University of Southern Queensland

Faculty of Health, Engineering, and Sciences

Airborne Observation Controller

A dissertation submitted by

Robert Schuster

In fulfilment of the requirements of

ENG4111 and 4112 Research Project

Towards the degree of

Bachelor of Engineering (Honours)(Electrical and Electronic)

Submitted November 2022

Abstract

Since the formation of NASA, airborne missions have been used to observe objects that are entering the earth's atmosphere. Airborne missions have the advantage that the aircraft can be deployed almost anywhere in the world. This makes the missions more cost effective than land or space-based observation systems. These systems need to be synchronised so that observations made on different equipment can be compared. Current observation systems generally use onboard timecode generators through a co-axial cable to synchronise the observation equipment. This project aims to determine the feasibility of combining multiple onboard systems into one portable solution designed around a Raspberry Pi.

A custom time synchronisation (CTS) code was written in C programming language and this was compared with the native network time protocol (NTP) on the Raspberry Pi. The performance of the two protocols was tested and compared. Power consumption was measured during both tests. A NAS was setup on the Raspberry Pi which used two USB devices in a RAID1 configuration and its performance was also tested.

The CTS was written so that the time was updated every second. It performed just as well as the NTP in all the tests. The NAS performed as expect with write speeds taking longer as the drives are mirrored in RAID1. With the power consumption observed, the ideal battery backup would be an inverter that can charge the batteries when connected to mains power.

When it comes to an airborne observation mission, either CTS or NTP would work. The benefit of the NTP is the potential to build it into more than just a time synchroniser. A RAID 1 configuration and using an inverter with car batteries will ensure the life of the system and the safety of the data collected. Further work should be done on the effectiveness of a wireless system to further reduce wire clutter. If a more accurate system is needed, research on the use of a GPS with better antennas should be done. The EMC of the system should also be tested and the effect of radiation at high altitudes.

University of Southern Queensland

Faculty of Health, Engineering, and Sciences

ENG4111 & ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland do not accept any responsibility for the truth, accuracy, or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitles "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and any other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Certification

I certify that the ideas, designs, experimental work, results, analyses, and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Robert Schuster

Student Number: [redacted]

Contents

LIST OF FIGURES

LIST OF TABLES

List of acronyms

USB - Universal serial bus

SSH - Secure shell protocol

NTP - Network time protocol

UDP - User datagram protocol

SNTP - Simple network time protocol

PTP - Precision time protocol

UTC - Coordinated Universal Time

ECC - Error correction code

RTC - Real time clock

CTS - Custom time synchronisation

GPIO - General purpose input output

NMEA - National Marine Electronics Association

1.0 Introduction

1.1 Background Information

Airborne observation missions started when the National Aeronautics and Space Administration (NASA) wanted to observe space without the cost of sending anyone into space. Missions began with small planes modified to carry observation equipment and are now large ex-commercial aeroplanes containing a 106-inch telescope. Airborne missions can get above 99% of the water vapour in the atmosphere, which hinders infrared cameras. In recent years, airborne observations have been used to observe space vehicles or debris entering the atmosphere. These observations can be compared to the expected result that simulations have made, and the parameters for the simulation updated. The possibility of a catastrophic re-entry is reduced with the simulation accuracy increasing. One of the most critical re-entries that will happen in roughly ten years is the International Space Station (ISS) de-orbit. This iconic spacecraft will likely be de-orbited in stages for better control, and multiple airborne missions will observe the process. The instrumentation that collects the measurement and photo data needs to be time-accurate so that data from different instruments can be compared. The most common way of syncing the instrumentation time in aircraft is using the onboard timecode generator. This timecode is transferred using a coaxial cable which adds to the already clustered workspace. A system that would reduce the number of wires and organise the acquired data could increase the number of instruments used. The system could also improve the data collection's reliability and provide some redundancy to ensure the data is safe (Grinstead et al. 2011; Jenniskens 2016; Löhle et al. 2017).

1.2 Project Aim

This project aims to design and build a portable management system for aircraft-borne observations. This system should be able to synchronise the time of all connected devices, record GPS data, record aircraft logs as needed, and manage data collected from connected devices. As this data is critical, battery and file storage redundancies should be used to enhance data storage reliability and ensure the integrity of all information stored.

1.3 Project Objectives

1. Determine appropriate requirements to support a network of instruments onboard the aircraft for an observation mission.
2. Determine an appropriate processor speed and memory size, and review currently available microcontrollers to use as the management system's core.
3. Review the network time protocol to design code that uses a custom time protocol for the chosen microcontroller to synchronise connected devices.

4. Investigate the use of GPS hardware and determine the feasibility of incorporating it into the microcontroller with the custom time protocol.
5. Test the microcontroller and connected devices for synchronisation accuracy.
6. Investigate the current and voltage needed to support the network of instruments and the ability to deal with non-standard voltages. Review these parameters and determine a suitable battery size to ensure reliability and redundancy.
7. Review the amount of data to be collected from devices and choose appropriate file storage approaches, as well as the capability of ethernet and USB transfer rates on the microcontroller.
8. Investigate the use and implementation of a redundant array of independent disks (RAID) to add a level of redundancy and reliability to the chosen storage approach.

If time and resources permit

9. Implement a GUI and interface to directly control and observe the microcontroller.
10. Test the portable management system for radio frequency and electromagnetic interference.

1.4 Limitations

This project will focus on ensuring the management system works as intended. The finished project may require housing if it is to be used for actual airborne observation. The term “portable” may not apply if the proposed battery system is included in the complete package, given car batteries range from 15-30kg each.

1.5 Overview of Dissertation

Chapter 2 reviews literature on the subject of satellite reentry, clock drift, time synchronisation, data management and power supply.

Chapter 3 discusses the methodology used to implement and test the airborne observation controller.

Chapter 4 covers the aspects of software and hardware development for the project to function.

Chapter 5 shows the results and how all the research done has come together and been implemented.

Chapter 6 discusses the conclusions and areas of future work.

2.0 Literature Review

2.1 Satellite re-entry observation

Airborne observation missions started sixty years ago when NASA was founded to use aeroplanes to study space. Water vapour in the air at ground level blocks certain types of light, such as infrared, from reaching ground-based observation centres. Using aeroplanes to reach an altitude approximately 12km above sea level, the aircraft is above 80% of the atmosphere and 99% of the water vapour. The benefit of airborne observation over ground-based was the ability to travel almost everywhere. This was especially useful for studying transient events such as eclipses and eclipse-like events called occultations. Airborne missions also have the benefit over space-based telescopes that they can be upgraded as new technology is released. NASA started by modifying a small Convair 990 aircraft to study a solar eclipse using infrared light. In 1975, NASA converted a C-141 cargo aircraft to carry a 36-inch telescope which was in use until 1995. This is when NASA and the German Aerospace Centre (DLR) started development of the Stratospheric Observatory for Infrared Astronomy (SOFIA), which is a modified Boeing 747 carrying a 106-inch telescope. SOFIA is still used today to study astronomical objects and phenomena, such as comets and asteroids in our solar system (Bell 2018).

With the increase in satellites being launched in the latter half of the 20th century, predicting when and where a decommissioned satellite or known space debris will re-enter the atmosphere became essential. Observing how the space object interacted with the atmosphere also became crucial to confirming simulation models and aiding in redesigning specific parts. The airborne observation of the Hayabusa sample return capsule was to obtain measurements from remote observation that could further improve modelling techniques. These time-resolved measurements included the capsule's absolute spectral irradiance and trailing wake. The instrumentation used to record data on these airborne missions was modified or retrofitted to fit the chosen aircraft, as seen in figure 1. A typical aircraft used to observe re-entering space objects is the DC-8 and Gulfstream V. These are relatively common aircrafts worldwide. This enables the science teams to watch astronomical events worldwide with reasonably short notice (Grinstead et al. 2011; Löhle et al. 2017).

