MASSEY UNIVERSITY
ENGINEERING

SCHOOL OF ENGINEERING AND ADVANCED TECHNOLOGY

Design and Development of a Modular Framework to Integrate Sensors and Actuators

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

Masters of Engineering
in
Mechatronics

at Massey University, Manawatu, New Zealand.

Brendan Taylor

2016

SUPERVISORS

a. **Gourab Sen Gupta**
b. **Ken Mercer**

Abstract

This thesis details the research and development of a versatile electronic monitoring and control platform, influenced by the Internet of Things (IoT), mass configurability, modularity, expandability and ease of use. The generic framework which has been designed and tested aims to provide a platform to build a wide variety of specialised systems to integrate sensors and actuators.

A central processing unit manages modular hardware devices connected by a serial network. Only the required hardware units are chosen to constitute a system for an application. The processing unit uses modular task handlers to manage the system. The web-based user interface provides multi-platform system access using a web browser. The website is dynamically generated from the system configuration.

While the framework is generic, for testing its efficacy, it was applied to a seed and fertilizer spreader to monitor and control the application rate. This application requires coordinated control of actuators using inputs from multiple sources, including sensors, machine states, a database, other processing tasks, and the operator.

The implementation was successful in achieving reliable control of the seeding rate, based on the tractor ground speed. The practical implementation exhibited a high level of expandability and modularity. The prototype system has also highlighted a few issues which can be addressed in future revisions to improve the versatility and robustness of the framework.

Acknowledgments

I would like to express great appreciation to Associate Professor Gourab Sen Gupta and Ken Mercer from the School of Engineering and Advanced Technology at Massey University for their support, guidance and encouragement throughout this project. Thank you for your patience and the opportunities you have provided me.

I would like to thank Reese Engineering for the opportunity to undertake this project and for providing the resources necessary.

I would also like to thank Philip Williemsen from Reese Engineering for his support and guidance during the project development.

I would like to express my appreciation to Mathew Flemmer for proof reading my thesis.

Finally, I thank my family and friends for their encouragement and support throughout my studies.

Table of Contents

1.0 Introduction

1.1 Motivation

The system proposed in this thesis is being developed to provide a generic versatile electronic monitoring and control platform that can be used in a wide variety of applications. This thesis details the development of a flexible modular framework and its application to a seed and fertilizer spreader. The inspiration for this project came from the development of a dedicated monitoring and control system for a wide range of agricultural machinery. Due to the versatility and expandability required, the focus of the proposed architecture is to provide a generic platform.

Automation and supervised electronic control systems have a big impact on process efficiency across a wide range of fields. Most of these platforms are specialised standalone systems. With advancements in information technology, there is a trend towards remote system management through the internet. Many of these systems develop the framework from the ground up. These systems are specialised, providing a refined solution. The proposed system architecture aims to provide a universal platform, for which both specialised and generic implementations can be developed to a level comparable to the specialised systems. A generic versatile framework should provide a convenient platform for implementing electronic monitoring and control systems.

Agriculture is the dominant sector of national income for many countries [1]. Advancements in information technology have significantly changed the field of agriculture over the last two decades. This has led to major improvements in farm management and process control using the concept of precision agriculture. Precision agriculture improves crop performance and environmental quality by managing spatial and temporal variability with the use of information technology [2].

With the movement towards precision agriculture, control systems are needed to provide coordinated autonomous control, handling more data and carrying out complex tasks. Precision agriculture uses data analysis and process optimization to increase crop yields, reduce wastage and has a lower environmental impact [3]. Automation and precision agriculture is having a significant impact on modern farming, improving operational and process efficiency greatly.

Figure 1: John Deere 4560 Row-Crop Tractor, 155 hp, 1992-1994 [4]

Tractors are used to operate nearly all agricultural implements and are essential in agricultural operations [5]. Traditionally, the most important feature of a tractor was its horsepower. These tractors used driver operated hydraulics for control of attached implements. Figure 1 shows an example of a tractor available in the early 1990's. These tractors prioritised power over operator comfort. Operational efficiency relied heavily on the skill of the driver to make the adjustments required. Many machines also used a ground wheel to maintain the relationship between ground speed and shaft speeds.

Figure 2: John Deere 8245R Row-Crop Tractor, 245 hp, with the CommandView III Cab, 2014-present [6]

Tractors have advanced significantly over the 20 years; modern tractors now prioritise efficiency and comfort as well as horsepower. Tractors today range from 30 hp up to 600 hp, covering a large range of rolls, in both small and large farms [5]. They use embedded electronic systems to accurately control many operational aspects, including: Engine performance; auto steering and navigation; intelligent implement control; and performance

monitoring. Large farms use a coordinated fleet of tractors and machinery to improve operational efficiency and crop performance. An example of a modern tractor can be seen in Figure 2.

This project is being developed with Reese Engineering, a farm machinery manufacturer based in Palmerston North, New Zealand. Reese Engineering have identified that there is great demand for electrically driven farming machinery and believe that the future of the farming industry is precision agriculture. New Zealand's agricultural industry is largely driven by the rapidly growing overseas demand [7], and is the 12th largest agricultural exporter by value [8]. In 2009, agriculture contributed 4% to New Zealand's GDP [9].

There are many commercially available electronic monitoring and control systems for operating seed and fertiliser spreaders. Many of these systems are standalone, with a dedicated display mounted in the tractor cab. As more systems are becoming electronically controlled, these systems clutter the cab. The proposed architecture should provide a cooperative control system, which can be used to provide higher level of system integration and control, while reducing cab clutter and allowing for remote management.

Current state of the art agricultural control systems provide coordinated fleet management of a large range of farming machinery. These systems exhibit a high level of cross compatibility and can be managed by both local and external operators. This project focuses on providing a web based control system that features modularity, flexibility and simplicity, as well as complete system configurability. The system requirements for applying this system to agriculture provide a good benchmark for a modern generic monitoring and control system.

1.2 Operating Environment

The operating environment of this system depends on the application it is configured for. The working conditions could range from mild to harsh. The operating environment of agricultural machinery is diverse. The developed control system needs to operate effectively in: high and low temperatures; high humidity; dusty and muddy conditions; strong rain and water blasting; high levels of vibration; and mechanical shock. These aspects have influenced the system hardware development and construction. Developing for this reasonably harsh environment will allow the system to operate in a wide range of environmental conditions.

1.3 Aims and objectives

This project aims to develop a system architecture capable of being adapted to fulfil a wide range of monitoring and control tasks across many different applications. The platform function needs to be customisable, expandable and provide the tools to achieve complex control and automation of many different processes.

The developed platform will be configured to be implemented as a seed rate controller for agriculture. This demonstrates the system's configurability, versatility and the ability to be applied to different applications. The system functions of a modern seed rate controller are reasonably diverse, utilising inputs from multiple sources to carry out complex tasks.

Applying the developed platform to this application will provide a reasonable metric for evaluation.

1.4 System Overview

In its current state, the platform consists of a central controlling unit, which manages hardware peripherals attached to a serial network. The architecture has been influenced by: the Internet of Things (IoT); mass configurability; modularity; and expandability. The attached devices can be selected to handle any control or monitor tasks required of the system. These devises are strictly managed by the central controlling unit, following the master-slave model of communication. The master device has a completely customisable modular layer for handling and processing the slave tasks. A library of module templates defines the method of handling data streams between: the slaves; the database; the user interface; and other module instances. Any number of module instances can be created using these templates, providing the required data streams are available.

The module template library can be easily expanded upon to increase the systems functionality. The user interface is provided by a website, currently hosted locally on the master device, which provides a Wi-Fi access point. This allows for multi-platform connection using a web browser and accessibility in remote regions without internet access. The website is dynamically generated based on the module templates and module instances of the system. The website allows the user to view, create, edit and control module instances. This architecture allows the platform to be tailored to different specialised applications through the addition of slave devices and module templates. Over time, an extensive library will be created, further expanding the systems possible application, creating a multipurpose and multidisciplinary platform.

Module templates and slave devices were developed to provide ground speed dependent seed application rate control for a Seed Drill. The system construction was developed for the operating environment of farming machinery.

1.5 Thesis Overview

This document details the stages of development in producing a working system prototype in software and hardware, its application as a seed rate controller and the design challenges encountered. The project was broken into two stages. The first stage acted as a proof of concept for the proposed software architecture. The second stage took what was learnt in the first stage and developed a working system prototype to be applied to the new seed drill.

The thesis is presented as follows:

Chapter 2 Reviews methods and the current trends of sensor-actuator integration systems. This focuses specifically on techniques relevant to the development of a generic customizable sensor-actuator platform and its application in agriculture.

Chapter 3 Discusses the system requirements and design decisions determined in the initial development phase. This section also details the development of the initial system prototype. This system acted as a proof of concept, and

highlighted many important aspects to be included in the framework. This led to a second prototype being developed, described in chapter 4.

Chapter 4 Details the development of the proposed framework and its application as a seed rate controller for a Seed Drill. It covers the hardware and software design of the second prototype, as well as the environmental considerations.

Chapter 5 Collates and discusses the results of the experiments conducted. This includes testing of hardware subsystems as well as the system machine trials.

Chapter 6 Concludes the project and highlights the intended future work.

2.0 Survey of Sensor and Actuator Frameworks

This section reviews methods and the current trends of sensor-actuator integration systems. It focuses specifically on techniques relevant to the development of a generic customizable sensor-actuator platform and its application in agriculture. The key concepts investigated include: mass customization; the internet of things (IoT); wireless and web based monitoring and control systems; the key aspects and challenges of a flexible multi transducer system architecture; and precision agriculture. A review of existing monitoring and control systems for seeder machinery has also been conducted, identifying the prominent strengths and weaknesses.

The concept of mass customisation has been around since the late 1980s. It addresses the growing demand for variety and customisation in product design. Designing for mass customisation gives a competitive advantage and reduces manufacturing cost. Mass customisation is becoming increasingly popular in product design and is believed to be a fundamental element in future product development. There are many unique fields which mass customisation can be applied to. Successful mass customisation products must be versatile, modularised and well maintained. This concept is applied to hardware and software design [10].

The concept of mass customisation has become a dominant factor in product development and manufacturing, with wide adoption across many different markets. Various configurable web-based systems have been developed for real world applications [11].

The Internet of Things (IoT) paradigm is growing rapidly with the increasing integration of wireless communications in technologies. IoT refers to the interconnectivity of different technologies through a shared network to reach a common goal. The application of IoT is vast and is growing significantly, with more products and services becoming accessible online. IoT has many social and commercial applications, from interacting with house hold items, to providing automation of industrial processes. The greatest challenge for this technology is guaranteeing privacy and security. There are many interpretations of the paradigm and little regulation. There are many efforts to define standard protocols and procedures for wide spread adoption. Devices require a communication interface to be utilised on a network. Often the different technologies involved have their own unique dialect, requiring a layer of abstraction to align the communication with a common language and procedure [12]. There is a growing trend to develop sensor networks around IoT [13].

Flexible, generic wireless monitoring and control systems are being developed to interface with a variety of different sensors and actuators through a central control unit. This structure allows for new sensors and actuators to be integrated. The user interface for this example is provided by a computer application. Compared to wired systems, wireless communication provides reduced system space, lower maintenance cost and lower manufacturing cost. New sensor/actuator units can be easily connected and replaced [14].

The use of web-based wireless sensor networks is rapidly increasing, using advanced information technology to manage sensor and actuator networks. Al Nuaimi et al. (2012) provide a review of this field, identifying key principles and challenges. The key design

principles outlined include: Flexibility, ease of expandable development, and centralised high level decision making. Key challenges identified include: data management efficiency and effectiveness; interoperability and extendibility; data integrity; computational cost; multi-user support; real time information processing; and security. There are various systems that provide web-based sensor networks for various functions, with the primary focus revolving around data collection and processing [15].

Zhang et al. (2004) developed a proof of concept for real time remote data collection of sensors using a web-based user interface. This system was used to facilitate the operation of a plant nursery. This is an example of a web-based sensor network being applied in agriculture [16].

Scarlett (2001) highlights the need for a universal coordination system to monitor and control the electronic systems built into modern tractors and farming machinery. This reduces the workload of the operator and increases the system efficiency, providing agronomic and economic benefits [17].

Sen Gupta (2007) proposes a system architecture for an expandable sensor/actuator system with constant overhead and minimal modification using a data-centric framework. This generic platform has possible applications in robotics, surveillance, office and factory automation, process and environmental monitoring and the home. This flexible system architecture explored is designed to provide efficient data flow by utilising a modular systematic network structure. This segregates data collection from data processing and control into separate system components. The system detailed provided simple configuration, with minimal system alteration and utilises multithreading to process the queuing data more effectively. This thesis will explore an architecture that utilises similar data-centric principles in both hardware and software. [18]

Precision agriculture improves crop performance and environmental quality by managing spatial and temporal variability using technology [2].

According to Zhang et al. (2002), there are many different variable influences on agricultural production, characterized in to: yield variability; field variability; soil variability; crop variability; management variability; and variability in anomalous factors. These attributes can be quantized and recorded on a topological map for reactive spatial and temporal crop management. This article outlines various methods of monitoring these attributes. An intelligent integrated control system is required to decisively manage outputs based on the measured parameters [19].

Lütticken (2000) claims that precision agriculture should stem across the whole agro-business. This publication suggests the need for an open software platform that can be operated by farmers or contractors from the office as well as in the field. This breaks down information barriers and will act as a powerful tool for quality management [20].

The ISO/IEC 11783 [21], or ISOBUS protocol standardizes serial control and communication network for forestry or agricultural tractors. It defines the format and method of data transfer between integrated electronics, including: actuators, sensors, control blocks,

storage devices and user interfaces. This provides interoperability required for implementing systems for precision agriculture. There are many existing ISOBUS compatible implements. Oksanen et al. (2005) outlines the process of developing ISOBUS compatible equipment [22].

There are many examples of systems that utilize the ISOBUS protocol in various ways, for example:

- The X30 Variable seed rate controller
- The Case AFS Pro 700 Display Interface
- A web terminal to access monitored information online [23]
- A remote control, automatic seeding depth controller [24]

There are examples of patents that use a modular system structure. These focus mainly on providing modular expandability by adding additional hardware blocks to fulfil specific tasks. Flamme et al. (2000) focuses on the development of control hardware blocks to provide variable seed rate monitoring and control functionality for individual planting rows. This device uses a push button LCD display to control, configure and monitor the hardware blocks. [25]

There are many examples of expandable modular control systems on the market. This investigation is specifically interested in systems with sensor and actuator integration. The majority of the systems found are very specialized or have a narrow application range. Some example applications include:

- Gas-exchange analysis [26]
- Modular vehicle control system [27]
- Modular patient monitoring system [28]
- Building security and control [29]
- Home automation [30] [31]

Mikhaylov and Huttunen (2014) detail the development of a modular, plug-and-play, distributed wireless sensor-actuator network (WSAN) with customizable software and hardware architecture [32]. This is an example of a generic expandable modular sensor/actuator control system. This study attempts to develop a universal WSAN system architecture to provide a platform that can be adapted for specific applications. The paper proposes a system consisting of hardware nodes which can support custom functional hardware modules to be attached. The node will automatically discover the module and download the module drivers. This architecture provides modularity in both hardware and software. These nodes are part of a distributed network. The project has been influenced by the concept of IoT. The system proposed in this thesis investigates a different software and hardware architecture.

The system framework detailed in this paper is being developed to be adapted to suit a wide range of applications. Over time, an extensive library of module types will be created, further expanding the systems possible application, creating a multipurpose and multidisciplinary platform. Different user interface templates can be created to give the system unique

aesthetics for custom applications. Further development of this system could lead to application in many different fields, for example:

- Many aspects of Agriculture, including Dairy; harvesting; seeding; quality monitoring; …
- Horticulture
- Home automation
- Process automation
- Remote environmental monitor/control stations
- Healthcare
- Education
- Manufacturing
- Smart metering
- Building security and control

The concepts of mass customization, IoT expandability and versatility have been used to influence the development of this project, aiming to provide a universal, customizable and expandable web based sensor-actuator monitoring and control system. This platform would be well suited to the field of precision agriculture. ISOBUS was not used as this system is developed as a generic monitoring and control system for applications beyond agriculture.

2.1 Current Monitoring Systems

Reese Engineering currently has two electronic monitoring devices, the "Seedmatic Areameter" and the "AirPro Monitor". These devices have been designed for the Seedmatic and the AirPro seed drills, but can be used for general use in agriculture. They both are mounted in the cab of the tractor and have cables that run to the sensors on the machinery.

The Seedmatic Areameter

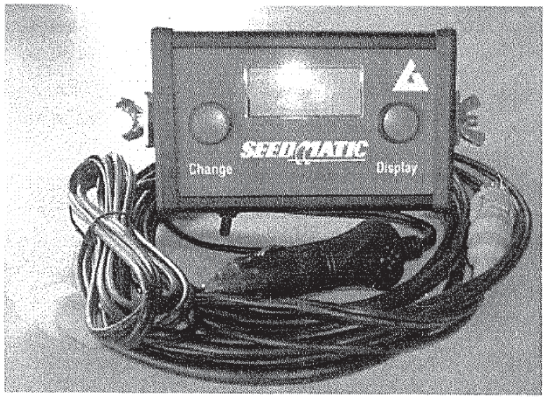


Figure 3: The Seedmatic Areameter

The Seedmatic Areameter uses a reed switch attached to a ground wheel to monitor: the area, the ground speed and the distance travelled. The system can display the data in metric or imperial units. The optional functions of this unit include: customisable low/high speed alarm, 4 section width switch inputs for spray area and the working time.

Table 1: Technical Specifications for the Seedmatic Areameter

Technical Specifications:	
Power Supply	DC 10 to 16 Volts, low current. Powered through cigarette lighter socket.
Fused	Not applicable.
Dimensions	150mm x 100mm x 63mm
Display	4 character, 7 segment display with optional backlight.
Processor	Standard Monitor unit - ST6265 processor 8MHz and a Memory Retention chip for storing options and counts.
Sensors	Magnetic Reed Switch – working distance: 10 to 25 mm.5 V • 1 for ground speed, distance and area calculations.
Alarms	Low or high shaft speed.

The "AirPro" Monitor is similar to the Seedmatic Areameter, with extended functionality. It has 3 monochromatic LCD displays, three status LEDs and 4 user input buttons for display selection and configuration. This unit has 5 reed switches to monitor: the ground speed, distance and area; the fertiliser and seed fan speeds; the drive shaft speed; and the drive leg position. It also has two capacitive proximity sensors to check the seed and fertiliser bin level. The system will sound an alarm when the bin levels are too low, the drive shaft has stopped and when the fertiliser or seed fan speeds are too low. When the drive leg is lifted, the area meter is paused and the alarms are disabled.

The AirPro Monitor

Figure 4: The "AirPro" Monitor

Table 2: The Technical Specifications for the AirPro Monitor

Technical Specifications:	
Power Supply	DC 10 to 30 Volts, low current. Powered through cigarette lighter socket.
Fused	Not applicable.
Dimensions	230mm x 170mm x 50mm
Display	3 x 4 character, 7 segment display with optional backlight.
Processor	Standard Monitor unit - ST6265 processor 8MHz and a Memory Retention chip for storing options and counts.
Sensors	Magnetic Reed Switch – working distance: 10 to 25 mm. 5 V • 1 for ground speed, distance and area calculations. • 2 for fertiliser and seed fan speed. • 1 for drive shaft speed. • 1 for the drive leg position. Capacitive proximity sensors. 12 V • 2 for checking if the fertiliser and seed bin levels are low.
Alarms	Low/high shaft and fan speeds

Existing Monitor Evaluation

These systems work well for basic monitoring functionality on the machines they are intended for. They are designed to be robust, water/dirt resistant and provide an easy way of monitoring the seeding machinery. Both of the units demonstrate a high level of reliability and durability in their working environment. The monochromatic LCD displays used have good visibility in harsh lighting and low production cost. Reese Engineering outsources the manufacture of these units. These systems are only capable of simple monitoring and the display is not suitable for providing a control interface.

There are many aspects from these systems that should be considered in the new system design. These include:

- The user interface is very minimal, bulky, and requires effort for the user to setup and configure. For a more complex, feature rich system, it is important to make it simple to configure and setup. The user interface needs to be simple, intuitive to operate and provide an easy way to monitor and control the machinery.
- These systems are dated with regards to the user interface and the components used. The new system should be made of newer components that will be available in 10 years. The system should also have a more modern interface.
- There is no data logging or job reporting built into the current monitors. Logging the monitored data and control updates will be useful for evaluating the performance of the machine, billing information and early detection of faults. This also provides the opportunity of adding environmental sensors for improved variable rate control.
- Hardwiring the system to the machinery requires additional setup and pack up times. Using a parallel sensor connection makes this process complex and time consuming. This issue becomes greater when the system functionality is scaled up as there will be more cables to connect.
- The information these systems monitor is very limited. The system are specialised for specific machines. This is why they want a new “all in one” monitoring and control system.
- The inputs to the system are fixed. For an “all in one” system, being able to change the inputs and outputs types available is preferred.
- The AirPRo monitor can only monitor 2 bins. Reese Engineering wants the new system to be capable of monitoring 4 bins.

2.2 Existing Seed Rate Controllers and Farm Management Systems

There is a wide range of products currently available that provide variable seed rate control. These devices range from being ridged and rudimentary, to sophisticated and customisable.

Standard system features that were exhibited across all devices investigated include:

- Monitors ground speed, distance and area
- Variable seed rate control from the ground speed
- Hopper level detection
- Fan speed monitoring
- High and low speed alarms
- Job reports
- User interface mounted in the tractor cab

The key features that present in many of the more sophisticated systems are:

- User friendly, full colour, touch screen graphical user interface.
- Customisable, multi-purpose platform to handle many tasks relating to the machinery.
- Customisable and configurable interface widgets and interface layout.
- GPS integration, with mapping and spatial seed rate control.
- Auto steering.
- Exportable boundary, coverage, logging and as-applied maps.
- Compatible with many different sensor-actuator components.

This review will focus on the more sophisticated systems, including: the X30, the Case IH Advanced Farming System, and the 7500.

The X30:

The X30 Seed Rate Controller claims to be the most flexible and accurate variable seed rate controller on the market [33]. It is the successor of the less feature rich X20 system. It was developed by Bourgault and Topcon Precision Agriculture and released in 2014 [34]. The key features of this device include:

- A large, 12.1" full colour sun readable display.
- Customizable display views with multi-touch and drag and drop info windows.
- Variable rate control for up to 8 products, including NH3 or liquid fertilizer.
- 3D Mapping and GPS to implement spatial variable coverage application rates.
- Job Export provides a summary of completed work including as-applied maps for each product.
- Auto steering.
- USB camera support.
- Utilizes the ISOBUS protocol.
- Unit conversion.
- Fan speed monitors and system alarms.

The X30 is compatible a wide variety of components for crop sensing and data mapping for precision farming [35]. These include: the AGI-4 Modular GNSS Steering system; the Apollo seeding application controller ECU; the ASC-10 Auto section and liquid rate controller; and the CropSpec crop canopy sensor. The X30 aims to replace all electric control systems on a tractor with one, easy to use interface.

This product is currently top of the market and comes in smaller, less feature rich models, the X25 and the X14. The difference between these products is mainly the hardware used, and the number of attachable peripheral systems. The X30 Seed Rate Controller can be seen in Figure 5.

Figure 5: The X30 Seed Rate Controller

These systems use computer processors to provide a feature rich, multi-tasking platform. The hardware specifications can be seen in Appendix 1.

Case IH AFS Section and Rate Controllers

Case IH have developed a range of products Advanced Farming System (AFS) for precision agriculture [36].

The **AFS Pro 700** display provides a colour touch screen interface to connect with all Case IH system equipment. This unit can be used with Case IH tractors, combines, cotton pickers, sprayers, balers, planters, seeders and tillage to control and monitor key functions and track important information. It has been developed to be compatible with a wide variety of equipment, utilising the ISOBUS protocol. This means that it can be used to control third party equipment, without the need for a full system replacement. The display has up to 6 tabs for, providing a large variety of monitoring and control features, as well as up to 3 video inputs. This device has been designed to replace all electronic display systems in the tractor cab with one simple display. It can be used for: remote valve timers and flow control, auto PTO, engine speed settings, the wheel slip alarm, calibrations and implement settings, as well as other key machine functions. The AFS Pro 700 can be seen in Figure 6.

Figure 6: The Case IH AFS Pro 700 display system

The graphical user interface is well laid out, with reasonable sided display widgets. The tabs allow the user to quickly navigate to the required information and reduce the on screen clutter. The graphics used are simple and easy to understand, but their aesthetics could be improved.

The **AFS Connect™** provides real time management and control of an entire fleet of machinery over the internet. It provides instant access to machinery location, system diagnostics, fuel levels, engine stats and more. It is accessed using a web-browser on any smart device and integrates the concept of precision agriculture across the entire operation. The ability to manage an entire fleet of machines from one remote device is a very powerful tool. This system can be accessed anywhere, providing the machinery has internet access. This system provides the operator with the same level of monitoring and control access as the tractor driver. This system has two way file transfer, allowing the user to download reports as well as upload spatial application rate maps directly to the machine. This system can be configured to be used for third party equipment, providing a great range of customisation and flexibility. This system is effectively an internet port of the operations AFS Pro 700 display units. The AFS Connect interface can be seen in Figure 7.

Figure 7: The Case IH AFS Connect interface shown on a tablet

The Connect interface is more modern than the AFS Pro 700, with higher quality widget graphics. All Case IH AFS systems use the AFS Pro 700 to provide the interface to the tractor operator.

The **AFS AccuGuide™** system provides sub-inch level steering and guidance for tractors, combines and sprayers. This works to reduce skips and overlaps to save on fuel, time, labour and product costs. It can also be connected with the AFS RowGuide for hands off steering during harvesting.

The **AFS Harvest** system is used to monitor map and evaluate the crop performance.

The **AFS AccuControl** is used to manage product application and hydraulic drives, with individual stream shutoffs to control to reduce overlap. It can control seed, fertilizer and liquid application for up to four sections and up to 48 rows. Each section can have a different rate.

The **ISO Task controller** is used to monitor and control equipment that utilises the ISO 11783 protocol using the AFS Pro 700 display.

Case IH have developed a software suite to manage, analyse and utilise precision farming information from the AFS product range. This package can be used to generate: yield maps, prescription maps, soil quality maps, and reports.

Case IH have developed a product range to address a wide variety of agricultural systems, implementing the concept of precision agriculture across the entire fleet operation.

The 7500:

The 7500 seed rate controller from Farmscan AG was released in 2015 [37]. It provides flexible task control for air-seeder machines. It has integrated GPS for map based variable seeding rate control. The 7500 console has a 4.8” colour touch screen display, smart warning alarms, and a USB port for file transfer (coverage mapping and job reports). The console device can be seen in Figure 8.

Figure 8: The 7500 Seed Rate Controller from Farmscan AG [37]

The console connects to the 3CH UniPOD using a CANbus line. The UniPOD is a configurable hardware device that can be customised to suit many different applications. It can be used to control hydrolic drives, linear actuators, DC electric motors and liquid flow control. It can monitor ground speed, shaft speed, fan speed pressure and more. To expand on this, multiple UniPOD devices can be connected to the CANbus line.

The information and controls are displayed using a tile and tab display. This provides a simple and user friendly interface that is easy to read and interact with. The tiles can be customised and rearranged to display any of the systems information. The UI has been designed to provide simple machine and display setup.

Mobile Agricultural Data Management Systems:

There are many systems that provide mobile precision agriculture data management, including: FarmLogs from FarmLogs; AgFiniti from AgLeader; AFS Connect from Case IH; AgroMet Weather Station from RainWise; Drone Management System from Event 38; AgStudio Pro from Map Shots; Libra Cart from Agrimatics; Connected Farm Bundles from Trinble; Harvest Mobile from John Deere; Claas Telematics from Claas; PLM Connect Wireless File Transfer from New Holland [38].

The AFS Connect system from Case IH is detailed above in the 'Case IH AFS Section and Rate Controllers' section.

These systems provide real time remote multi-platform access to deployed farming equipment. These systems cover: crop management; field quality surveying; fleet management; remote monitoring and control; and task automation.

Summary:

The control systems investigated address a wide range of the elements in precision agriculture, from rate control to fleet management and data analysis. These systems provide user friendly all in one, expandable package for farm machinery operation. These systems are reasonably expensive. The system requirements of a modern monitoring and control system in agriculture provide a reasonable metric for the development of a diverse universal system. These devices use inputs from multiple sources (sensors, maps, GPS, and the operator) to carry out complex control tasks to improve performance. They have user friendly interfaces, dynamic job reports and many support remote connection. The application of these systems is very specific to agriculture. This is the case with most sensor-actuator control systems, which are developed for a specific application. This provides a highly refined solution that tends to achieve better results than a universal system. The proposed system architecture aims to provide a universal platform, for which both specialised and generic implantations can be developed to a level comparable to specialised systems.

3.0 Initial Sensor and Actuator Framework Development

This project started out as a modern monitoring and control system for application over Reese Engineering's product range. The initial project stages lead the development focus towards designing a generic versatile electronic monitoring and control platform that can be applied to a much wider variety of applications.

Reese Engineering want a tablet type interface that can be used to monitor or control the machinery they produce. This system is to replace their existing Seeder monitor units. They currently produce 5 different types of machinery, with 15 models in total. Reese Engineering are also developing a new model of Seed Drill that incorporates electric drives for seed and fertilizer application. This machine is to be operated using this electronic monitoring and control system.

The purpose of this project is to develop an expandable, multi-purpose coordinated control system that is capable of being installed on any of the machinery Reese Engineering produce.

Reese Engineering would like one system that fits all applications, as opposed to creating individual solutions for each machine. The focus of this project is on creating a multi-purpose, versatile framework that Reese Engineering can incorporate across their product range. Versatility has become a huge focus of this project. To provide this level of versatility, the software has been developed with expandability as the primary influence. To implement this, modular software architecture has been chosen.

There is a great variety of aspects to monitor and control in agriculture, making it a reasonable application for the initial development of a generic modular framework. Many of these aspects are easily related to other applications, including: home automation, industrial automation, process control, building security, parking management, etc.

The focus of the initial framework was to set up the systems core software functionality for a monitoring and control system. This stage acts as a proof of concept and the results of this have been used to influence the development of the final framework. The recommendations and improvements noted lead to a complete revision of both the hardware and software implementations.

An iterative development process was used. Over the course of development, three different software versions were created, each improving on the modular architecture.

- **Version one** focused on creating an initial proof of concept and becoming familiar with the technologies and techniques required.
- **Version two** aimed to establish the core software architecture for both monitoring and control. However, due to various limitations and the suggested improvements, a third revision was required.
- **Version three** focused on creating a stable software and hardware platform that can be used on the machinery as well as significantly changing the way modules were

implemented. Versions one and two were developed as part of stage one, and version three was for stage two.

Version one and two are discussed in this section. Version three is discussed as part of the Proposed Framework section.

3.1 System Requirements

The system requirements for this project were derived during the initial project planning of stage 1, and are based on meetings with representatives from Reese Engineering. The system requirements were categorised into: the user interface, the hardware design, the system design and general. The system design category incorporates the software design, the supporting software and the processor capabilities.

User Interface:

1. Reese Engineering wants a tablet type Interface.
2. Needs to be simple, user friendly, and intuitive to use.
3. Requires good visibility in harsh lighting conditions.
4. Enable the user to change data units. For example: km or miles, hectares or acres...
5. Built in operators manual, accessible through the interface and easy to find.
6. Have the ability to display video in real time.
7. Provide user input for calibration and control.

Hardware Design:

1. The date and time needs to be maintained when the system is powered down.
2. Have an uninterruptable power supply to enable the system to save and shutdown safely.
3. Durable construction, capable of surviving in harsh working conditions.
4. At least 25 GPIO pins, with 4 capable of PWM for Boolean and speed inputs, Boolean output and motor control. This does not include the functions with unidentified I/O requirements.
5. Needs to be capable of video and GPS input.
6. Needs data storage capacity for data logging.

System Design:

1. This system needs to be multipurpose, an ‘all in one’ package that can be used for a variety of machinery.
2. Keep a record of the monitored data and control updates.
3. The system needs to be able to survive being in the working environment for an extended period of time.
4. Stream real time video from video inputs.
5. Can maintain outputs, and reliably monitor inputs to the system.

General:

1. Safe failure mechanisms need to be incorporated to ensure the machinery does not undergo undesired changes if the system fails.
2. Machinery operation will not be reliant on this system. If the system goes down, the machinery can still be operated.
3. Buying components and boards off the shelf is preferred to creating them from scratch to minimise construction time and the resources required.

4. All hardware and software used needs to be accessible or compatible with new releases in 10 years.

Machinery Description

Reese Engineering provided a list of the machinery that could benefit from this system as part of the initial specifications. This list identifies possible functions that can be monitored and controlled for 5 different types of equipment. The two tables below summarise the monitoring and control aspects for each product listed. These tables indicate the I/O types required to achieve the functions.

Table 3: The Aspects to monitor for the 14 machinery Reese Engineering produce

Equipment type	Aspects to Monitor →	Run Time	GPS	Ground Speed	Area covered meter	Fertiliser Application Rate	Seed Application rate	Shaft Stop Alarm	Bin Level Sensor	Fan Speed	Video Link	Seeding Depth	Amount of Bales Wrapped
Fertiliser Spreader Range	Small ATV Spreader	X	X	X	X	X							
	Tractor Linkage Spreader	X	X	X	X	X							
	Bulk Trailed Spreader	X	X	X	X	X							
Hay Mowers Range	Linkage Hay Mowers	X	X	X	X								
	Trailed Hay Mowers	X	X	X	X								
Helliwrapper	Helliwrapper	X	X										X
BaleBug	Trailed self-powered	X	X										
	Tractor linkage model	X	X										
Seed drill Range	Grassfarmer	X	X	X	X		X	X	X				
	Roller Drills	X	X	X	X		X	X	X				
	4000 Series	X	X	X	X	X	X	X	X				
	8000 Series	X	X	X	X	X	X	X	X				
	AirPro	X	X	X	X	X	X	X	X	X	X		
	SeedKing	X	X	X	X	X	X	X	X	X	X	X	
	Required I/O	1x Bool	4x Bool	1 x Speed	N/A	~4 Speed	N/A	4x Bool	~4 speed	USB	?	1x Bool	

Table 4: The Aspects to control for the machinery Reese Engineering produce

Equipment type	Aspects to Control →	Calibration	GPS Map Rates	Fertiliser Application Rate	Seed Application Rate	Variable Application rate	Tramlining	Depth Control	Hopper Gate
Fertiliser Spreader Range	Small ATV Spreader	X	X	X					X
	Tractor Linkage Spreader	X	X	X					
	Bulk Trailed Spreader	X	X						
Seed drill Range	Grassfarmer	X	X		X	X			
	Roller Drills	X	X		X	X			
	4000 Series	X	X	X	X	X			
	8000 Series	X	X	X	X	X			
	AirPro	X	X	X	X	X	X		
	SeedKing	X	X	X	X	X	X	X	
	Required I/O	?	N/A	~4x PWM	?	?	?	1x Bool	

Many of the machines share similar aspects to monitor and control. The majority of these can be achieved through Boolean and speed inputs and outputs. For this project the core system design will be around providing Boolean and speed inputs and outputs.

3.2 Initial Design Decisions

The initial design decisions shaped the development of the system. These were made based on the system requirements determined during the initial planning. This includes: the software model; the user interface; the processor; and the software tools used.

The Software Model:

The required tasks, such as monitoring the speed or controlling the application rate, will be implemented in software as modules. A module type will be designed for each unique task, acting as a template. Multiple instances of a module type can be created from these templates. A configuration file is used to create the module instances required for the machines operation on startup. The library of module types can be expanded, increasing the possible applications. The systems functionality is defined by the configuration file. This approach provides the expandability and versatility required to suit a wide variety of monitoring and control applications.

Each module type has a data handler which can:

- Process sensor and user input data
- Display information to the user
- Save the data to a database
- Change the state of actuators

The structures of all modules are standardized for simple creation and execution. The number of module types and module instances that can be created using this system is unlimited.

This approach provides Reese Engineering with the ability to configure the system for a new machine, or extend the user interactivity with existing ones. Using a modular software architecture vastly extends the possible applications of this system.

The Processor:

To achieve the desired versatility and the tablet type interface, a powerful processor is required. A single board computer (SBC) is ideally suited for this application, with many useful, advantageous features:

- Data logging
- Expandable storage
- Data acquisition through general purpose input/output (GPIO) pins or a serial GPIO extension
- Support high level programming languages such as C++ and Python
- Serial communication protocols, such as: SPI, UART, USB, I2C, CAN interfaces.
- ADC and PWM generator
- Video recording and streaming
- Computer networking and hosting a webserver
- Multi-tasking and multi-treading

A SBC is a small computer similar to those found in smart phones and are relatively cheap, ranging from between $5-300USD. The SBC can connect to the sensors and actuators using the integrated GPIO pins, or using a serial GPIO extension.

The User Interface:

The two main options considered for providing the tablet type user interface were: using an HDMI screen with a push button or touch interface; or a touch screen smart device, connected to the computer through Wi-Fi or Bluetooth.

Using the HDMI screen means that the system will need to be completely wired. This can cause some complications when the machinery is detached from the tractor. The off the shelf mobile smart device will be more convenient as it:

- Reduces the required length of the wiring loom and does not require costly cable sockets for disconnecting the machinery.
- Has a similar production cost to creating a custom tablet, with a shorter development time.
- Provides a portable user interface that can be mounted in the tractor cab, or carried around the machinery during operation or calibration.
- The device can be reused as an interface for other Aitchison machinery.
- The operator can use their personal devices as the user interface
- Data can be downloaded directly to the operator's personal device.

A mobile device will be used for this system. This will have a mount in the tractor cab with a built in sun shield and will communicate wirelessly to a central 'hub'. This hub will be mounted permanently on the machinery and will handle data processing, data storage, and interfacing directly with the sensors, actuators and user interface.

There are two main ways of providing a user interface using this system. These are, using a mobile application, or a website. The mobile application will require significantly less data transfer, as the interface will not be sent over the network, and will provide a simple, user friendly way of interfacing with the machinery. A website is very versatile, has a shorter development time and will run independent of the device's operating system. The operator's devise will only need a supported web browser to access and operate the system. This provides cross compatibility across different tablet models, as well as different operating systems, including Windows, Linux, Apple's IOS and Android.

To provide access to the webpage, the SBC will host the website and will be configured as a Wi-Fi access point that the operator can connect to.

The Software Tools:

This system is dependent on many software packages, libraries, and programming languages. The programming languages include: Python, HTML, CSS, JavaScript and SQL.

Third party software packages include: Python 2.7, MySQL, Apache, HostAPD, Dnsmasq, and Samba (Optional)

Libraries include: JQuery, JQuery Mobile, MySQLdb, Autobahn, and Twisted.

The operating system used is the Raspbian distribution of Linux. This distribution was developed specifically for the Raspberry Pi. It was run in headless mode to reduce the system processing, as the graphical user interface (GUI) is not needed for this system.

To setup the virtual Wi-Fi access point, the packages "hostapd" [39] and "dnsmasq" [40] were installed. Hostapd is an IEEE 802.1X/WPA/WPA2/EAP/RADIUS authenticator and IEEE

802.11 access point. Dnsmasq provides network infrastructure like router advertisement, network booting, DNS and DHCP for small networks.

Python was chosen as the programming language for the server side applications [41]. Python is an interpreted language, and hence is slower than compiled languages like C++. The slower execution speed is more than sufficient for this application. Python is a high level language that is open source, dynamic, portable, object oriented, modular and is extensible into C and C++. It can also be used for web development.

The website will use HTML, CSS and JavaScript. HTML is used for the page structuring, CSS (Cascading Style Sheets) is used for visual styling, and JavaScript is used for all interactive functionality. These are built into every web browser. JavaScript is executed in the client's web browser, which means that functions can be run immediately as it does not need to wait for the server to respond. JQuery and JQuery mobile were used to simplify the web development and provide tools to create a responsive website for mobile devices.

