Omni-Directional Mobile Platform for The Transportation of Heavy Objects

A thesis presented in partial fulfilment of the requirements for the degree of

Masters
In
Engineering

At
Massey University,
Palmerston North,
New Zealand

Ryan Thomas
2011

"The more that you read, the more things you will know. The more that you learn, the more places you'll go."

Dr. Seuss

Abstract

As the cost of work related injuries continues to rise, it becomes more economically viable to employ robotic help in the workplace. This thesis details the development of a robotic platform for the purposes of aiding in the transportation of heavy objects, specifically hospital beds. Because in most workplaces space is at a premium, it is important for any device to be highly manoeuvrable. As anthropomorphic robots are not at a stage of providing this kind of service, a wheeled platform is the most cost effective option. To maximise the manoeuvrability of such a platform an omni-directional approach is advantageous.

Existing commercial products have been investigated and found to be lacking true omni-directional capabilities. These platforms also rely on lifting the beds, usually with large forklift like mechanisms that encumber bed attachment in small room.

Mecanum wheels have been used to achieve omni-directional movement for this platform. For Mecanum wheels to function correctly all four wheels must be individually powered and must maintain contact with the ground at all times. To achieve this, various suspension systems have been looked at with the most suitable being chosen. Various methods of speed feedback have been investigated and a sensorless method has been compared to a traditional quadrature encoder.

Motor drivers/controllers are necessary to enable the precise control of the motors which is essential for the omni-directional capabilities. Commercial drivers have been investigated but found to be inadequate for the purposes of this project. Therefore a motor driver has been developed, built and tested including the sensorless speed feedback system. The directional user input is handled by a joystick which is interpreted by a microcontroller. This microcontroller controls all the aspects of the platform including the motor drivers, LCD screen, an inclination sensor and another microcontroller for any possible add-on equipment.

The add-on equipment for this project is a bed attachment device. A headboard gripping system provides the most robust, compact and flexible system for bed attachment. A combined microcontroller and motor driver arrangement has been designed for the gripper. The gripper and the platform have been combined with a human interface and the system has been tested as a whole. The system has been fully tested in the lab and is awaiting clearance for field trials.

Acknowledgements

I would like to take this opportunity to thank the many people who have aided in my completion of this project. Firstly I would like to thank my supervisor Dr Gourab Sen Gupta, he has provided continuous support in all aspect of this project and his knowledge has been invaluable. I would also like to thank Ken Mercer for his incredible knowledge of electronics and his insightful aid in the development of the various electronic systems of the project.

I would like to thank Clive Bardell, Kerry Griffiths, Greg McLeay and Caleb Millen for their help in the design and fabrication of the mechanical aspects of the project.

Finally I would like to thank Collin Plaw and my fellow Masters students Peter Barlow, Mark Seelye and Quan Vu for their advice throughout this process.

Table of Contents

List of Figures

1 Introduction

The cost of work related injuries sustained from lifting, pushing or pulling large objects has been steadily rising over the years [1, 2]. "The cost of back injury claims has increased by 131% over the last decade and continues to increase"[1]. As this trend continues, it becomes increasingly economically viable to have a Wheeled Mobile Platform to aid human workers and lower the risk of injury.

Due to cost restraints most work places tend to be restricted in floor area. An omni-directional vehicle offers many advantages in confined spaces with the ability to move in any direction eliminating the need for turning space [3]. This advantage is especially true when pushing/pulling large wheeled objects in limited space. A more specific example of this would be a device for moving hospital beds. Currently hospitals are hiring workers specifically for moving patient beds as many nurses do not have the strength necessary to move them, leading to increased workplace injuries. An easily controlled platform, capable of moving the beds in confined spaces, would greatly reduce cost to the hospital in workplace injury claims and reduced staff numbers. This would also result in both an improved work environment and a rise in productivity for the nurses who would be less prone to injury and no longer dependent on bed moving orderlies.

Another example is described in [4] where an omnidirectional robot is used to assist the disabled in a factory setting. There would be a practical use for a platform of this type in almost every industry whether it be moving heavy equipment or acting as a mobile work station.

1.1 Background

Massey University was approached by a local business man who had previously owned a hospital. He was very aware of the lack of a capable hospital bed moving device and the problems this posed to nurses. Although the current devices were adequate for hall way travelling they failed to move the beds from the small hospital room into the hall way. He had already constructed a prototype that proved that an omni-directional with a gripping mechanism was capable of attaching to and moving beds. However, for reasons stated in section 1.2 this platform was not capable of moving a hospital bed from the room to the hallway. For the purposes of this investigation the problem will be approached with the final measure of success being the platform's ability to safely and easily move a hospital bed from

the patient's room to an operating theatre. However an eye will be kept to maintaining the flexibility of the system so it can be easily modified for other purposes.

1.2 Existing Platform

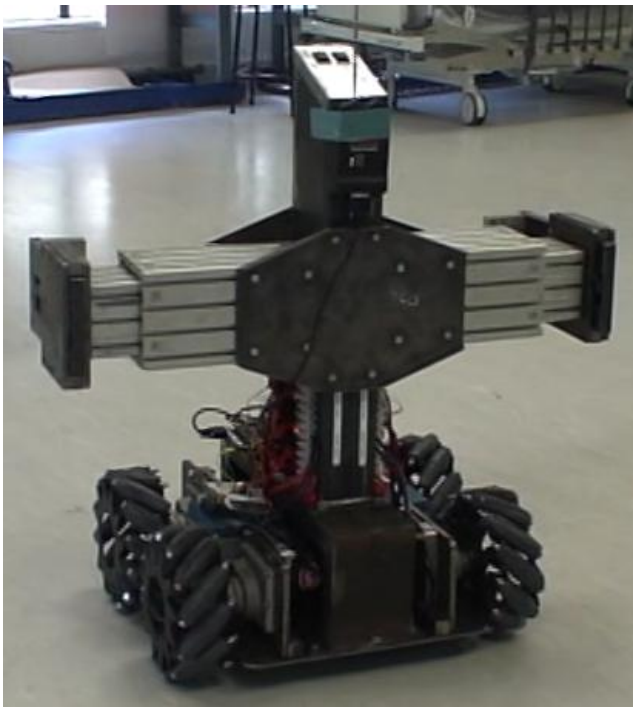


Figure 1-1: Original prototype of bed moving platform

The main purpose of the first prototype (Figure 1-1) was to prove that the concept would work. Although it was successful in proving that a bed can be moved by gripping the head board and moving it in an omni-directional fashion, the platform failed to achieve the main objective of moving a bed out of a hospital room and into the adjacent hallway. There are several reasons for this that will be explained below.

The main reason for this failure is the control inadequacies of the platform. First the joystick used was a very old computer gaming joystick; this had a large amount of slop and relied on worn variable resistors to provide the positional feedback. This resulted in erratic movement of the platform caused by inaccurate user input through the joystick.

Another limiting factor of the original platform was the fact that it was unable to move in a diagonal fashion. The control scheme consisted of the joystick being pushed forward or backwards for speed control, forwards and reverse, and the side to side movement of the joystick enabled the platform to turn. For sideways movement a button on the joystick must be pushed and held to change the movement of the joystick from a turning to a sideways

movement. At first this was thought to be a programming issue, but upon speaking to the original program designer, it was discovered that he found it very difficult to get any kind of usable diagonal functionality. Further investigation found the reason for this difficulty; the drivers (Innovation First (now VEX) Vector 883) would not output a voltage to the motors until the input pulse width modulated signal (PWM) was at ten percent of the maximum value. This resulted in a jerky operation or no operation at all when the wheels were required to spin at low speed as is often the case when traveling in a diagonal direction. Also, counter movement of the joystick due to sudden (jerky) movement of the platform resulted in cascading backwards and forwards movement. These drivers will be further investigated in section 3.7.1.

The disassembling of the original platform was very difficult and time consuming. In order to access the motors or batteries the entire gripping mechanism had to be detached from the platform. This made it very difficult to service the platform or quickly swap out depleted batteries. A more modular design will be needed in future prototypes.

The controller used was the "Innovation First Robot Controller" (Figure 1-2); the "Operator Interface" (Figure 1-3) module was also needed to interface with the joystick. It was found that these were no longer produced so could not be a viable option in the future. Although there is a replacement option, the "VEX Cortex Microcontroller", this provides limited flexibility and is designed to interface with the Vector driver that would no longer be used. This shows that a redesign of the controller system is necessary for this project.

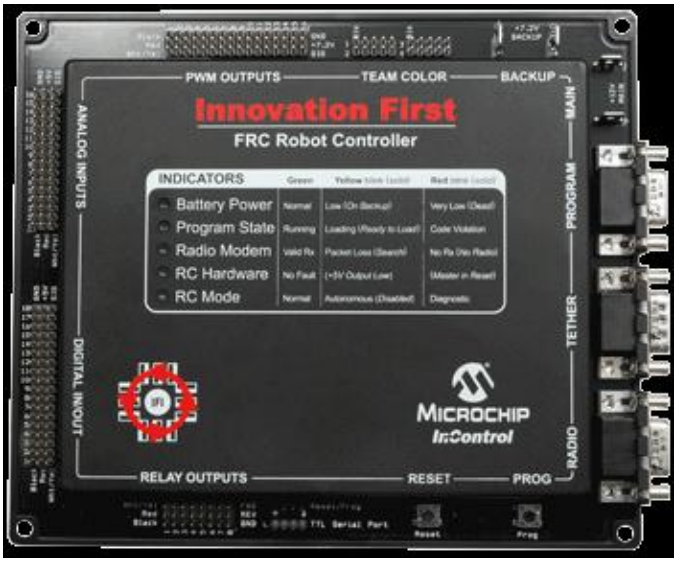


Figure 1-2: Innovation First Robot Controller

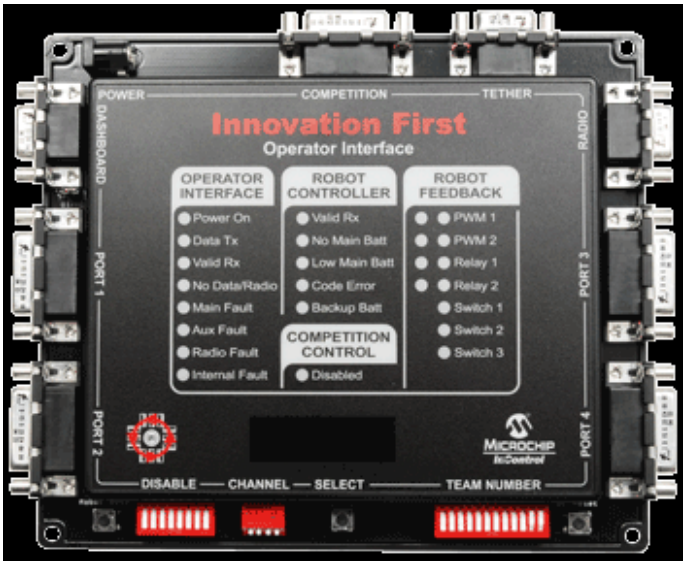


Figure 1-3: Innovation First Operator Interface

The bed gripping system on the existing platform consisted of a double rack and pinion electric actuator shown as M1 in Figure 1-4. This enabled the arms of the gripper to extend out past the end of the bed's headboard. From here the fingers can be extended out past the bed headboard using motors 2 and 3 (M2, M3). The arm actuator and finger actuators can then be closed on the headboard to lock the platform to the bed.

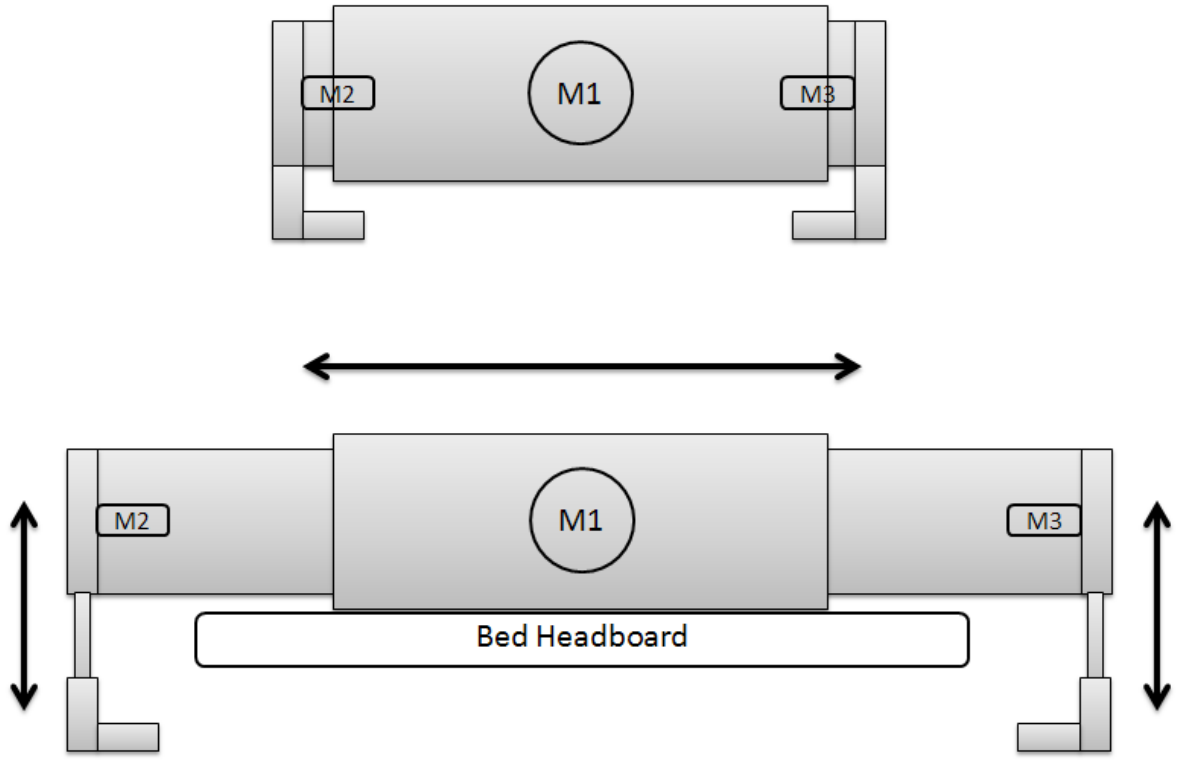


Figure 1-4: Existing Bed Gripper Mechanism

These motors are controlled by two switches that have three positions. One of the switches is used for opening and closing of the arm extender and the other for controlling the fingers. Upon arrival of the platform the arm extender did not operate raising concerns about the reliability of the system. However, it was reported that the gripper system worked well when operating correctly.

1.3 Specification

Following are the specifications for the new design of the bed mover.

Ergonomics

- It shall be designed with the end user in mind – hospital nurses who might be small in stature and not too mechanically minded.
- It shall be simple to operate with the minimum of controls (relying on the machine controls to exercise the correct restraints e.g. slower maximum speed when connected to the bed).

Function

- It shall be attached to the foot board of a hospital bed using the same or similar mechanism as found on the first prototype.
- It shall be able to push or hold a 300kg bed (including patient) on a 1:12 gradient on carpet.
- It shall use the redundant space under the foot of the hospital bed – beds without this feature will not be considered.
- It will be able to move forwards, backwards, sideways, crabwise and in a circular movement.
- It will be charged from a 12 or 24v DC source

Dimensions

- It will be able to fit between the foot of the bed and the wall 600mm distant from the foot of the bed.
- The depth of the machine from the foot of the bed shall be kept to a minimum.

Controls

- A joystick or similar device shall be used as the “nurse” / machine control interface.
- The control shall be “soft start” and progressive.

- Maximum unladen speed shall be in the order of 5 km/h. Maximum laden speed shall be in the order of 2 km/h.

Other desirable features

- Parking brake that is engaged when platform is not in use
- Extendible control interface to provide foot space for user while maintaining low profile for elevator travel

1.4 Thesis Organisation

This thesis has been organised into separate chapters for each major aspect of the project. The "Introduction" (1) chapter will discuss the background of the project, the existing prototype and its shortcomings, as well as the specifications for success. The "Investigation" (2) chapter examines existing products and how they compare to the outlined specifications. The theory necessary to complete such a project is also covered in the "Investigation" chapter. Justification as to why the particular hardware was chosen is provided in Chapter 3 "System Hardware" including detail on the designed motor driver boards. Chapter 4 "Design of Mechanical Structure" details the design process of the platform and bed attachment system. The "Control Circuit Boards" chapter (5) outlines the design and construction of the control boards. The software the control boards will execute is discussed in the "Software" chapter (6). Chapter 7 "Testing" discusses how the platform and bed attachment systems were tested and then integrated and tested as a complete system. Finally, the "Conclusions and Recommendations" chapter (8) discusses the overall success of the project and what improvement could be made in future design iterations.

2 Investigation

A complete and thorough investigation is needed. To gauge the need and scope for this project existing commercial products need to be investigated. Also the theory necessary to complete the project must be understood before any progress is made.

2.1 Existing Designs

There are a few commercial machines available that are dedicated to moving hospital beds. The investor in this project has had a couple of them trialled in a local hospital that he used to own. They were adequate for hall way travelling but all failed to move a bed out of the confined space of a hospital room.

2.1.1 Gzunda Hospital Bed Mover

Figure 2-1 shows the Gzunda Hospital Bed Mover [5]. This mover has no omni-directional capabilities, thus limiting its manoeuvrability in the confined space of a hospital room. It utilises twist grip variable speed control with a handle bar type steering system. It features a key start, horn and emergency stop for very simple (intuitive) control.

Figure 2-1: Gzunda Hospital Bed Mover [5]

2.1.2 MasterMover Bed Mover

The MasterMover Bed Mover [6] shown in Figure 2-2 is an electric tug designed for moving hospital and patient beds. The design is also incapable of omnidirectional movement. It features a tethered joystick control so it can be operated from either end of the bed. This design uses an electric actuator to lift the bed off the ground from under the bed wheels at one end. It has a maximum weight transfer of 600kg and drive motors rated at 0.4kW. It also features an on-board charger that accepts mains 110-240V input and regenerative breaking.

Figure 2-2: MasterMover Bed Mover [6]

2.1.3 Nu-Star Patient Transport System

Figure 2-3 shows the Nu-Star Patient Transport System [7]. This design uses a duel wheel drive system that is capable of moving a load of 750kg at 3.2mph, but not omni-directionally. It contains a sealed, maintenance free, 24V, 60Amp/hour gel battery. It features an emergency stop, safety horn, on-board battery charger and a handle bar type control system. It uses a forklift hitching mechanism that is likely to be difficult to attach to the bed end if it is close to wall, as in a small room situation.

Figure 2-3: Nu-Star Patient Transport System [7]

2.1.4 Atlas PTS 4 Bed Mover

This is the only design that has some form of omni-directional capabilities. However, these are limited to sideways movement, i.e. it cannot move in a diagonal fashion. The Atlas PTS 4 Bed Mover (Figure 2-4) [8] features tethered control, compact design and claw hitching system that may require excessive room at the bed end.

Figure 2-4: Atlas PTS 4 Bed Mover [8]

2.2 Wheels

There are many different methods of creating an omnidirectional drive system. These can be created by using conventional wheels or by using specially designed wheels arranged in a way that enables omnidirectional movement. A good overview is available in [9]. These different systems will be detailed below.

The first omnidirectional wheel was patented in 1919 by J. Grabowiecki in the United States [10]. Figure 2-5 shows such omnidirectional wheels. The design relies on the main wheel having 90 degree rollers on the circumference. This means that the wheels can be driven forward by a motor, yet have the ability to freewheel at a 90 degree angle.

Figure 2-5: Two designs of a classical omnidirectional wheel

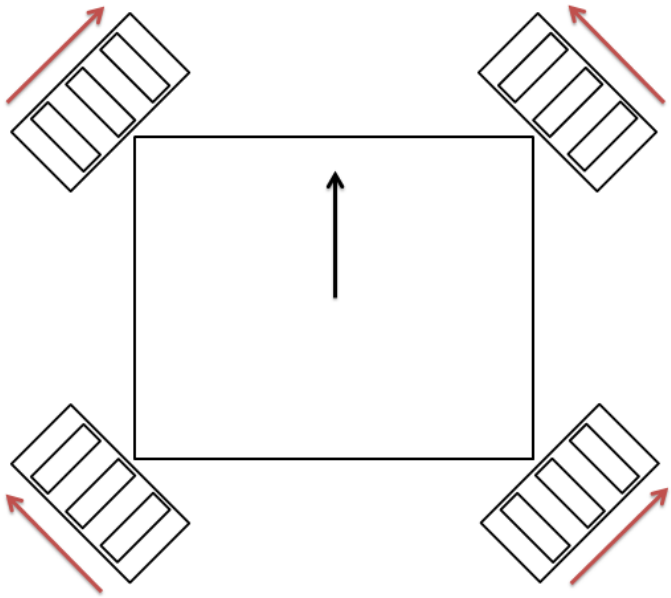


Figure 2-6: Wheel arrangement for omni-wheel design

Figure 2-6 shows the arrangement necessary to enable the platform to be omni-directional. The red arrows show the force vectors of each wheel and the black arrow shows the direction of the platform's resulting movement. Because the small rollers at a 90 degree angle on the drive wheel allow perpendicular movement, the opposing forces caused by the wheel orientation cancel each other out and result in smooth forward drive. The symmetry of the platform means that it can travel in any perpendicular direction. By adjusting the speed of each wheel the force vectors can be changed to allow the platform to travel in any direction.

Figure 2-7: Mecanum wheel design

The omnidirectional wheels shown in Figure 2-7 is called the "Mecanum wheel". This was developed by Swedish inventor Bengt Ilon in 1973 [11] for Swedish company Mecanum AB.

The principle behind this is very similar to the omni-wheel arrangement shown in Figure 2-6. However with Mecanum wheels the motors can be arranged in a more traditional fashion as shown in Figure 2-8. This arrangement still relies on the same principle of force vectors cancelling each other out to produce the desired motion. This is true to such an extent that the control equations would be very similar to that of a omni-wheel system. The kinematic equations are explained in section 2.4.

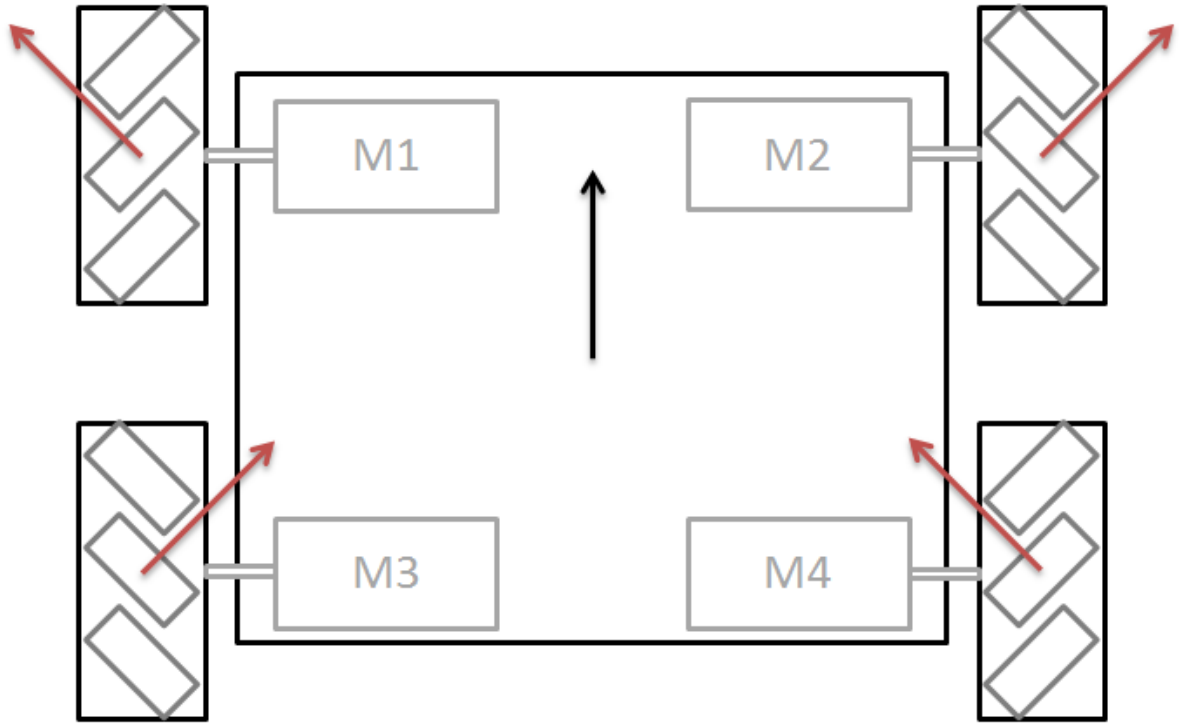


Figure 2-8: Wheel arrangement for Mecanum wheel design

Omni-wheels can have many different design characteristics resulting in varying performance as far as smooth motion is concerned. [12] describes an in depth method of how to construct an Omni-wheel, that will result in performance comparable to a traditional wheel. Mecanum wheels provide an even more complex design process, with many variables contributing to the performance and efficiency of the wheel. Two different methods of achieving high efficiency are outlined in [13]. One method is to make the rollers lockable so that when the platform is moving forward the wheels act efficiently as normal wheels. The other, more complex, method is to make the angle of the rollers adjustable to extract maximum efficiency from the wheels in all directions. This design is shown in Figure 2-9 where a rotating disk controlled by an actuator is used to change the angle of the rollers.

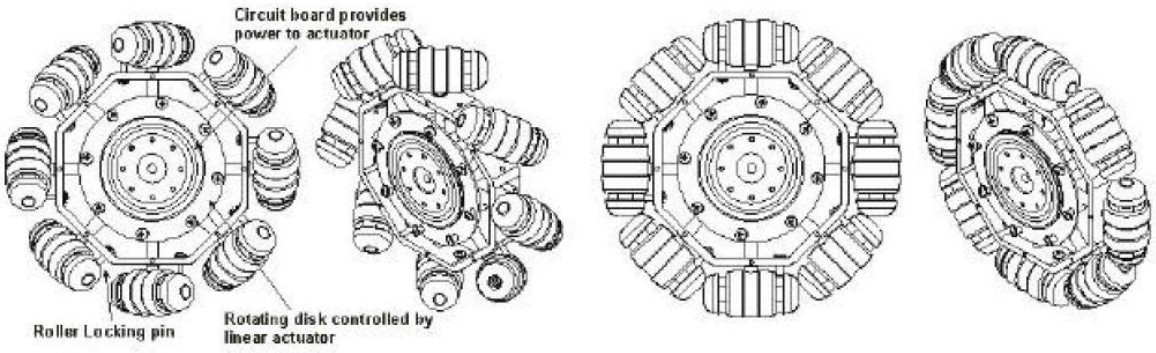


Figure 2-9: Mecanum wheel with rotatable rollers [13]

One main drawback of the Mecanum wheel design is the tendency for there to be a "bump" as the load moves from one roller to the next. There are two main variables that cause this. One is the number of rollers on each wheel as outlined in [3]. Here six is found to be the optimum number, a single point of contact is maintained and the minimum roller radius is maximised. Questions are often raised about the traction of Mecanum wheels. Because of the 45 degree roller, static friction in that direction is zero, so that in a forward direction the Mecanum wheel can be said to have 50% of the static friction of a conventional wheel. However, conventional wheeled systems tend to have only two drive wheels, where as a Mecanum drive system needs four drive wheels in order to operate. This means that the two systems would actually have the same static traction.

Another key factor in Mecanum wheel design to eliminate roller "bump" is the profile of the rollers themselves. [14] describes the necessary equations for calculating the profile of the rollers to ensure smooth transition from roller to roller.

2.3 Feedback Mechanisms

This section will discuss the two important feedback mechanisms of velocity and current. Velocity feedback is important as it allows precise control of the velocity of the platform including acceleration curves regardless of the load. This provides smooth control of the platform from an operator perspective. Current feedback is important as it can protect components of the drive system from excessive current being drawn in case of a malfunction like a jammed wheel.

2.3.1 Speed Feedback

There is an argument outlined in [15] that encoder speed feedback is unnecessary on an Omni-directional vehicle, as there is too much slipping in the wheels for the feedback to have

any accuracy. Although this is partially true, especially when considering a human controlled platform, encoder speed feedback still plays other important roles, the most important of which is to ensure that all four wheels are turning at identical speeds. Acceleration curves can also be matched exactly for each wheel. This becomes especially critical when the motors start to wear, causing even matched motors to perform differently.

Of course for automated platforms there are ways round the problems with encoder positional feedback. These include using artificial vision to control the position. One system that could be implemented in a factory setting, where the platform would move from one position to another, would be a global vision system to monitor and correct the position of the platform [16]. However, in a situation where the platform is moving objects from room to room, this becomes uneconomical as many cameras will be needed. An alternative to this is outlined in [17], where a stereo vision sensor is used to simulate a person following the platform. Through the use of the stereo vision the platform can position itself by referencing objects in 3D space. Because the vision sensor is mounted on the platform, this feedback system is not limited to a single space.

Because the proposed platform will be directly human controlled at this stage a complex vision based system would be unnecessary. Traditional rotary encoders rely on a point on the rotating part (e.g. wheel shaft) creating a countable event on the stationary part (e.g. platform/motor case). The most common method of creating such an event is by using either optical sensors or magnetic sensors. To gather more accurate information about the speed, several interrupt sources can be positioned on the shaft and to gather directional information two offset sensors can be used. Although these kind of sensors will provide adequate feedback to control the rotational speed and acceleration of the wheels, which is sufficient for a human controller platform, there are other options to explore.

One such option is optical ground tracking. Two optical mice can be fitted to a small Mecanum wheel platform to produce a dead-reckoning speed and position control system, as in [18]. This overcomes the problem of any wheel slip the shaft encoders may encounter. However, the optical mice need to be in contact with the ground limiting the platform to operating on completely flat surfaces. This may cause problems in navigating bumpy surfaces, like the ones usually found in doorways. This problem could be overcome by making the optical tracing system float on a suspension system. A more elegant solution

would be to use an optical sensor that can operate at a distance, more like a camera with appropriate processing to output positional feedback.

2.3.2 Current Feedback

An excellent overview of various current sensing technologies is presented in [19]. It details six different technologies; integrated current shunts, current transformers, Rogowski coils, Hall-effect current sensors, Giant Magneto Resistive sensors (GMR) and Magneto Impedance sensors (MI). Shunt, Hall-Effect, GMR and MI are capable of measuring DC current.

Integrated current shunts use the voltage, current and resistance relationship to exploit the fact that the voltage drop will increase across a known resistance as current increases. Because they need to be inserted in the direct current path they decrease efficiency and produce some heat.

Hall-effect current sensors sense magnetic field changes based on the Hall Effect. Because of this the device is completely isolated and has no effect on the efficiency of the system. It also causes the device to be sensitive to outside magnetic interference which is prevalent when using large electric motors. Another more detailed explanation can be found in [20].

Integrated Giant Magneto Resistive current sensors are based on the principle of the change of electrical resistance with the magnetic field. These are also very sensitive to magnetic fields. They are usually deposited and etched into a Wheatstone bridge pattern. These are complicated devices that are very difficult to integrate into a circuit [19].

"The MI (Magneto Impedance) effect refers to the variation of the impedance $Z=R(\omega,H)+iX(\omega,H)$ of a magnetic material carrying a low intensity, high frequency AC current when subjected to an external magnetic field." [19]. These devices are very sensitive to external magnetic fields. However, they are cheap, small, have quick response times and are highly sensitive. They are unfortunately difficult to integrate.

2.4 Control Mechanisms

In order to control a Mecanum wheel based vehicle it is critical to understand the kinematic equations involved. The explanation below gives the control equations for each motor based on a x, y, w (longitudinal, lateral, and yaw) input.

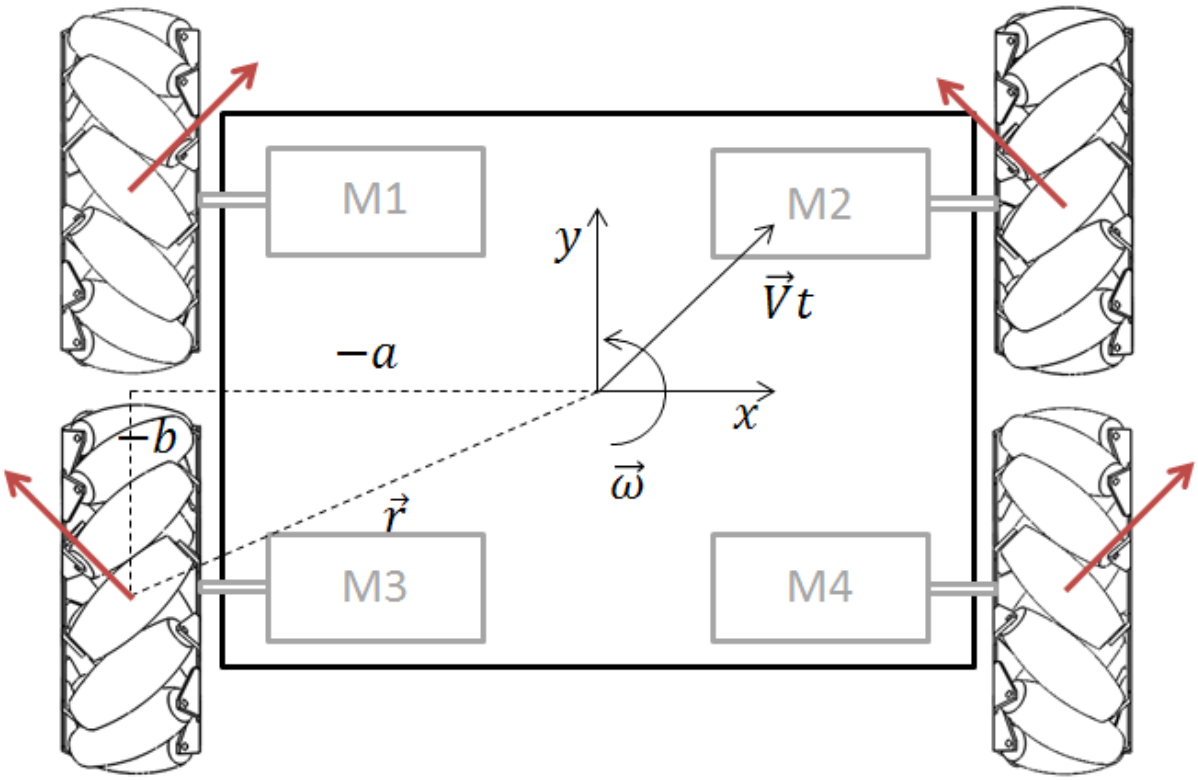


Figure 2-10: Birds eye view of Mecanum wheel layout

Figure 2-10 shows the force vectors (red arrows) resulting from the wheels being driven forward. It can be seen that if all four wheels are driven forward at the same speed the x-axis components cancel out and the platform will move forwards [21].

Here $\vec{V}t$ is the desired velocity of the platform, $\vec{\omega}$ is the rotational velocity of the platform and $\vec{r}$ is the distance from the centre of the platform to the wheel contact point with a and b being the scalar components.

The vector component of each wheel can be shown as:

$$\vec{V} = \vec{V}t + \vec{\omega} * \vec{r} \tag{2.1}$$

Where $\vec{V}$ is vector velocity of a single wheel

Therefore the scalar components are:

$$Vx = Vtx - \omega \cdot ry \tag{2.2}$$

$$Vy = Vty + \omega \cdot rx \tag{2.3}$$

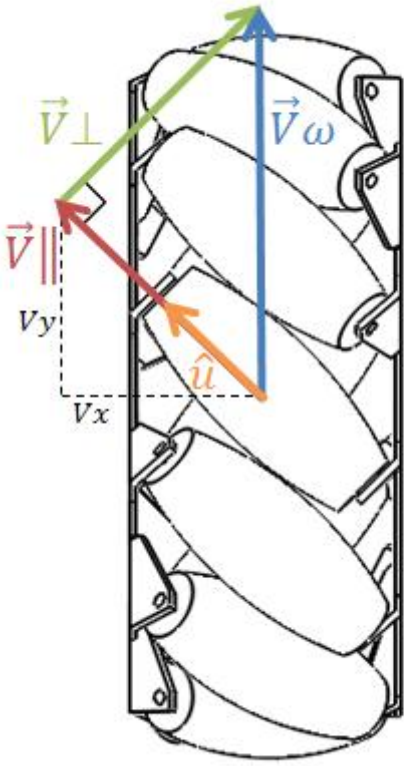


Figure 2-11: Force vectors acting on right Mecanum wheel (Bottom view)

Calculate $\vec{V}\omega$ for right wheel (Figure 2-11)

$$\cos 45 = \frac{\vec{V}\parallel}{\vec{V}\omega} \tag{2.4}$$

$$\vec{V}\omega = \frac{\vec{V}\parallel}{\cos 45} \quad \vec{V}\parallel = -\vec{V}x\cos 45 + \vec{V}y\cos 45 \tag{2.5}$$

$$\vec{V}\omega = \frac{-\vec{V}x\cos 45 + \vec{V}y\cos 45}{\cos 45} \quad \cos 45 = \frac{1}{\sqrt{2}} \tag{2.6}$$

$$\vec{V}\omega = \sqrt{2}(-\frac{1}{\sqrt{2}}\vec{V}x + \frac{1}{\sqrt{2}}\vec{V}y) \tag{2.7}$$

$$\vec{V}\omega = -\vec{V}x + \vec{V}y \tag{2.8}$$

The two motors that use this wheel orientation are the right front and the left rear.

$$a = rx \tag{2.9}$$

$$b = ry \tag{2.10}$$

Right Front Motor:

$$V_2x = Vtx - \omega b \tag{2.11}$$

$$V_2y = Vty + \omega a \tag{2.12}$$

$$\vec{V}_2\omega = -\vec{V}_2x + \vec{V}_2y \tag{2.13}$$

$$\vec{V}_2\omega = -Vtx + \omega b + Vty + \omega a \tag{2.14}$$

$$\vec{V}_2\omega = Vty - Vtx + \omega(a + b) \tag{2.15}$$

Left Rear Motor:

$$V_3x = Vtx - \omega(-b) \tag{2.16}$$

$$V_3y = Vty + \omega(-a) \tag{2.17}$$

$$\vec{V}_3\omega = -\vec{V}_3x + \vec{V}_3y \tag{2.18}$$

$$\vec{V}_3\omega = -Vtx - \omega b + Vty - \omega a \tag{2.19}$$

$$\vec{V}_3\omega = Vty - Vtx - \omega(a + b) \tag{2.20}$$

For the other two motor wheel orientations (Figure 2-12), $\hat{u}$ has changed directions.

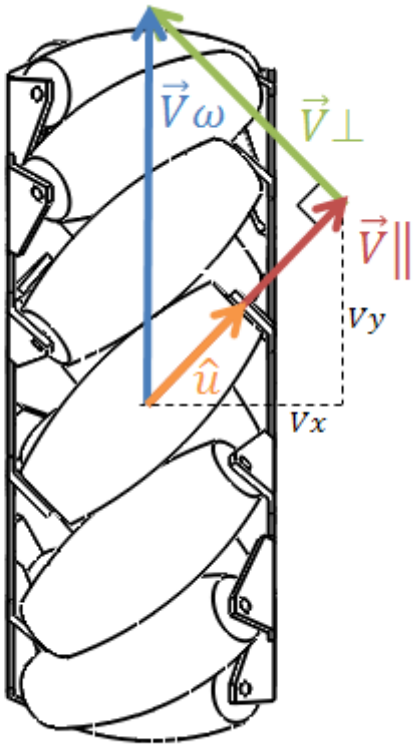


Figure 2-12: Force vectors acting on left Mecanum wheel (Bottom view)

Calculate $\vec{V}\omega$ for left wheel (Figure 2-12)

$$\vec{V}\| = \frac{1}{\sqrt{2}}Vx + \frac{1}{\sqrt{2}}Vy \tag{2.21}$$

$$\vec{V}\omega = \frac{\left|\vec{V}\|\right|}{\frac{1}{\sqrt{2}}} \tag{2.22}$$

$$\vec{V}\omega = Vx + Vy \tag{2.23}$$

The two motors that use this wheel orientation are the left front and the right rear.

Left Front Motor:

$$V_1x = Vtx - \omega b \tag{2.24}$$

$$V_1y = Vty + \omega(-a) \tag{2.25}$$

$$\vec{V}_1\omega = \vec{V}_1x + \vec{V}_1y \tag{2.26}$$

$$\vec{V}_1\omega = Vtx - \omega b + Vty - \omega a \tag{2.27}$$

$$\vec{V}_1\omega = Vty + Vtx - \omega(a + b) \tag{2.28}$$

Right Rear Motor:

$$V_4x = Vtx - \omega(-b) \tag{2.29}$$

$$V_4y = Vty + \omega a \tag{2.30}$$

$$\vec{V}_4\omega = \vec{V}_4x + \vec{V}_4y \tag{2.31}$$

$$\vec{V}_4\omega = Vtx + \omega b + Vty + \omega a \tag{2.32}$$

$$\vec{V}_4\omega = Vty + Vtx + \omega(a + b) \tag{2.33}$$

This result in the complete kinematic equations being:

$$\vec{V}_1\omega = Vty + Vtx - \omega(a + b) \quad \text{Left Front Motor}$$

$$\vec{V}_2\omega = Vty - Vtx + \omega(a + b) \quad \text{Right Front Motor}$$

$$\vec{V}_3\omega = Vty - Vtx - \omega(a + b) \quad \text{Left Rear Motor}$$

$$\vec{V}_4\omega = Vty + Vtx + \omega(a + b) \quad \text{Right Rear Motor}$$

With these equations, basic open loop control can be achieved easily with a three axis joystick. However, Mecanum wheel platforms have some control ambiguities, these are discussed in [22], they include; slippage between the sub roller and the floor, axel friction in sub rollers limiting supposed frictionless direction, and point contact friction between the wheel and the floor limiting rotational slip. These factors influence the effectiveness of

Mecanum wheels in a dead-reckoning system. To overcome these limitations a wide variety of control methods have been developed. Some of these will be discussed below.

The problems discussed in [22] have also been addressed. In an attempt to eliminate positional errors in a dead-reckoning positioning system Kyung-Lyong, H., K. Hyosin, and J.S. Lee introduced parameter adjustments on the corresponding wheel velocities. From here an adjustment equation can be acquired and calibrated accordingly. This was then tested and proved to result in a significant improvement in positional errors.

There are several control methods that involve using artificial vision as a corrective positioning technique [16, 17, 23]. One Bird's Eye View method involves treating camera feedback as a slower corrective feedback for the positional dead-reckoning control provided by the on board encoders [23]. While the stereo vision following technique outline in [17] relies on the vision system for positional control entirely with wheel encoders used for speed control.

Finally there is a strong argument outlined for the merits of using fuzzy logic to tune the parameters of a PID controller. Various methods of achieving this are outlined in [14], [24], [25] and [26]. The tuning parameters of a PID controller are critical if smooth motion is to be achieved. These can be very difficult to tune through a trial and error method, so a method of adaptive gain scheduling could prove very useful.

2.5 Chapter summary

The existing designs outlined fall short of offering an onmi-directional, compact device that can be used to transport a hospital bed from the patient room to a hallway. The theory discussed shows two different approaches to enabling omni-directional movement of a platform and derives the control equations. The importance of feedback mechanisms for the function and safety of the platform have been discussed and a variety of different sensing methods have been outlined for both speed and current feedback. Finally different control theories have been discussed and along with potential problems along with solutions that may be encountered.

3 System Hardware

This chapter will take a detailed look at the selected hardware for the system. It will also assess why the particular hardware has been chosen over other alternatives. The hardware considerations are the wheels, motors, batteries, microcontroller, encoders, human interface, motor drivers and inclination sensor.

3.1 Wheels

There are a few commercially available Mecanum wheels to be found. “Airtrax” make Omni-directional forklifts that use large Mecanum wheels. However, these prove to be too large for our purposes. This leaves two options remaining for Mecanum wheel acquisition, use the existing wheels from “AndyMark, Inc.”, or design and manufacture custom Mecanum wheels. The specification sheet for the “AndyMark, Inc.” Mecanum wheels [27] outlines that the load maximum is 440 pounds (200kg) per wheel. As this load is adequate, and because of the great time commitment designing and manufacturing in-house Mecanum wheels would involve, it was decided to continue using the “AndyMack, Inc” 10 inch Mecanum wheels show in Figure 3-1.

Figure 3-1: AndyMark 10" Mecanum Wheels [28]

The other advantage of using these Mecanum wheels was the availability of technical drawings and co-efficient of friction data. This will greatly aid in the design of the platform.

3.2 Motors

The worst case scenario for the platform would be accelerating up a 4.6˚ (1:12, maximum allowed in hospital) slope with a 400kg load at 2 km/hr speed. Two seconds has been chosen as a reasonable acceleration time.

Acceleration of platform over 2 seconds:

$$2km\ per\ hour = 0.556ms^{-1} \tag{3.1}$$

$$\frac{0.556ms^{-1}}{2s} = 0.278ms^{-2} \tag{3.2}$$

Acceleration to overcome gravity:

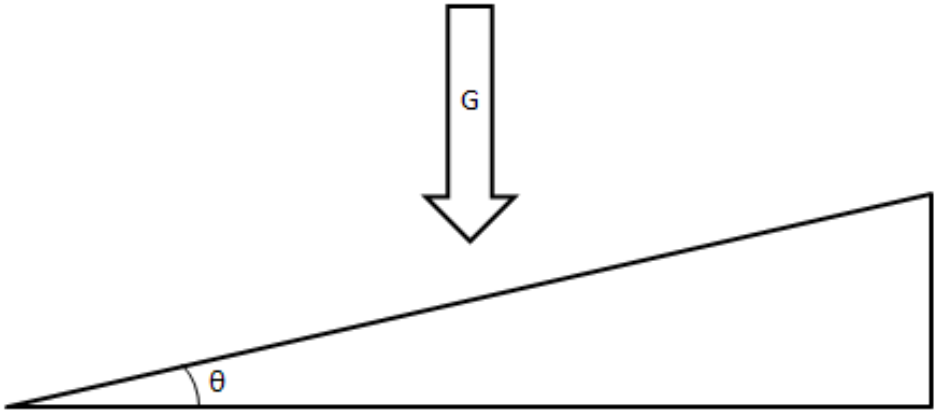


Figure 3-2: Gravity considerations

$$G \times \sin\theta = 9.81ms^{-2} \times \sin 4.6^{\circ} \tag{3.3}$$

$$= 0.787ms^{-2} \tag{3.4}$$

Total acceleration:

$$A = 0.278ms^{-2} + 0.787ms^{-2} = 1.065ms^{-2} \tag{3.5}$$

Required force:

$$F = M \times A = 400kg \times 1.065ms^{-2} = 426N \tag{3.6}$$

Work done:

$$W = F \times d \quad \Big| \quad d = \frac{1}{2} \times 2s \times 0.556ms^{-1} = 0.556m \tag{3.7}$$

$$W = 426N \times 1.065ms^{-2} = 236.856J \tag{3.8}$$

Power required:

$$P = \frac{W}{t} = \frac{236.856J}{2s} = 118.428W \tag{3.9}$$

Add 50% for losses in motor efficiency and frictional losses (safety margin):

$$\frac{177.642W}{4\ motors} = 44.411W \tag{3.10}$$

This means that at least 45W motors are required for the platform.

Torque calculation (where r is radius of wheel):

$$\tau = F \times r \quad \Big| \quad F = 426N \tag{3.11}$$

$$\tau = 426N \times 0.127m \quad \Big| \quad r = 0.127\ for\ 10\ inch\ wheel \tag{3.12}$$

$$\tau = \frac{54.102Nm}{4\ motors} = 13.526Nm \tag{3.13}$$

This means that each motor must be able to provide 15Nm of torque.

The motors on the existing platform are the "NPC-T64" 24V permanent magnet motor from "NPC Robotics". These motors can deliver the required 15Nm of torque at 21A continuous current. These motors meet the power requirements but they do not feature a parking break or a fitted encoder making them unsuitable for the new prototype.

There are many different types of motors that could be used for the drive purposes of the platform. These include brushed DC motors, brushless DC motors and stepper motors.

Because of the nature of the usage of the platform, the motors will not sustain high levels of usage. This negates the advantage that brushless and stepper motors have of being essentially maintenance free. As the motors will not be continuously running, like in some factory circumstances, it is hard to justify the far greater cost of the brushless motor and the added complexity for the controller.

A common type of motor that does feature the required parking break is a right angled gear motor found in many electric wheelchairs. However, these motors do not usually feature encoders and the parking break can hinder encoder mounting efforts by blocking the motor shaft. “Allied Motion” offers high quality right angled gear motors and offer factory modifications including the fitting of a heavy duty parking break (Figure 3-4) that allows a through shaft for an encoder to be fitted.