The measurements and images of the different instrumentation used to record the re-entry of the space objects needs to be comparable so the datasets can be combined. For the Hayabusa re-entry observations, a DC-8 aircraft was used. This aircraft is equipped with a timecode generator synchronised to the global positioning satellite receiver. All the instruments used on the mission were synchronised to this timecode which made any measurements taken comparable to other time-accurate sources (Grinstead et al. 2011). The power supplied to the measuring instruments also needs to be consistent and up to specification ensuring the reliable operation and recording of results. As the Cygnus OA6 de-orbit report mentioned, “the aircraft 110V/60Hz voltage was a low 96 V… The low

voltage made some of our pointing cameras noisier, but the instruments were able to cope." – (Jenniskens 2016).

Figure 2.1 – High-resolution video imaging on Cygnus OA6 re-entry observation mission. (Löhle et al. 2017)

The synchronisation of the connected devices is a critical component of the portable management system. Even if there is a timecode generator on the aeroplane, the management system should be able to provide support in case of instrument failure. The management system should also be able to record GPS data and aircraft log data. Most importantly, redundancy must be built into the data storage system to keep the measurement and image data.

2.2 Microcontroller

2.2.1 Raspberry Pi

Many factors need to be considered when choosing a microcontroller for a project. Availability is one of the more critical design choices that need to be made with the current silicon shortage. The most used microcontrollers are the Raspberry Pi, Arduino Uno and the Beagle-Bone Black. While all of these would likely work for this project, the Raspberry Pi is more beneficial when considering in the cost-for-performance ratio and the community size that can provide troubleshooting and documentation for different aspects of the project (Pereira et al. 2018).

The Raspberry Pi 4 model B microcomputer, seen in figure 2.2, is roughly the size of a credit card, supports multiple inputs and output peripherals, and is low-powered. Several modules have been done for the Raspberry Pi to be applied in different scenarios. Raspberry Pi's run a Linux kernel called Raspbian and can be used as a standard Linux-based computer if the right peripherals are added. The Raspberry Pi 4 B has a Broadcom BCM2711 processor, which has four cores with a CPU clock of 1.5GHz. When it comes to random access memory, there are four options for DDR4 memory: 1GB, 2GB, 4GB and 8GB. There are four USB connections, two USB 3.0, two USB 2.0 and one 1Gigabit ethernet port. There is also a 2.4 and 5GHz IEEE 802.11ac wireless module built into the Raspberry Pi. The microcomputer requires a 15watt USB-C power supply to run, and the operating system needs to be installed onto a micro-SD. The microcomputer can be accessed and controlled using a screen, mouse and keyboard, or it can be accessed through SSH (Secure Shell Protocol) if it is connected to the local network (Pi 2022).

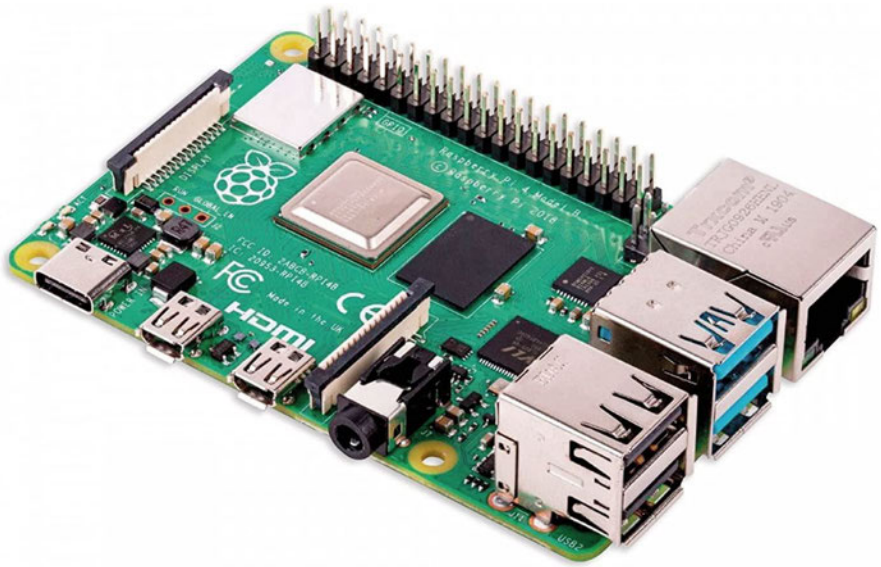


Figure 2.2 – Raspberry Pi Model B 8GB board. It is roughly the size of a credit card. (Pi 2022)

2.2.2 Application for Project

For this project, the Raspberry Pi will need to synchronise the time of the other devices connected to the network. It should also be able to receive messages from connected devices and store them. The measurement and image data collected from the observation instrumentation should be collected and placed into a network storage device.

2.3 Clock Drift and Time Synchronisation

Time synchronisation between computers is critical, especially regarding financial, legal, transportation and distribution systems. Any timing error between systems can cause disruptions to the entire network. In 1972, the standard time was based on the International Atomic Time (IAT), which uses cesium clocks to keep time accurate within a few picoseconds. This time is adjusted to the micro changes in the solar rotation earth, measured by observatories around the world, which results in Coordinated Universal Time (Mills 1991). The first use of time synchronisation was in the late 1970s with the demonstration of the internet operating over trans-Atlantic satellites. This exchange used the first iteration of the internet clock service (Mills 1981). In 1985, the first version of the NTP was developed, enabling local ethernet networks a time accuracy of tens of milliseconds. Over the years, NTP has been further developed and improved and now NTPv4 is the most common time protocol used (Mills 1991).

2.3.1 Clock Drift

Computer clocks are generally an order of magnitude worse at keeping time than an average wrist watch. Most computer have two clocks, a software and a hardware clock. The hardware clock is used to keep time while the computer is off and the software manages time when it is on. When the software clock is keeping time, there is uncertainty as multiple interrupts can cause a disruption. This disruption could cause the computer to gain or lose time. Over the course of a day, the clock may have changed from a few seconds to tens of seconds. The rate of drift can depend of many factors such as age of the computer, type of operating system and type of software it's running. The hardware clock is often updated every second while the computer is on, so when it's turned off, both clocks are synchronised. The accuracy of the hardware clock depends on the quality of the quartz crystal. These crystals are sensitive to temperature, as seen in figure 2.4, and are often known for drifting up to 10 seconds a day. Fortunately, most computers are connected to the internet, where they can use NTP to keep their time accurate (Lombardi, 2000).

2.3.2 Network Time Protocol

The NTP works by assigning a hierarchy to time sources. Those that are known to be very accurate, such as the cesium clocks, are assigned a stratum value of 0 and are also known as a reference clocks. Stratum 1 clocks are servers directly synchronised to reference clocks and are accurate to within a few microseconds of the stratum 0 clock. After each subsequent computer or server, the stratum number increases, as does the synchronisation distance which will decrease accuracy. Computers and servers in each stratum can compare their time to ensure accuracy. When a connection is lost to the NTP node, the subnet can reconfigure itself, which may result in a computer stratum increasing as seen in Figure 2.3 (Mills 1991; Chao et al. 2009).

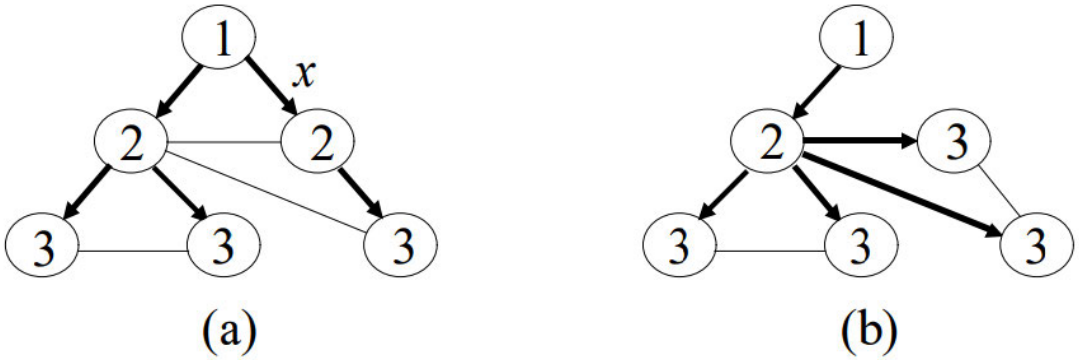


Figure 2.3 – Stratum subnet synchronisation (Mills 1991).