A database is required to provide the data logging and can also be used for the system configuration. MySQL was chosen as the database [42]. It is an industry standard program for use in web design and providing data logging. It is also easy to interface with from many different programming languages, including both shell script, Python and JavaScript. The Python library "MySQLdb" will be used to provide an object-oriented API that gives easy access and management of the MySQL database [43].

Apache will be used to host the website. It is an industry standard application for hosting webserver's and provides access to the web pages stored in its directory [44]. This system will implement the "LAMP" model for a web server [45]. LAMP is largely made up of interchangeable components, and for this application it will consist of: **L**inux, the **A**pache HTTP Server, **M**ySQL database and **P**ython.

Python dictionaries were used to store all of the module data. A python dictionary is an unordered set of key - value pairs, where the key is used to access the value. The key can be any immutable type, and the value can be any data type including dictionaries, objects and function calls. Python dictionaries are in the same format as JavaScript objects [46]. A dictionary can be sent to the webpage using JSON encoding, and decoded into a JS object. This makes them a good fundamental building block for expandable modular software architecture.

The installation and configuration of the operating system and system dependencies can be seen in Appendix 2.

3.3 Initial Framework

The aim of stage one was to set up the core system functionality of a monitoring and control system for use on agricultural machinery. The outcome of this stage will determine whether the project will be continued towards commercialization.

3.3.1 Hardware design

This computer will require Wi-Fi (either built in or a USB Wi-Fi dongle) and an I/O extension cape to provide the required functionality. Using the Wi-Fi dongle, the SBC will be set up to host a local network wirelessly. The webserver, hosted on the Hub, will be accessed using a Wi-Fi capable smart mobile device, such as: a tablet, smart phone, netbook or laptop. The system can operate in remote areas as the webserver will be hosted locally and will not require an internet connection.

The I/O extension cape is needed to connect the SBC with the sensors and actuators, and to protect it from electrical damage. Many of the sensors and actuators have significantly different power requirements which the computers GPIO cannot safely tolerate. As a result, the I/O extension cape may need to: electrically isolate the computer; apply voltage conversions; and to drive the outputs and read the inputs. This board may include motor drivers, relays, optical isolators and voltage regulators. Power to the system will come directly from the tractor. The voltage regulators are needed to supply the computer and components with the correct voltage. To insure the computer does not shut down unexpectedly, an uninterruptable power supply may be required if the power from the tractor is not reliable *[Hardware Design Specification 2].* The sensors and actuators used will be application specific, and will require a different I/O cape for each application. As the emphasis of stage one is to setup the core system functionality, the focus was to develop the software architecture for the system. As a result, this stage did not cover selection of the sensors and actuators required.

The sensors and actuators, I/O extension cape and single board computer will be mounted on the machinery. The mobile device can either be portable, or mounted in the cab of the tractor. This system is shown below in Figure 9.

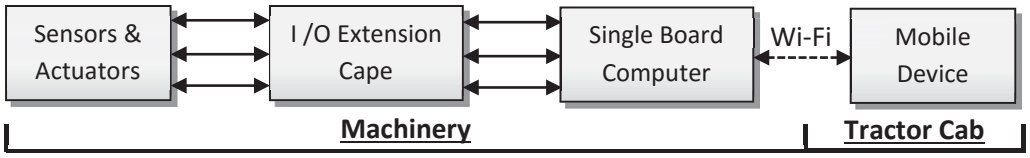


Figure 9: The hardware system structure design

Single Board Computer Selection:

The processor, memory and networking specifications for most SBC's on the market are more than capable of implementing this system. The choice in computer comes down to: providing Wi-Fi, have video input, and at least 25 GPIO pins, with at least 4 pins capable of PWM. This criteria is based on the system requirements outlined in section 3.1.

Initially 8 different single board computers were considered for this system. Of these, only 6 met the GPIO requirement. These devices are: the UDOO, the CubieBoard 2, the CubieTruck (CubieBoard 3), the Raspberry Pi Model B+ and the BeagleBone Black. Table 5 shows a summary of the considered Single Board Computers investigated.

Table 5: Feature summary of the main single board computers initially investigated

	GPIO	Other Features	Approximate Price (February 2015)
UDOO	76	2 USB hosts, uSD, HDMI , SATA II, 1GB DDR3 RAM,1G Ethernet, built in Wi-Fi	$173 NZD
CubieBoard 2	>96*	2 USB hosts, uSD, HDMI , SATA II, 1GB DDR3 RAM,4GB NAND Flash,10/100 Ethernet, 2 PWM	$109 NZD
CubieTruck	>52*	2 USB hosts, uSD, HDMI , SATA II, 1GB DDR3 RAM,4GB NAND Flash,10/100/1G Ethernet, built in Wi-Fi and Bluetooth	$165 NZD
Raspberry Pi Model B+	26	700MHz Processor, 4 USB hosts, uSD, HDMI, 512 SDRAM, 10/100Ethernet.	$49 NZD
BeagleBone Black rev C	65	1 USB hosts, uSD, uHDMI, 512 DDR3, 4GB eMMC, 10/100Ethernet, 8 PWM	$75 NZD

From these, two options stuck out from the others. These options were, using the BeagleBone Black (BBB) [47], or the Raspberry Pi B+ (RPi B+) [48] paired with a microcontroller to provide a GPIO extension and to maintain outputs. Both of these options cost approximately the same, however, the BBB has 65 on board GPIO pins, 8 of which are capable of PWM. The BBB is preferred as it has on board storage, superior GPIO, is faster, and it does not require a secondary program developed for a microcontroller. As many of the computers operate with the same OS and have a similar GPIO interface, it is possible for the programs developed to be ported to a new platform. This means that porting the system to future single board computers will be possible. This system will be able to be manufactured over the next 10 years.

The BBB does not have a real time clock, therefore one need to be incorporated with the I/O extension cape along with a power cell. The RTC is required for network authentication and accurate data logging.

I/O Cape Design:

A PCB cape for the BBB was developed to test the input and output functionality of the software system with hardware.

The aim of this circuit was to demonstrate the core system functionality.It consisted of:

- 2x LED's to demonstrate Boolean output.
- 2x on/off switches for Boolean input.
- 1x L298 dual full h-bridge motor driver for 2 speed outputs. This will drive 2 Pololu 25D mm gear motors [49].
- 2x 48 pulse per revolution quadrature hall effect encoder for speed input. These are attached to the motors, and have a total of 4 signal channels.

A 2x4 header was also added to access the BBB's SPI pins. This was added to experiment with an RF transceiver, but was later deemed unnecessary.

Motor encoders were used as the speed input providing a good speed input source, and allows for future development into providing feedback speed control. The L298 motor driver was used because Massey University has a large stock of them. This H-Bridge motor driver can control two motors and handle up to 2 amps per motor [50]. Figure 10 shows the pin layout for the L298 Motor Driver.

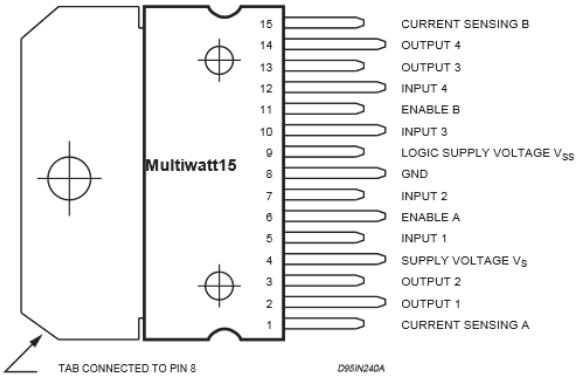


Figure 10: The L298 Motor Driver pin layout

Table 6 details the pin setup for the L298 motor driver:

Table 6: Description of the L298 pin usage

Pin Number	Setup description
1, 15	Not required
2, 3	Connected to motor A's power terminals
4	The Supply voltage V_s to the motors (6-12 Volts)
5,7	Motor A's direction control. 00: Break,10: Forward, 01: Backward, 11: Break
6	Motor A's Enable pin. Connect to a PWM signal for variable speed.
8	GND
9	The Logic supply voltage V_{ss} is the voltage source for the driver itself (5 Volts)
10, 12	Motor B's direction control. 00: Break,10: Forward, 01: Backward, 11: Break
11	Motor B's Enable pin. Connect to a PWM signal for variable speed.
13, 14	Connected to motor B's power terminals

As this PCB is being designed for testing and demonstrating the functionality, the L298 input pins will be connected to Boolean output pins of the computer so that the direction mode can be changed. The Input and enable pin voltages require a minimum of 2.3V to be set high. This means that the 3.3V output voltage of the computer will be enough to control the driver. This driver does not have built in fly back diodes; therefore, they need to be added externally.

The logic supply voltage V_{ss} (5V) for the motor driver, and the supply voltage V_s (6-12 V) to the motors require separate power tracks to the rest of the board, and cannot be supplied by the computer. These will be supplied externally for this test board via two larger screw headers. In future implementations, voltage regulators will be required to supply the computer and components with power from the tractors battery.

The BeagleBone Black has 65 GPIO pins across 2 headers attached to P8 and P9 [51]. Not all of these are available for use. The BeagleBone has two virtual capes already on it, one for the eMMC storage and the other for the HDMI output. It is important to not use any of the eMMC pins, as this can corrupt the data and potentially cause circuit damage. To use the pins allocated to the HDMI (LCD), the driver needs to be unloaded. The remaining GPIO pins can be utilised. These pins are shown in Figure 11.

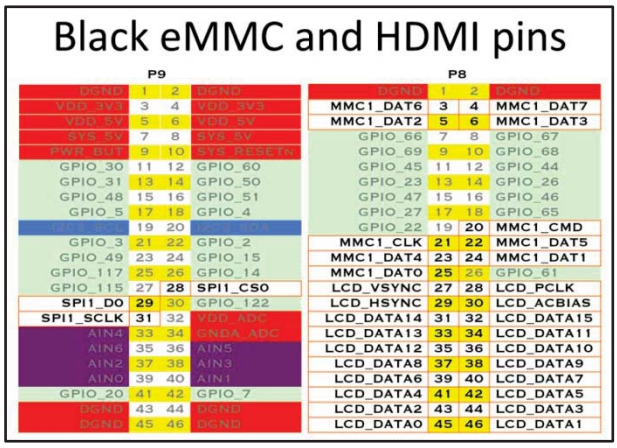


Figure 11: The BeagleBone Black Pin headers (8 & 9) showing the pins used for HDMI (LCD) and the eMMC (MMC1) [51]

Only up to 8 pins can be used to provide PWM signals for motor control or Analogue output. Some of the pins share the same PWM source. These are shown in Figure 12.

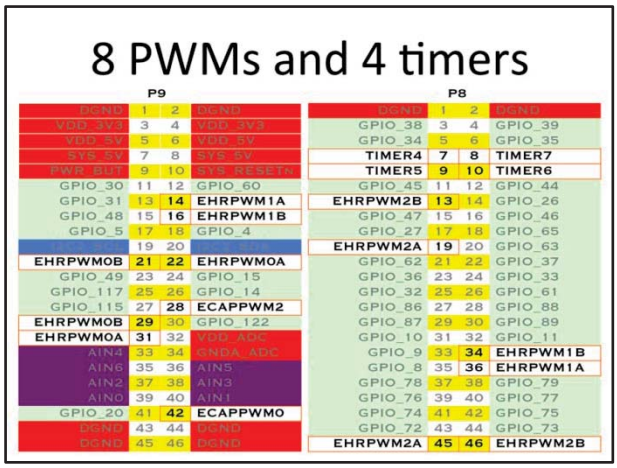


Figure 12: The BeagleBone Black pin headers showing the pins capable of PWM and their source [51]

This information was used to choose convenient pin allocations in the PCB designing process. Four of the PWM pins are unusable as they are allocated to HDMI.

To use a pin in a mode that is not its default, a device tree overlay needs to be created to set it up. A device tree overlay alters the I/O pin MUX registers using the BeagleBone's cape manager. The pin allocation for the eMMC and the HDMI are done using this. The Adafruit_BBIO python library loads device tree overlays to drive pins in modes that are not default (i.e. PWM, analog input, SPI, I2C and UART) when they need to be setup. In future

development of this system, a custom device tree overlay may need to be created to provide more complex monitoring and control functionality [52]. The component overview is shown in Figure 13.

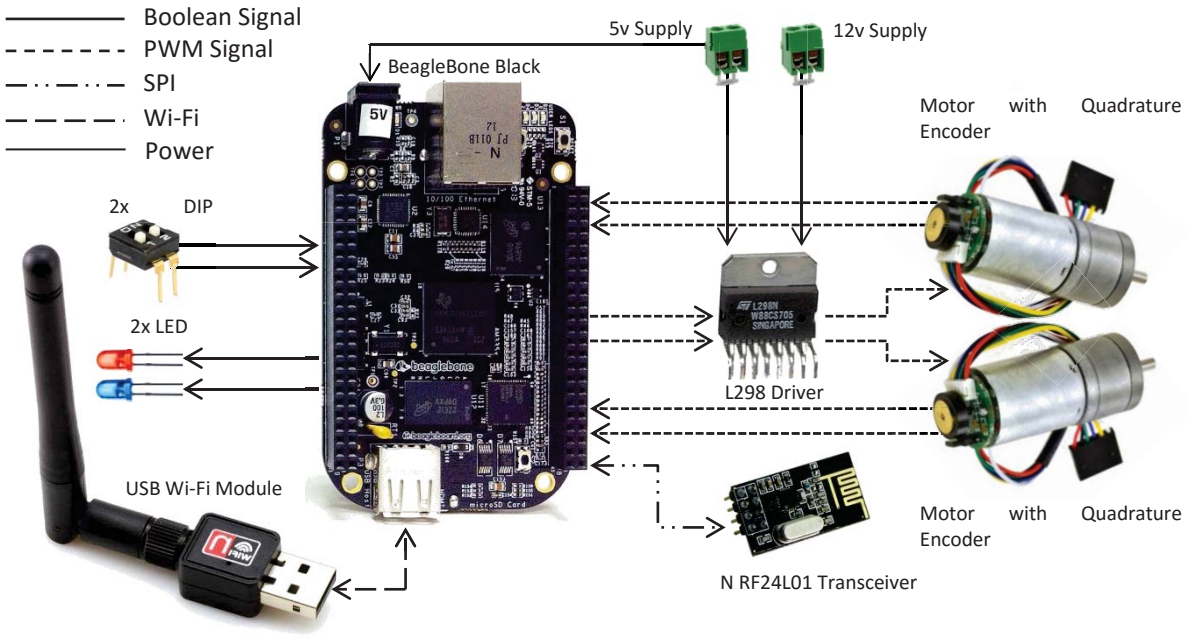


Figure 13: The component overview for the test cape design and the data lines

The test cape PCB developed is shown in Figure 14. The pin assignments for this cape are shown in Appendix 3.

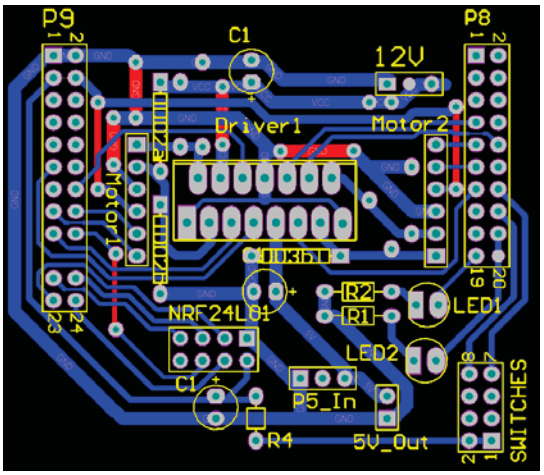


Figure 14: PCB layout for the test cape

Five of the diodes are not shown as the desired footprint was not avaliable and were replaced with pads. Initially, many of the pads were too small, resulting in them lifting of the fiberglass when the hole were drilled. A second PCB was created with larger pads. The constructed PCB is shown in Figure 15.

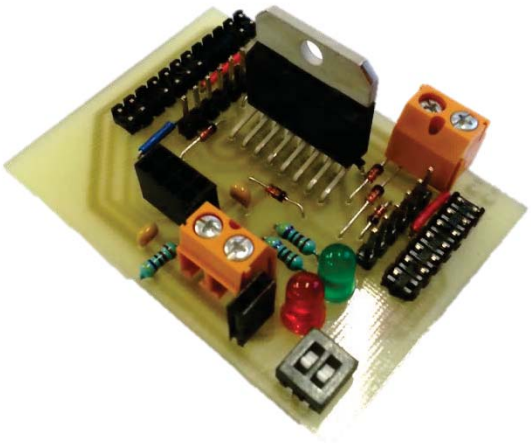


Figure 15: Final construction of the test cape

This PCB successfully demonstrated the functionality of the modules in hardware. Each module can be monitored or controlled from the user interface and the configuration can be changed by editing the database.

3.3.2 Software design

During the initial framework development, two software implementations were developed.

Implementation one

The first implementation was a proof of concept mockup and used four different applications, these were:

1. MySQL Database: This acted as the backbone of the server. All monitored information or control updates would pass through here. This provided a clear record of everything that happened in the system. The database also was used to configure the applications on startup.
2. Pin interface and calculations: This program is in charge of managing the I/O pins and performing the required calculations.
3. Webserver: This maintains a bi-directional communication to the operator's web page using WebSockets. This managed updating the user interface and processed the user requests/updates. For this implementation, "Pywebsockets" were used as it does not have any dependencies, and is a good library for learning to use WebSockets.
4. Web page: This is hosted on the SBC using an Apache webserver and acts as the user interface. This can be accessed by entering the SBC's IP address (10.2.2.2) in a connected web browser. This Webpage is dynamically created on load to have all of the required interface modules based on the systems configuration

The software architecture is illustrated in Figure 16. This shows the communication between the four software applications.

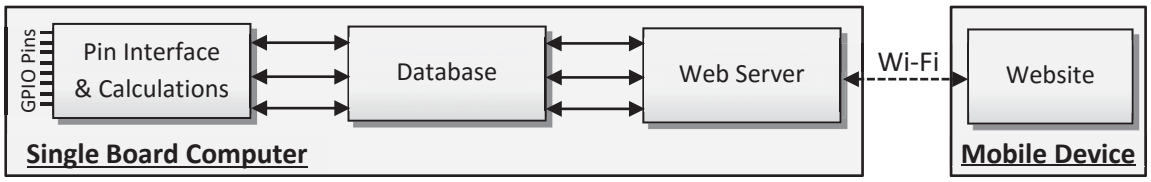


Figure 16: Initial Prototype system software structure

These were split into different programs as they each perform different tasks and need to operate simultaneously. This makes use of the computers multi-tasking operating system.

The task modularity is implemented by populating multi-level dictionaries with the configuration information, then iterating through the keys and values using task handler functions. This means that more tasks can be added to the system by expanding the configuration database.

This implementation only developed the monitoring functionality and is only capable of displaying simple monitored inputs to the user, data logging, and it implements a primitive modular design. It became evident that major changes would need to be made before the control functionality could be implemented.

Monitoring and control information was considered to have a clearly opposing data flow. For simple monitoring and control modules, such as the ones investigated, this was sufficient. However for more complex control, it was important to use information from both the user and the sensors. This is one of the biggest reasons for the revision in stage 2.

This initial prototype system demonstrated: core monitoring functionality, expandable modular design, data logging, and developed a better understanding of how the technologies used can work together. It acted as an initial proof of concept, and was used to determine that, using this technology can provide the functionality required for a diverse monitoring and control system. Many aspects of this system need to be improved for the control functionality to be implemented.

- The database structure needs to be improved to provide configuration and data storage for the control modules.
- Pywebsockets are not appropriate for this system as they are not stable or expandable. They are intended for experimentation, and are not intended for industrial application. This was evident during testing as the system could not maintain a connection with the client for an extended period of time. A different Websocket handler should be used.
- There is unnecessary propagation delay, as well as redundant data transfer and storage, associated with the systems structure. This can only be mitigated through changing the systems software structure and reliance on the database for trafficking information.
- The dictionary structure in this system is quite messy and needs to be improved before implementing control. It will also require revision if the systems software structure is remodelled. As Python dictionaries can be mapped to JavaScript objects, the data structure should be suited to be used in both the main programs and the web page.

Implementation two

The suggested improvements from version one required significant changes to the system structure. The focus of implementation two is to combine the monitoring and control functionality, while addressing the identified issues in the first implementation. The 'web server' and 'pin interface & calculations' programs were combined to remove unnecessary propagation delay as well as redundant data storage and data transfer. Instead of the database being the centre of the system, it is now solely used for initialisation and data logging. This makes the database table layout more flexible and means time isn't being spent on unnecessary database reads. The new software structure can be seen in Figure 17.

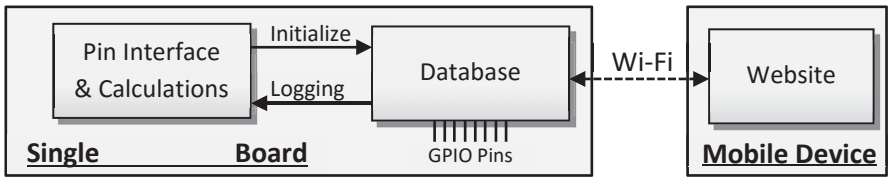


Figure 17: Second prototype software structure

This structure provides better data flow, is more responsive, and makes it simpler to integrate the monitoring and control modules.

The **Main Program** is in charge of managing the I/O pins, performing any required calculations, logging data and communicating with the web page. The following figure shows a simplified flowchart of the system.

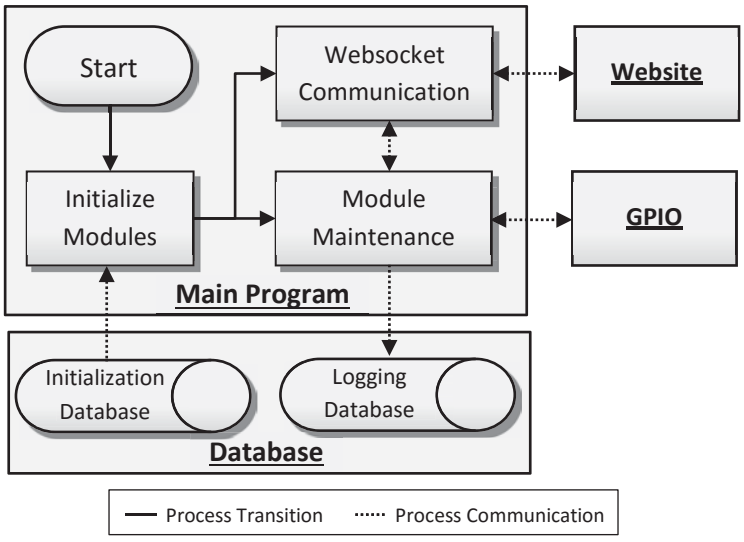


Figure 18: Flowchart of the main programs functionality

On start-up, the main program initialises the modules from the database configuration tables. This sets the pin modes, creates and populates the dictionaries for data storage, creates software pin change interrupts for the inputs, and insures the tables are set up properly for data logging. Once this is complete, two processes are run in parallel, the module maintenance loop and the websocket communication handler.

The module maintenance process is in charge of:

- Regularly monitoring the GPIO inputs
- Logging the inputs to the database
- Sending updates to the webserver
- Maintaining GPIO outputs
- Executing received user controls
- Logging these changes to the database

The websocket communication handler is in charge of:

- Receiving webpage websocket connection requests and establishes the connection.
- Sending webpage initialisation information for either the monitor or the control page.
- Regularly sends monitored data updates to the web page.
- Receives control requests form the webpage and sends them to the module maintenance loop.

As pywebsockets are not suitable for this application, they were replaced with Autobahn.ws for python. Unlike pywebsockets, Autobahn is an object oriented API, this means that it is more expandable, and can be managed easier. To send information to the connected clients, the Autobahn “serverFactory” class needs to be updated. This can be done by receiving a message, or using server ticks. The tick rate was set to two seconds, and is also used to log the monitored data to the database.

This implementation uses a hierarchal structure, utilising multi-tiered dictionaries. The first layer of keys is the data type. This includes: speed, distance, area, Boolean, total distance, total area, speed output and Boolean output. The second layer keys refer to the display names of the modules. The third layer contains the module data names and the fourth layer contains the data. This structure was used as it provides the web page with all the information needed to create and update the data display boxes easily. Four different dictionaries were used for this implemetation.

1. The “initClient” dictionary stores all of the inherent module data like the name, type, description and alert level. This dictionary is primarily used to initialise the webpage.
2. The “updateClient” dictrionary stores all of the monitored data and is sent to the clinet to update the display.
3. The “control” dictionary stores the most recent actuator states. This is also sent to the webpage to initialise the control module displays.
4. The “dispName” dictionary is used to map each module to their pins and its display name.

Pulse Width Modulation (PWM) is used to control the motor speeds. PWM is a constant frequency square wave with a variable pulse width. The duty cycle is the percentage of on time over the signal period. The motor speed is proportional to the duty cycle of the PWM signal.

For this prototype, the **website** consists of two pages: the Control page, and the Monitor page. These have been split by function to make the interface less crowded on small devices. When the web page loads, it tries to establish a websocket connection from the server. If successful, it sends an initialisation request to the server to setup the module display boxes. The server will send an initialisation packet, and then start streaming relevant updates when required. The monitor page has a collapsible settings panel which is used for choosing the display. All unit conversion is done on the client side. Implementing the conversion client side also provides the option of saving the user's unit preference information as cookies in future development. JQuery Mobile was used for the interface design as it provides tools specifically for mobile web development [53]. This allows the page layout to remain consistent across different screen sizes and orientations without distortion. This library also provides many simple, easy to use and customisable widgets. The library JQuery was used to provide an easier way of creating and editing the display units and using the JavaScript objects [54]. This library significantly simplifies programming in JavaScript. When an update is received, the program iterates through each of the elements, changing the values in the list fields. List boxes were used to display the module data. Lists provide a neat and easy method of dynamically creating the display modules. An example of the web based user interface can be seen in the figure below.

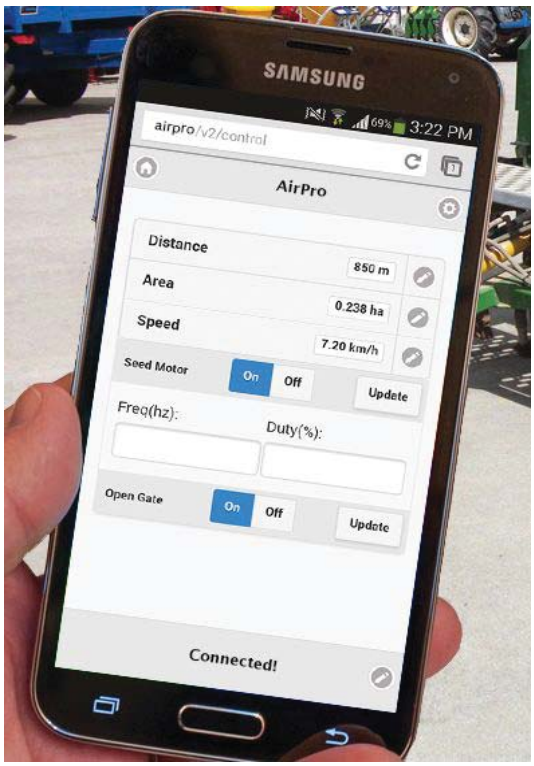


Figure 19: Example of the web based user interface

This system successfully demonstrated the core functionality required in a versatile monitoring and control system. The changes made in the software structure significantly improved the data flow and the operation efficiency. The monitoring and control code was written together, which resulted in a smoother and neater implementation. This means adding code for new monitoring and control module types should be easier. The database structure was also revised, as it needed to incorporate control and there was no need to

read the stored data after initialisation. The new structure incorporates both monitoring and control, as well as providing customisable names for each module display.

This new implementation makes it possible to add modules without restarting the program. This allows for future work into incorporating a configuration wizard into the web interface. It also allows the users to change the names and descriptions of the existing modules.

The main aspects that need to be improved for this system are as follows:

- The user interface needs to be higher contrast for better visibility, and have a Reese Engineering theme to it. This can be done by altering the theme of the widgets and background.
- Little knowledge of good web development practices were known at the beginning of this project. This resulted in the development process being the same as the main program. Although the webpage fulfils the required functionality, the way this is achieved is unconventional and results in the client device performing more tasks than required.
- The speed control modules need to be changed so the user can specify an RPM instead of the frequency and duty cycle. To do this, a feedback control loop is required.
- During testing, it was discovered that some devices could not connect to the WebSockets while using a browser that is known to work. This is a known bug and can be fixed through using long polling AJAX for back compatibility.
- Connecting using a web browser through a mobile device is not a clean and intuitive way of getting to the user interface. If there is a system failure, the operator will not be able to connect with the server and will get frustrated with no response.

3.3.3 Experiments and Results

Tests were performed using the developed hardware and software architecture. The aim of the experiments was to exhibit the expandable modular functionality of the system.

Four modules were created to test the system:

1. Motor duty cycle and frequency control
2. Motor encoder speed monitor
3. Boolean input monitor
4. Boolean output control

Two instances of each module type were created and mapped to the I/O extension cape pins. The system successfully interfaced with the I/O board, providing monitoring and control functionality of the attached sensors and actuators. The motor speeds were set within the web-interface, and the correct motor speed was observed from the encoders. Figure 20 shows the motor driver output when the frequency of the control module is set to 10 Hz and 5 Hz. The user interface could control the LED states and display the DIP switch positions. The system was responsive and all modules worked correctly. This demonstrated the expandability of the modular software architecture.

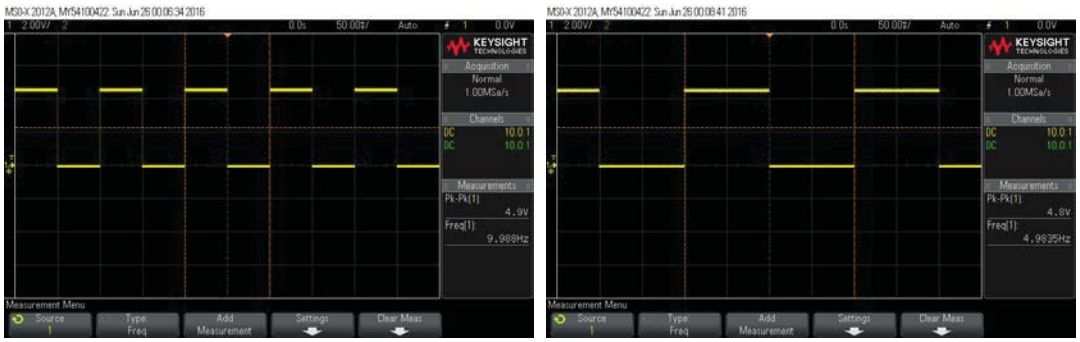


Figure 20: PWM output of the L298 Motor driver. Frequency set to 10 Hz and 5 Hz respectively.

3.3.4 Discussions and Conclusions

Stage one provided the core monitoring and control functionality for future development in integrating this system with agricultural machinery. The system has been tested using the test cape, and has demonstrated the configurability and versatility required for the systems application. Using a web interface to monitor and control system inputs and outputs has been proven successful. The system is responsive, with the control updates being implemented quickly and the monitoring data being displayed regularity. The system keeps a log of all data changes that occur for the operator to download. From this project, Reese Engineering has decided to continue the system towards commercialisation.

All of the system requirements discussed in section 3.1 are achievable with this system, with a majority of them already satisfied. The remaining system requirements need to be addressed more thoroughly in future work.

The hardware framework will need to be revised. The number of available I/O pins is not sufficient to apply the system to the larger machines. Although the software implementation is expandable and versatile, the hardware design is ridged and limited. New circuits will need to be designed for each unique application of the system. This goes against what this project is trying to achieve and will need to be revised in later development.

The software architecture provided a versatile and expandable framework; however, the possible tasks are quite limited by the data flow of the system. The modules do not allow for multiple inputs and outputs within one module type. This restricts the development of complex control systems. Even for simple tasks, such as ground speed dependant seeding rates, the modules require a multiple inputs and outputs within one module type.

Based on these limitations alone, the system architecture should be revised in future development.

There are many other improvements that need to be addressed in future work. These include:

- Add back compatibility to the system if the browser is not compatible with WebSockets. This can be done using a long polling AJAX implementation, or Flash.
- As JSON can be used to encode and decode python dictionaries to and from a string, the dictionaries themselves can be stored in the configuration database instead splitting it into many columns of data.
- Add new modules for more complex monitoring and control hardware that will be required for the machinery. In some cases, this may require a micro controller to manage the task, and communicate back with the single board computer.
- Add a PID control layer module for the speed output. Currently the user needs to specify the frequency and duty cycle of the motor. This is not useful in maintaining a desired speed and is not an intuitive input for the user. The PID control loop will mean that the user speed control can change from frequency and duty cycle to RPM.
- Change the web page implementation to conform to more standard web design practices and to reduce the processing required for the users device. This will include generating the webpage on the server side and then sending it to the client page.
- Provide more user customisable features such as: module organisation; hiding and showing modules; changing themes and module size; machine user manual; and system calibration.
- Save user preference data as cookies so they can be loaded every time the user logs in.
- Now that more is known about the operation of the system, it would be useful to revisit the modular implementation, and investigate using a combination of dictionaries and classes to represent the different modules.
- Some of the PWM pins are not accessible as the pins are allocated to HDMI. The HDMI output is not required, but is very useful as a fall back interface if something goes wrong.
- PhoneGap is a tool that can be used to create mobile apps using web technologies like HTML, CSS and, JavaScript [55]. This tool should be investigated to see if it can be used for this application. Porting the user interface to an application will make it simpler and more intuitive for the operator to use, provide useful error messages or help if the system goes down and can store a user manual client side.
- The sensors and actuators required for the machinery this system will be applied to need to be selected. An I/O interface cape needs to be designed for each machine to connect to the system.
- The sensor and actuator mounts for the machinery need to be designed and installed.
- The actuators and I/O interface needs to be designed with safe failure mechanisms. If the system fails, the machinery should still be useable.

4.0 Proposed Framework

This stage focuses on revising the system framework to provide coordinated, autonomous control of the new seed drill Reese Engineering are developing. A working system prototype will be created fitted and tested on a seeder machine. This stage will be a major step towards product commercialisation. Using the knowledge and techniques developed in stage one, a more holistic view could be taken in the development of stage two. To address the improvements identified in section 3.3.4, a revision of both software and hardware was undertaken. It became apparent that for these to be implemented, major changes needed to be made.

The main changes made include: a complete revision of the hardware architecture; redesigning the modular software to handle multiple data streams and complex coordinated control; abstracting the module handlers from the main code body to simplify new module creation; and redesign the website to provide module creation and automatic module user interface creation.

Stage one's focus was primarily on developing the software framework, as such, the hardware was not developed. The system was designed to allow development to satisfy all of the system requirements. By the end of the initial framework, many of these requirements were not implemented. These aspects include: the uninterruptible power supply (UPS); sensor and actuator selection; the driver circuitry; enclosures; connectors; and environmental protection.

4.1 Revised System Requirements

The results and conclusions made for the initial framework have led to a complete revision of the system requirements, as well as the software and hardware framework.

The initial system requirements outlined in section 3.1 remain largely unchanged, with the exception of the following additions:

- The tractor alternator will act as the primary power supply for the system. The system must be protected against potentially harmful transient noise.
    - Power filtering and protection circuits will be developed.
- The system needs to shutdown safely once the power has been disconnected.
    - An uninterruptible power supply will be included in the system design.
- The software architecture needs to be capable of complex coordinated autonomous control. This is important for precision agriculture functionality.
    - This will influence the redesign of the modular software structure.
- The system will be configured to provide seed rate control of the new 8128 series seed drill Reese Engineering are developing.

4.1.1 8128 Seed Drill System Requirements

A seed drill is a machine that is towed by a tractor to automatically sow seed. It cuts through the earth to create a series of straight, constant depth furrows. A hopper is used to house a reservoir of seed. The seed is singulated and dropped into the furrows. There are many different methods of seed singulation, for example: rotating perforated barrel, a series of

cups and rotating sponge disks. Some machines use harrows to push the soil back over the seed. Seed drills can apply a wide range of different products, including both seed and fertilizer.

The 8128 Seed Drill is Reese Engineering's first electrically driven seed drill. It was developed in parallel with the electronic monitoring and control system. It is an air seeder with three hoppers, two large and one small. This seed drill will apply seed and fertiliser. The seed is collected at the bottom using a rotating disk which is driven by an electric motor. The application rate is proportional to the motor speed. For the two large hoppers, the seed is then blown up to a distribution head, which randomly disperses the seed into tubes which dispense the seed into the furrows. The fan speed is controlled hydraulically.

The two back wheels are attached to hydraulic rams, to lift and drop the machine for transport and seeding. The prototype CAD model can be seen below, in Figure 21.

Figure 21: 8128 Seed Drill CAD Model

The constructed prototype can be seen in Figure 22.

Figure 22: Constructed 8128 Seed Drill

This machine requires ground speed dependant seed application control for each of the three hoppers. To achieve this, the system will need: 3 motor drivers; 3 motor speed encoders; 1 ground speed encoder; and 1 switch to indicate when the machine is seeding. Additional required sensors include: 3 bin level sensors and 2 fan speed encoders.

The ground speed dependant seed application rate control will require calibration for each hopper to determine the relationship between motor speed and application rate.

4.2 System Architecture Changes

It was initially intended to use the SBC's GPIO pins to interface with the transducers mounted on the machinery. This approach has many significant flaws and limitations that can be avoided using a different architecture. The proposed software architecture provides expandable and customizable modularity. This must be reflected in the hardware architecture developed. Developing dedicated hardware for each implementation goes against the systems fundamental design principle of versatility.

The new hardware structure has been influenced by the modular approach taken for the software development. It consists of a serial network, connecting multiple smart transducer blocks to the SBC in a "master-slave" relationship.

There are many possible technologies to provide this interface. In future development, both wired and wireless methods will be implemented. A data management layer will be incorporated for each communication technology, increasing the systems compatibility. As the development is focusing on providing a generic monitoring and control system, the initial communication interface implemented should be universal to some degree. As this is a proof of concept, only one communication interface will be incorporated. The ISOBUS communication protocol is only used for farm equipment and tractors. Using this will limit the possible application of the system. The RS-485 standard will be used to handle data transmission.

Each smart transducer block can have multiple sensors and/or actuators connected. They will all have a microcontroller for preprocessing the data, maintaining the actuators and communicating back to the hub. They will also include any essential components required for the sensors and actuators.

This implementation allows for expandable hardware, as it is no longer limited to what the SBC can provide. Much of the computational processing can be offloaded to the blocks microcontroller, which frees up the single board computers processing and allows for more complex control.

Serial Communication – RS-485

The RS-485 (TIA-485) standard is used, providing an inexpensive serial network with multi-drop communication between the master and slave devices. It uses differential balanced signals over a twisted pair to reduce the effect of noise on the line, allowing it to reach distances up to 1200 m. It is a widely used standard for serial communication in industry. Each device will require an RS485 transceiver chip to communicate on the network. The ends of the wires require termination to prevent signal reflection. The standard offers data transmission speeds of 100 kbit/s at 1200 m and 35 Mbit/s up to 35 m. The transmission speed is limited by the wire length using the general rule of thumb that the speed (bits/s) multiplied by the wire length (m) should not exceed 10^8. The standard allows for up to 32 devices [56].

The half-duplex system will be used, minimizing the number of wires required. This configuration uses the same differential line to transmit and receive data. This means that only one device can transmit at a time.

Uninterruptible Power Supply (UPS)

In this application, the power will be sourced from the tractors alternator. This can be quite noisy and will cut off abruptly when the tractor is turned off. An Uninterruptible Power Supply (UPS) is required to provide regulated and filtered power to the single board computer and the transducer blocks. The UPS will delay the shutdown to give the system time to power down safely. For more information on the UPS development, see the 'Uninterruptible Power Supply (UPS) Design' part of section 4.3.2.