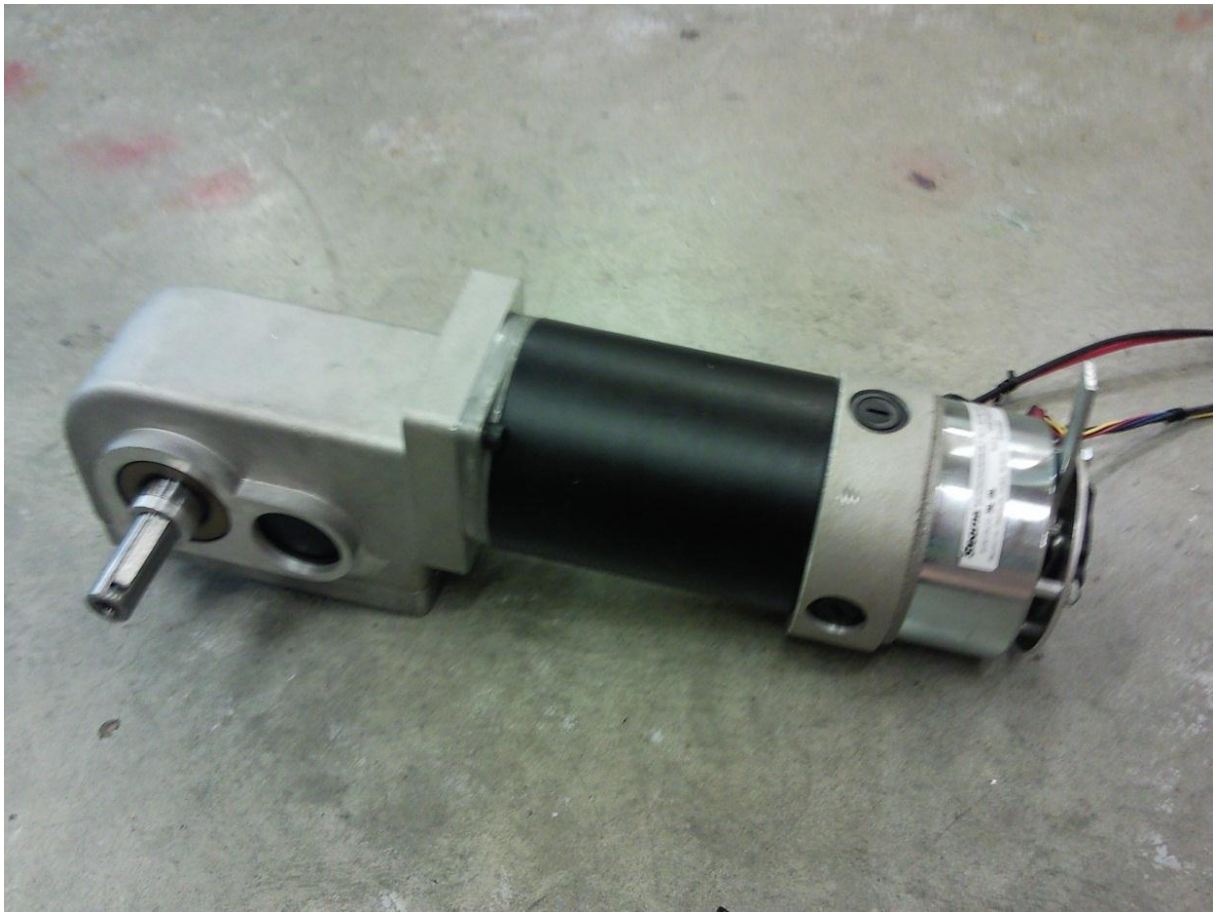


Figure 3-3: Allied Motion RAD series right-angle motor

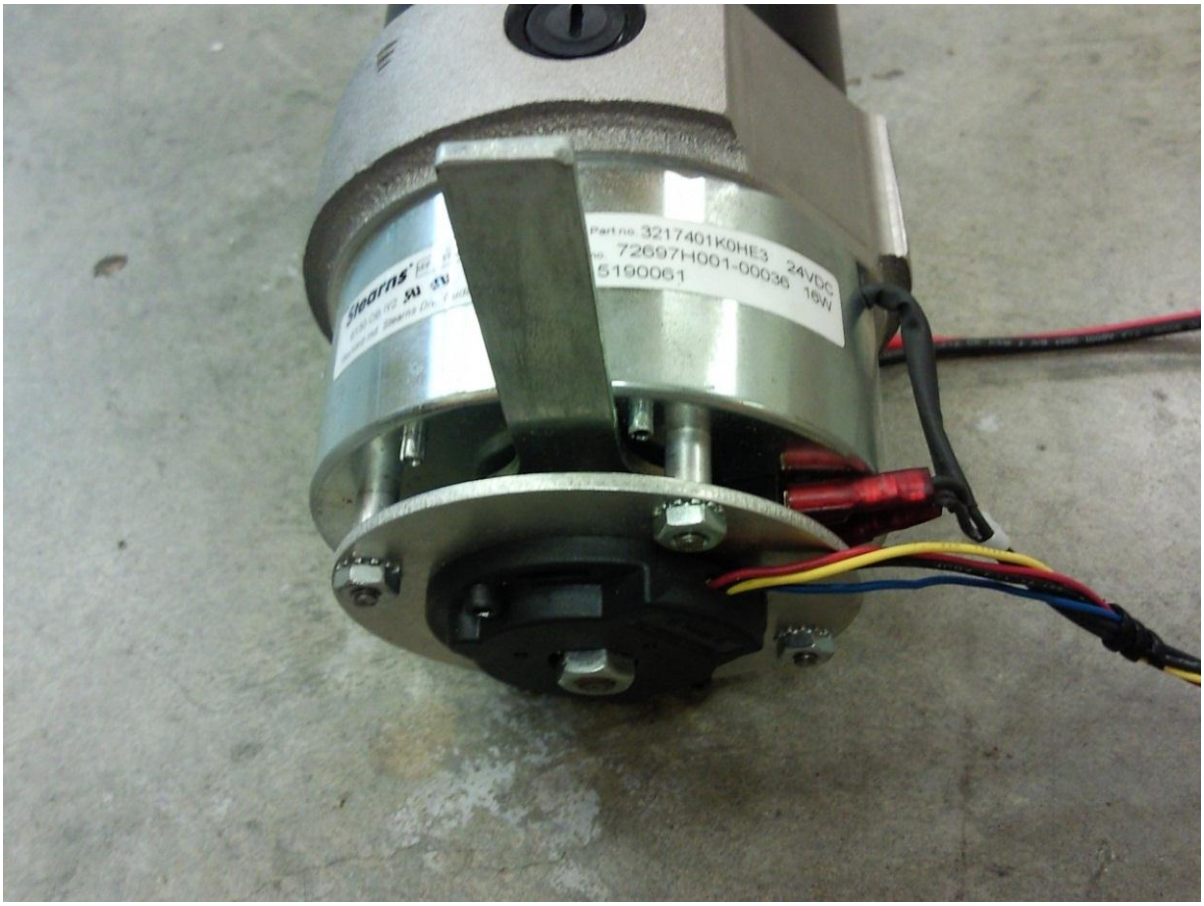


Figure 3-4: Heavy duty parking break with through shaft

The "Allied Motion" RA-DTPL-36DR-18-00, show in Figure 3-3, is a 24V brushed DC motor. It has a right angle gearbox with an 18:1 gearing, can produce 15Nm of continuous torque with a maximum constant current of 18A and a speed of 160rpm.

3.3 Batteries

This platform will be electrically powered and highly mobile so there is a need for a battery powered system. The selected motors require a 24V power supply that needs to have sufficient capacity to power the four drive motors for an appropriate amount of time. There are four main viable commercially available battery systems that could be considered for an application like this. These are: Nickel Cadmium, Nickel-Metal Hydride, Lead Acid and Lithium Ion. Table 1 taken from [29] and [30] compares the characteristics and cost of each.

Table 1: Comparison of rechargeable cells

	Nickel-cadmium	Nickel-metal-hydride	Lead-acid	Lithium-Ion
Gravimetric Energy Density(Wh/kg) 80 (initial)	45-80	60-120	30-50	110-160
Fast Charge Time	1h typical	2-4h	8-16h	2-4h
Cost per kWh ($US)	$7.75	$19.50	$1.00	$20

As can be seen from Table 1 the best battery to use from a longevity standpoint would be Lithium-Ion. However, this technology is very expensive. Nickel-Metal-Hydride, the next best Wh/kg rating, has similar problems with cost and discharge current as Lithium-ion. This leaves Nickel-cadmium and Lead-acid batteries to choose from. Because the product is a mid-sized wheeled platform that will be moving heavy objects, the weight of the batteries becomes less important, meaning that the gravimetric energy density is not a strong factor in the battery type decision. The other factor to consider is the charge time. Although, the Nickel-cadmium has a clear advantage in this category there is a problem of battery memory. Nickel-cadmium needs to be periodically discharged completely and recharged in order to maintain the life of the battery. This reason, combined with the fact that overall battery weight is not an important factor, and the cost of Lead-acid batteries being significantly less, a sealed lead acid battery has been chosen.

Because the platform will be operating in places such as hospitals it is important that there is no possibility of the battery leaking. Sealed Gel Lead-acid batteries ensure that this cannot happen. The main current draw in the system will be the motors. Under normal working conditions these will draw 8A of current, i.e. 32A with four motors. In order to achieve at least 30 minutes of continuous operation a battery with a rating of 33Ah will be needed. The Commander Gel Deep Cycle Sealed Battery (Figure 3-5) is one such suitable product. It uses the latest lead-acid technology and the features include [31]:

- 33Ah capacity
- 13.7V float charging voltage

- Maximum discharge of 300A (5s)
- ~1100 cycles at 30% discharge
- 11kg weight

Figure 3-5: Commander Gel deep cycle sealed batteries

3.4 Microcontroller

A microcontroller is required to integrate and control the various systems on the platform. There are a wide variety of microcontrollers on the market with varying features and architectures. Because this system is relatively small and there will be no need for a great deal of processing power the obvious choice of processor is the "Silicon Labs 8051" series. The reason for this it is that is a widely used within Massey University with development boards available for necessary testing. Also my supervisor, Dr Gourab Sen Gupta, has vast experience using these microcontrollers and has even written a book [32] detailing their operation. These factors are of great value to the successful integration of an 8051 microcontroller.

More specifically, the microcontroller that will be used is the "Silicon Labs" C8051F020. This microcontroller utilises five programmable timers, a Programmable Counter Array (PCA) that may be used for up to five Pulse Width Modulation (PWM) signals, eight 12-bit analogue inputs, eight 8-bit analogue inputs, two Digital to Analogue converters, two UART

serial interfaces and up to 64 digital port pins for both receiving and transmitting digital signals.

With this impressive list of features and the 22MHz oscillator that the development boards use, this controller provides a powerful, easily accessible and cost effective option.

3.5 Encoders

There are many options available when it comes to fitting rotary encoders to the selected motors. The two most common forms of rotary encoders are optical and magnetic. Optical encoders have the advantage of being able to perform very high precision encoding using very small slits in a glass or plastic disk. The disadvantage of optical encoders is their durability vs. price trade off. Because they rely on light passing through small slits to a sensor on the other side, any dust or moisture that enters the mechanism can interfere with this critical interaction. They can also be susceptible to shock, vibration, operating temperature, high speeds and wear and tear. Sealed optical encoders are available; however these are much more expensive than the alternative. Magnetic rotary encoders exploit the Hall Effect phenomena by having a magnetic disk rotating past a Hall Effect sensor. A detailed description is available in [33]. The disadvantage of this system is the inability to achieve the high resolution of an optical encoder. However, it possesses a great advantage in the reliability/durability aspects at a low cost. These encoders are not affected by wear and tear like the cheaper optical encoder and are not susceptible to dust or moisture. They are affected only by magnetic interference, and even this is minimal.

The other question about encoders is the number of channels. Single channel encoders only provide speed feedback with no directional knowledge. A quadrature encoder on the other hand consists of two channels that are offset by a quarter of a wavelength to provide speed as well as directional feedback. The other advantage of a quadrature encoder is that it has twice the resolution of the single channel equivalent because of the extra channel providing double the pulses pre revolution.

As directional feedback is essential to the control of the platform, a two channel encoder will be necessary. Magnetic encoders would also be desirable for the superior reliability but only if an acceptable resolution can be achieved.

“Phoenix America” is a company that specialises in magnetic encoders. The maximum encoder wheel they supply can produce 30 pulses per revolution, although this is low

revolution, the sensor unit they supply is a quadrature encoder and it mounts on the motor shaft. As the gear box of the motors has a ratio of 1:18 and the pulse resolution can be doubled by counting the positive and negative transitions, the encoder can produce a resolution shown by the equation below.

$$30 \times 2 \times 2 \times 18 = 2160\ counts/rev \tag{3.14}$$

Because of the quadrature nature of the encoder and the 1 to 18 gearbox ratio, an acceptable resolution can be achieved.

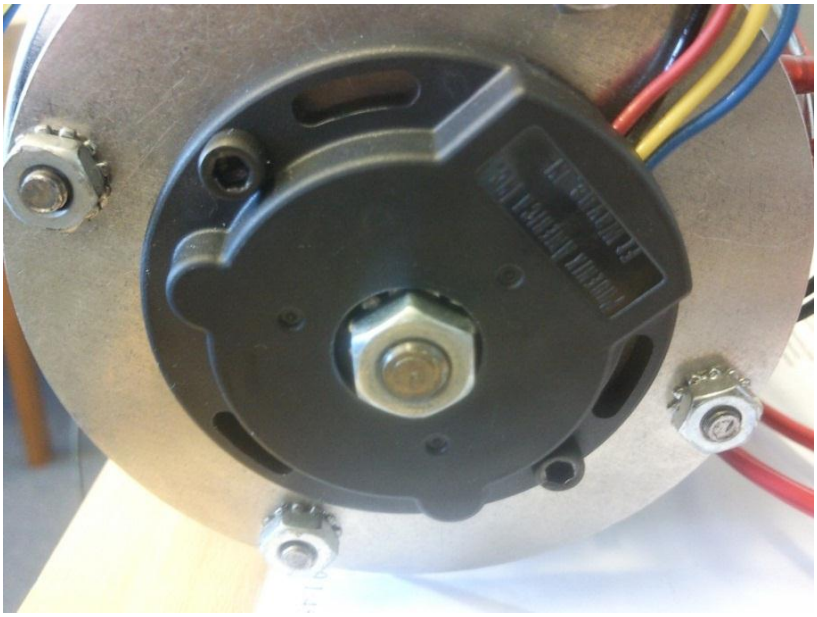


Figure 3-6: Phoenix America quadrature magnetic encoder

3.6 Human Interface

The human interface is a critical aspect of the project. In order for this platform to be successful it needs to be intuitive and easy to control. This section will look at the hardware chosen to perform this task and the reasons for these choices.

3.6.1 Directional input

There are a few different options for the directional feedback of the platform. Some of these are used by the existing machines discussed in section 2.1. There is a handle bar with twist throttle design. This is an intuitive system that most people will understand quickly. However it becomes difficult to input a sideways movement. One way to do this would be to enable the handle bars to move sideways. However, this would be difficult to implement and may cause the control to become problematic, especially when diagonal movement is desired.

The other common directional input is the joystick. This form of input is more suited to an Omni-directional vehicle as the joystick can be moved in any direction. The only limitation with this system is the ability to input a rotational input. This can be overcome by having a three axis joystick that has the ability to spin, although this could also be difficult to control.

Because a steering wheel is the most common and easy to use control input device, it needs to be investigated. Although this would be the best system from an adoptability perspective, considering that most operators have driven a car, there are some problems. First there is the problem of speed input. In a car the speed is controlled by a foot pedal; this is not possible in a walk along platform. A twist or thumb throttle could be incorporated but there is still the problem of sideways movement where a leaning wheel could be used. The other disadvantage of this system would be the space taken up by a steering wheel.

As space is limited on the platform and there is a need for a three axis input, a joystick is the best choice for the human interface directional input device from a feasibility and simplicity standpoint. There was a problem with the existing platform involving the movement of the joystick when the platform moved and the hand holding the joystick stayed stationary. This would have a very bad oscillating effect on the platform. In order to overcome this it is important that the operator rests his hand on the platform while using the joystick. This holds the hand steady in relation to the platform and will allow smoother control. To achieve this, a small finger operated joystick is required.

Figure 3-7: Penny + Giles JC2000 joystick

The Penny + Giles JC2000 Fingertip 3 axis industrial joystick, with push button, shown in Figure 3-7, uses Hall Effect sensors to output 3 analogue signals ranging from 0.5V to 4.95V. It is smaller than a traditional joystick, measuring only 78mm from the mounting flange to the push button, meeting the requirement of finger operation. The three signals from the joystick are too high to be fed straight into the microcontroller's analogue input. They must first be passed through a voltage divider in order to drop the voltage to within the 2.4V reference of the 8051.

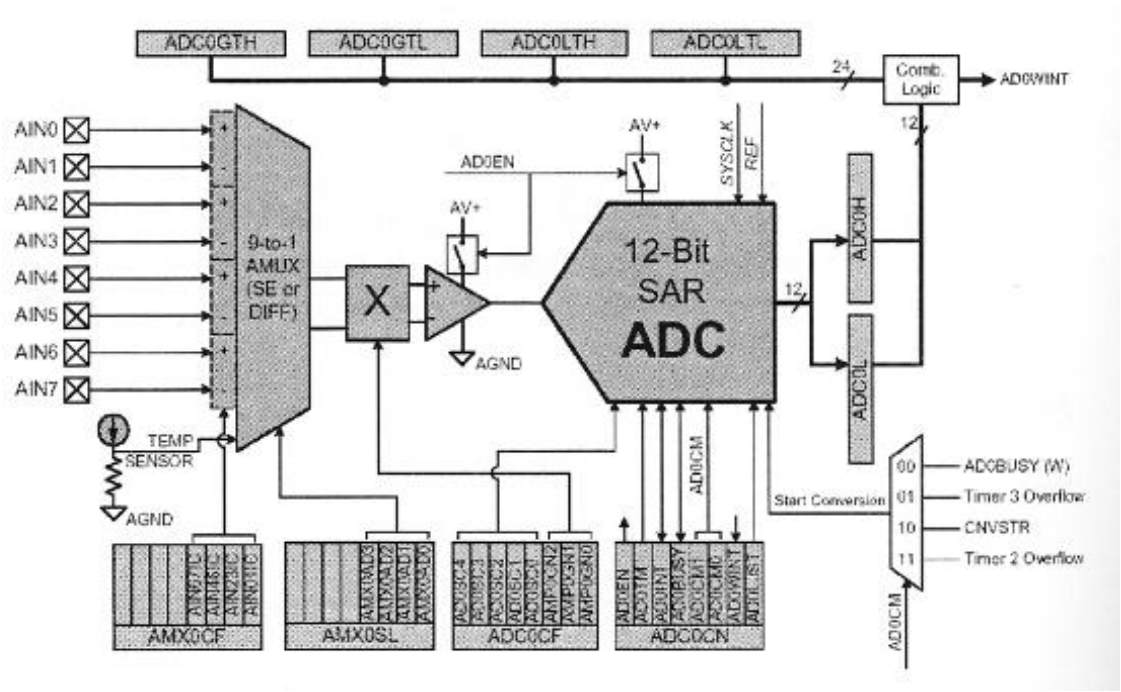


Figure 3-8: Functional Block Diagram a ADC [32]

An ADC has been used to receive the analogue signal from the joystick. The block diagram of the ADC is shown in Figure 3-8. There are several options available as to when and how to initiate the ADC conversion. These include, software command using AD0BUSY flag, overflow of timer 2 or 3, or an external signal (CNVSTR). For the purposes of the control system and to maximise the system's available resources, it was decided that software command was the best approach as this means the system resources were only used when a reading was needed. When the ADC is polled the value is written to the ADC register. This value is between 0 and 4095 as there are 12 bits available from the ADC. The joystick's zero output is in the middle of this range so this needs to be shifted and scaled in the software to produce the appropriate ranges of inputs to be transformed into the duty cycle of the motors, this is further explained in section 6.1.1.

3.6.2 LCD

LCDs (Liquid Crystal Display) provide added functionality to any system. For a large number of systems an LCD display is a critical piece of hardware. The main function of the LCD for this machine is to provide battery level feedback. Although this could be achieved by using a row of LEDs, an LCD provides many more diagnostic features for testing and maintenance. The Surplustronics LA0320 was used as this is the display used in the Universities development boards, providing a large amount of knowledge and personal experience. These displays consist of two rows capable of displaying 16 characters per row. They run off a 5V supply and feature a blue backlight. As the LCD would not be attached to the main microcontroller board a box was needed for it to be mounted in. Figure 3-9 shows the screen mounted in its plastic box. The LCD module was soldered to a PCB using plastic spacers to hold it in place. The signal lines were routed to a 16 strand ribbon cable which could be plugged into the main controller board.

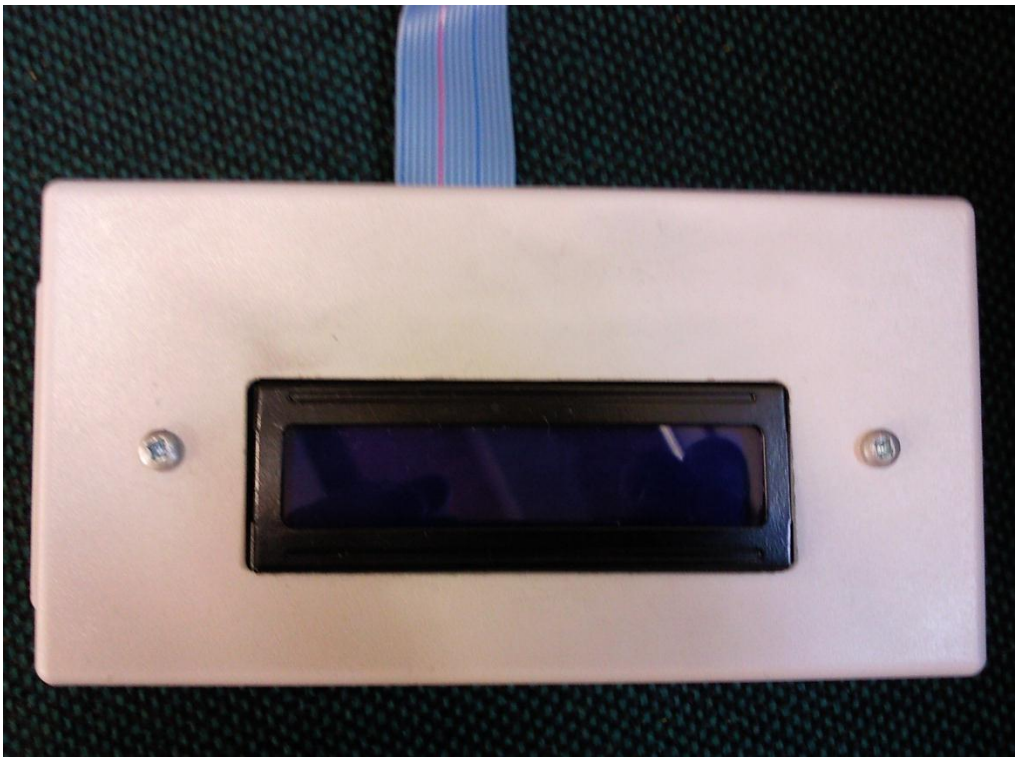


Figure 3-9: LCD mounted in plastic box

3.7 Motor Drivers

The motor drivers were a large undertaking of this project. These are of critical importance to enable smooth control of the platform, as proven by the drivers used on the original prototype.

3.7.1 Innovation First Victor 883

The “Innovation First Victor 883” was the motor driver used on the original prototype. It features a large 60A continuous current rating achieved through the use of fan cooled MOSFET (Metal Oxide Field Effect Transistor) H-bridge design. The strengths of this driver are its very good power to size ratio and its cost effectiveness. However, there are many weaknesses, the greatest of which being the 10% PWM output cut off. This resulted in “choppy” operation of the platform and eliminated the ability of diagonal movement as this requires very slow rotation of some wheels. Another weakness is the fact that the driver requires a fan for heat management. Fans are a continuously moving part that provides an additional possible mode of failure. Thus, from a reliability standpoint it is better to not need fan cooling. The final failing of the motor driver is the fact that it needs a RC (Radio Control) PWM signal to control the speed. This kind of PWM requires a 50Hz signal consisting of 1 to 2ms pulses that determine the speed of the motors, shown in Figure 3-10. A 1ms pulse will result in full speed reverse, while a 2ms pulse will be full forward with 1.5ms pulse being neutral (i.e. motor not moving). The problem with this system is that it is very difficult to recreate in the microcontroller. The 16-bit PCA (Programmable Counter Array) could be used to achieve the desired resolution, but a custom external clock would need to be implemented in order to achieve the required 50Hz signal.

Figure 3-10: RC PWM signal

Another approach would be to use a fast internal timer to create the signal. An interrupt was created at 100 kHz and a counter was used to divide this to a high resolution 50Hz RC PWM signal. It was important to make the timer interrupt high priority to ensure a clean signal to the motor driver. A development board was used to test this with a single driver and motor

and it was found to work. The problem with the 10% cut off was obvious, as was a problem of too much current drawn from the signal line causing a protection diode on the power supply to blow. This could be overcome with a signal buffering circuit but at this stage it was decided that the Victor 883 was not a suitable option for motor control for this application. Other designs would need to be investigated.

3.7.2 Commercial Motor Driver Options

The requirements for the motor driver are as follows:

- Must be able to handle at least 20A continuous current, must consider 120A stall current of motors
- Should accept traditional PWM signal of around 20kHz. This ensures smooth, quiet operation and direct accurate control over H-bridge configuration
- Should preferably not be reliant on fan cooling

There are not many commercially available motor drivers that meet the power requirements. Several options are listed below.

The Sabertooth 2x50HV is a 50A continuous current motor driver made by "Dimensions Engineering" [34]. This driver can control two motors using a wide range of control inputs. These include analogue 0-5V, RC PWM and two serial input modes. Unfortunately there is no direct PWM input, with the manufactures documentation [35] suggesting a low pass filter to the analogue filter. This will negatively affect the accuracy of the control signal. With RC PWM control ruled out, this leaves the serial interface. There is a simplified serial mode and a packetized serial mode available. Packetized serial will be investigated because of its higher resolution. In this mode many drivers can be driven off a single serial port by utilising an address byte of the packet sent. The next byte in the packet is the command byte that selects the motor to be driven as well as the direction. The data byte then determines the velocity of the motor as a number between 0 and 127. The packet is then ended with a checksum byte. The problem with this method lies in the resolution of the data byte. The 8-bit PWM output has a resolution of 256 levels while this serial output has half at 128. This essentially rules out this driver as a viable option.

3.7.3 OSMC Driver

In the search for a suitable commercial motor driver the Open Source Motor Controller [36] was found. This is an open source motor driver that is capable of delivering a large 160A

continuous current and up to 300A pulse current with a fan installed. This is achieved through the use of four parallel MOSFETs for each leg of the H-bridge arrangement.

The board is based around a HIP4081A H-bridge driver chip [37]. This provides all necessary signals to drive both the high and low side MOSFETs as well as the required voltage boosting circuitry for the high side.

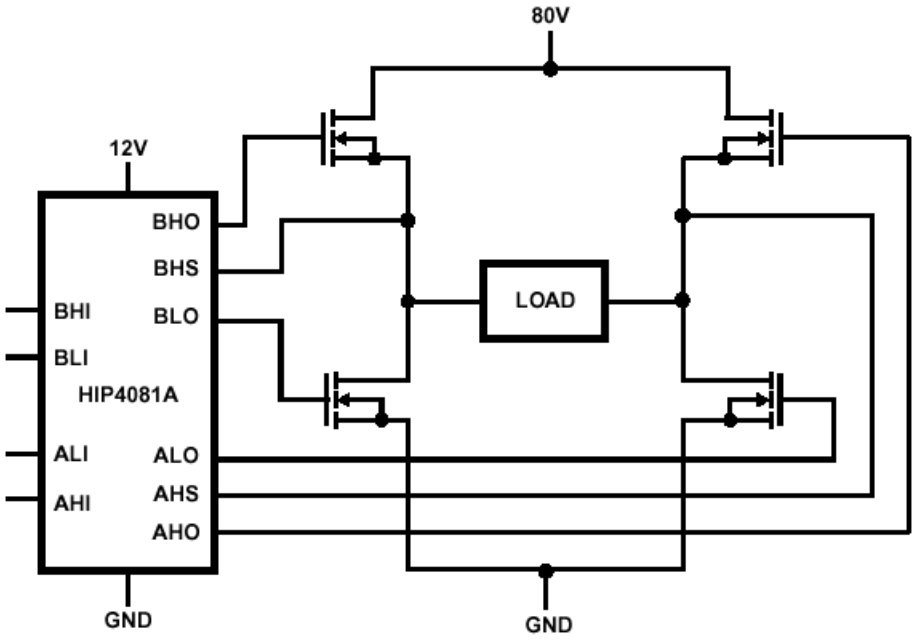


Figure 3-11: HIP4081A block diagram [36]

The block diagram in Figure 3-11 shows how the HIP4081A is connected to an H-bridge. The AHS and BHS pins allow the driver to produce a voltage equal to the source voltage +10V that is required to turn the high side FETs on; in this case 90V. This is achieved through the use of an external diode and capacitor in a capacitor switched charge-pump system, shown in Figure 3-12. Other features of this chip include a disable pin, timing delay pins to ensure there is no shoot through when switching, and inbuilt logic that makes it impossible to have a shoot-through situation, by turning off the high side if the corresponding low side FET is turned on.

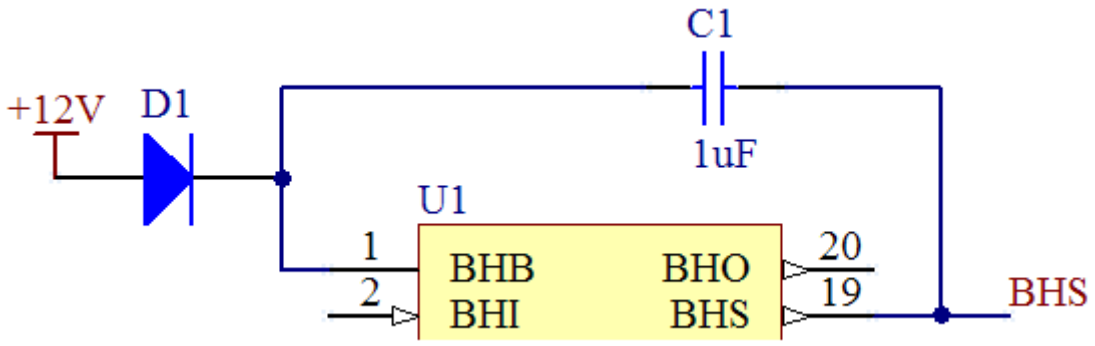


Figure 3-12: Charge pump circuit

In order to achieve the very high current rating of the OSMC, parallel MOSFETs had to be used. The circuit for these is detailed in Figure 3-13. The MOSFETS are arranged in parallel by connecting the drain pins together and the source pins together for each leg of the H-bridge. Although the gates are also connected together there is some additional circuitry present. Because of the ever present danger of shoot through and the tendency for MOSFETs to turn on faster than turn off, due to gate capacitance, additional protection against this is necessary. The resistor and diode circuit shown on each gate ensures that the FET can turn off fast, through the diode, and turn on a little slower, through the resistor. The resistors also aid in limiting the gate drive current, which, with the combined capacitance of four MOSFETs, can be large enough to damage the HIP4081A chip. Also two zener diodes are used to clip the high and low voltages of the gates to ensure they are not damaged by a voltage spike.

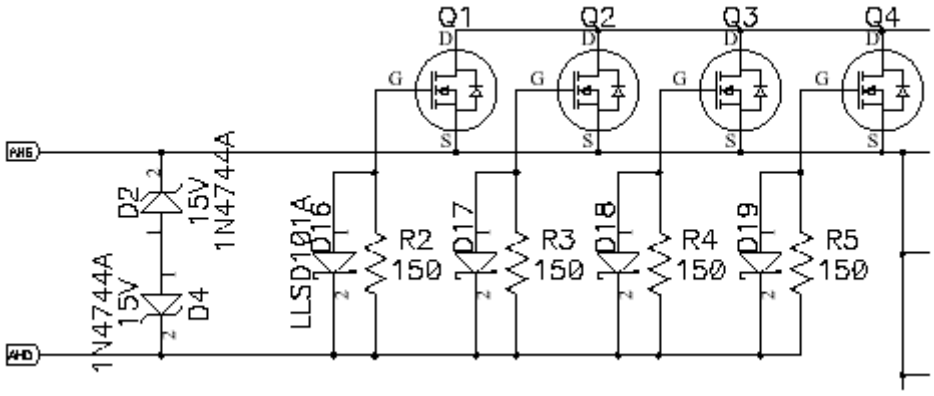


Figure 3-13: OSMC Gate Drive [36]

Other features of the OSMC include the use of Transient Voltage Suppressors (TVS) which handle any voltage spikes that come from inductive loads and can potentially exceed the

FETs drain to source breakdown limit. A RC-snubber is also used to filter any high frequency noise.

Although this driver meets most of the requirements for the desired motor controller, including direct PWM control, high current capabilities and a fan not required at the desired current, it is still missing some important features of most commercial controllers. Current sensing for over current protection is the most important missing aspect, with the ability to detect when the driver is drawing too much current so that it is possible to shut down and avoid any damage. Smarter control circuitry would also be useful; the current design involves having two PWM signals, one for forward and one for reverse. It would be better to have a single PWM input with a directional control line. Another negative aspect of the driver is the fact that it is over engineered for the required purposes. The OSMC can handle a continuous current of 160A, when only 20A is required; this can lower the overall cost and size of the driver board.

Because the OSMC is open source, the design can be modified to suit the desired application and any extra features can be added to enhance the functionality of the design. Because of this a single pre-fabricated unit was purchased to test and observe if it would be suitable for the application.

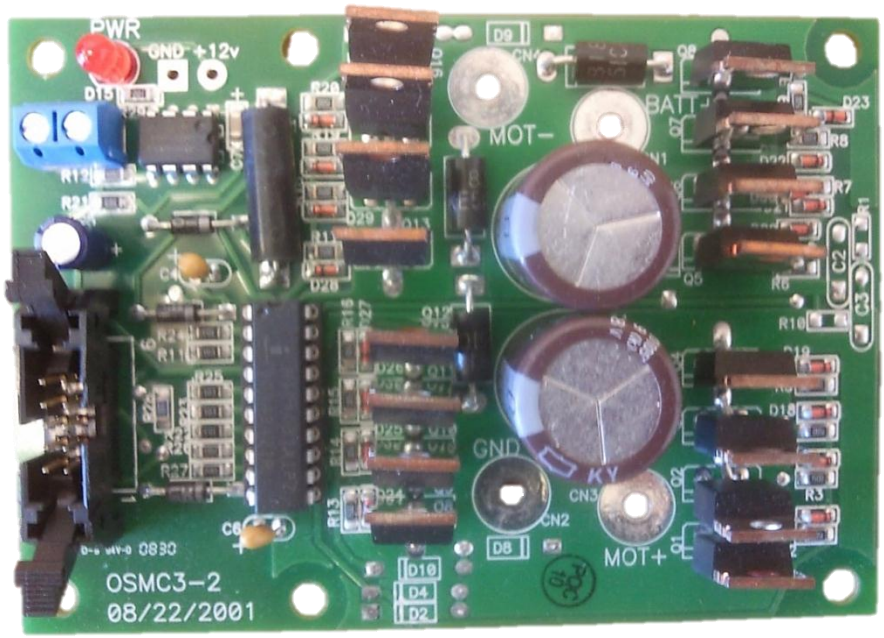


Figure 3-14: OSMC pre-fabricated unit

This motor driver, shown in Figure 3-14, was first tested with a smaller motor using a breadboard to provide the four signals to the driver from the PWM and directional outputs of the microcontroller. The board proved to work correctly and had no problem controlling the motor. A joystick was integrated into the system and the driver continued to perform with the variable speed input. This trend continued when one of the larger motors was used with a 24V battery source.

Because of the satisfactory performance of the driver OSM, it was decided that designing and building an in-house driver, based on the OSMC, would provide the best performance and functionality for the platform.

3.7.4 H-Bridge Design and Control

The main modification made to the H-bridge design was the halving of the number of transistors. Because the OSMC was rated at 160A (with fan) it is safe to assume that two FETs, fan cooled, would handle 80A continuous current. If the fan was removed a continuous current of 40 – 50A could still be achieved. Considering the motors used are rated at 20A continuous current, this arrangement will provide plenty of head room.

The H-bridge layout is shown in Figure 3-15. The transistors used were the "IRF1407" from International Rectifier (130A, 75V, 0.0078Ω). Two transistors per leg were used and several other adjustments were done to the circuit. The first is the lack of a resistor, diode (R1 – R4, D1 – D4) gate circuit per transistor, now there is only one per leg. This is because all the gates are driven off the same source on each leg so having one per transistor is not necessary. The next adjustment is the inclusion of a zener diode per transistor. Because of the reflection present at the termination of transmission lines it is very important to have these zener diodes (D5 – D12) as close as possible to the legs of the transistor gate to avoid damaging the FETs. The other modification was to add flywheel diodes to the H-bridge configuration (D13 – D16). Because of the questionable quality of the internal flywheel diodes operating in the IFR-1407 MOSFETs extra flywheel diodes were also installed to aid in the conducting of the flywheel current.

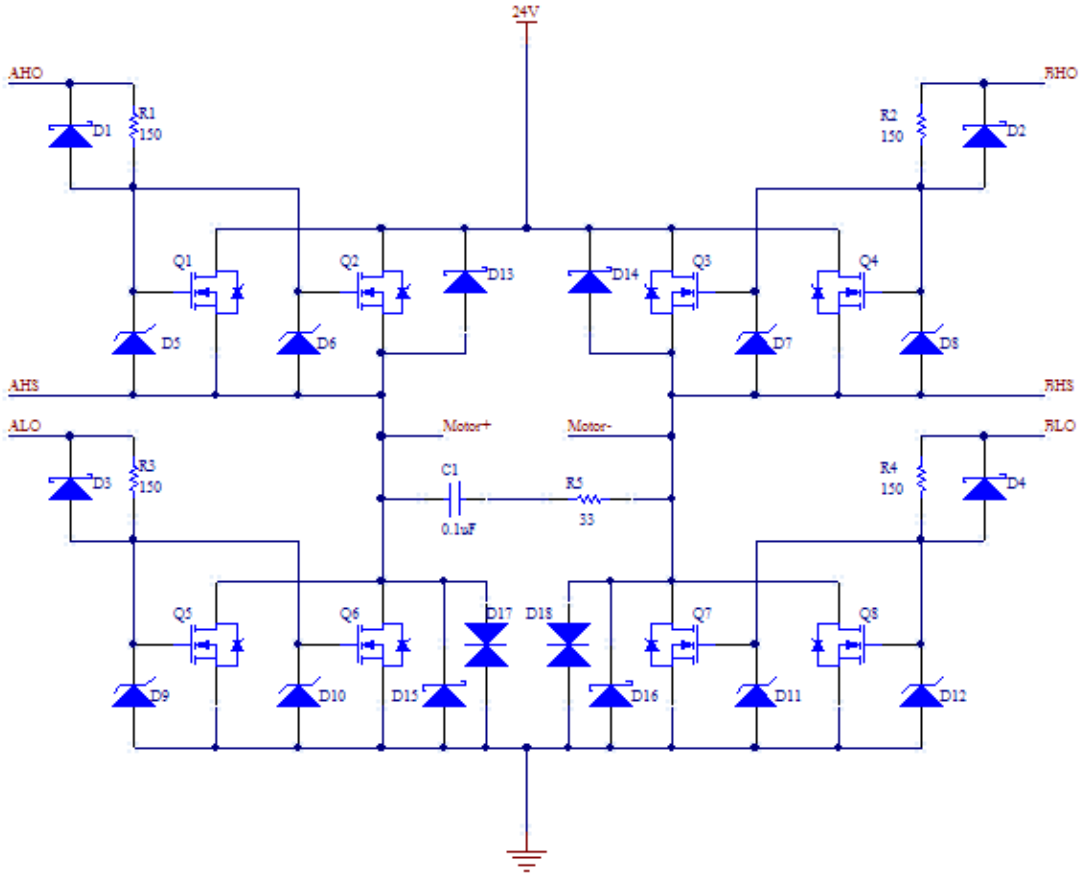


Figure 3-15: H-bridge layout of new design

Other features not shown include a large 60A diode in series with the positive battery terminal to protect against incorrect wiring. Two large parallel capacitors between the battery terminals help provide smooth power to the rest of the circuitry. There is also a 12V linear regulator used to provide stable power to the HIP4081A.

The control circuit for the driver involves converting a PWM signal and a direction line into the four control signals required by the HIP4081A IC. In order to enable the alternative speed feedback discussed in section 3.7.5 the driver needed to be freewheeling rather than braking when the PWM was zero. The best way to do this was to drive the high side of the H-bridge and tie the low side depending on the directional input. This can be achieved through the use of NAND gates to produce the four desired outputs detailed in Figure 3-16.

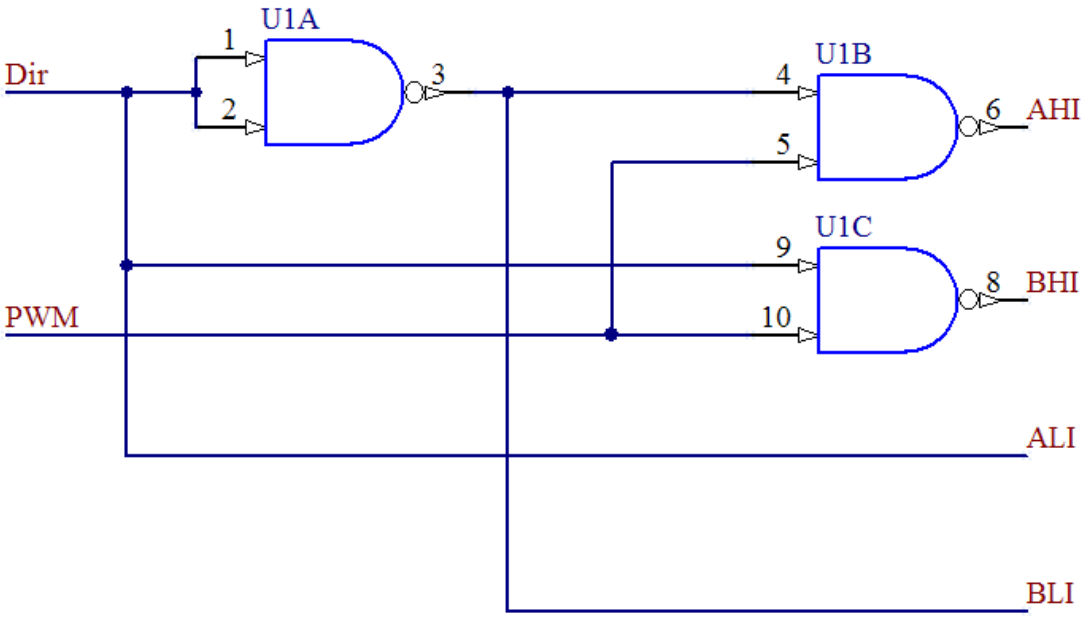


Figure 3-16: NAND gate configuration of PWM handling

Because there was other logic involved in the functions of the driver board it was decided to use a Programmable Logic Device (PLD) to handle all the logic. This provides the most flexible solution and allows more advanced features to be integrated such as a brake.

The braking incorporated into the driver relies on the fact that when the motor terminals are both connected to the high voltage or ground the motor brakes. Figure 3-17 shows the logic required to integrate the brake. This uses a AND, OR and a NOT gate, which will be easy to implement in the PLD on the driver board. The brake can also be pulsed to control the strength of the brake.

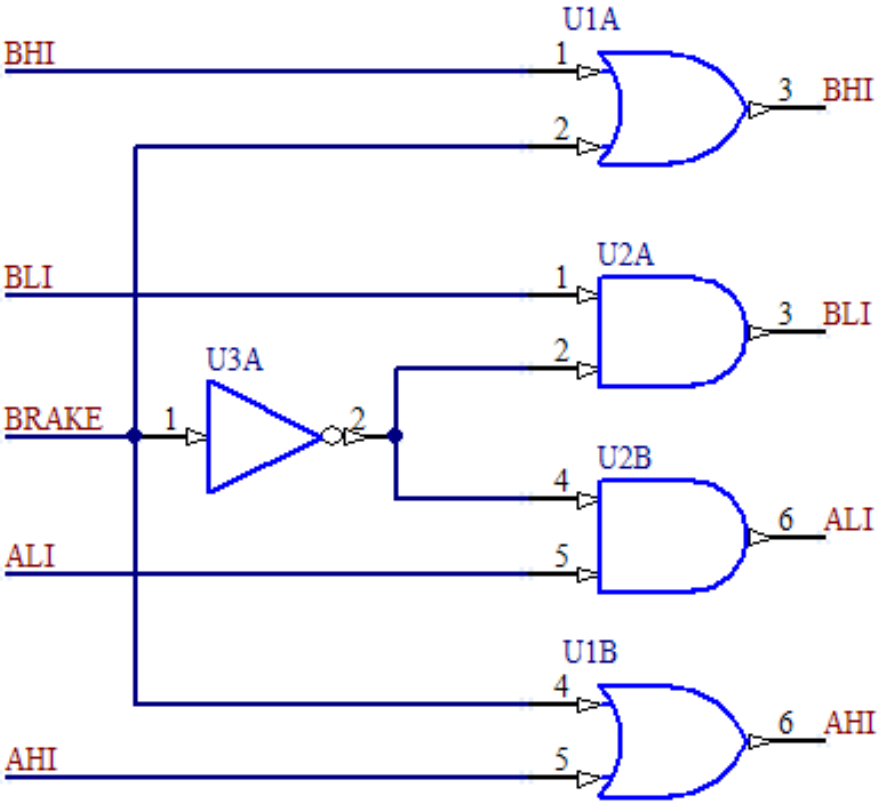


Figure 3-17: Logic required for braking of motor

3.7.5 Alternative Speed Feedback

Examples of sensor less motor feedback being a successful alternative are shown in [38], [39] and [40]. The alternative speed feedback incorporated into the motor driver is based on measuring the back EMF of the electric motor while freewheeling. As it is driven by a pulse width modulated (PWM) signal through the H-Bridge arrangement, the voltage across the motor terminals can be measured when the pulse is low; this voltage is proportional to the speed at which the motor is turning. However, it was found that because the PWM signal used has a period of 50μs this method does not work as expected. This is due to the large inductance values of the motors being used. This causes the current to continue flowing for the duration of the off period. By periodically shutting the PWM signal off for an appropriate length of time, in this case 400μs, the current decays and the voltage can be measured. Figure 3-18 shows the back EMF period, followed by a period in which the voltage settles to a certain level which can be read by the microcontroller through an ADC (shown by the short sampling pulse).