NTP uses the UDP to send timestamp data over the network as a server, client and peer. The server delivers the time, the client receives it and the peer can confirm it. The offset time and roundtrip delay from one computer or server to another is done by using the four most recent timestamps between the two computers. T1 is the time the message was sent from computer A, T2 and T3 are the timestamps the message was received and returned from computer B, and T4 is the timestamp the return message was received at computer A.

$$offset\ time\ =\ \theta\ = \frac{1}{2}((T2\ -\ T1) + (T3\ -\ T2))$$

$$roundtrip\ delay\ time\ =\ \delta\ = (T4\ -\ T1) - (T3\ -\ T2)$$

These timestamp values are 64bit unsigned values which result in a 63bit signed value. The offset time and delay time are the sum and difference of these timestamps which results in a 62bit number with two signed bits. Calculated out, 62bits is roughly 34 years in the past or future which means that the client and server need to be set within this range for the NTP calculation to work. Using these calculated parameters, the client time is gradually changed to match the server (Mills, D. L. 2006).

To use NTP on a Linux system, there are several packages that use NTP to update the time. Timedatectl comes preinstalled on the newest version of Raspbian while Chrony and the original NTP packages can be installed using the command line interface (CLI). Once installed, commands are entered in the CLI to set up the servers that update the time and configure the settings (Dinar et al. 2021).

2.3.3 Simple Network Time Protocol

SNTP is used for networks with a solitary reference clock, such as local networks that don't need to have many different stratum. SNTP is interoperable with NTP subnets, as they are essentially NTP subnets with no upstream servers. As SNTP is a simplified version of NTP, it does not have the redundancies of a full NTP network. It is recommended that SNTP networks use a server stratum no lower than 1 to receive time as there are no backup paths (Mills, D. 2006; Dinar et al. 2021).

2.3.4 Precision Time Protocol

PTP was developed to enable networks to have a time accuracy of less than 100 nanoseconds over an ethernet connection. Previously, to receive this timing accuracy, inter-range instrumentation group timecodes (IRIG-B) were used. Delivering these timecodes required the use of an additional coaxial cable. These PTP networks require a hardware clock, called a grandmaster clock, that serves time to the rest of the network. The rest of the clocks that are receiving the grandmaster's time are slave clocks. Any devices that are in between the grandmaster and slave, such as a switch, need to measure the time the message takes to traverse this device and add that measurement to the message. This type of network doesn't require the slave to message the grandmaster to measure the roundtrip delay, as the delay is measured from peer to peer. This allows PTP networks to scale better than NTP networks that have multiple stratum levels. The issue with PTP networks is the upfront cost of acquiring a suitable hardware clock and ensuring the network devices are compatible with PTP messaging. PTP networks are being implemented into modern power system applications and large data centres where sub-microsecond timing accuracy is needed (Watt et al. 2015).

2.3.5 Custom Time Synchronising

To change the time using custom time synchronising code, the Raspberry Pi needs to run code that will let the server "socket" listen for other nodes to connect to. Once a connection has been confirmed, the current time will be sent to the connected node. The connecting node will record the time it received the message, and the time it responded to with different timestamps. When the server node receives the response, the offset time and round-trip time will be calculated. Further messages sent to the receiving node will include an amount to offset the time with so it matches the Raspberry Pi time. This code will use the offset and roundtrip calculations of the NTP to hopefully achieve similar timing accuracy (Xue & Zhu 2009).

2.3.6 Modules to aid time synchronisation

The Raspberry Pi has several different modules that could aid in keeping accurate time relative to UTC. A RTC uses a tuning-fork crystal, or quartz crystal, to keep time. Simple RTCs are used in electronics to keep relatively accurate time but even a quartz crystal will drift as environmental variables change. The ideal temperature for a RTC is around 22°C and any change in this temperature will cause it to run fast or slow as seen in figure 2.4. A USB-based GPS device that can output GPS data at 1Hz will act as the stratum 0 clock for the network. This GPS, paired with a RTC module, will ensure accurate time is kept even if the GPS signal is lost. The RTC chosen to use is the DS3231 which can maintain the time to ±2minutes over a year or 3.5ppm. If this RTC isn't available, a less accurate ds1307 can be used as they are readily available, however also have an error rate of ~20ppm. The RTC will take up a few GPIO pins while the GPS can be inserted into one of the USB 2.0 inputs (Intergrated 2015).

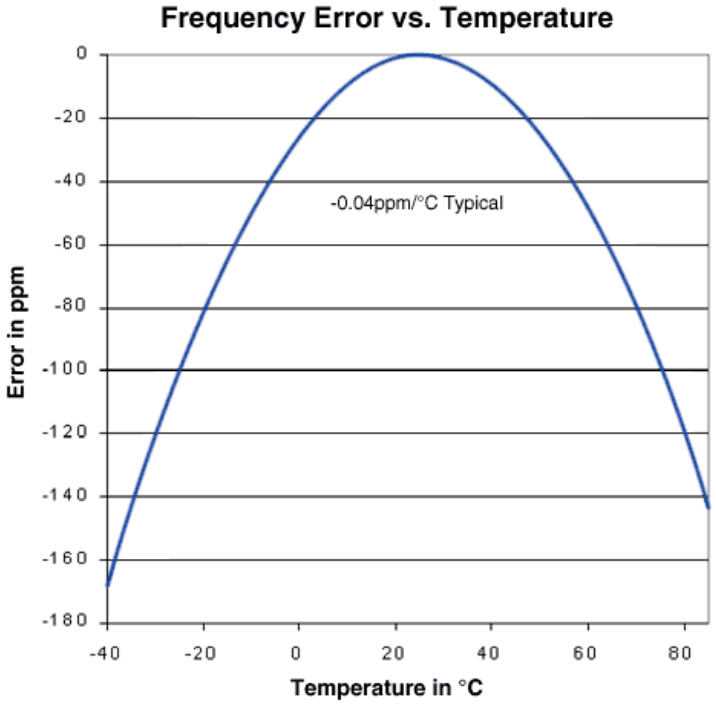


Figure 2.4 - Error in quartz crystals with temperature (Integrated, 2002)

2.3.7 Application for Project

For this project, the management system will use both NTP and a custom time synchronisation. The use of PTP would be ideal for this application but the upfront cost of its implementation does not

outweigh the benefit. The extreme time accuracy PTP provides is not needed in this application and the time accuracy of 1-10 milliseconds the NTP provides should be sufficient. The effectiveness of each type of system will be compared with the time accuracy they keep the network at and the bandwidth that each type of communication takes.

2.5 Data Management

2.5.1 Hard Disk Drives

There are two ways to store digital data: on hard disk drive (HDD) or on a solid-state drive SSD). HDDs are a complex electromechanical device that uses magnetic recording principles for data storage. A magnet on the end of an actuated arm can detect and change the polarity of an area on the magnetic disk. This polarity is converted into binary which makes up digital data. As the areas on the disk get smaller, more data can fit on a disk. Any unplanned change in the magnetic head location or a strong magnetic field near the disk could possibly corrupt the data on the HDD (Blattau & Hillman 2004; Strom et al. 2007; Rizvi & Chung 2010).

2.4.2 Solid State Drives

SSDs use flash memory and have benefits over HDDs such as smaller size, faster read and write times, reliability and shock resistance. Although the price of SSDs has dropped, they are still more expensive than HDDs per gigabyte. SSDs are either single level cell (SLC) or multi-level cell (MLC) flash memory, which indicates the amount of data stored on one memory cell. With more bits on a single memory cell, the separation between bits is reduced, increasing the possibility of errors when writing and reading data. While more reliable, SLC SSDs are almost non-existent in the consumer space as the required storage density is unfeasible (Rizvi & Chung 2010; Ho et al. 2018).