Single board Computer (SBC)

Changing the interface between the SBC and the transducers from parallel to serial has many effects on the systems design considerations. For one, the single board computer chosen in stage one was selected solely for its superior number of GPIO pins. This is no longer a requirement, and as the only code specific to implementing the system on the BBB is the GPIO library used, the new system development can easily be ported to a more suitable device. After reviewing the initial SBC options, the Raspberry Pi 2B was chosen to continue development. This choice was largely due to the low cost, high level of community support and resource availability. As the new system version is no longer dependent on device specific libraries, it has the potential to be simply ported to a wide variety of devices.

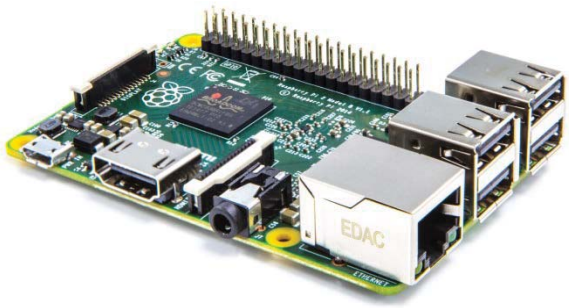


Figure 23: Raspberry Pi 2B [48]

Master Slave Structure

The master/slave model of communication will be used to regulate the data transfer between the SBC and the attached transducer blocks over the serial network. This is important for serial communication because only one device can transmit at any one time. The master's job is to regulate the communication to ensure that there is no cross talk. The master-slave hierarchy can be seen in Figure 24.

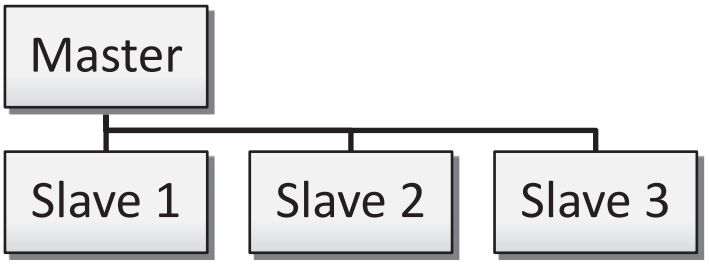


Figure 24: Master-Slave hierarchy used in the system architecture

The SBC will act as the master, with the smart transducers being the slave devices. The master has strict control over the serial networks communication. The Master dictates every message that passes through the network. This ensures that only one device can communicate at any one time. For this model to work with multiple devices on the network, each slave requires a unique ID number. This is used to identify which device the message is for, and which device the reply came from.

To ensure the master knows that the slave received the message, the slave will always reply. This reply can be the requested data, or simply an acknowledgment. If the reply is not received within a given time, the initial message is resent. If this continues, the master marks the device as unreachable and notifies the user of the error.

If an error is detected in the received message, the device will request that it be resent. If this persists, the master will mark the device as unreachable, and notifies the user of the error.

Unless something goes wrong and needs attending to, the slaves should not be unreachable by the master. If this does occur, the error is logged and the user is notified.

When a device is sending a message, it is in write mode and cannot receive data. This is useful to stop echoing and means that the SBC does not need an ID, as all messages from the slaves are intended for the master. The packets device ID will either be of the destination slave, or the transmitting slave.

New Hardware Structure

Figure 25 shows the new systems hardware structure.

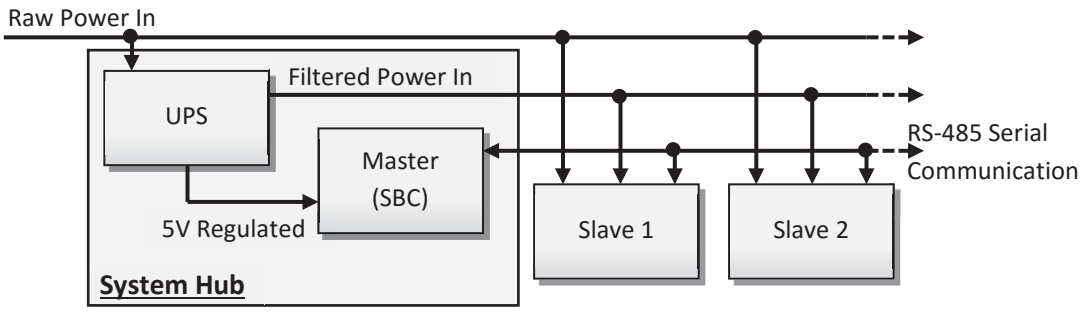


Figure 25: System hardware structure.

The 'Filtered Power In' line is primarily for powering the SBC, the transducer block microcontroller and low current sensors. More power intensive sensors and actuators will draw from the raw power input and may require additional power filters. This isolates the supply to the processors from power intensive components.

This new hardware design has many significant advantages, including:

- The circuit design for different system configurations is now significantly simpler, as each smart transducer will have its own standard circuit design and is no longer required to be built into the main hub. The previous option would require a different hub circuit design for each unique implementation. Now transducers can be added easily without any hardware redesign.
- Using RS-485, up to 32 smart transducers can be added to the serial bus. Each smart transducer can handle any number of tasks utilising its GPIO pins. This is a vast improvement on the number of useable pins available on the BeagleBone Black computer.
- Using Smart transducers offloads a lot of the processing onto the microcontrollers from the computer and ensures that the system has time to process all of the information. This makes the programming simpler, cleaner and results in less redundant. This is important for future development.
- The serial interface makes the wiring and installation significantly simpler, as only one cable is required to reach all of the sensors instead of one for each.
- The hub housing design can be generic, and does not have to be different for each implementation.
- A wide range of smart transducers can be developed to perform any task. A system can be built by attaching the required transducer blocks and importing the required module library. This implementation reduces development time of each configuration significantly and makes the entire system more versatile and simpler to use.
- Abstracting the transducer processing and having a well-defined messaging protocol allows for easy development and integration of more complex transducers. This could lead to future development in things like: tramline control, soil monitoring and more advanced seeding control.

- This structure allows the choice of SBC to be more flexible as the GPIO pin layout is no longer a design factor. This means that there is the potential to change the SBC in the future, in the event of discontinuing the intended mode.

New Software Architecture

The fundamental system functionality has been redefined. The functionality of the system has been broken into four key system blocks: Slave Tasks, Modules, Module User Interfaces (UI), and the Database. Each of these blocks have been developed using the same fundamental model to vastly increase the number of possible applications. Each System Block can have any number data streams connecting to the adjacent components. A data stream refers to any incoming or outgoing data of the block. Figure 26 illustrates the fundamental model. Each arrow represents a data stream.

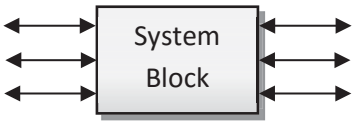


Figure 26: Illustrates the fundamental model used to develop each system block

The system structure is illustrated in Figure 27.

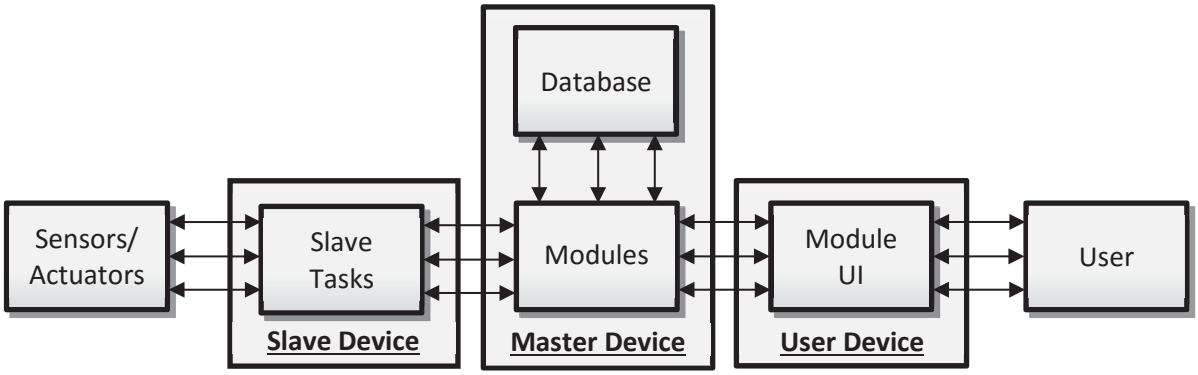


Figure 27: Basic system architecture

There can be any number of slave tasks and module instances. Each module will have a corresponding module UI. Modules can access multiple slave tasks from any slave device. Tasks on the same slave device can interact with each other directly. Figure 28 shows an example of this architecture, highlighting the possible data flow. The bold arrows represent any number of data streams.

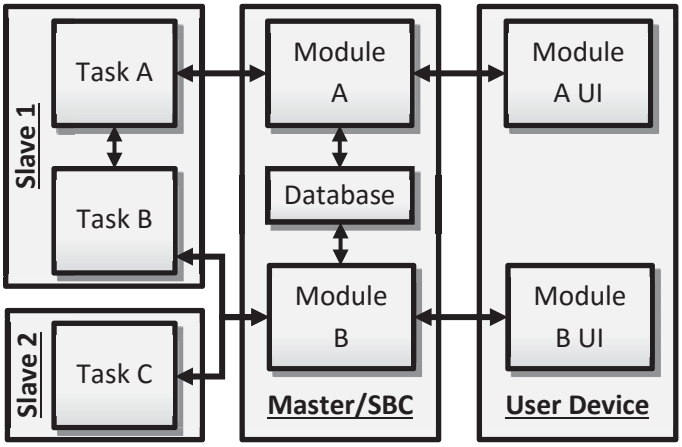


Figure 28: Example of the possible data flow with this system architecture

Slave Tasks handle all direct interaction with the systems sensors and actuators. Each task is capable of bidirectional communication with the master using the developed messaging protocol. They can be designed to carry out any task capable of the slave hardware. Examples of possible slave tasks include: Boolean reads/writes, encoder handler and PID motor control. The slave task will: setup the I/O pins, carry out processing, and communicate with the master.

A **Module**'s primary function is to manage slave tasks using information from slaves, the database and the user. The module functionality is carried out by the module handler.

The **Module UI** displays relevant data and provides user input fields for module configuration and control.

The **Database** is used for both recurring and infrequent data storage. Data storage is managed by the module handler.

Data stream management is critical for this architecture to work effectively. This is the largest consideration when developing the data streaming protocol between each of the system blocks. The data streaming protocol will be different between each system block type due to the different data transfer mediums.

- Between slave tasks, data streams can be accessed directly.
- Between slave tasks and the modules, a packet based transfer protocol will be used transmission over the RS485 serial network.
- Between the modules and the module UI's, entire dictionaries will be transmitted using a websocket protocol. Python dictionaries can easily be encoded and decoded as JavaScript objects.

4.3 Hardware Architecture

The hardware architecture is outlined in section 4.2.

4.3.1 Serial Communication

The slave microcontrollers will use the EXAR SP483ECP-L RS485 Transceiver chip, which interfaces between RS485 and UART. This comes in an 8 pin DIP package.

By default, the raspberry pi UART is used to provide access to the terminal. This may be a useful tool in the future for onsite debugging. Because of this, a USB to RS485 dongle was used. This can be seen in Figure 29.

Figure 29: USB to RS-485 dongle

4.3.1 Enclosure and Environmental Considerations

This product will be installed on the farm machinery and operated outdoors under various weather conditions. The system needs to be capable of operating under the extreme conditions the machine may be subjected to. When designing the system hardware the following factors need to be considered:

- High and low weather temperatures
- Thermal shock due to temperature fluctuations of the electronic components
- High humidity
- Dusty and muddy conditions
- Strong rain and water blasting (cleaning)
- High and low frequency vibrations from machine operation and terrain roughness
- Mechanical shock

The System will be mounted on the machinery out of direct sunlight and rain, under a protective cover.

Humidity, water, dust and dirt are a major threat to the electronics. To prevent exposure to these elements, air tight enclosures and connectors will be used. This cuts off all airflow to the system, which can cause cooling issues. The SBC, motor drivers and voltage regulators are a major source of heat. This heat needs to be dissipated to prevent component temperatures exceeding the rated operational limits. As the electronics will be enclosed, there is no airflow across them. Introducing airflow to help cool the system is not viable as it will expose the electronics to dust and moisture.

The Ingress Protection (IP) Rating is used to rate the degree of protection a product provides against water and dust. This rating is defined by IEC standard 60529 [57]. The first digit of the IP rating corresponds to the products protection against solid objects. The second digit corresponds to the protection rating against liquid. Based on this standard, the lowest rating suitable for this application is IP65. This gives total protection against dust (6) and no harmful effect of water splash from all directions (5). Higher liquid protection ratings would be preferred. IP67 indicates; total protection against dust and water immersion to a depth of 1m for 30 min. This specification will be used when selecting the enclosure and connectors for the system.

The SBC processor, motor drivers and voltage regulators are the major sources of heat in this system. The heat dissipation of these components is reduced significantly by being in a sealed enclosure. In a sealed enclosure, heat generated by the internal components must be dissipated through the external walls. There is no airflow within the enclosure; therefore heat transfer between the surface of the components and the surface of the walls will be via natural convection and radiation. Heat transfer through the wall will be via conduction and heat transfer from the outer surface of the enclosure and the environment will be though convection, radiation and conduction.

Passive heat sinks could be added to the components inside of the enclosure to increase the surface area for heat transfer; however this will only have a small effect, as there will be no air flow across it. Liquid cooling is too costly for this system and will not be very effective.

The enclosure will be mounted away from direct sun light. The enclosure itself can be designed as the primary heatsink. A material with a high thermal conductivity, such as steal or aluminium can be used as the base of the enclosure. The base will securely mount the components, and raised features will contact all major heat sources. Thermal paste will be used to improve thermal conductivity between the base and the components. This provides a larger heat sink, with further heat dissipation to either the air, or machine chassis. A rubber seal or gasket will be used to make the enclosure water tight. Figure 30 illustrates how this might look.

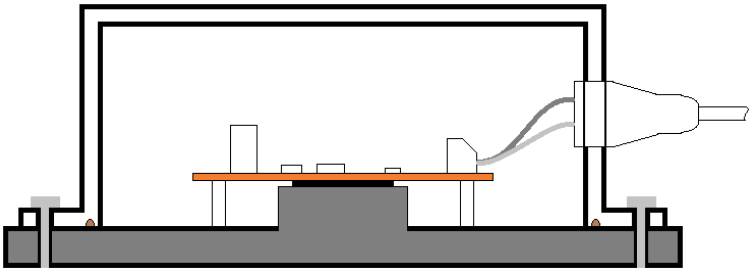


Figure 30: Illustration of custom enclosure, with built in heat sink

Potting compound or conformal coating is widely used in industry to protect electronics from mechanical shock, thermal shock, vibration, moisture and corrosive agents. Potting compound fully encapsulates the components. A low glass transition temperature potting compound will be required for PCB's with surface mount electronics to prevent potential damage to the solder bonds as it hardens and shrinks. The choice of potting compound for the raspberry pi and motor drivers need to be influenced by the heat transfer, thermal range and thermal expansion of components. The Raspberry Pi foundation is unsure whether potting the Raspberry Pi is viable [58]. Potting compound or conformal coating will be considered as a final measure of protection.

Vibration is a major source of failure in many systems, causing damage through fatigue. It can affect both the electronics and the housing itself. Reducing the vibration felt by the system is important. The components within the enclosure will be secured firmly and spring washers will be used to prevent bolts loosening. All vibration dampening will be handled using vibration resistant mounts.

There are many types of compressible mounts that can absorb vibration to some degree. Spring washers will be used with all nut and bolt joints. They are used to prevent loosening due to vibration by applying pressure against the nut, creating more friction and resistance to rotation on the bolt. Spring washers continue to secure the bolt after being loosened slightly. Foam padding will be used between the enclosure and the machine for the initial prototype. The system will be secured using zip ties.

As the system hardware design is likely to change from the initial implementation and components edited during debugging, it is not reasonable to develop the purpose built enclosure, or use potting as part of the initial functional prototype. Off the shelf Polycarbonate and ABS IP65 enclosures will be used to enable testing the system in the field. During Field testing, sensors will be added to measure the vibration and temperature in the

system. This information will be useful for further development of the system enclosure. The enclosure will be chosen based on the electronics dimensions and will be covered in the hardware construction sections.

The connectors used are IP67 7 pin male and female solder connectors and gland sockets. Each of the 7 pins is rated to 10A. The connectors were sourced through Captron Electronics Ltd., who is an established supplier of Massey University. Ideally, the different connector types will have unique connectors, to prevent plugging leads into the incorrect socket. This protects the electronics and makes it simpler to assemble. High current wires will use gland sockets instead of the connectors. The connectors are one of the major expenses of this system design. These connectors can be seen in Figure 31.

Figure 31: The connectors used. Cable gland, Male box connector (PWE-07PMMS-SC7001) and Female cable connector (PWE-07BFFA-SL7001).

For testing the system's functionality, the first functional prototype will be built without potting or a custom heat sink. Future development needs to be undertaken to create a reliable system enclosure design and provide adequate system heat dissipation. Further investigation needs to be undertaken into:

- The viability of potting.
- The vibration intensity and frequencies felt by the system during operation over various terrain roughness.
- The effectiveness of the vibration damping methods.
- The structural integrity of the system.
- The temperatures reached within the enclosure during operation, and the effectiveness of different methods of dissipation.

4.3.2 MASTER Hardware Design

The master hardware consists of: The Raspberry Pi SBC, a custom made UPS, two connectors, an enclosure and a mounting plate.

Uninterruptible Power Supply (UPS) Design

An uninterruptible power supply provides emergency power when the main power source is disconnected. These are typically used to ensure that a computer can survive short power outages and if needed allows it to shut down safely, preventing damage or loss of information. The UPS is a critical component of this system.

The tractor's alternator is the primary power source for this system. This can be quite noisy, with spikes over 100 V when revved and drops from sudden loading. During steady conditions, the average alternator voltage ranges between 14.2 V and 14.5 V. The alternators in modern tractors can easily supply the power requirements of this system.

The alternator voltage is effected by: engine cranking; electrical loads, such as electric motors and head lights; jump starting the engine; and the engine speed [59]. Figure 32 shows a representation of typical automotive transients. Table 7 describes the causes of transient noise, their regularity and the length.

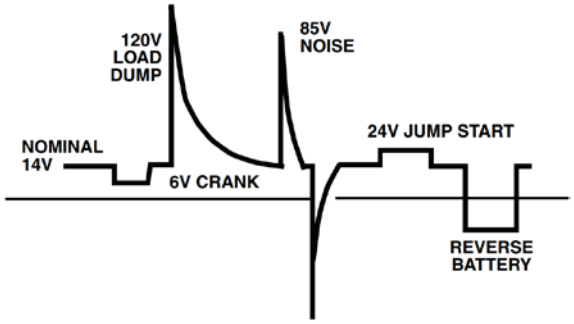


Figure 32: Typical automotive transients [59]

Table 7: Typical automotive transients [59]

LENGTH OF TRANSIENT	CAUSE	ENERGY CAPABILITY / VOLTAGE AMPLITUDE	FREQUENCY OF OCCURRENCE
Steady State	Failed voltage regulator	• / +18V	Infrequent
5 Minutes	Jump Start with 24V battery Infrequent	● / +/-24V	Infrequent
200ms to 400ms	Load dump; disconnection of battery while at high charging	<10J / <125V	Infrequent
<320 us	Inductive-load switching transient	<1J / 300V to +80V	Often
200ms	Alternator field decay	<1J / -100V to -40V	Each Turn-Off
90ms	Ignition pulse, battery disconnected	<0.5J / <75V	<500Hz Several Times in Vehicle Life
1ms	Mutual coupling in harness	<1J / <200V	Often
15us	Ignition pulse, normal	<0.001J / 3V	<500Hz Continuous
Burst	Accessory noise	<1.5V	50Hz to 10kHz
Burst	Transceiver feedback	~ 20mV	R.F.
<50ns	ESD	<10mJ / 15kV	Infrequent/ Random

The UPS requirements are quite different from an off the shelf device. This UPS needs to:

- Operate off of the tractor power supply from the alternator.
- Filter the main tractor supply.
- Provide a steady, uninterrupted 5 V supply to the SBC.
- Signal the SBC when the main power is disconnected.
- Continue to power the SBC as it shuts down safely. After which, cut the power to the SBC.
- Provide a filtered supply (~12 V) to the sensor network, shutting it down when the SBC is powered off.
- Allow the SBC to keep the UPS operating if required.

The Raspberry Pi does not have any buttons or pins to trigger a restart or shutdown this can only be done in software. After shutdown, the Raspberry pi needs the power supply to be disconnected completely before it can boot again. The UPS will turn the SBC on when the tractor turns on, and turn it off safely when the tractor is turned off. No commercial UPS system could be found that achieves all of the requirements.

The Raspberry Pi does not have a built in real time clock. It normally relies on an internet connection to keep the time up to date. As this system needs to operate with the correct time, without internet access, a real time clock needs to be connected. This will be built into the UPS circuit board.

This UPS consists of 6 parts: the power filter; the backup battery, battery charging circuit, a timing circuit, main power voltage sense, 5V regulator and a shutdown override. The UPS functional diagram can be seen in Figure 33.

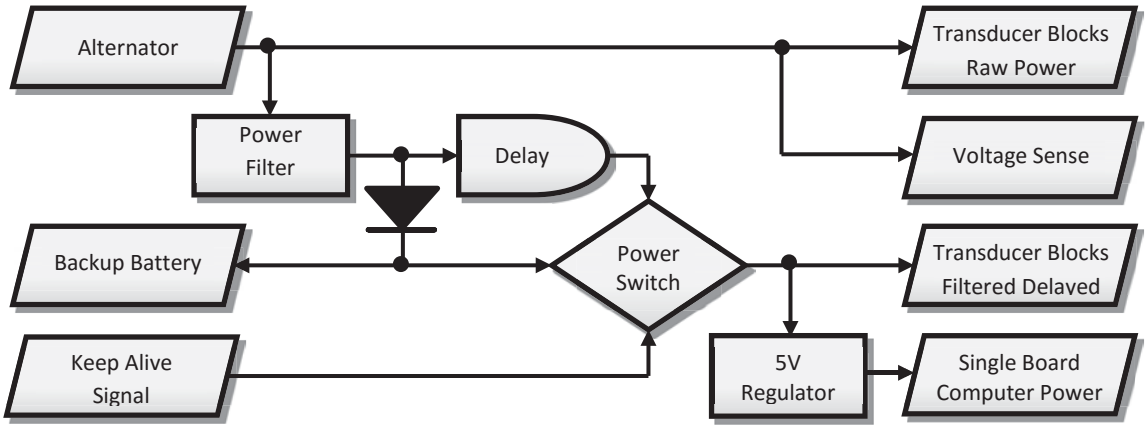


Figure 33: UPS Functional diagram

The **Power Filter** is required because the main power source is high in transient noise, with voltage spikes potentially reaching 100 V when revved. This filter design is commonly used in the automotive industry to suppress the transient noise highlighted in Figure 32 and Table 7. The power filter consists of:

- A transient-voltage-suppression (TVS) Diode for suppressing voltage spikes.
- An LC low pass filter to reduce the effects of high frequency noise.
- A fuse to prevent damage to the system.

TVS Diodes are commonly used for fast voltage clamping. They suppress all overvoltage above their breakdown voltage, with a high reaction speed [60]. Although their performance does not degrade, it is inefficient to have it clamping constantly. An 18 V reverse stand-off voltage will ensure the subsequent components are protected, while reducing the clamping rates. The TVS Diode used is the SMCJ18A.

The LC low pass filter is a DC line filter designed to attenuate transient noise. A large capacitance is required to act as a reservoir for voltage drops. The inductor effectively opposed changes in voltage. A 1000 µF electrolytic capacitor, rated to 50 V was used. Using a 150 uH inductor, the low pass LC filter will have a cut-off frequency of 410.94 Hz (Equation (1)). This inductor used (2314-RC) is rated at 5 A.

$$f = \frac{1}{2\pi\sqrt{LC}} = \frac{1}{2\pi\sqrt{150 * 10^{-6} * 1000 * 10^{-6}}} = 410.94Hz \quad (1)$$

A resettable polyfuse will also be used to protect the system, the value of which will be chosen once all components are chosen. The circuit schematic for the power filter can be seen in Figure 34.

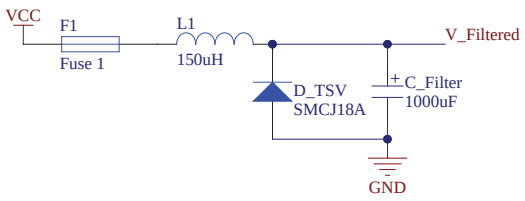


Figure 34: Power filter circuit design

The **backup battery** will be a sealed lead acid (SLA) battery. These batteries are often used in agriculture as they have a high capacity and a high discharge current, as well as being rugged and reliable. As this system will be mounted on the farm machinery, the battery weight is not important. The battery chosen is a small 12 V 1.2 Ah SLA (ID: 537-5444), with an initial current of less than 360mA. When the main supply is disconnected, the SLA transitions from being trickle charged to powering the system. The initial current is not sufficient to reliably power the Raspberry Pi 2 Model B, which can draw up to 1.2 A [61]. However, the large capacitor in the power filter ensures that the source power is not instantaneously cut off, allowing time for the SLA to provide higher current by prolonging the transition. The chosen SLA battery is shown in Figure 35.

Figure 35: Sealed Lead Acid Battery (ID: 537-5444)

Farm machinery usage is very seasonal. The battery needs to be able to survive being on standby for most of the year. The chosen SLA has a 5 year standby life, making it ideal for this application. For safe shutdown, the backup battery needs to be able to supply the system for up to 30 seconds. This battery can easily achieve this. With a capacity of 1.2Ah, the SBC can be powered solely from the battery for over an hour. Being able to turn the system on without turning the tractor on is a useful feature for debugging, setup and database downloads.

The battery's charge is maintained through trickle charging from the filtered supply with the **battery charging circuit**. The backup battery will be in series with a power diode and the filtered power. This diode prevents the battery from supplying the tractor when the power is low or off. This means that the backup battery voltage will be maintained at the supply voltage, minus the voltage across the diode (Equation (2) and (3)).

$$V_{Bat} = V_{alt} - V_{Diode} \tag{2}$$

$$V_{Bat} = 14.4 - 0.6 = 13.8\,V \tag{3}$$

The only time the battery is being drawn from is when the voltage supplied to the battery drops below the battery voltage. The life span of a trickle charged SLA battery can be very long. The ideal charging voltage for maximum service life is between 2.25 and 2.3 volts/cell. This is 13.5 to 13.8 V for a 12 V SLA battery. Charging at a higher voltage will charge the battery faster but it can lead to damage. Charging a 12 V SLA with below 14.7 V is acceptable, with low risk of over saturation which leads to significant damage. The charging voltage in normal conditions is well within the ideal range [62].

A Resistor Capacitor (RC) **Timing Circuit** is used to delay the systems shutdown. The power to the SBC and the transducer blocks is shut off once the timing circuit voltage drops below a voltage reference. The voltage reference is set at 0.73 V by using two diodes in series. This is compared to the discharge curve of the RC circuit using an LM311 comparator. Once the voltage drops below the reference voltage, the comparator output will be pulled low. This will control the power MOSFET which will switch the power off to the system.

A diode is included in series with the RC circuit to prevent the capacitor from discharging through the main circuit when the source power fails. A small resistor is also placed in series to limit the current through the diode during the charging of the capacitor. The resistor should be sufficiently small as to: not impact the RC time constant; ensure fast capacitor

charging; and should limit the current to below 450 mA, the maximum current for the 1N4148 Diode. A 47 Ω resistor was selected. Assuming that the main voltage input does not exceed 15 V, the current through the diode and resistor could reach 320 mA (Equation (4)).

$$I = \frac{V}{R} = \frac{15}{47} \approx 320\ mA \tag{4}$$

Another resistor has been placed in series with the reference diodes to limit the current. As this part of the circuit is always powered by the battery, a larger resistor of 10 kΩ was selected to limit the current draw while the system is off.

The voltage across the capacitor is defined by the following equation:

$$V_C = V_S * e^{-\frac{t}{RC}} \tag{5}$$

Where V_C is the voltage across the capacitor, V_S is the supply voltage, t is the time since the supply was removed and RC is the timing constant. This equation can be solved for the timing constant (RC) and the system shutdown time (t):

$$RC = -\frac{t}{\ln\left(\frac{V_C}{V_S}\right)} \tag{6}$$

$$t = -RC * \ln\left(\frac{V_C}{V_S}\right) \tag{7}$$

These equations are used to determine the timing constant needed to give the desired shutdown time. The system will shut down when the voltage across the capacitor falls below the reference voltage of 0.73 V.

For a shutdown time of 25 seconds, assuming the source voltage is approximately 13.8 V, a time constant of 8.5 is required. Using this, an 800 kΩ resistor and a 10 μF capacitor were chosen, giving a time constant of 8, resulting in a shutdown time of approximately 23.5 seconds.

The comparator and the voltage reference are the only components powered by the backup battery during a full shutdown. The rated power consumption of the LM311 comparator is 135 mW, assuming a 12 V supply, this gives a current draw estimate of 0.01125 A. The current through the current limiting resistor should be 0.0012 A. The 1.2 Ah SLA battery will last approximately 96 hours (Equation (8))

$$\frac{1.2\ A\ h}{0.01125\ A + 0.0012\ A} = 96.38\ h \tag{8}$$

To protect the battery from depletion, a mechanical switch will need to be included to electrically isolate the battery when the system is not being used. The circuit schematic for the delayed system shutdown can be seen in Figure 36.

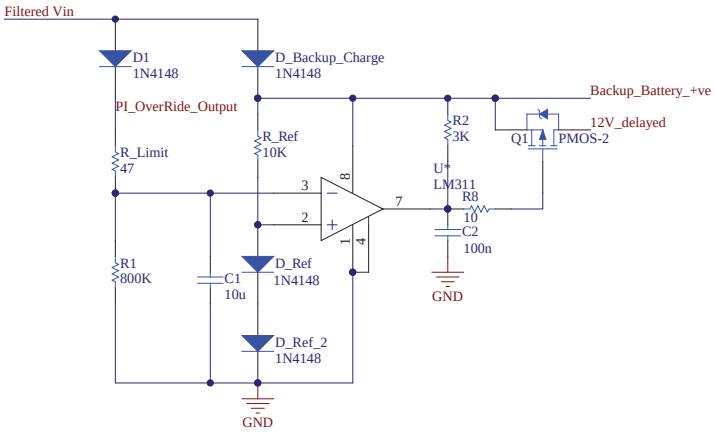


Figure 36: Circuit Schematic to provide delayed shutdown

The **mains power sense** circuit uses a reverse biased 1.8 V Zener Diode and a current limiting resistor in series with the filtered supply voltage. The Raspberry Pi GPIO pins are rated to 3.3 V, with the minimum high level voltage being 1.3 V, and the maximum low voltage level being 0.8 V [63]. When the supply voltage is high, the voltage drop across the Zener diode will be 1.8 V. This can be safely connected directly to a pin on the raspberry pi. A program will be running on start-up which monitors this pin. See section 4.4.3. The mains power sense circuit schematic can be seen in Figure 37.

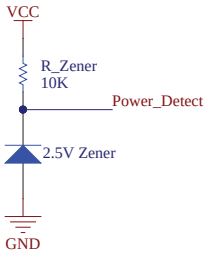


Figure 37: Power sense circuit schematic

The **5V regulator** was initially intended to be a universal battery eliminator circuit (UBEC), which is a switch mode DC step down regulator package commonly used in radio control projects. This was replaced with a TSR 1-2450 step down switching regulator because it could be mounted on the UPS circuit board, providing a more compact and ridged solution. The TSR 1-2450 has a high efficiency of up to 96%, does not require a heat sink and requires no external components to operate. The TSR 1-2450 has a wide input range of 6.5-36 V DC and an output 5 V DC. The maximum output current is 1.0 A. The TSR 1-2450 can be seen in Figure 38.

Figure 38: TSR 1-2450 switch mode regular

The **shutdown override** function is implemented by using the backup battery to charge the timing circuit. This can be triggered using either a switch or a signal from the SBC. As the Raspberry pi GPIO operates at a 3.3 V logic level, a high side switching circuit is needed to switch the backup battery to the RC circuit. The shutdown override circuit can be seen in Figure 39. The PNP transistor Q2 acts as the main switch connecting the backup battery to charge the timing circuit. An additional NPN transistor (Q1) is used to isolate the I/O pin from higher voltage on the base of the PNP transistor.

When the PI_Override signal is low, Q1 will not conduct, and hence no current flows through the Q2. In this state, the output will be off.

When the PI_Override signal is High, the Q1 transistor will conduct, pulling the collector towards ground. This switches the Q2 transistor on, connecting the backup battery to the RC timing circuit.

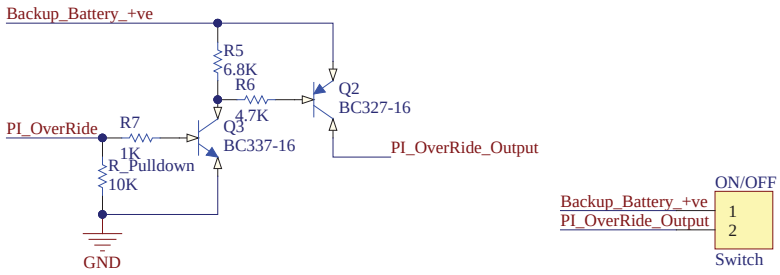


Figure 39: The shutdown override circuit schematic

Two prototype UPS systems were created. The first design acted as a proof of operation and worked successfully, however, a number of important improvements were noted and a second prototype was made. The first prototype did not test the power filter circuit. The PCB design can be seen in the Appendix 4.

In the first prototype, a 470 µF capacitor and a 20 kΩ potentiometer was used to tune the resistor value required for the RC circuit. This gave a value of 17 kΩ for a time constant of 8, and a shutdown time of 24 seconds. The RC values were then scaled to have an 800 kΩ resistor and a 10 µF capacitor. This gives a time constant of 8 which results in a theoretical shutdown time of 23.5 seconds. The actual shutdown time, is 51.4 s. This is because of the power filter capacitor discharging after shutdown.

The changes made for the second prototype include:

- Using a single comparator chip (LM311) instead of the quad comparator chip (LM339).
- A fuse was added in series with the battery to protect from short circuits.
- The UPS footprint was designed to be a Raspberry Pi extension cape, so it can be mounted directly on top of the computer. Screw terminals were also added to make the design compatible with other devices. The PCB was designed to fit easily above the raspberry pi board.
- A real time clock (RTC) was added to keep the date time as the raspberry pi does not do this on board and the system will be operating on a closed network.

A Raspberry Pi compatible PCB using the DS3231 RTC has been used. This is a low cost, high precision I2C real time clock. This product was chosen as it is easy to use, and can sit on top or underneath the UPS board.

2.5M stand offs were used to separate the Raspberry Pi and the UPS, along with a 7x2 tall female header to connect the UPS to the Raspberry Pi. Screw terminals were also incorporated in the design to allow the UPS to be used for other applications.

The UPS does not currently protect the system against jump starting or reverse battery connection. This will need to be addressed before the system is commercialised.

Other design considerations for future development include:

- Using a super capacitor to replace the SLA to provide the UPS backup battery. This will reduce the hub size.
- Reducing trickle loading for the comparator.
- Changing UPS to use a micro controller instead of a comparator. This micro can also handle communication to slave devices, removing the need for a USB RS485 dongle. This will further decrease the Master size and improve the master's structural integrity.
- Preventing premature shutdown and dead states.

Master Construction

The UPS has been designed as a raspberry pi cape, connecting directly to the GPIO pins. The UPS dimensions are 58mm(W)x70mm(L)x25mm(H), fitting well within the bounds of the raspberry pi. A long pin female header was used to mount the DS3231 RTC between the two boards. The top of the UPS reaches a height of 40 mm above the Raspberry Pi PCB when attached. The Raspberry Pi and UPS assembly is shorter than the selected SLA battery.

The master enclosure needs to house the raspberry pi, UPS, SLA battery, the USB-RS485 dongle and the Wi-Fi dongle. These components were organised to minimise the space required, and an IP65 171x141x80 polycarbonate enclosure with mounting flange was chosen. A mounting plate was made to fasten the components to the enclosure. The UPS, raspberry pi and mounting plate are attached stacked and connected with nylon standoffs and screws. The Battery will be mounted using zip ties. This is the first practical prototype of the master device. The construction will need to be revised for the final design. Two connectors were used. A gland was used for the input power and a four pin IP67 Line Socket connector was used for the RS485 and filtered power connection to the slave devices. The mounting plate was developed by modelling the different components and lining up the mounting holes. The plate was made using a laser cutter. The CAD model can be seen in Figure 40.

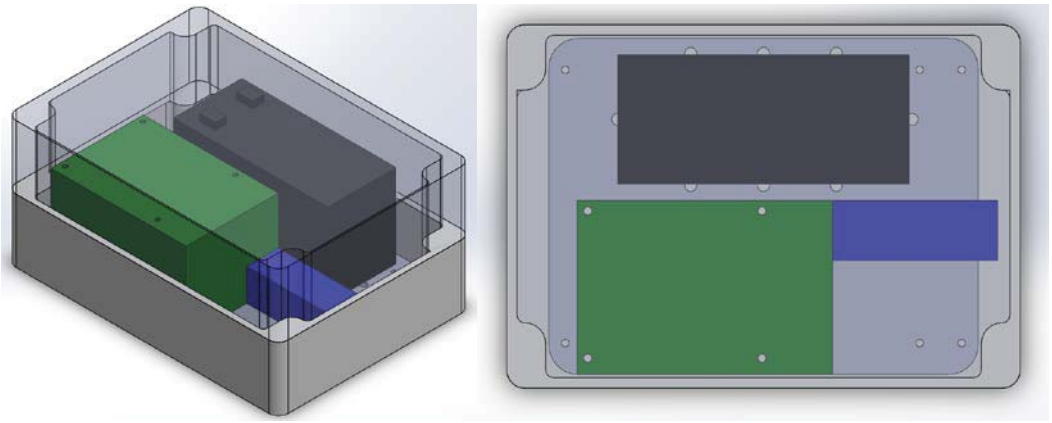


Figure 40: Master enclosure and mounting

The constructed master hardware can be seen in Figure 41.

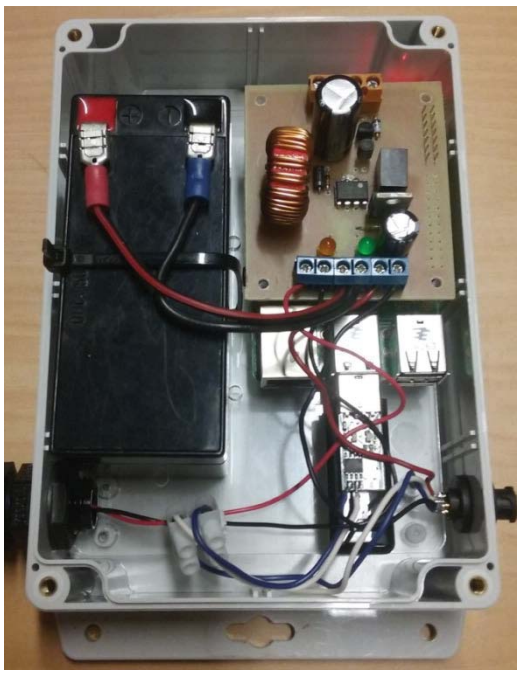


Figure 41: Master Construction

4.3.3 SLAVE Hardware Design

Over the course of this project, the slave hardware was developed over three stages:

1. The first stage consisted of an Arduino Nano mounted on a bread board with the EXAR SP483ECP-L RS-485 chip. This device was used to develop the communication protocol and the core slave software architecture.
2. Three identical Slave PCBs were developed using the Arduino Pro-Mini and the EXAR SP483ECP-L chip. These devices were developed to test the system using a multi-slave network and influence future slave PCB designs.
3. The final stage developed slave hardware to be applied on Reese Engineering's new 8128 Seed Drill. This hardware addressed all of the applications system requirements.