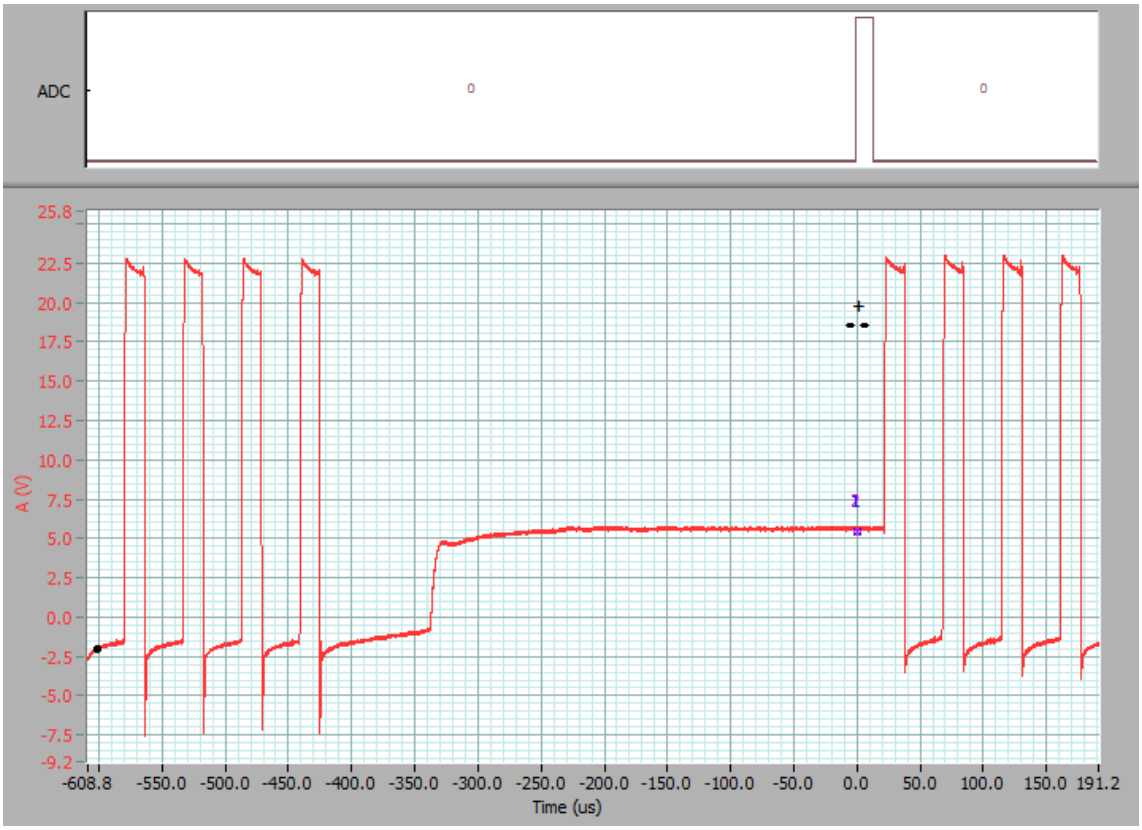


Figure 3-18: M2 terminal of rotating motor, with freewheel voltage reading

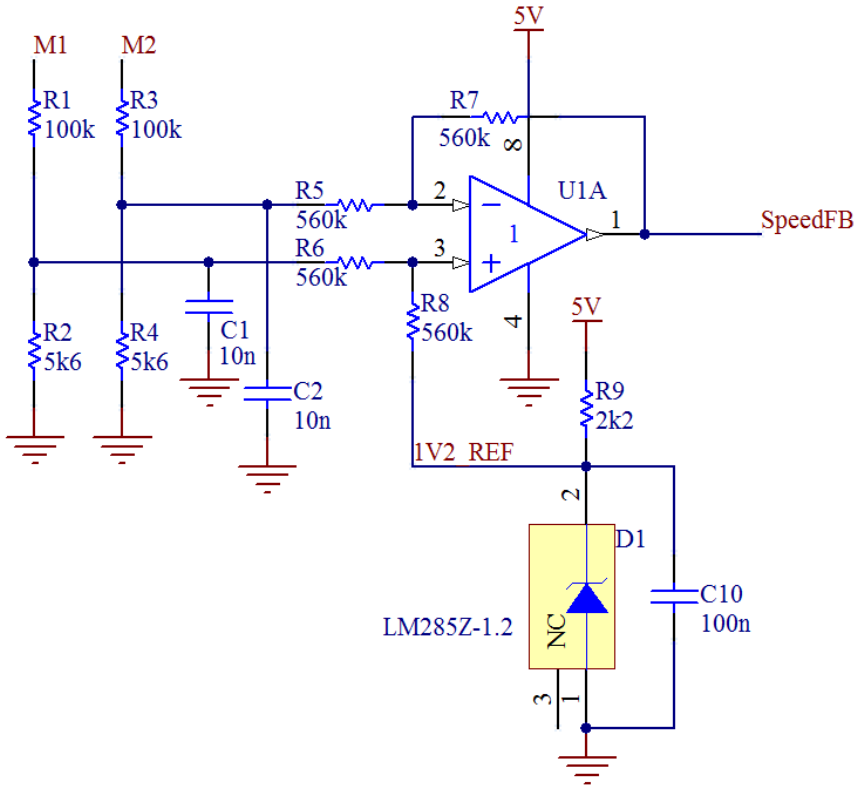


Figure 3-19: Voltage speed feedback circuit schematic

To determine the direction of the motor rotation it is essential to know which terminal is generating the voltage. In order to save Microcontroller resources it is also desirable to contain this information on a single line going to a single ADC input. This is achieved by utilising a difference amplifier with a 1.2V offset. Figure 3-19 shows this circuit. The voltages from the motor terminals is dropped to workable levels by two voltage dividers and then smoothened by capacitors C1 and C2. This is then fed into a unity gain difference amplifier with a 1.2V band-gap reference producing an offset for the amplifier. This causes the voltage at the output of the amplifier to be 1.2V when the motor is stationary. The SpeedFB voltage varies based on the rotational speed and direction of the motor, between 0 and 2.4V to conform to the microcontroller's ADC input requirements.

With a test board completed (section 3.7.7) experimental testing was carried out. This was to test the feasibility of the voltage speed feedback compared to the more traditional quadrature encoder feedback. The test was carried out using one of the "Allied Motion" RAD right angle motors with the mounted encoder. The motor was connected to the driver which was connected to the 24V battery source and a microcontroller development boards. The speed feedback will be compared in two ways. The first will be a direct comparison between the two systems in order to determine if there is a proportional relationship between them. The second comparison will focus on the variation in the two systems when the motor is set to rotate at a constant speed.

To compare the speed feedback systems the two datasets need to be normalized. This makes a direct comparison between the variation of the values possible. In this case the resolution of the voltage feedback is limited by the resolution of ADC0, which is 12 bits. This is much higher than the resolution of the encoders when read every 50 milliseconds; the voltage value was divided down to match the encoders.

The results shown in Figure 3-20 were obtained by incrementing the 8-bit PCA by 10 up to 250, resulting in 26 evenly spaced PWM signals. For each PWM increment 30 values of both the encoder reading and the voltage level were sent over a serial port to a computer for analysis. The averages of these values are used in this comparison.

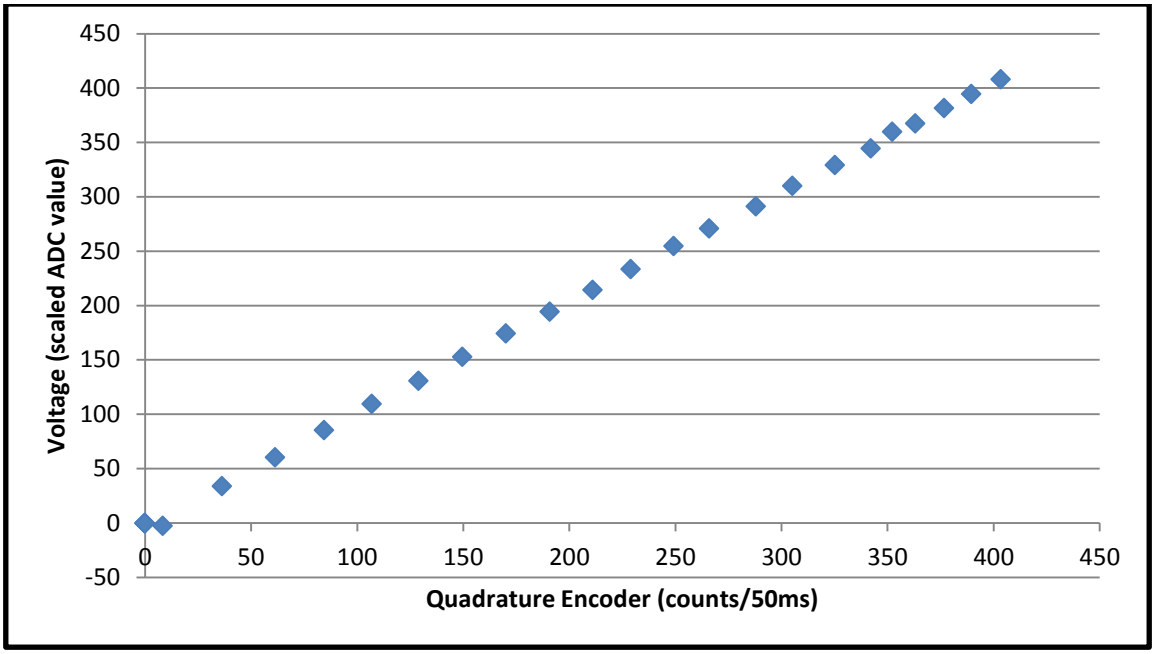


Figure 3-20: Direct Comparison between Encoder and Voltage Speed Feedback

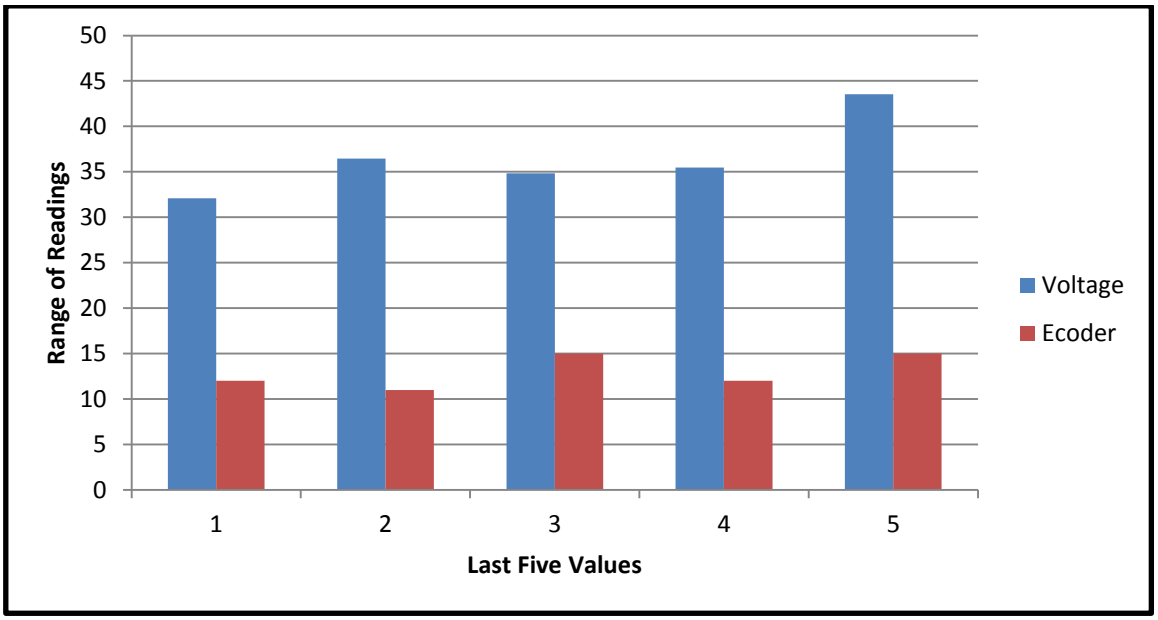


Figure 3-21: Comparison of variation between Encoder and Voltage Speed Feedback

From the plot shown in Figure 3-20, it can be seen that there exists a linear relationship between the two speed measurement systems. From this it could be concluded that the voltage speed feedback could be used in place of a traditional encoder. However, there exists an abnormality in the dataset occurring at very low speeds; the voltage drops slightly as if the motor is spinning in the opposite direction. The reason for this irregularity is unclear. Although the average of the values shows promise, the variation in the two systems also needs to be investigated to ensure a viable system. The plot shown in Figure 3-21 reveals the

advantage of the quadrature encoders being three times more stable in their output, especially at higher speeds. This can cause problems when implementing PID control as the control algorithm's goal is to keep the speed constant. One way to overcome this problem is to use some kind of averaging in an attempt to eliminate some of the variations. As the motors already have quadrature encoders fitted it may seem unnecessary to implement this form of speed feedback. However, by having two sensors, a form of redundancy is achieved. This can be used to periodically check compliance between the two separate systems to assess the overall performance. This information can be stored for later analysis to ensure that the motors and drivers are performing as expected or in a worst case scenario the platform can be shut down to avoid potential damage. Speed feedback also has safety applications such as limiting the speed based on specific operating conditions or driver board malfunction, both of which can benefit from added redundancy. Unfortunately as explained in section 3.7.8, a problem arose that caused this type of speed feedback to be deactivated. To investigate this form of speed feedback further, significant changes will need to be made to the board design (also explained in section 3.7.8); therefore no further investigation was possible at this stage.

3.7.6 Current Sensing

Current sensing is an important aspect of any motor driver. It not only provides important control information about torque output of the motors but also has safety implications when dealing with potential fire risk involved with high current applications. There are two commonly available options for enabling current sensing capabilities. The first is the shunt resistor method detailed in section 2.3.2. Specially designed low Ohm, high wattage resistors are used to provide a voltage drop proportional to the current passing through. Unfortunately resistors that meet the high current requirements of the motor drivers were not readily available. The other current sensing device that was available was a Hall Effect based device; this was capable of high current sensing (up to 200A).

The Allegro ACS756 is a Hall Effect based current sensing IC. It works on the principles outlined in section 2.3.2 and [19, 20]. The IC works by converting the magnetic field produced by the unknown current to a proportional voltage which is isolated from the current. The details of the device are given in [41]. The model used in the driver board is capable of sensing currents up to 100A.

Because the motor is driven by a PWM signal, current is flowing only during the 'on' phase of the signal. To ensure a correct reading of the current, it is important to measure it at the

right time; this occurs at the end of the "on" pulse. To achieve this timing it is necessary to produce an inverted PWM signal ($\overline{PWM}$) and a delayed PWM (PWM') signal which are then fed through an AND gate to produce a pulse indicating when to measure the current. The circuit used to achieve this is shown in Figure 3-22 and a plot of the various signals is shown in Figure 3-23.

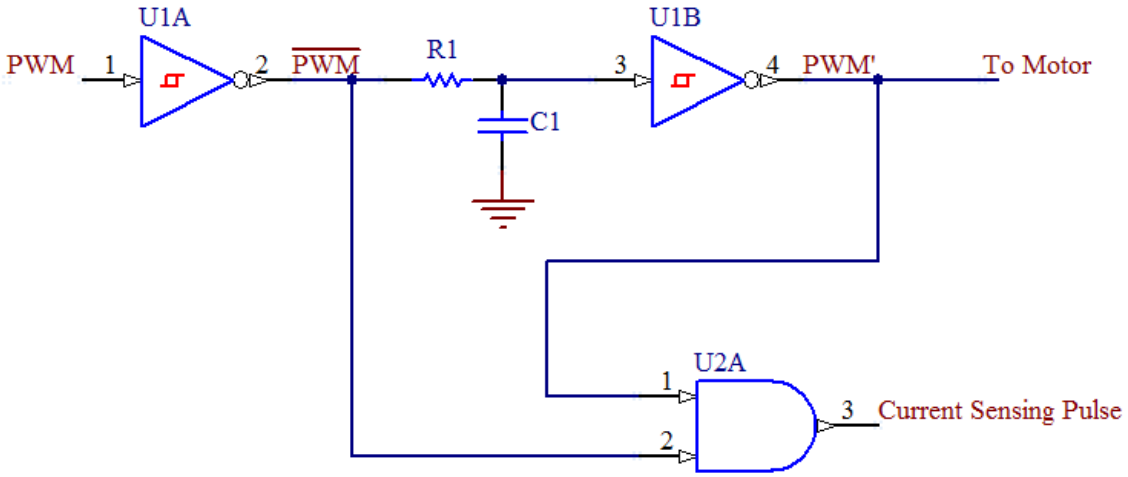


Figure 3-22: Circuit schematic of the current sensing pulse

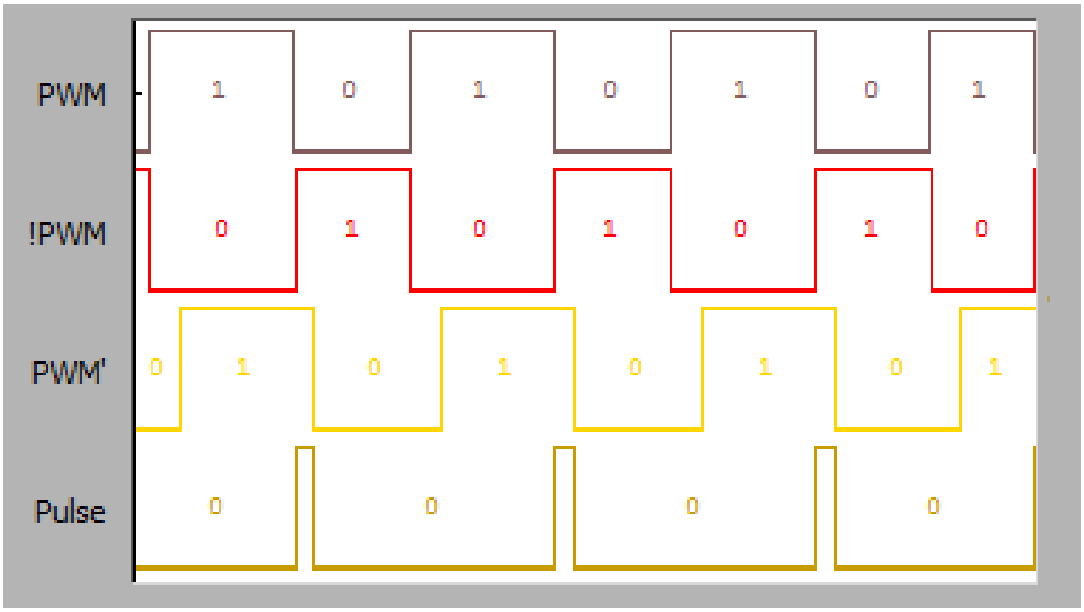


Figure 3-23: Output of the current sensing circuit

With this current sensing pulse, it is now possible to construct a circuit that can provide an appropriate output to the central microcontroller. Figure 3-24 shows such a circuit.

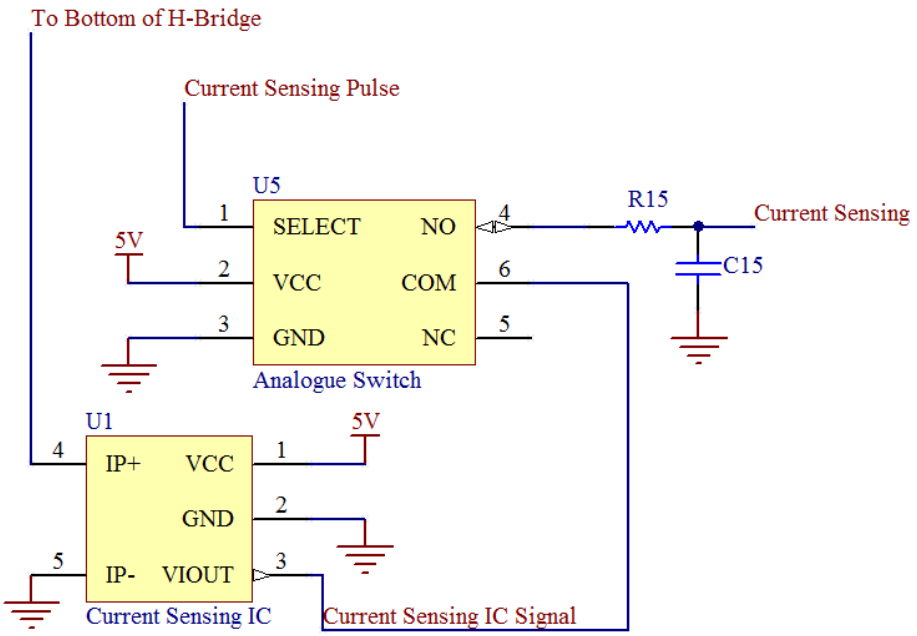


Figure 3-24: Circuit schematic providing voltage proportional to the motor current drawn

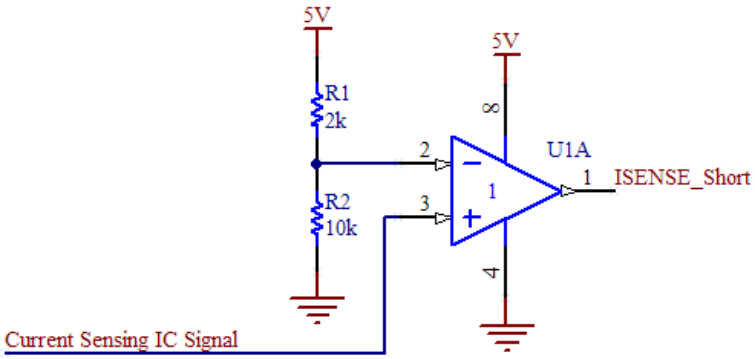


Figure 3-25: Circuit schematic providing fast response signal for current spikes

Through the use of an analogue switch and the "Current Sensing Pulse" (Figure 3-24) to connect the current sensing IC to the sample and hold capacitor the voltage reading from the output of the current sensing IC can be stored where it may be read "at will" by the microcontroller. As this is a relatively slow feedback system that is not constantly monitored, it is important to have an alternative system that can respond quickly to any potentially damaging current spikes. For this an operational amplifier was used as a comparator to compare the output of the current sensing IC to a reference voltage. When this voltage is exceeded, a signal is sent to a high priority interrupt on the microcontroller from where the necessary shutdown procedures can be executed.

3.7.7 Board Layout and Assembly Process

Because there is a need for all boards to receive a delayed PWM and an inverted PWM it was decided to produce these signals off board. The final header pin layout consists of:

- 5V
- GND
- Delayed PWM
- Inverted PWM
- Direction
- Brake
- Current sensing
- Current short sensing
- Disable
- Voltage Speed Feedback

There were several considerations that needed to be adhered to in the layout of the components. The most important of which was the track size in the high current zones of the board. This means that the MOSFETs need to be carefully arranged to allow the large pours of copper to be organised around them. This layout and the large polygon pours around the H-bridge can be seen in Figure 3-26. Both the top and bottom layers are used in the high current areas with a large ground track around the outside connecting to the battery terminal. The other important consideration was the fact that the through-hole vias would not conduct through to the other side, due to the in-house board manufacturing method used. This meant that any vias on the board need to be manually conducted with wire. This also causes any through-hole components that cannot be soldered on both sides to only be able to be connected from one side.

For initial testing one driver board was made. The finished product is shown in Figure 3-27. Because of the nature of the via obtained by this process, some of the through hole components need to be soldered on both the top and bottom of the board. This proved to be difficult in some instances. The overall process is aided by following a set order of assembly. This is listed below:

- Drill all holes to the appropriate size
- Solder all surface mount components moving from smallest to largest

- Solder all through hole components moving from smallest to largest

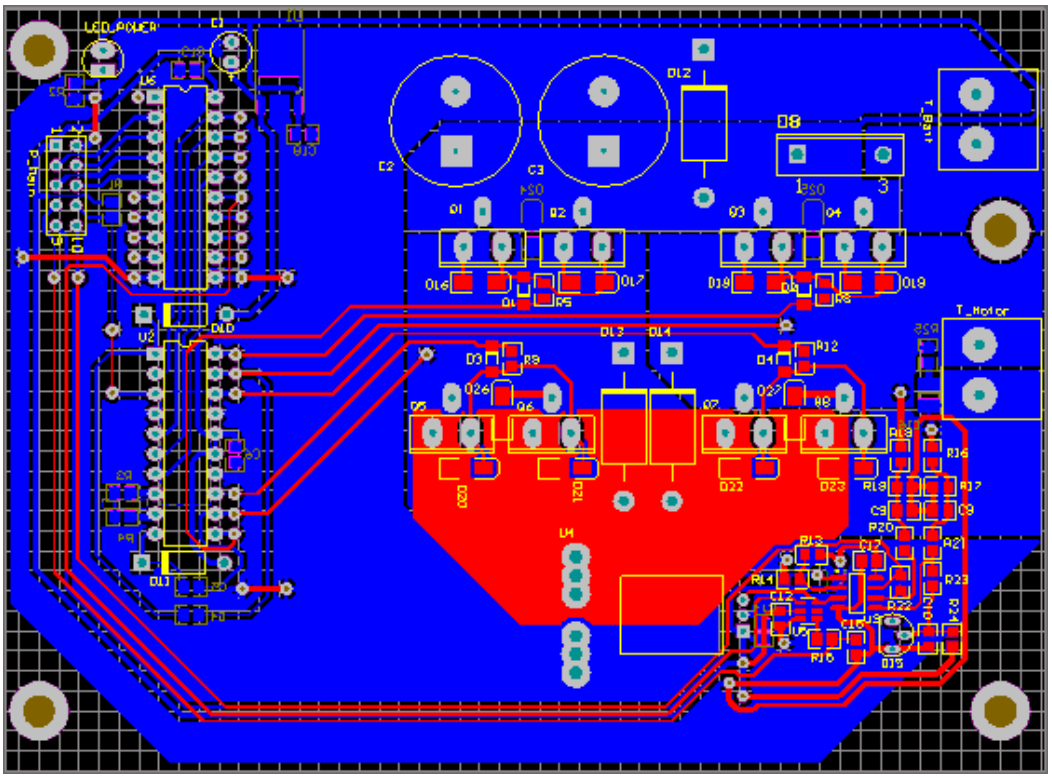


Figure 3-26: Printed Circuit Board layout

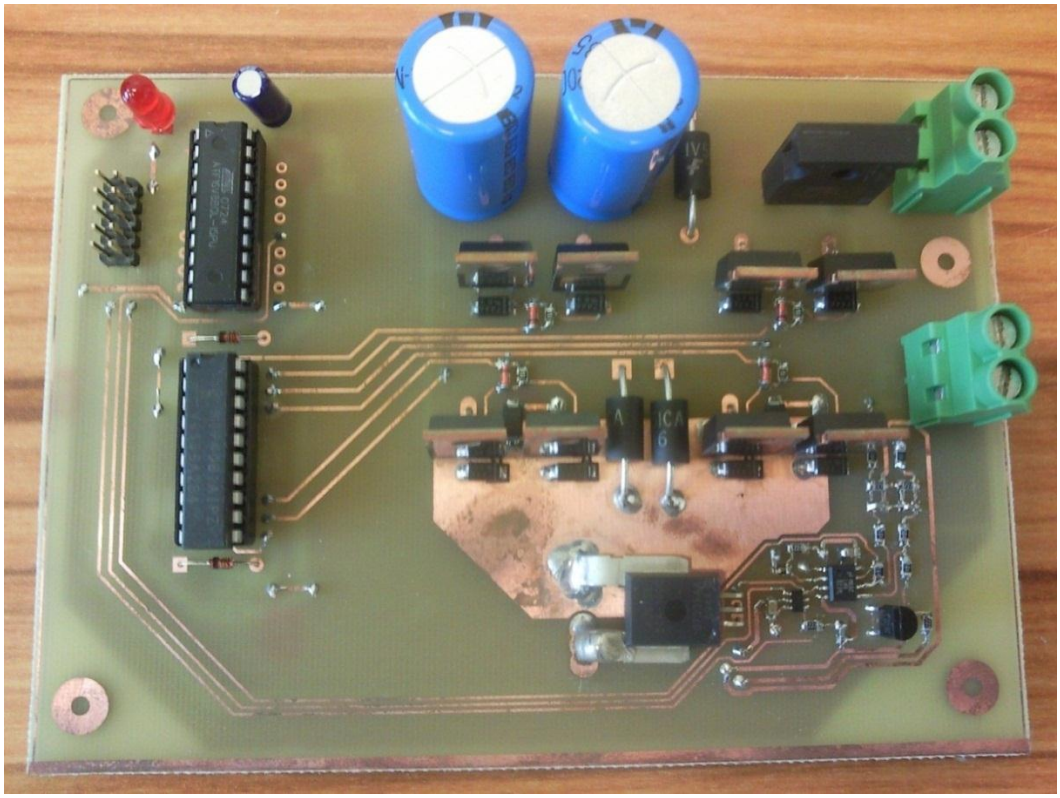


Figure 3-27: Fully assembled driver board

At this stage the PLD also needed programming. Figure 3-28 shows the code that was used to program the PLD. This code follows the logic for the PWM handling, braking and current sensing outlined in Figure 3-16, Figure 3-17 and Figure 3-22.

```
/* *************** INPUT PINS *********************/
PIN  2    = PWM;
PIN  3    = NOTPWM;
PIN  4    = DIR;
PIN  5    = BRAKE;
PIN  11   = !OE;

/* *************** OUTPUT PINS *********************/
PIN  15    = ISENSE_Pulse;
PIN  16    = AHI;
PIN  17    = ALI;
PIN  18    = BLI;
PIN  19    = BHI;

ISENSE_Pulse = PWM & NOTPWM;
AHI = (PWM & !DIR) # BRAKE;
ALI = DIR & !BRAKE;
BLI = !DIR & !BRAKE;
BHI = (PWM & DIR) # BRAKE;
```

Figure 3-28: PLD code for motor driver

3.7.8 Driver Testing and Modification

The initial testing of the driver consisted of running a smaller motor from a bench power supply with a PWM signal provided from a microcontroller development board. A bread board and two breakout boards were used to provide the required inverted and delayed PWM signal and route the signals to their corresponding pins, as well as receive and divide the voltage, and route the joystick inputs. This arrangement is show in Figure 3-29.

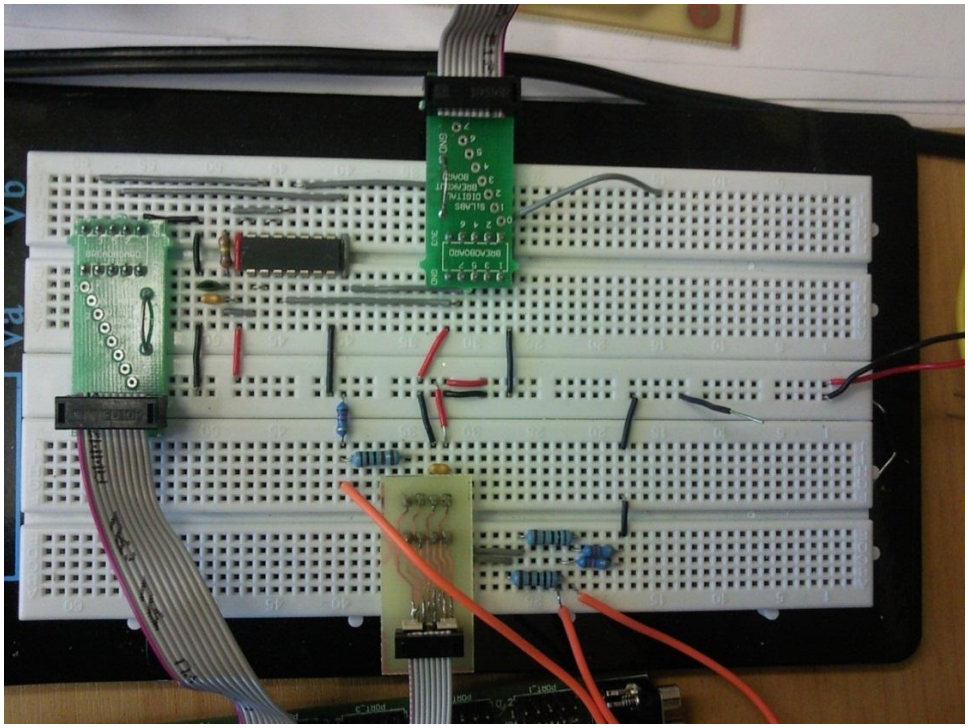


Figure 3-29: Driver test circuitry

The driver performed this task well so it was then tested with the "Allied Motion" RAD right angled motor and the 24V battery source. The driver also performed well under these conditions. From here it was decided to make the other three boards. Each board followed the same testing procedure as the first and each performed correctly. Two drivers were wired to motors, battery and controller to test how running two at once would respond. This also went well.

The next big test for the motors came when they were mounted on the platform and ready to be run on the platform. With the platform on blocks so that the wheels were under no load all four drivers performed as expected. When the blocks were removed and the wheels were put under load the drivers continued to perform. It was not until the platform had been driven for a few minutes, stopped for a few minutes and then started again that one of the drivers MOSFETs blew. Upon investigating the driver board it was found that the flywheel diodes had been the cause of the problem. The diodes had either overheated, melting the solder and moving off their pads or blown the small track that was running between them and the MOSFET.

The reason for this behaviour lies with the notion that diodes do not share current when in parallel unless they are a matched pair or have a current sharing resistor. Because the

flywheel diodes had a lower voltage drop than the internal diodes they took the full flywheel current and were not rated high enough to handle this. To overcome this problem all these flywheel diodes were removed and the damaged MOSFET, believed to be caused by the heat created by the nearby diode, was replaced. It was inferred that the internal diodes are large enough to handle the flywheel current.

Low quality internal diodes have the potential to create failures. It is possible to drive the H-bridge in such a way that minimises the load carried by the diodes. With the present driving method, the high side of the H-bridge is driven. Figure 3-30 can be used to explain this. To make the motor (M) travel in the forward direction the AH switch is pulsed (speed control) and the BL switch is turned on while AL and BH are left in the off position. This means that when the driving pulse is in the off stage the flywheel current is going through the diode at AL and the closed BL switch. However, if the AL switch was turned on when the AH switch was off the flywheel current would have a path to follow without relying on any diodes. It turns out that the inbuilt logic of the HIP4081A driver IC will handle this itself if the low side terminal is driven and both high sides are tied to high.

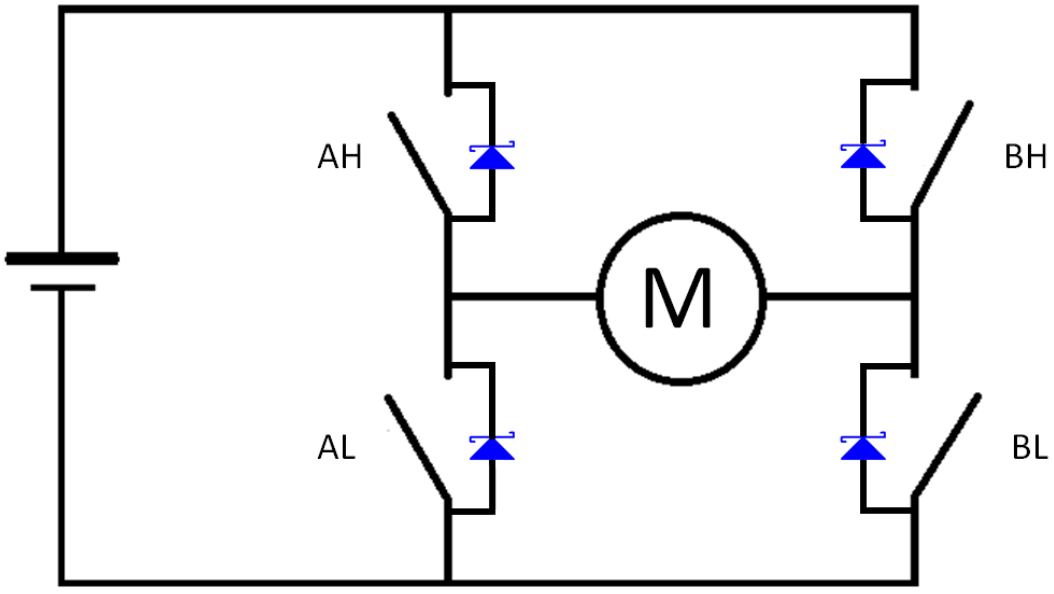


Figure 3-30: Simplified H-bridge driver

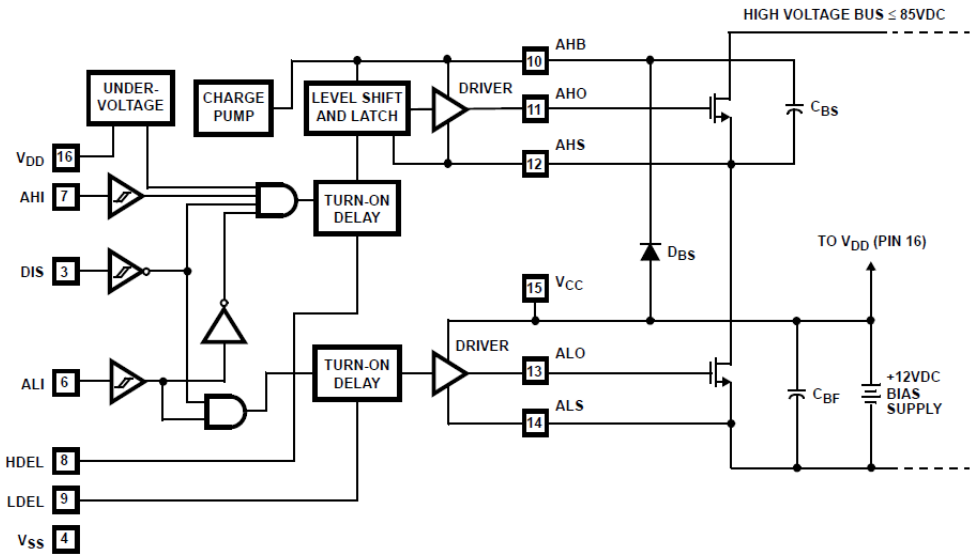


Figure 3-31: Internal protection logic of H-bridge driver IC [37]

This inbuilt logic exists to protect the H-bridge from being driven into a shoot-through condition. This logic ensures that if the low side is turned on the high side output is off no matter what the high side input reads (Figure 3-31).

With this logic the high side switches (AH and BH) will be on when the motor is not being driven causing the flywheel current to flow through them, putting the motor in a braking mode. This was an easy modification to make, requiring only the PLD code to be adjusted.

```
/* *************** INPUT PINS *********************/
PIN  2    = PWM;
PIN  3    = NOTPWM;
PIN  4    = DIR;
PIN  5    = BRAKE;
PIN  11   = !OE;

/* *************** OUTPUT PINS *********************/
PIN  15    = ISENSE_Pulse;
PIN  16    = AHI;
PIN  17    = ALI;
PIN  18    = BLI;
PIN  19    = BHI;

ISENSE_Pulse = PWM & NOTPWM;
AHI = 'b'1;
ALI = PWM & DIR;
BLI = PWM & !DIR;
BHI = 'b'1;
```

Figure 3-32: Modified PLD code

The modified code is shown in Figure 3-32. There are two disadvantages with this method. One when the motors are not being pulsed they are always braking. This means there is no longer a need for the brake feature. Secondly because the two terminals are tied to the positive battery terminal there is never a freewheeling state to measure the back EMF voltage to determine speed. Therefore, this feature can no longer be used in this iteration of the driver boards. However, future improvements to the board could re-establish the voltage speed feedback; these are discussed in section 8.1.

While investigating the reason for the blown MOSFET, using the "CleaverScope" USB oscilloscope, it was found that the switching of the MOSFETS was not as clean as it should be.

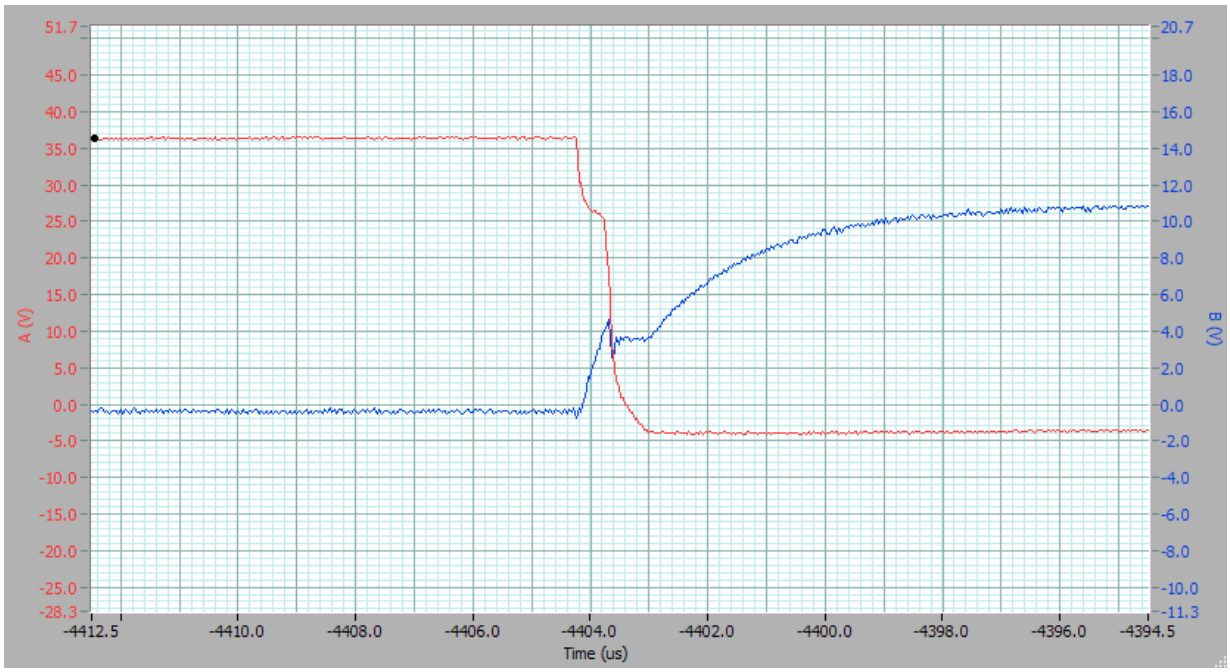


Figure 3-33: Gate voltages of B side MOSFETs OSMC reference board

The noise shown in Figure 3-34 is far more prominent than that of the reference board in Figure 3-33. This is due to the fact that the gate legs of the MOSFETs are directly connected in the in-house board causing gate ringing, where as in the reference board there is a resistor and diode circuit for each MOSFET. A simple way to compensate for this without making major modifications to the boards is to use a ferrite bead. The beads encircle the gate leg of the MOSFETs and provide dampening to any reflective noise. The result of using the beads can be seen in Figure 3-35.

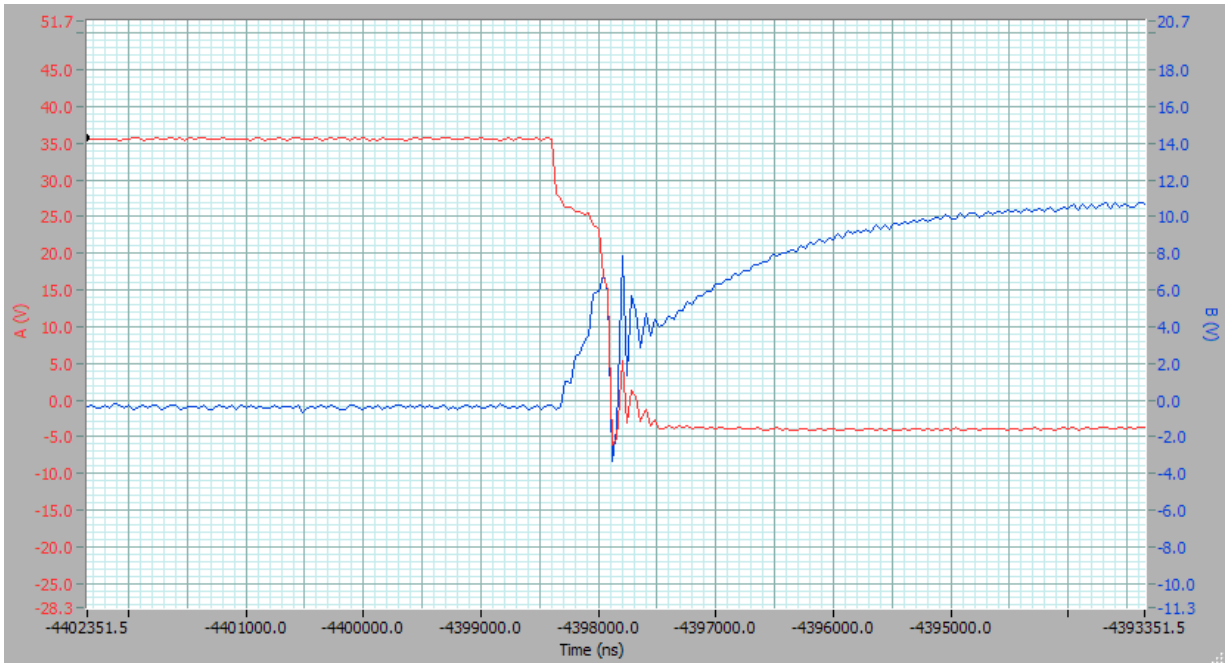


Figure 3-34: Gate voltages of B side MOSFETs in-house board

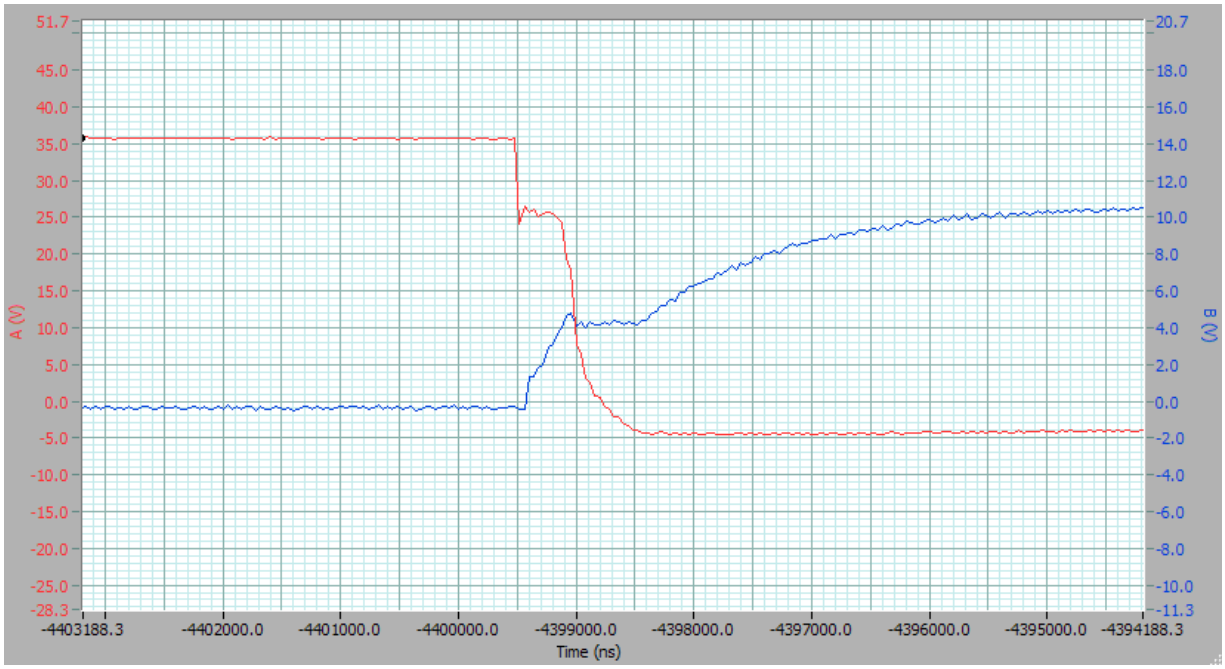


Figure 3-35: Gate voltages of B side MOSFETs in-house board with ferrite beads

With these modifications the drivers are now fully functional.

3.8 Inclination Sensor

Because this platform will be interacting with patients in a hospital setting, safety is a key design consideration. Part of this involves providing hill descent functionality. To achieve

this there needs to be some form of inclination sensing. There are several different inclination sensors available on the market.

The Memsic MXD2020EL [42] is a PCB mountable accelerometer that can be used to detect the change in angle with reference to the gravitational pull of the earth and therefore can be used as a tilt sensor. This sensor has the advantage of being able to be mounted on a PCB board and therefore be integrated into the control circuitry very easily, i.e. without external mounting or wiring. This sensor outputs a PWM signal that is proportional to the acceleration it is experiencing. The disadvantage with this sensor is that it is very sensitive to any movement especially vibration. This could prove to be a major problem, with a Mecanum wheel platform known to produce vibration if the wheels are not manufactured to very strict specifications. It may be possible to be overcome this problem with appropriate digital filtering but there is another, more robust, option.

The Zhichuan Electronics ZCT245AL-TTL Two-Axis Tilt Sensor Module provides inclination sensing abilities in a rugged, fully enclosed, industrial module. This module is a two axis tilt sensor that communicates using enhanced TTL serial communication protocol. It communicates by sending and receiving either hexadecimal or ASCII characters. The specified accuracy is 0.5 degrees, this will be sufficient for the purposes of slope detection.

The sensor has two different modes of operation. The first outputs a continuous stream of data packets over the serial line. This data packet, in ASCII format, is shown in Figure 3-36.

The other operation mode is a polling approach. By sending the ASCII character "*P" the sensor enters the polling mode. Sending "*P" again causes the sensor to output a single data packet, same as Figure 3-36. This mode offers the advantage of saving microprocessor resources in that the processor does not have to continuously process the serial input. This is especially true when the inclination data does not need to be updated often. Because of this the sensor will be used in this polling mode.

Because of its robust design and minimal use of processor resource, the Zhichuan Electronics ZCT245AL-TTL tilt sensor was chosen. Although this is large and bulky there will be ample room for it on the mobile platform, especially considering it can be mounted vertically or horizontally.