2.4.3 Soft Error Rates

Soft errors are errors that are caused by ionising radiation which can come from cosmic rays. This radiation can cause a bit flip in the SRAM and latches on SSDs as they are more sensitive to small changes in current. These errors are infrequent but do increase at higher altitudes. To mitigate these soft errors, integrated circuits are radiation hardened to stop most of the cosmic rays. Error correction code is used on the software side to stop the soft error from corrupting any data. Well-designed ECC is shown to reduce the soft error rate impact to insignificant levels (Blattau & Hillman 2004; Slayman 2011; Mielke et al. 2015).

2.4.4 Redundancy

To increase the reliability of the data storage, disk array architecture such as RAID (Redundant Array of Inexpensive Disks) can be used. RAID can increase the apparent disk read and write speeds depending on the number of disks and level of RAID used. RAID1 simply mirrors two disks so that if one fails, all the data is backed up on the second. RAID2 – RAID5 use different types of parity to recover data if one or more drives are unrecoverable. (Chen et al. 1994).

2.4.5 Network Accessed Storage

To store the acquired measurement and image data, a storage device should be integrated into the management system. This storage will need to be accessible on the network so that all the devices can access the storage, a Network Accessed Storage (NAS). There are many packages for the Raspberry Pi that enables it to act as a NAS. When it comes to the speed that the NAS can read and write data, the read and write speed of the disks and CPU of the microcomputer will be the limiting factors (Shrivastava & Gadge 2017).

2.4.6 Application to Project

Using the Raspberry Pi as a NAS with either HDDs or SSDs will work. An issue with the power draw of the drives may indicate that a separate power supply for the USBs will be needed. In this case, an external USB hub that can supply power will have to be added. RAID1 should be used so that there is storage redundancy. Tests should be done to compare the performance of the SSDs and HDDs.

2.5 Power supply

2.5.1 Uninterruptible Power Supply

An uninterruptable power supply (UPS) is used to protect electrical equipment when there is an unannounced power disruption. The UPS will take over the supply of power while informing the user that power has dropped through an alarm. These power supplies usually aim to supply power for up to ten minutes at their rated load. This gives the user enough time to properly save and shutdown the electrical equipment. A UPS also helps when there are brown outs on the electrical grid. These are times when the power supply is not at the standard it should be. The UPS will cleanly supply the correct voltage to the connected equipment. The size of the UPS depends on the rated power, which can range from 300VA to 2000VA for residential customers. The UPS can also act as a surge protector with some UPSs having a surge protection value of 936J (Guerrero et al. 2005; Aamir et al. 2016).

2.5.2 Battery Power

A standard car battery can supply anywhere from 30Ah (amp hours) to, in some of the bigger batteries, 120Ah. The standard voltage for a car battery is 12V. If this project was to use car batteries to supply power for the laptops, the 12V supply will need to be changed to suit the laptops required voltage. An average laptop requires roughly 60Wh (watt hours) at a voltage that depends on the brand of the laptop but is usually from 16-20V (*Guide to technology energy usage* 2022). If it is assumed that there will be six laptops that require power:

$$total\ power = 6 \cdot 60$$

$$= 360Wh$$

Total power from a single car battery (assuming the smallest battery):

$$total\ battery\ power = 30Ah * 12V$$

$$= 360Wh$$

One small car battery should be able to power six laptops for one hour if there is 100% efficiency with the DC-to-DC conversion. This calculation does not consider the power of the Raspberry Pi, the storage drives and the network switch needs. The Raspberry Pi requires 15Wh while the switch needs 3Wh. The combined power that the storage devices need, based on USB 3.0 standards, is 9Wh. This works out to a total of 397Wh. If space is more crucial than price, 12V lithium battery generally has upwards of 100Ah which would be able to supply power to all devices for roughly three hours. Lithium batteries offer more consistent power delivery, lighter weight, longer expected lifetime, faster charge time and better power delivery at extreme temperatures. These benefits of a lithium-ion battery over a car or lead-acid battery make it an easy choice if cost is a non-issue.(Powersonic, 2022)

2.5.3 Inverter

An inverter converts DC into AC and is most commonly used to convert the power produced from solar panels so that households can use it. This conversion is accomplished by constantly switching the DC direction to produce a sine wave. Portable inverters are often used in camping scenarios so that power from car batteries can be used by corded appliances. The efficiency of an inverter depends on the amount of power the setup uses. Peak efficiency often occurs around 40% of the inverter's rated power output. Using the above calculations, an inverter to power the Raspberry Pi and the connected laptops should have a rated-power output of 1000W.

2.5.3 Project Application

Considering the amount of power needed, at least two lead-acid or one lithium battery would be needed. A battery coupled with an inverter should be able to power the connected devices for over an hour. An inverter with a power pass-through would be ideal as this would charge the batteries while acting as a battery backup.

3.0 Project Methodology

3.1 Overview

The aim of this project is to have a working portable management system that can synchronise devices that are a part of its network. The system will also act as a NAS with a redundant drive for reliability. Logs from the aircraft and other notifications should be kept in the storage as well. The steps followed for this project will generally follow the steps from the Project Management Book of Knowledge (Cynthia Stackpole 2013) which are:

I. Initialisation
II. Planning
III. Executing
IV. Monitoring and controlling
V. Closing

3.2 Initialising

The project initialising stage involved choosing a topic and starting to research the background of the topics covered. The scope of the project and its intended outcomes were realised and can be seen in Appendix A – Project Specification. From the initial understanding of the project, some materials and parts were ordered, and a preliminary project plan was devised. This part of the process involved research and information gathering to prevent possible miscalculations in the executing phase. Packages on the Rasberry Pi were also investigated to see most effective way to implement the CTS. Guides on how to properly implement the RTC and GPS into the Raspberry Pi were found and saved to be used when the hardware arrived. The research and guides found, seen in Hardware and Sowfware Development chapter, were used achieve the aims of the project seen in the Implementation and Testing chapter.

3.3 Planning

Once the project scope has been defined in the Project Specification, more precise research can be done on how each part of the portable management system will come together. With more information on each part of the project, parts that were initially decided upon can be confirmed and ordered. Research on how the performance of the different parts, such as the time synchronisation method, can be measured. The final project plan can be confirmed now that there is a full bill of materials as seen in appendix B. The risk management plan can also be completed as seen in Appendix H.

Planning to write the code for the custom time synchronisation began with a flowchart. This flowchart helped organise and coordinate the processes that needed to be accomplished with the code.

3.4 Executing

Here is where the hardware and software implementation will continue to build from the plans made in the previous section. The Raspberry Pi should have its operating system, Raspbian, up and running and should be plugged into an ethernet network. There should be at least one other device on the network to ensure the time synchronising is working. Once the Raspberry Pi is setup with the NAS packages and the drives are setup with RAID1, a test should be done to make sure it's working. The battery system parts should be prepared if the custom battery setup is being used. The software development can start and design focused iteration (DFI) should be used instead of code focused iteration (CFI). DFI is the process of changing the design of the code to meet the new requirement rather than just adding a test case and refactoring late in CFI. In DFI, the code should evolve with the the design (Fairbanks. 2022).

3.5 Monitoring and Controlling

This phase will involve testing the management system to check it is working as intended and recording the results. The tests that will be run and results will be documented:

1. Performance of the CTS.
2. Performance of the NTP time synchronisation.
3. Ability to log any messages sent to the Raspberry Pi.
4. NAS performance under different conditions.
5. Power usage during the above test

Each test should share the same environmental variable so there is no external input that may change the result. This includes temperature to keep the RTC the same, behind the scenes upgrading, same version of packages and operating system, and same GPS placement.

3.6 Closing

This will involve explaining the results and discussing what they show and why they show it. Any results will be organised so that a clear outcome can be seen. As the final dissertation is being written, a condensed version of the project is to make into a presentation for the project conference. Recommendations for future work will be made.