Slave Microcontroller Selection

The ATmega328 microcontroller will be used as the processor for the slave devices. This microcontroller is relatively cheap and is widely used as part of the Arduino devices. The main factor in choosing this device is the vast quantity of community support and resources that come with the Arduino libraries. “Arduino is an open-source prototyping platform based on easy-to-use hardware and software” [64] The Arduino libraries have been designed to make programming simple and clear, making it easy to use for beginners. This is ideal for a company with no background in programming. The Arduino Pro Mini PCB can be seen in Figure 42.

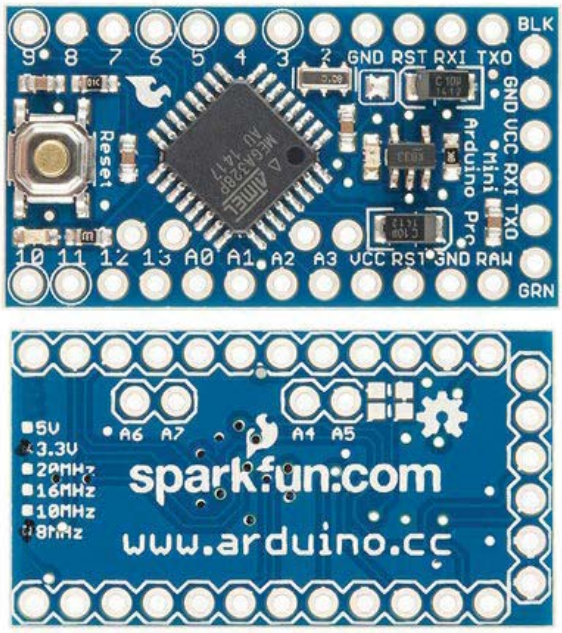


Figure 42: Arduino Pro-Mini top and bottom view

The 5 V, 16 MHz Arduino Pro-Mini was chosen as the microcontroller board of the slave devices. This board utilises the ATmega328 microcontroller [65]. It was chosen as it provided all the essential components to operate the ATmega328 microcontroller, simplifying the design and manufacturing of in-house circuit designs.

A summary of the Arduino Pro-Mini board can be seen in the table below. The 5 V model was selected as it has a higher clock speed and input/output voltage. The table below summarises the Arduino Pro Mini specifications.

Table 8: Specifications of the Arduino Pro-Mini. [65]

Microcontroller	ATmega328
Operating Voltage	3.3 V or 5 V (depending on model)
Input Voltage (RAW Pin)	3.35 -12 V (3.3 V model) or 5 - 12 V (5 V model)
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
Flash Memory	32 kB (of which 0.5 kB used by bootloader)
SRAM	2 kB
EEPROM	1 kB
Clock Speed	8 MHz (3.3 V model) or 16 MHz (5 V model)

The Arduino Pro-Mini does not have a USB port for serial communication. The USB port is not required, as the primary form of serial communication will be over RS485. To program the microcontroller, a USB to UART adapter is will be used.

Initial testing was conducted using an Arduino Nano mounted on a bread board. This device can be seen in Figure 43. This board is very similar to the Arduino Pro-Mini as both boards use the ATmega328. The Nano has a built in USB to UART transceiver chip and has a breadboard friendly pin layout. This made it easier to regularly reprogram the microcontroller and test the GPIO functionality. The Nano also has a built in LED on pin 13, which proved useful as a debugging tool.

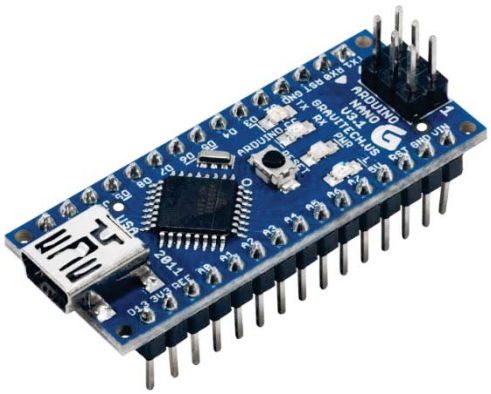


Figure 43: Arduino Nano

A USB-UART device was used for programming the Arduino Pro-Mini. A veroboard was used to align the pin layouts. The bootloader requires the reset line to be pulled low shortly before uploading new code. The chip reset pin of the device used did not reset the microcontroller correctly for the bootloader to flash the new code. To program the microcontroller, the on-board reset button needs to be held until the serial transmission started. The correct timing of this process can be reliably judged by releasing the reset button when the transmit light on the USB-UART device is lit. This device was chosen because it was available and suitable. There are more suitable devices on the market with the correct pin layout and built in chip reset.

Stage One: Breadboard Slave Test

To develop the communication protocol and the core slave software architecture, an Arduino Nano was used. This was mounted on a breadboard to interface with the GPIO. The Nano UART was connected to the EXAR SP483ECP-L RS485 converter, which communicated to the Raspberry Pi SBC using a USB to UART converter. A second USB to UART converter was used to observe the data transmission on a separate computer. This was done to debug the communication protocol. When reprogramming the Nano, the microcontroller receive pin needed to be disconnected from the RS-485 transceiver chip as it pulls the pin low.

Using this setup, the communication protocol and the core slave software architecture were developed and tested. Simple monitoring and control tasks were created to read the state of a switch, and set the state of an LED. Server module instances successfully interacted with the slave tasks. This setup helped develop and demonstrate the whole systems core functionality.

Stage Two: Slave Network Hardware Design.

Three identical slave PCBs were developed using the Arduino Pro-Mini and the EXAR SP483ECP-L RS-485 transceiver chip. These devices were developed to test the system using a multi-slave network and influence future slave PCB designs. Each of the test slave's hardware will be identical. The slave needs to be designed to test a variety of different tasks.

The analogue pins (A0, A1, A2 and A3) were used as they can be used for analogue input and digital input/output. As the serial interface of the micro is used for network communication, an LED was attached to pin 13 to provide an indicator for debugging. Each analogue pin is routed to a breakout header, with 5 V and ground pins. The 5 V is sourced from the Pro-Mini's on-board voltage regulator.

Pull up resistors were added for the I/O pins. It was later determined that the microcontroller has internal pullups. Two decoupling capacitors, 100 μF and 10 nF, have been added to the filtered voltage input. The schematic and PCB design can be seen in Figure 44.

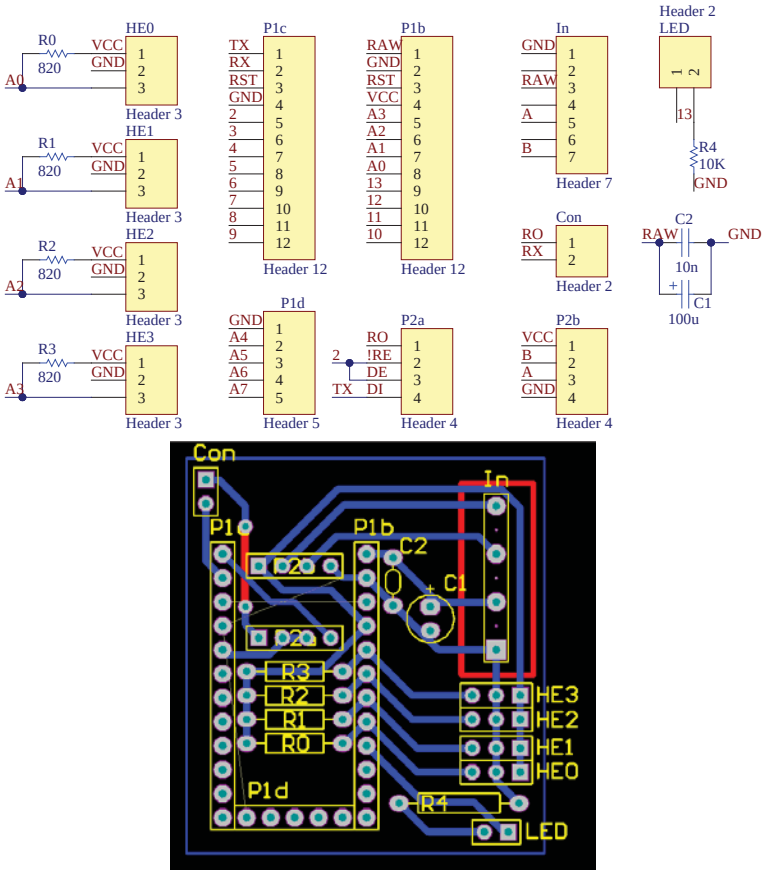


Figure 44: Circuit Schematic and PCB for the Test Slave

This hardware was important in developing the slave software architecture and the multi slave network communication protocol. It was used to develop and test: the system communication; the slave firmware; the systems modularity; and device connection/disconnection handling. The design overlooked catering for PWM output during pin selection. The stage one setup was used for testing tasks that require PWM output.

Stage Three: Practical Application

The monitoring and control system is intended to be applied on Reese Engineering's new 8128 seed drill machine described in section 4.1.1. This machine is the first electrically driven seed drill Reese Engineering has produced and will have the most features to monitor and control in their product range. The slave hardware will be designed to be fitted to this machine.

The machine will be fitted with three electric motors, controlling the feed rates of the three hoppers. Each motor will have an encoder attached for speed control and alerts. There will be encoders on the ground wheel and the two seeding fans to monitor their speeds. Three capacitive proximity sensors will be fitted to the three hoppers to indicate the hopper's product level. A master switch will be used to indicate if the machine is engaged and override the ground sensitive motor control. The ground speed encoder is mounted on one of the machines wheels and was manufactured in-house using a reed switch and magnets imbedded into a shaft collar.

Motor Selection

Reese Engineering selected the motor from Motion Dynamics in Australia [66]. This motor can be seen in Figure 45. This motor was chosen as it provided the required torque at a low cost. This motor is not suitable for extended outdoor use as it is not water tight. It was purchased for development to act as a proof of concept before committing to the more suitable, high cost motors. The motor specifications can be seen in Table 9.

Figure 45: The DC Motor selected [66]

Table 9: DC Motor Specifications [66]

Specification	Value
Voltage	12 V
Rated Current	27 A
Weight	5.7 kg
Gearbox	50:1 Worm drive
Rated Motor Speed	3200 RPM
Rated Gearbox Speed	64 RPM
Torque	50 Nm

The motor was retrofitted with a 20 pulse per revolution Hall Effect encoder also from Motion Dynamics.

Motor Driver

The IMS-2B single h-bridge motor driver was selected. This motor driver is well suited for driving the 27 A DC motor using the tractors alternator as a power source. Figure 46 shows the IMS-2B motor Driver.

Figure 46: IMS-2B Motor Driver

The Specifications of the IMS-2B motor driver are listed in the table below.

Table 10: IMS-2B Motor Driver Specifications

Specification	Value
Rated Voltage	3-14.6 V
Rated Current	50 A
Current Peak	110 A
Switching frequency	1 k to 200 kHz
Dimensions	50mm(L)x50mm(W)x35mm(H)
Duty Cycle	0-~98%

The pin descriptions of the IMS-2B motor driver is listed in the table below.

Table 11: The IMS-2B Motor Driver Pin mapping

Pin	Description
V+	Power Input +
GND	Power Input -
IN1	Forward PWM input (active high)
IN2	Reverse PWM input (active high)
EN	Driver Enable (active high)
CT	Current Sense: CT_voltage = Current(A)x0.155
VT	Voltage Sense: VT_voltage = Voltage(V)x0.32

PID Tuning for Motor Control

The manual method of PID tuning was used. The motor control software was developed and tested using the Arduino Nano and a breadboard. The motor driver pins: IN1, IN2, EN, CT and VT were wired to the Nano. A car battery was used to power the motor during testing. Motor testing software was written for the Nano to test the motor and determine the PID coefficients required for reliable speed control. The tuning process is detailed in section 5.2.

PCB Design

The ATmega328 has a total of 14 GPIO pins, of which 6 provide PWM output. This means that a minimum of two slaves will be required for this application. For responsive ground sensitive motor control, it is important that the motor driver, motor encoder, ground speed and master switch are all allocated to the same slave. This device is called the Control Block. The two fan encoders and three proximity sensors will be installed on the second slave, called the Monitor Block.

The Control Block

A PCB was designed as a breakout board for the Arduino Nano, with three IMS-2B Motor Drivers, three motor encoders, one ground speed encoder and the Master Switch. A 5 V Regulator was included to supply the 4 encoders using the 12 V filtered source from the UPS. This was to reduce the current draw through the Pro-Mini's on board voltage regulator.

The enclosure for this block was chosen to fit the three motor drivers. The PCB dimensions were determined based on the enclosure and the motor driver dimensions. The pin allocation was determined by the I/O requirements and ease of track routing.

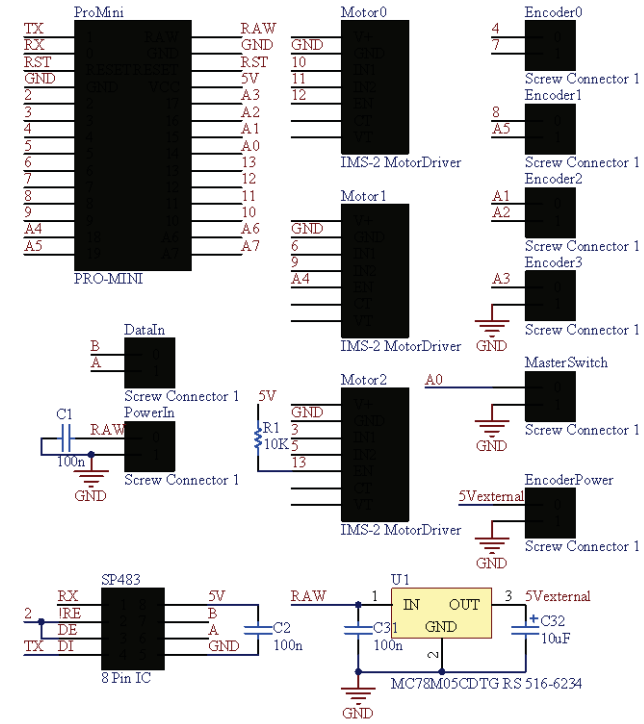
A pullup resistor was added to pin 13, as it is the only pin on the Pro-Mini board that doesn't have one built in.

The 5 V voltage regulator was used to supply the sensors. The metal flange is connected to ground and was bolted to a ground plane of the PCB to secure it and act as a larger heat sink.

Four decoupling capacitors were used. These capacitors were used to suppress the high frequency noise on the voltage supply tracks.

- 100 nF capacitor across the 5 V and GND pins of the transceiver and microcontroller.
- 100 nF capacitor across the RAW input and GND of the 5 V regulator.
- 100 nF capacitor across the RAW input and GND of the voltage supply terminals.
- 10 µF electrolytic capacitor across the output of the 5 V regulator.

The Control Block PCB schematic and board design can be seen in Figure 47. Screw terminals were used to provide simple connection to the enclosure connectors. This was done as this is a prototype board. Some wires need to share screw terminals. In future versions of the board, each wire will either have its own terminal, or be soldered onto the board. Headers were used to allow the microcontroller and the motor drivers to be easily connected or removed. This was useful for debugging as the microcontroller had to be removed for programming. The Schematic and PCB can be seen in Figure 47.



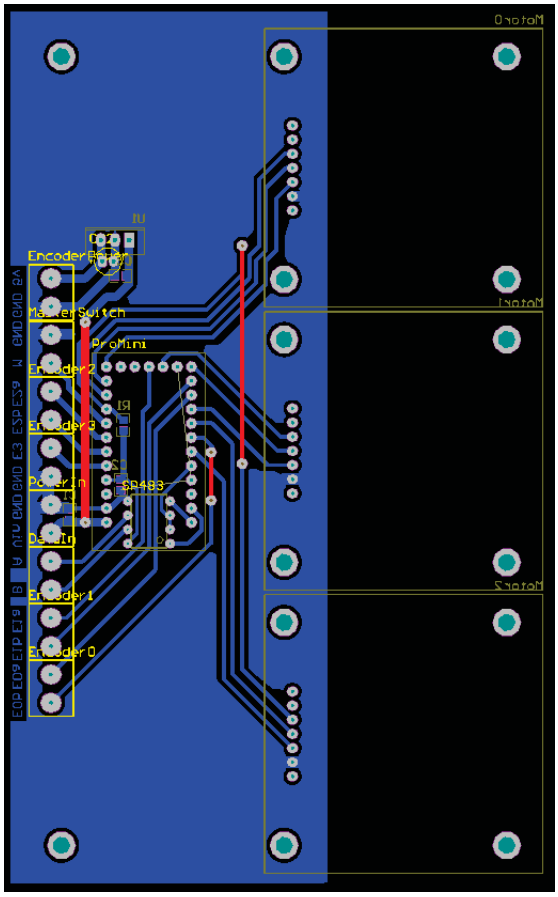


Figure 47: Control Block schematic and PCB design

Monitor Block

The capacitive proximity sensors operate at 12 V. This will be supplied by the ~12 V filtered supply from the UPS. As the micro-controller has a maximum voltage input of 5 V, a voltage

divider is required. Although the supply is filtered, it is not regulated at 12 V. This is not an issue for the capacitive proximity sensor, as the sensors currently used are rated to 35 V. This does however; change the voltage at the voltage regulator. A 5 V Zener diode has been added to the voltage divider to clamp the voltage at 5 V.

Similar to the Control Block implementation, a 5 V voltage regulator was used to supply the encoders. The metal flange of the regulator is bolted to a ground plane on the PCB to secure it and act as a larger heat sink.

As the encoders operate at 5 V, the signal lines can be wired directly to the micro controller.

Four decoupling capacitors were used. These capacitors were used to suppress the high frequency noise on the voltage supply tracks.

- 100 nF capacitor across the 5 V and GND pins of the transceiver and microcontroller.
- 100 nF capacitor across the RAW input and GND of the 5 V regulator.
- 100 nF capacitor across the RAW input and GND of the voltage supply terminals.
- 10 µF electrolytic capacitor across the output of the 5 V regulator.

The Schematic and PCB design can be seen in Figure 48.

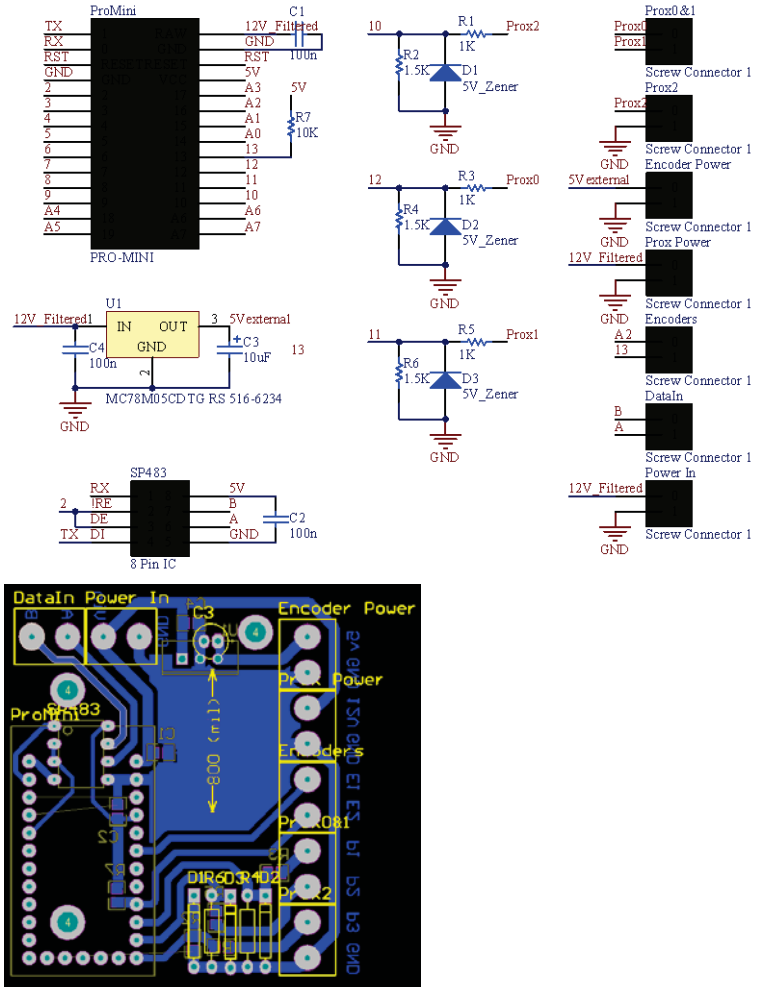


Figure 48: Schematic and PCB design for the Monitor Slave

Custom Motor Driver Design

Before the IMS-2B Motor Drivers were chosen, a custom, single direction motor driver breakout board for the Arduino Pro-Mini was designed. It was developed to control the three motors required for the 8128 Seed Drill, with built in current and voltage level sensing. This PCB was designed as the "Control Block" board, with inputs to read: the Master Switch; the three motor encoders; and the ground wheel encoder. This circuit was never manufactured or tested, as it was simpler and more reliable to use the IMS-2B Motor drivers.

Grassi (2008) details the development of a high current motor driver [67]. This article influenced the design of this custom motor driver.

This board needs to drive three 12 V 27 A DC motors. This board will have two voltage inputs. The motors are powered by the raw source from the alternator. The ~12 V filtered source from the UPS is used to power the microcontroller and supply the MOSFET gates.

The microcontrollers 5 V PWM signal drives one transistor, which in turn, drives a pair of complementary transistors, providing buffering and the voltage level transition required to drive the power MOSFETs. The power MOSFETs are used to drive the motor. Three IRF1405 N-channel MOSFETs were used, distributing the load and heat generation, reducing the need for an external heat sink. The gates of these MOSFETs are driven via a 15 Ω resistor which allows for quick charging and discharging of the gate due to its internal capacitance, reducing the transition time and heat generation. A 16 V 1 W Zener diode is used to clamp the gate voltage of the power MOSFETs. The parallel power MOSFET drains are protected using a 33 V 5 W Zener diode in parallel with a 100 nF capacitor. A fly wheel diode has been placed across the motor terminals, along with a 2200 µF capacitor to minimize the high voltage transients generated by the motor, alternator and when the battery is connected.

The back electromotive force (EMF) is sensed using a voltage divider to scale the motor voltage to the microcontrollers ADC voltage range. The back EMF is proportional to the motors speed and can be used in control as a substitute for encoders. The motor PWM must be switched off for a short time to record the back EMF. The ACS713 current sensing chip was added to monitor the current draw of the motor. This can be used for over current protection, and soft starting. The circuit was designed to handle 30 A for each motor.

When designing the PCB, it is important to consider inductive noise, track current capacity and component placement. The initial design was developed using a single sided PCB, which made track routing challenging.

The circuit schematic can be seen in Appendix 4. The PCB design can be seen in Figure 49. The physical size of this design is larger than the IMS-2B Motor Driver implementation.

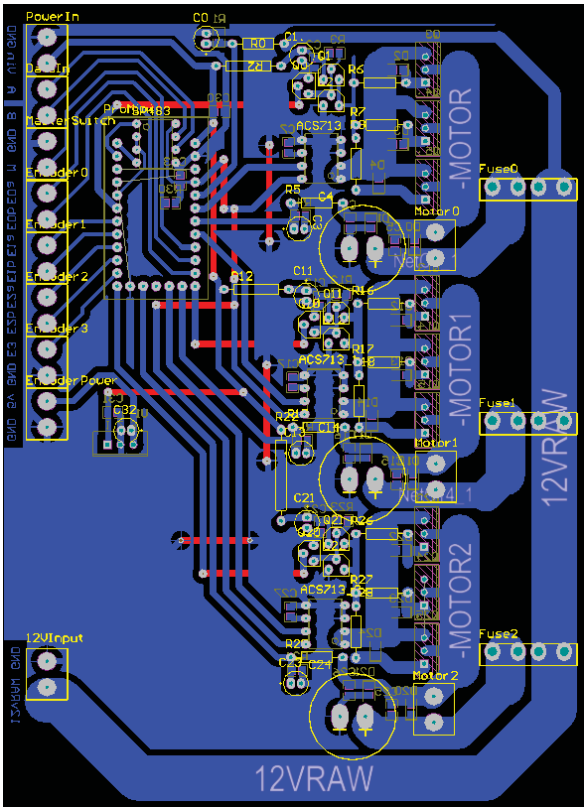


Figure 49: Custom Control Block PCB Design

Slave Construction

The physical construction of the slave devices is the same as the master device. Each device consists of an IP65 enclosure, a mounting plate, nylon standoffs, the slave PCB and connectors.

Ideally, unique connectors would be used for the different connections, making it impossible to plug a line into the wrong socket. As this is a prototype, it was not economical to get unique connectors. IP68 glands and IP67 seven pin connectors were used.

Each sensor or actuator had a connector to plug into the system. The monitor slave requires seven connectors and the control slave requires 10. As each connector only requires a maximum of four pins, the pin assignments were defined to prevent damage to the system if an incorrect connection was made. This pin mapping can be seen in Table 12, with the connector pin numbering shown in Figure 50.

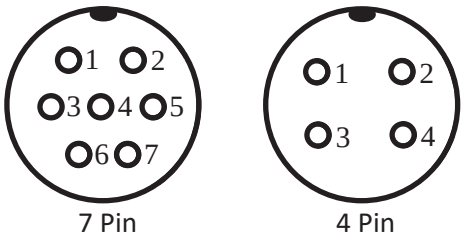


Figure 50: Connector pin numbering

Table 12: Connector pin mapping

Name	Connector Type	Pin Mapping
Single Channel Encoder	7 Pin	2: 5 V 4: GND 5: Input
Quadrature Encoder (Duel Channel)	7 Pin	2: 5 V 4: GND 5: Input 1 7: Input 2
Boolean Input	7 Pin	4: GND 5: Input
Proximity Sensor	7 Pin	1: Filtered 12 V 4: GND 5: Input
Data Out	7 Pin	1: Filtered 12 V 3: A 4: GND 6: B
Master Data Out	4 Pin	1: Filtered 12 V 2: GND 3: A 6: B

The constructed monitor slave device can be seen in Figure 51. The constructed master slave device can be seen in Figure 52. The connector assignment and pin mapping diagrams for the master and slave devices can be seen in Appendix 5.

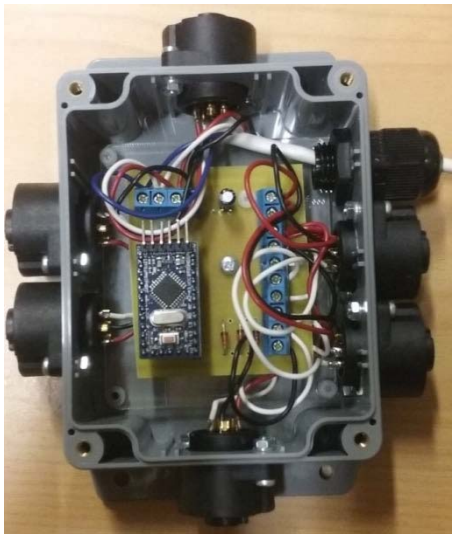


Figure 51: Monitor slave construction

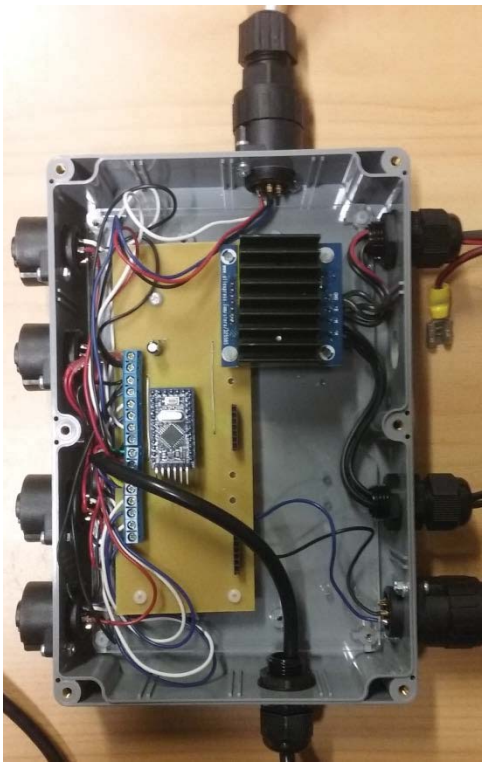


Figure 52: Control slave construction

4.4 Software Architecture

The system software architecture can be segregated into 3 major parts: the slave firmware, the master software and the website. For the slave and master applications to communicate, a serial data transfer protocol was developed. This achieves ordered, reliable

communication, regulated by the master device. The master application has been designed as a webserver, handling the web communication directly. The website dynamically generates its content based on the system parameters. The master device has three custom applications that need to be run on startup. These include: the UPS Auto-Shutdown handler; the RTC System Time update application; and the main master application.

A system image will be created that can be written to the SD cards of new devices. This effectively clones the original system to ease system setup for manufacture. Each implementation will require system parameters to be set during initial configuration. This includes: Wi-Fi SSID, password and channel; date-time; and module instance creation.

4.4.1 MASTER-SLAVE Data Transfer Protocol

A standardised messaging structure is required for the devices to communicate effectively. This is analogous to speaking the same language. Each message will be sent using a packet structure. Each packet is an array of bytes with a defined order, making it simple to decode.

Bytes were used to represent the packet information to minimise the information redundancy and increase the transfer speed. The alternative methods are integers and character strings to encode the message information.

The packet structure used consists of 3 parts: a header, the payload and a trailer.

Table 13: Packet Structure

Header	0xAA	0x55	Slave ID	Command	Size(n)
Payload	Payload				
Trailer	CRC				

This packet structure is the same for both master to slave and slave to master communication. The header is comprised of five bytes. The first two bytes are allocated to mark the start of the packet and are constant across all messages sent on this network. 0xAA55 was chosen as it is bit balanced and unlikely to occur randomly. Two bytes are used to reduce the chance of false packet detection.

The next three bytes in the header are: the slave ID, the command and the payload size.

- The slave ID indicates which slave the packet belongs to.
- The command indicates how the receiver should react to the message.
- The payload size defines how long the message is. This can be 0 if there are no payload bytes.

This header limits the number of slaves, commands and payload bytes to 255. This is more than enough for this application. The RS-485 to UART chip used has a network limit of 32 devices. A variable length payload was used to minimise redundant data transfer. The payload contains the message data.

The trailer byte marks the end of the packet, and consists of a cyclic redundancy check (CRC). Although RS-485 is inherently noise resistant, it is still possible that there could be an error in

the message. This can potentially have catastrophic effects on the machinery. To avoid this, a cyclic redundancy check (CRC) has been used to reject erroneous messages. CRC is a widely used tool for detecting errors in message blocks. CRC uses the remainder of polynomial division of the message as a representation of the data.

Each device has a packet manager class which handles: receiving and unpacking packets; calculating the CRC value; storing the received information; as well as packaging and sending messages. The slave implementation is written in C++ and the master implementation is written in Python, but are both fundamentally the same. Unpacking packets is done using a state machine, processing the message one byte at a time. As the packet has a clear structure, the state machine is simple. This can be seen in Figure 53.

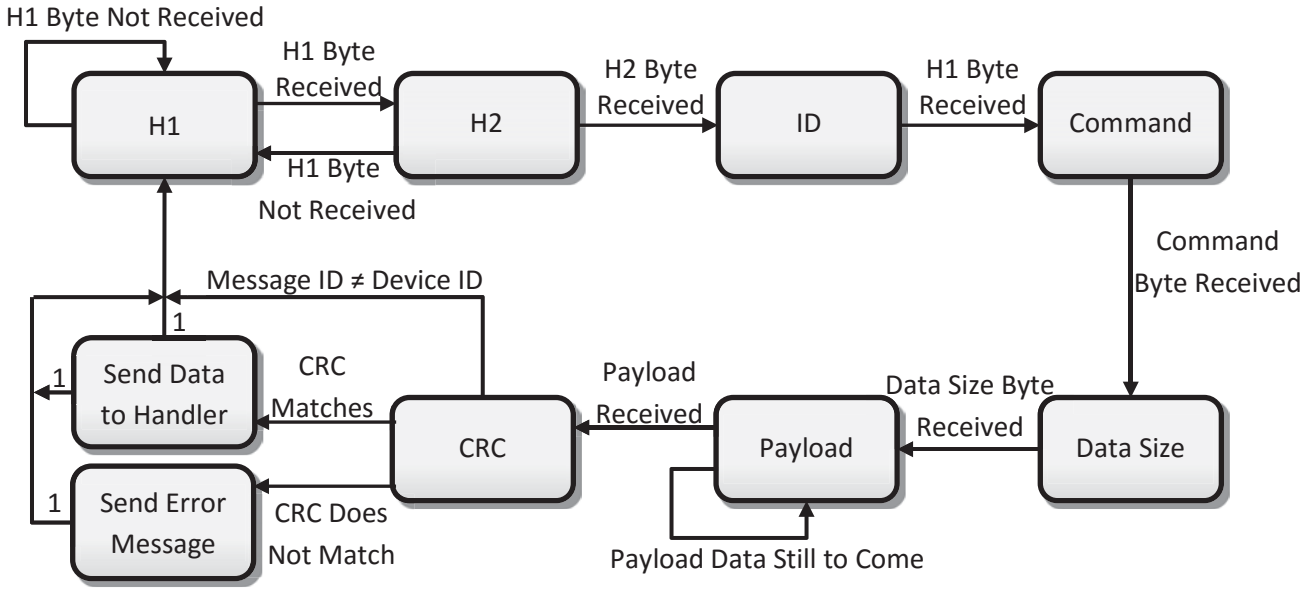


Figure 53: Functional block diagram used for unpacking packet messages

Each device uses serial interrupts to receive every byte sent over the network. Only one packet can be sent over the serial network at any one time. As the packets are of variable length, every device needs to individually un-package all packets sent over the network. This allows the device to know when the packet has ended and stops false packet detections midway through packet transmission. Once all of the information is extracted from the packet, if the device ID and the calculated CRC match the received values, a 'New Data' flag is set for the information to be processed. The information is buffered so new packets do not over write it while being processed. This packet structure provides the framework for reliable communication between the master and slave devices.

The concepts of integer division can be applied to polynomial division. Numbers of any base numerical system can be represented in polynomial form, where the (n-1)th digit is the coefficient to the (n-1)th power of the base. This is often used to convert the values to base 10.

- A base 10 example: $84.69_{10} = (8 * 10^1) + (4 * 10^0) + (6 * 10^{-1}) + (9 * 10^{-2})$
- A base 2 example: $10011100_2 = (1 * 2^7) + (1 * 2^4) + (1 * 2^3) + (1 * 2^2)$

Any binary number can be represented as a sum of powers of two. Modulo-2 arithmetic is used for the base-two number system, simplifying the addition and subtraction used in long division to XOR operations. CRC has been widely adopted for error detection and can be easily implemented using XOR gates and registers. The example below calculates the remainder of the message value “109” (01101101b) divided by generator polynomial “19” (10011b) using modulo-2 division. In this example, the CRC value would be equal to the remainder 14 (1110b).

```
                    1 0 1 = 5
              -------------
1 0 0 1 1 / 0 1 1 0 1 1 0 1
            0 1 0 0 1 1 | |
            ----------- | |
                1 0 0 0 0 |
                0 0 0 0 0 |
                --------- |
                1 0 0 0 0 1
                  1 0 0 1 1
                  ---------
                    1 1 1 0 = 14 = remainder
```

Figure 54: Modulo-2 long division to find the CRC remainder

The sender and receiver independently calculate the CRC value as a function of the packets header and payload. The receiver then compares these values to determine whether corruption is present. It is not practical to calculate the remainder from the entire message at once. Instead, this was calculated byte-by-byte. The calculation function used in the master device can be seen in Figure 54.

```
1  def calcCRC(self,header, data):
2      crc = 0xff
3      info = header+data
4      for byte in info:
5          if type(byte) is not int:
6              byte = ord(byte)
7          crc = crc ^ byte
8          for bit in range(0,8):
9              crc = crc & 0xff
10             if (crc & 0x01) == 0x01:
11                 crc = (crc >>1) ^ 0x9c
12             else:
13                 crc = crc >> 1;
14     return crc
15
```

Figure 55: Python implementation of the CRC calculation used in the master device

Although the packet payload is represented as an array of bytes, the payload can contain any data type as long as both devices know how to encode and decode the information properly. Different datatypes can be sent by transforming the value before and after transmission in the slave and the module handler respectively. The precision will be limited by the number of bytes used to transmit the transformed data. All task data types for application on Reese Engineering's machinery can be sufficiently represented using two bytes, restricting precision of the transformed value range to between 0 and 65535. Each value type will be allocated two bytes each in the outgoing slave transmission. The master device packet manager will decode the task variables by splitting the payload into 2-byte pairs.

This approach does introduce some redundancy for task data that can be represented using one byte (e.g. Boolean values). Tasks with counted data can send the count since the last transmission, keeping track of the total on the master device. As the slave data is requested regularly, limiting the count to 65535 is not a problem.

4.4.2 Running Programs on Start-up

All of the applications used for this system need to be executed at start-up. This sets up the system completely when it is powered on. The third party applications, such as Apache and MySQL do this by default.

The current version of Raspbian uses a 'Sys-V' initialisation system for executing commands during run level changes [68]. Many Linux distributions use 'systemd' instead. There are a number of ways to execute a program on boot, with the most popular for Sys-V distributions being: the software utility Cron and editing the /etc/rc.local file. Cron is a time based job scheduler that can execute tasks as a specified time, including reboot. The rc.local file is executed after all other start-up system processes. Bash commands can be appended to the rc.local. Both methods are suitable. Using the rc.local file for start-up execution was chosen.

Three scripts need to be called on start-up. These are:

1. The UPS Shutdown Handler: 'sudo python /var/www/restartTest/shutdown.py &'
2. The Master Software : 'sudo python /var/www/main.py'
3. Update the SBC date time from the attached RTC:
   'sudo echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
   sudo hwclock –s'

These lines are appended to the '/etc/rc.local' file before the 'exit 0' command.

During development, the Master software start-up call was omitted for testing and debugging purposes.

4.4.3 Uninterruptible Power Supply Software

The UPS will pull one of the SBC's GPIO pins low when the main power source from the tractor is disconnected. To shutdown safely, the SBC needs to run the 'shutdown' command.

Upon receiving the main power disconnection signal, the SBC can either shut down, or signal the UPS to maintain battery power by pulling a second GPIO pin high.

This is handled by software run on start-up. This program is separate from the main server program for two reasons.

1. During development, the main program will be restarted frequently and will often crash in debugging, making it unreliable for power down control.
2. If the main program becomes un-responsive during field operation, the shutdown script will still be running.

The shutdown script was written in python. To access the GPIO of the Raspberry Pi, the 'RPi.GPIO' library was used. The setup of this library is detailed in Appendix 2. To execute the system shutdown, the 'os' library was used to execute the system shutdown bash script.

The script will poll the UPS power off pin, shutting the Raspberry Pi down when the pin is pulled low.

To reduce the CPU usage of this application, the polling will be delayed with a sleep command for one second. This uses the 'time' library, utilising the 'time.sleep()' command. A snapshot of the code can be seen in the figure below.

```
# Import the libraries to use time delays, send os commands and access GPIO pins
import RPi.GPIO as GPIO
import time
import os
GPIO.setmode(GPIO.BOARD) # Set pin numbering to board numbering
GPIO.setup(11, GPIO.IN) # Setup pin 7 as an input
while True: # Setup a while loop to wait for a button press
    if(not GPIO.input(11)): # Setup an if loop to run a shutdown command when button press sensed
        print "Main Supply Disconnected!!!"
        print "This script was called on startup from '/etc/rc.local'."
        print "Shutting Down now!"
        os.system("sudo shutdown -h now") # Send shutdown command to os
        break
    time.sleep(1) # Allow a sleep time of 1 second to reduce CPU usage
```

Figure 56: UPS Shutdown script

4.4.4 Real Time Clock (RTC) Installation and System Clock Update

The systems data logging requires an accurate date-time to be recorded with each entry. The Raspberry Pi does not have a built in real time clock (RTC), and relies on date-time updates from the connected network using the Network Time Protocol (NTP). As this system is being designed to operate 'offline', an external RTC is required. An RTC uses a dedicated power supply, most commonly a coin cell battery, to maintain accurate time when the computer is powered off.

