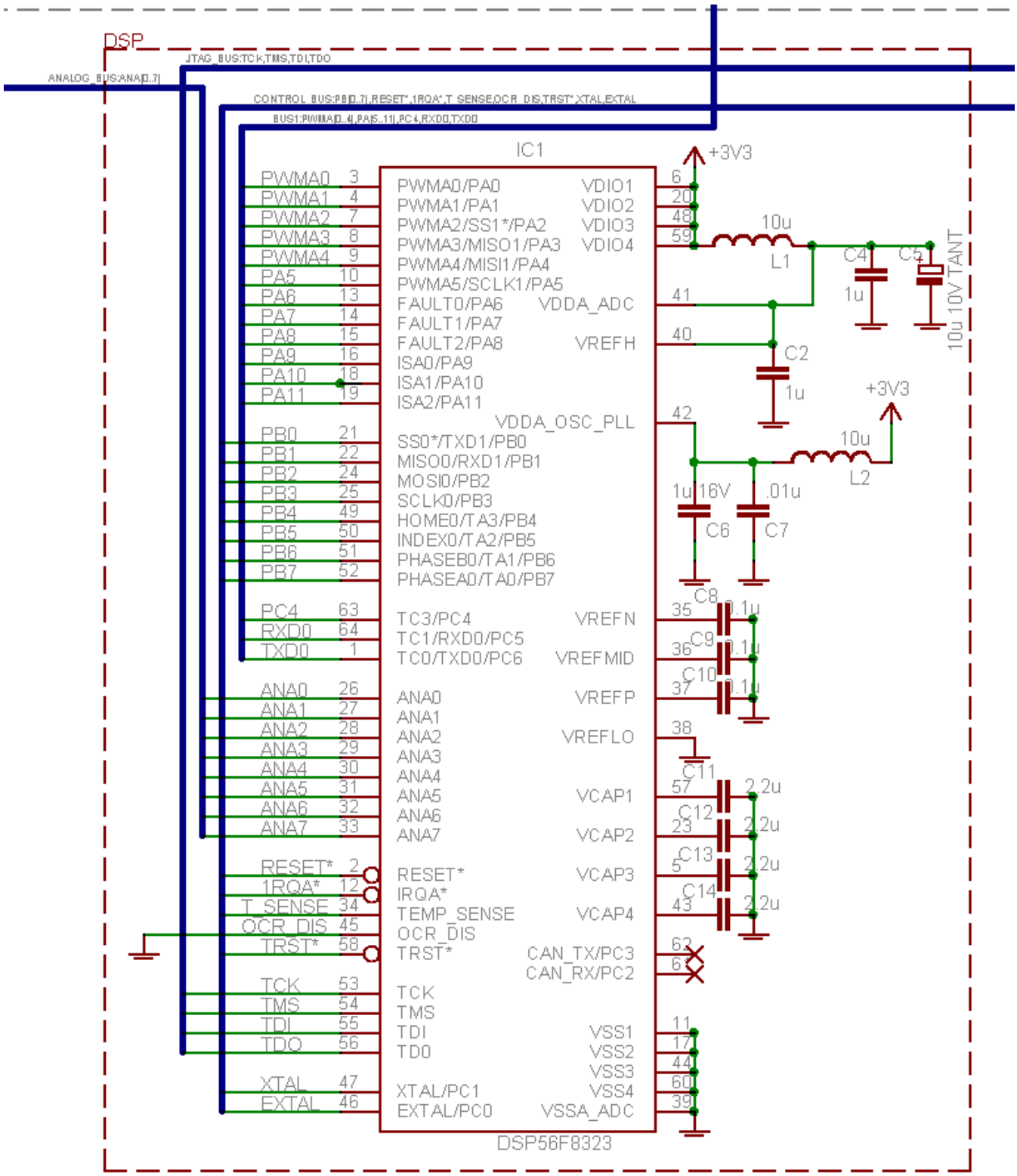


Figure H-3: DSP and DSP peripheral circuitry

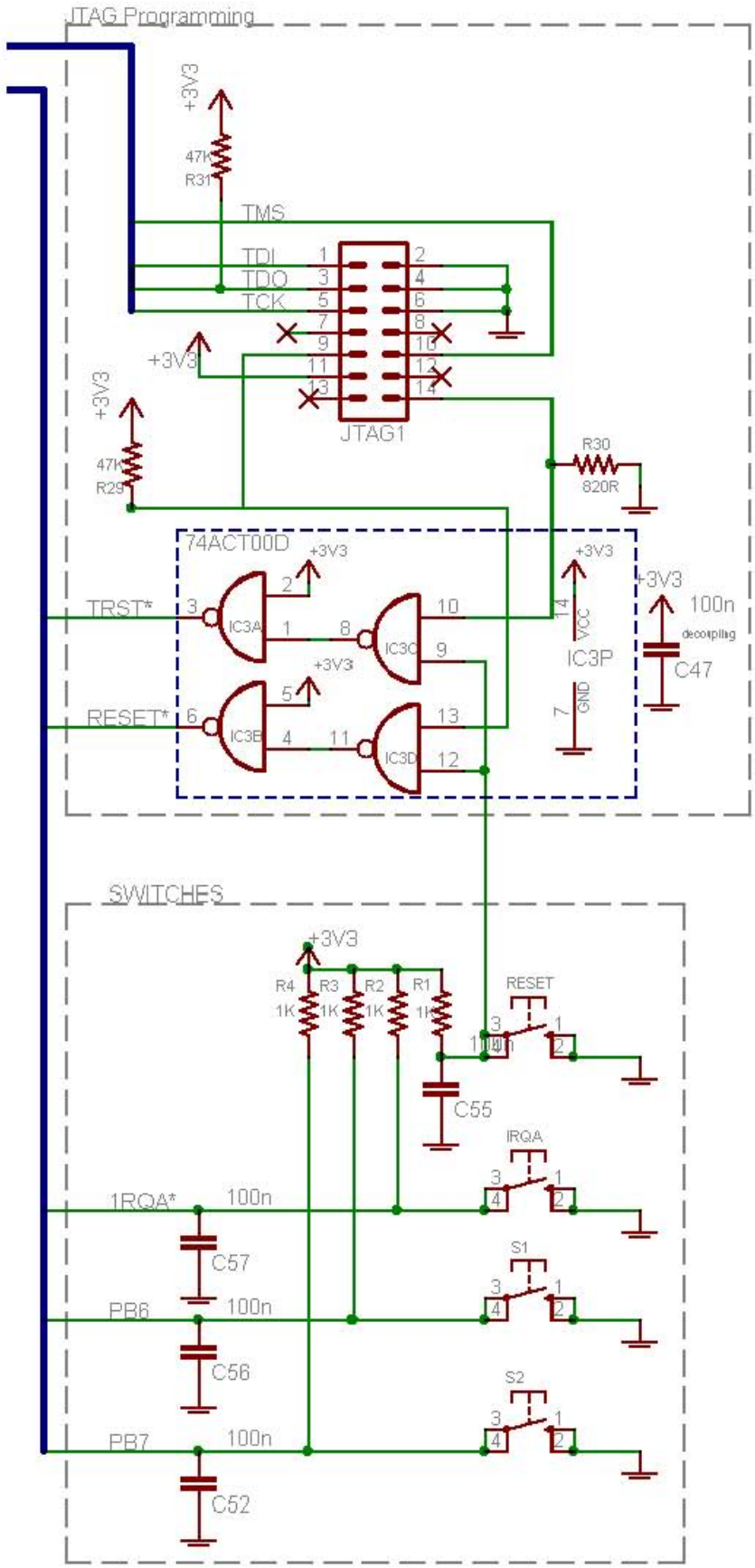