4.0 Hardware and Software Development

4.1 Project Overview

This project involves the process of creating a prototype that can fulfil the aims set out in the project specification (appendix A). This involves acquiring hardware to build a prototype and developing software that runs on the hardware. As there is a worldwide chip shortage, partly due to COVID-19, some hardware was harder to acquire while other parts took longer to be delivered. As time was limited, there were situations where a specific component choice was infeasible. However, there wasn't enough time to replace it and start again. Once the hardware development was complete, work was started on the software development. This involved creating custom time synchronising files to run on the server and the client. Another piece of software was developed to enable messaging from the client to the server. These messages would then be placed in a text file with the time and ID of the computer that sent it. The software development also involved integrating some of the hardware with the Raspberry Pi so it performed as expected. Finally, tests were run to measure the performance of the time-synchronisation compared to NTP and the amount of power used.

4.2 Project System Diagram

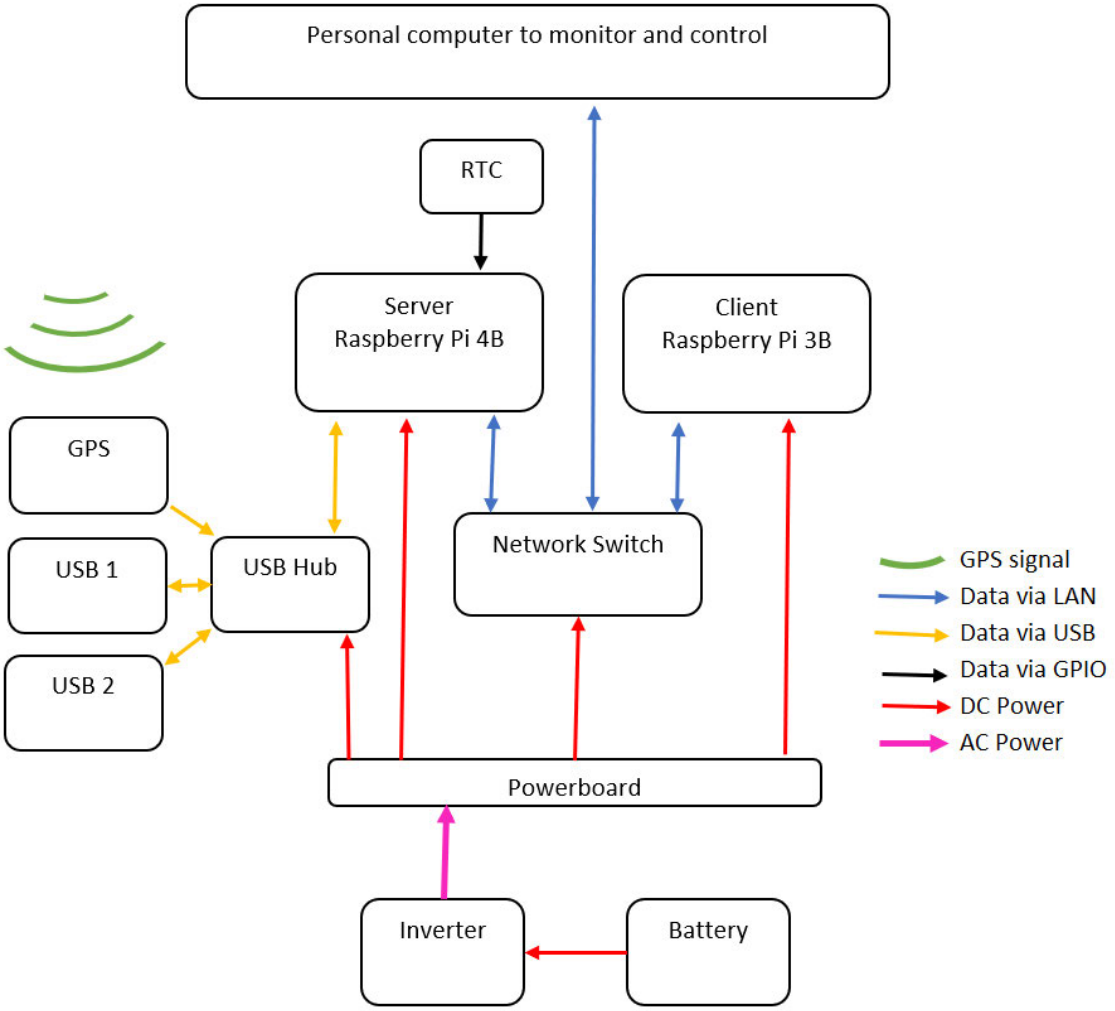


Figure 4.1 - Airborne Observation Controler system diagram.

The prototype system seen in figure 4.1 consists of the server, a client and all the sub-components to power and transfer data. The server and client were only connected to the internet to receive software updates and to install the packages needed to complete the prototype. The devices were accessed and controlled using VNC viewer on a windows computer. VNC is a remote access software for multiple operating systems. In situations where VNC could not connect to the server or the client, a separate screen, keyboard and mouse were used. The completed build can be seen in figure 4.2 though there is no client connected. The clips connecting the battery to the inverter in figure 4.2 were supplied with the inverter and were covered with a shield to avoid an accidental short circuit.

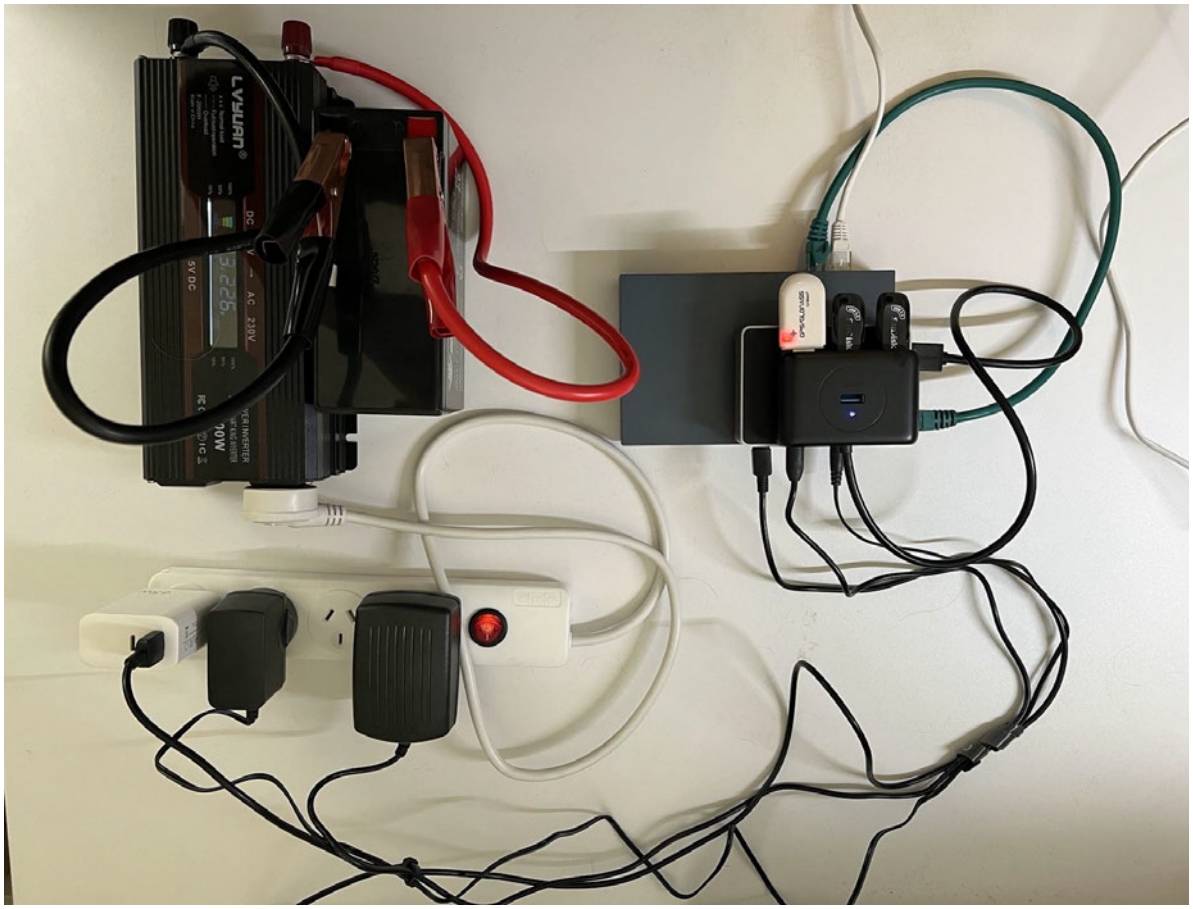


Figure 4.2 - finished prototype system. The server, USB and switch are stacked on the right with the inverter and battery on the left.