As the Raspberry Pi UART communication pins are being left as an emergency terminal access, I2C will be used to interface with the RTC. A high precision, I2C, Real Time Clock (RTC) circuit board for the Raspberry Pi will be used as it is premade and fits the requirement perfectly. This PCB uses the DS3231 RTC. In future versions, the RTC will be incorporated into the UPS.

The steps to get the I2C RTC working with the raspberry pi were followed from a tutorial on the Adafruit website.

4.4.5 MASTER Software Architecture

The Master Software consists of three major parts: the main program, the website and the database.

The main program acts as the interface between the website, the database and the slave network. It handles much of the core processing and resource handling.

The website provides the user interface (UI) of the system. It allows the user to interact with the machinery, providing them with monitored information, system configuration and control input.

The database stores a log of all events and data recorded by the system. This will store the history of the system and can be used for data analysis and report generation.

Main Program Software

The role of this application has not changed from the initial framework. Fundamentally, it manages the input and outputs, preforms calculations, logs the data and communicates to the website. During the redesign, many changes were made to improve the system and make it capable of being applied to the machinery.

The most notable of these changes include:

- System information used for loading on start-up is saved and loaded directly to file.
- The system data flow and module structure have been redesigned to enable more complex tasks and increases possible application.
- The database has been redesigned to suit the new data flow implementation and to allow for future support of graphing and reporting.
- Module type definition and handlers have been separated from the core software body to make the module development simpler.

A simplified representation of the main program architecture can be broken down to three main operations: Initialisation, Websocket handler and the Main Loop. The websocket handler and the main loop operate in parallel using multithreading. This structure can be seen in Figure 56.

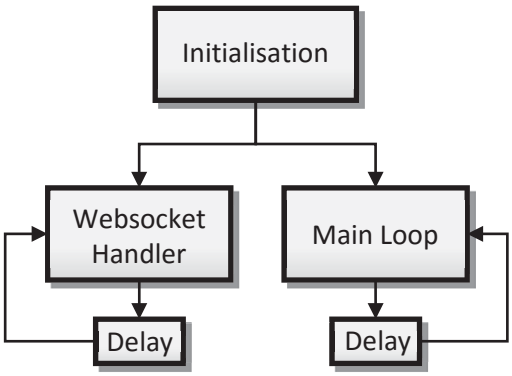


Figure 57: Core Structure of the Main Program Software

Initialisation:

1. Loads system information from file.
2. Connects to the slaves.
3. Sets up existing module instances.
4. Sets up the Database.
5. Sets up the display mode dictionary data mapping.

Main Loop:

1. Sends pending control updates to the slaves.
2. Requests data from all connected slaves.
3. Runs module handlers for each module instance.
4. Sends new data to the website.
5. Saves data to the database
6. Attempts to reconnect to any disconnected slaves.

The Websocket Handler is in charge of all communication with the website clients. One of the most notable functions of this block is processing client requests. This covers:

- Initialising the client by sending all information to setup the module user interface (UI) blocks.
- Handling user control field inputs by storing them to be processed by the main loop.
- Changing the module UI display order.
- Changing the display mode.
- Sending configuration information to the client.
- Saving updated configuration information.
- Creating new module instances.

The main program uses the following libraries: time, threading, MySQLdb, sys, json, logging, autobahn.twisted.websocket, twisted.internet, and collections.

Two custom libraries were created: PacketManager and Modules.

The **PacketManager** library was developed to handle packet based serial communication over the RS485 network. This is 'PacketManager.py' file in the working directory. The packet based messaging protocol and its implementation is described in section 4.4.1.

The **Modules** library contains all module specific code, segregating it from the core program software. This simplifies development of new module types. This is the 'Modules.py' file in the working directory, and is the only file that should be edited to create new module types.

Within 'Modules.py', module types are created in two stages: listing the required resources and creating a module handler. The required resources for the module type need to be defined in the 'moduleStructure' dictionary. The module handler is created utilising five dictionaries for data management between the core program software. The 'processSlaveData' dictionary is used to map the module id directly to the module handler.

The **moduleStructure** dictionary defines the structure of a module type. This dictionary defines all of the resources required for the module. This includes: the module type name; the required data stream types; parameters; user inputs; display data; and both recurring and irregular data log variables. This dictionary acts as a template to create new instances of the module type. It is a key stage in defining a new module. Each module type has a unique numerical ID key and seven information fields. These fields define the resources, parameters and data streams required for the modules operation.

1. **Name**: the name of the module type.
2. **Streams**: defines the data stream types required for the module. This is done by using the stream type ID and a custom user friendly name. The 'moduleData' dictionary is used to store the data stream pointers. The nth data stream defined in the 'Streams' field is accessed using: moduleData[{moduleID}][n].val()
3. **Parameters**: any additional parameters required by the module are defined here. Currently, only numerical parameters are possible. This is likely to change in future updates. Each entry is an array with two elements: a user friendly name, and a Boolean

value indicating if the parameter is required for creating the module instance. The parameter values are stored in the 'dataInfo' dictionary, accessed using: datainfo[{moduleID}][{parameter name}][{0:value,1:new/old(True/False)}].

4. **DispalyData**: defines the data to be displayed to the user. Each entry is an array with two elements: the display name and the value type. The value types are defined as: '0' for Numerical, '1' for Boolean and '2' for string. The display data is stored in the 'displayData' dictionary. The nth display data element defined in the 'DisplayData' field is stored in : displayData[{moduleID}][n]
5. **UserInput**: defines the user input variables. This uses the same format as the 'DisplayData' field. User inputs are stored in the 'dataInfo' dictionary, accessed using: datainfo[{moduleID}][{user input name}][{0:value,1:new/old(True/False)}].
6. **RegularData**: defines the data to be saved to the recurring data table. This is an array of variable names used to create the database column name. This data will be saved to the dictionary for every system iteration. The 'saveData' dictionary is used to store the values to be logged. The nth variable defined in the 'RegularData' field array is stored at: saveData[{moduleID}]["Regular"][n]
7. **IrregularData**: defines the data to be saved to the irregular data table. This is an array of variable names used for the table entry. The 'saveData' dictionary is used to store the values to be logged. The nth variable defined in the 'IrregularData' field array is stored at: saveData[{moduleID}]["Irregular"][n]

The information for each field (excluding 'Name') is stored as an array. It is important to note that the method of addressing the data values associated to the elements defined in the above fields is not the same across all field types. Accessing the element data defined by the 'Streams', 'RegularData' and 'IrregularData' fields are done by using the element array index. The 'Parameter', 'DisplayData', and 'UserInput' array elements are accessed using the variable name defined. This inconsistency will be addressed in future development. Each module instance can have a unique name. This set on creation, and can be changed in the same way as the 'parameter' field. Control and monitored data storage is no longer segregated.

The ModuleStructure dictionary and the five data management dictionaries, make it possible to create/initialise module instances and provide access to all required data streams simply, and efficiently without interacting with the core program software. These dictionaries are:

1. **deviceData**: stores all of the data received from the slave. Pointers are used to provide modules access to these variables. A 'dataStream' class was created to store the data stream values to ensure the address location of the data stream does not change when the variable is changed. This means that when the data stream value is updated in the 'deviceData' dictionary, it effectively updates all modules using that data stream. The 'deviceData' dictionary is a nested dictionary, and has the structure: deviceData[{device ID}]["Data"][{stream number}].val({data stream value})
2. **mapSlaveData**: is used to keep track of all of the modules data stream mappings. This dictionary is saved to file, and loaded on start up to initialise the main program.
3. **dataInfo**: is used to store the parameters of each module. A parameter is any information from the user concerning the module. This includes: user inputs, the module

name, and module configuration inputs. There are many cases where it is important to know if the data value is new. This is achieved using an array with two values. Index 0: the data value, Index 1: new value (True/False).

4. **displayData**: is used to manage all module UI display data.
5. **saveData**: is used by modules to schedule both irregular and recurring data to be saved to the database.

These dictionaries are managed by the core program software in both the main loop and websocket handler. They are created and set up within the initialisation stage, and when new module instances are created.

The core framework manages the defined data streams in the background. It is in charge of initial setup, slave communication, data mapping, calling the module handlers, database management and communication with the web client interface.

Website Software

The website needs to provide a user interface where the machine operator can interact with the system.

The expandable modular software architecture used in the main program means that the website design cannot be static. The website needs to dynamically create and maintain each module instance display UI. In the previous implementation, module display UIs had to be coded for each module type. This would be used as a template for each module instance display. This was one more step in creating a new module type. Using the new modular implementation of the main program, the website was designed to incorporate dynamic creation of module UI displays and a new module instance creation page.

JQuery Mobile [53] was used to provide resizable display widgets, allowing the webpage to be viewed on platforms with different screen sizes.

The website currently consists of three web pages: Home, New Module and Edit Module. The ‘Home’ page shows an ordered list of all module UI displays. The ‘New Module’ page provides the user interface to create new module instances. The 'Edit’ page is used to change the configuration information of individual module instances.

Provisions for implementing additional features to improve user experience were also developed, such as: customisable display tabs, unit conversion, graphing, and data downloading. Although the system functionality for custom display tabs has been implemented, the interface for editing and selection is not yet developed. This feature will enable the user to organise the module UI displays into different pages. By default, the ‘all’ display mode is used, which shows all module UI’s in the order of creation. Unit conversion will be done within the module UI edit page, but has yet to be developed.

When the ‘home’ page is opened, the website establishes a websocket connection with the main program. This interface can be seen in Figure 57. Upon connection, the main program sends an initialisation message to the website client, followed by regular data updates. The initialisation message is used to: populate the ‘Home’ page with the module UI displays;

create each module UI ‘Edit’ page; and create the ‘New Module’ page. This message includes a detailed list of all existing module instances and the display page module order.

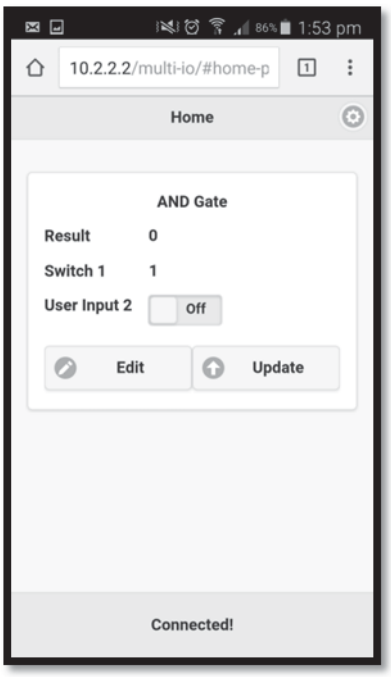
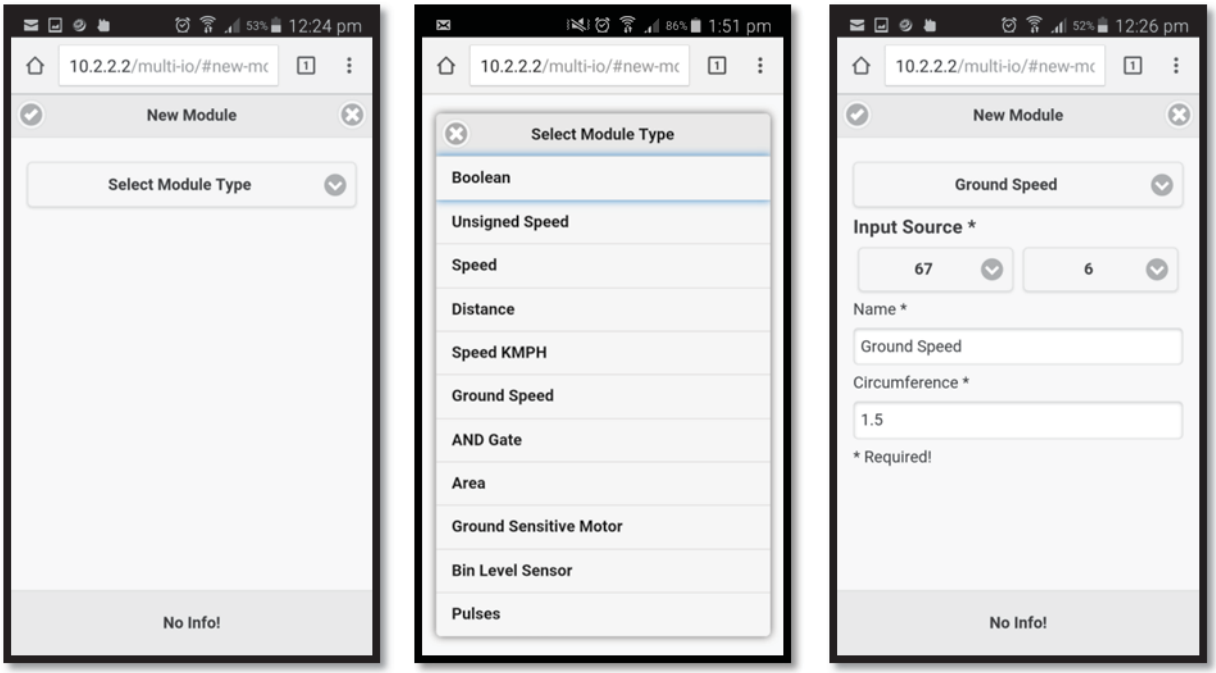


Figure 58: Example Module UI display on the Home page

The Module UI displays are created dynamically using the module type information defined by the ‘moduleStructure’ dictionary. The display order of these module UI displays is defined by the display mode tab. Currently module UI display components consist of a name, numerical display fields, Boolean display fields, numerical input fields, Boolean input fields and an edit button. If the module display has any user input fields, an update button is also included. This button sends all user inputs for that module to the master. All modules updates can be sent simultaneously by clicking the footer bar. Custom module type display templates can be created to control the aesthetics. If no custom display template is defined for the given module type, the website will create it itself.

The ‘**New Module**’ page is used to create new module instances from the defined module types within the website. This provides a user friendly way of setting up and configuring the system. Previously, this was done by editing the configuration database. The ‘New Module’ page has been designed to only allow creation of modules that are possible given the resources from the attached slave device. This prevents the creation of module instances that cannot work on the given system. To create a new module instance, the creator needs to select the required data streams and input the required configuration information. The process of creating a new module using the ‘New Module’ interface can be seen in Figure 58.



a. 'New Module' menu **b.** Module type selection **c.** Module creation form

Figure 59: New Module creation interface

The New Module page provides a list of the possible modules on the system (Figure 58 a & b). Module types that are not possible will not be available. Once the module type is selected, the page gets populated with form inputs to select the data streams and configuration information to create a new module instance (Figure 58 c). All required information has an asterisk suffix. At any time, the module type can be changed, and the page is repopulated with the appropriate input fields. To accept or reject the new entry, the tick and cross button at the top of the page are used. If any of the required fields are missing information, the user will be notified. A new module instance cannot be created with missing required information. The submitted information will be used to setup the new module immediately during run time. The user will be redirected to the home page where the new module UI display will be shown. The data stream is currently defined by its raw source information: Slave ID and resource number. Ease of use will be addressed in future development.

Each module UI display will have an 'Edit' button. This button takes the user to the '**Edit Module**' page, which is populated with all editable configuration information of the module instance. This page operates in the same way as the 'New Module' page. Each field will contain the current configuration information. Tick and cross buttons at the top of the page are used to accept or reject the changes.

System Storage

The system storage has changed significantly form the initial framework. System configuration is no longer stored in the configuration table. In the initial implementation, configuration information loaded from the configuration table was used to populate the system dictionaries. Instead, the core dictionaries are saved directly to file using the JSON Stringify function, simplifying the initialisation process. This also makes it easier to create new module instances within the system, as opposed to having to edit the configuration table directly. This function is also used to send information stored in dictionaries directly to the website client.

Data logging has been separated into two categories: regular and irregular. Regular data is logged every main loop cycle, whereas irregular data is only updated when required. When designing the two tables to store this information, two major factors were considered: minimising redundancy for improved storage capacity and graph/report making.

Each module type defines variables to be stored in the database in the 'IrregularData' and 'RegularData' fields of the 'moduleStructure' dictionary. Each module instance stores data to be saved in the 'saveData' dictionary.

The Irregular Table is simple. Entries are infrequent and not all variables are updated simultaneously. The table structure is as follows:

Table 14: 'Irregular' table structure

Column names	Datatypes	Example	Description
id	int(11)	1	Stores the entry number. Increments by 1
dateTime	timestamp	2016-03-21 02:19:47	Stores the date and time of the entry.
name	varchar(30)	'2-Boolean'	In the format '{moduleID}-{variable name}'
val	float	1	Stores the value.

Example entry: (1,'2016-03-21 02:19:47','2-Boolean',1)

The Regular Table requires a new column for each module instance regular variable. This makes each update simpler, as it only appends one new row to the table. The naming convention for each column is the same as the irregular name: '{moduleID}-{variable name}'. During the module instance setup done by the 'setupModule' function, the regular database is setup, ensuring that all module instance regular data columns are created. If they are not, the setupModule function will create the column. New columns can be appended to a table at any time, making it possible to create new module instances at run time. The table structure is as follows:

Table 15: 'Regular' table structure

Column names	Datatypes	Example
Id	int(11)	1
dateTime	timestamp	2016-03-21 02:19:47
{moduleID}-{variable name}	float	201.36
{moduleID}-{variable name}	float	0

4.4.6 SLAVE Software Architecture

This system needs to be capable of being applied to all of the machines that Reese Engineering produces. Reese Engineering also wants the system to be compatible with machines that they produce in the future. The monitoring and control requirements of each machine type can vary. However, many of the individual monitoring and control tasks will be the same across the product range. For example: Boolean inputs, speed inputs and ground sensitive product application.

To make it easier to develop the slave software for each of these applications, the software architecture will be broken up into functional blocks and a standard structure will be used.

The ATmega328 will be used as the slave's microcontroller. This micro controller is commonly used in Arduino devices and has a large amount of community support. This will be programmed in C++ using the android IDE. Utilising object oriented programming; each unique task will be implemented as a class, creating a library of classes that can be used across all of the slave devices. Each class will handle setting up the pins; doing the required calculations and reading/writing to the pins.

The slaves communication is handled by a class called "PacketManager". This handles all incoming and outgoing data over the serial network. Once a new packet is available, the received information is processed. The slave uses the command byte to determine how to act on receiving the packet. Handling the packet command is done using a switch-case statement inside the main loop. Each case calls a handler function and replies to the master. Using this structure, the slave can be set up to handle up to 256 unique commands with the current message transfer protocol and packet structure. The packet manager class is described in section 4.4.1.

The default master to slave commands used can be seen in the table below.

Table 16: Master to Slave Commands

Command Byte	Name	Description
0	Alive	Queries the slave's availability on the network. Used to determine whether the slave device is reachable by the master.
1	Discover	Requests an ordered list of the resource types available to the master.
2	Start	Enables the start flag. Used to ensure succinct operation of the system
3	Shutdown	Used to put the slave in a safe state to be shut down.
4	Data Update	Requests the resource data to be sent to the master.
5	Control Update	Sends new actuator handler state information to the slave.
6	Configuration Update	Sends new actuator handler configuration information to the slave.

Each slave can have a unique set of tasks. The master needs to know how to interact with each slave and its tasks. To do this, a defined structure needs to be put in place defining how to communicate. An array was used to store a list of the resource type's that are available on the slave device. This array defines each instance of the resource type in a specific order. This order is used to interact with the resource. The array is stored on the slave and sent to the master when the '**Discover'** command is received. The '**Data Update**' reply payload variables are arranged using this order, making it possible for the master to decode. The array index of each resource is used as the resource ID. This reduces redundant data transfer, as there is no need to prefix each variable with additional information. The resource type defines what the variable is, and how it should be used by the master modules. Each resource type ID can be used as a 'data stream', providing bidirectional communication with the slave tasks and the master. This allows for calibration and control information to be sent to the slave and monitored information to be sent to the master. Each command handler can process the resource ID differently. The resource type ID's are represented by one byte, limiting the number of types to 256.

This approach allows for custom devices to be connected to the network. A new slave can be successfully incorporated into the system if it uses the same communication protocol and the modules to handle the slave resources are defined on the master device. New tasks can be implemented, and new modules can be created to handle the tasks resources.

This implementation was initially developed to handle monitored data transfer and was later adapted to incorporate control tasks. Each control resource sends its current value in the data update. To distinguish between monitor and control types, the 0-100 range was set for monitoring, and the 101-200 range was set for control. This is just nomenclature, and has no real effect on the system. Although this implementation for slave resource management works, it does not synergise with control resources effectively. This can be addressed in future development

The defined resource types can be seen in Table 17.

Table 17: Slave Resource Type mapping

ID	Name	Monitor/Control Type
1	Boolean	Monitor
2	Unsigned Speed	Monitor
3	Signed Speed	Monitor
4	Distance	Monitor
101	Boolean	Control
102	Unsigned Speed	Control
103	Signed Speed	Control
104	Distance	Control

The '**Control Update**' and '**Configuration Update**' payloads are structured differently to the Data Update messages. Sending all of the information at once in the defined order can result in unwanted state changes, redundant data transfer and redundant slave processing. Instead, the configuration and control update payload structure will consist of the resource

ID, followed by all information required by the resource update. Each resource type will have a predetermined number of variables. Each variable is two bytes. Multiple resource updates can be sent in the same packet. These resource updates are used to control and configure the slave tasks. An example of the payload structure can be seen in the table below.

Table 18: Payload structure for the Control and Configuration update messages

Control or Configuration Payload						
0x01	0x0165	0x0000	0x02	0x0001	0xAF26	0x00A2
ID 1	Variable 1	Variable 2	ID 2	Variable 1	Variable 2	Variable 3

The Control Update and the Configuration Update command handlers use a switch-case statement on the resource ID to decode the message payload. It is possible to store configuration information on the slave's EEPROM. This will ensure the slave does not reset if the power briefly drops. The EEPROM is limited to 100,000 writes per memory location, so should be used sparingly. [69]

The function of the master and the slaves are fundamentally different, so the command bytes can be assigned different functions depending on the intended destination. The default slave to master commands can be seen in the table below.

Table 19: Slave to Master Commands

Command Byte	Name	Description
1	ACK	Acknowledged. Confirms that the master message has been received without corruption and has been processed.
2	NACK	Not Acknowledged. Indicates that the message cannot be processed, or contains corruption.
3	Alive	Used to notify that the device is available.
4	Data Update	Payload contains the resource data in the order specified by the 'Alive' message.
5	Discover	Sends an ordered list of the types of resources available to the master.

The Arduino 'setup()' function is used for all initialisation required by the slave. The Arduino 'loop()' function is used to handle the slave tasks and message commands.

Slave Tasks

Four slave task classes are required for the 8128 Seed Drill.

1. Boolean Input (BI): to handle the three capacitive proximity bin level sensors and the master switch.
2. High Speed Encoder (HSE): to monitor the speed of the three motors and two fans.
3. Low Speed Encoder (LSE): to monitor the ground speed.
4. Motor Control Task (MCT) Loop: to control the three electric motors.
5. Calibration: to determine the product application weight per motor revolution (kg/rev)

The pins used for each task are setup within the class constructor. The task functions for each object instance need to be called within the main code body.

The **Boolean Input** class is the simplest task implementation. This class has two functions: the constructor, which sets up the specified pin as an input; and the value() function, which returns the read value of the pin.

The Boolean input task uses a Boolean monitor recourse type to transmit its value to the master during a data update response.

Both **Encoder** classes assume rotation in one direction, as the ground wheel uses a single channel encoder, and the motors should only be used in one direction. Two different encoder handler classes were developed because different methods were required to calculate the speed for the low speed ground wheel and the high speed motors/fans.

The motor and fan speeds will range between 320 to 3200 RPM. With a pulse per revolution (PPR) of 40, the pulse frequency will range between 213.3 Hz and 2133 Hz. With a PPR of four, the pulse frequency will range between 21.33 Hz and 213.3 Hz. The ground wheel speed will range between approximately 0 and 320 RPM. With a PPR of four, the pulse frequency ranges between 0-21.33 Hz.

Both encoders use pin change interrupts to detect encoder pulses. The interrupt service routine (ISR) will call the corresponding encoder object's "pulse()" function.

The key difference between the two encoder implementations is the number of pulses used to calculate the speed. The high speed encoder measures the time between multiple pulses, as they occur more frequently. The low speed encoder measures the time between two pulses, as they occur infrequently. The RPM is calculated using the following equation:

$$RPM = \frac{60000.0 * (pulses - 1)}{(t_{end} - t_{start}) * PPR} \tag{9}$$

Where time is in milliseconds.

Once calculated, the end time (t_{end}) becomes the start time (t_{start}) for the next calculation.

A two point weighted average has been used to smooth the calculated speed. This uses the old average value as the first point, and the new value as the second. This filters out high frequency signal components by giving a larger weight to the previous average. This can be seen in Equation (10).

$$RPM_{Smooth} = \frac{2 * RPM_{old} + RPM_{new}}{3} \tag{10}$$

Where RPM_{old} is the previous RPM_{smooth}.

This method was used as it provides sufficient smoothing, without the need to record old values within an array.

Each encoder implementation provides speed and distance information to the master. The master knows the parameters of the class, such as: PPR, circumference and the kg/rev. Because of this, the master can calculate: the seeding rate, the distance travelled, and the product applied, from the RPM and the number of pulses since the last data update.

The motor control task uses the ground speed and the motor speed to calculate the new motor speed. The motor control task requires the most recent information to react appropriately. Based on this, the LSE speed calculation occurs each time a new pulse is detected and the HSE speed calculation is triggered by a timer interrupt.

The LSE "pulse()" function is used to calculate the speed. The HSE "pulse()" function counts the number of pulses, and records the time of the last pulse (t_{end}). The HSE has a "calcSpeed()" function which calculates the speed. Each class has a "getSpeed()" function, to get the latest speed value.

The LSE has a function to calculate the speed in meters per minute (getSpeedMPM()) for the motor control calculations.

$$V_{\frac{m}{min}} = circumference * RPM_{Smooth} \tag{11}$$

Each encoder needs to transmit both the change in pulse count, and the RPM to the master. This is done by using the distance and unsigned speed monitor resource types. As the encoder classes could be used to handle encoders with different PPR, the PPR and the circumference are sent to the slave using a configuration update from the master.

Speed monitoring becomes difficult when the encoder pulse frequency becomes low. It is important to know when the encoder has stopped, or is just moving slowly. Zero detection was implemented within the 'getSpeed' function. If the time since the last pulse is greater than a constant value, the speed will be set to zero. This constant value depends on the operational speed range expected.

The high speed encoder will be used to update motor speed to the motor control loop, as well as provide speed and distance information to the master. As the master knows the PPR of the encoder, it can calculate the distance from the number of pulses between data updates. The distance value is used to estimate the amount of product applied using the calibration value. In the same way, the application rate can be estimated from the RPM.

13.5 km/hr is a reasonable estimate of the average tractor speed (V_T). A four PPR single channel encoder will be installed on the ground wheel. The ground wheel circumference is 1.5 m. This gives an approximate average RPM of 150 and a pulse per minute (PPM) of 600.

$$RPM = \frac{V_T}{circumfrence} * \frac{1000}{60} = \frac{13.5}{1.5} * \frac{1000}{60} = 150 \tag{12}$$

$$PPM = RPM * PPR = 150 * 4 = 600 \tag{13}$$

The **motor control task** (MCT) is in charge of controlling the motor speed to maintain a constant application rate (kg/ha) with a variable ground speed. The application rate is defined by the user through the user interface. The MCT uses the ground speed and the motor speed to control the application rate by adjusting the motor speed. The motor speed is controlled using Pulse Width Modulation (PWM). The application rate is converted to kg per kilometre by the master before transmission to the slave as it is more useful for calculations and makes better use of the limited data size of two bytes.

The Proportional-Integral-Derivative (PID) Control method was used to influence the MCT. PID control aims to minimize the error($e(t)$) between the process variable (motor speed) and the desired set point (desired motor speed) by changing the control variable (the PWM duty cycle).

The control variable is calculated as a function of the correction factor, $u(t)$ given by the PID calculation. The correction factor is calculated by a weighted error sum of proportional, integral and differential terms. The control variable value is limited between 0 and 250 (98% duty cycle). The correction factor can be outside of this range. A clamping function was used on the correction factor to give the control variable duty cycle.

The PID calculation is represented by this equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)\,dt + K_d \frac{de(t)}{dt} \tag{14}$$

K_p, K_i,and $K_d$represent the proportional, integral and derivative coefficients respectively.

The proportional term results in an output proportional to the error. In a PID control system, the proportional gain (K_p) influences the how fast the control variable reacts to a change in the error. If the proportional gain is too large, the system can become unstable. The set point cannot be reached using proportional control alone, resulting in a steady state error. As the error gets smaller, so does the proportional term. The integral term is used to mitigate the steady state error. The Integral term is a weighted sum of past errors, making it proportional to both the magnitude and duration of the error. The Integral term is heavily influenced by past error, and can cause set point overshoot. The integral gain (K_i) needs to be selected for acceptable overshoot and settle time. The derivative term (K_d) uses the slope of the error to 'predict' the systems behaviour to reduce overshoot, improve settle time and stability. This is very noise sensitive. A PID controller will be used to maintain constant application rate (kg/ha).

The error can be calculated from the measured motor speed, and the required motor speed set-point. The following equation was derived to calculate the motor reference speed from the: weight per motor revolution (kg/rev) determined by the calibration; and the user defined seeding rate (kg/Ha). The reference speed calculated is used as the set point for the PID loop.

$$RPM_{ref} = \frac{V_{GND} * R}{C * 1000.0} \tag{15}$$

Where:

- V_{GND} is the ground speed in meters per minute, sourced from the ground speed encoder task.
- R is the application rate in kg per km.
- C is the calibration value in kg per rev.

The PID coefficient selection is described in section 5.2. The PID function will be used to calculate the duty cycle used to control the motor speed. As different motors will require different PID coefficients, they are passed to the constructor when the object is created. This allows the control class to be used for different motors as well as allow for changing the coefficients during operation for improved control.

Integral clamps were implemented to stop the integral value exceeding the possible control variable range. The maximum reference speed step size was also limited, to stop extreme changes, which can cause damage and significant overshoot of the motor. The PI calculation function is defined within the MCT class. The Arduino library function “analogWrite” is used to set the duty cycle of a PWM signal on a pin.

The MCT is regulated by a timer ISR, which handles the MCT objects and the motor encoder speed calculations. The ISR calculates the motor speed and, if in the correct state, will call the MCT PI() function, updating the motor duty cycle. The timer is set to overflow every 100ms (10Hz). The timer setup is discussed later in this section.

The motor control task uses an unsigned speed resource to send the motor speed set point. A configuration update is required to set the motor control tasks kg/rev and gear ratio. A control update is used to set the kg/km and the on/off state.

The **Calibration** task is required to determine the kilograms per motor revolution. This value is important for calculating the motor speed to achieve a given seeding rate (kg/ha). The calibration process consists of running the motor for a given number of rotations. The dispensed product is then weighed and the value is entered into the user interface. This task needs to turn the motor on, at a set speed for a number of rotations. To stop the calibration task conflicting with the ground sensitive motor control task, a state based system has been implemented. This method ensures that the motor is operated by one slave task at a time. States 0-3 can be entered by request of the master. The motor states are defined and described in Table 20.

Table 20: Motor Control States

State Number	Description
0	Off state. The motor is switched off.
1	Ground sensitive PI based motor control.
2	Unallocated.
3	Calibration.
4	Calibration wind down.

The calibration task does not require a regular data stream, as it occurs infrequently. The calibration task was implemented using a unique ‘Calibrate’ master to slave command (command 0x07). The current implementation of the calibration task does not relay information back to the master, only an acknowledgement packet.

The Calibration command payload contains the number of revolutions multiplied by 10. This is so the calibration can be set to rotate by a multiple of 0.1. The payload format is illustrated by the below table.

Table 21: Calibration payload format

0x00	0x0012	0x02	0x002A
Motor Number	Calibration Rotations	Motor Number	Calibration Rotations

The payload is decoded using a switch statement on the first byte. The next two bytes correspond to the calibration rotations. This implementation uses the same format as the control and configuration update commands.

The Calibration command handler sets the motor state to 'Calibrate'(State 3). This makes sure the motor control task is halted, the calibration pulse counter is reset and the motor is started with a fixed duty cycle of 50%. This corresponds to 32 RPM using the prototype motors.

The calibration task is used in conjunction with the motor control task. The calibration task was implemented for each motor control task within the pin change ISR. If the motor state is 'Calibrate'(State 3) the current calibration pulse count is compared to the target, if it is greater or equal, the motor is turned off, and the motor state is set to 'WindDown'(State 4). The motor state transitions back to 'Off'(State 0) when the motor speed reaches 0 RPM. This is monitored by the timer ISR.

Slave Task Software Implementation

The Arduino Pro-Mini has six analogue input pins and 14 digital I/O pins, of which: six are capable of PWM output. Four of the analogue pins can be used as digital pins. Two digital pins (0 and 1) will be used for serial communication.

The figure below shows the pin mapping in detail. This image provides all information required to utilise the board's pins using the Arduino IDE.

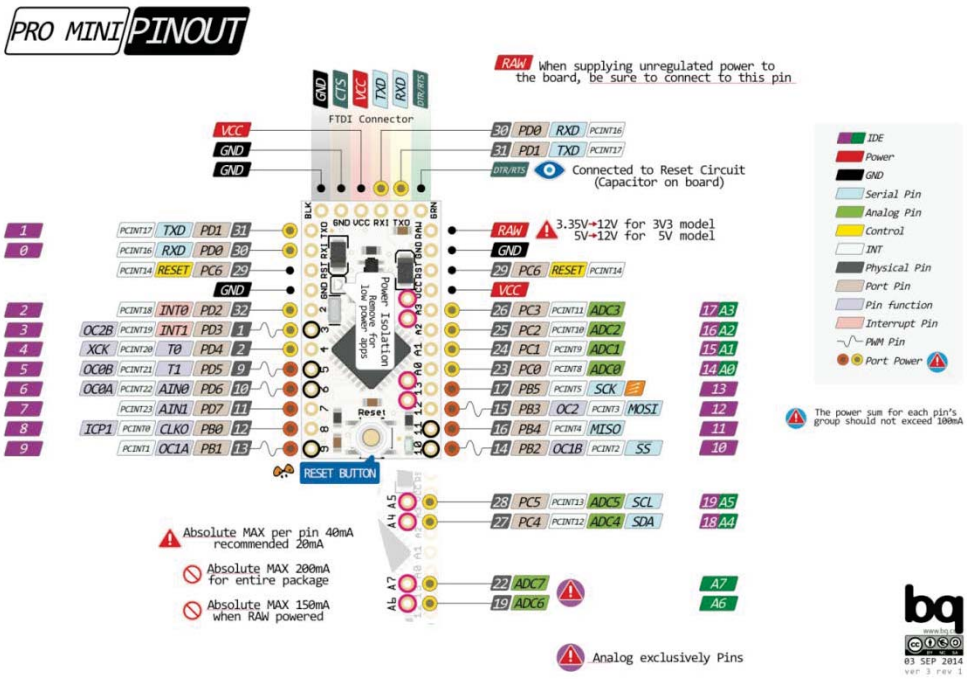


Figure 60: Arduino Pro-Mini Pinout [70]

Interrupts will be used to accurately time function calls, and detect pin state changes. It is not practical to use polling or delay methods, as accurate timing is important for most of the tasks. A serial interrupt is used to handle incoming serial data.

An interrupt signals the processor to take action on the event. An interrupt can be triggered by hardware, or software. The processor suspends its current activity to run the corresponding interrupt service routine (ISR) function to handle the event, returning to the previous activity on completion. Timer interrupts and pin interrupts will be used.

Pin Interrupts

The ATmega328 has two pin interrupt implementations, external interrupts and pin change interrupts. Pin interrupts are triggered by the pin changing value by either a falling edge, or rising edge.

1. **External pin interrupts** can be setup to trigger from rising edges, falling edges or both. Each pin has a unique ISR. The microcontroller has two external pin interrupts (INT0 and INT1) on pin 2 and 3. This can only support one quadrature encoder, or two single channel encoders. This is not sufficient for this application.
2. **Pin change interrupts** (PCINT) are attached to the three ports on the ATmega328. This covers all I/O pins excluding A6 and A7, which are exclusively used as analogue input pins. A pin change will call the corresponding port ISR. The pin toggled and the state transition will be unknown and will need to be determined in the ISR.

Pin change interrupts will be used as they can be used to support more time critical devices.

Pin Change Interrupt Setup

The pin change interrupts configuration is done in the setup function. PCINT setup is done using the following steps:

1. Setup pin to be an input with weak pullups. This can be done by either using: the DDRx and PORTx registers; or the "pinMode('pin',INPUT);" and "digitalWrite('pin',HIGH);" functions. The function method is a lot easier.
2. Switch Interrupts off during configuration using the cli() function.
3. Enable flags have to be sent in the PCICR register (seen in Table 22) to enable interrupts on the required port.

   Table 22: The PCICR Register [71]

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
PCICR	-	-	-	-	-	PCIE2	PCIE1	PCIE0

   The PCIE2, PCIE1 and PCIE0 enable pin change interrupts on the PCINT16-23, PCINT8-14 and PCINT0-7 respectively. The Arduino Pro-Mini PCINT pin arrangement can be seen in Figure 59.

4. Enable pin change interrupts on the individual pins using the PCMSK (pin change mask) registers. The register format can be seen in Table 23

   Table 23: The PCMSKx Registers [71]

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
PCMSK1	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16

5. Switch interrupts on using the sei() function.

The pin change interrupt ISR used depends on the pins port number. ISR (PCINT2_vect), ISR (PCINT1_vect) and ISR (PCINT0_vect) correspond to PCINT16-23, PCINT8-14 and PCINT0-7 respectively.

There are two limitations with this implementation.

1. The ISR will be triggered for any change in pin state. This can be overcome by reading the current state of the pin. If HIGH, the ISR was triggered by a rising edge. If LOW, the ISR was triggered by a falling edge.
2. Up to 8 pins can share the same PCINTx vector. When multiple pins on the same port are setup with pin change interrupts, the ISR needs to determine which pin caused the event. This can be done by storing the pin states, and comparing the values to determine which pin changed. The PCMSK0, PCMSK1 and PCMSK2 pinouts match the port registers: PINB, PINC and PIND.

Example: PCINT setup for pin 12

Figure 59 shows pin 12 corresponds to PCINT3. Bit 0 (PCIE0) of PCICR and bit three of PCMSK0 need to be enabled. The value of pin 12 is bit three (PINB3) in port B register "PINB". The entire code required to setup a pin change interrupt for pin 12 is shown below.

```
// portbhistory is used to store the previous pin states of port B.
volatile uint8_t portbhistory = 0xFF; // default is high because the pull-up

void setup(){
  setupPin12PCINT();          // Setup pin 12 pin change interrupt
}

void setupPin12PCINT(){
  pinMode(12,INPUT);          // set pin 12 to input.
  digitalWrite(12,HIGH);      // enable weak pull-up.
  cli();                      // disable interrupts.
  PCICR |= (1 << PCIE0);      // enable pin change interrupts on port 0.
  PCMSK0 |= (1 << PCINT3);    // enable pin change interrupt on pin 12 (PCINT3).
  // The following is also valid:
  //PCICR |= 0b00000001;      // enable pin change interrupts on port 0.
  //PCMSK0 |= 0b00001000;     // enable pin change interrupt on pin 12 (PCINT3).
  sei();                      // enable interrupts.
}

ISR (PCINT0_vect) {           // ISR for port 0, PCINT0-7
  // compare previous and current port pin state to determine which pin changed.
  uint8_t changedbits;
  changedbits = PINB ^ portbhistory;  // Changed Pins: 1, Unchanged Pins:0
  portbhistory = PINB;

  if(changedbits & (1 << PINB3)){     // pin 12 is the third bit of port B.
    // put your pin 12 ISR code here
  }
}
```

Timers

A timer ISR will be used to manage the PI loop for each motor control task.