Byte1：X
Byte2：+/-
Byte3：X-axis tens digit of angle value
Byte4：X-axis units digit of angle value
Byte5：point "."
Byte6：one digit after the decimal point of X-axis angle value.
Byte7：20
Byte8：20
Byte9：Y
Byte10：+/-
Byte11：Y-axis tens digit of angle value
Byte12：Y-axis units digit of angle
Byte13：point "."
Byte14：one digit after the decimal point of Y-axis angle value.
Byte15：0d
Byte16：0a

Figure 3-36: Arrangement of inclination sensor data packet [43]

3.9 Chapter Summary

Because of the proven reliability it was decided to use the Mecanum wheels from the existing platform. The right angle DC gear-motor has been chosen because it meets the power requirements of the platform and includes the relay brake for parking. Two 12V sealed lead acid batteries were chosen because they are the most cost effective solution where size and weight are not important factors. The "Silicon Labs" 8051 microcontroller is widely used at the university and was chosen because of the vast expertise and many resources avaliable. Magnetic quadrature encoders have the advantage of being very robust and less susceptible to dust than optical encoders. For these reasons and the ease of installation on the right angle motors these encoders have been used. The display used was chosen because of its wide use in the university while the joystick uses Hall Effect devices to give a high cycle rate, while the small size and fingertip operation helps reduce undesirable movement of the platform. Due to the lack of appropriate commercially available drivers, drivers were designed and built in house. These included current sensing and an alternative method of speed feedback. Finally, an inclination sensor was chosen for its robust design and minimal processing requirements through the use of a serial communication protocol.

4 Design of Mechanical Structure

This chapter will discuss the mechanical design of the machine. This includes discussion of all design constraints and why and how these were met. The process of machining all the parts is also included as well as the assembly of those parts. There are two main sub-sections; one for the design and assembly of the platform and another for the attached gripping system.

4.1 Platform Envelope

In order to ensure the device can manoeuvre in most workplaces it was important that it met a required size envelope. Most door widths are around 800mm in residential environments, however in most work places, including hospitals, they are usually 900mm. A maximum width of 700mm has been set for the platform as this will ensure the platform can easily fit through most doorways. The other envelope constraints are the length and height. The length is governed by the gap between the wall and bed foot board where the platform has to fit into attach to the bed. The height is also restricted by the space available underneath the hospital bed.

4.2 Suspension System

In order for Mecanum wheels to work effectively it is important that all the four wheels are always in contact with the ground. Although a hospital generally has very flat, smooth flooring, there are cases were the surface may be uneven. The doorways are one such example of where there can be an uneven surface; another is the transition from a level surface to a ramp. To ensure that all four wheels stay in contact with the ground, some form of basic suspension system is necessary. Because of the limited unevenness of the terrain the platform will be traversing, only one wheel will need a suspension system to ensure continuous contract with the ground.

4.2.1 Dependent Suspension

A very simple and effective suspension system shown in Figure 4-1. It is called dependent because one side depends on the other: if one wheel was to lower, the other opposite wheel would rise. This is a proven robust system that is used extensively in the agriculture industry as an effective cost affective suspension system. Another advantage is that with the relatively flat terrain it will be navigating this system will not need any kind of shock absorbing spring as the two sides are counter balanced.

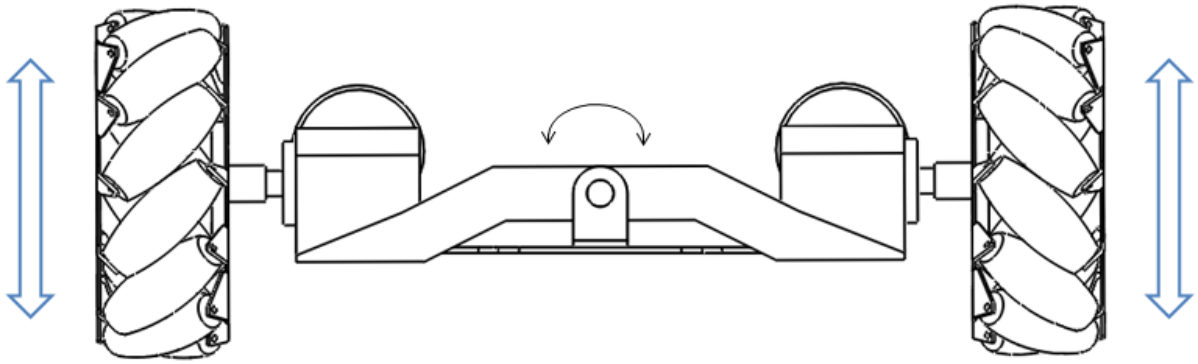


Figure 4-1: Dependent suspension system

The disadvantage with this system is that it will create situations where the vertical wheel plane is non-orthogonal to the floor. This would not matter with traditional wheel design but with the chosen Mecanum wheels this will be problematic because of the "flatness" of the wheel. However with some simple calculations the effect of this can be assessed and controlled.

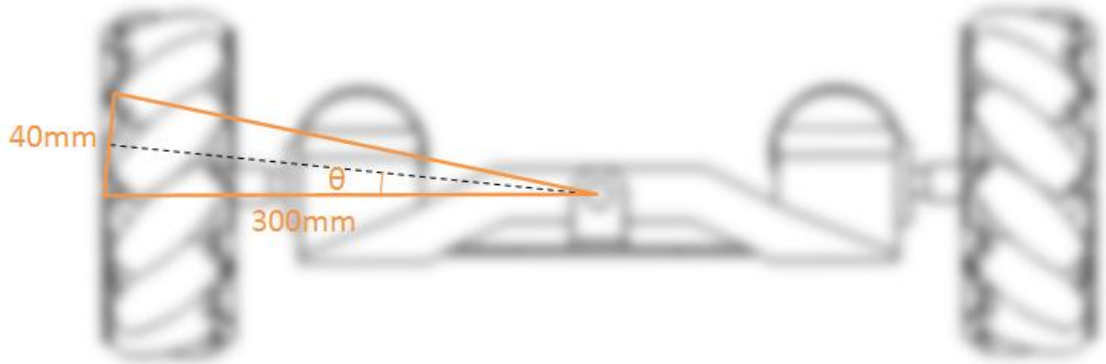


Figure 4-2: Wheel angle calculation diagram

Assuming that 40mm would be the maximum disturbance the platform will encounter (Figure 4-2) and remembering that the other wheel will go down doubling the maximum disturbance, the following calculations can be completed.

$$\cos\theta = \frac{20}{300} \tag{4.1}$$

$$\theta = \cos^{-1}\frac{20}{300} = 86.177° \tag{4.2}$$

$$90° - 86.177° = 3.823° \quad (4.3)$$

This means that the wheel will only deviate by approximately 4˚ from perpendicular under these conditions. This will have little or no effect on the operation of the wheels making this suspension system a viable option.

4.2.2 Independent Suspension

There are several different types of independent suspensions systems. The advantage with independent suspension is that each wheel is isolated from the other, i.e. one wheel can rise without affecting any other wheel.

The double wishbone system shown in Figure 4-3 is an example of independent suspension. The advantage of this system is that the wheel will stay perpendicular with the floor no matter what position the swing arm is in. The disadvantage with this system is that it is more complicated to design and manufacture. The wheels also have a slight sideways movement with the rising and lowering of the swing arm, although this is insignificant and will not affect the operation of the platform.

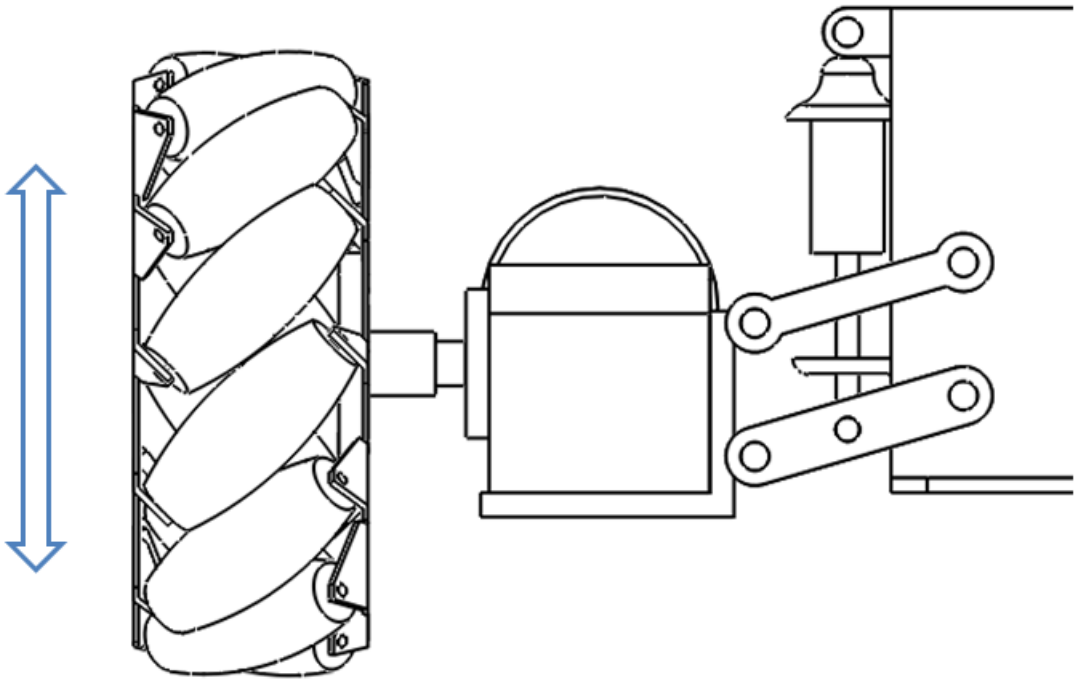


Figure 4-3: Double wishbone suspension

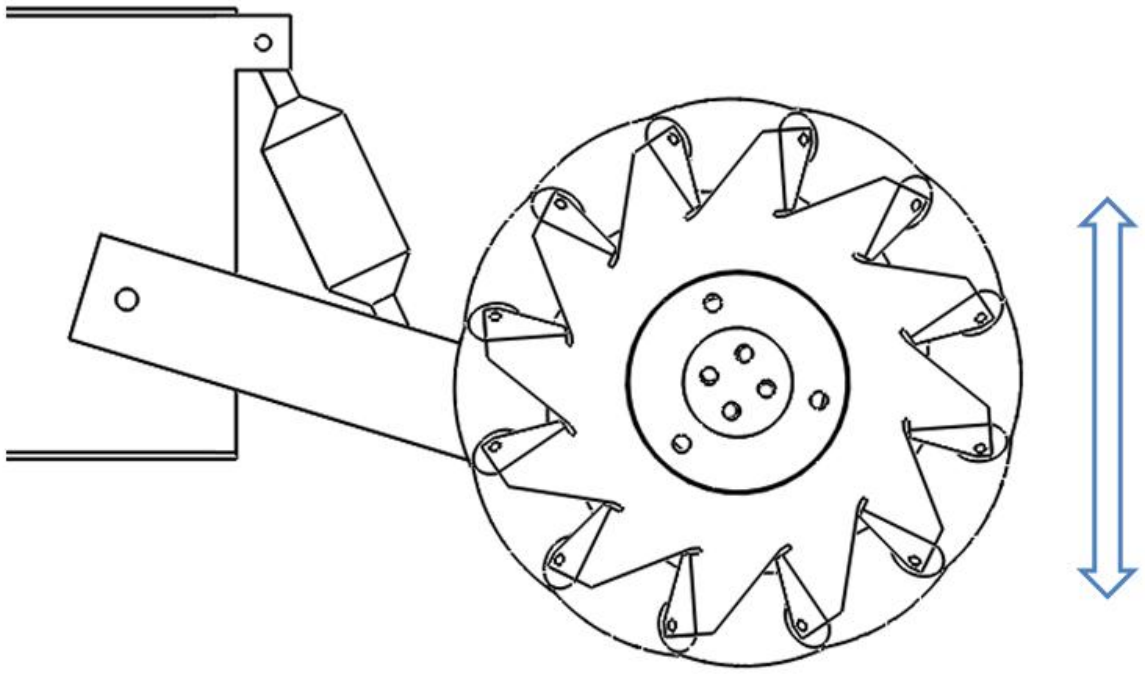


Figure 4-4: Leading arm suspension

Figure 4-4 shows a leading/trailing arm suspension. This type of suspension also stays perpendicular to the ground. It also does not suffer from the side to side movement of the wishbone system. Instead the wheel will move back and forwards as the arm swings. The other advantage of this system is its ability to use a rubber stop instead of the expensive shock absorber. Because there will be very little movement in the suspension, the need for damping is reduced. This means that a rubber stop can be used as long as it is positioned as close to the pivot point as possible to ensure the longest sprung stroke.

4.2.3 Chosen Design

Each of these suspension systems offers its own advantages and disadvantages. However, because the surface the platform will be traversing is relatively flat, the disadvantage of the wheels being non perpendicular with the ground is inconsequential. This, combined with the simplicity, reliability and lower cost of the dependent pivot suspension system makes it the best choice for this platform.

4.3 Platform Design

The design constraints had to be kept in mind throughout the design process. As the batteries take up a large amount of room in the platform it was decided to build the platform around them.

4.3.1 Battery Enclosure

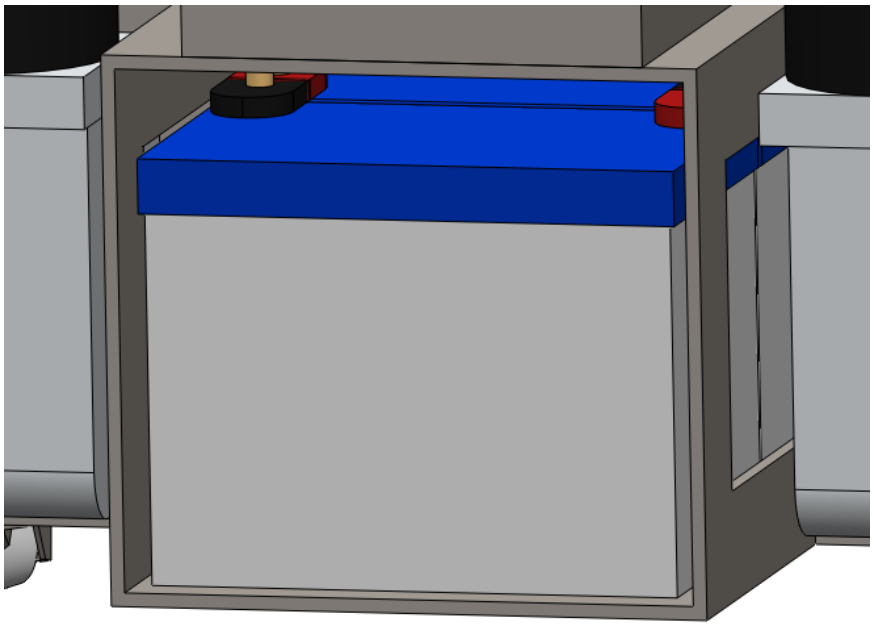


Figure 4-5: New battery enclosure

One of the main weaknesses of the original prototype was the fact that the batteries were very difficult to access. This made servicing and a quick battery swap difficult. This has been addressed in the new design building around a central battery enclosure, shown in Figure 4-5. The new design ensures that the batteries can be easily slid out the back of the platform. A rubber strap has been attached to make sure the batteries cannot slide out or move around while the platform is in motion.

4.3.2 Motor Orientation

The motor orientation was very important if the overall design was going to fit the desired design envelope. The first design for the motor orientation can be seen in Figure 4-6. This design seemed acceptable at first until the "Allied Motion" RAD test unit arrived and it was found to be longer than the technical drawing suggested. This produced the problem of the front motors causing the platform length to exceed the envelope requirements.

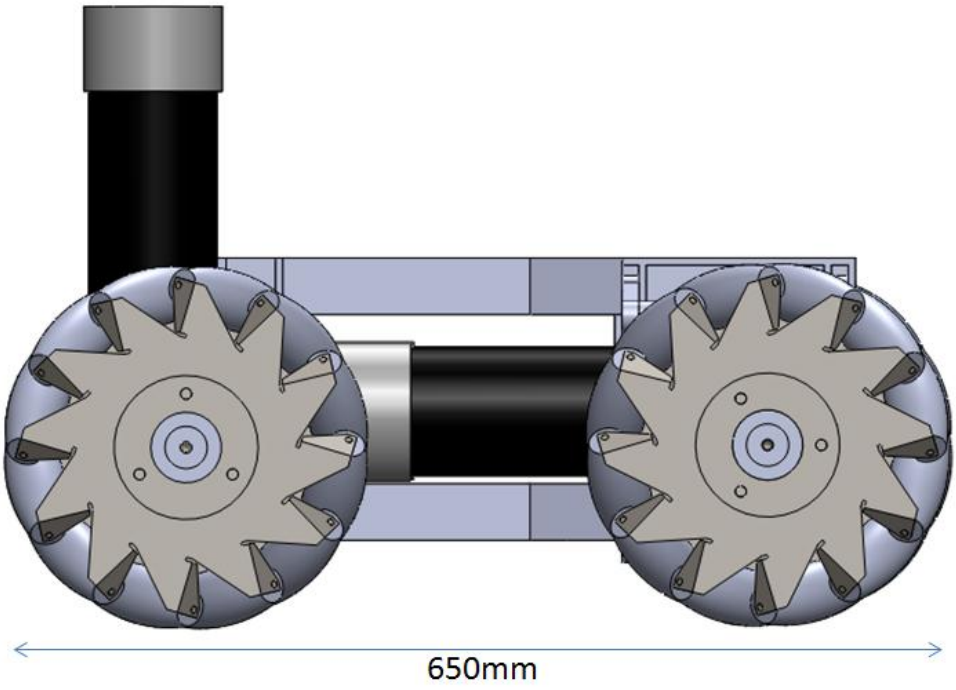


Figure 4-6: RAD right angle motor orientation

An alternative design was tried (Figure 4-7) by angling the motors to reduce overall length, but this proved to exceed the height requirements of the platform.

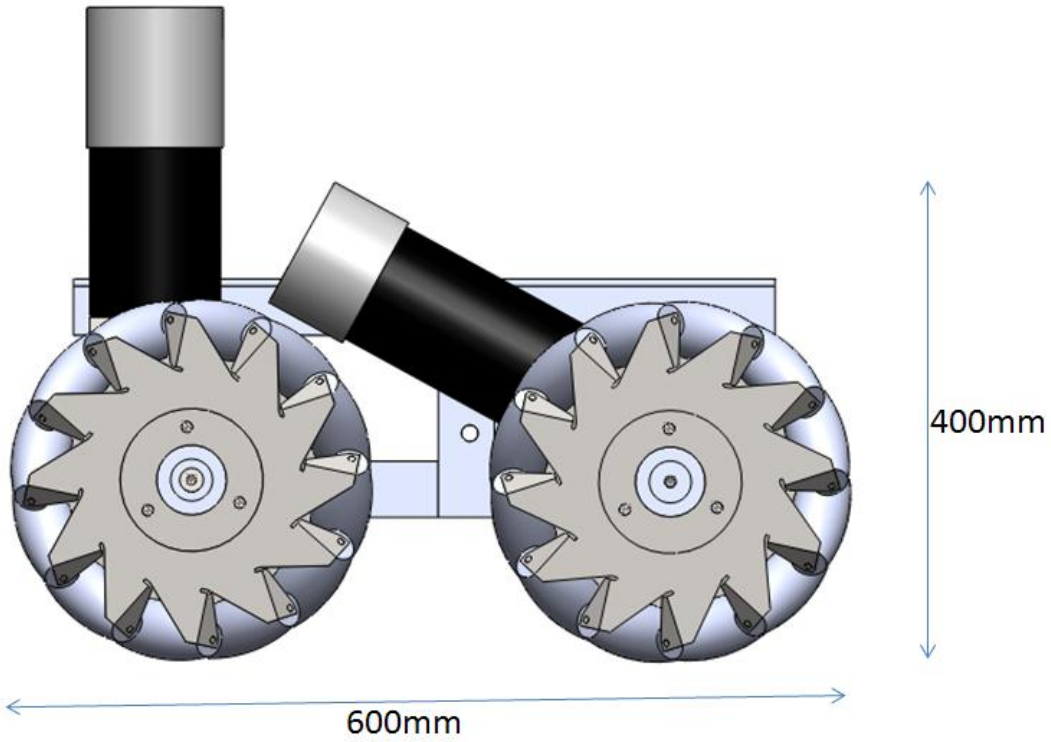


Figure 4-7: RAD right angle motor orientation alternative

One way around this problem is to use different motors in the front. Such as parallel gear-motors (NPC-T64) used in the original platform. There are other options available for this motor including the "Allied Motion" PLC parallel shaft gear-motor shown in Figure 4-8. These motors have similar power ratings to the right angle rear motors so would make a good match. The problem is that they are too long to meet the width requirements of the platform. Although a custom motor could be made, time constrains meant that the original NPC motors will be used as they are proven to be adequate and can be replaced by a higher quality motor if needed.

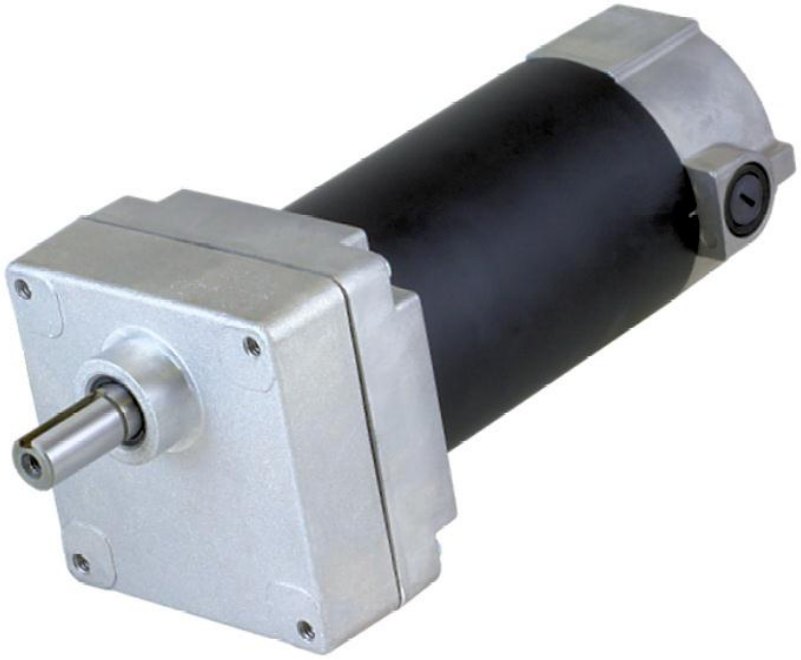


Figure 4-8: Allied Motion PLC Series Parallel-Shaft Gear-motors [44]

Encoders need to be fitted to the NPC motors. For this to happen some modification was necessary. A new end plate needed to be machined to provide a proper mounting platform for the encoders to be fitted to. A new shaft was made to thread into the existing motor shaft and provide a sleeve for the magnetic wheel to sit on. Figure 4-9 shows the mounted encoder. Because the NPC motors have a different gear ratio, the speed readings will be different for the front and rear motors. Section 6.1.1 will discuss how this is handled.

Figure 4-9: Encoder mounted to NPC motor

4.3.3 Other Design Features

There were many other parts that needed to be designed. This included motor mounts and wheel hubs. The rear motors were easily mounted using a wing design attached to the main frame of the platform. This design is shown in Figure 4-10.

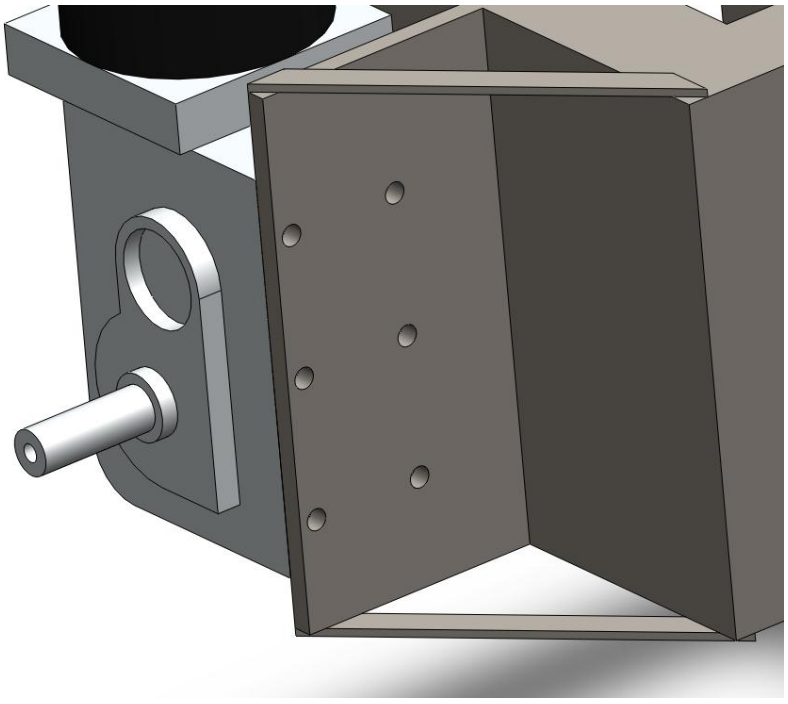


Figure 4-10: Rear motor mount

The front motor mounts where quite different in that they had three different mounting points. A locating mount was designed to fit around the motor and utilise the three mounting points. This was then attached to the pivot tunnel with four bolts in two different planes. This is shown in Figure 4-11.

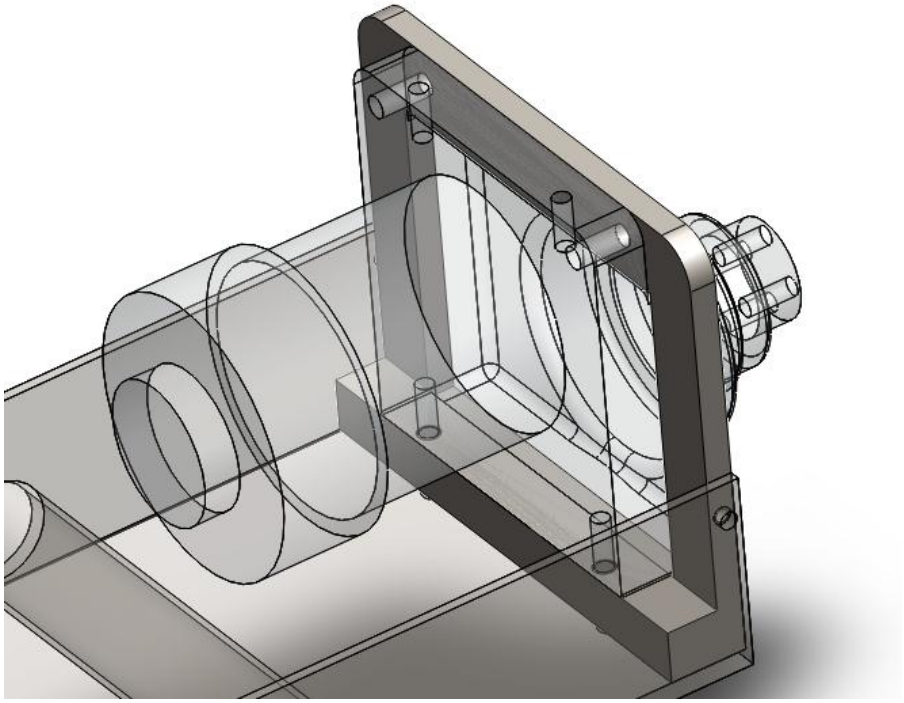


Figure 4-11: Front motor mount

There was a need for different motor hubs for each motor as they had different mounting arrangements. The rear has a single bolt and key system while the front utilised four bolts. These are shown in Figure 4-12.

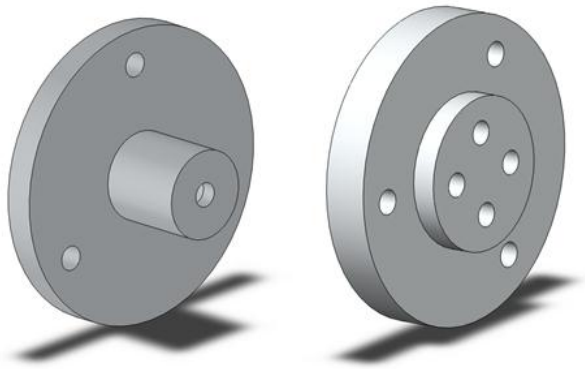


Figure 4-12: Motor Hubs Rear on left Front on right

4.3.4 Final Platform Design

The final design of the suspension allows the front motors and wheels to pivot in relation to the rest of the platform. This is achieved through the uses of a precision machined solid silver-steel cylinder that is bolted at both ends of a front protruding folk. An oil impregnated Nylon bush is then used to enable smooth rotation of the cylindrical tube and the tunnel steel the motors are attached to. This system is very simple and therefore would be suitable for a mass produced product. It is also a very robust system that would require very little maintenance because the suspension will not be moving often. This configuration is shown in Figure 4-13 with the motors removed.

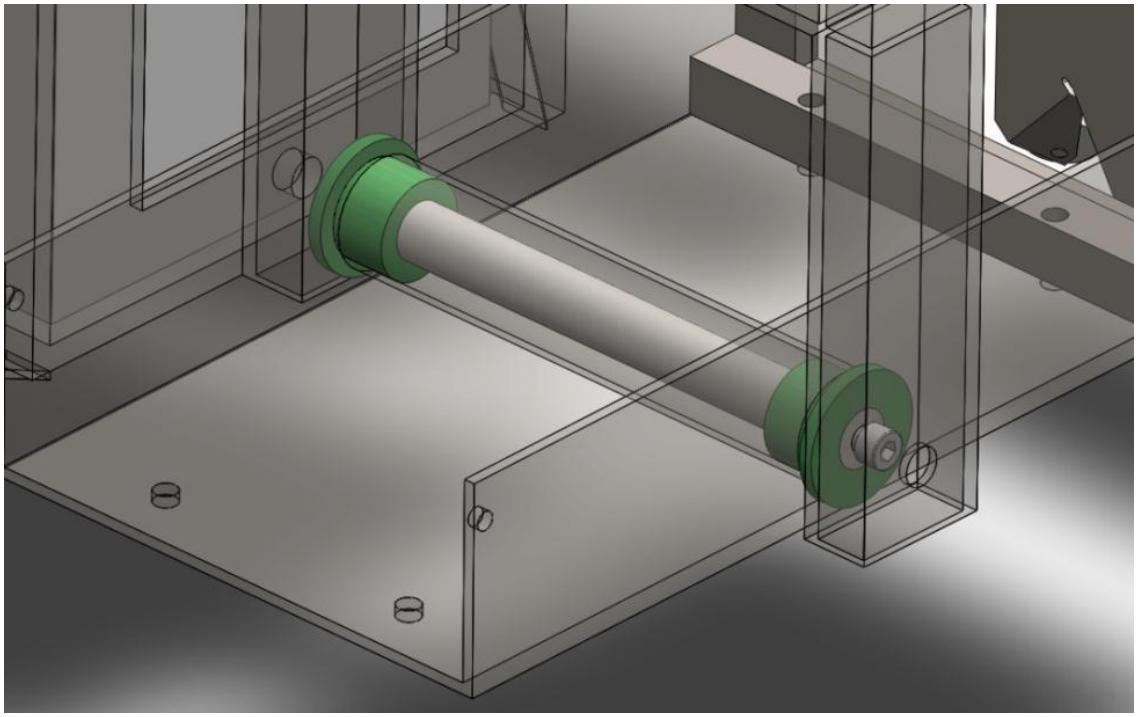


Figure 4-13: Transparent view of the suspension system

The pivoting motion that is enabled by this system is shown in Figure 4-14. This demonstrates an extreme pivoting motion that would not be experienced by the finished platform. Stoppers could be incorporated to limit the movement and protect the motor from any potential damage in extreme circumstances.

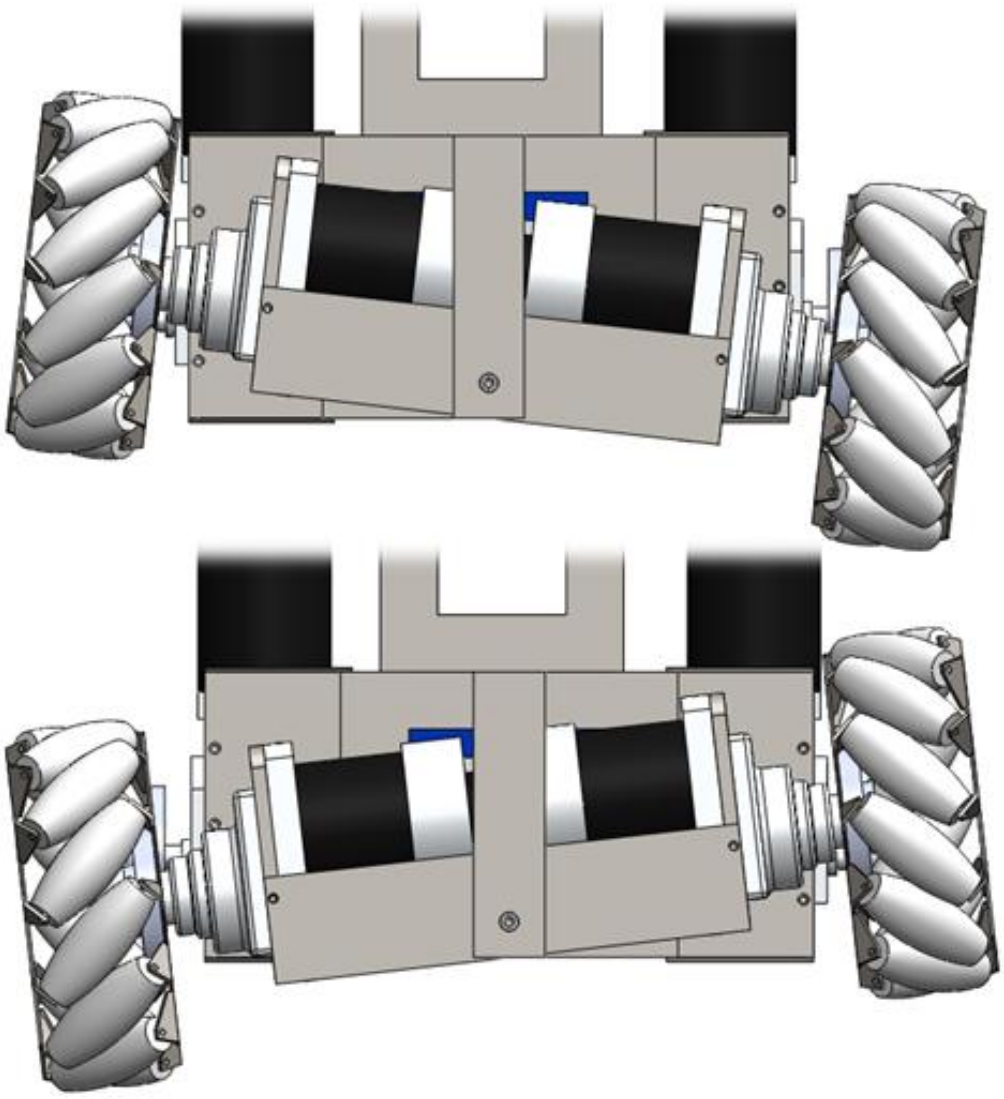


Figure 4-14: Pivot suspension system

The final design of the platform is shown in Figure 4-15. This design meets all the specified requirements for the size envelope. It also incorporates a suspension system while housing motors and batteries that meet the power requirements of the system.

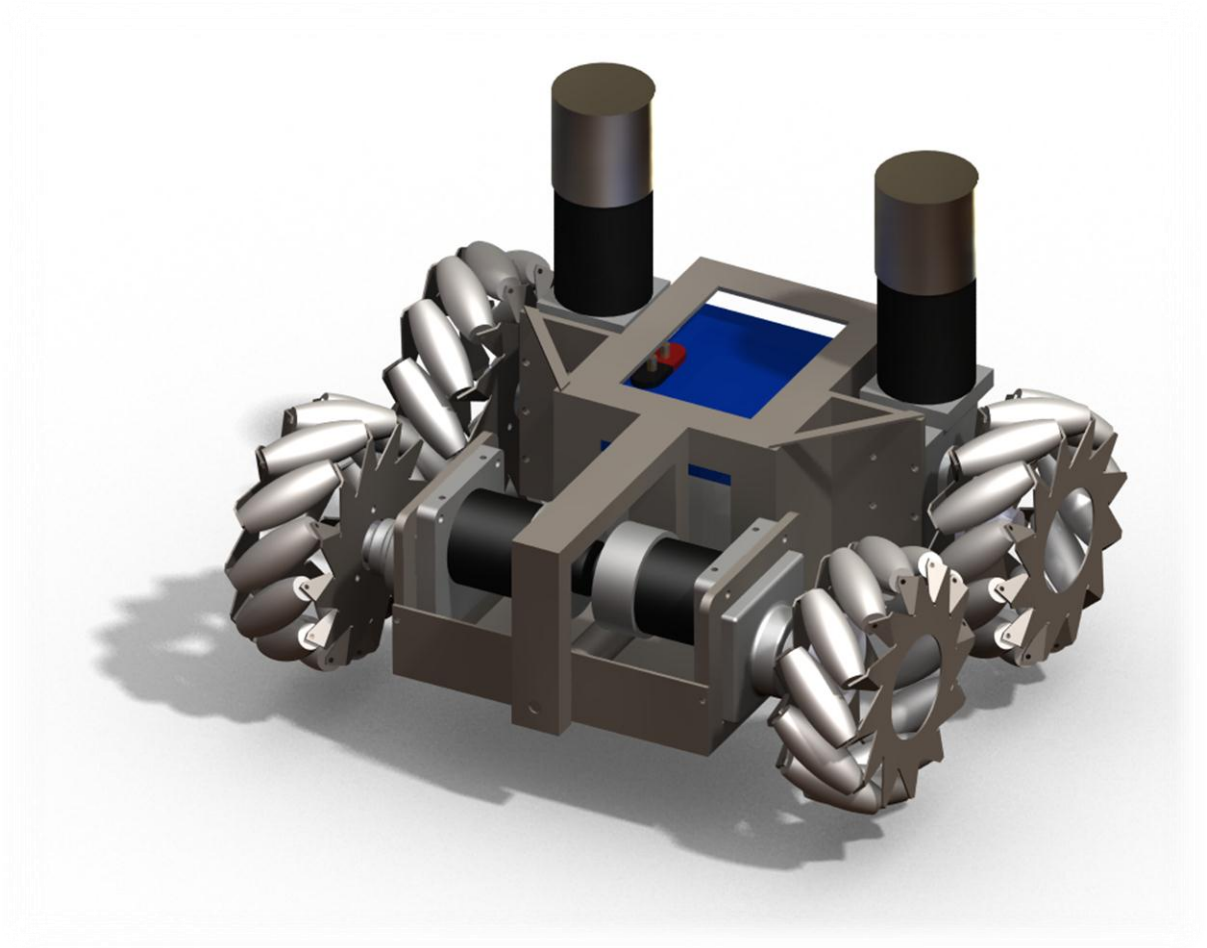


Figure 4-15: Final design of platform render in SolidWorks

4.3.5 Machining and Assembly Process

The main frame of the platform was made out of a combination of 45mm wide, 5mm thick angle steel, 50mm x 25mm x 3mm rectangular extrude and 200mm x 75mm x 4mm tunnel extrude. These materials were chosen because they are common sizes that can be easily obtained. These were cut and welded as per the desired design, all mounting holes were drilled and the suspension system was implemented. Although this method is fine for a limited production run, if mass production becomes feasible the design may need to be modified slightly to allow the components to be pressed or machine cut (i.e. more modular). This main structure is shown in Figure 4-16 and Figure 4-17. Another feature to note is the rubber attached to the ceiling of the battery enclosure. Because there is exposed steel above the battery terminals it was important to include this rubber to ensure the batteries could not short across the frame. Figure 4-18 shows a close up of the nylon bush used to ensure smooth and solid movement of the pivoting suspension system.

Figure 4-16: Main structure of platform

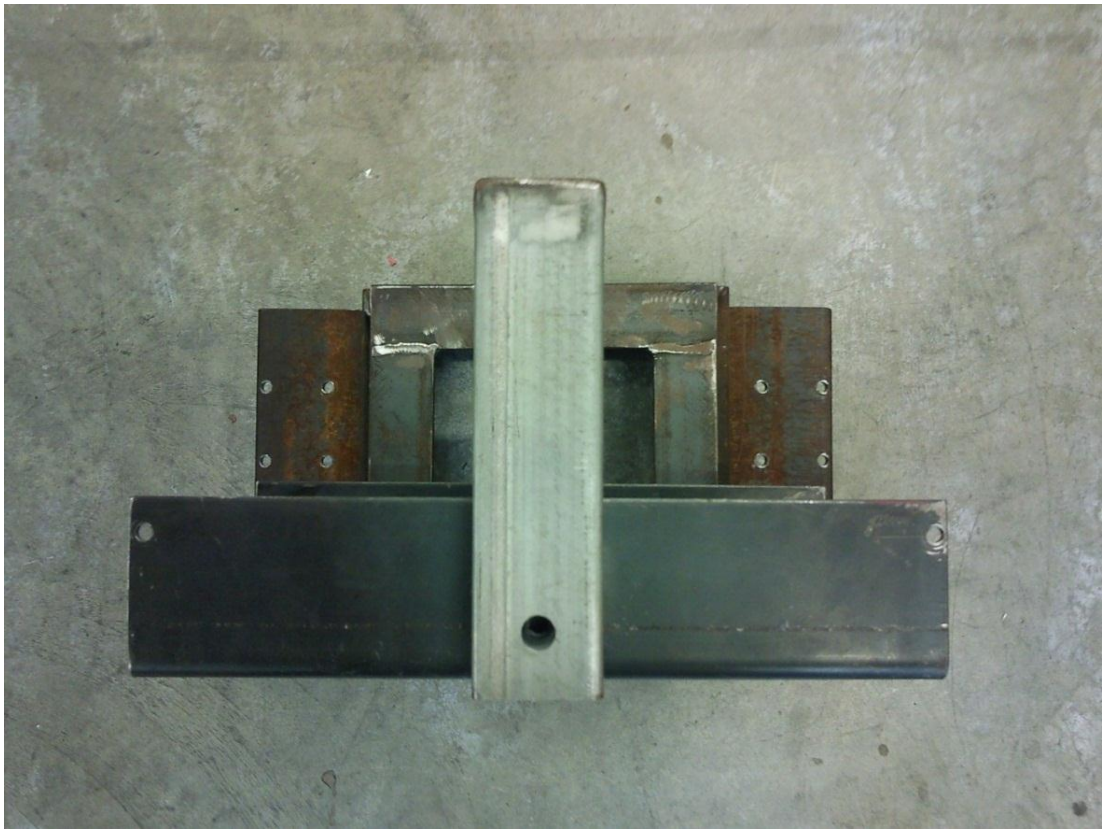


Figure 4-17: Close up of suspension system

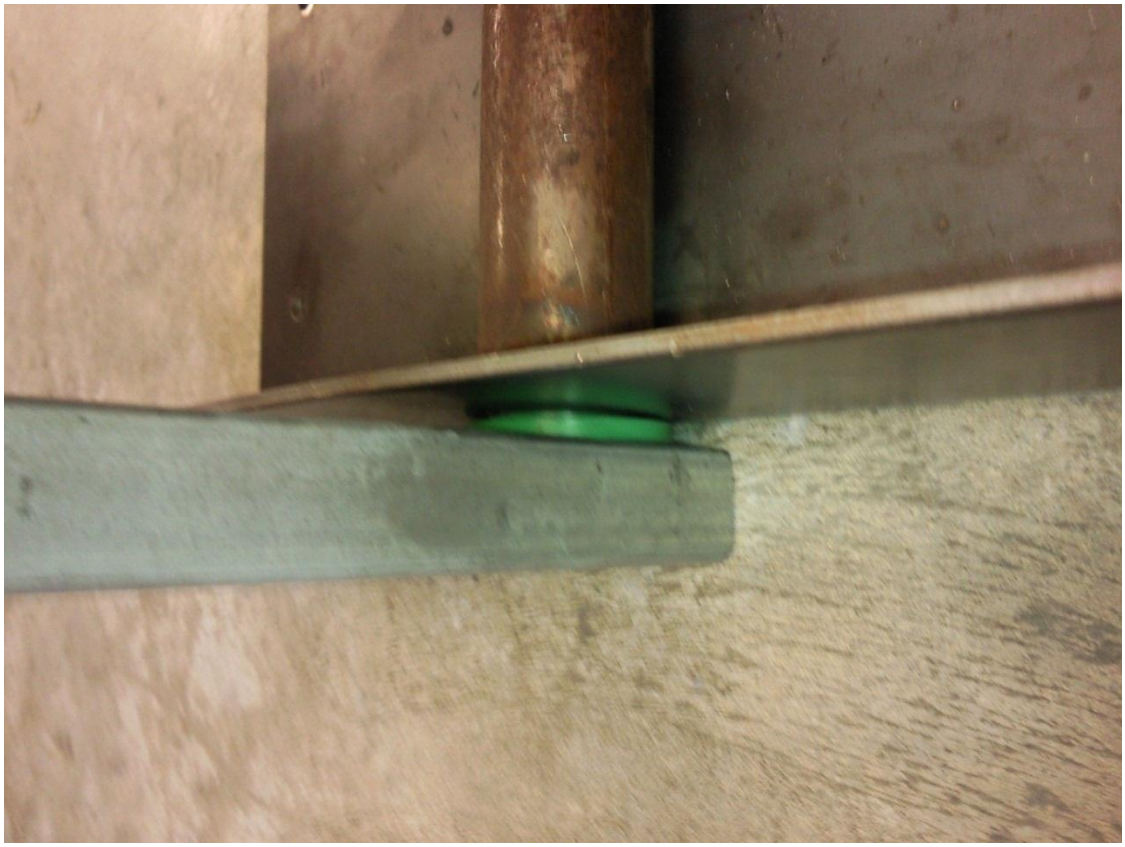


Figure 4-18: Close up of oil impregnated nylon suspension bush

4.4 Bed Attachment System

The main requirement for the bed attachment system is for it to securely attach a hospital bed to the platform so that it can manoeuver the bed in an omni-directional fashion. This system must have a high level of reliability if it is to move several beds per maintenance cycle. Another requirement of the attachment system is that it should to be easy to use. Because this device will be used by nurses that are not trained in machine operation, it is very important that the device involves as few steps as possible to hitch the bed. It is also important that the hitching method does not put any unnecessary stresses on the machine. For instance when transitioning from flat ground to an incline there is a stage when there is a twisting motion on the platform. Being able to fit in an elevator with a bed attached is another important requirement that must not be interfered with in the attachment system design. The last requirement that the attachment system must meet is to provide an appropriate user interface for the operator to use.

4.4.1 Attachment System Design Envelope

To safely fit inside an elevator with a bed attached it was decided that a distance of 220mm was the maximum allowable to be exposed from the end of the bed. The height of the user

interface was set at 1000mm. This is to provide a comfortable height for the operator to control the platform and view the LCD screen. These constraints combined with the platform envelope are detailed in Figure 4-19.

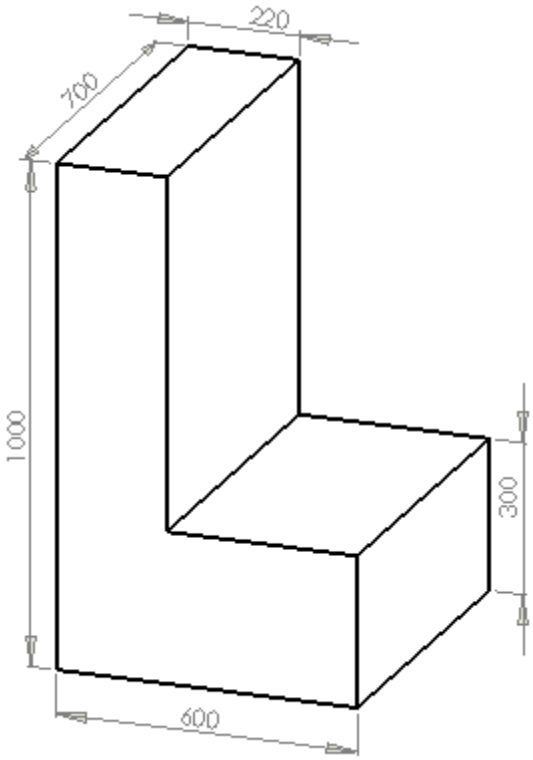


Figure 4-19: Attachment system design envelope

There are two different approaches to hitching the bed. One is to grip onto the bed and push it around with the drive wheels; the other is to lift or partially lift the bed and use this weight to push the bed. These approaches would put different stresses on the envelope size with a lifting system being situated in the area under the bed and the gripper system being at the bed's head/tail.