4.3 Battery System

The battery system chosen for this project was a lead-acid battery with an inverter. The two Rasberry Pis, the switch and the USB hub could theoretically draw a maximum of 30W. With this maximum power, a battery with 7.2Ah was chosen, as this would last for around two hours depending on the efficiency of the inverter. The inverter found was intended to power camping equipment from a car battery. It had a rated sustained power output of 1000W and a peak power draw of 2000W. For this small prototype setup, the inverter wasn't ideal as less than 30W on the efficiency curve would result in a low efficiency rating (Pearsall, 2016). Given the limits of the budget, this battery system was deemed adequate to prove that it works. This system can be seen in operation in figure 4.2.

4.4 Setup of the Microcontroller - Raspberry Pi

This section explains how the Raspberry Pi server and client were set up before software development began. The RTC being used is the DS1307 and the GPS module is the VK-172.

The microcontroller used for this project is the Raspberry Pi 4B 4GB model. As stated before, the cost/performance ratio was seen as the best among similar microcontrollers (Pereira et al. 2018). The 4GB model was chosen as it was available and had reasonable delivery times. The Raspberry Pi has a large user base and runs Rasbian, a Linux-based operating system, which is optimised for the inbuilt hardware. The Pi 4B model was chosen as it has four USB ports and an ethernet connection.

4.4.1 Install Raspbian

The latest release of Rasbian, kernel version 5.15, was downloaded and mounted to an SD card using Raspberry Pi Imager seen in figure 4.3. This software allows host names and SSH options to be toggled. Once the imager is done, the SD card can be inserted into the Raspberry Pi and turned on. If VNC or SSH was not set up, a mouse, keyboard and screen will be needed to complete the bootup sequence.

Figure 4.3 - Raspberry Pi Imager v1.7.3

4.4.2 Enable I2C

Once the desktop has been loaded, the Raspberry Pi configuration can be opened through the drop down menu in the top left. In this menu, select Interfaces and turn on I2C which will be used to setup the RTC later on. This is also the menu to turn on VNC and SSH if not done already as seen in figure 4.4.

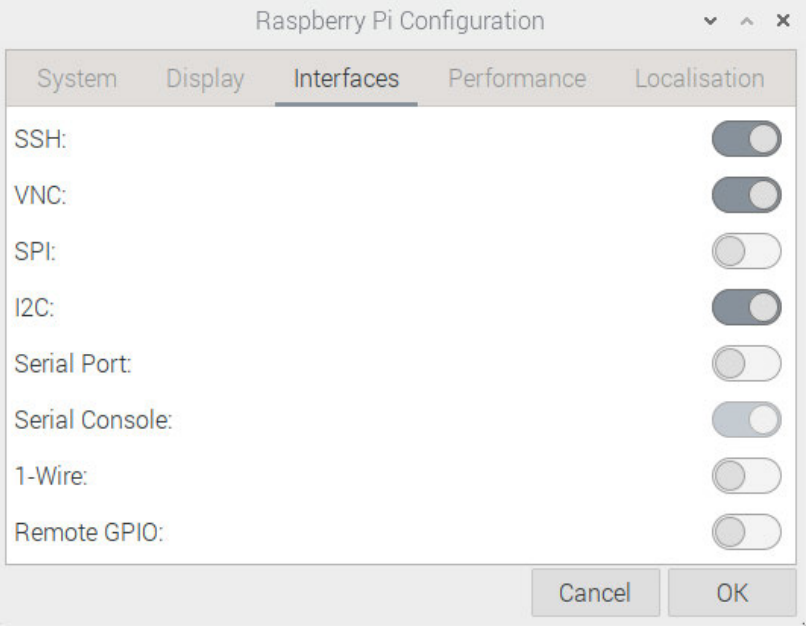


Figure 4.4 - Raspberry Pi configuration menu

4.4.3 Update and Upgrade

Before installing any of the needed packages, the Raspberry Pi should be completely up-to-date. Open the terminal and type the following commands:

1. sudo apt update
2. sudo apt upgrade
3. sudo rpi-update

This will ensure the Raspberry Pi installs any updates found and checks the for the latest kernel of Raspbian.

4.4.4 RTC Installation

The RTC chosen for this project was the DS1307. This RTC is a low power device that uses full binary coded decimal clock (Integrated, 2015). Data is transferred serially through an I2C connection on the GPIO which is why it needs to be turned on before. These are the steps to follow to install the RTC on the Raspberry Pi:

1. The RTC needs to plugged onto the GPIO pins on the second row to the far left as seen in figure 4.5.
2. Verify the Raspberry Pi detects the RTC by running “sudo i2detect -y 1” in the terminal. #68 should appear in row 60, column 8.
3. The RTC module needs to be loaded by running “sudo modprobe rtc-ds1307”.

4. Next, enter root access mode by entering “sudo bash”.
5. Enter “echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device”.
6. Exit root access by typing “exit”.
7. If the Raspberry Pi is connected to the internet, enter “sudo hwclock -w” to write the current time to the RTC.
8. To make the Raspberry Pi use the RTC, the module needs to be placed in the module file. Enter “sudo nano /etc/modules” and enter “rtc-ds1307” at the end of the file.
9. To enable this module on startup enter the rc.local file by typing “sudo nano /etc/rc.local”
10. Enter the following “echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device

    sudo hwclock -s

    date” just before the exit line.

The RTC is now installed and used by the Raspberry Pi if there is no active time synchronisation. (PiHut, 2015).

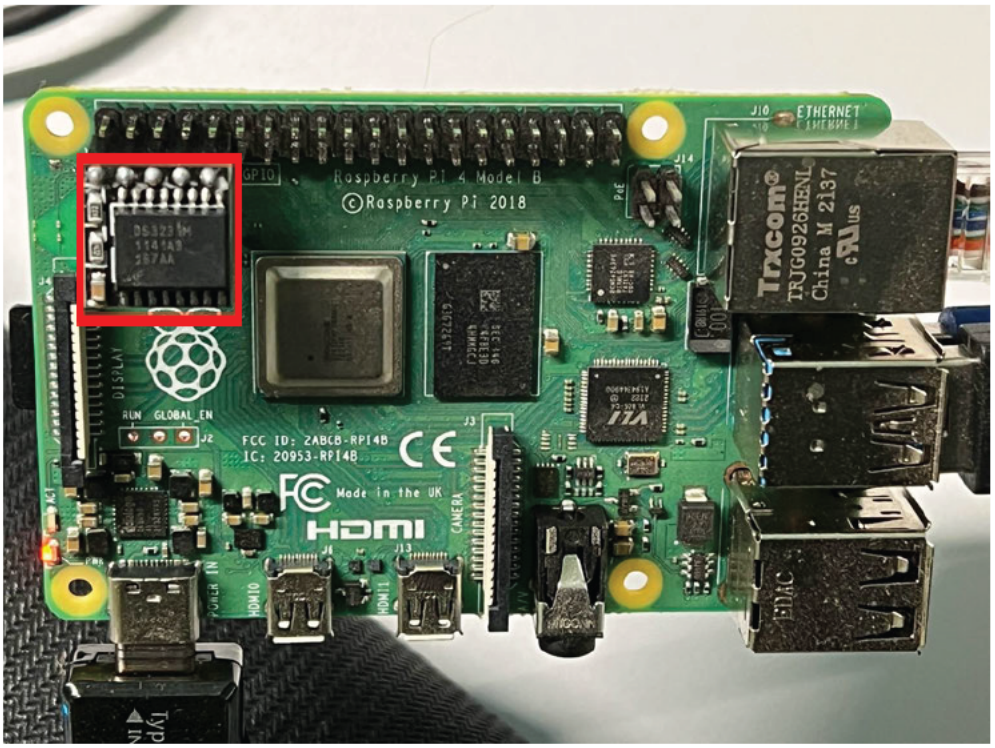


Figure 4.5 - RTC highlighted on Raspberry Pi 4B

4.4.5 Chrony

There are a few NTP packages that are available to be used on Raspbian. Chrony was chosen for this project because of its ease of use and ability to log time data. Several guides have also used chrony in the research for this project. To install chrony is as simple as entering into the terminal “sudo apt install chrony”.