A timer is a counter used to measure time events, much like a clock. The clock source (CS) is the system clock divided by a prescaler (1, 2, 64, 256, 1024). The ATmega328 timers have eight modes, of which, two are commonly used to generate timer interrupts at a desired frequency. These modes are described in Table 24.

Table 24: Timer mode descriptions

Mode Number	Description
0	"**Normal**" mode generates an interrupt when the timer reaches its maximum value and overflows back to zero. The timer can be preloaded within the ISR, controlling the start time to get the desired interrupt frequency.
2	"**Clear Timer on Capture**" (**CTC**) or Output compare mode generates an interrupt when the count reaches the value set in the output compare register (OCRx, for timer x). Once this value is reached, the timer is reset to zero. This value is chosen to achieve the desired interrupt frequency.

The interrupt frequency is set by selecting the timer prescaler and the preload/OCRx value. Equations (16) and (17) are used to determine the preload and OCRx values.

$$OCRx = \frac{clock\ speed}{prescaler * desired\ frequency} - 1 \tag{16}$$

$$Preload = 2^n - OCRx \tag{17}$$

Where *n* is the number of bits of the timer.

The ATmega328 has 3 hardware timers: timer0, timer1 and timer2. Timer0 and timer2 are 8 bit timers. Timer1 is a 16 bit timer. By default, timer0 is used for the Arduino timer functions, for example: delay(), micro() and millis(). Timer1 is used for the Arduino servo library, which is not used for this project. Timer2 is used for the Arduino tone() function. Each timer controls the PWM of two PWM pins. The PWM pins for each timer and the default PWM frequency can be seen in Table 25.

Table 25: Timer PWM pins and default frequency [72]

Timer	PWM Pins	Default frequency (Hz)
0	5, 6	976.56
1	9, 10	490.20
2	3, 11	490.20

The PWM frequency and duty cycle can be changed using the timer registers. The PWM frequencies were not changed for the prototype. The duty cycle was changed using the Arduino analogWrite() function. If a timer is used to generate an interrupt, it cannot be used to generate PWM on the respective pins. This was not known during the control block

hardware development, causing complications with the motor control. The motor control block will have to be redesigned to operate the three motors. As the motors only need to travel in one direction, this will be a simple design change.

The PWM frequency effects the motors vibration and audio noise. Higher frequencies result in quieter operation and lower amplitude vibrations. Lower frequencies are more energy efficient, as there is less switching loss. As the tractor and machine will be quite loud anyway, motor noise is not an important consideration. The default PWM frequencies for the ATmega328 will be used.

Pin change interrupts will be used to accurately time the encoder inputs and the master switch state changes. The ISR will determine which pin triggered the interrupt and then call the appropriate task handler function. For encoder tasks, this is the “pulse()” function. The master switch ISR code runs the motor control “pause()” function for each motor. The calibration task is implemented within the encoder pin change ISR to accurately count the motor pulses and turn the motor off when the distance has been reached.

A timer overflow ISR is used to create the motor PID control loop. Timer 1 was used, making PWM unavailable on pins 9 and 10. The 256 prescaler was used with a timer reload value of 59285, giving an ISR frequency of 10Hz. A low frequency was used as the motor is slow to react to control signal changes. This is largely due to the motors high rotational inertia. The timer ISR was also used to manage the calibration tasks wind down state transition.

5.0 Experiments and results

Software testing and debugging of the server and webpage consisted largely of trial and error. This section covers: UPS testing, motor tuning, slave hardware and machine trials.

5.1 UPS Testing

To validate the UPS operation, the following aspects need to be tested: power sense, shutdown timing, battery trickle charging, battery life time and the Raspberry Pi operation.

Power Sense Testing:

When the power source is connected, the power detect voltage was measured as 1.81 V. The power filter discharge curve affects the voltage level of the power detection circuit. Figure 60 shows the voltage level of the power detection circuit on connection and disconnection from the main supply. When the main supply is connected, the power detection circuit voltage goes to 1.81 V almost immediately. When the main supply is disconnected, the voltage level decays slowly due to the power filter capacitance. This is beneficial, as it stops false shutdowns when if the power is disconnected briefly.

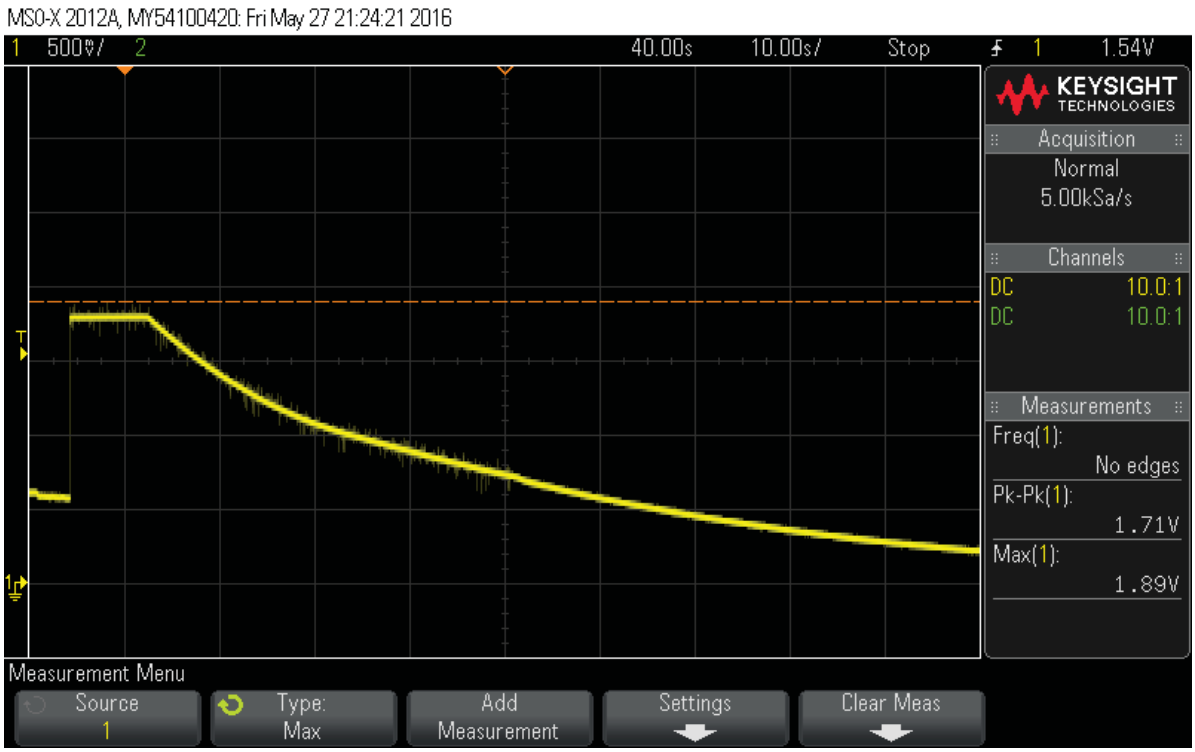


Figure 61: Power detection voltage level on connect and disconnect from the main supply

The Raspberry pi voltage transition level is between 0.8 - 1.3 V. From the above discharge curve, it could take between 10-22 seconds for the raspberry pi to detect the power being disconnected.

Through observation, it takes the Raspberry Pi approximately 18 seconds to realise the power has been disconnected and execute the shutdown command. This is within the

expected range. This means that if the power is lost for less than 18 seconds, the raspberry pi will remain on.

UPS Testing:

The reference voltage was measured as 0.730 V. The UPS shutdown time is 51.4 s. This is a result of the timing circuit and power filter being in series. The timing circuit has a time constant of 8, without the power filter, the shut down time will be 23.5 s. The RC values should be reselected to reduce the actual shutdown time to approximately 20 seconds. Figure 61 shows the RC circuit and 5 V output responses to connecting and disconnecting the main power supply. As expected, the RC circuit charges quickly, switching the system power on almost immediately. When the main power is disconnected, the timing circuit voltage level begins to decay. The rate of decay is reduced as the power filter capacitor takes time to discharge as well. Once the timing circuit voltage is below the reference voltage of 0.73 V, the comparator switches the system power off, resulting in the 5 V signal going low.

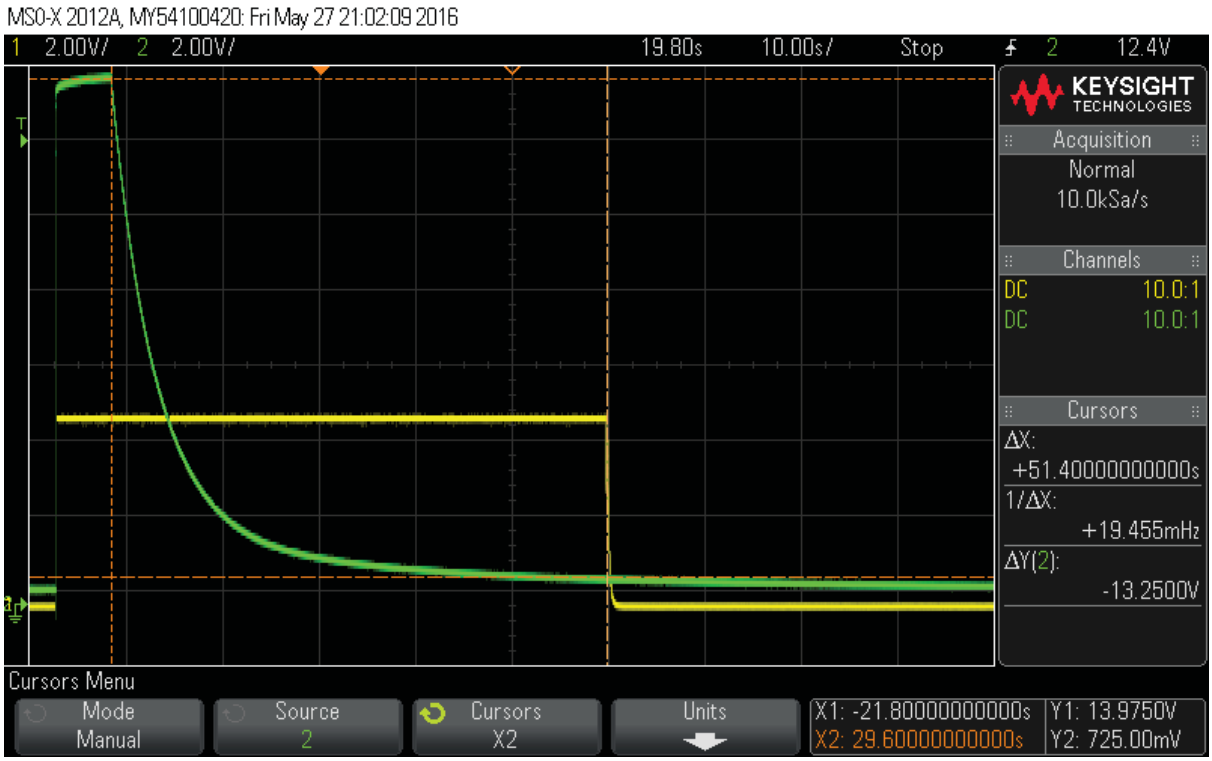


Figure 62: The timing circuit and voltage output response to connecting and disconnecting the main power. Yellow: 5 V output, Green: timing circuit voltage

Trickle charging:

As expected, with a 14.4 V power source, the battery voltage is maintained at 13.8 V.

Battery Life:

The current draw from the battery when the system is off was calculated by adding a resistor in series with the battery and measuring the voltage. The expected current draw is approximately 0.0125 A from the calculations in section 0.

The current can be calculated using Ohm's law ($V = IR$).Using a 1 kΩ resistor, it is expected that the voltage across the resistor will be in 12.5 V. The measured voltage was 2.37 V. This results in a trickle current draw of 2.37 mA. This is significantly lower than the predicted value. This is because the LM311 comparator draws much less than expected during the systems off state. The expected battery life time is 21 days:

$$\frac{1.2\ A\ h}{0.00237} = 506.32\ h = 21\ days \tag{18}$$

The current through the 10 kΩ current limiting resistor (~0.0012 A) is more significant than anticipated. The current draw can be reduced by using a larger resistor. For example: assuming 12V source, a 100 kΩ will draw 0.00012 A. This will almost double the expected battery life time.

The battery will need to be disconnected when the system is not being used for periods longer than the expected life span.

Raspberry Pi Testing:

The Raspberry Pi takes approximately 26 seconds to boot completely. This was measured by recording the time it takes to respond to a ping. The Raspberry Pi will begin to shut down approximately 18 seconds after the main power is disconnected. The UPS will disconnect the power to the Raspberry Pi after approximately 51 seconds. The raspberry pi can successfully override the UPS shutdown using the UPS override signal.

If the power is turned back on after the Raspberry Pi initiates its shutdown sequence, the system will remain off. To turn the system back on, the user has to disconnect the power for at least 60 seconds. This time can be reduced to approximately 30 seconds by changing the timing circuit.

A state based UPS will eliminate this problem by ensuring the power to the Raspberry Pi is disconnected if it goes into its shutdown sequence. This can be done using a micro controller.

5.2 Motor Tuning and Testing

The PID coefficients depend on the motor control library used. Once this was developed, a tuning program was also created. This program is controlled using a serial interface. The commands used are shown in Table 26.

Table 26: Motor testing commands

Command	Usage	Description
'd'	'd{x}\n'	Sets the motor duty cycle to x%
'i'	'i\n'	Returns motor information: Motor Back EMF; Motor Current draw; and Duty Cycle.
'f'	'f\n'	Sets the motor direction to 'Forward'
'b'	'b\n'	Sets the motor direction to 'Backward'
'y'	'y\n'	Turns the motor on
'n'	'n\n'	Turns the motor off
'q'	'q\n'	Test Mode 1: Ramps motor from 0 to 98% duty in steps of 5% every 2 seconds.
'p'	'p\n'	Test Mode 2: Step motor between a low and high set point using the PI control loop. Ki is set a 0, Kp increases by 0.01 every period.
'w'	'w\n'	Test Mode 3: Step motor between a low and high set point using the PI control loop. Kp is constant, Ki decreases by 5 every period.

When the motor is turned on, the microcontroller will stream data to the computer with the format: ['Time(ms)', 'K_val', 'error', 'motor Speed', 'PWM', 'Encoder Pulses'].

Ramp Test:

The ramp test was conducted to characterise the relationship between the motor speed and the PWM duty cycle. The duty cycle started at zero and increased by 4% every two seconds. The data was logged every 50 ms. The results of the test can be seen in Figure 62.

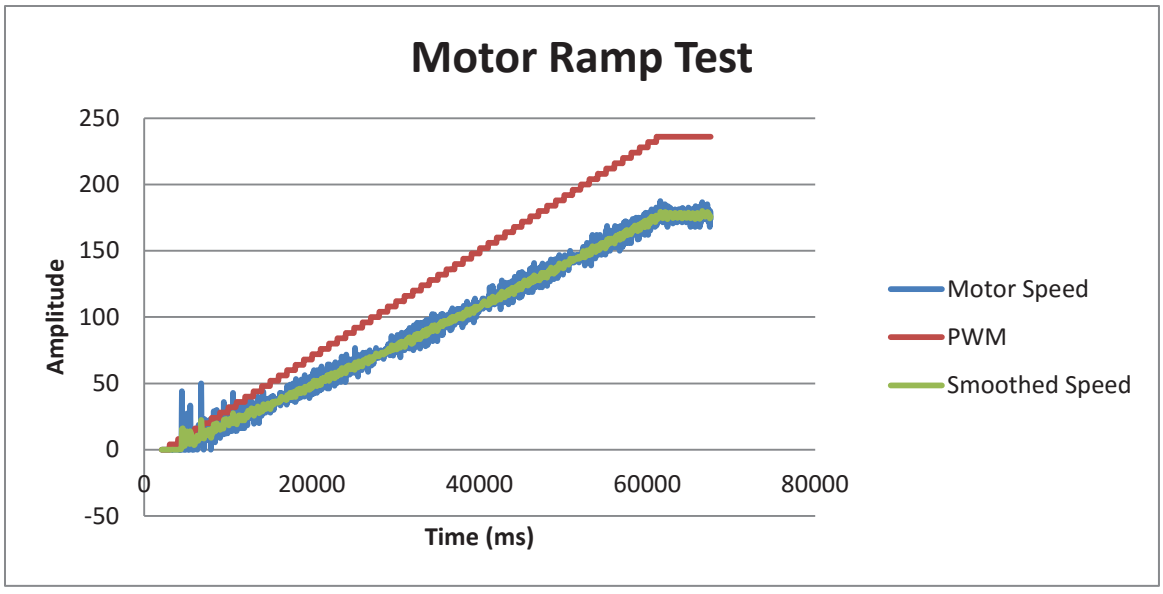


Figure 63: Motor Ramp Test Results

It can be seen that the motor speed is linearly proportional to the PWM duty cycle. These results showed that the motor can only be operated within 10 RPM and 98% duty cycle. The motor speed measured by the encoder is too noisy to be used for reliable speed monitoring and control. A smoothing function was used to reduce the noise of the monitored motor

speed. This showed a significant improvement. The noise level in the motor encoder is too large to effectively use the derivative component in PID control. Equation (10) shows the two-point weighted smoothing equation used.

The motor is rated to 64 RPM, the highest motor speed recorded is 180 RPM. This discrepancy is due to the wrong number of pulses per revolution being used. It was later determined that the 20 PPR motor encoder that came with the motor was swapped out for a 4 PPR motor encoder. There was also an error in the encoder pin change interrupt that caused false triggers. These factors were accounted for, and smoothing was implemented.

PID Tuning Test:

The manual tuning method was used to determine the PID coefficients. All coefficients are initially set to zero. The set point switches between 30-50% every three seconds. This provides a step response. The PWM duty cycle has been limited between 0-80% in the forward direction. The set point range was chosen because the motor jumps around violently when a large speed change is applied. This is due to the high motor inertia. To prevent damage, the motor was secured to the table.

Each coefficient was selected by incrementing the coefficient value for each pulse, keeping all other coefficients constant. This was first implemented over a broad range to identify which range should be focused on. The Kp value is chosen where the system starts to exhibit oscillation. This is because a high response time is desired. The Ki value is chosen to have a short set point convergence time, with small ripple. The Kd value is chosen to dampen the oscillation and ripple.

To determine the appropriate proportional coefficient (Kp) range, the Kp value is incremented by 0.1 after each step period, Ki and Kd were set to zero. The motor response can be seen in Figure 63.

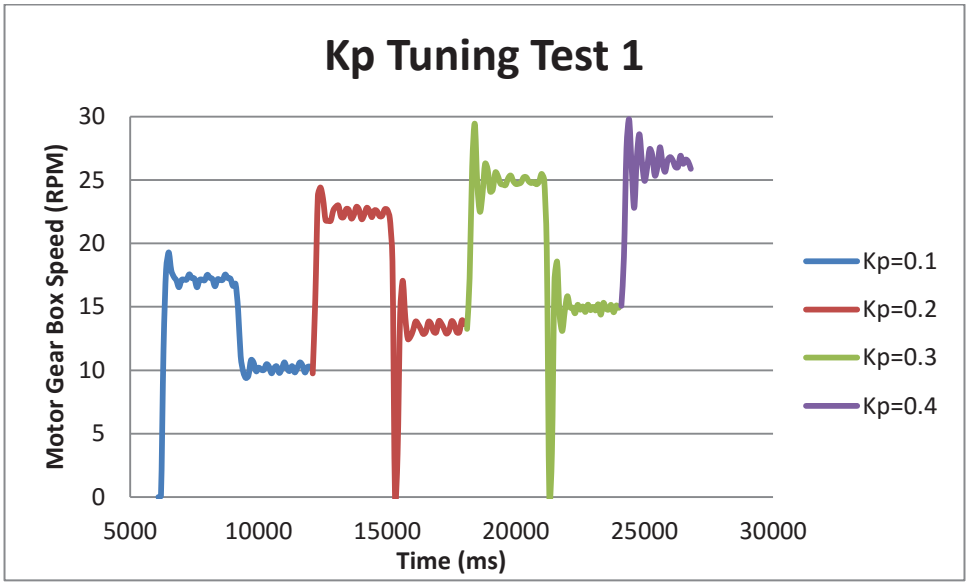


Figure 64: Motor Step Response with Kp Values incrementing by 0.1

The motor showed significant oscillation by Kp=0.3 and the test was aborted. The test was repeated with a step size of 0.01. The results of this can be seen in Figure 64.

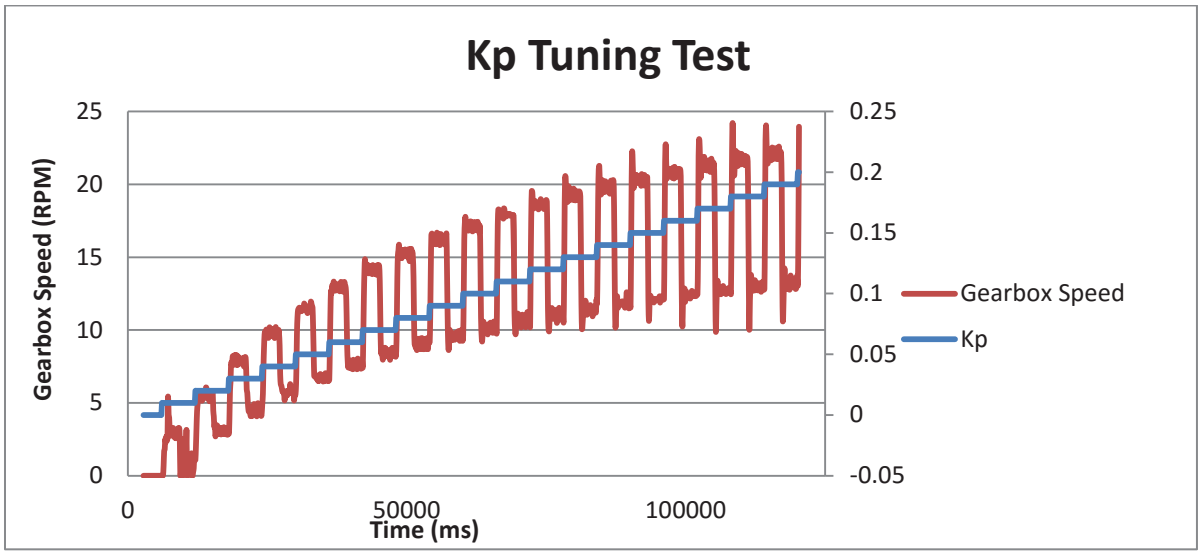


Figure 65: Motor Step Response with Kp Values incrementing by 0.01

Oscillation occurs at 0.13 and above. A close up of this range can be seen in Figure 65.

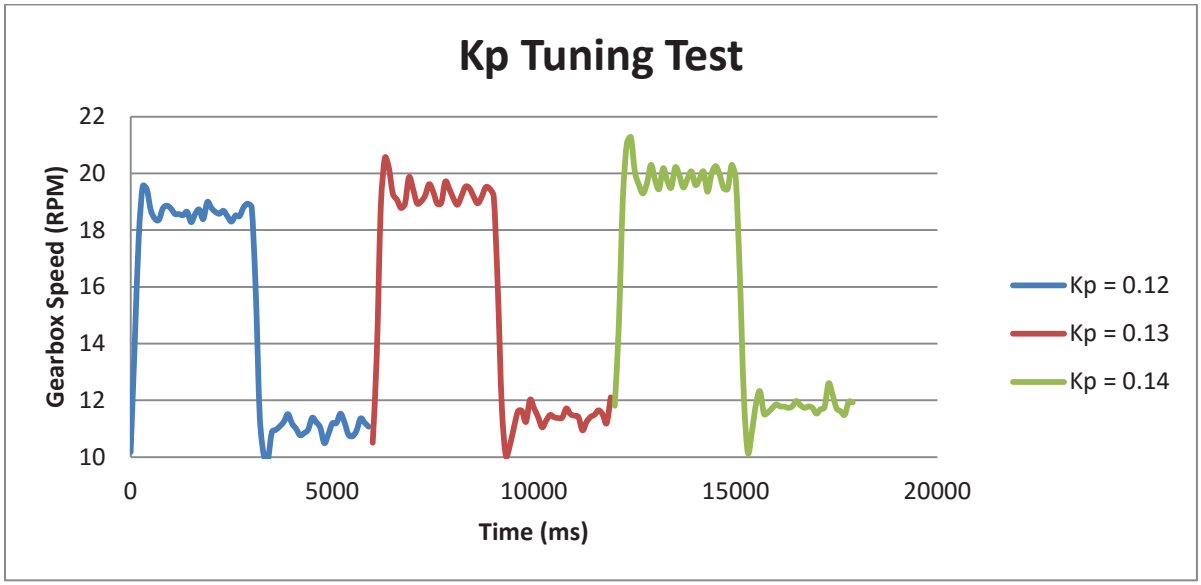


Figure 66: Motor Step Response with the Kp Values: 0.12,0.13 and 0.14

To give a high response speed, 0.14 was chosen for the proportional coefficient.

To select the Integral coefficient, the Kp value was set to 0.14 and Ki was changed using the equation:

$$K_i = \frac{1}{100 - 5x} \tag{19}$$

Where x is the pulse number.

The results can be seen in Figure 66.

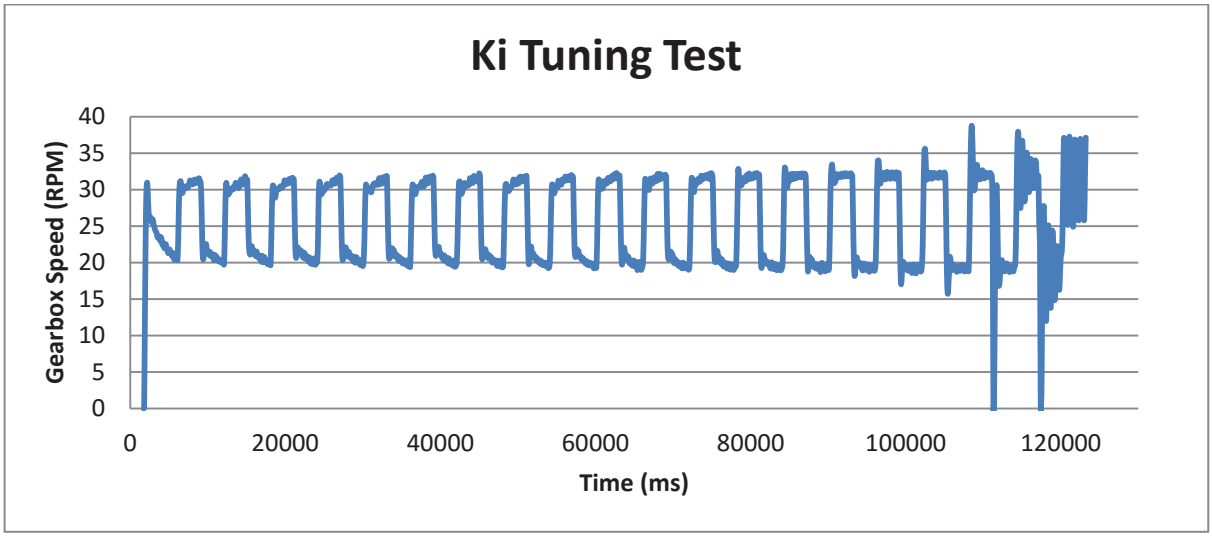


Figure 67: Motor Step Response with a constant Kp (0.14) and variable Ki

From these results, 0.04 was chosen for the integral coefficient.

The derivative coefficient was chosen using the same method. A sample of the results can be seen in Figure 68.

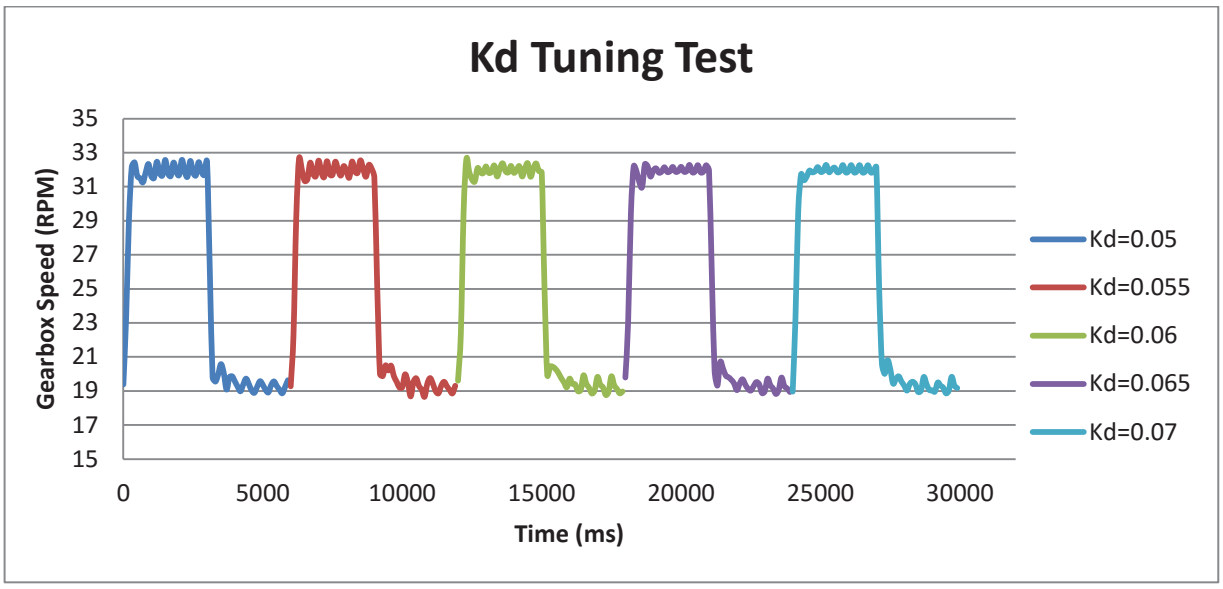


Figure 68: Motor Step Response with constant Kp (0.14), constant Ki (0.04) and a variable Kd

From these results, 0.06 was chosen for the derivative coefficient. The effect of adding each PID coefficient can be seen in Figure 68.

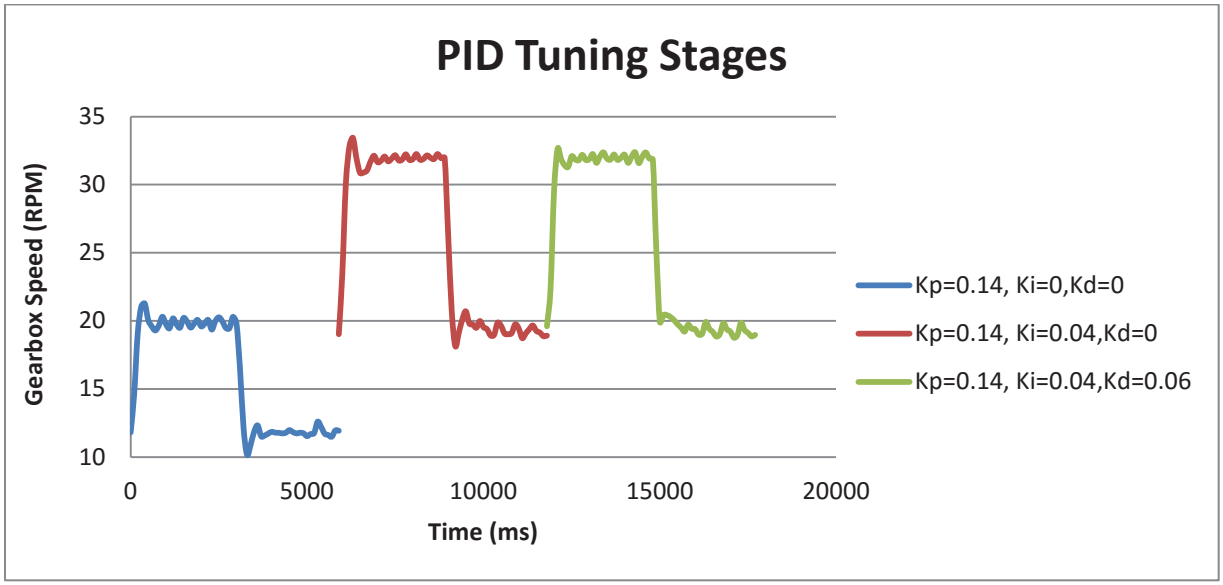


Figure 69: PID Tuning Stages

This figure shows the effect of each tuning stage. The Kp coefficient was chosen to give a fast response. The Ki coefficient was chosen to converge to the speed set point quickly. The Kd value was chosen to reduce the oscillation.

5.3 Machine Trials

The new 8128 Seed and Fertiliser spreader was not available for the initial system trials. A smaller, single hopper machine called the 'GrassFarmer' was used. This machine provides a suitable platform to conduct the initial tests.

The GrassFarmer uses a ground wheel to drive a variable speed gear box, which is used to control the seed rate. This machine uses a chart to set the seeding rate by manually changing the gear ratio of the variable speed gear box. The chart shows 5 generalised relationships between the desired seeding rate and the gearbox setting for 20 different seed types. This chart can be seen in Figure 69.

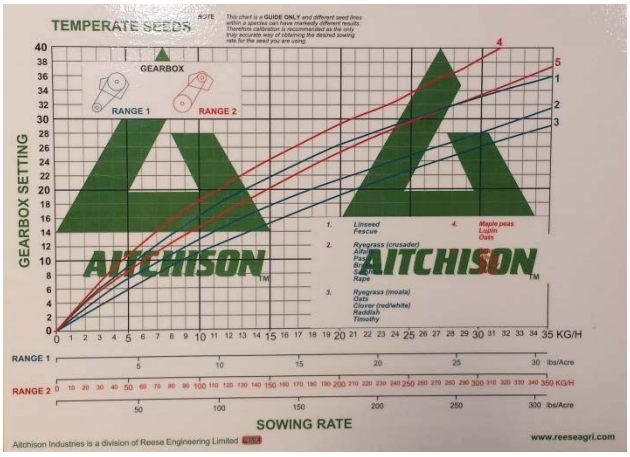


Figure 70: The relationship between the desired seed rate and the gearbox settings for the Grassfarmer

Currently, Reese Engineering uses a mechanical method for seed rate control. The application shaft is driven by an attached ground wheel through a variable speed gear box. To set the desired seeding rate, the gearbox ratio will need to be set manually. This provides a fixed seed application density for a variable ground speed. This set up can be seen in Figure 70. The drive wheel can be seen on the far left and the gear box control can be seen on the right.

Figure 71: Mechanical seed rate controller

The ground wheel and gearbox were replaced by an electric motor. Using an electric drive for this machine will allow for a variable seed rate that can be changed during operation. This will also significantly reduce the number of specialised parts required to manufacture the machine, as the drive wheel and gearbox will be replaced with an electric motor.

A mounting plate was created to attach the master and slave devices. A 4 PPR encoder was mounted to one of the ground wheels. The wheels have a circumference of 1.5m. All modifications made to the machine were temporary, so the machine can be rebuilt for sale. The modifications made to the machine can be seen in Figure 70.

Figure 72: Modifications made the GrassFarmer Seeder. Left: Motor and gearing to drive the drive shaft and stirring shaft. Right: Mount for the Master (top) and Control Slave (bottom) devices

The hopper has 14 application tubes distributed evenly across a span of 2.065m. The seed feeds into a sponge disk, which is rotated by the drive shaft. This disk collects seed as it rotates and drops it down a tube into the furrow. The speed of the drive shaft is directly

proportional to the seeding rate. To simplify the testing process and minimise the volume of seed required, only two of the hopper feeds were used. It is assumed that the seeding rate for each sponge feeder is the same. The hoses were used to feed the seed directly into a bucket for weighing. A second bucket was placed under the third hose to catch seed spillage. A car battery was used to power the electric motors. The machinery and test setup can be seen in Figure 71. A laptop was used as an SSH client to the Raspberry Pi. This was used to: run the main program; manage the database; debug the system and reprogram the applications. An Android tablet and Phone were used to provide the user interface.

Figure 73: GrassFarmer test setup

The gear ration between the motor and the drive shaft is 17:38. The two main gears can be swapped to reach higher shaft speeds.

The shaft speed range for the 17:38 configuration is 2.7 RPM to 26.3 RPM. The shaft speed range for the 38:17 configuration is 13.4 RPM to 131.4 RPM. This provides a decent range to cover the possible seeding rates.

To simulate the machine moving, two methods were available. Reese Engineering have motorised platform which can be placed under the wheel with the encoder. This platform does not have reliable speed control. A variable frequency generator was made using an Arduino Nano. The frequency output can be controlled using a potentiometer or serial commands. The design and operation of this device is described in Appendix 6. This device can be used in place of the ground wheel encoder.

A ‘Record’ switch was added to the seed application module UI to control when the data is logged.

Testing Method

1. Put seed in the first two hopper sponge feeds, disconnect the first two hoses and move them to the collection bucket. The third hose should be moved to a second bucket to catch overflow seed.
2. Start the main program by running the command:
   “python /var/www/multi-io/main.py”

3. Prime the system by running the calibration command until seed starts to flow. Power the seed motor off and put the seed back into the hopper. Stop the main program by running the command:
   "[ctrl]+c"
4. Clear the database by entering the commands:
   "sudo mysql -u root –pneo
   drop database gpio2;
   create database gpio2;
   \q
   sudo mysql -u root -pneo gpio2 < /var/www/Database/multi-io_setup.sql"
5. Start the main program; connect to the website.
6. Set the 'Record' switch to true run the test.
7. Weigh the seed dispensed and record values shown on the display.
8. When the test is complete close the main program and save the data to file by running the command:
   "sudo mysqldump -u root -pneo gpio2 > [test file location]"
9. Go to step 4 to repeat tests.

Calibration Trials

The calibration process is used to determine the weight of seed applied per motor revolution. This value is used to calculate the motor speed for a given seeding rate (in kg/ha) and ground speed. The calibration process is triggered by a switch widget in the user interface module UI.

Ideally, each seed type will have a constant calibration value. If this is the case, a seeding table could be used within the system. This is equivalent to the seeding chart shown in Figure 69.

This test aims to determine:

- The weight variation in the calibration process.
- If the seed application profile for a constant shaft speed is uniform.
- If the calibration weight for a given seed provides a reasonable estimate of the seed applied over operation.
- If the seed applied for a given number of revolutions is independent of the drive shaft speed.

The first test compared the calibration weight of two seed types: Wheat and rye-clover. Wheat is a medium sized seed and rye-clover is relatively small. The rye-clover was of seeding grade; however the wheat seed was of feeding grade and had loose seed husks mixed in. Each husk is slightly larger than the wheat seed, and significantly lighter. The results can be seen in Figure 72 and Table 27.