4.4.2 Gripping Mechanism

The advantages of using a gripper system is that it provides great flexibility in gripping many different types of beds, as head and foot boards are a common feature of hospital beds. A gripper system is also simple to design and build, providing reliable operation with little force involved. This kind of system would also involve minimal input from the user, with lining the platform up and initiating the gripper being the only necessary operations. This kind of system can also allow some movement, especially of the type encountered when traversing a ramp. Disadvantages include the fact that the platform will need to be very heavy to be able

to push the 300kg beds around. This produces a problem with safety as large heavy machines tend to do more damage when colliding with other objects or people. Having a gripper system may also increase the overall width of the machine breaking the design envelope.

There are several different design concepts for this type of attachment system. The first shown in Figure 4-20 is the original design that was implemented on the existing platform. This design holds the headboard tight against the turret by using the three inbuilt actuators. There are many problems with this system including reliability issues through the use of three separate motors. This system is also difficult to control because of the two stages of operation. It also does not allow any movement at all for the grasped bed.

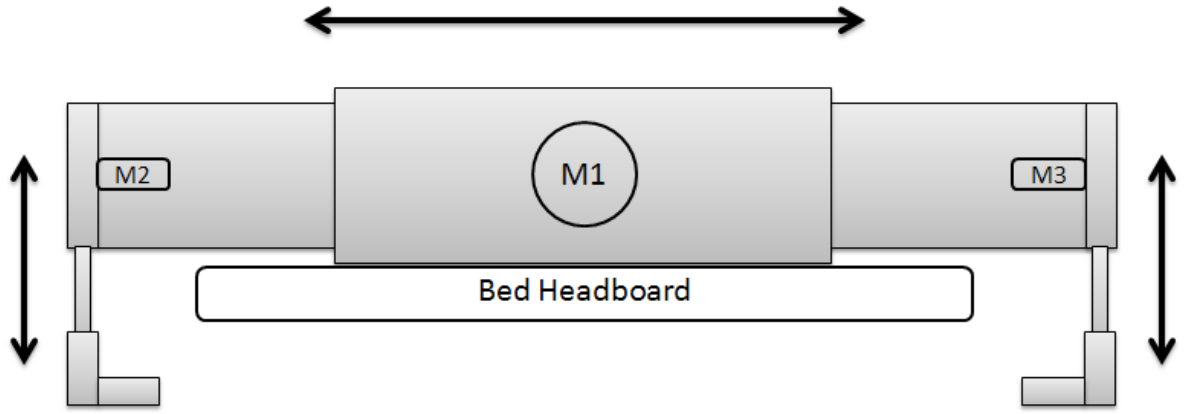


Figure 4-20: Original gripping system

To simplify the design and control it was decided that the same effect could be achieved with just one actuator motor. The cam approach shown in Figure 4-21 demonstrates how this can be achieved. As the bed makes contact with the sprung lobe, the finger swings round and secures the headboard against the turret. Although the control and maintenance for this system is less complicated, there is still the issue of providing movement for the bed in gradient changing circumstances. This could be overcome by including a pivot in the gripper arms to allow the headboard to swivel in the desired direction. However this adds unnecessary complexity to the system and increases the likelihood of device failure.

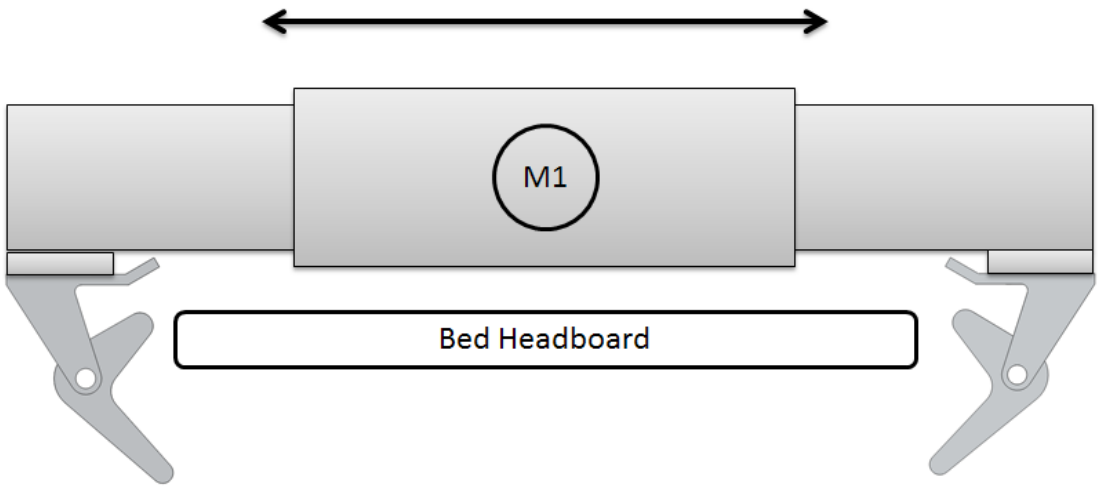


Figure 4-21: Cam gripping system

The alternative to this is to use a locating finger. Figure 4-22 shows how this kind of a system would work. This design is very simple with a minimum of moving parts. It also has an advantage over other designs in the way of not holding the head board against the turret. This allows pivoting movement that would lower stress on the platform when transitioning to a gradient.

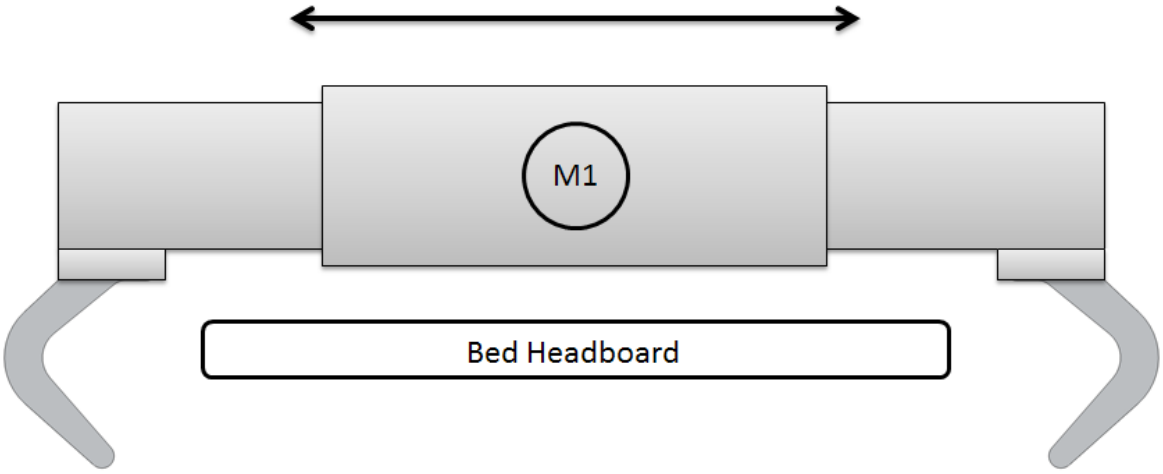


Figure 4-22: Locating gripper system

4.4.3 Lifting Mechanism

Included in the requirements of this project is the statement that the new prototype is to use a gripping mechanism similar to the original prototype. However, a lifting mechanism is worth investigating. A lifting system has the advantage of using the beds weight for the pushing power. This means that the platform itself can be made lighter and therefore safer. This type

of a hitching system also does not require a turret for housing a gripper system, thus alleviating any stress on the design envelope. However a lifting system would not be as universal as a gripping system with a secure hitching mechanism that will attach to many different bed types being very difficult to achieve. Because of the nature of the lifting mechanism there is a potential for the hitching point to swivel causing problems when trying to manoeuver a bed, especially in sideways movement. Deterioration of the mechanism is also potentially increased by the continual lifting and lowering of a heavy load. The final disadvantage of this kind of a system is the added complexity from an operator point of view. As the mechanism for attachment and lifting will be under the bed it will be more difficult to line up and more complicated to attach the system. Examples of lifting systems can be found in section 2.1, most of these rely on a forklift system that is not viable for this design as more space is needed for the four motors to enable the omni-directional movement.

4.4.4 Chosen Design

A gripping system was preferred over a lifting system because it offers far greater flexibility as an attachment system. Because the vast majority of beds have head and footboards a gripping system provides the largest market for the product. Other advantages include the simplicity of the system and the lack of continual large stress being exerted on the system. This will improve the robustness and keep maintenance cost to a minimum. This design also provides a secure mount for a human interface. The final advantage this system has is that it has proven to work successfully in the last prototype. The need for a large weight is counter acted somewhat by the fact that the four motors and two batteries themselves weigh upward of 50kg, meaning that the platform is going to be heavy no matter the system used.

Of the three gripping systems presented in section 4.4.2 the locating finger system is the most suitable design (Figure 4-22). It out performs the other two designs with a simpler design. The fingers allow for some operator error while also holding the bed away from the frame, permitting movement and alleviating stress.

4.4.5 Machining and Assembly Process

The gripper was attached to the platform through the use of a turret. This was chosen as it also provided a good place to attach a user interface. This was constructed from the same angle steel used in the platform. A double rack and pinion actuator, from “E Motion LLC”, was used to enable the gripping motion and several bump switches were incorporated to aid in the control of the mechanism. The extending tubes were constructed out of square

aluminium extrudes with nylon runners to provide smooth motion. The turret was constructed out of 5mm angle steel with added strength across the gripping arms being provided by steel straps and screws. Figure 4-23 shows the final design of the gripper system using the locating hook design fingers.

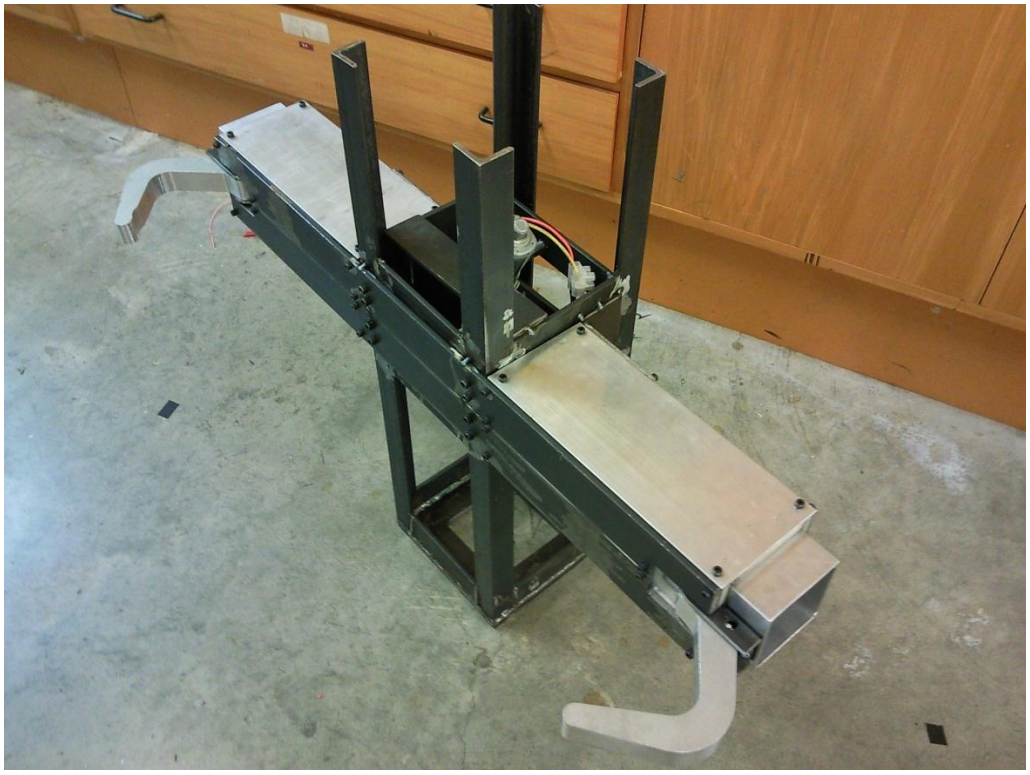


Figure 4-23: Final gripper design

Four bump switches were incorporated into the design of the gripper. Two of these, one shown in Figure 4-24, are used to determine if the gripper is completely open or closed. The other two are triggered when the fingers are pushed against. This is achieved by having the fingers on a sprung pivot with hard limits. The small micro switch is shown Figure 4-25. The small screw was incorporated to provide some adjustability so that the limit stops can be used to avoid damaging the switch. The purpose of these switches is to aid the controller in knowing when there is a footboard present. It can also aid in determining when the gripper is not centred correctly on a footboard.

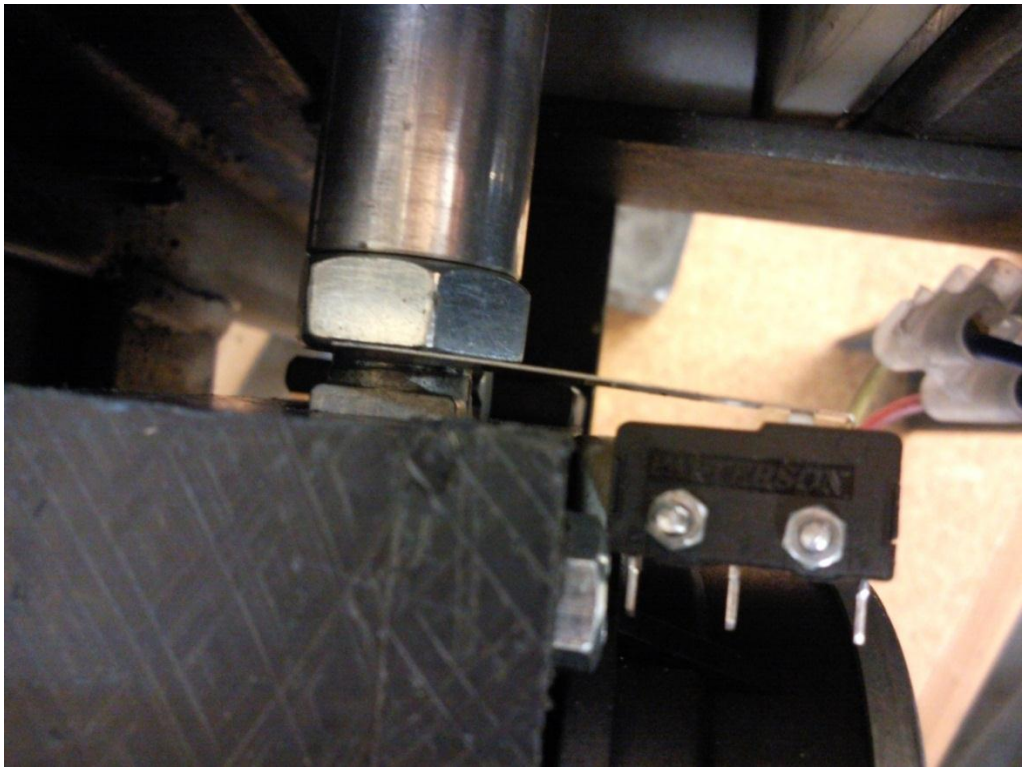


Figure 4-24: Gripper limit bump switch

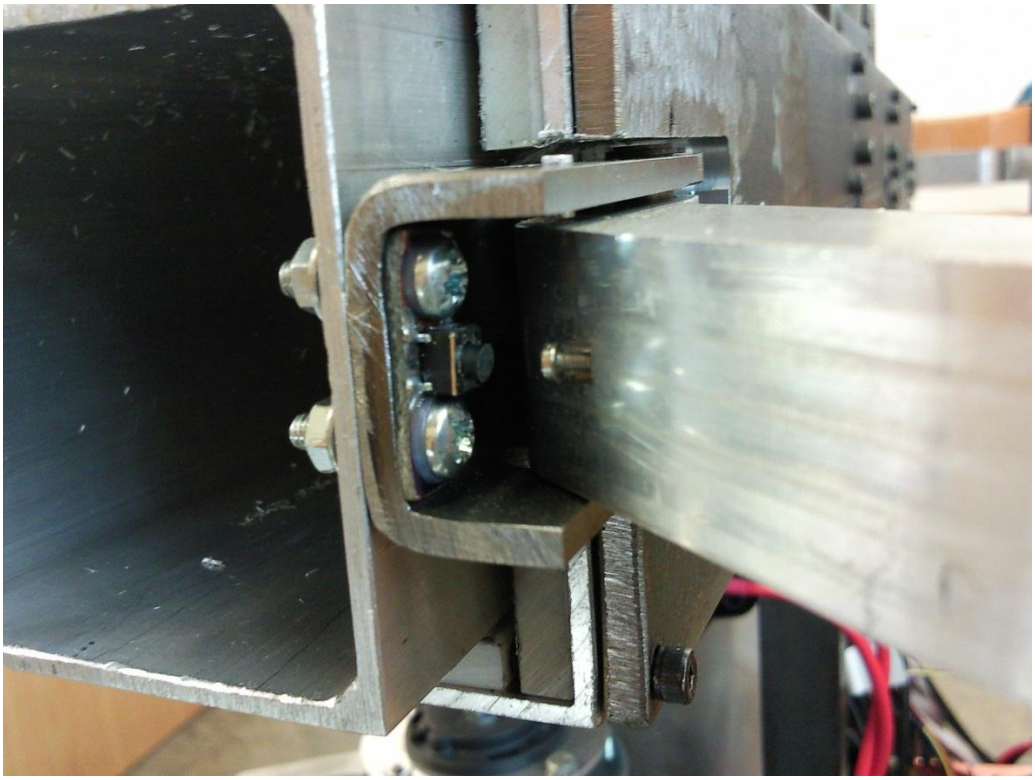


Figure 4-25: Finger bump switch

4.5 Final Mechanical Design

With the two main components of the system completed, they needed to be combined together to realise the full functionality. Also the user interface needed to be designed.

The gripper was attached to the platform through the use of mounting wings welded to the platform; these are shown in Figure 4-26.

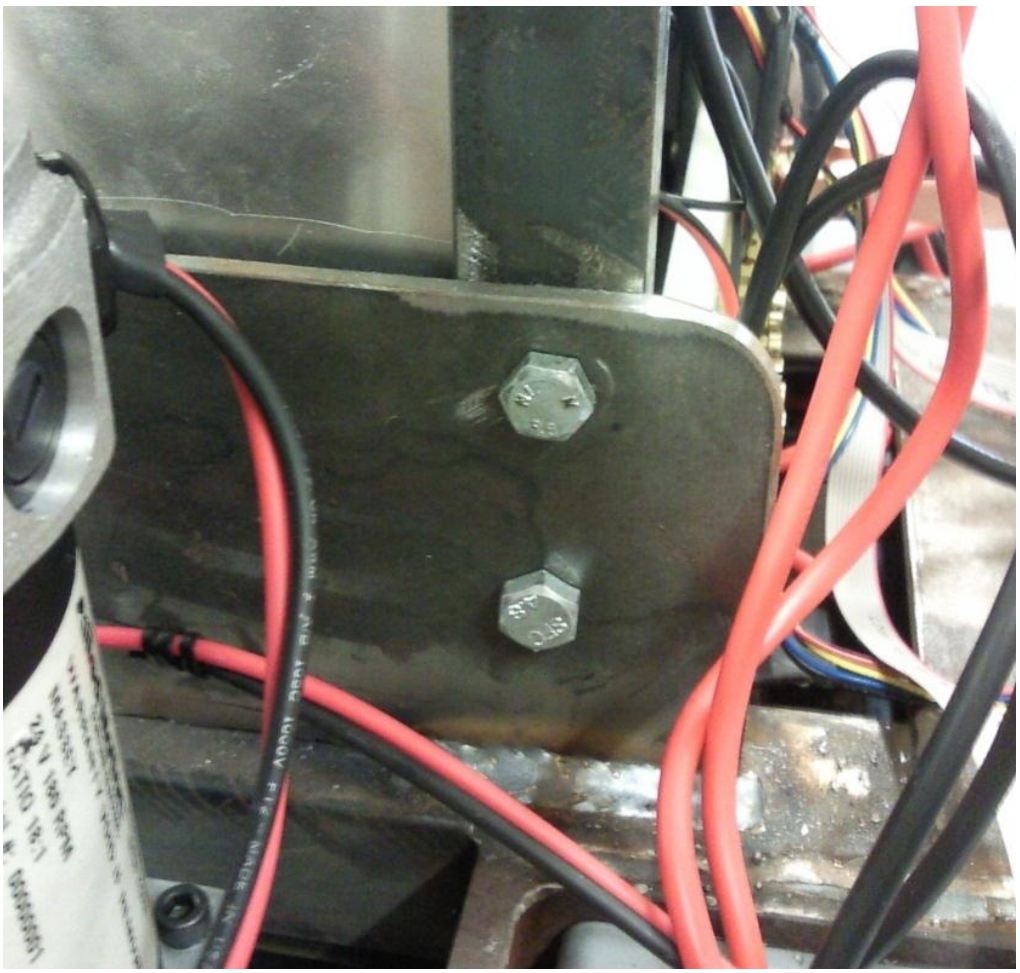


Figure 4-26: Gripper/Platform attachment

In order to provide a little more foot room for the operator whilst maintaining the backend width profile for elevator travel, a simple swinging mechanism was implemented. This was done in a way so that the interface would remain parallel with the ground. Figure 4-27 shows this system. Figure 4-28 illustrates the layout of the user interface with the joystick, LCD screen and the emergency stop button. The emergency stop button is connected to a contactor that cuts off power to all the motors.

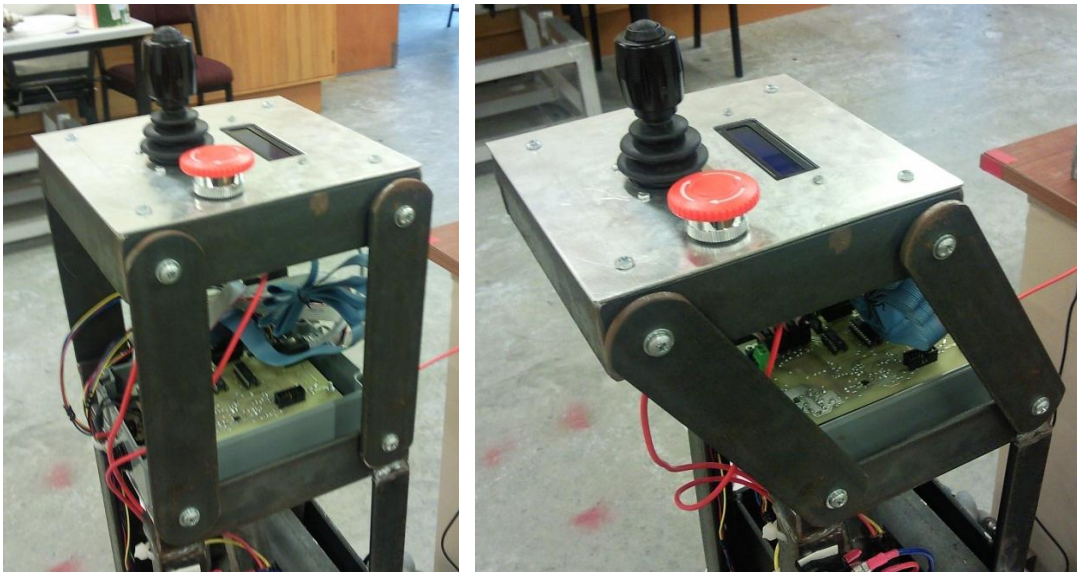


Figure 4-27: Swing mechanism for adjustment of user interface table

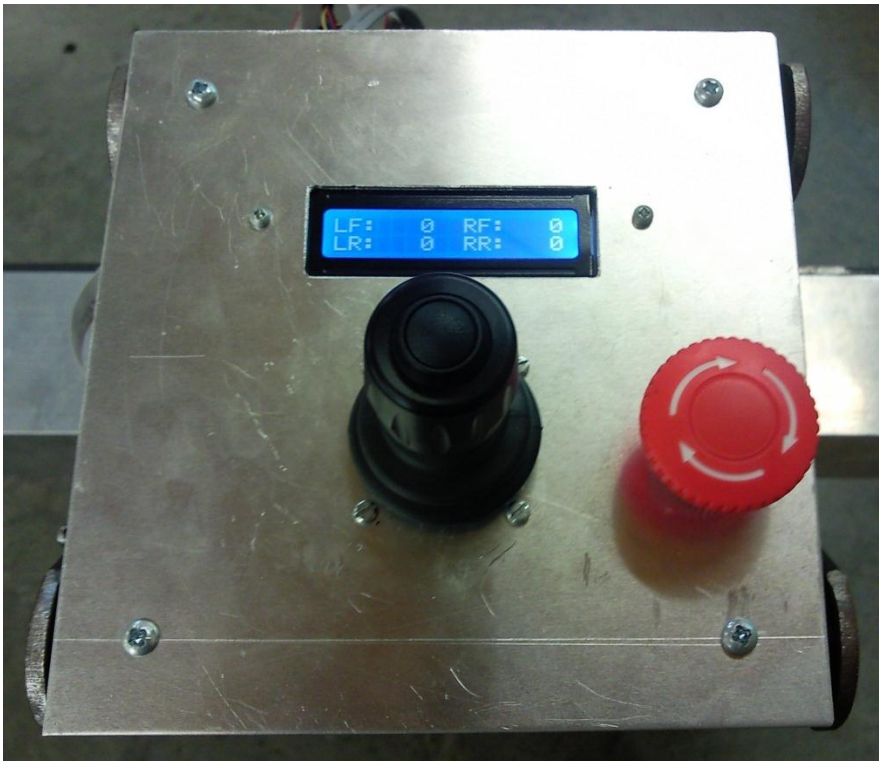


Figure 4-28: User interface layout

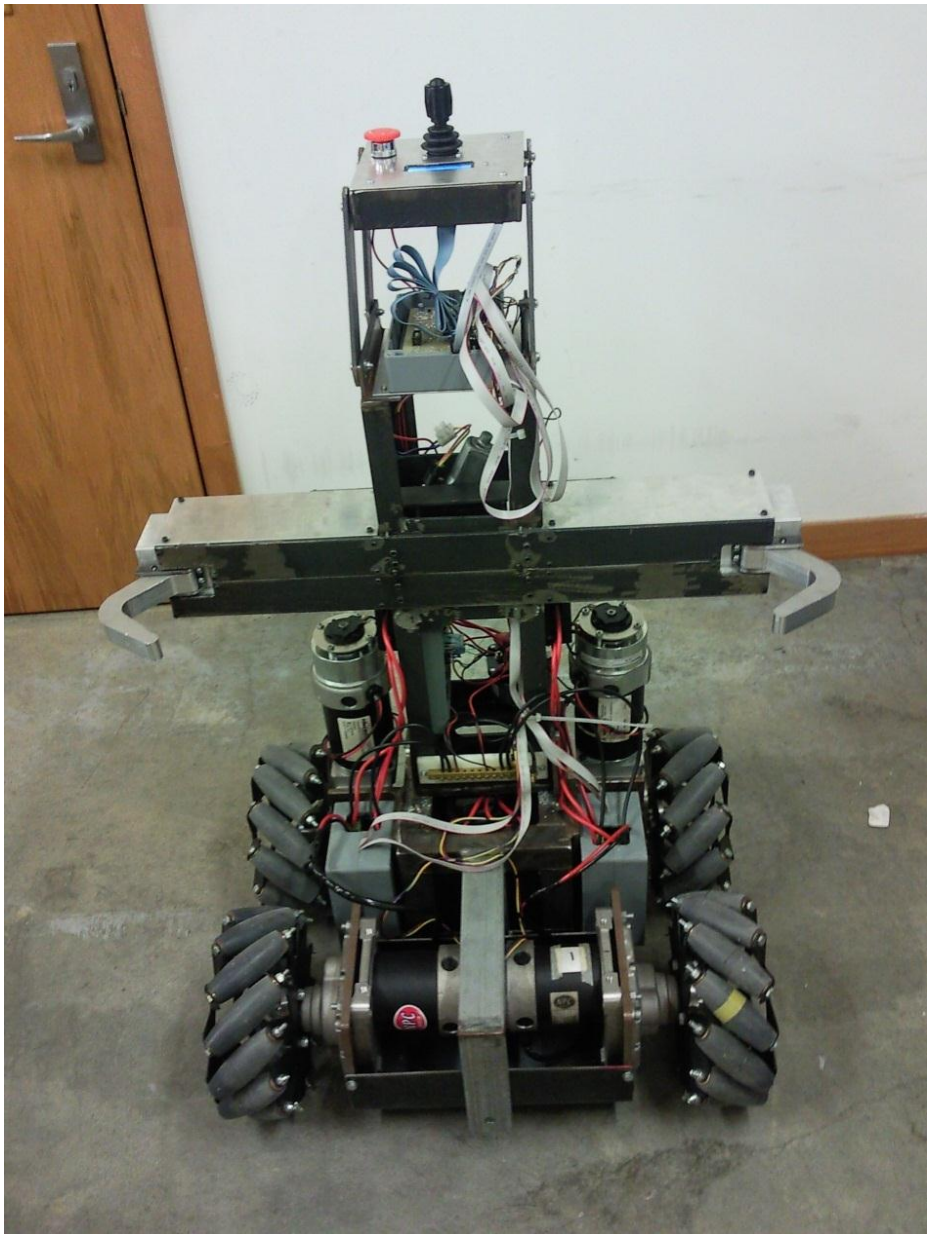


Figure 4-29: Completed bed moving platform

4.6 Wiring and Fuse Selection

The wiring consisted of a main battery fuse rated at 250A to protect the battery from being shorted. The four driver boards were protected by a 80A fuse each as they should be able to handle this current for a short amount of time. The main power to the motor drivers can be shut off by a 200A contactor that is controlled by an emergency stop button located on the user interface. For testing purposes the microcontrollers were run directly from the battery with their own 2A glass fuse. This enabled testing and program updates to be undertaken without the main motors being driven. A temporary isolation switch was added to the gripper motor for similar reasons. The wires and signal cables were loomed using holes drilled into

the frame and cable ties to hole the wire against the frame. The fuse holders were mounted on the side of the turret under the gripper arms. The 250A fuse was housed inside the turret, close to the battery. The contactor is also housed inside the turret.

4.7 Chapter Summary

The platform has been designed with strict size constraints in mind. This has caused two different motors to be used for the front and back to maintain the parking brake feature while staying within the size envelope. A suspension system is necessary to ensure that all four Mecanum wheels remain in contact with the floor at all times. A pivot system has been chosen because of its simplicity, durability and effectiveness. There were many different options for the bed attachment system. A gripping system was chosen as it provided the most flexibility for attaching to different types of beds. The finger grippers possessed the advantage of limiting the number of motors to one while maintaining a strong grip and providing a locating system for the bed footboard. The human interface has been kept as simple as possible containing only the three axis joystick with button, the LCD display and an emergency stop button. These have all been mounted on a swing platform that can swing out to increase foot space for the operator.

5 Control Circuit Boards

To maintain flexibility of the platform it was decided to use two microcontrollers. This chapter will investigate the boards' design around these microcontrollers. The first main controller is dedicated to platform motion control. This interfaces with joystick, encoder, driver board, battery sensor and brake relays to control the operation of the platform. The other (slave) microcontroller will determine the type of operation the platform engages in. Through an initialisation process the slave micro will identify itself depending on the appendage it is controlling. This causes the master microcontroller to enter the mode that corresponds to the attachment it is manoeuvring. From here the master microcontroller will be able to send commands to the slave board to perform any necessary tasks and the slave will be able to send any information back to the master. This ensures full functionality of the platform while enabling the flexibility to perform a multitude of tasks.

5.1 Block Diagram of Required System

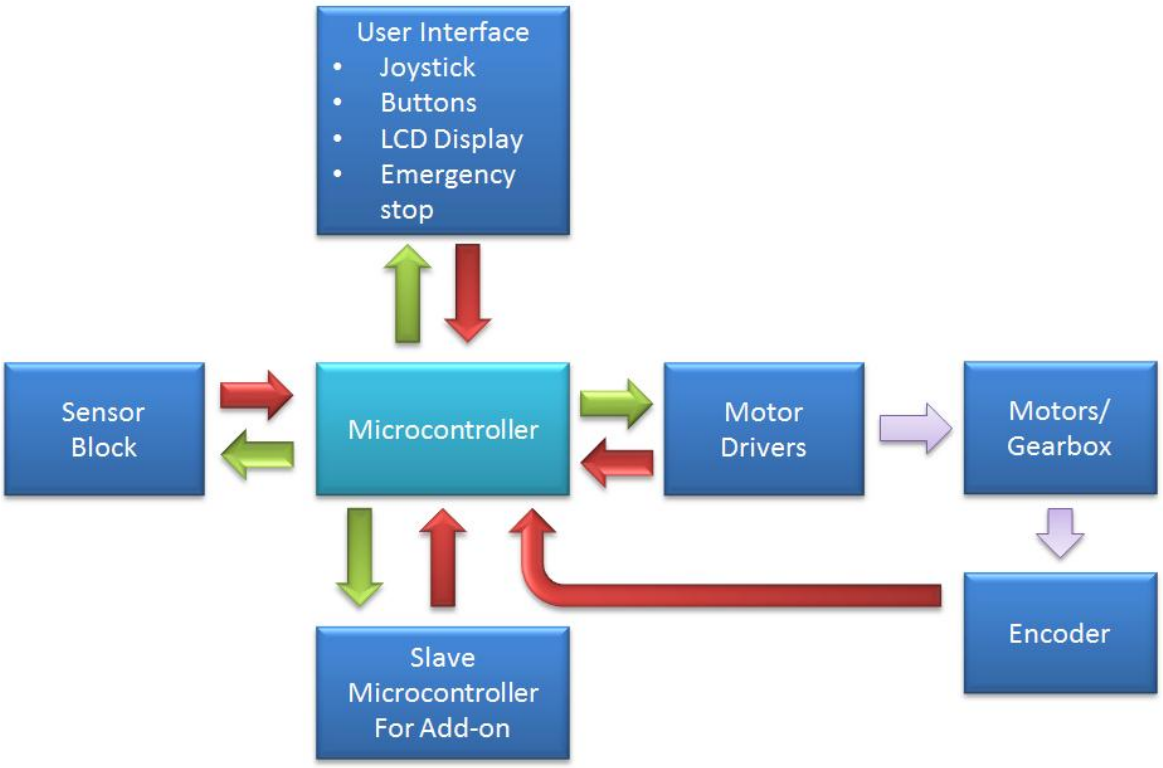
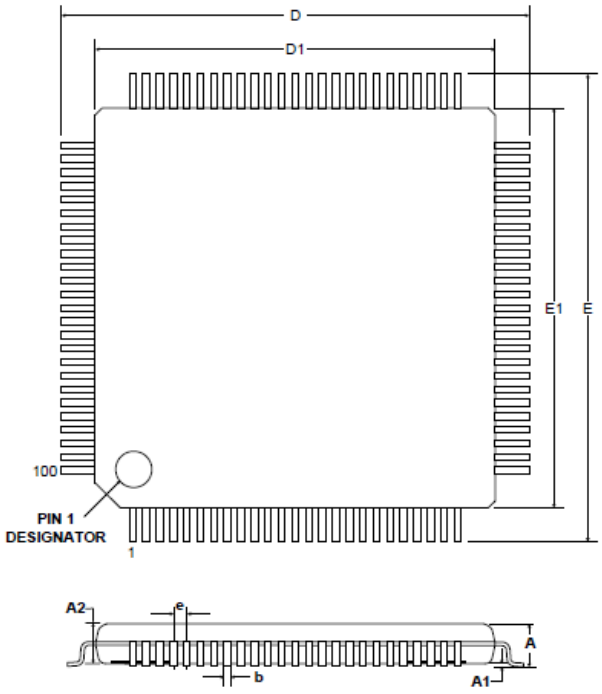


Figure 5-1: Block diagram of system

Figure 5-1 shows a simplified block diagram of the overall system. The central (master) microcontroller can be seen in the centre of the diagram with everything else communicating with it.

5.2 The Silicon Labs 8051 Microcontroller

Details of the features of the "Silicon Labs" C8051F020 can be found in section 3.4. Figure 5-2 shows the dimensions of the microcontroller and the foot print of the 64 pins. It shows the compact size and low profile of the controller that is ideal for the purposes of this project.



	MIN (mm)	NOM (mm)	MAX (mm)
A	-	-	1.20
A1	0.05	-	0.15
A2	0.95	1.00	1.05
b	0.17	0.22	0.27
D	-	16.00	-
D1	-	14.00	-
e	-	0.50	-
E	-	16.00	-
E1	-	14.00	-

Figure 5-2: Dimensions of C8051F020[45]

5.3 Platform Controller Board

The requirements for the platform controller board are as follows

- J-Tag port
- Joystick connection
- Four motor driver ports
- 5V switching regulator to supply power to logic circuits on micro-board and driver boards
- 3.3V linear regulator to supply power to the 8051 microcontroller
- Two serial ports, one with 5V supply (tilt sensor) and one with 3.3V (slave microcontroller)
- LCD screen port
- Four encoder ports
- Two brake relay ports with switching transistor

- Logic for inverted and delayed PWM
- Spare port for added flexibility
- Battery charge indicator circuit

All critical ports had to be established and dedicated to their assigned function. This was achieved through using the port assignment table in [32] for the integrated functions, such as serial communication, PCA used for PWM, and external interrupts. The analogue inputs that required the high resolution inputs needed to be routed to ADC0 while the remaining would be assigned a pin from Port 1 in an order that enabled the easiest track routing on the board.

5.3.1 Detailed Functional Block Diagram

A detailed functional block diagram of the platform controller board is shown in Figure 5-3. It shows all the components and ports on the board as well as the microcontroller resources going to each.

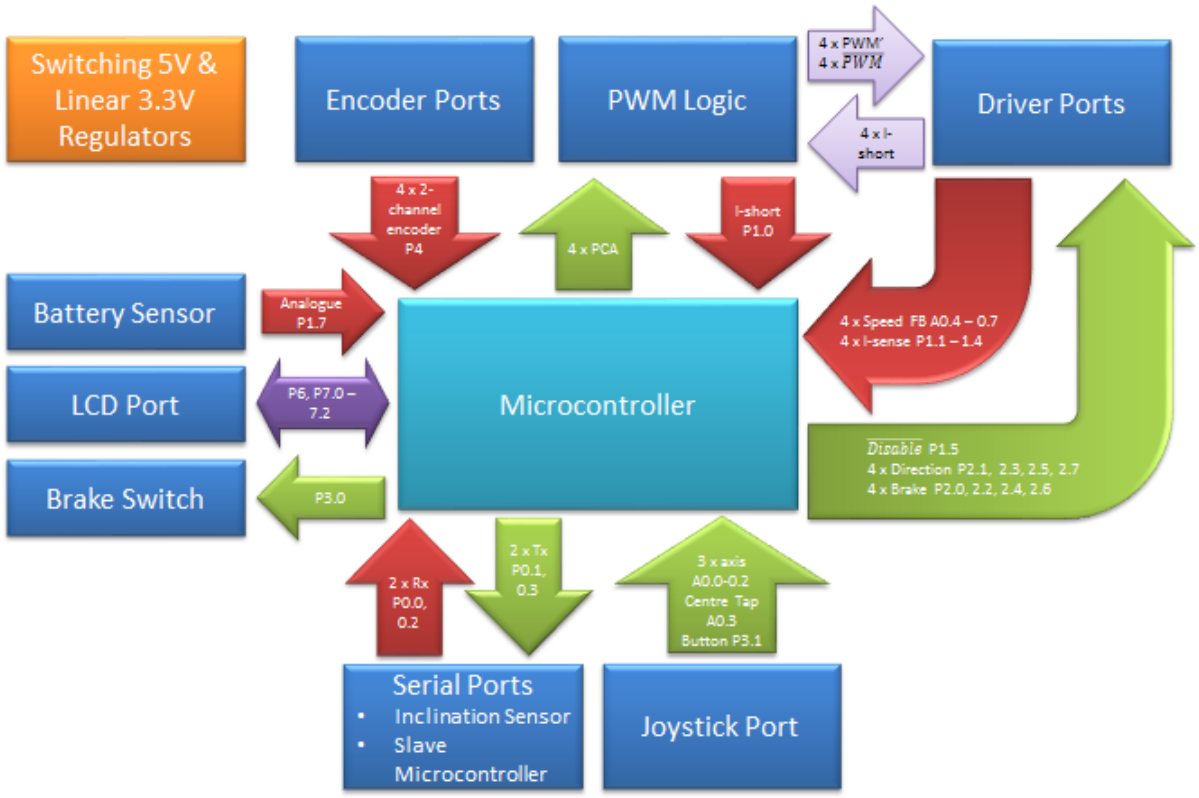


Figure 5-3: Functional block diagram of platform controller board

5.3.2 Circuit Diagram and Explanation

The logic for producing the delayed and inverted PWM signals is shown in Figure 5-4. The four PWM signals generated by the microcontroller's PCA are fed into a PLD. The PLD

simply inverts these inputs, thus generating the inverted PWM signal. This is then passed through a low pass filter to take advantage of the preceding inverters hysteresis to generate the delayed PWM signal.

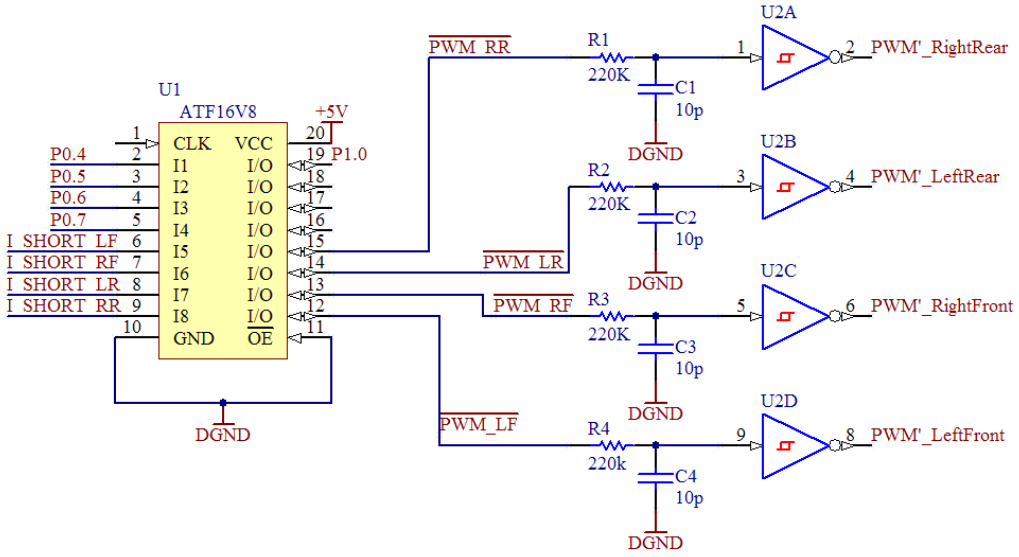


Figure 5-4: PWM logic

There are three isolated 3.3V and two isolated ground supplies created by the linear regulator shown in Figure 5-5. The VDD supply powers the 8051 microcontroller with an isolated supply to ensure a clean voltage. The +3V3 supply is for the JTAG interface which is recommended to have an isolated supply by the manufacture. Finally the analogue components of the microcontroller have their own supply, AV+. The zero Ohm resistor (R4) ensures that the common ground between the digital and analogue aspects of the PCB are connected in one place providing some isolation for the sensitive analogue components.

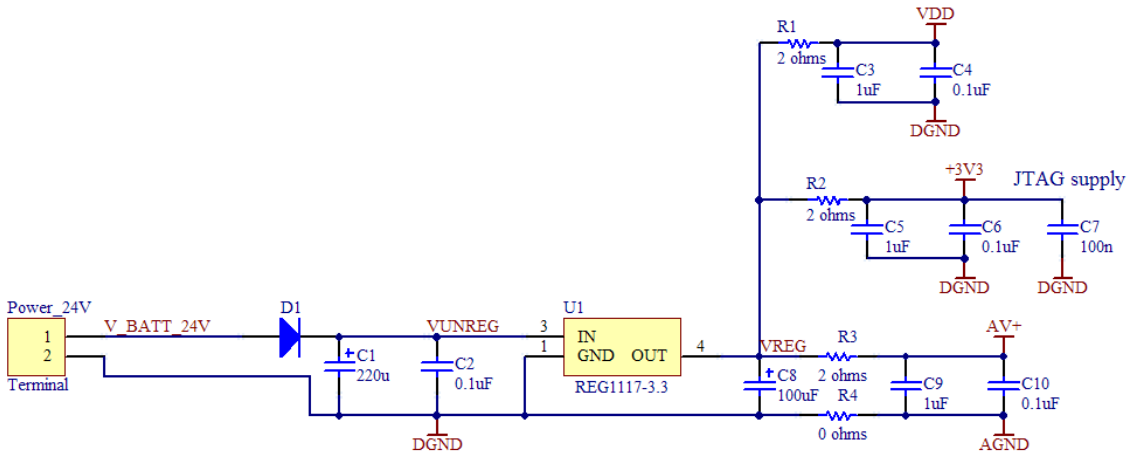


Figure 5-5: 3.3V regulator

Another important aspect of the controller board is the motor driver ports. These had to match the port on the driver and provide some additional functionality. This centred on the fact that the output of the current sensing IC had a maximum of 5V at 100A and a minimum of 2.5V at 0A. This is above the maximum input of the ADC. This is addressed by the voltage divider comprised of R1 and R2 (Figure 5-6). The two 1nF capacitors (C1 and C2) are to smooth any small voltage spikes that are a result of the transmission from the driver to the microcontroller.

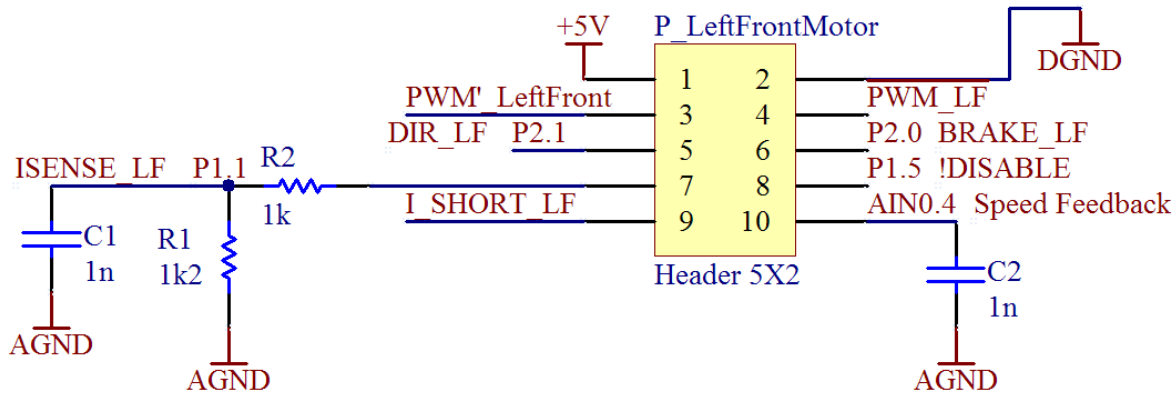


Figure 5-6: Motor driver port

The relay brakes on the RAD right angled motors are the heavy duty variety to enable the through shaft for the encoder. Because of this they require a large amount of power to run. At the 24V rated voltage the relay draws around 600mA. This causes the relay to heat up. With two relays running there is a continuous draw of 1.2A; this is significant from a battery conservation perspective. It was found that the relay would activate at voltages as low as 7V; this voltage could be lowered further to 4V before the relay would switch off. At these voltages the relays were drawing less than 100mA. The simplest way to lower the voltage to the relay is to pulse the battery voltage using a PWM signal. Figure 5-7 shows how this is achieved. Q1 is used to pulse the low side to lower the perceived voltage going to the relay. D2 is used to clip the gate voltage and protect the transistor and D1 is a flywheel diode for the relay back EMF.

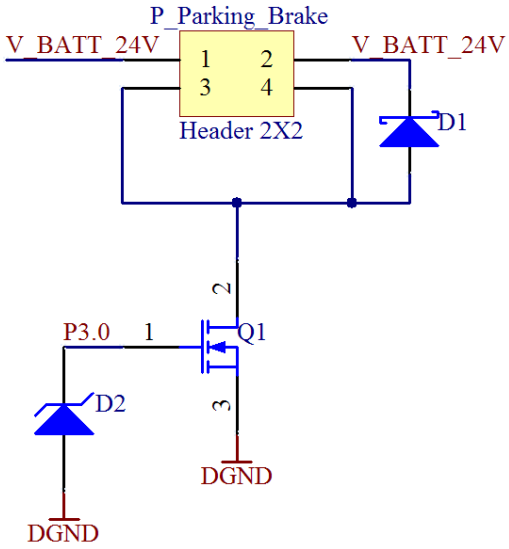


Figure 5-7: Relay brake circuit

The battery sensor works by providing a voltage proportional to the battery level from which the remaining charge can be estimated. This value needs to be in the range of 0 to 2.4V in order to take full advantage of all the eight bits available on ADC1. The arrangement used for the platform is two 12V batteries connected in series to provide a 24V supply. To give a good estimate of the battery life we need to know the voltage between the values of 22V and 30V. To achieve this, an offset difference amplifier is required. Figure 5-8 shows the circuit diagram of the battery level sensor.