Figure H-4: JTAG programming circuit and switches

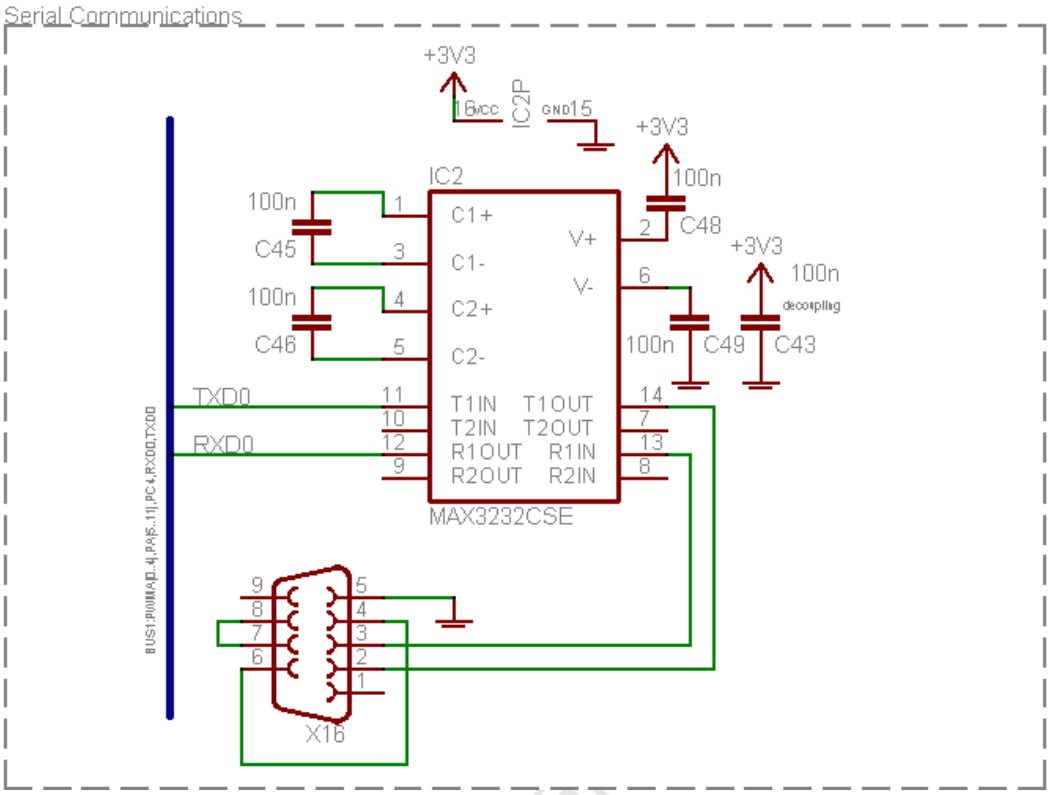


Figure H-5: Serial communications circuit