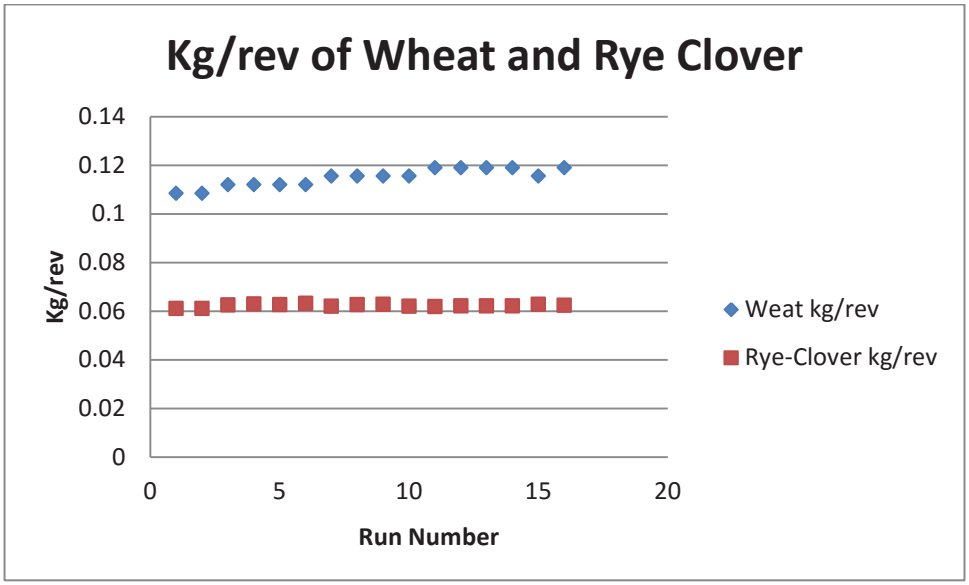


Figure 74: kg/rev of Wheat and Rye-Clover seed

Table 27: kg/rev results for Wheat and Rye-Clover seed

Run Number	Wheat (kg/rev)	Rye-Clover (kg/rev)
1	0.1085	0.061075
2	0.1085	0.061005
3	0.112	0.062475
4	0.112	0.062895
5	0.112	0.062615
6	0.112	0.063175
7	0.1155	0.061985
8	0.1155	0.062685
9	0.1155	0.06279
10	0.1155	0.061985
11	0.119	0.06188
12	0.119	0.062125
13	0.119	0.06216
14	0.119	0.06216
15	0.1155	0.062755
16	0.119	0.06237

The calibration value of the two seeds is clearly different. The steady increase in calibration value for the wheat seed was due to the seed husks being separated to the surface over time. Rye-clover seed was used for all future tests.

The second test ran the calibration function 30 times over three runs. Each test was conducted back to back over one day. Each run is described below:

- Run 1: 20 revolutions at 30 RPM.
- Run 2: 20 revolutions at 45 RPM.
- Run 3: 50 revolutions at 45 RPM.

The collated results can be seen in Figure 73 and Table 28. Each value is normalised to the weight per motor revolution.

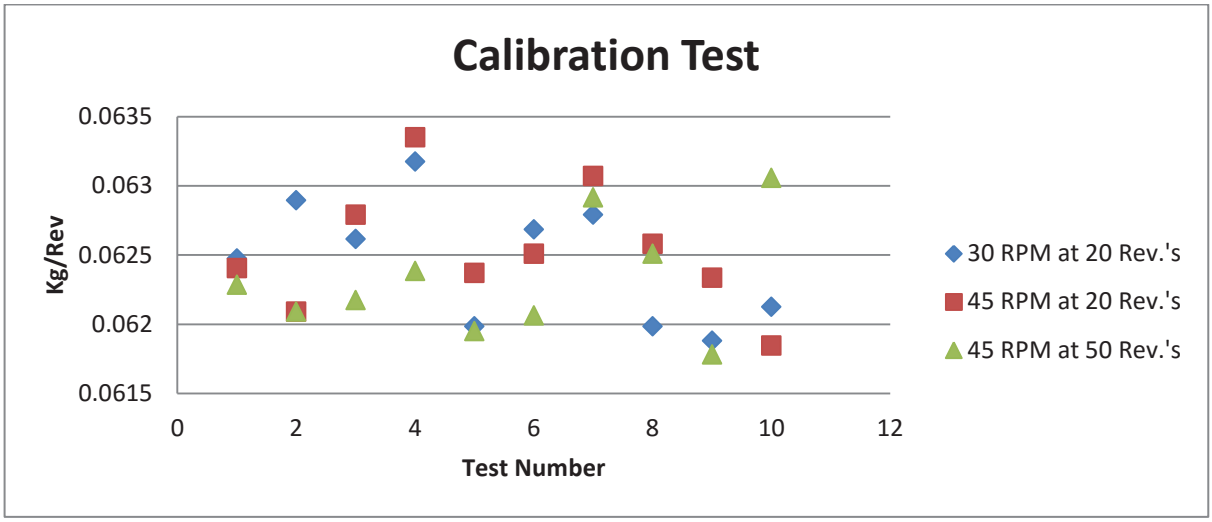


Figure 75: Calibration test results for Rye-Clover seed at different speeds and revolutions

Table 28: Calibration test results for Rye-Clover seed at different speeds and revolutions

	Kg/Rev		
Test Numeber	**30 RPM at 20 Rev's**	**45 RPM at 20 Rev's**	**45 RPM at 50 Rev's**
1	0.062475	0.062405	0.062286
2	0.062895	0.06209	0.06209
3	0.062615	0.06279	0.062174
4	0.063175	0.06335	0.062384
5	0.061985	0.06237	0.06195
6	0.062685	0.06251	0.062062
7	0.06279	0.06307	0.062916
8	0.061985	0.06258	0.06251
9	0.06188	0.062335	0.061782
10	0.062125	0.061845	0.063056

All three runs showed similar levels of variation, indicating that motor speed and the number of revolutions play little effect on the recorded calibration weight (kg/rev). This indicates that the seed applied, for a given number of rotations, is independent of the shaft speed and that the calibration weight at a given speed can be used to represent the operational speed range.

The mean weight recorded per revolution for Rye-Clover is 62.4 g, with a standard deviation of 0.431 g. 97% of the variation in the calibration value for this seed will be within ±2.58g (6σ) of the mean. This variation appears to be intrinsic to the seed singulation method and can be used as a benchmark of acceptable error for the later tests.

Constant Speed Trials

The seed application control loop was run for a period of five minutes with a constant ground speed of and seeding rate using Rye-Clover seed. This was repeated 14 times, split equally over two days.

This tests the reliability of the control system, the accuracy of the seed application estimate and the variation in the seeding weight. This seed is generally applied at 25 kg/ha with an approximate ground speed of 13.5 km/hr. All constant speed trials were conducted with a ground speed of 13.5 km/hr. At this speed, the machine will have travelled 1.125 km over the five minute test.

Each test showed similar levels of variation. The motor speed error can be calculated by comparing the actual speed with the calculated speed to achieve the seed density in kg/ha. The calculated speed is based on the calibration value determined at the beginning of each trial. An example of the motor speed error can be seen in Figure 74. This run had a calibration weight of 1.24 kg for 20 motor rotations and had a seeding rate of 25 kg/ha. This data can be seen in Appendix 7, Table 33.

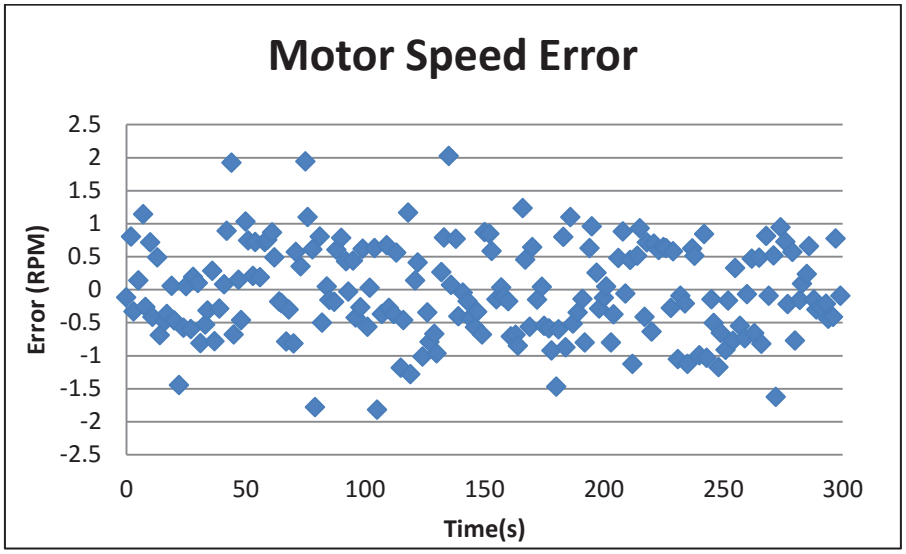


Figure 76: Typical Motor speed error observed

The mean error value for this run is -0.03 and the standard deviation is 0.7. All tests had a similar level of variation and a mean close to zero. The above figure is a good representation of the error seen across all runs.

The motor variation observed has the same range as the variation seen in the motor tuning process. This level of variation is acceptable for this system. To reduce this variation, the PID coefficients should be fine-tuned.

The difference between the measured and predicted weights for each run can be seen in Figure 75 and Table 29.

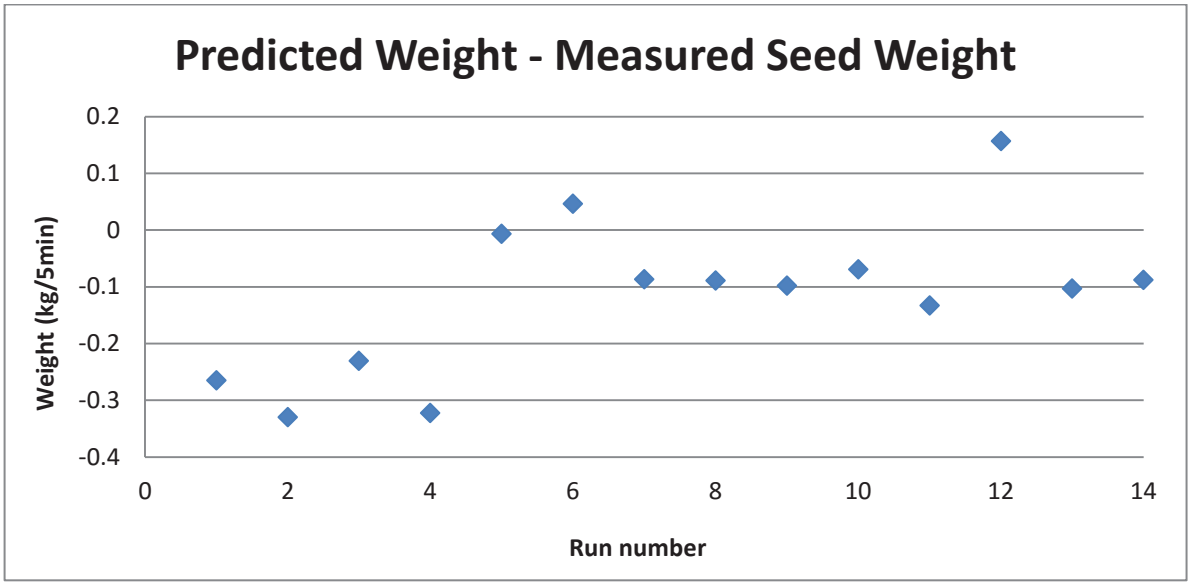


Figure 77: Predicted applied seed error (Predicted weight – measured weight)

Table 29: Predicted applied seed error (predicted weight – measured weight)

Run Number	Error Weight (kg/5min)
1	-0.265
2	-0.33
3	-0.231
4	-0.323
5	-0.0071
6	0.046
7	-0.087
8	-0.0895
9	-0.098
10	-0.0693
11	-0.1333
12	0.1569
13	-0.1035
14	-0.088

The first four value weights were recorded using scales with a resolution of 5 g. After this, scales with a resolution of 0.1 g were used. The scales resolution has a large effect on the accuracy of the calibration value. This explains why these values are quite different from the other data. The predicted value seems to be slightly lower than the measured weight. This is likely due to the motor wind down in the calibration process not being accounted for. At 25 kg/ha, the motor rotated an average of 91.5 times over the five minute period. At 50 kg/ha, the motor rotated an average of 187 times.

Based on the random seeding variation observed in the calibration trial, it is expected that 97% of the random variation seen will be within ±2.58 g per revolution. Extrapolating this for the five minute constant speed trials will give a rough estimate of the level random of

variation. For 25 kg/ha, 97% of the variation will be within ±236 g. For 50 kg/ha, 97% of the variation will be within ±483 g.

The observed application error is well within the expected random seeding variation intrinsic to the system. An error of 200 g for five minutes of operation at 13.5 km/hr extrapolates to approximately 1 kg per hectare. This calculation is shown in equation (20) and (21). This level of error is well within the currently accepted error level of the mechanical seeding system.

$1ha = 10000m^2$

$Span = 2.065m$

$$distance = \frac{Area}{Width} = \frac{10000\ m^2}{2.065\ m} = 4842.615\ m = 4.84\ km \tag{20}$$

$$Error_{hectare} = \frac{200\ g}{1.125\ km} * 4.84\ km = 861g \approx 1\ kg \tag{21}$$

The final weight measured for each run was normalised to kg per revolution and plotted in the graph shown in Figure 76. The normalisation removes variation from ground speed and run time. The first four runs of each day (1-4, 8-11) were at 25 kg/ha and the last three of each day (5-7, 12-15) were at 50 kg/ha.

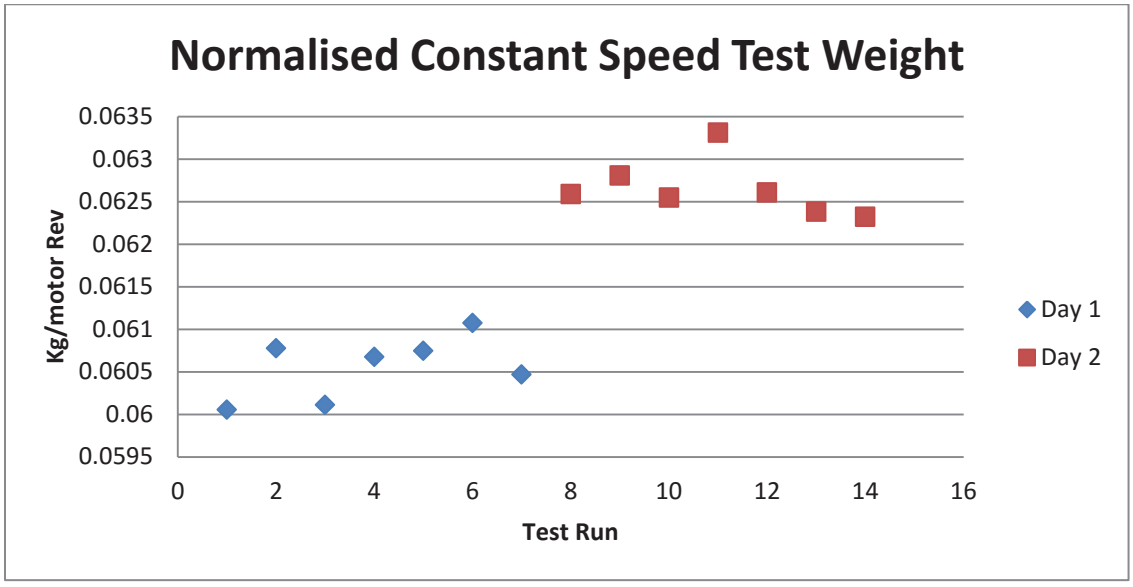


Figure 78: The normalised constant speed test results

Table 30: The normalised constant speed test results

	Kg/(motor Revolution)	
Run Number	**Day 1**	**Day 2**
1	0.060054	0.062587
2	0.060774	0.062805
3	0.060107	0.062547
4	0.060673	0.063308
5	0.060745	0.062606
6	0.061073	0.062382
7	0.060467	0.06232

A clear discrepancy can be seen between the two days. This is most likely due to moisture retention in the seed as it was left sitting in the hopper for 6 days between tests. This shows that the weight of seed applied can vary for a given seed. This variation is relatively small, with a range of 3.25 g/rev. Further investigation should be carried out to determine if this variation is significant. This will influence how often the calibration process needs to be carried out.

Variable ground speed test

The ground speed was changed over time to test the responsiveness of the seed rate control. The first test was conducted to observe the motor response over for a wide range of speeds. The ground speed was stepped down from 13.5 km/hr, and back up once the motor became unstable. The seeding rate for this test was 50 kg/ha. The expected motor speed was calculated using the information available and compared to the actual motor speed and the ground speed. The results of this test can be seen in Figure 77.

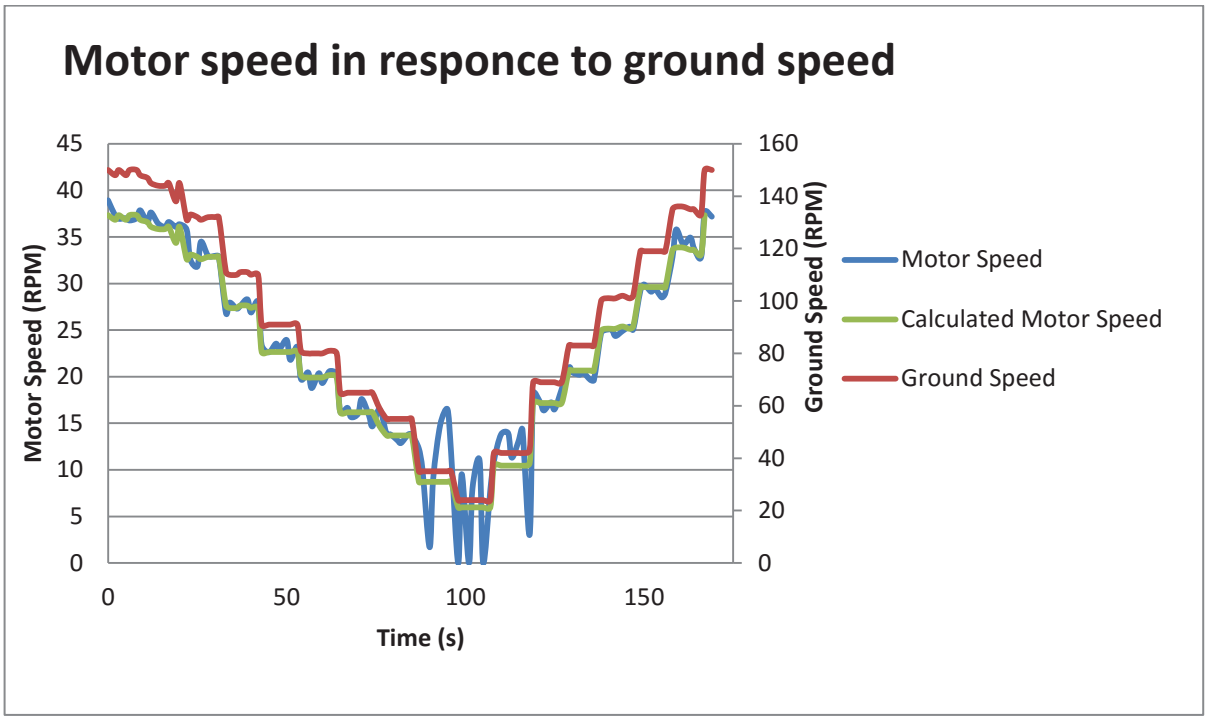


Figure 79: Motor speed in response to ground speed

The motor cannot run at speeds lower than 10 RPM. This caused the motor control loop to become unstable. Currently, motor speed alarms can be used within the user interface to notify the user that the motor speed is low. A limit should be put into place to prevent the motor from becoming unstable. A motor speed range indicator should be incorporated to indicate the motor speed and the limits. The motor speed converges quickly to the new set point, showing a good response to the ground speed change. The oscillation seen in the motor speed is within the acceptable range, but could be improved by further tuning the PID coefficients.

The second test introduced random variation in the ground speed between 40 and 20 RPM. This range covers 60-20 % of the motors usable speed range (~60-10 RPM).

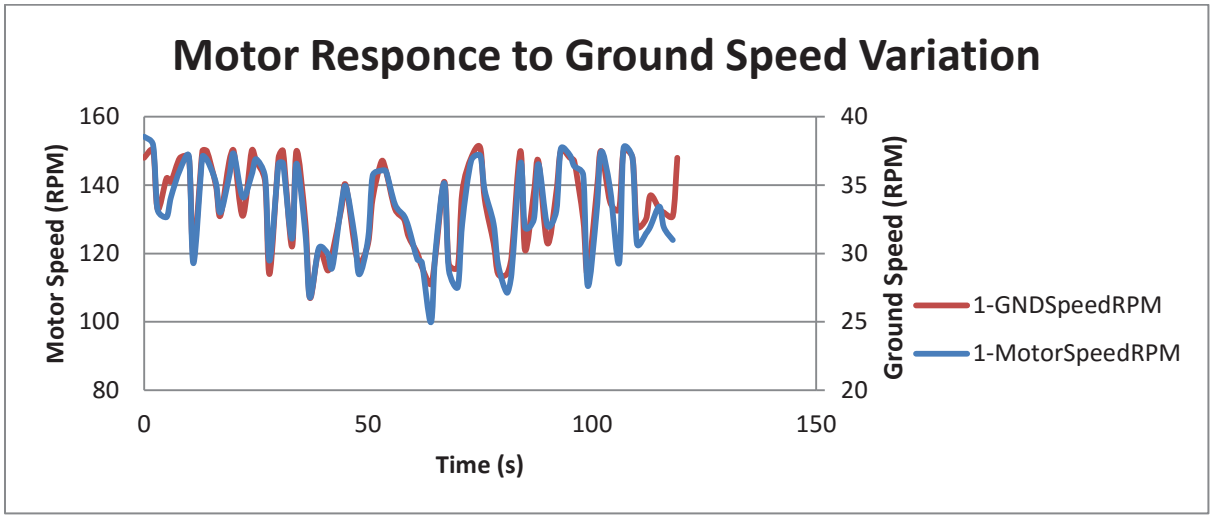


Figure 80: Motor response to ground speed variation

The control system showed a fast response to the random change in ground speed. The monitored/predicted seed weight of this test was 4.12 kg. The measured seed weight was 4.18 kg. This difference is not significant and well within the accepted error. This data can be seen in Appendix 7, Table 34. The motor speed error is within the acceptable range, with the variation being reasonably consistent across the different motor speeds.

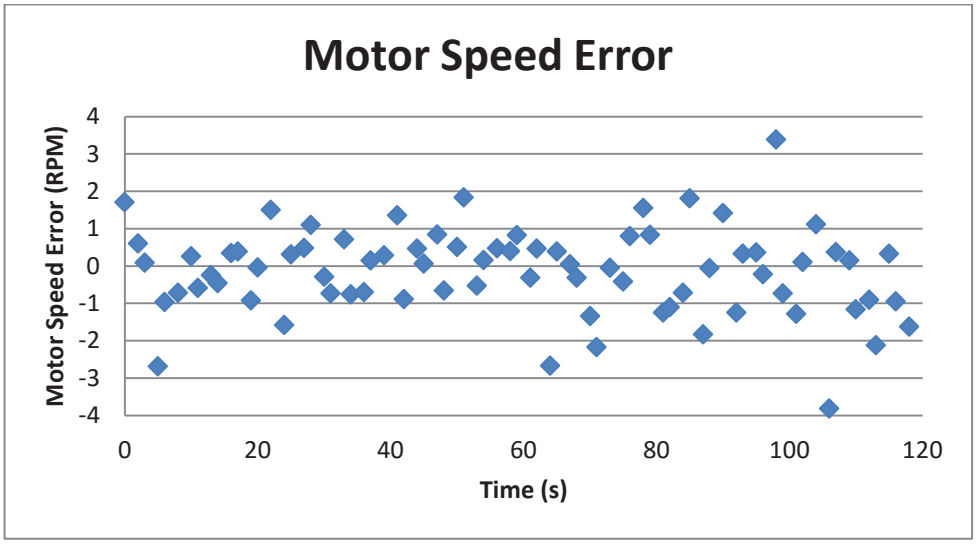


Figure 81: Difference between Motor Speed and the calculated motor speed

5.4 Testing Conclusions

The UPS implementation was successful in: filtering the main power; protecting the circuitry from harmful transient noise; delaying system shutdown once the main power is disconnected; signalling the SBC that the power is disconnected to shut down safely; allowing the shutdown to be override using either a switch, or a keep-alive signal from the SBC.

To improve the current UPS design, the UPS shutdown time should be reduced from 51.4 s to approximately 20 s. The current draw from the battery when the system is off is much lower than predicted. Based on the current draw measured (2.37 mA), the battery should remain charged for 21 days. The battery should be disconnected using a switch for if the system is not being used for longer than 21 days. A larger resistor should be used to limit the current through the voltage reference. Replacing the 10 kΩ resistors with a 100 kΩ resistor should almost double the battery life time to approximately 40 days. It is intended that the UPS be redesigned using a microcontroller or the SBC in the future.

PID coefficients were determined for responsive motor control using the manual tuning method. Kp, Ki and Kd were set to 0.14, 0.04 and 0.06 respectively. These values are specific to the motors used. The motor gearbox speed range is between 10 and 62.7 RPM.

The GrassFarmer seeder was an ideal platform to test the ground sensitive seed application functionality. The results from this machine should scale well to a larger three hopper 8128 Seeder as the seeding mechanisms are fundamentally the same. During the initial installation and system setup, a number of bugs were discovered and fixed that could not be detected in the lab.

The calibration tests showed that the seed feed rate has slight random variation over time. The variation observed with rye-clover seed gave a mean of 62.4 g per revolution, and standard deviation of 0.431 g per revolution. This variation is independent of motor speed and the number of calibration rotations. This variation was used to determine whether the variation seen in the control is reasonable. The weight per revolution of Rye-clover showed

an increase over a period of six days. This was most likely due to moisture retention. This means that, for more accurate seed application, the calibration process should be carried out for each job.

The predicted weight of seed applied during the constant ground speed test was well within the variation expected from the intrinsic seeding variation. The predicted weight provided a reasonable estimate of the seed applied.

The variable ground speed tests showed a good control response across the operable motor speed range, and to random speed changes. The predicted and measured seed weight was very close together, with a difference of 60g. The motor speed fluctuation should not affect the predicted seed weight, as it is calculated by the number of motor rotations and the application weight per motor revolution.

6.0 Conclusions and Future Improvements

This thesis covers the development of a generic versatile electronic monitoring and control platform using a flexible modular framework. This platform features modularity and configurability in both software and hardware.

The developed system architecture successfully demonstrated the versatility, customizability and expandability. The three main components (the master, the website and the slave network) synergized well, providing a stable platform for complex monitoring and control. The configurability available provides the ability to modify the system to different applications. The platform was successfully configured and fitted to an agricultural seed drill, which provided responsive ground speed dependent seed rate control. This implementation requires complex coordinated control based on inputs from many sources. This aligned well with the requirements of a universal monitoring and control system.

The developed system architecture allows new slave hardware to be developed, utilizing the packet management class to communicate with the master over the RS-485 bus. Once the master is notified of a new device, it will: connect to it; download the data stream description information; and allow new modules instances to be created, utilizing the new data streams. Master slave arrangement allows for expandable modular hardware as well as distributed processing and system customization.

The modular software architecture developed provides the foundation for implementing complex control tasks, utilizing information from the attached slave devices, the database, other module instances and the user interface. The master software architecture allows for simple creation of new module templates to handle new tasks and datatypes. The slave software architecture has been developed to make it simple to create new slave devices, with new tasks and datatypes. The communication protocol between the master and slave devices allows for multi-variable, bidirectional communication that is strictly regulated by the master device. Utilizing the packets command byte, each data stream can handle up to 250 different variables or tasks. The website provides the user with the means to manage and create module instances. The website back end handles module display creation, removing the need for designing a display box for each new module type. It is still possible to create custom display boxes if desired. The system flexibility allows for the creation of new tasks that can handle: new sensor/actuator types, new data types, and complex coordinated monitoring/control functions.

The software architecture performed well with its application on the GrassFarmer spreader, providing reliable and responsive seed rate control. The system configuration developed can be scaled to provide seed rate control across and monitoring across Reese Engineering's product range.

In applying the system as a seed rate controller, many environmental challenges had to be addressed. The developed system provides protection against water, humidity, and dust. Further development should be undertaken to provide reliable protection against heat and vibration. The Uninterruptible Power Supply (UPS) designed provided safe system shutdown functionality and protection from the transient noise of the tractor alternator. If the main

power is reconnected between the system shutdown sequence and the UPS power off, the system will remain off. To restart, the power must be disconnected for more than 60 seconds. This is a fundamental flaw in the UPS design. To fix this, the UPS should implement a state based shutdown sequence, by either using the main system processor or a microcontroller.

The work detailed in this report provides the core system framework for creating a generic monitoring and control system that is capable of providing a high level of specialised control. The architecture exhibited a high level configurability, versatility and flexibility. The project is currently in the prototyping phase, with many improvements to be implemented before a viable commercial product can be manufactured. The most notable improvements are listed in section 6.1.

6.1 Future Improvements

While this system framework exhibits a high degree of customisability and versatility, there are still aspects that require further development. Development is currently driven largely by its application towards farming machinery. From this, there are two main categories of future development: refining the platform for commercial use on farm machinery; and applying the generic framework to new applications.

The current platform needs to be refined for commercial use on farming equipment. This involves:

- Adding software functions to provide downloadable job reports and finish integrating the website view tabs.
- Develop a simple web based front end to configure the Wi-Fi password, SSID and channel, allowing for multiple standalone platforms in a given region.
- Improving the aesthetics of the user interface.
- Improving the platforms structural design to withstand the working environment, specifically for heat and vibration.
- Revising the slave hardware for the new motors and redesign the UPS.
- Revising the systems construction for ease of manufacture.
- Conducting robustness tests of the system in its working environment.
- Expand the library of module handlers to make it easier to implement on other applications.
- Expand the slave class collection to expand the platforms monitor and control tasks.
- Explore alternative communication mediums.

Future work to the generic framework will largely consist of incorporating internet security and adapting the system to provide management of new applications beyond farming machinery. This will build a diverse module library and refine the framework for a large variety of applications. A wrapping interface will be developed to provide accessibility to multiple systems. This will provide cloud storage and coordinated management of many remote systems, simultaneously in real time. The slave implementation will be revised to allow for more convenient hardware customisability and construction. This aims to make it easier to repurpose the system for handling a new task. The slave tasks will be separated into

individual hardware blocks, with on-board drivers. These blocks can be attached to the slave and the drivers transferred for configuration and control.

References

[1] J. W. Mellor, The economics of agricultural development., 1966.

[2] F. J. Pierce and P. Nowak, "Aspects of Precision Agriculture," *Advances in Agronomy,* vol. 67, pp. 1-85, 1999.

[3] J. Pretty, "Agricultural sustainability: concepts, principles and evidence," *Philosophical Transactions of the Royal Society B: Biological Sciences,* vol. 363, no. 1491, pp. 447-465, February 2008.

[4] P. Easterlund, "John Deere 4560," TractorData, [Online]. Available: http://www.tractordata.com/farm-tractors/000/1/5/155-john-deere-4560.html. [Accessed 2 May 2016].

[5] C. Füßler, "The Intelligent Tractor," *Frankfurter Allgemeine Sonntagszeitung,* 6 April 2014.

[6] John Deere, "2013 News Releases and Information," John Deere, 2013. [Online]. Available: https://www.deere.com/en_INT/our_company/news_and_media/press_releases/2013/august/john_deere_introduces_new_7r_and_8r_series_tractors.page. [Accessed 2 May 2016].

[7] H. Willer, Y.-M. Minou and N. Sorensen, The World of Organic Agriculture: Statistics and Emerging Trends 2008, Earthscan, 2010.

[8] Ministry of Primary Industries, "Growing Exprots," Ministry of Primary Industries, 13 May 2016. [Online]. Available: http://www.mpi.govt.nz/exporting/overview/growing-exports/. [Accessed 21 May 2016].

[9] Statistics New Zealand, "What New Zealand actually does for a living: from manufacturing to a services-oriented economy," Statistics New Zealand, April 2012. [Online]. Available: http://www.stats.govt.nz/browse_for_stats/economic_indicators/NationalAccounts/Contribution-to-gdp.aspx. [Accessed 29 April 2016].

[10] G. J. da Silveira, F. S. Fogliatto and D. Borenstien, "Mass customization: Literature review and research directions," *International Journal of Production Economics,* vol. 72, no. 1, pp. 1-13, 30 June 2001.

[11] F. S. Fogliatto, G. J. da Silveira and D. Borenstein, "The mass customization decade: An updated review of the literature," *International Journal of Production Economics,* vol. 138, no. 1, pp. 14-25, July 2012.

[12] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey," *Computer Networks,* vol. 54, no. 15, pp. 2787-2805, 28 October 2010.

[13] L. Mainetti, L. Patrono and A. Vilei, "Evolution of wireless sensor networks towards the Internet of Things: A survey," in *Software, Telecommunications and Computer Networks (SoftCOM)*, Split, 2011.

[14] H. Ramamurthy, B. S. Prabhu and R. Gadh , "Reconfigurable Wireless Interface for Networking," pp. 215-229, September 2004.

[15] K. Al Nuaimi, M. Al Nuaimi, N. Mohamed, I. Jawhar and K. Shuaib, "Web-based wireless sensor networks: a survey of architectures and applications," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, 2012.

[16] W. Zhang, G. Kantor and S. Sanjiv, "Integrated wireless sensor/actuator networks in an agricultural application," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004.

[17] A. J. Scarlett, "Integrated control of agricultural tractors and implements: a review of potential opportunities relating to cultivation and crop establishment machinery," *Computers and electronics in agriculture,* vol. 30, no. 1-3, pp. 167-191, February 2001.

[18] G. Sen Gupta, R. D. Weir and D. G. Bailey, "Implementation of a framework to integrate sensors," *International Journal of Intelligent Systems Technologies and Applications,* vol. 3, pp. 4-19, 2007.

[19] N. Zhang, M. Wang and N. Wang, "Precision agriculture—a worldwide overview," *Computers and Electronics in Agriculture,* vol. 36, no. 2-3, pp. 113-132, November 2002.

[20] R. E. Lütticken, "evelopment of an internet-based communication and information network to progress the implementation of Precision Agriculture," in *Proceedings of the 5th International Conference on Precision Agriculture*, Bloomington, MN, USA, 2000.

[21] International Organization for Standardization, "Tractors and machinery for agriculture and forestry — Serial control and communications data network," 15 06 2007. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:11783:-1:ed-1:v1:en. [Accessed 15 05 2016].

[22] T. Oksanen, P. Suomi, A. Visala and H. Haapala, "ISOBUS compatible implements in the project AGRIX," in *Precision Agriculture '05*, J. V. Stafford, Ed., 2005, pp. 565-572.

[23] G. Steinberger, M. Rothmund and H. Auernhammer, "Mobile farm equipment as a data source in an agricultural service architecture," *Computers and Electronics in Agriculture,* vol. 65, no. 2, pp. 238-246, 2009.

[24] P. Suomi and T. Oksanen, "Automatic working depth control for seed drill using ISO 11783 remote control messages.," *Computers and Electronics in Agriculture,* vol. 116, pp. 30-35, 2015.

[25] D. D. Flamme, A. Orbach, P. W. Haack and E. D. Jacobson, "Modular agricultural implement control system". USA Patent US 6070538 A, 6 June 2000.

[26] D. E. Campbell, "A simple, reliable, and expandable monitor and control system for gas-exchange analysis," *Analytical Biochemistry,* vol. 105, no. 1, pp. 287-290, 1980.

[27] W. G. Rado and S. S. Devlin, "Control system and method utilizing generic modules". United States of America Patent US5508689 A, 16 April 1996.

[28] M. J. Kiani, A. Al-ali, M. O'reilly, P. R. Jansen, N. E. Barker, A. Garfio and A. Sampath, "MODULAR PATIENT MONITOR". United States of America Patent 14/733781, 10 December 2015.

[29] J. Geiwitz, "Programmable and expandable building automation and control system". United States of America Patent US 10/830,653, 28 April 2005.

[30] R. O. Launey, P. A. Grendler, D. L. Packham, J. M. Battaglia and H. E. Levine, "Expandable home automation system". United States of America Patent US 5086385 A, 31 January 1989.

[31] R. J. Schwarzbach, M. S. Keeler, R. J. Cavaiani and M. K. Chapman, "Appliance control system". United States of America Patent US 4418333 A, 8 June 1981.

[32] K. Mikhaylov and M. Huttunen, "Modular wireless sensor and Actuator Network Nodes with Plug-and-Play module connection," in *IEEE SENSORS 2014 Proceedings*, Valencia, 2014.

[33] Topcon Precision Agriculture, "X30," Topcon Precision Agriculture, [Online]. Available: http://ag.topconpositioning.com/ag-products/guidance-systems/x30-console. [Accessed 16 5 2016].

[34] Bourgault, "BOURGAULT Product Catalogue 2016," [Online]. Available: http://www.bourgault.com/Portals/0/SiteAssets/Bourgault_Product_Catalogue_2016_lo_res%20_1.pdf. [Accessed 16 5 2016].

[35] Topcon, "Seeding & Planting," Topcon, [Online]. Available: https://www.topconpositioning.com/agriculture/seeding-planting. [Accessed 17 5 2016].

[36] Case IH, "Advanced Farming Systems (AFS)," Case IH, [Online]. Available: http://www.caseih.com/northamerica/en-us/products/advanced-farming-systems. [Accessed 16 5 2016].

[37] Farmscan Ag, "Seed Rate Controller 7500," Farmscan Ag, 2015. [Online]. Available: http://www.farmscanag.com/seed-rate-controller-7500/. [Accessed 16 5 2016].

[38] Data Management Software, "Data Management Software," Precision Farming Dealer, 6 November 2015. [Online]. Available: https://www.precisionfarmingdealer.com/articles/1732-data-management-software. [Accessed 16 February 2016].

[39] Linux Wireless, "Hostapd Linux documentation page," Linux Wireless, [Online]. Available: http://wireless.kernel.org/en/users/Documentation/hostapd. [Accessed 21 August 2014].

[40] S. Kelley, "Dnsmasq," Simon Kelley, [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/doc.html. [Accessed 21 August 2014].

[41] Python Software Foundation, "Python," Python Software Foundation, 2014. [Online]. Available: https://www.python.org/. [Accessed 12 September 2014].

[42] Oracle Corporation, "MySQL - The world's most popular open source database," Oracle Corporation, 2014. [Online]. Available: http://www.mysql.com/. [Accessed 17 June 2014].

[43] A. Dustman, "MySQLdb User's Guide," Andy Dustman, [Online]. Available: http://mysql-python.sourceforge.net/MySQLdb.html. [Accessed 12 June 2014].

[44] The Apache Software Foundation, "Apache HTTP Server Project," The Apache Software Foundation, 2014. [Online]. Available: http://httpd.apache.org/. [Accessed 12 June 2014].

[45] M. Rouse, "LAMP (Linux, Apache, MySQL, PHP)," [Online]. Available: http://searchenterpriselinux.techtarget.com/definition/LAMP. [Accessed 12 June 2014].

[46] Python Software Foundation, "Data Structures," Python Software Foundation, [Online]. Available: https://docs.python.org/2/tutorial/datastructures.html. [Accessed 2 March 2015].

[47] BeagleBoard.org Foundation, "BeagleBone Black," BeagleBoard.org Foundation, 22 August 2014. [Online]. Available: http://beagleboard.org/black. [Accessed 12 April 2014].

[48] Raspberry Pi Foundation, "Raspberry Pi," Raspberry Pi Foundation, 2014. [Online]. Available: http://www.raspberrypi.org/. [Accessed 16 March 2014].

[49] Pololu, "HP Motor with 48 CPR Encoder for 25D mm HP Metal Gearmotors (No Gearbox)," Pololu, 2014. [Online]. Available: www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00000240.pdf.

[Accessed 27 September 2014].

[50] STMicroelectronics, *L298 - DUAL FULL-BRIDGE DRIVER,* Italy: STMicroelectronics, 2000.

[51] Beagleboard.org, "BeagleBone: open-hardware expandable computer," Beagleboard.org, 2014. [Online]. Available: http://beagleboard.org/Support/bone101. [Accessed 2 September 2014].

[52] D. Molloy, "GPIOs on the Beaglebone Black using the Device Tree Overlays," derekmolloy.ie, 12 June 2013. [Online]. Available: http://derekmolloy.ie/gpios-on-the-beaglebone-black-using-device-tree-overlays/. [Accessed 29 September 2014].