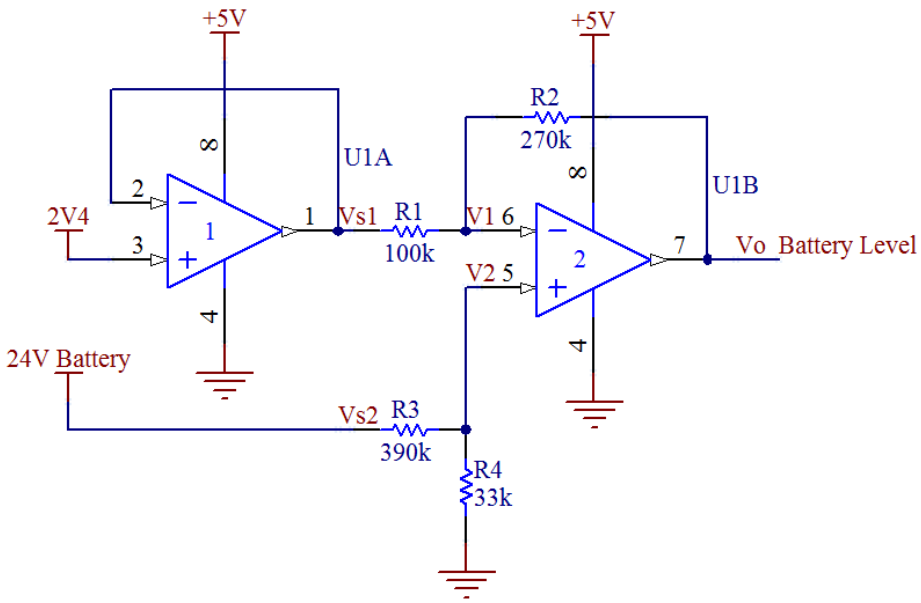


Figure 5-8: Circuit schematic of battery level sensor

The following equations explain this circuit:

$$v_1 = v_{s1} - R1\frac{v_{s1} - v_o}{R1 + R2} \tag{5.1}$$

$$v_2 = v_{s2}\frac{R4}{R3 + R4} \tag{5.2}$$

As 2.4V is the maximum voltage, v_2 will be arbitrarily set to this value when the battery is fully charged, i.e. 30V. Therefore:

R3 = 390kΩ

R4 = 33kΩ

As the maximum output required is the same as the reference voltage (2.4V) and v_1 must equal v_2, the following occurs when v_{s2} = 30V.

$$v_1 = 2.4V - R1\frac{2.4V - v_o}{R1 + R2} = 2.4V \tag{5.3}$$

Solve for v_o gives -

v_o = 2.4V, where R1 and R2 do not affect outcome.

When v_o = 0, $v_1 = v_2$, and v_{s2} = 22V, therefore v_2 = 1.7V.

$$v_1 = 2.4V - R1\frac{2.4V - 0}{R1 + R2} = v_2 = 1.7V \tag{5.4}$$

Making R1 = 100kΩ, R2 is calculated to be equal to 240kΩ.

Because of the E12 resistor constraints the actual output falls between 0V and 2.2V as shown in Figure 5-9.

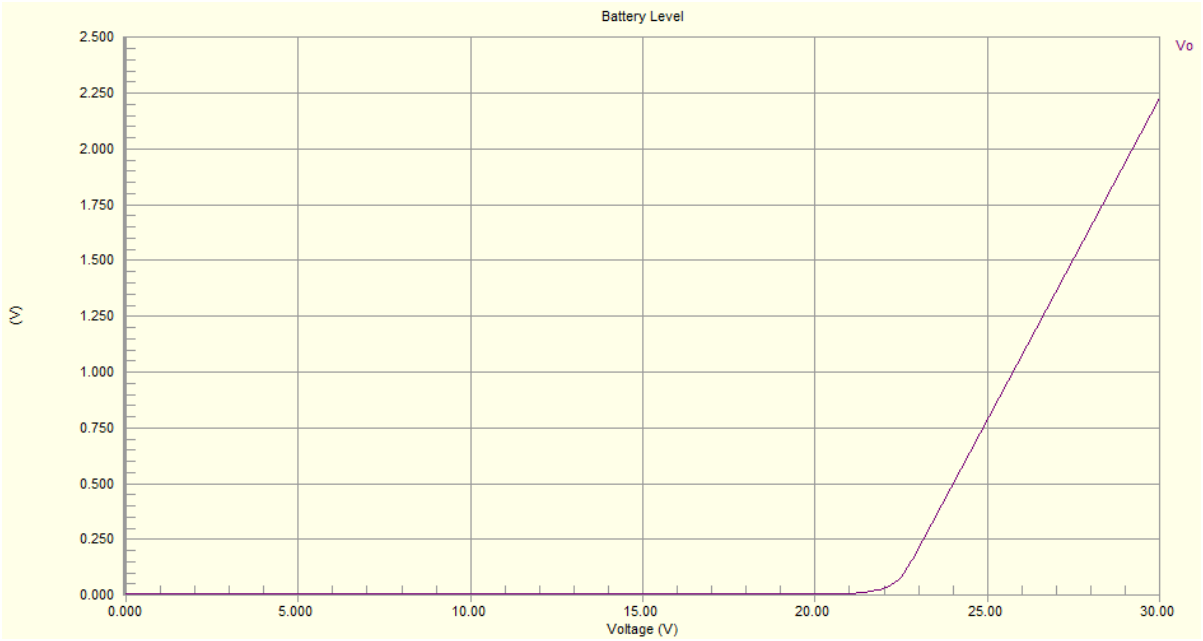


Figure 5-9:0 to 30V ramping simulation showing battery level sensor output

5.3.3 Assembly Process and Testing

The design of the PCB board was much more complex than the driver boards. This was due to the large number of pins being used on the microcontroller. This complexity was made worse by the fact that the header pins could only be soldered on one side, increasing the number of vias and space needed for each header.

The microcontroller was the only added complexity in the assembly of this board when compared to the driver boards. Because of the copper laying process used, the vias located under the microcontroller IC need special preparation so they do not protrude from the surface of the board and interfere with the bottom side of the microcontroller IC. This involved creating indents in the vias using a drill press and wide hole-punch. Then thin wire and solder was used to establish a connection to the other side of the board. Any solder left protruding from the indented hole was filed off so it was flat with the rest of the board. The microcontroller could then be soldered to the board (Figure 5-10). This was followed by the rest of the components.

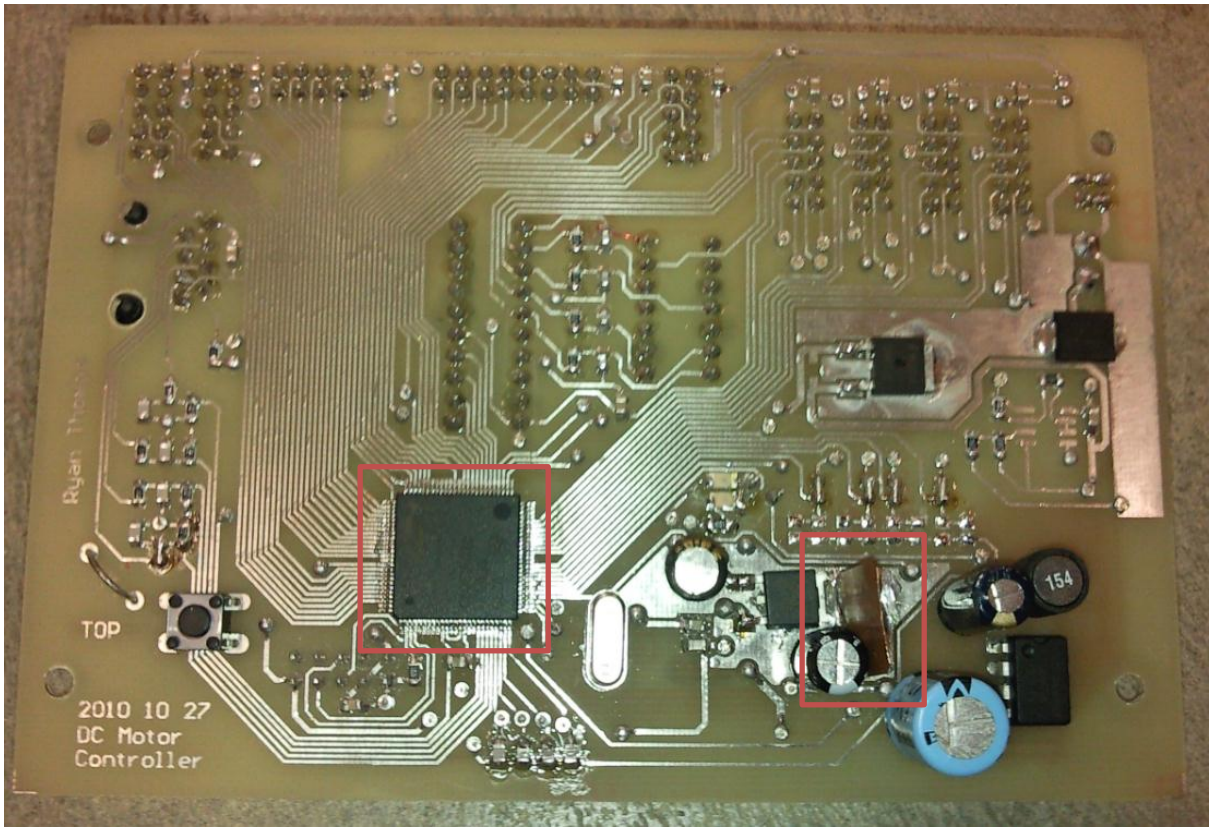


Figure 5-10: Microcontroller on the main controller board

With the board completed it was found that the microcontroller worked and programs could be loaded onto it. One problem that was apparent was the fact that the 3.3V linear regulator was generating a lot of heat when dropping the voltage from 24V. Several measures were taken to deal with this problem. The first was to increase the size of the resistors in series with the indicator LEDs to limit the current drawn by these. The next step was to add a small copper heat sink to aid in the cooling of the regulator (Figure 5-10). Finally provisions need to be made if the previously mentioned steps are not enough and the heat becomes a problem. The only device stopping the voltage being dropped before the microcontroller board is the battery level detector. However, if another battery detector is included on the slave microcontroller board this information can be sent to the main board for processing, freeing up the main controller to run off a lower voltage.

The relay circuit will also benefit from a lowering of the supply voltage. As the transistor used to control the motor's relay brake voltage are driven from a digital port rather than a PCA produced PWM signal, it is difficult to produce a signal that will be above the threshold of human hearing. A new fast (around 20kHz) interrupt will need to be implemented using unnecessary microcontroller resources. If the voltage was dropped to say 9V the transistor

could be turned on permanently, therefore not producing any noise or using processing resources.

5.4 Gripper Controller Board

The gripper controller board will need an integrated motor driver in order to control the gripper motor. It is also important for a form of current sensing to be present on the board. This will be used to detect when the gripper has gripped the footboard of a bed as the motor will draw more current as the footboard jams in the gripper fingers. Although a smaller and cheaper 8051 microcontroller could have been used with the lower requirements of this board it was decided to use the same model as the platform controller board as these are readily available and cut back on circuit board design time.

Because of the stress the 3.3V regulator was under on the platform controller board it was decided that the new microcontroller would be run off the 5V supply already available with the use of another 3.3V linear regulator. This created a problem with the two serial ports on the platform controller board. As it was anticipated that the gripper microcontroller would run off the same 3.3V supply as the platform microcontroller, a 3.3V pin was used for one of the serial ports the other using 5V for the inclination sensor. The simplest way around this is to use the inclinometer sensor port on the platform controller board as the communication and 5V supply and use the gripper controller to interface the inclinometer and send this information to the platform controller board.

5.4.1 Detailed Functional Block Diagram

Figure 5-11 shows the functional block diagram for the gripper controller. Each block in this diagram will be explained in detail below.

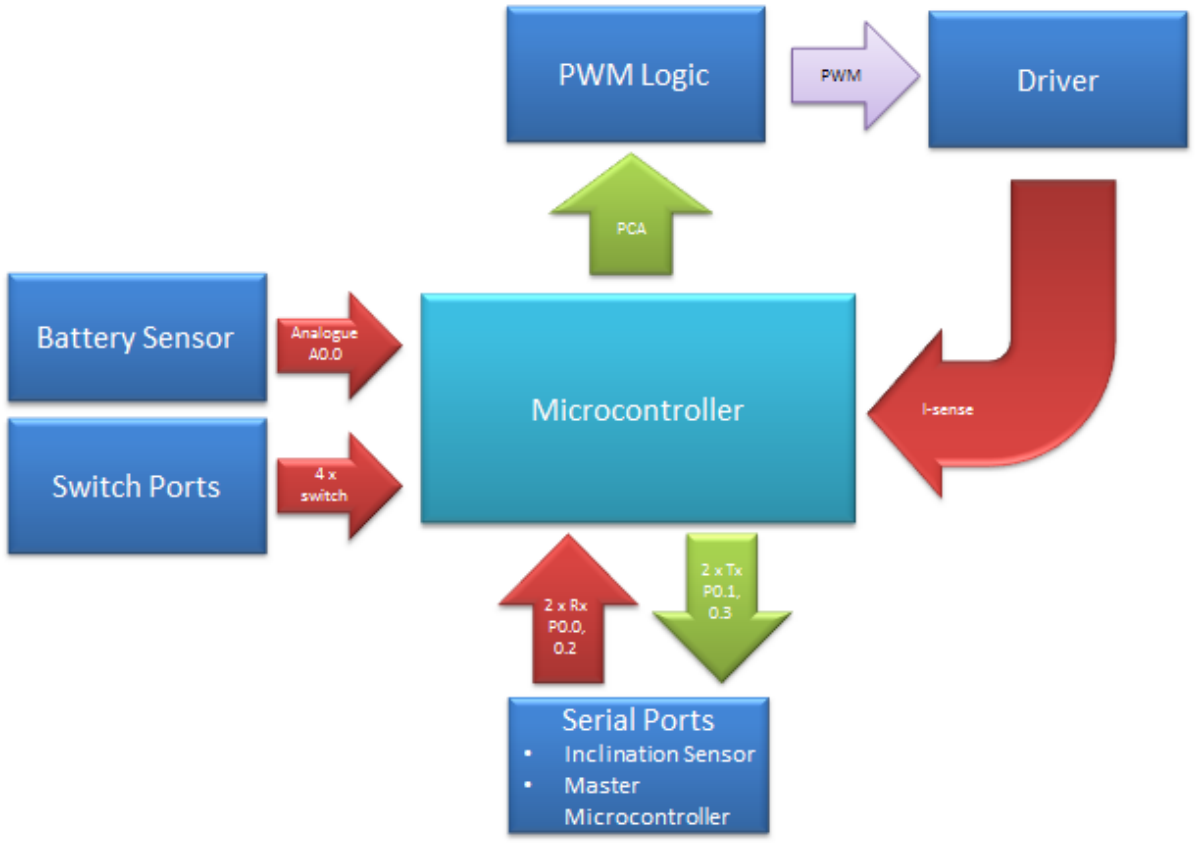


Figure 5-11: Functional Block diagram of gripper controller

5.4.2 Circuit Diagram and Explanation

For the gripper driver it was decided to go with a simpler design than the drive motor drivers. As the current requirements of the gripper motor are lower, a relay system can be employed. Figure 5-12 shows the configuration of the gripper driver. The relay enables the polarity of the motor terminals, thus changing the direction of the motor. The speed of the motor is controlled by the MOSFET (Q2); it is driven by three AND gates to ensure clean gate operation. These AND gates are controlled by the PCA's PWM output at port 0.4. D1 and D2 are flywheel diodes for the motor and relay respectively. The 0.01Ω resistor and SwitchSelect tag are part of the current sensing circuit.

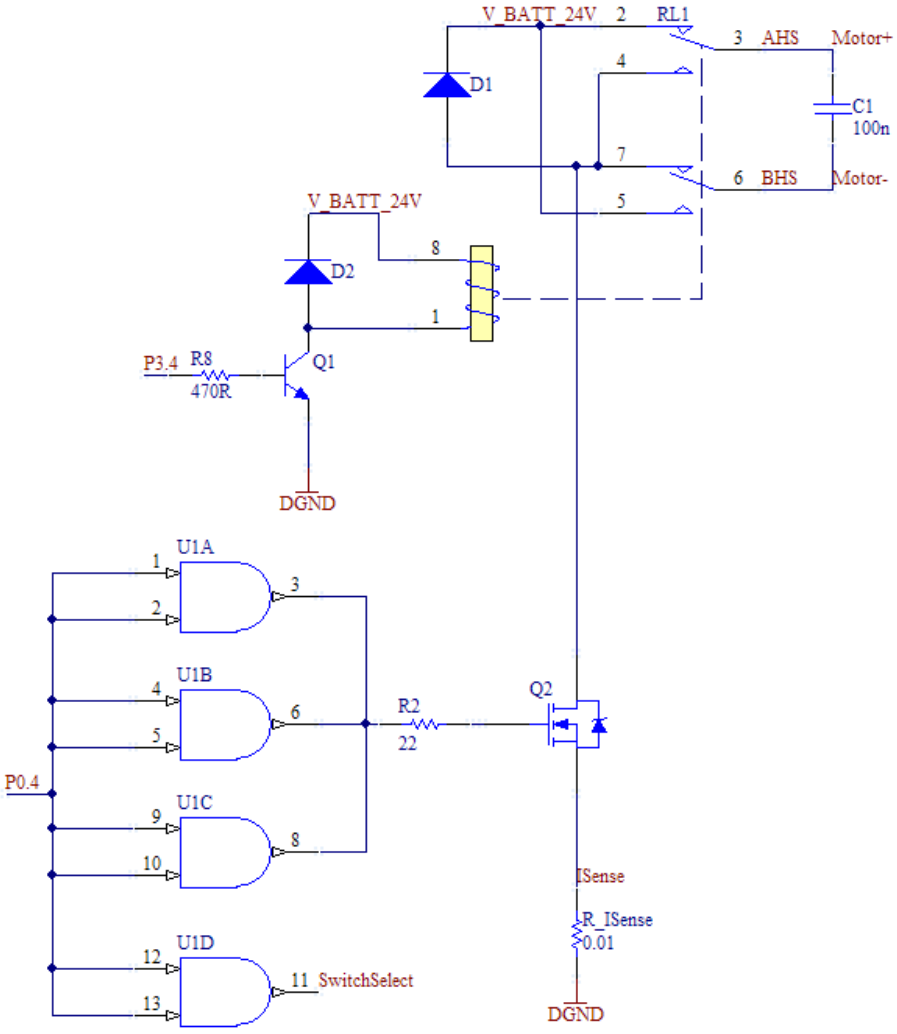


Figure 5-12: Gripper motor driver

The current sensing ICs that were used in the driver board circuitry were no longer available at the time of producing the gripper controller board. Because of this and the fact that the current will not be as high, a shunt resistor approach can be implemented. Figure 5-13 shows the circuit used to condition the signal for the ADC on the microcontroller. First the voltage drop across the resistor is amplified by the op-amp to take advantage of the full range the ADC can offer (2.4V). This is then sampled only when the motor is switched on by an analogue switch and capacitor arrangement driven by the PWM signal.

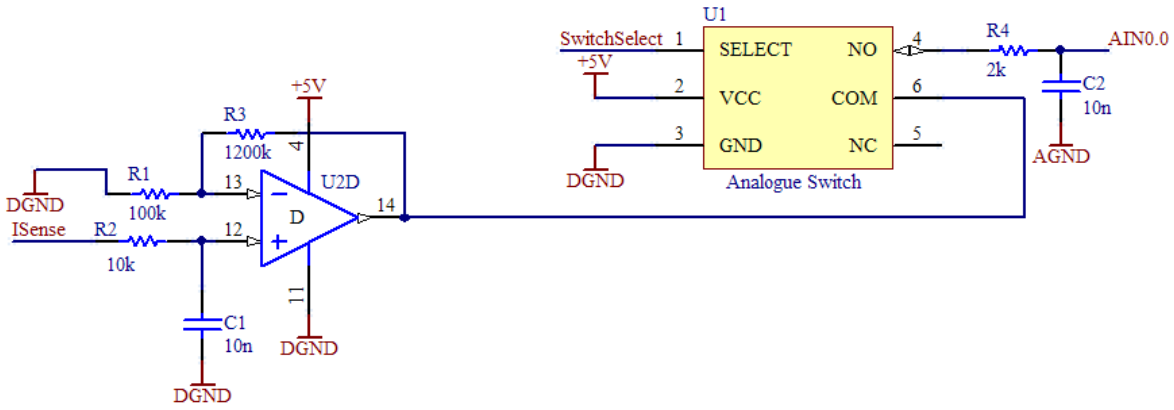


Figure 5-13: Current sensing circuitry

5.4.3 Assembly Process and Testing

The only difference in assembling the gripper controller board was the mounting of the large relay required for the motor driver. The relay was capable of handling 8A of continuous current and up to 16A for short periods. It had tabs that required large through-holes to be solder to the PCB. Care had to be taken to have large tracks where there was to be the high motor current flowing.

Upon testing the driver with the small test motor, it was found to work well. This continued when testing with the gripper motor was undertaken. One issue found when testing the motor was that it would turn on briefly when the system was first given power. This was found to be caused by the PCA taking time to initialise and remaining high for long enough for the motor to start moving. The best way to overcome this problem was to swap the AND gates used to drive the MOSFET for NAND gates. Fortunately this was a direct swap that needed no PCB modification, the only adjustment being inverting the PCAs PWM signal in software.

5.5 Circuit Board Box Design and Placement

To provide protection for the circuit boards it is important to have them mounted in a robust case. The driver boards where able to be mounted two per case by stacking them on top of each other. This arrangement then fitted on the side of the platform where the driver boards would be out of the way and easily accessible for wiring to both motors and the controller board. Figure 5-14 shows one pair of stack mounted driver boards.

Figure 5-14: Mounted double stacked driver boards

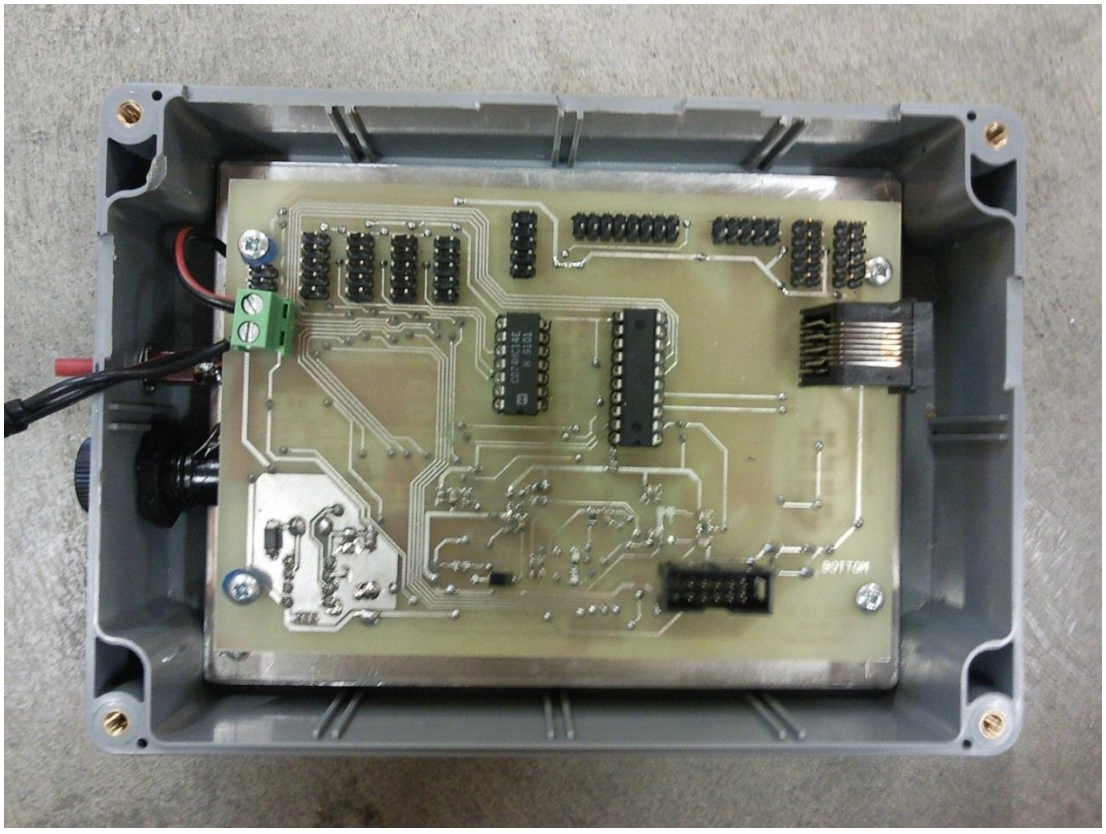


Figure 5-15: Platform controller board mounted within the case

The platform controller board used the same plastic case. As this board would be running on relatively low current it was decided to mount a fuse and isolation switch on the case. Also because the board was too small to use the case's mounting holes, an aluminium plate has been used to mount the board. All these features can be seen in Figure 5-15.

5.6 Chapter Summary

The control circuit boards are an essential component for the control of the platform. They must enable the software on the microcontroller to control every aspect of the platform. An inverter with hysteresis and a RC filter has been used to create the delayed PWM signals required for the current sensing of the driver boards. A 3.3V linear regulator provides power to the microcontroller while a 5V switching regulator provides power for all other systems on the platform. Two operational amplifiers provide battery level detection to the ADC of the microcontroller and a switching surface mount MOSFET controls the voltage seen by the parking brake relays. The gripper controller board incorporates a motor driver for the gripper motor. This uses a relay for directional control and a single MOSFET for speed. Current sensing is also incorporated using a shunt resistor and an analogue switch to monitor the load on the gripper motor so that it is known when a bed footboard is gripped. Both boards have an array of connectors to service the various components and communicate with each other.

6 Software

This chapter will explain how the software has been used to integrate the components of the platform to perform the tasks it is designed to accomplish. It will also look at the reasons behind the programming choices made. The software is a very important part of the system as it will enable the platform to be easily and safely controlled. Great care has been taken to ensure the smooth motion of the platform and that the safety precautions that the hardware allows have been implemented. The development environment used is the "Silicon Laboratories Integrated Development Environment (IDE)". This uses the "C" programming language for programming embedded devices. The complete code for the platform can be found in Appendix B.

6.1 Platform Controller

The platform controller is the main controller of the system. This controller is responsible for receiving control signals from the user and has control over all the movements of the platform through the motor driver interface and encoder feedback modules. It also controls a slave microcontroller through a serial port for the add-on bed gripper functionality. The software for this controller is required to enable smooth and simple control of the platform while monitoring all sensory systems of the platform.

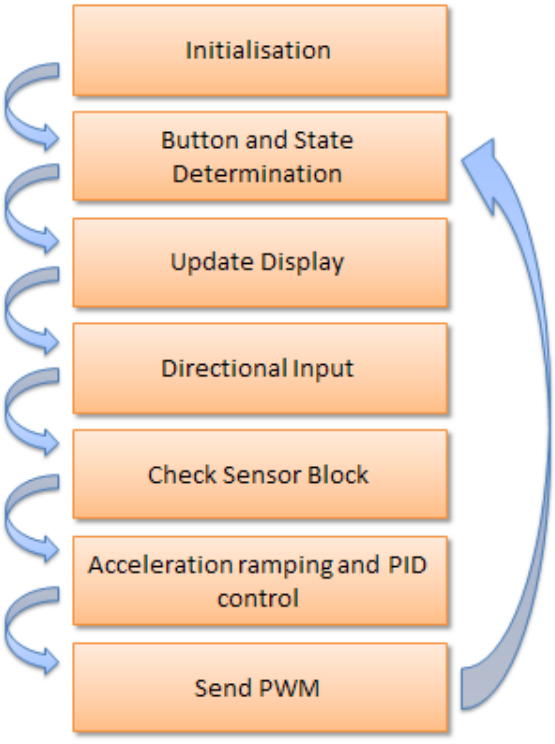


Figure 6-1: Platform software flow diagram

6.1.1 Platform Controller Software

Figure 6-1 shows the functional blocks of the platform controller software. When the platform is first turned on, after all the required functions have been initialised, the platform microcontroller enters the system initialisation function. This starts by sending a continuous stream of the character "S" followed by a stop character over the serial port to the add-on microcontroller (Figure 6-2). It then waits for a reply from the add-on microcontroller that indicates what the add-on is and therefore indicates what mode the platform should enter. Because there is only one potential add-on at this stage, the bed mover, there is only one, default mode the platform enters. If the initial communication fails the platform will not operate.

```
SBUF0 = 'S';                    // load the serial buffer
while ((~SCON0 & 0x02) == 0x02); // wait for transmission
TI0 = 0;                        // clear TI0

SBUF0 = 0x0A;                   // load the serial buffer
while ((~SCON0 & 0x02) == 0x02); // wait for transmission
TI0 = 0;                        // clear TI0
```

Figure 6-2: Code for serial transfer

From here the platform microcontroller enters the main operational loop. The operation of the joystick button is the first aspect handled in this loop. The main function of the button is to operate the gripping function. If the button is pressed while the platform is stationary the platform microcontroller will send the character "G" over the serial port. It then waits for a reply from the gripper microcontroller of one of three states. "1" indicates gripper closed, "2" indicates gripper open and "3" indicates that a bed is attached. The platform microcontroller will then adjust the speed profiles to the requirement of the state it is in. The other function the button performs is involved with the speed control. There are many options for controlling the speed of the platform. The problem with joysticks is that inexperienced operators, such as nurses, tend to use the extremes of its movement and expect the platform to go at the right speed, i.e. they are not used to controlling speed and direction with a single input. To counteract this it is beneficial to have different speed settings for different situations. Having the platform move slower in confined spaces would be highly desirable. One way to achieve this is to use the joystick button as a speed selection controller. In its current configuration the platform starts in the slower speed mode. If the joystick button is pressed while the platform is in motion the maximum speed limit of the platform is increased.

The button can be pressed again to lower the speed or the stopping of the platform will lower the speed again.

There are many options for the handling of the ease of control of the platform and because of the flexibility of the 8051 microcontroller any number of these could be implemented. Options include using a logarithmic scale for the joystick input resulting in slow speed for everything other than the extreme positions of the joystick input range. Another is to have a timer that will increase the speed of the platform if a constant joystick input is detected for a set amount of time. The button approach was chosen because it is simple to understand and gives the operator the utmost control over the system.

The display uses a state machine for what it is to display at any one time. The default display is the battery level indicator as this is the most important piece of information for the operator to know. Other states include indicating what the gripper is doing, while there are also diagnostic states like motor duty cycle and motor current drawn.

The directional input is a very important aspect of the overall system. There are two different approaches to ADC conversion. One is to set up a timer that causes an interrupt that starts the ADC conversion, an interrupt is triggered when the conversion is complete and updates a variable with the new information. The other approach is to use a software command to start a conversion, then wait till the conversion is completed and assign the value to a variable. The polling method was used for this application as the program followed a flow down approach that is more suited to controlled polling. This is because there is no need for this to be a time critical operation. Therefore this can be left in the main control loop where less computational resources are used and the polling can be interrupted by more important, time critical, operations. Figure 6-3 shows the acquisition of the joystick input for the x-axis. This input is then put through a dead-band check and scaled down to a required speed value. This compensates for the different gearing of the front and rear motors.

```
AMX0SL = 0x00;        // select input channel
small_delay(20);      // let input channel settle
AD0BUSY = 1;          // commence conversion
while ((ADC0CN & 0x20) ==0);  // wait for conversion
Vtx = ADC0 - 2170;        // save and offset ADC value
ADC0CN &= ~(0x20);        // clear AD0INT
```

Figure 6-3: Code for joystick x-axis acquisition

The sensor block for this controller consists of the motor encoders and the current sensing device. Current sensing is handled in much the same way as the joystick except ADC1 is used. The sensed current is saved to a variable for each motor and used to monitor the operation of the motor. There are several ways the encoders can be integrated with the microcontroller including a novel way to gain higher resolutions, as discussed in [46]. One approach would be to use external interrupts to detect each change on the encoder data lines. However because there are four motors, therefore eight data lines, there are not enough external interrupts on the microcontroller being used. One way to overcome this would be to use a Programmable Logic Device (PLD) in a state machine configuration that could detect when there is a change on the encoder inputs and produce a signal to be fed to the microcontroller so that it can interpret the change. Because of the number of PLD registers needed to achieve this, a large PLD would be needed. Another option is to use a fast high priority timer interrupt service routine in the microcontroller. This can be initialised to interrupt at 10 kHz and check the data lines of the encoders. In order to save microcontroller resources the new data line values could be exclusively ORed with the last values to see if any have changed and if there is a need to continue with the routine. However, if we work off the fact that the microcontroller will need to handle the worst case scenario at some stage, it can be concluded that it should be able to handle this scenario every time. Therefore, this approach was taken as there is no extra hardware required and it ensures there will not be any missed encoder counts.

The following code segment (Figure 6-4) shows how the encoder data lines are converted into an incrementing (forward), or decrementing (backwards) integer.

```
static code     char hall_tab[] = {0, 1, -1, 2, -1, 0, 2, 1,
1, 2, 0, -1, 2, -1, 1, 0};
char edge;
index = (P4 & 0x03)<<2;
index |= last_halls0;
edge = hall_tab[index];
posn[0] += edge;
last_halls0 = index >>2;
```

Figure 6-4: Encoder microcontroller code inside interrupt service routine at 10 kHz

This code relies on the fact that the character associated with the new values on the two data lines, combined with the previous values, equals the corresponding value in the “hall_tab” array. This method can cope with a missed transition indicated by the “2” values in the array.

However, this is only for the forward direction. To compensate for the missed transition whist travelling backwards the following code is necessary (Figure 6-5).

```
if (edge == 2 && dir == DIR_RV)
{
     posn -= 4;
}
```

Figure 6-5: Encoder microcontroller code for missed backwards transition

If the motor is going backwards and the value gained from the "hall_tab" array is two, four counts are deducted to compensate for the two that have already been added. This position variable ("posn") is taken and subtracted from the old value periodically, with a timed flag, to generate a variable that reflects the current speed of the motor.

The microcontroller takes the encoder speed feedback information along with the joystick input and performs acceleration ramping and PID control of the drive motors. The ramping of the acceleration is achieved by limiting the speed at which the "required speed" variable, gained from the joystick input, can increase to once every 50ms. This is a form of digital low pass filtering that is more aggressive for accelerating (i.e. slower acceleration, add to two to PWM) than for deceleration (so the platform stops faster, subtract four from PWM). PID (Proportional Integral Derivative) controller has been used to match the drive motor encoder feedback to the required speed. A good description of modern PID control can be found in [47]. Because there are two different motors employed in the platform, there needs to be two different PID loops to compensate for any performance differences in the motors. Once the motor PWM values are calculated the absolute value is taken, saving the direction to be used as the directional output to the driver, and sent to the PCA for PWM generation.

6.2 Gripper Controller

The gripper controller is responsible for the control and operation of the bed gripping mechanism. It knows what state the gripper is in and is tasked with relaying this information to the main platform controller board.

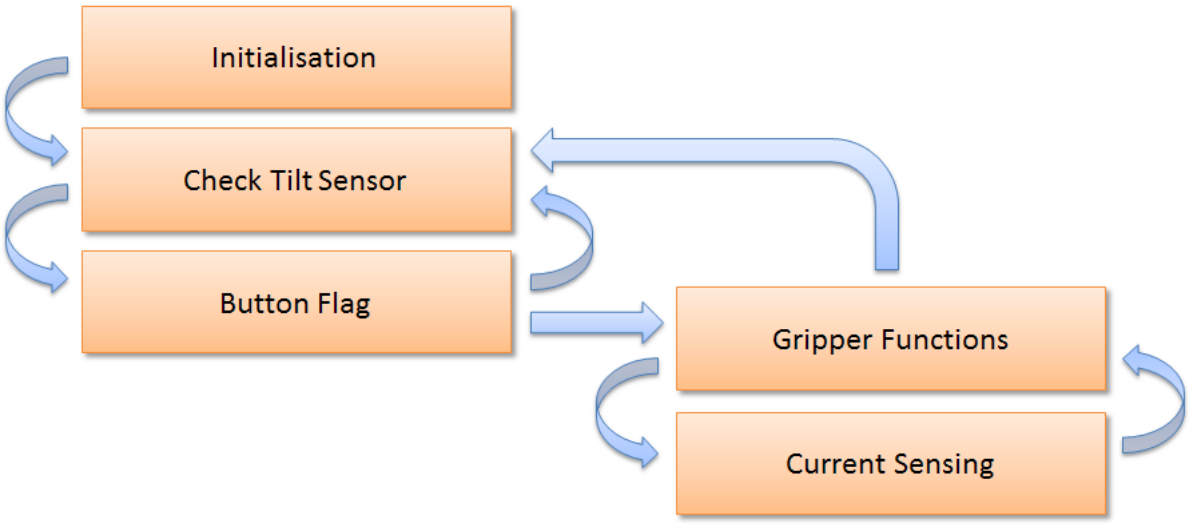


Figure 6-6: Gripper controller flow diagram

6.2.1 Gripper Controller Software

Figure 6-6 shows the gripper controller flow diagram. After the required functions have been initialised the microcontroller enters a system initialise function. First it waits for the start-up character to be sent by the platform controller. Once this is received the micro switches that indicate that the gripper is open or closed are checked. If the gripper is closed a "1" character is sent to the platform microcontroller indicating that the platform does not have a bed attached and the gripper is closed. If any other case the gripper controller will proceed to close the gripper by calling the "Close_Gripper" function. This function first sends the character "C" over the serial port to inform the platform controller that the gripper is closing. It switches the relay to the correct configuration and ramps the PCAs PWM signal. It then continuously monitors the closed gripper switch and the current sensing ADC input. If the gripper closed switch is triggered the gripper controller sends the character "1" over the serial port to the platform controller informing that the gripper is closed with no bed attached. If the current sensing ADC input exceeds a threshold for a period of time the gripper will stop closing and send a "3" over the serial port, indicating that the gripper is closed with a bed attached.

From here the gripper controller performs two tasks. It continually monitors the inclination sensor so that if the inclination breaks the 5˚ limit an "H" character is sent to the platform controller informing the platform to stop moving. The operation of the tilt sensor is explained in section 3.8. From a programming standpoint the "atof" function is used to convert the

ASCII characters into a floating point decimal value. The following code (Figure 6-7) achieves this:

```
for (x=1; x<6; x++)
{
     xaxis[y] = received_str[x];
     y++;
}
xaxis[5] = 0x00;
y = 0;
for (x=9; x<14; x++)
{
     yaxis[y] = received_str[x];
     y++;
}
yaxis[5] = 0x00;
xResult = atof(xaxis);
yResult = atof(yaxis);
```

Figure 6-7: Code for the integration of tilt sensor

This codes sorts the 5 numeric ASCII characters, including a decimal point, into arrays so they can be operated on by the "atof" function and compared to the threshold value. The other function the gripper controller undertakes is to monitor the button flag sent over the serial connection by the platform controller. This causes the next gripper function to be initiated. If a bed is attached or the gripper is closed this causes the gripper controller to enter the opening function. The relay is switched and the PWM signal is ramped up. The open gripper switch is then monitored to determine when the gripper is fully open. When this switch is triggered a "2" character is sent over the serial connection to inform the platform controller of the state of the gripper. Current sensing is also undertaken but as this should not occur for gripper opening, the gripper controller will stop the PWM and enter an error state if it is triggered. If the gripper is open and the button flag is triggered the gripper controller will enter the closing gripper state.

6.3 Chapter Summary

The software for the platform was written with controllability and safety as a high priority. There are two separate microcontrollers running different software that communicate with each other through serial communication to provide the bed gripper functionality. The main function of the platform controlling software is to provide acceleration ramping that enables

the platform to be easily controlled. Other functions include controlling the maximum speed of the platform depending on the bed attached state and user input through the interface button. This software also determines when to signal the other controller to use the gripper function of the platform. The software on the gripper controller monitors the state of the gripper and controls the acceleration and speed on it opening and closing. It is also responsible for monitoring the inclination sensor and signalling the platform controller when the inclination is to high so the platform can stop.

7 System Functional Testing

This chapter will consider the testing of the platform and show how this was achieved. The ultimate success of this project will be determined by the machine's ability to move beds in a confined space. More specifically, this involves moving a heavy hospital bed, with a patient, from a patient room to the operating theatre. The system is awaiting clearance for a full field trial in a hospital setting. Nevertheless, there are many other tests that can be undertaken to ensure the successful implementation of the work. The testing should ensure that all the sub-systems of the platform are functioning as expected.

7.1 Testing of Platform

After the wiring of the platform was completed each driver board was tested individually to see if the power LED would light. After this each board was tested in turn with a motor connected, but the system lifted and placed on blocks (i.e. no load), and an open loop control algorithm with joystick input. After each driver and motor was found to be functioning correctly it was time to test all in unison. The next step was to incorporate the encoders and a closed loop control algorithm. Again, each motor was tested separately before all four were run at once. With all four motors running in unison with closed loop control a load test was conducted. During this testing the problems outlined in section 3.7.8 were addressed. After this, further load testing was undertaken by running the platform into solid fixed obstacles. This showed that the platform could handle high loads that will be encountered with a bed attached.

7.2 Testing of Gripper

The initial testing of the gripper involved connecting the double rack and pinion motor directly to a power supply to test the smooth operation of the gripping arm on the nylon runners as well as the triggering of the limit switches. Next the previously tested gripper controller board (section 5.4.3) was attached and the gripper could be tested with signals from the limit switches determining when to stop. From here the current sensing circuitry could be tested. This was done by physically stopping the gripper from closing and forcing it into the "bed attached" state. After some fine tuning of the system the attachment system was found to work well.

7.3 System Integration Test

With the two sub-systems working correctly in isolation it was time for system integration to occur. This involved testing the communication protocol between the two controller boards. After some tuning of the communication protocol, the initialisation, button and inclination sensor communication could be tested. Next the drivers and drive motors were re-tested to ensure that the addition of the second controller did not affect normal operation of the platform. The operation of the gripper system also needed to be re-tested to ensure that it continued to function properly. Also safety features such as disabling the platform while the gripper is moving needed to be added to ensure the safety of the operator and any other person in the vicinity of the machine. A problem found was that the platform tended to vibrate at the highest speed setting. This vibration was amplified by the mast connecting the user interface to the platform. It was noticed that this vibration was in sync with the sound made by the Mecanum wheels slapping on the ground, a solution is proposed in chapter 8. An added test for the gripper involved gripping a piece of wood that was cut to the width of an average hospital bed (900mm). This was inserted between the gripper fingers as they were closed to test how securely the gripper can grip an object. With the precursor test successfully completed the bed moving robotic platform is awaiting field tests.

8 Conclusions and Recommendations

Reducing workplace injuries is a major factor in reducing the overall cost of running a company. As the cost of accident compensation rises, it becomes economically viable to spend more on reducing these injuries. Mobile platforms are an investment that can aid in the reduction of workplace injury. The platform developed has the potential to be used for many transportation uses. These vary from pushing large objects to carrying heavy items and even mobile workstation capabilities. The omni-directional abilities of the platform also provide the advantage of it being able to be manoeuvred in confined spaces as it needs no room to change direction. This enables space to be utilised more effectively enabling further cost saving.

This project has overall been a success based on the fact that all the requirements outlined in Section 1.3 have been met. Trials still need to be performed to test the platform's ability to push a 300kg bed up a 1:12 slope. Although the four motors provide more than enough power to achieve this, there are the questions of wheel slip and the ability of the motor drivers to handle this continuous load. Mecanum wheels have lower static friction values than traditional wheels, although this may not be the case (section 2.2). Either way there is the potential for the wheels to slip as the platform does not use the bed's weight for traction. To counteract this, more weight would need to be added to the platform to increase the traction of the wheels, thus increasing the load on the motors and wheels. This may necessitate a redesign of the motor drivers. In any case the four motors on the platform are capable of outputting far more power than that of a human being manually pushing the bed. The bed mover meets other specifications very well. The omni-directional movement is very smooth and intuitive; it is also capable of moving in any desired direction, not just sideways. The one exception to this is near the maximum speed where the vibration in the platform becomes obvious. The vibration felt matches up to the sound of the Mecanum wheels slapping on the ground. This indicates that the vibration is caused by poorly designed Mecanum wheels and the fact that future iterations of the prototype should include improved Mecanum wheels. However, this vibration will be greatly reduced when a bed is attached to the platform as this will stabilise the mast. The control interface is very simple with just one button on a single joystick providing all the input necessary for controlling the platform, with the emergency stop being the only other input. The platform meets all size constraints and the gripper system works well and is simple to operate.

Although the overall system works well, there are many aspects that can be improved. One obvious improvement that was not part of the scope of the investigation is a shroud for the system. This would greatly improve the aesthetics of the machine and provide a better surface for cleaning which is very important in a hospital environment. It would also provide protection from the wiring and any potential shock hazard that would exist. Finally a shroud would provide some protection for the feet of the operator and others in the vicinity. The inclusion of a rubber skirt around the base would provide some warning to anyone whose feet were about to be run over. If this proves to be a problem with the weight of the machine some kind of bump switch system could be implemented to stop the platform if an object is detected. Another improvement would be better integration of components. The platform controller PCB and the four motor driver PCBs could be integrated onto a single board. This would eliminate the need for a large amount of cabling on the platform and provide better communication between the different components. Also the inclination sensor could be changed to a surface mount design that could be mounted directly on the controller board.

8.1 Motor Driver Improvements

Because of the problems encountered with the motor drivers the back EMF voltage speed feedback had to be abandoned. However it could be reinstated with only slight modification to the existing design. With one more data line from the microcontroller's digital outputs the PLD on the driver board could be told to enter a mode that would allow a voltage reading to be taken. However an entirely different approach could be taken. If the drivers and controller PCBs are to be combined a more powerful microcontroller could be used to drive the H-bridge driver for each board directly. Alternatively if a more modular design is required a small microcontroller could be used for each motor driver. These microcontrollers could then communicate with the main controller through a digital communication protocol like I^2C (Inter-Integrated Circuit**)** or SPI (Serial Programmable Interface). This would allow on demand data relating to motor speed and current and would allow aspects like over-current protection and voltage speed feedback to be handled by the smaller motor driver microcontrollers, freeing up resources for more advanced features.

8.2 User Directional Input

Although the current arrangement of the joystick works well as an interface between the machine and operator there are more natural options available. The idea of having an interface that mimics the headboard of a hospital bed has been mooted. This involves having

a bar that, when a force is applied to it, causes the platform to move the bed in a manner that is intuitive to the force applied. A system like this could be achieved by utilising pressure sensing pads where the bar attaches to the platform and using this input to move the platform in a natural way. Although this system would offer intuitive control it will probably require two hands to operate unlike the single hand needed for the joystick.

8.3 Alternative Motors

The motors used in the current design are very large, using much of the platform's available space. Brushless alternative motors were found which are supplied by "Golden Motor". Brushless motors provide many advantages over brushed DC motors. The main advantage is improved service interval. Because the motors do not contain brushes and are essentially contactless, they have a much longer life cycle. They tend to have a larger power to weight ratio and power to size ratio due to the inbuilt gearing of the available hub design. The hub design also offers the advantage of being able to be housed in the hub of the wheel. This would free up space in the body of the platform, although a redesign of the Mecanum wheels would most probably result in little room inside the wheel due to efforts to smooth the motion of the platform [3]. However, these motors would still provide additional platform space even if mounted outside the wheel. If these motors were to be used in the next design iteration a complete redesign of the motor drivers would be necessary. Because brushless motors need a three phase power signal to be driven correctly these drivers would have added complexity [48]. Commercial options are available although they are not very flexible and hence not easy to integrate.

8.4 Safety Specifications

With this platform intended for workplace environments where it will be interacting directly with human operators or customers, safety is a critical factor that needs to be addressed. The environment specific safety standards need to be investigated and meet in future design iterations of the platform for it to be sold and used in a workplace environment. Some future safety features that could be implemented include: a mechanism for stopping the platform if a foreign object is detected underneath the shroud (e.g. operator's foot), collision warning/avoidance system, improved release mechanism for gripper, and better freewheeling capabilities in case of power failure. There are many other application specific safety systems that would need to be implemented to ensure safe operation in a range of environments.