Chrony is an implementation of the NTP and can synchronise with NTP servers, reference clocks and personal GPS devices. On a wireless network, it can achieve time accuracy within a few milliseconds, and hundreds of microseconds on a LAN network. For this project, chrony will be used to compare to the custom time synchronisation developed and be used to synchronise time to the connected GPS (Curnow, 2021). To view the NTP sources that chrony is currently using, “watch chronyc sources” can be entered in the terminal to view the available time sources. To change the way chrony works, the configuration file can be edited by entering into the terminal “sudo nano /etc/chrony/chrony.conf”. It is crucial to add an allow clause on the Raspberry Pi server so that the client devices can use the server as a time source. An allow clause is seen in figure 4.6, where the local network IP address is allowed to use this device to synchronise time. To enable chrony logging, the hash must be removed from the “log tracking measurements statistics” line. When setting up chrony, the minpoll and maxpoll number dictate how often it will request time. When first installed, the maxpoll number is is 5 and minpoll number of 4. This means it updates the time at least every 64 seconds but no more than once every 32 seconds. The numbers of the minpoll and maxpoll are used as powers of 2. When testing, to make NTP and CTS comparable, the minpoll and maxpoll were set to 0 and 1 or updating between 1 and 2 seconds. When the client is using NTP, it will only receive time from the server if there is an active stratum 0 or 1 clock that is updating the server. To bypass this, the chrony.conf file can be edited on the server to include “local stratum 1”. This will enable the client to synchronise with the server even if the time is not accurate.

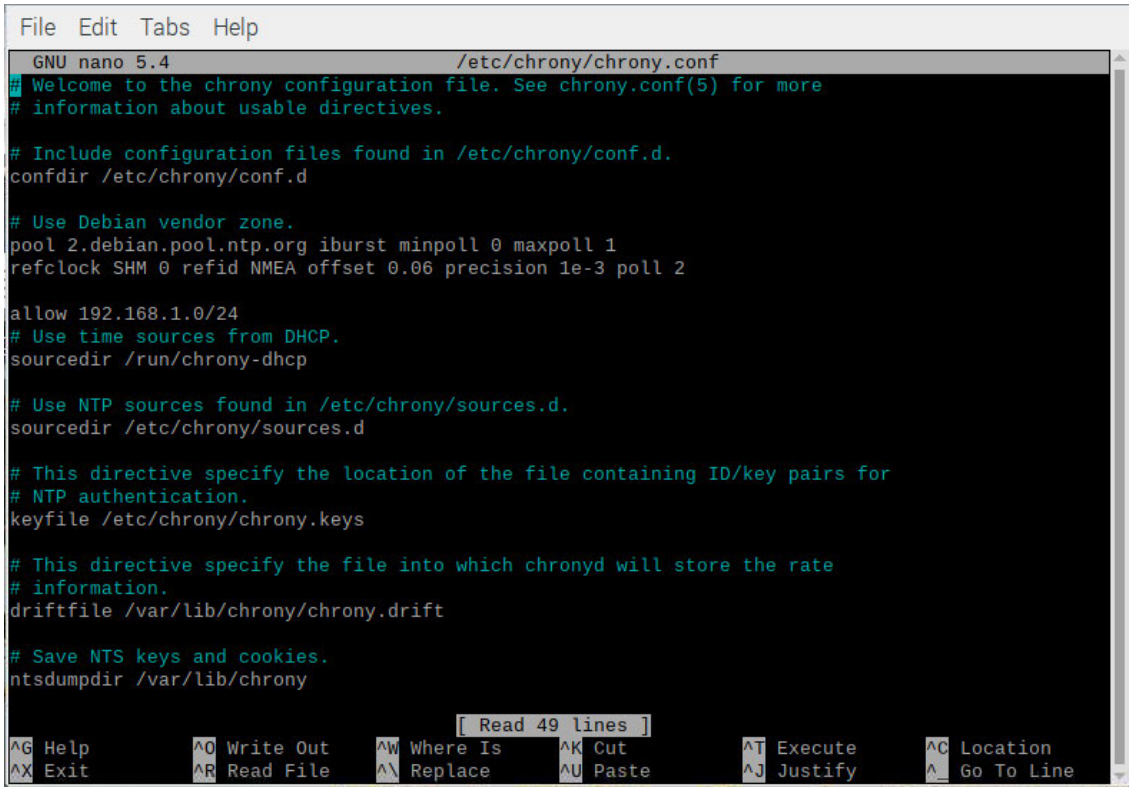


Figure 4.6 - Chrony configuration file

4.4.6 GPS Installation

The GPS used for this project was the VK-172, a USB-powered GPS that uses the G7020 u-blox chip to connect to the GPS and GLONASS satellites. To set up this GPS, an number of packages need to be installed and some configuration files need to be changed.

The packages that are needed are gpsd, gpsd-clients and gpsd-tools. Gpsd is a service daemon that enables the host computer to query data from a GPS devices that is connected via USB. Gpsd is widely used in mobile embedded systems and is heavily involved in driverless cars and drones (gpsd, 2021). For this project, gpsd will convert the data from the VK-172 GPS into meaningful data. this data can then be used to synchronise clocks and give latitude and longitude measurements. The GPS logger used in this project originates from one of the GPSD repositories which shows a simple way to extract values from the stream of data. The data stream is in NMEA format which always starts with a "$" and each value is separated by a comma. To install the packages and set up the GPS (Austin, 2021):

1. Enter into the terminal "sudo apt install gpsd gpsd-clients gpsd-tools".
2. Modify gpsd settings by entering "sudo nano /etc/default/gpsd"
3. Add or change the following:

a. START_DAEMON=”true”
b. USBAUTO=”true”
c. DEVICES=”/dev/ttyACM0”
d. GPSD_OPTIONS=”-n”

4. Reboot using “sudo reboot”
5. Add “refclock SHM 0 refid NMEA offset 0.000 precision 1e-3 poll 2” to the chrony.conf file seen in figure 4.6.
6. Entering into the terminal “gpsmon” will show the current GPS statistics shown in figure 4.7. The TOFF number can be used in the chrony.conf file as the offset to improve the time accuracy of the GPS.