[53] "jQuery mobile - A Touch-Optimized Web Framework," The jQuery Foundation, 2014. [Online]. Available: http://jquerymobile.com/. [Accessed 11 September 2014].

[54] The JQuery Foundation, "jQuery - write less, do more," The jQuery Foundation, 2014. [Online]. Available: http://jquery.com/. [Accessed 11 September 2014].

[55] Adobe PhoneGap Enterprise, "Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript," Adobe PhoneGap Enterprise, 2014. [Online]. Available: http://phonegap.com/. [Accessed 2 November 2014].

[56] M. Soltero, J. Zhang and C. Cockril, "RS-422 and RS-485 Standards Overview and System Configurations," Texas Instruments, 2001.

[57] National Electrical Manufacturers Association, "ANSI/IEC 60529-2004: Degrees of Protection Provided by Enclosures (IP Code)," National Electrical Manufacturers Association, 2004.

[58] L. Upton, "Cast or Epoxy Potted Packaging," Raspberry Pi, 14 September 2011. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=7&t=740. [Accessed 12 August 2016].

[59] Littlefuse, "Suppression of Transients in an Automotive Environment," Littlefuse, 1999.

[60] R. Regensburger, "Active High-Voltage Transient Protectors Trump Conventional Approaches in Automotive Electronics," Maxim Integrated, 20 June 2008. [Online]. Available: https://www.maximintegrated.com/en/app-notes/index.mvp/id/4240. [Accessed 15 September 2015].

[61] The Raspberry Pi Foundation, "FAQS," The Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.org/help/faqs/. [Accessed 12 November 2015].

[62] PowerStream Technology, "Lead Acid Battery Charging Basics and Chargers," PowerStream Technology, [Online]. Available: http://www.powerstream.com/SLA.htm. [Accessed 12 November 2015].

[63] Broadcom Corporation, "BCM2835 ARM Peripherals," Broadcom Corporation, Cambridge, 2012.

[64] Arduino, "Introduction to Arduino," Arduino, [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 12 March 2015].

[65] Arduino, "Arduino Pro Mini," Arduino, [Online]. Available: https://www.arduino.cc/en/Main/arduinoBoardProMini. [Accessed 15 November 2015].

[66] Motion Dynamics, "Motion Dynamics," Motion Dynamics, [Online]. Available: https://www.motiondynamics.com.au/. [Accessed 1 November 2015].

[67] M. Grassi, "12V-24V High-Current Motor Speed Controller," *Siliconchip,* pp. 30-39, March 2008.

[68] S. Kemp, "Making scripts run at boot time with Debian," Debian Administration, 11 October 2004. [Online]. Available: https://debian-administration.org/article/28/Making_scripts_run_at_boot_time_with_Debian. [Accessed 28 November 2015].

[69] ATMEL, "ATMEGA328 Datasheet (PDF) - ATMEL Corporation," [Online]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/392243/ATMEL/ATMEGA328.html. [Accessed 25 November 2015].

[70] PIGHIXXX Software Electronics & More, "Pro Mini Pinout," [Online]. Available: http://www.pighixxx.com/test/portfolio-items/pro-mini/. [Accessed 22 July 2015].

[71] Atmel, "ATmega48A/PA/88A/PA/168A/PA/328/P [Datasheet]," Atmel, 2015.

[72] K. Shirriff, "Secrets of Arduino PWM," 2009. [Online]. Available: http://www.righto.com/2009/07/secrets-of-arduino-pwm.html. [Accessed 23 September 2015].

[73] croston, "Raspberry-gpio-python - A Python module to control the GPIO on a Raspberry Pi," croston, 2014. [Online]. Available: http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/. [Accessed 12 March 2015].

[74] J. Cooper, "Using the Adafruit_BBIO Library," Adafruit, 2014. [Online]. Available: https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library. [Accessed 20 March 2014].

Appendix

Appendix 1: Existing Variable Seed Rate Controllers

X30 Specifications:

Interface:

- 4 X USB 2.0, all bootable, 500mA, 1 side, 1 rear, 2 harnessed
- 4 X RS-232 serial, 1 ea. RX/TX/CTS/RTS/GND 3, ea. RX/TX/GND only
- 1 JR45 Ethernet 100 Base-T, 100 Mbps, rear
- 4 X CANBUS (ISO11783 compliant)
- 4 X channel digital I/O
- 1 X channel analog in
- Bluetooth® module

Display:

- 12.1" (30.7 cm) touchscreen
- Projected capacitive touchscreen
- Enhanced sunlight readability
- 24 bit color
- 3D Graphics
- RAM mounting system

Processor:

- Intel chipset, 1.6GHz processor
- 1GB DDR2 533 MHz RAM
- 4GB compact flash (up to optional 32GB)
- COM-Express System-on-Module (SOM)

Input Voltage 9V-36V with voltage/polarity protection.

Internal N-MH Battery (2000mA/hr., 7.2V)

X20 variable seed rate controller

Figure 82: The X20 seed rate controller

Appendix 2: System Setup and Configuration

System image:

To install a new Raspbian system image using a windows computer, follow these steps.

Step 1: Downloads.

1. Download the latest version of Noobs from: www.raspberrypi.org/help/noobs-setup/
2. Download SDFormatter 4.0 from: https://www.sdcard.org/downloads/formatter_4/

Step 2: Setup SD Card.

1. Format the SD Card
    - Insert the SD Card into an SD reader.
    - Run "SDFormatter 4.0", select the correct drive and click format.
2. Copy all of the files from the "NOOBS_v?_?_?.zip" folder onto the SD Card.

Step 3: Set up the Operating System.

1. Safely eject the SD card and insert it into the Raspberry Pi.
    a. Connect a in a mouse, a keyboard, a monitor and network cable to the RPi. Then power it up.
2. when prompted, select "Raspbian" and click install. This will take some time.
    a. Select "US" as the Keyboard layout in the menu at the bottom of the screen.
    b. Click "OK" when the installation is complete.
3. A configuration menu should show up. Use the "TAB" button, the "ENTER"/"SPACE" button and the arrow keys to Navigate. [You can visit this menu any time using the command "raspi-config"]
    a. TIME ZONE: select "4 Internationalized Options" >> "Change Timezone". then select the corresponding time zone. ie. >> Pacific Ocean >> Auckland
    b. Boot into Command line [Optional]: select "3 Enable Boot to Desktop/Scratch" >> "Console Text console, requiring login (default)"
    c. Once finished, select "<Finish>", and then reboot.
4. Login with:
    a. Username: pi
    b. Password: raspberry
5. Step 3: Install System Dependencies.
    a. Update package lists.
    b. Run "sudo apt-get update"
6. Download and install Sysyem Dependencies.
    a. Run "sudo apt-get install samba apache2 mysql-server python hostapd dnsmasq python-MySQLdb libapache2-mod-python python-setuptools python-dev python-pip firmware-ralink"
    b. Set the mysql user name to "root" and the password to "neo"
    c. Run "sudo easy_install -U RPIO"
    d. Run "sudo pip install autobahn[twisted]"

Python:

1. Run “sudo apt-get install python”

Database:

MySQL and the python MySQL Library are installed using the following steps:

1. Run “sudo apt-get install mysql-server python-MySQLdb”
2. When prompted, set the user name and password.

To use the MySQLdb API in python, see: mysql-python.sourceforge.net/MySQLdb.html

To recreate the database, use the following steps:

1. Open Mysql using “sudo mysql -uroot -pneo” and execute the following:
    a. drop database gpio2;
    b. create database gpio2;
    c. \q
2. Run “sudo mysql -u root -pneo gpio2 < /var/www/Database/multi-io_setup.sql”

This method is useful for clearing the database. The SQL code in the “multi-io_setup.sql” file can be seen below.

```
DROP TABLE IF EXISTS `Irregular`;
CREATE TABLE `Irregular` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `dateTime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `name` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
  `val` FLOAT NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

DROP TABLE IF EXISTS `Regular`;
CREATE TABLE `Regular` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `dateTime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

To dump all of the database data to a file, use the following command:

“sudo mysqldump -u root -pneo gpio2 > /[location]/[file name].sql”

Apache:

To install and configure Apache:

1. Run “sudo apt-get install apache2”
2. To integrate MySQL with apache, Run “sudo apt-get install libapache2-mod-auth-mysql”

This will create a folder at “/var/www/” that will be used as the directory for hosting the webpages. The pages in this folder can be accessed through a web browser with a URL in the form [DEVICE IP]/[FILENAME].

Samba (optional):

Samba was used to access the Linux file system using a windows machine over the local network. This was useful for system development.

It is installed and configured using the following steps:

1. Run “sudo apt-get install samba”
2. In " /etc/samba/smb.conf”, add:

```
[ProjectFiles]
  comment = Project Files
  read only = no
  locking = no
  path = /var/www
  guest ok = yes
```

3. To enable read, write and execute access permission for the owner, group and others, run: “sudo chmod 777 /var/www”

This file system is accessed over the network using the device IP address.

Wi-Fi Access Point:

To download and setup the virtual Wi-Fi access point, implement the following using the terminal:

1. Run “sudo apt-get install hostapd dnsmasq”
2. In “/etc/dnsmasq.conf”, add:

```
# disables dnsmasq reading any other files like /etc/resolv.conf for nameservers
##no-resolv
# Interface to bind to
interface=wlan0
# Specify starting_range,end_range,lease_time
dhcp-range=10.0.0.3,10.0.0.20,12h
# dns addresses to send to the clients
##server=8.8.8.8
##server=8.8.4.4
```

3. In “/etc/hostapd/hostapd.conf”, add:

```
interface=wlan0
driver=nl80211
ssid=RaspberryPi
hw_mode=g
channel=11
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=3
wpa_passphrase=raspberry
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

4. In “/etc/default/hostapd”, add:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
RUN_DAEMON="YES"
```

5. In “/etc/network/interfaces”, add:

```
#allow-hotplug wlan0
iface wlan0 inet static
    address 10.2.2.2
    netmask 255.0.0.0
```

Python GPIO Library:

There are two main ways to interface with the GPIO pins on a single board computer. The first is creating file-type access to the GPIO. This requires reading and writing to the file system to provide the I/O interface. This can easily be interfaced through code, but cannot provide a high update speed. The second is to read and write directly to the registers allocated for GPIO. This is significantly faster than the file-type access method. The Raspberry Pi and the BeagleBone both have python libraries to provide the GPIO interface using the register method. These libraries make use of Pythons extensibility into C++ to provide faster access to the registers. On top of being able to write and read the Boolean value of a pin, these libraries provide software pin interrupts on rising and falling edges, PWM functions for the dedicated PWM pins and Analogue input on the dedicated ADC pins. These functions are important for providing the monitoring and control facilities. They also provide an interface for I2C, SPI and UART.

For Raspberry Pi, the library "RPi.GPIO" was used [73]. It is installed by:

1. Run "sudo apt-get install python-rpi.gpio"

To use this library see: http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/.

For the BeagleBone Black, the library "Adafruit_BBIO" was used [74]. It is installed by:

2. Run "sudo pip install Adafruit_BBIO"

To use this library, see: https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library.

Unfortunately, the Massey proxy would not allow access to the Python Package Index (pip). This meant that any pip packages needed to be installed at another location.

Appendix 3: Test Cape Pin Assignments

Table 31: Pin assignments for the beaglebone black test cape

			BeagleBone Black		External	Setup as:
	Name	Pin	Pin	Name		
L298	Current sence 2	1			GND	
	Output 1	2			Motor 1	
	Output 2	3			Motor 1	
	V+ for Motor	4			VCC	
	Input 1	5	P9_17	GPIO_30		Bool output
	Enable 1	6	P9_14	EHRPWM1A		PWM
	Input 2	7	p9_11	GPIO_60		Bool output
	Ground	8			GND	
	V+ for L298	9			5V	
	Input 3	10	P8_7	GPIO_45		Bool output
	Enable 2	11	P8_13	EHRPWM2B		PWM
	Input 4	12	P8_10	GPIO_44		Bool output
	Output 3	13			Motor 2	
	Output 4	14			Motor 2	
	Current sence 2	15			GND	
NRF24L01	GND	1			GND	
	VCC	2	P9_3 or 4	VDD_3v3		3.3V
	CE	3	P9_23	GPIO_49		Bool output
	CSN	4	P9_24	GPIO_15		Bool output
	SCK	5	P9_22	SPI0_SCLK		SCLK
	MOSI	6	P9_18	SPI0_D1		D1
	MISO	7	P9_21	SPI0_D0		D0
Motor 1	Motor GND	1			L298 O1	
	Motor VCC	2			L298 O2	
	Encoder GND	3			GND	
	Encoder VCC	4	P9_3 or 4	VDD_3v3		3.3V
	Encoder Ch1	5	P9_16	GPIO_31		Speed input
	Encoder Ch2	6	P9_15	GPIO_48		Speed input
Motor 2	Motor GND	1			L298 O3	
	Motor VCC	2			L298 O4	
	Encoder GND	3			GND	
	Encoder VCC	4	P9_3 or 4	VDD_3v3		
	Encoder Ch1	5	P8_11	GPIO_26		Speed input
	Encoder Ch2	6	P8_9	GPIO_46		Speed input
LED 1	positive	1	P8_19	EHRPWM2A		Bool output
	negative	2			R2	
LED 2	positive	1	P8_18	GPIO2_1		Bool output
	negative	2			R1	
Switch 1	Pin 1	1	P8_16	GPIO1_14		Bool input
	Pin 2	2			R4	
Switch 2	Pin 1	1	P8_14	GPIO0_26		Bool input
	Pin 2	2			R4	

Appendix 4: UPS Circuit Schematics and PCB's

UPS Prototype One

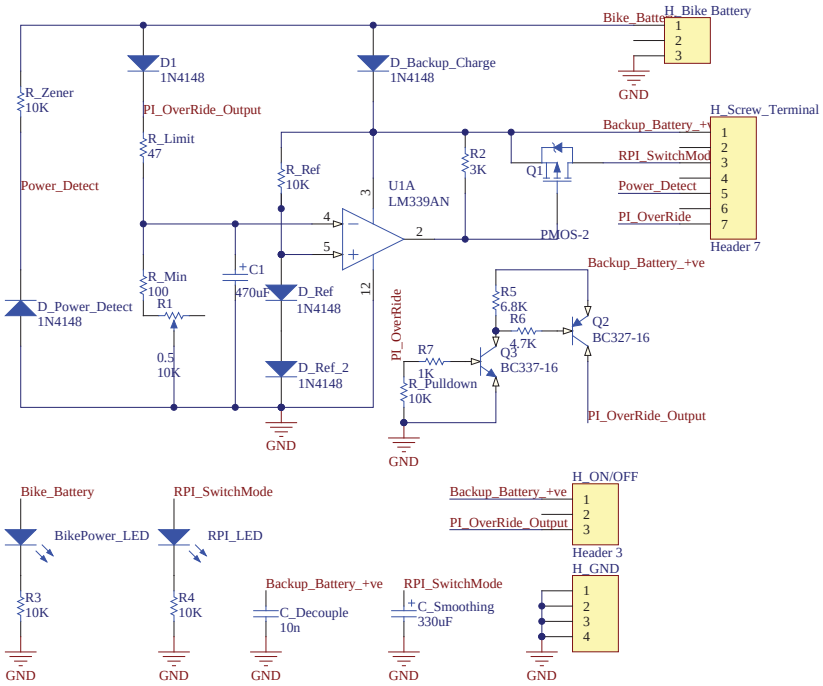


Figure 83: UPS Prototype One Circuit Schematic

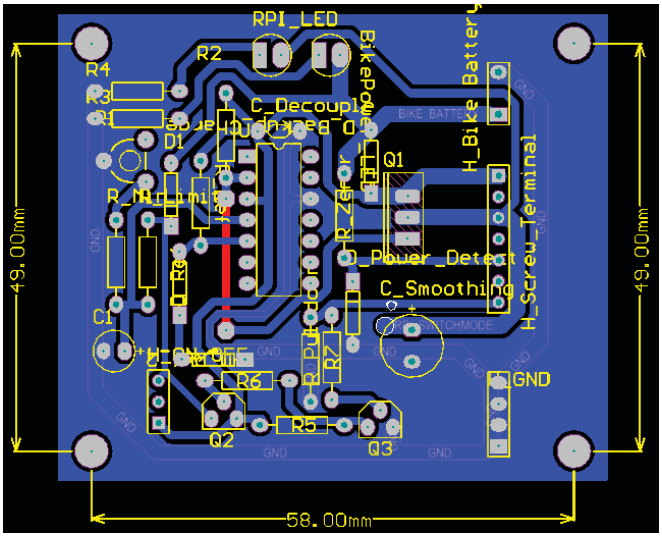


Figure 84: UPS Prototype One PCB Design

UPS Prototype Two

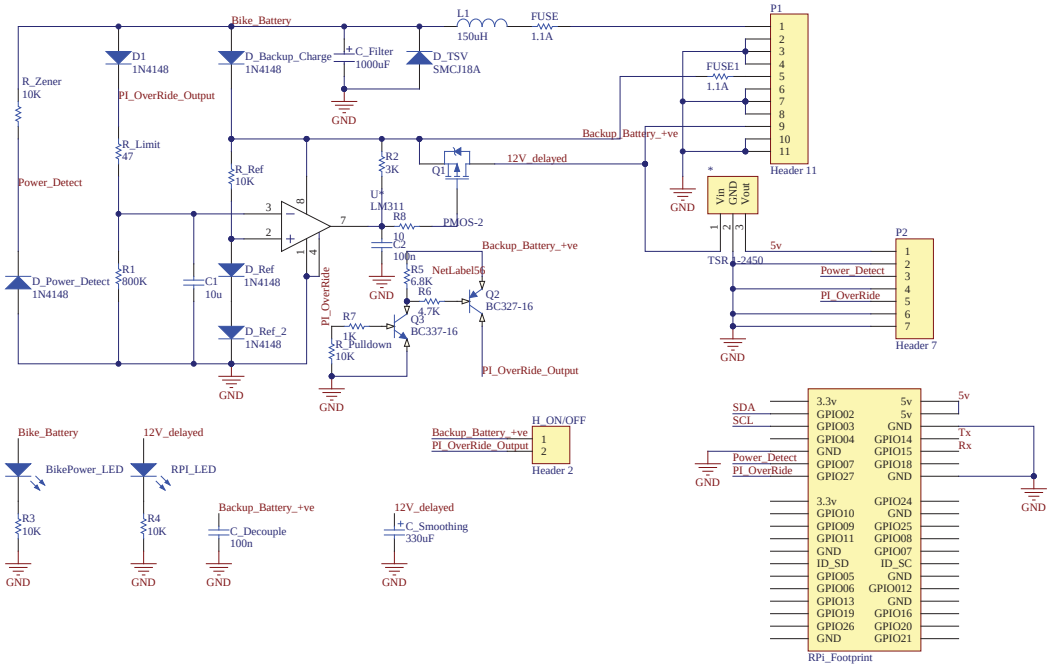


Figure 85: UPS Prototype Two Circuit Schematic

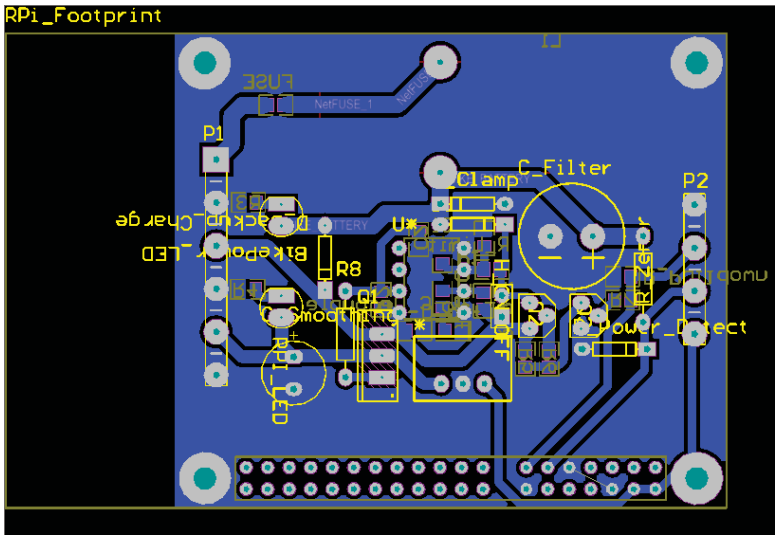


Figure 86: UPS Prototype Two PCB Design

Custom Motor Driver Schematic

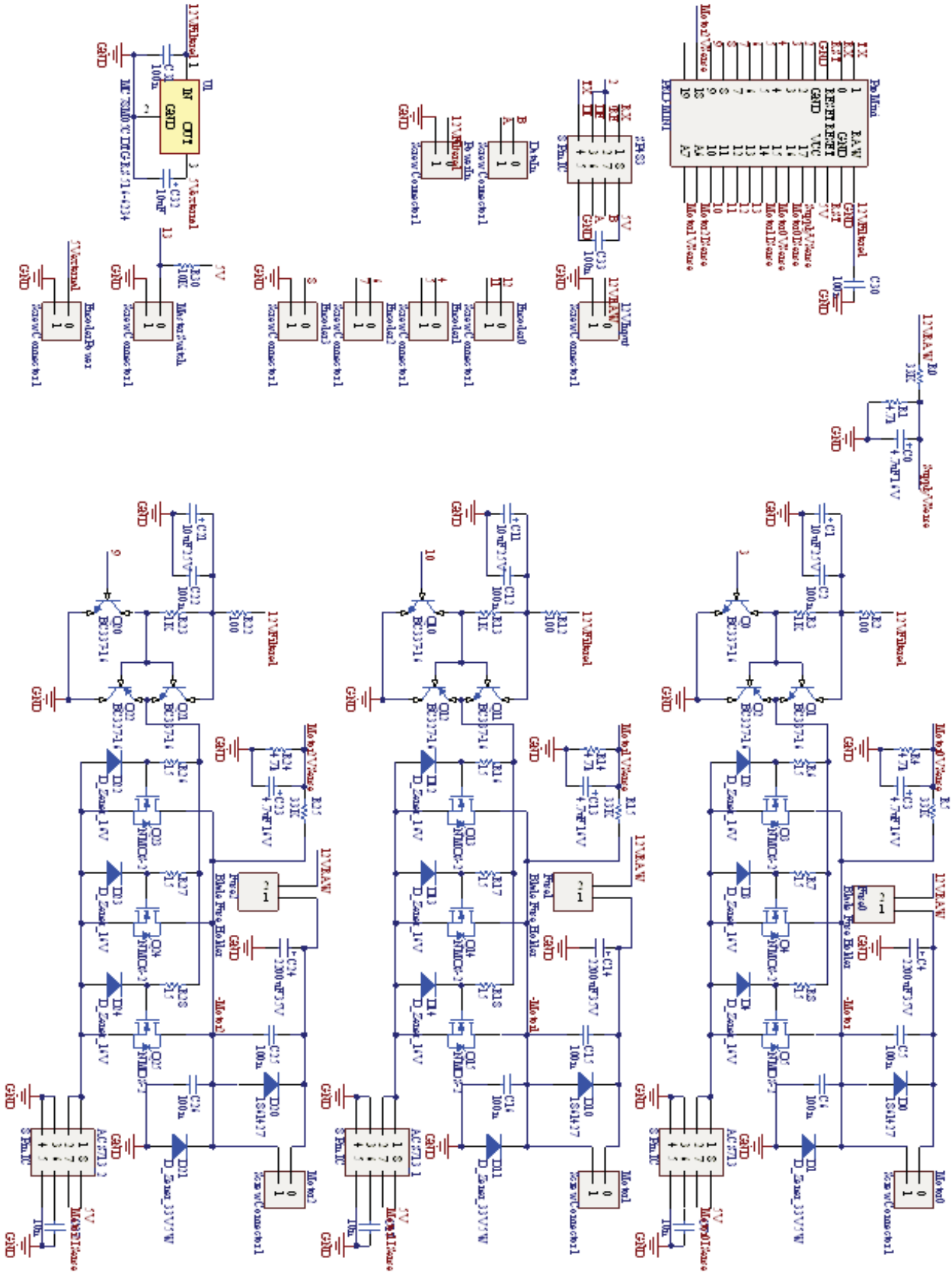


Figure 87: Custom Motor Driver Circuit Schematic

Appendix 5: Master and Slave Construction

The CAD model used to make the mounting plate for the monitor slave can be seen in Figure 86. The monitor slave connector diagram can be seen in Figure 87. The wiring diagram used can be seen in Figure 88. The final construction can be seen in Figure 89.

The CAD model used to make the mounting plate for the control slave can be seen in Figure 90. The control slave connector diagram can be seen in Figure 91. The wiring diagram used can be seen in Figure 92. The final construction can be seen in Figure 93.

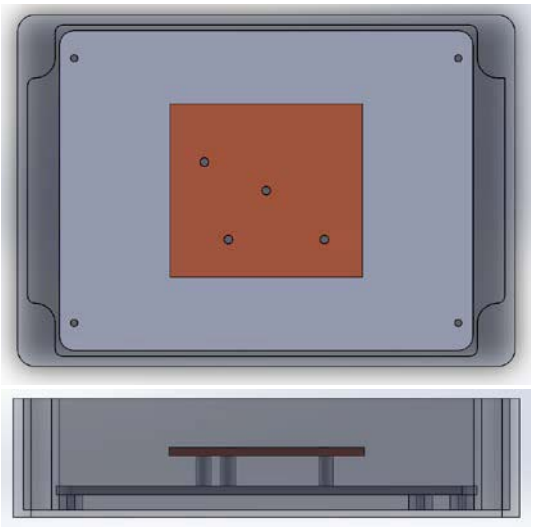


Figure 88: Monitor Slave Enclosure and mounting

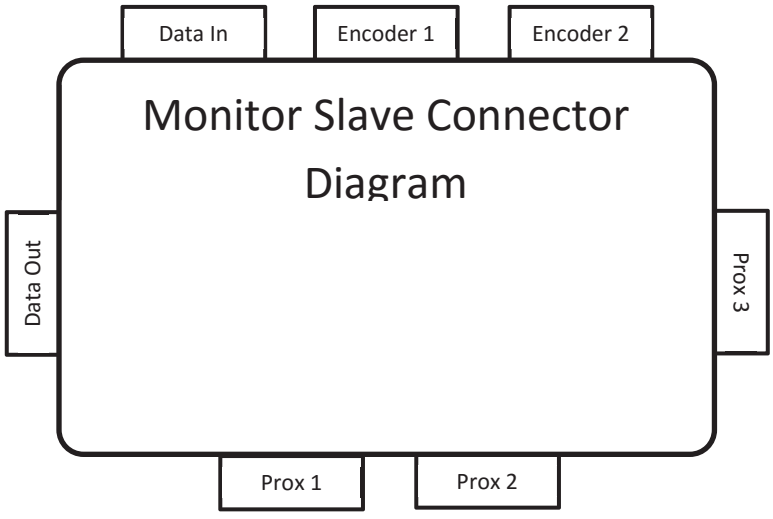


Figure 89: Monitor slave connector diagram

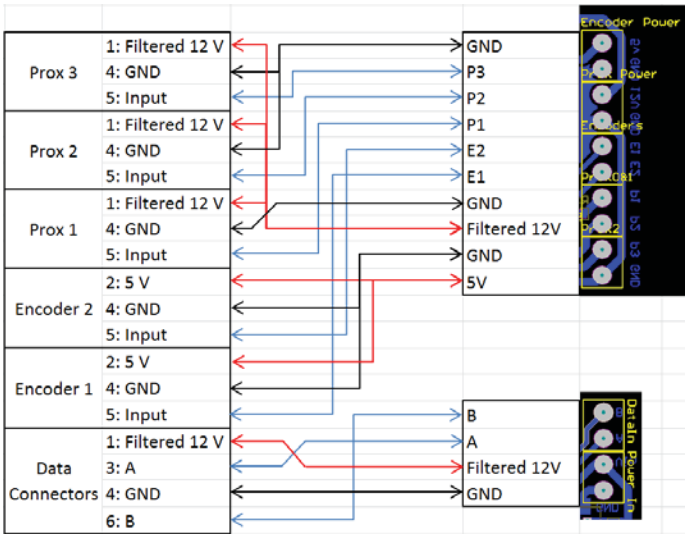


Figure 90: Monitor slave wiring diagram

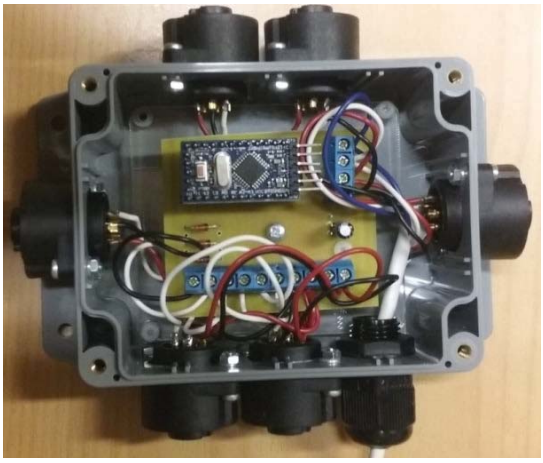


Figure 91: Monitor slave construction

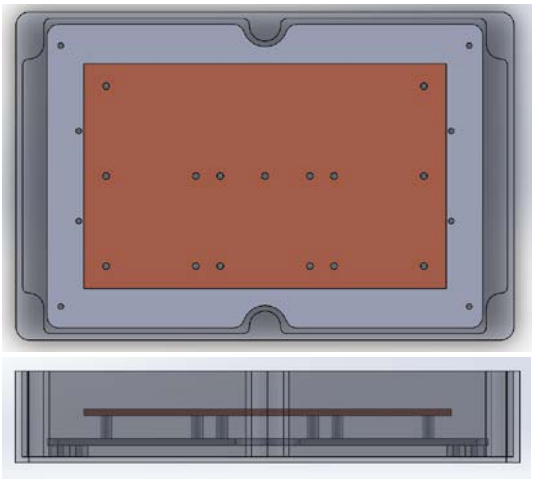


Figure 92: Control slave enclosure and mounting

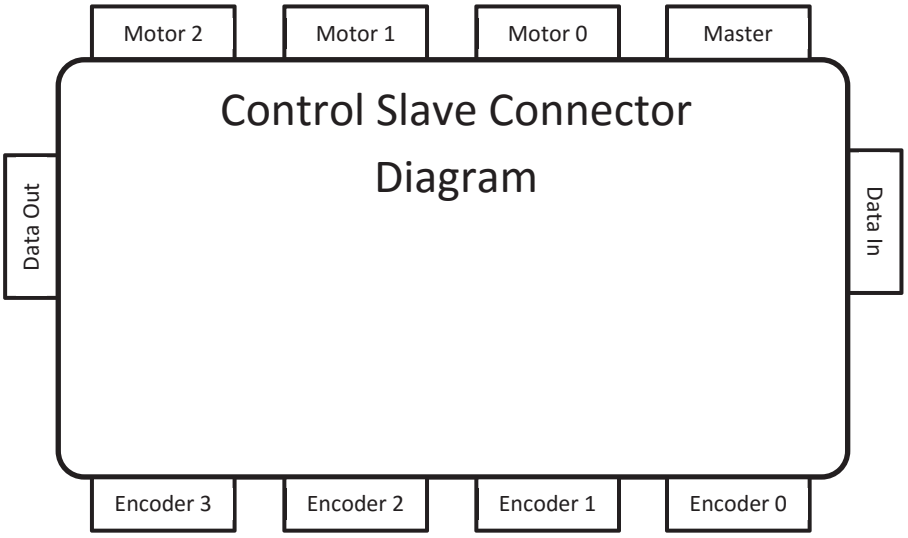


Figure 93: Control slave connector diagram

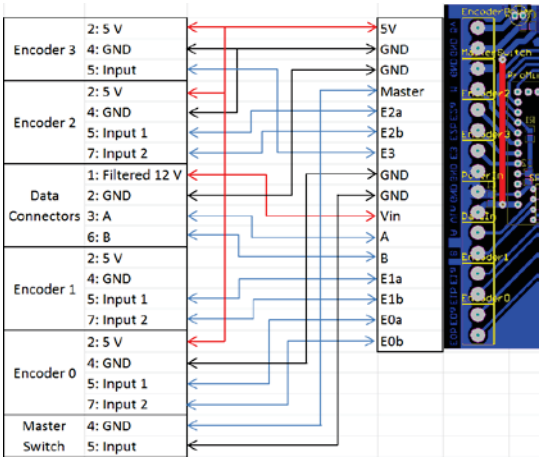


Figure 94: Control slave wiring diagram

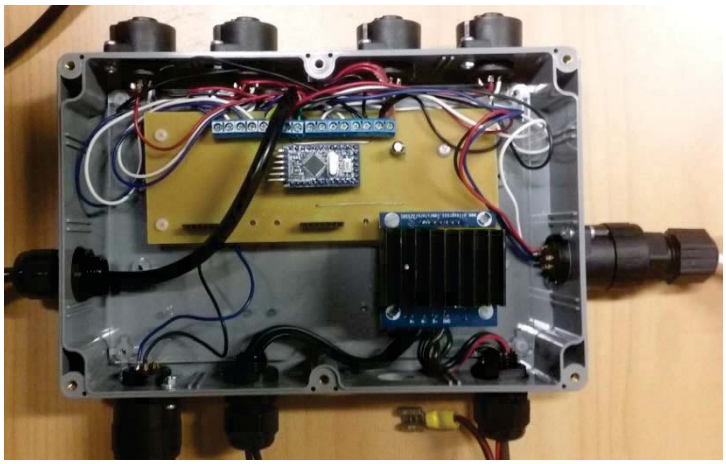


Figure 95: Control slave construction

Appendix 6: Variable Frequency Generator

The variable frequency generator outputs a square wave. This device can be controlled using a serial connection, or the POT mounted on the circuit board. By default, the duty cycle is 50% and the maximum frequency is 150 Hz, which simulates a ground wheel speed of 0-13.5 km/hr. The serial commands are detailed in the table below.

Table 32: Variable Frequency Generator Serial Commands

Command	Usage	Description
f	“f[x]\n” where [x] is the frequency.	Sets the output frequency.
d	“d[x]\n” where [x] is the duty cycle.	Sets the PWM duty cycle.
y	“y\n”	Switch PWM on.
n	“n\n”	Switch PWM off
u	“u[x]\n” where [x] is the max frequency.	Sets the maximum frequency for the POT.
i	“i\n”	Prints the current frequency and duty cycle.

The POT can be used to tune the frequency between 0 Hz and the defined maximum frequency. The constructed device can be seen in the figure below.

Figure 96: The Variable Frequency Generator

Appendix 7: Experimental Data

Table 33: Typical Motor speed error observed in the constant ground speed test

Time (s)	RPM Error	Time (s)	RPM Error	Time (s)	RPM Error	Time (s)	RPM Error	Time (s)	RPM Error
0	-0.11941	65	-0.20541	130	-0.96741	195	0.958446	260	-0.06755
2	0.802587	67	-0.78541	132	0.266587	197	0.256587	262	0.470587
3	-0.32941	68	-0.30341	133	0.788446	198	-0.28755	263	-0.65941
5	0.140587	70	-0.81541	135	2.026587	200	-0.12141	265	0.482587
7	1.140446	71	0.570446	136	0.070446	201	0.048587	266	-0.81834
8	-0.26141	73	0.352587	138	0.768587	203	-0.79941	268	0.812446
10	0.714446	75	1.942587	139	-0.40541	204	-0.37341	269	-0.09541
11	-0.41141	76	1.098446	141	-0.04355	206	0.478587	271	0.516587
13	0.490587	78	0.610587	143	-0.17541	208	0.880587	272	-1.62341
14	-0.68741	79	-1.77955	144	-0.42634	209	-0.06141	274	0.942587
16	-0.47941	81	0.802446	146	-0.56941	211	0.450587	276	0.724446
17	-0.36941	82	-0.49941	147	-0.33541	212	-1.12341	277	-0.22141
19	0.058446	84	0.046446	149	-0.68141	214	0.510587	279	0.572587
20	-0.46434	85	-0.15741	150	0.872587	215	0.928587	280	-0.77155
22	-1.44541	87	-0.18741	152	0.848587	217	-0.41541	282	-0.15541
24	-0.57741	88	0.602587	153	0.584446	218	0.716446	283	0.092446
25	0.051658	90	0.784587	155	-0.14541	220	-0.63541	285	0.234587
27	-0.59955	92	0.428587	157	0.030446	221	0.694446	286	0.659658
28	0.192587	93	-0.02941	158	-0.10755	223	0.600446	288	-0.15155
30	0.102587	95	0.440587	160	-0.17741	225	0.644587	289	-0.30341
31	-0.81034	96	-0.42355	161	-0.70955	226	0.630446	291	-0.27834
33	-0.52741	98	-0.26941	163	-0.68341	228	-0.27955	293	-0.21541
34	-0.31741	99	0.620587	164	-0.84941	229	0.578587	294	-0.44341
36	0.282587	101	-0.56141	166	1.234446	231	-1.05341	296	-0.41355
37	-0.77941	102	0.027658	167	0.456587	232	-0.09541	297	0.776587
39	-0.28555	104	0.628446	169	-0.56341	234	-0.20941	299	-0.09141
41	0.081658	105	-1.81834	170	0.642587	235	-1.12555		
42	0.890446	107	-0.36741	172	-0.15341	237	0.622587		
44	1.924587	109	0.670446	174	0.044446	238	0.514587		
45	-0.67741	110	-0.27941	175	-0.54741	240	-0.99434		
47	0.150446	112	-0.36941	177	-0.58941	242	0.840587		
48	-0.45941	113	0.560446	178	-0.91955	243	-1.03141		
50	1.032587	115	-1.18434	180	-1.46741	245	-0.14541		
51	0.746446	116	-0.46141	181	-0.60155	246	-0.50541		
53	0.206587	118	1.166446	183	0.800446	248	-1.17155		
54	0.722587	119	-1.27941	184	-0.87141	249	-0.65541		
56	0.186446	121	0.138446	186	1.098446	251	-0.91155		
58	0.718587	122	0.408587	187	-0.51541	252	-0.16755		
59	0.755658	124	-1.01541	189	-0.34541	254	-0.77834		
61	0.864587	126	-0.34741	191	-0.14155	255	0.328446		
62	0.490587	127	-0.79234	192	-0.80234	257	-0.54755		
64	-0.17941	129	-0.67141	194	0.626587	259	-0.73741		

Table 34: Difference between Motor Speed and the calculated motor speed observed in the ground speed test

Time (s)	Error RPM	Time (s)	Error RPM
0	1.704191	61	-0.31974
2	0.604329	62	0.461987
3	0.090158	64	-2.66936
5	-2.69222	65	0.389194
6	-0.96929	67	0.042709
8	-0.72181	68	-0.31494
10	0.260191	70	-1.34201
11	-0.58974	71	-2.1745
13	-0.24767	73	-0.04981
14	-0.46567	75	-0.4166
16	0.34764	76	0.793365
17	0.38202	78	1.549469
19	-0.92488	79	0.82985
20	-0.04367	81	-1.25415
22	1.50402	82	-1.10681
24	-1.58567	84	-0.71767
25	0.311122	85	1.809332
27	0.484709	87	-1.8345
28	1.09785	88	-0.05688
30	-0.28781	90	1.415469
31	-0.73967	92	-1.2465
33	0.708401	93	0.322329
34	-0.76167	95	0.358191
36	-0.70412	96	-0.21888
37	0.144368	98	3.386951
39	0.279332	99	-0.74015
41	1.354919	101	-1.2885
42	-0.88774	102	0.100329
44	0.460158	104	1.118296
45	0.06364	106	-3.82384
47	0.844401	107	0.366329
48	-0.65494	109	0.146191
50	0.505469	110	-1.16119
51	1.835365	112	-0.90505
53	-0.52688	113	-2.12757
54	0.151915	115	0.324158
56	0.476158	116	-0.94891
58	0.396951	118	-1.62598
59	0.819607		