8.5 Other Future Work

Other aspects that will need attention in the further development of the platform include the manufacturing process. If the product is to be mass produced, work needs to be done on optimising the design for this process. This includes modifying the frame so that the majority of the components can be pressed out of sheet steel. The shroud material and manufacturing process will need to be decided. Plastic would offer the best durability, cost and hygiene payoff. This could be either injection or rotation moulded. The wiring and fuse layout will also need adjusting. If tube steel is used for the mast construction many of the exposed wires can be hidden inside the tube steel providing a better attachment surface for a shroud. Having a single fuse box would also improve the manufacturability and maintainability of the machine. Finally, quality control is an important aspect for any product. To ensure that each bed moving platform produced is working correctly a testing rig could be designed. This would involve a dynamometer type system that can test the load that the drivers and motor can handle as well as testing gripper functionality.

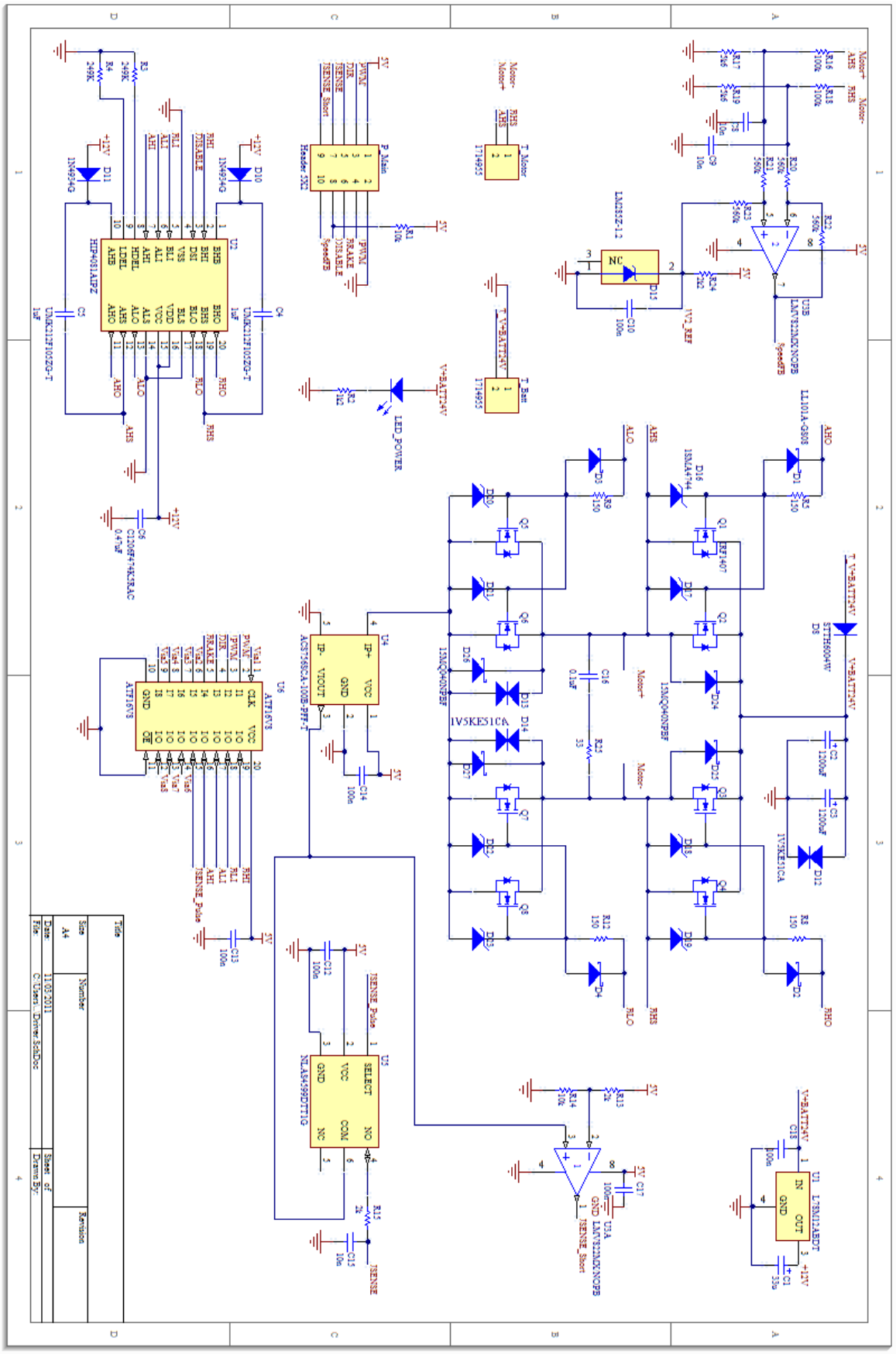
9 References

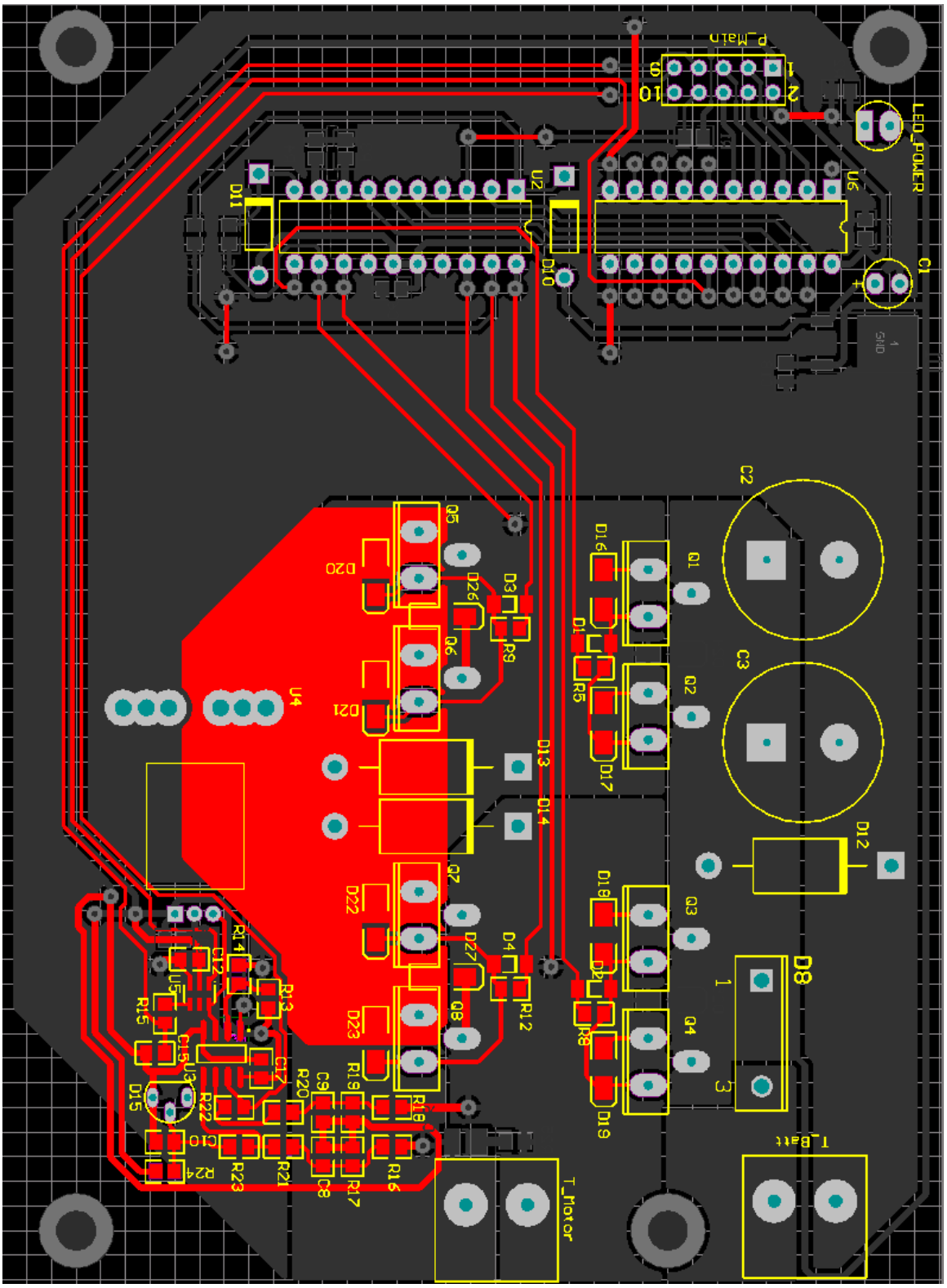
1. WorkCover Corporation South Austrlia, *Industry Brief 1999-00 to 2003-04 Hospitals (except Psychiatric hospitals)*, WorkCover Corporation South Austrlia, Editor. 2005: Adelaide.
2. Hoskins, A.B., *Occupational Injuries, Illnesses, and Fatalities among Nursing, Psychiatric, and Home Health Aides, 1995-2004*, United States Department of Labour, Editor. 2006.
3. Dickerson, S.L. and B.D. Lapin. *Control of an omni-directional robotic vehicle with Mecanum wheels*. in *Telesystems Conference, 1991. Proceedings. Vol.1., NTC '91., National*. 1991. p. 323-328.
4. Kang, J.W., et al., *Development of Omni-Directional Mobile Robots with Mecanum Wheels Assisting the Disabled in a Factory Environment*. 2008 International Conference on Control, Automation and Systems, Vols 1-4. 2008, New York: Ieee. 1775-1780.
5. Humphries Casters and Supplies. *Gzunda Hospital Bed Mover*. 2011 [cited 2011 26 January]; Available from: http://www.electrictuggers.com/more-details.html.
6. MasterMover Ltd. *MasterMover Bed Mover*. 2009 [cited 2011 26 January]; Available from: http://www.mastermover.com/products/bed-mover#.
7. U.M.S Pty Ltd. *P.T.S. Bed Mover*. 2010 [cited 2011 2 February]; Available from: http://ums.net.au/nu-star.html#gpm1_4.
8. Commercial Images. *Atlas PTS 4 Bed Mover PDF*. [cited 2011 2 February]; Available from: http://www.commercialimages.com.au/zone_files/PDF_Files/atlas_pts_4_bed_mover.pdf.
9. Braunl, T., *Embedded Robotics Third Edition*. 2008, Berlin, Germany: Springer-Verlag.
10. Grabowiecki, J., *Vehicle Wheel*. 1919, US Patent 1,303,535.
11. Ilon, B.E., *Wheels for a course stable self-propelling vehicle movable in any desired direction on the ground or some other base*. 1975, US Patent 3,876,255.
12. Byun, K.-S. and J.-B. Song, *Design and Construction of Continuous Alternate Wheels for an Omnidirectional Mobile Robot.* Journal of Robotic Systems, 2003. **9**(9): p. 569-579.
13. Diegel, O., et al. *Improved Mecanum Wheel Design for Omni-directional Robots*. in *2002 Australasian Conference on Robotics and Automation*. 2002. Auckland, New Zealand. p. 117-121.
14. Kyung-Lyong, H., et al. *Design and control of omni-directional mobile robot for Mobile Haptic Interface*. in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*. 2008. p. 1290-1295.
15. Woods, B., *Omni-Directional Wheelchair*, in *Faculty of Engineering, Computing and Mathematics*. 2006, The University of Western Australia: Perth.
16. Shimada, A., et al. *Mecanum-wheel vehicle systems based on position corrective control*. in *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*. 2005. p. 6 pp.
17. Kiddee, P. and A. Shimada. *A controller design on person following omni-directional vehicle robots*. in *SICE, 2007 Annual Conference*. 2007. p. 1043-1047.
18. Xu, P., *Mechatronics Design of a Mecanum Wheeled Mobile Robot*. Cutting Edge Robotics, ed. V. Kordic, A. Lazinica, and M. Merdan. 2005, Mammendorf, Germany: pIV pro literatur Verlag Robert Mayer-Scholz.

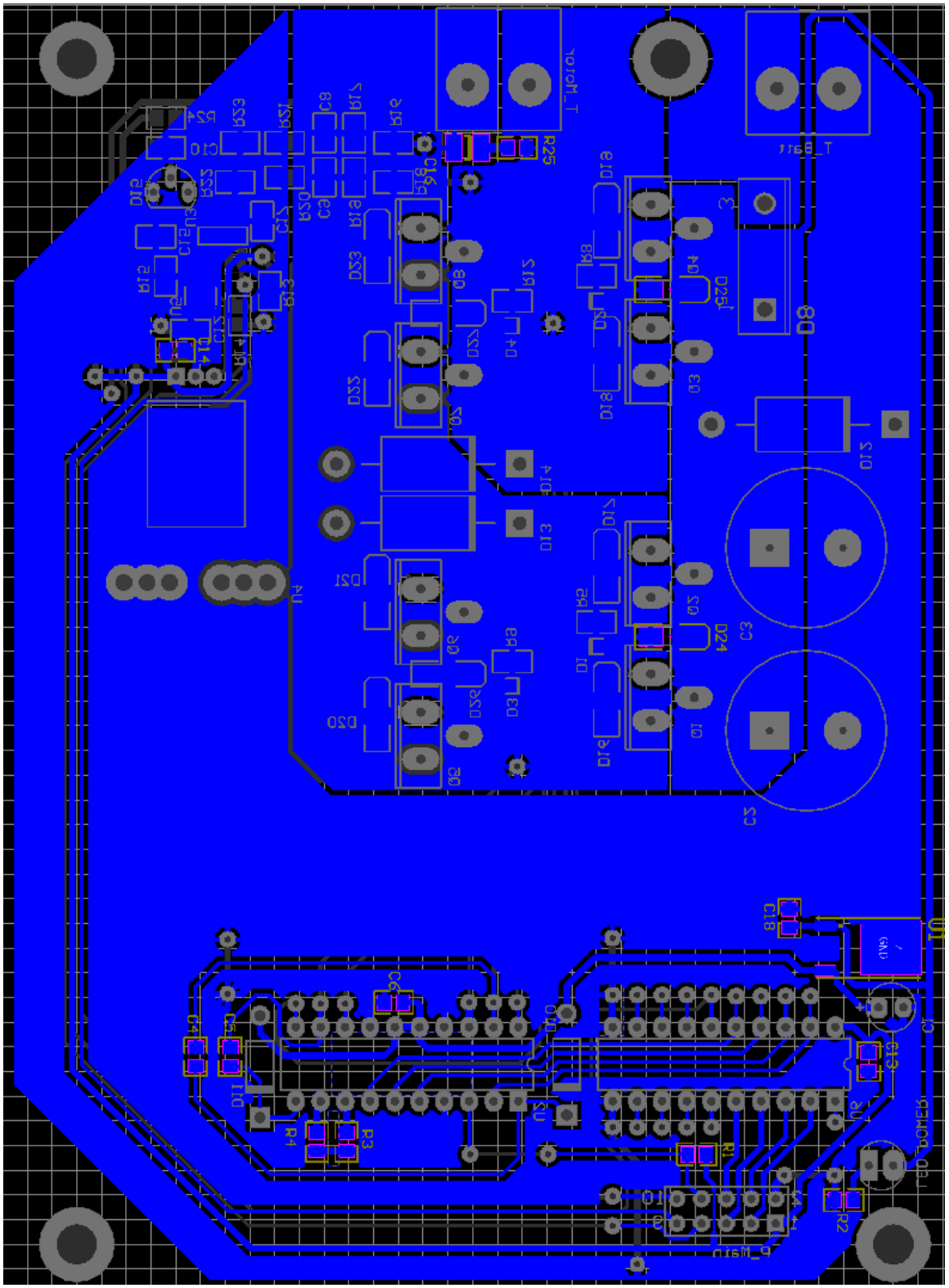
19. Xiao, C., et al. *An overview of integratable current sensor technologies*. in *Industry Applications Conference, 2003. 38th IAS Annual Meeting. Conference Record of the*. 2003. p. 1251-1258 vol.2.
20. Major, R.V. *Current measurement with magnetic sensors*. in *Magnetic Materials for Sensors and Actuators (Digest No. 1994/183), IEE Colloquium on*. 1994. p. 5/1-5/3.
21. Baker, A. and I. Mackenzie. *Omnidirectional Drive Systems Kinematics and Control*. in *FIRST Robotics Conference*. 2008.
22. Kyung-Lyong, H., K. Hyosin, and J.S. Lee. *The sources of position errors of omni-directional mobile robot with Mecanum wheel*. in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. 2010. p. 581-586.
23. Viboonchaicheep, P., A. Shimada, and Y. Kosaka. *Position rectification control for Mecanum wheeled omni-directional vehicles*. in *Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE*. 2003. p. 854-859 vol.1.
24. Kyung-Lyong, H., et al. *Design and control of mobile robot with Mecanum wheel*. in *ICCAS-SICE, 2009*. 2009. p. 2932-2937.
25. Xu, J. and X. Feng. *Design of adaptive fuzzy PID tuner using optimization method*. in *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*. 2004. p. 2454-2458 Vol.3.
26. Zhao, Z.Y., M. Tomizuka, and S. Isaka. *Fuzzy gain scheduling of PID controllers*. in *Control Applications, 1992., First IEEE Conference on*. 1992. p. 698-703 vol.2.
27. AndyMark. *Mecanum Wheel Spec Sheet PDF*. 2011 [cited 2011 7 February]; Available from: http://d1pytrrjwm20z9.cloudfront.net/MecanumWheelSpecSheet.pdf.
28. AndyMark Inc. *Mecanum Wheels, 10 inch*. 2011 [cited 2011 7 February]; Available from: http://www.andymark.com/ProductDetails.asp?ProductCode=am-0257.
29. Buchmann, I. *The cost of portable power*. 2011 [cited 2011 5 February]; Available from: http://batteryuniversity.com/learn/article/the_cost_of_portable_power.
30. Buchmann, I. *What's the best battery?* 2011 [cited 2011 5 February]; Available from: http://batteryuniversity.com/learn/article/whats_the_best_battery.
31. Reid Technology. *COMMANDER GEL Deep Cycle Sealed Batteries* 2011 [cited 2011 6 February]; Available from: http://www.reidtechnology.co.nz/shop/Solar+Power+%26+Alternative+Energy/Batteries/Deep+Cycle/Commander+Gel+Deep+Cycle.html.
32. Chew, M.T. and G.S. Gupta, *Embedded Programming with Field-Programmable Mixed-Signal µControllers - Second Edition (enhanced and updated)*. 2008, Austin, Texas, United States: Silicon Laboratories Inc.
33. Miyashita, K., T. Takahashi, and M. Yamanaka, *Features of a magnetic rotary encoder*. Magnetics, IEEE Transactions on, 1987. **23**(5): p. 2182-2184.
34. Dimension Engineering LLC. *Sabertooth dual 50A motor driver*. 2011 [cited 2011 8 February]; Available from: http://www.dimensionengineering.com/Sabertooth2x50HV.htm.
35. Demension Engineering, *Sabertooth 2x50 User's Guide*. 2009: http://www.dimensionengineering.com/datasheets/Sabertooth2x50HV.pdf.
36. Lloyd, S., et al., *Supporting documentation for the OSMC project*, C. Baron, Editor. 2002, http://www.robot-power.com/.
37. Intersil Co., *HIP4081A, 80V High Frequency H-Bridge Driver*. 2007.
38. Praesomboon, S., et al. *Sensorless speed control of DC servo motor using Kalman filter*. in *Information, Communications and Signal Processing, 2009. ICICS 2009. 7th International Conference on*. 2009. p. 1-5.

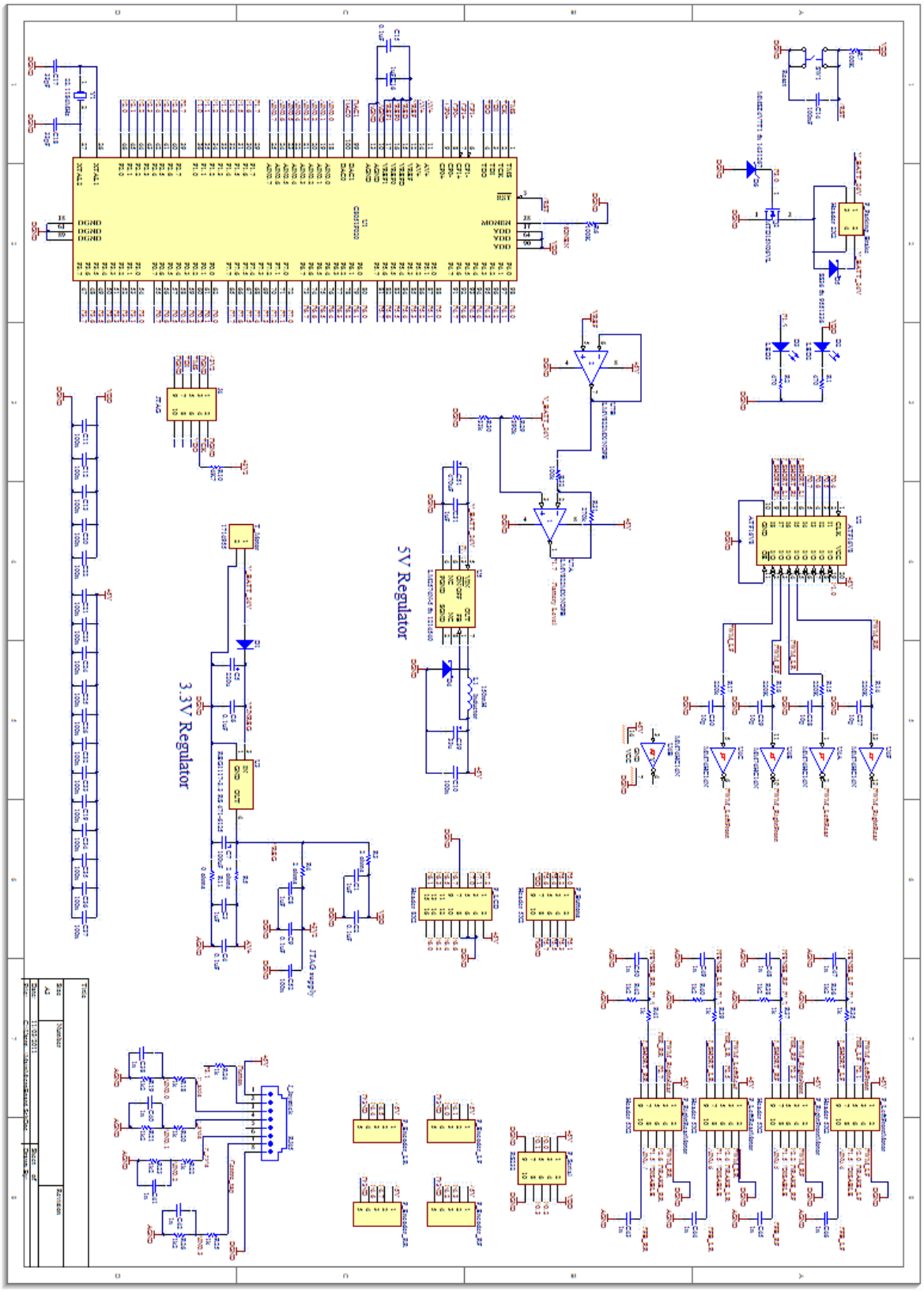
39. Qingbo, H., L. Zhengyu, and Q. Zhaoming. *Research on a Novel Close-loop Speed Control Technique of Brushless DC Motor*. in *Power Electronics Specialists Conference, 2007. PESC 2007. IEEE*. 2007. p. 2575-2578.
40. Hamad, S.H., et al. *Speed drive of single-phase induction motor*. in *Power and Energy Conference, 2004. PECon 2004. Proceedings. National*. 2004. p. 121-125.
41. Allegro MicroSystems Inc. *Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 3 kVRMS Voltage Isolation and a Low-Resistance Current Conductor*. 2010; Available from: http://www.allegromicro.com/en/Products/Part_Numbers/0756/0756.pdf.
42. MEMSIC Inc., *Ultra Low Noise, Low offset Drift ±1 g Dual Axis Accelerometer with Digital Outputs*. 2007: North Andover, USA.
43. Shanghai Zhichuan Tech, *ZCT245AL-232 2-axis Tilt Sensor*. 2007: Minhang, Shanghai, China.
44. Allied Motion Technologies Inc. *PLC Series Parallel-Shaft Gearmotors*. 2011 [cited 2011 14 February]; Available from: http://www.alliedmotion.com/Products/Series.aspx?s=14.
45. Silicon Laboratories, *C8051F020/1/2/3 Reference Document*. 2003, www.silabs.com: Austin, TX, USA.
46. Ming-Shyan, W., et al. *Novel interpolation method for quadrature encoder square signals*. in *Industrial Electronics, 2009. ISIE 2009. IEEE International Symposium on*. 2009. p. 333-338.
47. Kiam Heong, A., G. Chong, and L. Yun, *PID control system analysis, design, and technology*. Control Systems Technology, IEEE Transactions on, 2005. **13**(4): p. 559-576.
48. Rashid, M.H., *Power Electronics Handbook*. 2007, London, England: Elsevier Inc.

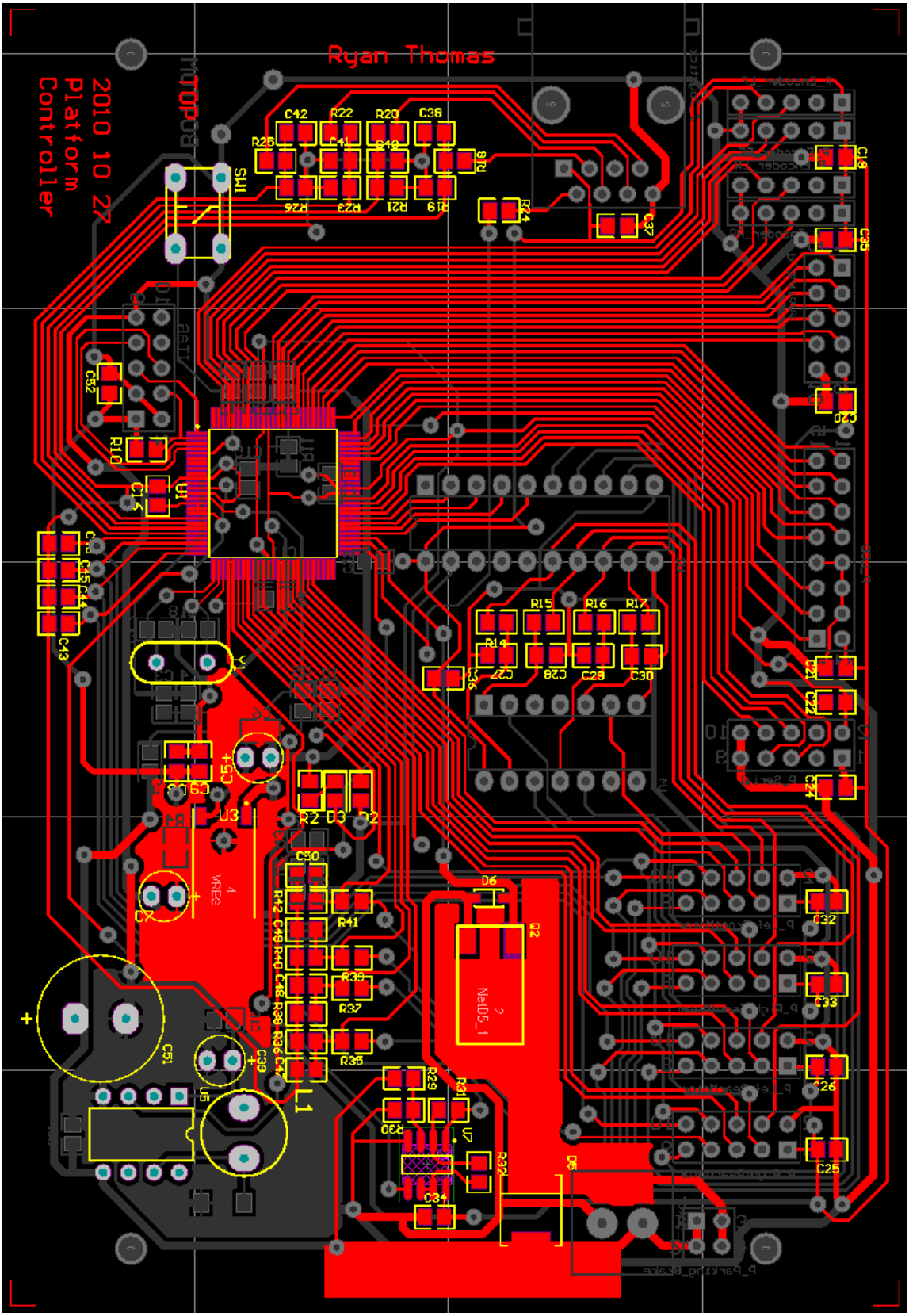
Appendix A

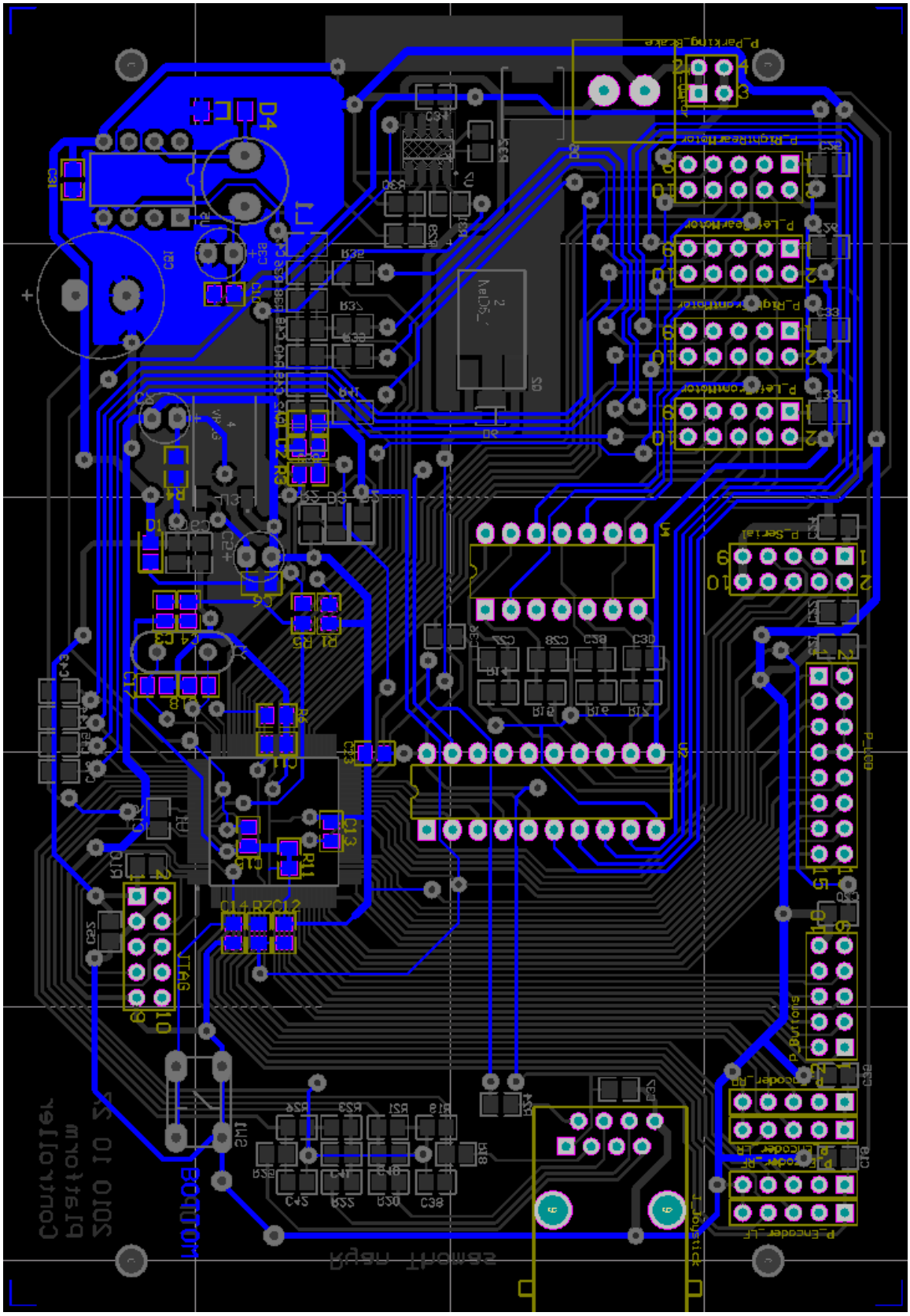


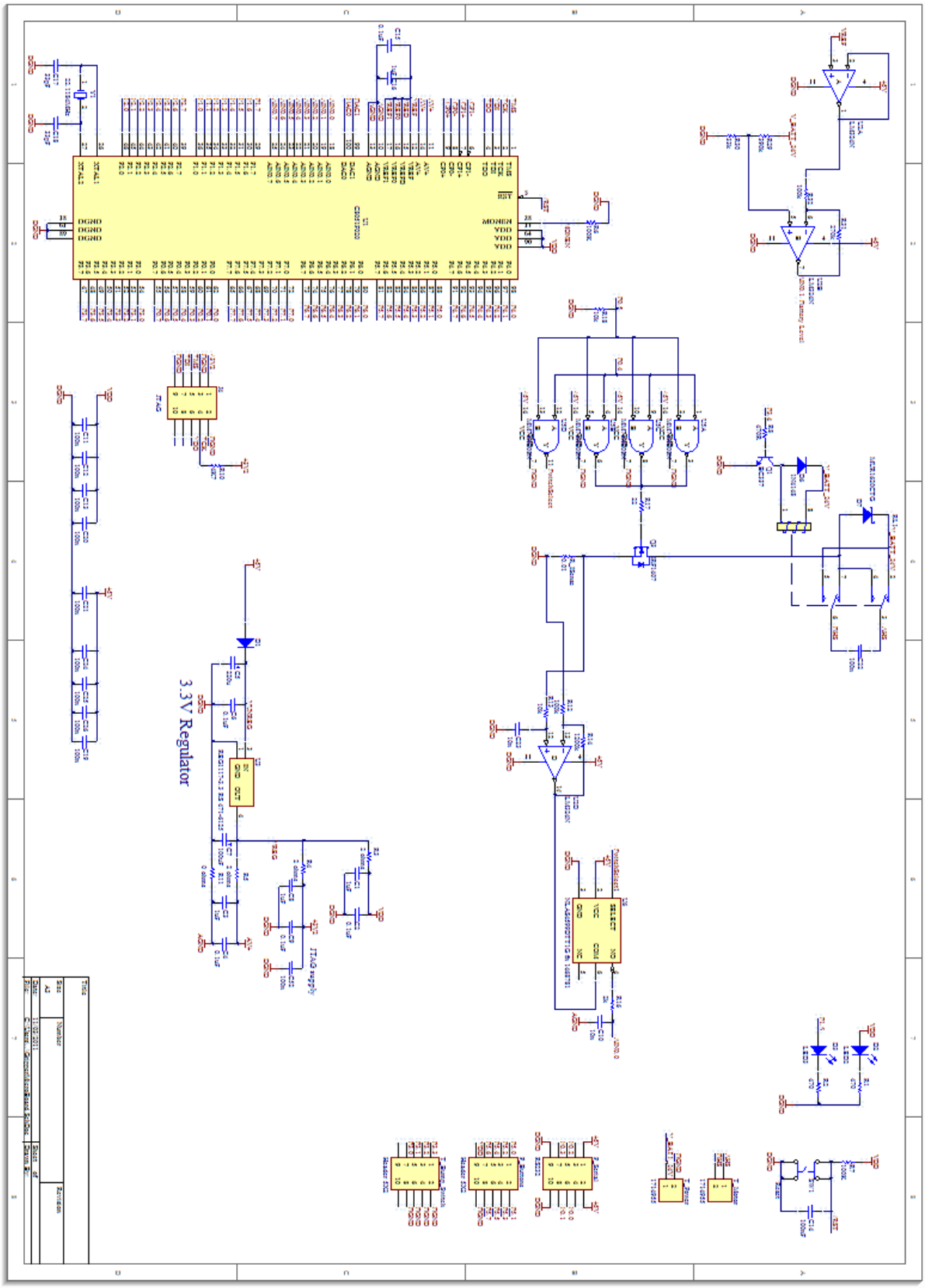

P_Main
LED_POWER
C1
C2
C3
D8
T_Batt
T_Motor
U2
U4
U5
U6
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
D1
D2
D3
D4
D10
D11
D12
D13
D14
D15
D16
D17
D18
D19
D20
D21
D22
D23
D26
D27
R5
R8
R9
R12
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
R23
C5
C8
C10
C12
C15
C17

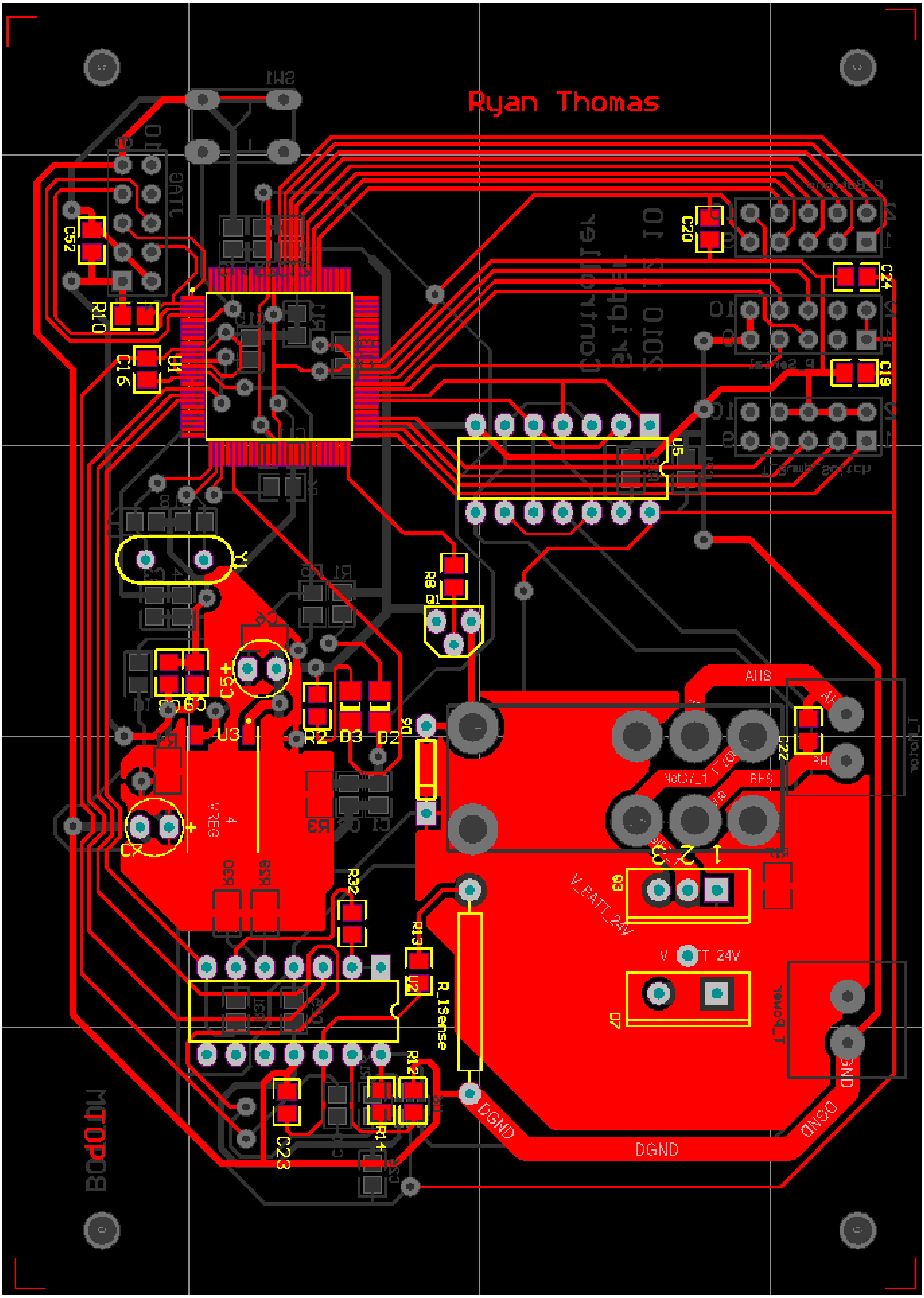


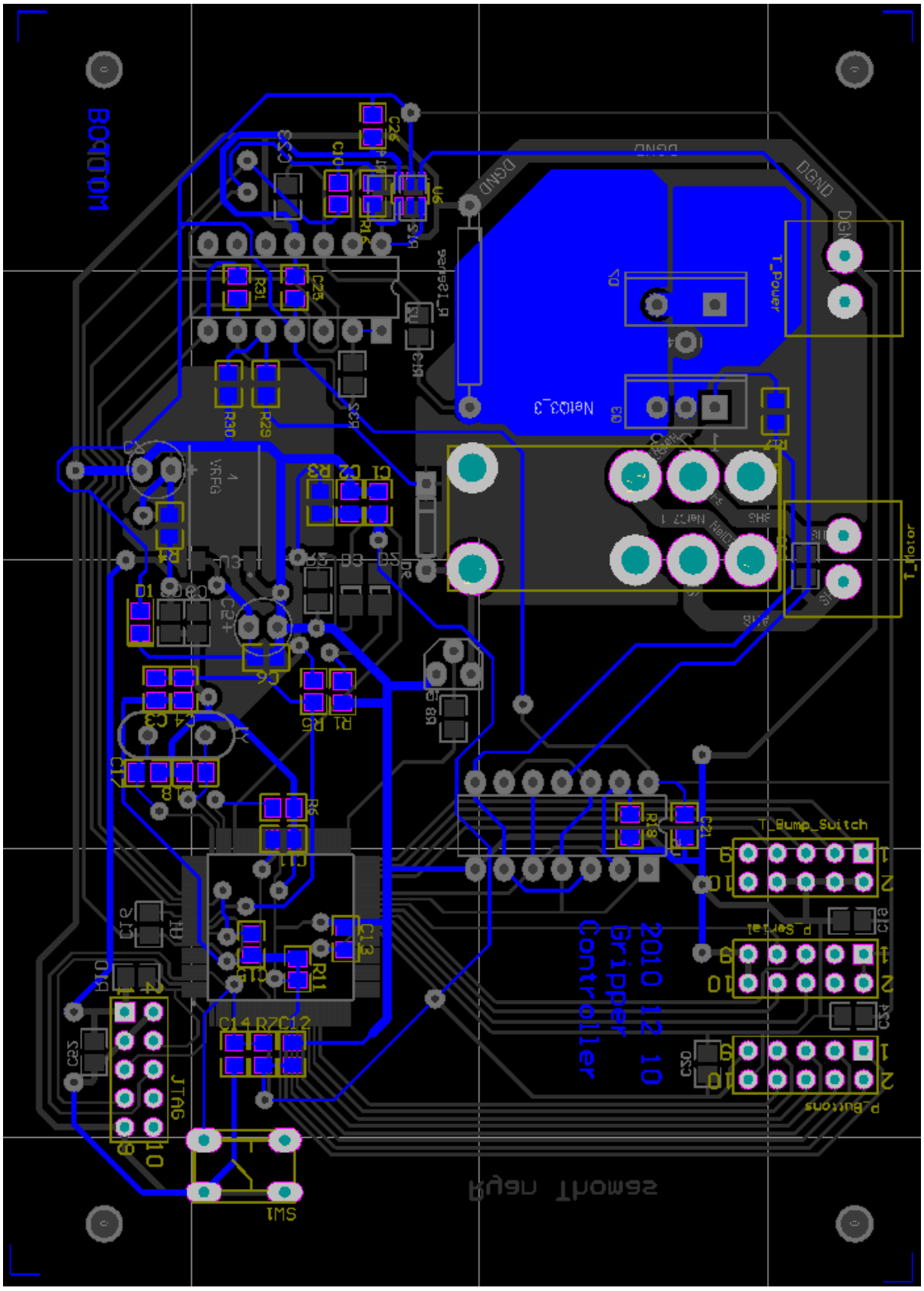

5V Regulator
3.3V Regulator


Ryan Thomas
2010 10 27
Platform
Controller
MyTOP03
C42
R22
R20
C38
R25
C41
C40
R24
R26
R23
R21
R19
SW1
R14
C37
C19
C35
C52
R10
C16
U1
C45
C44
C43
R15
R16
R17
C27
C28
C29
C30
C36
C21
C22
C24
Y1
C54
C53
U3
R2
D3
D2
C50
D6
C32
R41
R42
C49
R40
C48
R38
R36
C47
R37
R35
C39
C51
C33
C26
C25
L1
R29
R31
R30
U7
R32
C31
Q5
Q2
Net05_1




3.3V Regulator


Ryan Thomas
C20
C24
C19
C52
R10
C16
U1
U5
Y1
R8
C53
U3
R2
D3
D2
C22
R32
R13
U2
R_1Sense
R12
R14
C23
V_BATT_24V
D7
DGND
DGND
DGND


BOTTOM
DGND
T_Power
T_Motor
T_Bump_Switch
P_Serial
P_Buttons
JTAG
SW1
2010 12 10
Gripper
Controller

Appendix B

Software

Platform Microcontroller Code

```
//-----------------------------------------------------------------------------------------
// Platform - Bed Mover Project.c
//-----------------------------------------------------------------------------------------
// Copyright 2010 Massey University
//
// AUTH: Ryan Thomas
// DATE: 27 April, 2010
//
// This program does the following:
// Controls all aspects of an omni-directional platform, joystick, through to sending PWM
// signal. Also communicates with other Microcontroller for gripping mechanism.
//
// Clock:  22.11845 MHz external
// Target: C8051F020
//
// Tool chain: SDCC
//
//-----------------------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------------------
#include <c8051f020.h>                // SFR declarations
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>


//-----------------------------------------------------------------------------------------
// Global DEFINES
//-----------------------------------------------------------------------------------------
#define uchar unsigned char
#define uint  unsigned int

#define SYSCLK        22118450        // system clock frequency in Hz
#define LCD_DAT_PORT  P6              // LCD is in 8 bit mode
#define LCD_CTRL_PORT P7              // 3 control pins on P7
#define RS_MASK       0x01            // for assessing LCD_CTRL_PORT
#define RW_MASK       0x02
#define E_MASK        0x04
#define BUF_LEN 4

//-----------------------------------------------------------------------------------------
// Global MACROS
//-----------------------------------------------------------------------------------------
#define pulse_E();\
        small_delay(1);\
        LCD_CTRL_PORT = LCD_CTRL_PORT | E_MASK;\
        small_delay(1);\
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~E_MASK;\

//-----------------------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------------------
int Vtx, Vty, w, CT;

uchar          t_100us;
uchar          t_1ms;
uchar          t_10ms;
uchar          t_100ms;
uchar          t_1s;
uchar          t_1min;

bit                   display_update_flag;
bit                   ramp_flag;
bit                   calc_speed;
bit            Move_Right_flag;
```

```
bit             Move_Left_flag;
bit                     Button_flag;
bit                     finished;
bit                     display_state_change;
bit                     bed_attached;
bit                     Hill_mode;

uchar           start_up_timer;

uchar           Platform_state;
uchar           Gripper_state;
uchar           display_state;

char            battery_level;
idata int       Duty[4];
uchar           Direction;

idata int       posn[4];
idata int       old_posn[4];
char            last_halls0;
char            last_halls1;
char            last_halls2;
char            last_halls3;
char            index;

idata int       Encoder_speed[4];
idata uchar Motor_Current[4];

idata int       missed_edge_ctr;
idata int       negEncode;
idata int       requi_speed[4];
idata int       old_requi_speed[4];

idata int       Perror;
idata int       Ierror;
idata int       derror;
idata int       error;
idata int       last_error[4];

char received_byte1;
xdata char received_str1[16];                       //-- received string from PC
                                                    //-- 16 characters to display + NULL
character
xdata short index1;                                 //-- index into received_str

char received_byte0;
xdata char received_str0[16];                       //-- received string from PC
                                                    //-- 16 characters to display + NULL
character
xdata short index0;                                 //-- index into received_str

unsigned char x,y;
bit flag;
xdata float xResult;
xdata float yResult;
char xaxis[6];
char yaxis[6];

bit Tilt_flag;

uchar test;
char a;

idata int max_speed;
xdata int bedSpeedHigh;
xdata int bedSpeedLow;
xdata int SpeedHigh;
xdata int SpeedLow;

uchar buttonTimer;
bit buttonLock;


//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
```

```
void init           (void);                  // general system initialization
void lcd_init       (void);                  // initialize the lcd to 8 bit mode
void lcd_clear        (void);
void lcd_busy_wait  (void);                  // wait until the lcd is no longer busy
void putchar        (char c);         // replaces standard function and uses LCD
void lcd_cmd        (char cmd);              // write a command to the lcd controller
void lcd_goto       (uchar addr);     // move to address addr
void small_delay    (char d);         // 8 bit,  about 0.34us per count @22.1MHz
void large_delay    (char d);         // 16 bit, about 82us   per count @22.1MHz
void Timer0_Init    (void);                  // Timer3 will divide by counts
void Timer0_ISR     (void) interrupt 1;
void ADC0_Init      (void);
void ADC1_Init                (void);
void PCA0_Init        (void);
void UART0_Init               (void);
void UART0_ISR        (void) interrupt 4;
void UART1_Init               (void);
void UART1_ISR        (void) interrupt 20;
void Calc_Speed       (void);
void Check_Motor_Current              (void);
void JoyStick_to_MotorSpeed           (void);
void Update_Display   (void);
void Ramping_Duty     (void);
void Gripper_function                 (void);
bit     Dead_Band             (int Vtx,int Vty,int w);
bit     Platform_Stationary           (void);
void Send_PWM         (void);
void Start_Up         (void);

/* *INDENT-ON* */
//-------------------------------------------------------------------------------------
// Main Routine
//-------------------------------------------------------------------------------------
//
void main(void)
{
        init();
        PCA0_Init();
        P3_2 = 0;                            // configure the watchdog, crossbar, oscillator
        Timer0_Init();
        ADC0_Init();
        ADC1_Init();
        UART0_Init();
        lcd_init();
        EA = 1;                                      // Enable global interrupts

        //************** Varibles init **************//
        battery_level = 10;
        P2_0 = 0;
        P2_2 = 0;
        P2_4 = 0;
        P2_6 = 0;
        P3_0 = 1;
        Duty[0] = 0;
        Duty[1] = 0;
        Duty[2] = 0;
        Duty[3] = 0;
        Send_PWM();
        large_delay(255);
        large_delay(255);
        large_delay(255);
        large_delay(255);

        bedSpeedHigh = 200;
        bedSpeedLow = 100;
        SpeedHigh = 300;
        SpeedLow = 150;

        start_up_timer = 0;
        Gripper_state = 0;
        Start_Up();
        P1_5 = 0;
        display_state = 1;

        //************** Main Loop **************//
        while (1) {
```

```
                if (buttonLock == 0){
                        if (P3_1 && !Hill_mode) {
                                buttonTimer = 0;
                                buttonLock = 1;
                                if (Platform_Stationary() == 1) {
                                        Gripper_function();
                                }
                                else {
                                     if (max_speed == bedSpeedLow && Gripper_state == 3) {
                                              max_speed = bedSpeedHigh;
                                     }
                                     else if (max_speed == bedSpeedHigh && Gripper_state == 3) {
                                              max_speed = bedSpeedLow;
                                     }
                                     else if (max_speed == SpeedLow && Gripper_state != 3) {
                                              max_speed = SpeedHigh;
                                     }
                                     else if (max_speed == SpeedHigh && Gripper_state != 3) {
                                              max_speed = SpeedLow;
                                     }
                                }
                        }
                }
                if (buttonTimer > 50 && buttonLock == 1){
                        buttonLock = 0;
                }

                Update_Display();
                if (ramp_flag) {
                        ramp_flag = 0;
                        Check_Motor_Current();
                        JoyStick_to_MotorSpeed();
                        Calc_Speed();
                        Ramping_Duty();
                        Send_PWM();
                }
        }
}

//-------------------------------------------------------------------------------
// Start Up
//-------------------------------------------------------------------------------
//
// Start up communication with secondary micro-controller
//
//
void Start_Up (void)
{
        lcd_cmd(0x01);
        printf("Initialising...");
        large_delay(200);
        while(1){
                SBUF0 = 'S';           //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;                        //-- clear TI0

                SBUF0 = 0x0A;                   //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;                        //-- clear TI0

                if (Gripper_state != 0)// || start_up_timer > 100)
                        break;
        }
        lcd_cmd(0x01);

        if (Gripper_state == 0)
        {
                while(1)
                {
                        lcd_goto(0x00);
                        printf("Timeout Error");
                }
        }
        if (Gripper_state == 1){
                max_speed = SpeedLow;
        }
```

```
        if (Gripper_state == 3){
                max_speed = bedSpeedLow;
        }
        lcd_cmd(0x01);
}

//-------------------------------------------------------------------------------
// Update_Display
//-------------------------------------------------------------------------------
//
// Show appropriate display on LCD
//
//
void Update_Display    (void)
{

        if (display_update_flag){
                display_update_flag = 0;
                if (display_state_change){
                        display_state_change = 0;
                        lcd_cmd(0x01);
                }
                if (display_state == 1){
                        lcd_goto(0x00);                                 //-- First row of LCD
                        printf("LF:%4d", Duty[0]);
                        lcd_goto(0x09);
                        printf("RF:%4d", Duty[1]);
                        lcd_goto(0x40);                                 //-- Second Row of LCD
                        printf("LR:%4d", Duty[2]);
                        lcd_goto(0x49);
                        printf("RR:%4d", Duty[3]);
                }
                if (display_state == 2){
                        lcd_goto(0x00);                                 //-- First row of LCD
                        printf("Opening Gripper");
                }
                if (display_state == 3){
                        lcd_goto(0x00);                                 //-- First row of LCD
                        printf("Closeing Gripper");
                }
        }
}

//-------------------------------------------------------------------------------
// Check_Motor_Current
//-------------------------------------------------------------------------------
//
// Gives Current feedback
//
//
void Check_Motor_Current (void)
{
        AMX1SL = 0x01;
        small_delay(20);
        ADC1CN |= 0x10;//AD1BUSY = 1;
        while((ADC1CN & 0x20) ==0);
        Motor_Current[0] = ADC1;
        ADC1CN &= ~(0x20);

        small_delay(10);

        AMX1SL = 0x02;
        small_delay(20);
        ADC1CN |= 0x10;//AD1BUSY = 1;
        while((ADC1CN & 0x20) ==0);
        Motor_Current[1] = ADC1;
        ADC1CN &= ~(0x20);

        small_delay(10);

        AMX1SL = 0x03;
        small_delay(20);
        ADC1CN |= 0x10;//AD1BUSY = 1;
        while((ADC1CN & 0x20) ==0);
        Motor_Current[2] = ADC1;
        ADC1CN &= ~(0x20);
```

```
        small_delay(10);

        AMX1SL = 0x04;
        small_delay(20);
        ADC1CN |= 0x10;//AD1BUSY = 1;
        while((ADC1CN & 0x20) ==0);
        Motor_Current[3] = ADC1;
        ADC1CN &= ~(0x20);

        small_delay(10);
}

//-------------------------------------------------------------------------------
// Check_Drive_Motor_Speed
//-------------------------------------------------------------------------------
//
// Gives Speed feedback for PID control
//
//
void Calc_Speed (void)
{
        uchar i;
        for (i=0; i<4; i++){
                Encoder_speed[i] = posn[i] - old_posn[i];
                old_posn[i] = posn[i];
        }
}

//-------------------------------------------------------------------------------
// JoyStick_to_MotorSpeed
//-------------------------------------------------------------------------------
//
// Gets the joystick coordinants, normalises and then calculates each motor velocity
//
//
void JoyStick_to_MotorSpeed (void)
{
        long Duty_Calc[4];
        uchar i;

                AMX0SL = 0x03;
                small_delay(20);
                AD0BUSY = 1;
                while((ADC0CN & 0x20) ==0);
                CT = ADC0;
                ADC0CN &= ~(0x20);

                small_delay(10);

                AMX0SL = 0x00;
                small_delay(20);
                AD0BUSY = 1;
                while((ADC0CN & 0x20) ==0);
                Vtx = ADC0 - 2170;
                ADC0CN &= ~(0x20);

                small_delay(10);

                AMX0SL = 0x01;
                small_delay(20);
                AD0BUSY = 1;
                while((ADC0CN & 0x20) ==0);
                Vty = ADC0 - 2170;
                ADC0CN &= ~(0x20);

                small_delay(10);

                AMX0SL = 0x02;
                small_delay(20);
                AD0BUSY = 1;
                while((ADC0CN & 0x20) ==0);
                w = ((ADC0 - 2170) * -1);
                ADC0CN &= ~(0x20);

                w = w / 2;
```

```
        if (Hill_mode)
        {
                Vtx = 0;
                Vty = 0;
                w = 0;
        }

        //************** Duty calculations ***************//

        Duty_Calc[0] = (Vty + Vtx - w);// * 256 / 5323;             //Left Front
        Duty_Calc[0] = Duty_Calc[0] * 111 / 100;                   //gear ratio
        Duty_Calc[1] = (Vty - Vtx + w);// * 256 / 5323;             //Right Front
        Duty_Calc[1] = Duty_Calc[1] * 111 / 100;                   //gear ratio
        Duty_Calc[2] = (Vty - Vtx - w);// * 256 / 5323;             //Left Rear
        Duty_Calc[3] = (Vty + Vtx + w);// * 256 / 5323;             //Right Rear

        for (i=0; i<4; i++){
                Duty_Calc[i] = Duty_Calc[i] * max_speed / 5323;
                requi_speed[i] = Duty_Calc[i];
                if(Dead_Band(Vtx,Vty,w)){
                        if (requi_speed[i] < 10 && requi_speed[i] > -10){
                                requi_speed[i] = 0;
                        }
                }
        }
}

//-------------------------------------------------------------------------------
// Ramp_Duty
//-------------------------------------------------------------------------------
//
// Provides ramp for PWM duty cycle
//
//
void Ramping_Duty (void)
{

//*************** Ramp Function **************//
        uchar i;
//      P3_4 = 1;
        for (i=0; i<4; i++){
                if (old_requi_speed[i] >= 0){         // going forwards
                        if (requi_speed[i] > old_requi_speed[i] + 2){
                                requi_speed[i] = old_requi_speed[i] + 2;
                        }
                        if (requi_speed[i] < old_requi_speed[i] - 4){
                                requi_speed[i] = old_requi_speed[i] - 4;
                        }
                }

                if (old_requi_speed[i] < 0){          // going backwards
                        if (requi_speed[i] > old_requi_speed[i] + 4){
                                requi_speed[i] = old_requi_speed[i] + 4;
                        }
                        if (requi_speed[i] < old_requi_speed[i] - 2){
                                requi_speed[i] = old_requi_speed[i] - 2;
                        }
                }


                old_requi_speed[i] = requi_speed[i];

                error = (requi_speed[i] - Encoder_speed[i]);

                if (i < 2){
                        Perror = error / 2;

                        Ierror += error;
                        Ierror /=  8;
                        if (Ierror > 5){
                                Ierror = 5;
                        }
                        if (Ierror < -5){
                                Ierror = -5;
                        }
```

```
                        derror = error - last_error[i];
                        derror /= 5;

                        last_error[i] = error;
                        error = Perror + Ierror + derror;

                }
                else{
                        Perror = error / 2;

                        Ierror += error;
                        Ierror /=  8;
                        if (Ierror > 5){
                                Ierror = 5;
                        }
                        if (Ierror < -5){
                                Ierror = -5;
                        }

                        derror = error - last_error[i];
                        derror /= 8;

                        last_error[i] = error;
                        error = Perror + Ierror + derror;
                }

                Duty[i] += error;

                if (Duty[i] > 210){
                        Duty[i] = 210;
                }
                if (Duty[i] < -210){
                        Duty[i] = -210;
                }

                if (requi_speed[i] == 0 && Encoder_speed[i] == 0){
                        Duty[i] = 0;
                }
        }
}

//-------------------------------------------------------------------------
// Dead_Band
//-------------------------------------------------------------------------
//
// Set the deadband for the joystick
//
//
bit Dead_Band (int Vtx,int Vty,int w)
{
        int db;
        db = 350;

        if (Vtx > -db && Vtx < db && Vty > -db && Vty < db && w > -db && w < db) {
                if (Gripper_state == 1 || Gripper_state == 2)
                {
                        max_speed = SpeedLow;
                }
                if (Gripper_state == 3)
                {
                        max_speed = bedSpeedLow;
                }
                return 1;
        }
        else{
                return 0;
        }
}

//-------------------------------------------------------------------------
// Send_PWM
//-------------------------------------------------------------------------
//
// Sends the new duty cycle to the hardware PWM
//
```

```
//
void Send_PWM (void)
{

uchar i,j;
uchar Sent_PWM[4];

        j = 0;
        for (i=0x80; i>15; i=i>>1){
                if (Duty[j] < 0){
                        Sent_PWM[j] = Duty[j] * (-1);
                        Direction |= i;
                }
                else{
                        Sent_PWM[j] = Duty[j];
                        Direction &= ~i;
                }
                j++;
        }

//*****************Direction***************

        if (Direction & 0x80){
                P2_1 = 1;
        }
        else{
                P2_1 = 0;
        }
        if (Direction & 0x40){
                P2_3 = 1;
        }
        else{
                P2_3 = 0;
        }
        if (Direction & 0x20){
                P2_5 = 1;
        }
        else{
                P2_5 = 0;
        }
        if (Direction & 0x10){
                P2_7 = 1;
        }
        else{
                P2_7 = 0;
        }

//*************** Send to PCA **************

        if (Duty[0] == 0) {
                PCA0CPM0 &= ~(0x40);
        }
        else {
                PCA0CPM0 |= (0x40);
                PCA0CPH0 = 256 - Sent_PWM[0];
        }
        if (Duty[1] == 0) {
                PCA0CPM1 &= ~(0x40);
        }
        else {
                PCA0CPM1 |= (0x40);
                PCA0CPH1 = 256 - Sent_PWM[1];
        }
        if (Duty[2] == 0) {
                PCA0CPM2 &= ~(0x40);
        }
        else {
                PCA0CPM2 |= (0x40);
                PCA0CPH2 = 256 - Sent_PWM[2];
        }
        if (Duty[3] == 0) {
                PCA0CPM3 &= ~(0x40);
        }
        else {
                PCA0CPM3 |= (0x40);
                PCA0CPH3 = 256 - Sent_PWM[3];
```

```
        }


}

//-----------------------------------------------------------------------------------
// Gripper Function
//-----------------------------------------------------------------------------------
// Gripper interation
//
//
void Gripper_function(void)
{

        SBUF0 = 'G'; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;              //SCON0 &= ~0x42;                           //-- clear TI0

        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;              //SCON0 &= ~0x0A;                           //-- clear TI0

        Gripper_state = 0;
        while(Gripper_state == 0)
        {
                Update_Display();
        }
        display_state = 1;
        display_state_change = 1;
        if (Gripper_state == 1 || Gripper_state == 2)
        {
                max_speed = SpeedLow;
        }
        if (Gripper_state == 3)
        {
                max_speed = bedSpeedLow;
        }
        finished = 0;
}

//-----------------------------------------------------------------------------------
// Platform Stationary
//-----------------------------------------------------------------------------------
// is the platform stationary
//
//
bit Platform_Stationary(void)
{
        if (Duty[0] == 0 && Duty[1] == 0 && Duty[2] == 0 && Duty[3] == 0) {
                return 1;
        }
        else{
                return 0;
        }
}

//-----------------------------------------------------------------------------------
// init
//-----------------------------------------------------------------------------------
// general initialization
//
//
void init(void)
{
uchar n;
uint m = 0;
        WDTCN = 0x07;                 // Watchdog Timer Control Register
        WDTCN = 0xDE;                 // Disable watch dog timer
        WDTCN = 0xAD;

        OSCXCN = 0x67;                // select crystal oscillator, f>6.7MHz
        for (n = 0; n != 255; n++);          // wait 1ms for osc to start
        while ((OSCXCN & 0x80) == 0){ // wait for xtal to stabilize
                if(!++m)
                        break;        // abort if it won't start
        }
```

```
        OSCICN = 0x0C;                // switch to external oscillator

// Configure the XBRn Registers
        XBR0 = 0x24;
        XBR1 = 0x00;
        XBR2 = 0x44;                                    // Enable the crossbar

// Port configuration (1 = Push Pull Output)
        P1MDOUT = 0x60;               // Output configuration for P1
        P1MDIN &= ~0x9E;
        P1 |= 0xFF;

        P0MDOUT = 0xFF;               // Output configuration for P0 CEX0,1,2,3 push pull
        P0 &= ~0xF0;

        P2MDOUT = 0xFF;               // Output configuration for P2

        P3MDOUT = 0xFD;               // Output configuration for P3
        P3_2 = 1;

        P74OUT = 0x48;                // Output configuration for P4-7 (P7[0..3] push pull)
        P4 = 0xFF;
        P5 = 0x0F;                    // Turn the P5 LEDs off

}

//-------------------------------------------------------------------------------------
// PCA0_init
//-------------------------------------------------------------------------------------
//
void PCA0_Init(void)
{
        PCA0MD = 0x03;         // use SYSCLK/4 as time base
                                          // enable counter overflow interrupt
        PCA0CPM0 = 0x42;       // use PCA module 0 for 8 bit PWM generation
        PCA0CPM1 = 0x42;       // use PCA module 1 for 8 bit PWM generation
        PCA0CPM2 = 0x42;       // use PCA module 2 for 8 bit PWM generation
        PCA0CPM3 = 0x42;       // use PCA module 3 for 8 bit PWM generation
        PCA0L = 0;

        PCA0CPL0 = 0;
        PCA0CPH0 = 256;               // duty cycle
        PCA0CPL1 = 0;
        PCA0CPH1 = 256;               // duty cycle
        PCA0CPL2 = 0;
        PCA0CPH2 = 256;               // duty cycle
        PCA0CPL3 = 0;
        PCA0CPH3 = 256;               // duty cycle
        PCA0CN = 0x40;                // enable PCA0 counter

        PCA0CPM0 &= ~(0x40);
        PCA0CPM1 &= ~(0x40);
        PCA0CPM2 &= ~(0x40);
        PCA0CPM3 &= ~(0x40);
}


//-------------------------------------------------------------------------------------
// lcd_init
//-------------------------------------------------------------------------------------
//
void lcd_init(void)
{
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;    // RS = 0
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;    // RW = 0
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~E_MASK;     //  E = 0
        large_delay(200);                     // 16ms delay
        LCD_DAT_PORT = 0x38;          // set 8-bit mode
        pulse_E();
        large_delay(50);                      // 4.1ms delay
        LCD_DAT_PORT = 0x38;          // set 8-bit mode
        pulse_E();
        large_delay(2);                                 // 1.5ms delay
        LCD_DAT_PORT = 0x38;          // set 8-bit mode
        pulse_E();
```

```
        large_delay(2);                             // 1.5ms delay
        lcd_cmd(0x06);                          // curser moves right
        lcd_cmd(0x01);                          // clear display
        lcd_cmd(0x0E);                          // display and curser on
}

//-------------------------------------------------------------------------------------
// lcd_busy_wait
//-------------------------------------------------------------------------------------
//
// wait for the busy bit to drop
//
void lcd_busy_wait(void)
{
        LCD_DAT_PORT = 0xFF;
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;     // RS = 0
        LCD_CTRL_PORT = LCD_CTRL_PORT | RW_MASK;      // RW = 1
        small_delay(1);
        LCD_CTRL_PORT = LCD_CTRL_PORT | E_MASK;       //  E = 1
        do {                                          // wait for busy flag to drop
                small_delay(1);
        } while ((LCD_DAT_PORT & 0x80) != 0);
}

//-------------------------------------------------------------------------------------
// lcd_dat (putchar)
//-------------------------------------------------------------------------------------
//
// write a character to the lcd screen
//
void putchar(char dat)
{
        lcd_busy_wait();
        LCD_CTRL_PORT = LCD_CTRL_PORT | RS_MASK;      // RS = 1
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;     // RW = 0
        LCD_DAT_PORT = dat;
        pulse_E();
}

//-------------------------------------------------------------------------------------
// lcd_cmd
//-------------------------------------------------------------------------------------
//
// write a command to the lcd controller
//
void lcd_cmd(char cmd)
{
        lcd_busy_wait();
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RS_MASK;     // RS = 0
        LCD_CTRL_PORT = LCD_CTRL_PORT & ~RW_MASK;     // RW = 0
        LCD_DAT_PORT = cmd;
        pulse_E();

}

//-------------------------------------------------------------------------------------
// lcd_goto
//-------------------------------------------------------------------------------------
//
// change the text entry point
//
void lcd_goto(uchar addr)
{
        lcd_cmd(addr | 0x80);
}

//-------------------------------------------------------------------------------------
// delay routines
//-------------------------------------------------------------------------------------
//
// delay using spin wait
//
void small_delay(char d)
{
//      a = d;
        while (d--);
```

```
}

void large_delay(char d)
{
        while (d--)
                small_delay(255);
}

//-----------------------------------------------------------------------------------------
// Interrupt Service Initialization and Routines
//-----------------------------------------------------------------------------------------

//-----------------------------------------------------------------------------------------
// Timer0_Init
//-----------------------------------------------------------------------------------------
// Configure Timer0 to auto-reload and generate an interrupt at interval
// specified by <Freq> using SYSCLK/12 as its time base.
//
void Timer0_Init(void)
{
        int     Freq;
        Freq = 10000;
        //-- set up Timer 0
        CKCON &= ~0x08;        //-- T0M=0; Timer 0 uses the system clock / 12
        TMOD |= 0x02;  //-- Timer 0 in Mode 2 (8-bit auto-reload)
        TH0 = -(SYSCLK / 12 / Freq);
        IE |= 0x02;
        TR0 = 1;               //-- start Timer 0 (TCON.4 = 1)
}

//-----------------------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------------------
// - reset timer 0 overflow flag, which causes the interrupt
// - update 100us, 10ms and 1s counters
void Timer0_ISR(void) interrupt 1
{
//      TMR3CN &= ~(0x80);                    // clear TF3

/*  read the hall effect sensors and work out whether to inc or dec position
    build an offset by moving 4 bits into index.
    NOTE1: This hard codes the address of the hall sensors inside this routine to
           pins 3.3 and 3.4. (move to 3.2 and 3.3 sometime?)
    NOTE2: The lookup table used will return 2 if an edge is missed (both bits
           will have changed) This can also happen at startup and must be detected.  */

   static code    char hall_tab[] = {0, 1, -1, 2, -1, 0, 2, 1, 1, 2, 0, -1, 2, -1, 1, 0};
   char edge;
    index = (P4 & 0x03)<<2;  /* copy of P4 in case it changes while this routine executes */
    index |= last_halls0;        // merge the 4 bits
    edge = hall_tab[index];
    posn[0] += edge;
    last_halls0 = index >>2;          // use >>2 instead

//    if(edge == 2) {            // have skipped an edge
//        missed_edge_ctr++;
//        if(dir == DIR_DN)
//            posn-=4;
//    }

    index = (P4 & 0x0C);    /*  copy of P4 in case it changes while this routine executes */
    index |= last_halls1;        // merge the 4 bits
    edge = hall_tab[index];
    posn[1] += edge;
    last_halls1 = index >>2;          // use >>2 instead

//    if(edge == 2) {            // have skipped an edge
//        missed_edge_ctr++;
//        if(dir == DIR_DN)
//            posn-=4;
//    }

    index = (P4 & 0x30)>>2;    /* copy of P4 in case it changes while this routine executes */
    index |= last_halls2;        // merge the 4 bits
    edge = hall_tab[index];
    posn[2] += edge;
```

```
    last_halls2 = index >>2;          // use >>2 instead

//    if(edge == 2) {            // have skipped an edge
//        missed_edge_ctr++;
//        if(dir == DIR_DN)
//            posn-=4;
//    }

       index = (P4);
    index = (P4 & 0xC0)>>4;    /* copy of P4 in case it changes while this routine executes */
    index |= last_halls3;        // merge the 4 bits
    edge = hall_tab[index];
    posn[3] += edge;
    last_halls3 = index >>2;          // use >>2 instead

//    if(edge == 2) {            // have skipped an edge
//        missed_edge_ctr++;
//        if(dir == DIR_DN)
//            posn-=4;
//    }

       if (++t_100us > 9){
              t_100us = 0;
              if(++t_1ms > 9){
                     t_1ms = 0;
                     if(t_10ms == 4 || t_10ms == 9){
                            ramp_flag = 1;
                     }
                     buttonTimer++;
                     if(++t_10ms > 9) {
                            t_10ms = 0;
                            calc_speed = 1;
                            display_update_flag = 1;
                            start_up_timer++;
                            if(++t_100ms > 9) {
                                   t_100ms = 0;
                                   if(++t_1s > 60) {
                                          t_1s = 0;
                                          ++t_1min;
                                   }
                            }
                     }
              }
       }
}


//-------------------------------------------------------------------------
// ADC0_Init
//-------------------------------------------------------------------------
// Initialise ADCs
//
//
//
//
void ADC0_Init(void)
{

       REF0CN = 0x07;     //-- Enable internal bias generator and internal reference buffer
                                   //   Select ADC0 reference from VREF0 pin
       ADC0CF = 0x80;        //-- SAR0 conversion clock=1.3MHz approx., Gain=1
       AMX0CF = 0x00;        //-- 8 single-ended inputs
       AMX0SL = 0x01;        //-- Select AIN0.1
       ADC0CN = 0x80;        //-- enable ADC0, Continuous Tracking Mode
                                   //   Conversion initiated on bit write, ADC0 data
is right justified
}

void ADC1_Init (void)
{

       REF0CN |= 0x03;     //-- Enable internal bias generator and internal reference buffer
                                   //   Select ADC0 reference from VREF0 pin
       ADC1CF = 0x80;        //-- SAR0 conversion clock=1.3MHz approx., Gain=1
       AMX1SL = 0x01;        //-- Select AIN1.1
       ADC1CN = 0x80;        //-- enable ADC0, Continuous Tracking Mode
```

```
                                        //   Conversion initiated on bit write, ADC0 data
is right justified
}

//-------------------------------------------------------------------------
// UART0_Init
//-------------------------------------------------------------------------
//
// Initialise UART0
//
//
//
void UART0_Init(void)
{
        CKCON |= 0x10; //-- T1M=1; Timer 1 uses the system clock 22.11845 MHz
        TMOD |= 0x20;  //-- Timer 1 in Mode 2 (8-bit auto-reload)
        TH1 = 0x70;            //-- Baudrate = 9600
        TR1 = 1;               //-- start Timer 1 (TCON.6 = 1)


        //-- Set up the UART0
        PCON |= 0x80;  //-- SMOD1=1 (UART0 baud rate divide-by-2 disabled)
        SCON0 = 0x50;  //-- UART0 Mode 1, Logic level of stop bit ignored
                                  //   and Receive not enabled

        //-- enable UART0 interrupt
        IE |= 0x10;

        RI0 = 0;       //-- clear the receive interrupt flag; ready to receive more
        TI0 = 0;       //-- TX1 ready to transmit
}

//-------------------------------------------------------------------------
// UART0_ISR
//-------------------------------------------------------------------------
//
// UART0 Interrupt
//
//
//
void UART0_ISR(void) interrupt 4
{
        //-- pending flags RI0 (SCON1.0) and TI1(SCON1.1)

        if (RI0 == 1)  //-- interrupt caused by received byte
        {
                received_byte0 = SBUF0;       //-- read the input buffer
                RI0 = 0;                          //-- clear the interrupt flag
                received_str0[index0] = received_byte0;
                index0++;
                if (index0 > 15) index0 = 0;  //-- wrap around the array

                if (received_byte0 == 0x0a)           //-- check for NULL character
                {
                        index0 = 0;

                        if (received_str0[0] == 'R'){
                                Move_Right_flag = 1;
                        }

                        if (received_str0[0] == 'L'){
                                Move_Left_flag = 1;
                        }

                        if (received_str0[0] == 'O'){
                                display_state_change = 1;
                                display_state = 2;
                        }

                        if (received_str0[0] == 'C'){
                                display_state_change = 1;
                                display_state = 3;
                        }

                        if (received_str0[0] == 'F'){
                                finished = 1;
```

```
                        }

                        if (received_str0[0] == '1'){
                                Gripper_state = 1;
                        }

                        if (received_str0[0] == '2'){
                                Gripper_state = 2;
                        }

                        if (received_str0[0] == '3'){
                                Gripper_state = 3;
                        }
                        if (received_str0[0] == 'H'){
                                Hill_mode = 1;
                        }
                        if (received_str0[0] == 'N'){
                                Hill_mode = 0;
                        }
                }
        }
        if (TI0 == 1)  //-- interrupt caused by transmitted byte
        {
        }
}
```

Gripper Microcontroller Code

```
//-----------------------------------------------------------------------------------------
// Gripper - Bed Mover Project.c
//-----------------------------------------------------------------------------------------
// Copyright 2010 Massey University
//
// AUTH: Ryan Thomas
// DATE: 27 April, 2010
//
// This program does the following:
// Controlls the functioanlity of the gripper add-on for bed moving Platform
//
//
// Clock:  22.11845 MHz external
// Target: C8051F020

//
// Tool chain: SDCC
//
//-----------------------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------------------
#include <c8051f020.h>                // SFR declarations
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

//-----------------------------------------------------------------------------------------
// Global DEFINES
//-----------------------------------------------------------------------------------------
#define uchar unsigned char
#define uint  unsigned int

#define SYSCLK        22118450        // system clock frequency in Hz

//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------

uchar          t_100us;
uchar          t_1ms;
uchar          t_10ms;
uchar          t_100ms;
uchar          t_1s;
```

```
uchar          t_1min;

bit                   ramp_flag;
bit                   button_flag;
bit                   finished;
bit                   Hill_mode;
bit                   Hill_mode2;
bit                   Gripper_error;

char           battery_level;
idata uchar    Duty;
idata uchar    old_Duty;
bit                   Direction;

idata unsigned int Motor_Current[10];
idata unsigned int Motor_Current2;
idata unsigned int Current_limit;
uchar current_trip;
uchar startup_timer;
bit startup_lock;

char received_byte1;
xdata char received_str1[16];                       //-- received string from PC
                                                    //-- 16 characters to display + NULL
character
xdata short index1;                                        //-- index into received_str

char received_byte0;
xdata char received_str0[16];                       //-- received string from PC
                                                    //-- 16 characters to display + NULL
character
xdata short index0;                                        //-- index into received_str

unsigned char x,y;
bit flag;
xdata float xResult;
xdata float yResult;
char xaxis[6];
char yaxis[6];
bit start;

bit Tilt_flag;

uchar test;
char a;

uchar Gripper_state;

//-------------------------------------------------------------------------------------
// Function PROTOTYPES
//-------------------------------------------------------------------------------------
void init           (void);                 // general system initialization
void small_delay    (char d);         // 8 bit,  about 0.34us per count @22.1MHz
void large_delay    (char d);         // 16 bit, about 82us   per count @22.1MHz
void Timer0_Init    (void);                 // Timer3 will divide by counts
void Timer0_ISR     (void) interrupt 1;
void ADC0_Init      (void);
void PCA0_Init        (void);
void UART1_Init               (void);
void UART1_ISR        (void) interrupt 20;
void UART0_Init               (void);
void UART0_ISR        (void) interrupt 4;
void Start_Up         (void);
void Check_Motor_Current              (void);
void Send_Gripper_State               (void);
void Check_Tilt               (void);
void Open_Gripper     (void);
void Close_Gripper    (void);
void Move_Left        (void);
void Move_Right               (void);
void Ramping_Duty     (void);
void Send_PWM         (void);

/* *INDENT-ON* */
//-------------------------------------------------------------------------------------
// Main Routine
```

```
//-----------------------------------------------------------------------------
//
void main(void)
{
        init();                                                         // configure the
watchdog, crossbar, oscillator
        PCA0_Init();
        Duty = 0;
        Send_PWM();
        UART0_Init();
        UART1_Init();
        Timer0_Init();
        ADC0_Init();

        EA = 1;                                                 // Enable global
interrupts

        //************** Varibles init **************//
        battery_level = 10;
        button_flag = 0;
        Current_limit = 450;   // was 340
//      P3_0 = 0;

//      P3_0 = 0;
//      posn = 0;

        //************** Main Loop **************//
        Start_Up();
        while (1) {
                Check_Tilt();
                if (button_flag == 1){
                        button_flag = 0;
                        if (Gripper_state == 1 || Gripper_state == 3) {
                                Open_Gripper();
                                Send_Gripper_State();
                        }
                        else if (Gripper_state == 2) {
                                Close_Gripper();
                                Send_Gripper_State();
                        }
                }
        }
}

//-----------------------------------------------------------------------------
// Start Up
//-----------------------------------------------------------------------------
//
// Commuicates start up state with other micro
//
//
void Start_Up (void)
{
        start = 1;
        while (!finished)
        {
                small_delay(1);
        }

        large_delay(25);

        if (P3_0 == 0){
                Gripper_state = 1;

                Send_Gripper_State();
        }
        else
        {
                large_delay(200);
                Close_Gripper();
                Send_Gripper_State();
        }
        start = 0;
}

//-----------------------------------------------------------------------------
```

```
// Send Gripper State
//-------------------------------------------------------------------------
//
// Sends the gripper state
//
//
void Send_Gripper_State (void)
{
        if (Gripper_state == 1){
                SBUF0 = '1'; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x42;
        //-- clear TI0

                SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x0A;
        }

        if (Gripper_state == 2){
                SBUF0 = '2'; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x42;
        //-- clear TI0

                SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x0A;
        }

        if (Gripper_state == 3){
                SBUF0 = '3'; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x42;
        //-- clear TI0

                SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x0A;
        }
}

//-------------------------------------------------------------------------
// Check_Motor_Current
//-------------------------------------------------------------------------
//
// Gives Current feedback
//
//
void Check_Motor_Current (void)
{

        char i;
        for (i=9; i>0; i--){
                Motor_Current[i] = Motor_Current[i - 1];
        }
        small_delay(20);

        AD0BUSY = 1;
        while((ADC0CN & 0x20) == 0);
        Motor_Current[0] = ADC0;
        Motor_Current2 = ADC0;
        ADC0CN &= ~(0x20);

}



//-------------------------------------------------------------------------
// Check_Tilt
//-------------------------------------------------------------------------
//
// Gives Tilt sensor feedback
//
//
void Check_Tilt (void)
```

```
{
        if (Tilt_flag == 1){
                Tilt_flag = 0;
                SBUF1 = 0x2a; //-- load the serial buffer with the char to send
                while ((~SCON1 & 0x02) == 0x02);   //-- wait while the transmission is going on
                SCON1 &= ~0x02;                                                 //-- clear TI0


                SBUF1 = 0x50; //-- load the serial buffer with the char to send
                while ((~SCON1 & 0x02) == 0x02);   //-- wait while the transmission is going on
                SCON1 &= ~0x02;                                                 //-- clear TI0
        }
        if (Hill_mode != Hill_mode2)
        {
                Hill_mode2 = Hill_mode;
                if (Hill_mode)
                {
                        SBUF0 = 'H'; //-- load the serial buffer with the char to send
                        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is
going on
                        TI0 = 0;              //SCON0 &= ~0x42;
                //-- clear TI0

                        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is
going on
                        TI0 = 0;              //SCON0 &= ~0x0A;
                }

                if (!Hill_mode)
                {
                        SBUF0 = 'N'; //-- load the serial buffer with the char to send
                        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is
going on
                        TI0 = 0;              //SCON0 &= ~0x42;
                //-- clear TI0

                        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is
going on
                        TI0 = 0;              //SCON0 &= ~0x0A;
                }
        }
}


//-------------------------------------------------------------------------
// Open Gripper
//-------------------------------------------------------------------------
//
// Opens Gripper
//
//
void Open_Gripper (void)
{
        uchar c;
        uchar i;
        SBUF0 = 'O'; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;              //SCON0 &= ~0x42;                                       //--
clear TI0

        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;              //SCON0 &= ~0x0A;//-- clear TI0
        startup_timer = 0;
        startup_lock = 0;
        Gripper_state = 0;
        for (i=0; i<10; i++){
                Motor_Current[i] = 0;
        }
        while (Gripper_state == 0) {
                if (ramp_flag == 1){
                        ramp_flag = 0;
                        if (Duty < 90) {
                                Duty = Duty + 3;
```

```
                        }
                        else
                        {
                                Duty = 91;
                        }
                        Send_PWM();
                }
                Check_Motor_Current();
                if (P3_1 == 0){
                        Duty = 0;
                        Send_PWM();
                        Gripper_state = 2;
                }
                if (startup_timer > 20 || startup_lock == 1)
                {
                        startup_lock = 1;
                        c = 0;
                        for (i=0; i<9; i++){
                                if (Motor_Current [i] > Current_limit){
                                        c++;
                                }
                        }
                        if (c == 10){
                                Duty = 0;
                                Send_PWM();
                                P1_6 = 1;
                                Gripper_error = 1;
                                while(1);
                        }
                }
        }
}

//-------------------------------------------------------------------------
// Close Gripper
//-------------------------------------------------------------------------
//
// Closes Gripper
//
//
void Close_Gripper (void)
{
        uchar i;
        uchar c;
        P3_4 = 0;
        if (start == 0)
        {
                SBUF0 = 'C'; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x42;
        //-- clear TI0

                SBUF0 = 0x0A; //-- load the serial buffer with the char to send
                while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
                TI0 = 0;               //SCON0 &= ~0x0A;
        //-- clear TI0
        }
        startup_timer = 0;
        startup_lock = 0;
        Gripper_state = 0;
        for (i=0; i<10; i++){
                Motor_Current[i] = 0;
        }
        while (Gripper_state == 0) {
                if (ramp_flag == 1){
                        ramp_flag = 0;
                        if (Duty < 90) {
                                Duty = Duty + 3;
                        }
                        else
                        {
                                Duty = 91;
                        }
                        Send_PWM();
                }
                Check_Motor_Current();
```

```
                if (P3_0 == 0){
                        Duty = 0;
                        Send_PWM();
                        Gripper_state = 1;
                }
                if (startup_timer > 20 || startup_lock == 1)
                {
                        startup_lock = 1;
                        c = 0;
                        for (i=0; i<9; i++){
                                if (Motor_Current [i] > Current_limit){
                                        c++;
                                }
                        }
                        if (c == 10){
                                Duty = 0;
                                Send_PWM();
                                P1_6 = 1;
                                Gripper_state = 3;
                        }
                }
        }
}

//-------------------------------------------------------------------------
// Move Left
//-------------------------------------------------------------------------
//
// Moves Platform to the left
//
//
void Move_Left         (void)
{
        SBUF0 = 'L'; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;//SCON0 &= ~0x02;                                          //-- clear TI0

        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;//SCON0 &= ~0x02;                                          //-- clear TI0
}

//-------------------------------------------------------------------------
// Move Right
//-------------------------------------------------------------------------
//
// Moves Platform to the right
//
//
void Move_Right               (void)
{
        SBUF0 = 'R'; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;//SCON0 &= ~0x02;                                          //-- clear TI0

        SBUF0 = 0x0A; //-- load the serial buffer with the char to send
        while ((~SCON0 & 0x02) == 0x02);   //-- wait while the transmission is going on
        TI0 = 0;//SCON0 &= ~0x02;                                          //-- clear TI0
}


//-------------------------------------------------------------------------
// Ramp_Duty
//-------------------------------------------------------------------------
//
// Provides ramp for PWM duty cycle
//
//
void Ramping_Duty (void)
{

//*************** Ramp Function **************//
        if (Duty < 180) {
                Duty = Duty + 3;
        }
        else {
```

```
                Duty = 180;
        }

}

//-----------------------------------------------------------------------------
// Send_PWM
//-----------------------------------------------------------------------------
//
// Sends the new duty cycle to the hardware PWM
//
//
void Send_PWM (void)
{

//*************** Send to PCA **************

        if (Duty == 0) {
                PCA0CPH0 = 256;
        }
        else {
                PCA0CPH0 = Duty;
        }
}

//-----------------------------------------------------------------------------
// init
//-----------------------------------------------------------------------------
// general initialization
//
//
void init(void)
{
uchar n;
uint m = 0;
        WDTCN = 0x07;                                   // Watchdog Timer Control Register
        WDTCN = 0xDE;                                   // Disable watch dog timer
        WDTCN = 0xAD;

        OSCXCN = 0x67;                                  // select crystal oscillator, f>6.7MHz
        for (n = 0; n != 255; n++);           // wait 1ms for osc to start
        while ((OSCXCN & 0x80) == 0){ // wait for xtal to stabilize
                if(!++m)
                        break;                                  // abort if it won't start
        }

        OSCICN = 0x0C;                                  // switch to external oscillator

// Configure the XBRn Registers
//      XBR0 = 0x04;
        XBR0 = 0x0C;                                    //
        XBR1 = 0x00;                                    //
        XBR2 = 0x44;                                    // Enable the crossbar

// Port configuration (1 = Push Pull Output)
        P1MDOUT = 0x60;                                         // Output configuration for P1
        P1 |= 0xFF;
        P1_6 = 0;

        P0MDOUT = 0xFF;                                         // Output configuration for P0
CEX0,1,2,3 push pull
        P0_4 = 1;
        P0 &= ~0xF0;

        P2MDOUT = 0xFF;                                         // Output configuration for P2

        P3MDOUT = 0x10;                                         // Output configuration for P3
        P3_0 = 1;
        P3_1 = 1;

        P74OUT = 0xFF;                                  // Output configuration for P4-7
(P7[0..3] push pull)
        P4 = 0xFF;
        P5 = 0xFF;                                              // Turn the P5 LEDs off

}
```

```
//-----------------------------------------------------------------------------
// PCA0_init
//-----------------------------------------------------------------------------
//
void PCA0_Init(void)
{
        PCA0MD = 0x03;         // use SYSCLK/4 as time base
                                        // enable counter overflow interrupt
        PCA0CPM0 = 0x02;       // use PCA module 0 for 8 bit PWM generation

        PCA0L = 0;

        PCA0CPL0 = 0;
        PCA0CPH0 = 0;          // duty cycle

        PCA0CN = 0x40;                 // enable PCA0 counter
}

//-----------------------------------------------------------------------------
// delay routines
//-----------------------------------------------------------------------------
//
// delay using spin wait
//
void small_delay(char d)
{
        while (d--);
}

void large_delay(char d)
{
        while (d--)
                small_delay(255);
}

//-----------------------------------------------------------------------------
// Interrupt Service Initialization and Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Timer0_Init
//-----------------------------------------------------------------------------
// Configure Timer0 to auto-reload and generate an interrupt at interval
// specified by <Freq> using SYSCLK/12 as its time base.
//
void Timer0_Init(void)
{
        int     Freq;
        Freq = 10000;
        //-- set up Timer 0
        CKCON &= ~0x08;        //-- T0M=0; Timer 0 uses the system clock / 12
        TMOD |= 0x02;  //-- Timer 0 in Mode 2 (8-bit auto-reload)
        TH0 = -(SYSCLK / 12 / Freq);
        IE |= 0x02;
        TR0 = 1;               //-- start Timer 0 (TCON.4 = 1)
}

//-----------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------
// - reset timer 0 overflow flag, which causes the interrupt
// - update 100us, 10ms and 1s counters
void Timer0_ISR(void) interrupt 1
{


        if (++t_100us > 9){
                t_100us = 0;
                if(++t_1ms > 9){
                        t_1ms = 0;
                        if(t_10ms == 4 || t_10ms == 9){
                                ramp_flag = 1;
                                startup_timer++;
                        }
                        if(++t_10ms > 9) {
```

```
                                t_10ms = 0;
                                if(++t_100ms > 9) {
                                        t_100ms = 0;
                                        Tilt_flag = 1;
                                        if(++t_1s > 60) {
                                                t_1s = 0;
                                                ++t_1min;
                                        }
                                }
                        }
                }
        }
}

//-------------------------------------------------------------------------
// ADC0_Init
//-------------------------------------------------------------------------
//
// Initialise ADC0
//
//
//
void ADC0_Init(void)
{
        REF0CN = 0x07;          //-- Enable internal bias generator and internal reference
buffer
                                        //   Select ADC0 reference from VREF0 pin
                                        //      Internal Temperature Sensor ON
        ADC0CF = 0x80;          //-- SAR0 conversion clock=1.3MHz approx., Gain=1
        AMX0CF = 0x00;          //-- 8 single-ended inputs
        AMX0SL = 0x00;          //-- Select channel 0
        ADC0CN = 0x80;          //-- enable ADC0, Continuous Tracking Mode
                                        //   Conversion initiated on bit write, ADC0 data
is right justified
}


//-------------------------------------------------------------------------
// UART1_Init
//-------------------------------------------------------------------------
//
// Initialise UART1
//
//
//
void UART1_Init(void)
{
//      CKCON |= 0x10; //-- T1M=1; Timer 1 uses the system clock 22.11845 MHz
//      TMOD |= 0x20;  //-- Timer 1 in Mode 2 (8-bit auto-reload)
//      TH1 = 0x70;             //-- Baudrate = 9600
//      TR1 = 1;                //-- start Timer 1 (TCON.6 = 1)


        //-- Set up the UART1
        PCON |= 0x10;  //-- SMOD1=1 (UART1 baud rate divide-by-2 disabled)
        SCON1 = 0x50;  //-- UART1 Mode 1, Logic level of stop bit ignored
                                //   and Receive not enabled

        //-- enable UART1 interrupt
        EIE2 |= 0x40;
//      EIP2 |= 0x40;  //-- set to high priority level

        SCON1 &= ~0x01;         //RI1 = 0;      //-- clear the receive interrupt flag; ready to
receive more
        SCON1 &= ~0x02;         //TI1 = 0;      //-- TX1 ready to transmit
}

//-------------------------------------------------------------------------
// UART1_ISR
//-------------------------------------------------------------------------
//
// UART1 Intterupt (Tilt Sensor)
//
//
//
void UART1_ISR(void) interrupt 20
```

```
{
        //-- pending flags RI0 (SCON1.0) and TI1(SCON1.1)

        if ((SCON1 & 0x01) == 0x01)//( RI1 == 1)     //-- interrupt caused by received byte
        {
                received_byte1 = SBUF1;       //-- read the input buffer
                SCON1 &= ~0x01;       //      RI1 = 0;                          //-- clear the
interrupt flag
                received_str1[index1] = received_byte1;
                index1++;
                if (index1 > 15) index1 = 0;  //-- wrap around the array

                if (received_byte1 == 0x0a)           //-- check for NULL character
                {

                        index1 = 0;
                        y = 0;

                        for (x=1; x<6; x++)
                        {
                                xaxis[y] = received_str1[x];
                                y++;
                        }
                        xaxis[5] = 0x00;
                        y = 0;
                        for (x=9; x<14; x++)
                        {
                                yaxis[y] = received_str1[x];
                                y++;
                        }
                        yaxis[5] = 0x00;
                        xResult = atof(xaxis);
                        yResult = atof(yaxis);

                        if (yResult < 0)
                        {
                                yResult = yResult * -1;
                        }

                        if (yResult > 6)
                        {
                                Hill_mode = 1;
                        }

                        if (Hill_mode == 1 && yResult < 4)
                        {
                                Hill_mode = 0;
                        }
                }
        }
        if ((SCON1 & 0x02) == 0x02)   //TI1 == 1)    //-- interrupt caused by transmitted byte
        {
        }
}

//-------------------------------------------------------------------------
// UART0_Init
//-------------------------------------------------------------------------
//
// Initialise UART0
//
//
//
void UART0_Init(void)
{
        CKCON |= 0x10; //-- T1M=1; Timer 1 uses the system clock 22.11845 MHz
        TMOD |= 0x20;  //-- Timer 1 in Mode 2 (8-bit auto-reload)
        TH1 = 0x70;           //-- Baudrate = 9600
        TR1 = 1;              //-- start Timer 1 (TCON.6 = 1)


        //-- Set up the UART0
        PCON |= 0x80;  //-- SMOD1=1 (UART0 baud rate divide-by-2 disabled)
        SCON0 = 0x50;  //-- UART0 Mode 1, Logic level of stop bit ignored
                                    //   and Receive not enabled
```

```
        //-- enable UART0 interrupt
        IE |= 0x10;
//      EIP2 |= 0x40;  //-- set to high priority level

        RI0 = 0;       //-- clear the receive interrupt flag; ready to receive more
        TI0 = 0;       //-- TX1 ready to transmit
}

//-------------------------------------------------------------------------------
// UART0_ISR
//-------------------------------------------------------------------------------
//
// UART0 Intterupt (Communication)
//
//
//
void UART0_ISR(void) interrupt 4
{
        //-- pending flags RI0 (SCON1.0) and TI1(SCON1.1)

        if (RI0 == 1)  //-- interrupt caused by received byte
        {
                received_byte0 = SBUF0;       //-- read the input buffer
                RI0 = 0;                          //-- clear the interrupt flag
                received_str0[index0] = received_byte0;
                index0++;
                if (index0 > 15) index0 = 0;  //-- wrap around the array

                if (received_byte0 == 0x0a)           //-- check for NULL character
                {
                        index0 = 0;

                        if (received_str0[0] == 'G'){
                                button_flag = 1;
                        }
                        if (received_str0[0] == 'S'){
                                finished = 1;
                        }
                }
        }
        if (TI0 == 1)  //-- interrupt caused by transmitted byte
        {
        }
}
```