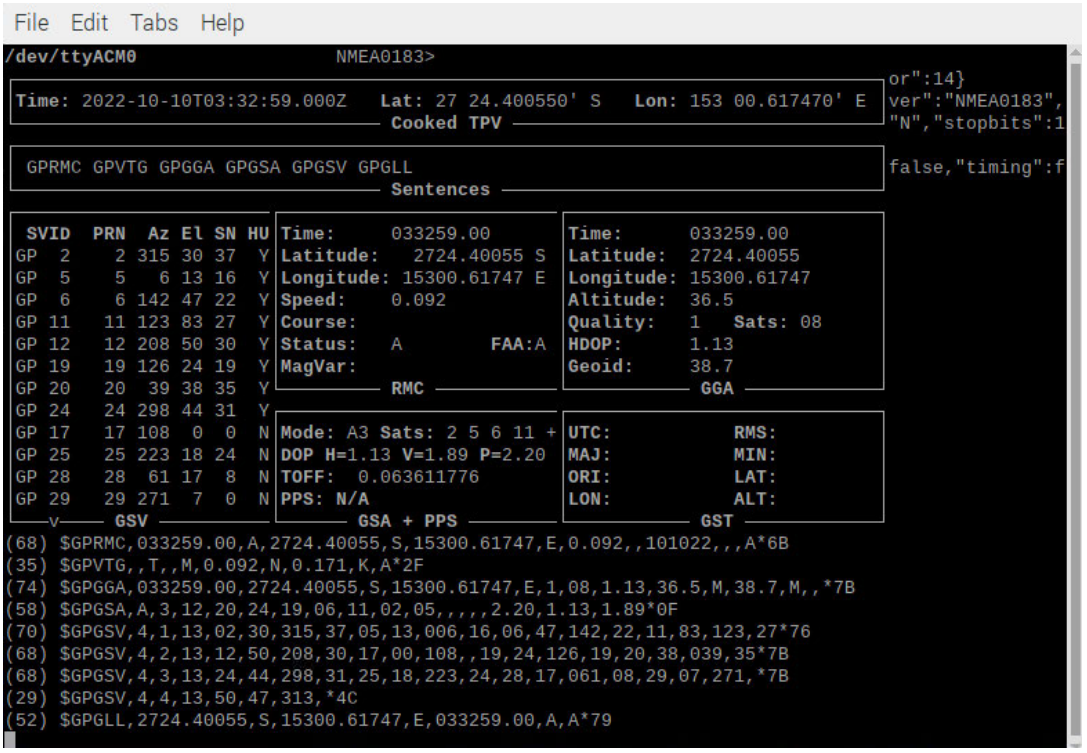


Figure 4.7 - gpsmon showing current GPS statistics

4.4.7 Telnet

The telnet package enables devices to communicate using the telnet protocol. This is a simple TCP communication protocol that allows messages to be sent between server and client. For this project, ther Raspberry Pi server will listen on a chosen port for any incoming telnet connections and pair with multiple clients. The clients will be able to send messages to te server where they will be collated into a singe text file (die.net, 1994). To install the telnet package, enter “sudo apt install telnet” into the terminal on both the server and client devices. When the Raspberry Pi server is running the message

server, the client enters “telnet IP port”. In this case the host name of the Raspberry Pi was used, piserver.local, with the port number 9007.

4.4.8 Mdadm

Mdadm is a Linux package that is used to create RAID arrays. A RAID device is a virtual device that is created from multiple real drives. This allows multiple drives to hold a single file system. This gives advatages in redundancy and speed depending on what type of RAID is used. For this project, RAID1 is used which is drive mirroring. By mirroring, one drive is always a copy of the other and can replace any data that is faulty on the other drive (die.net, 2010). Mdadm installed by entering into the terminal: “sudo apt-get install mdadm”. To create the RAID1 array, the instructions in Appendix G were followed.

4.4.9 Samba

Samba is a package that allows local drives to be hosted on the network. It does this by using the SMB networking protocol. For this project, samba enables the RAID1 drive created to be access by all devices on the local network. Important data that has been collected on the observation mission can be backed-up on these drives to improve redundancy (Hertel, 1994). To install samba, the following is entered into the terminal on the Raspberry Pi server: “sudo apt-get install samba, samba-common-bin”. The instructions on how to use samba can be found in Appendix G.

4.4.10 UFW

Ufw, or uncomplicated firewall, is a simple to use firewall that was used in testing. Ufw enables the user to easily block certain connectections or all of them. By configuring ufw to block all incoming data, the accuracy of the custom time synchroniser could be verified. To install ufw, enter into the terminal: “sudo apt-get install ufw”.

4.5 Software Development

The aim of the time synchronisation software for this project was to synchronise the time of the clients to the server time. The aim of the messaging software was to be able to record and store the messages from all the clients. Both of these tasks require the use of socket and fork programming. The flow charts used for each of the programs can be seen in figure 4.8.

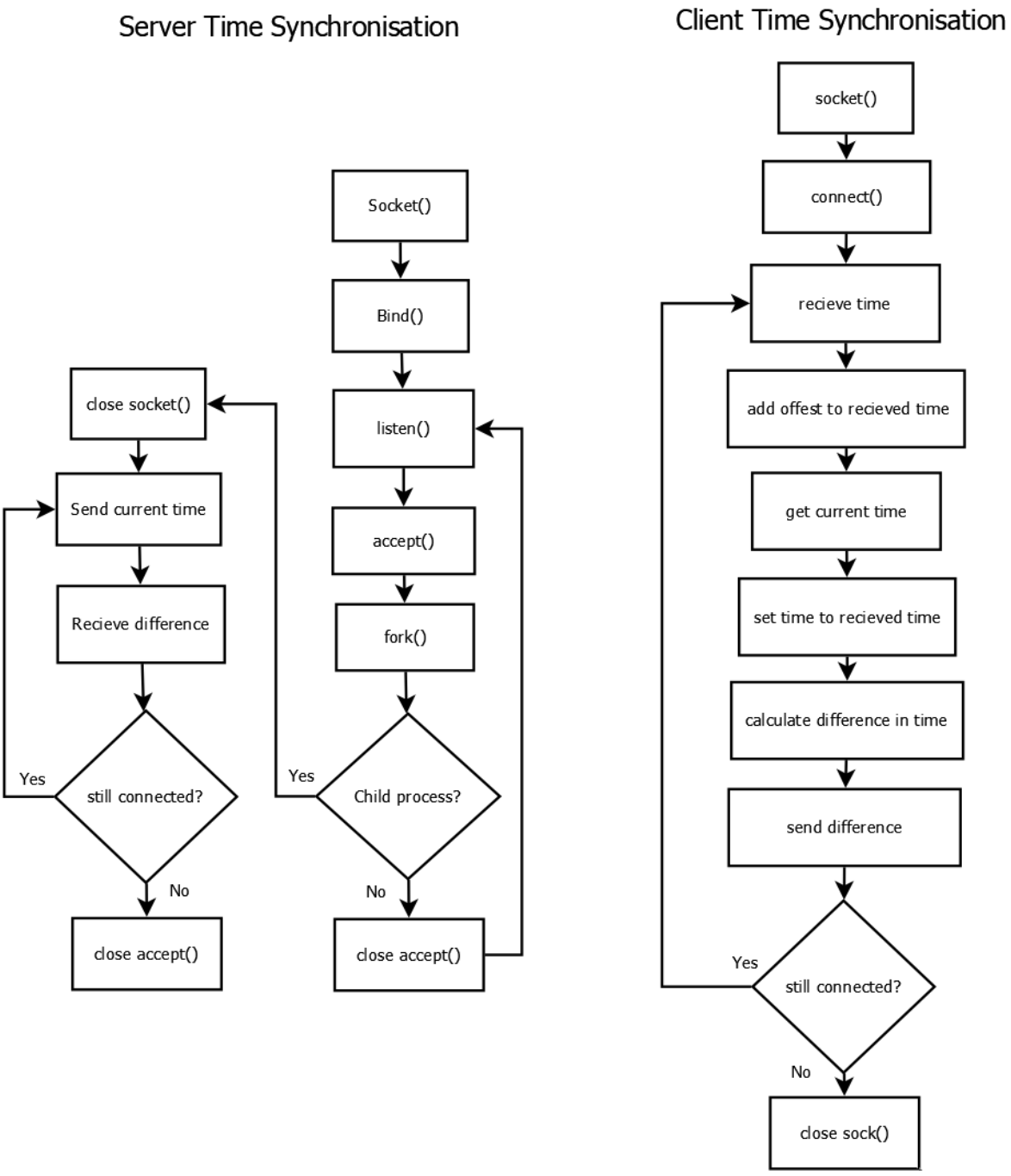


Figure 4.8 - Flowchart for custom time synchronisation software development

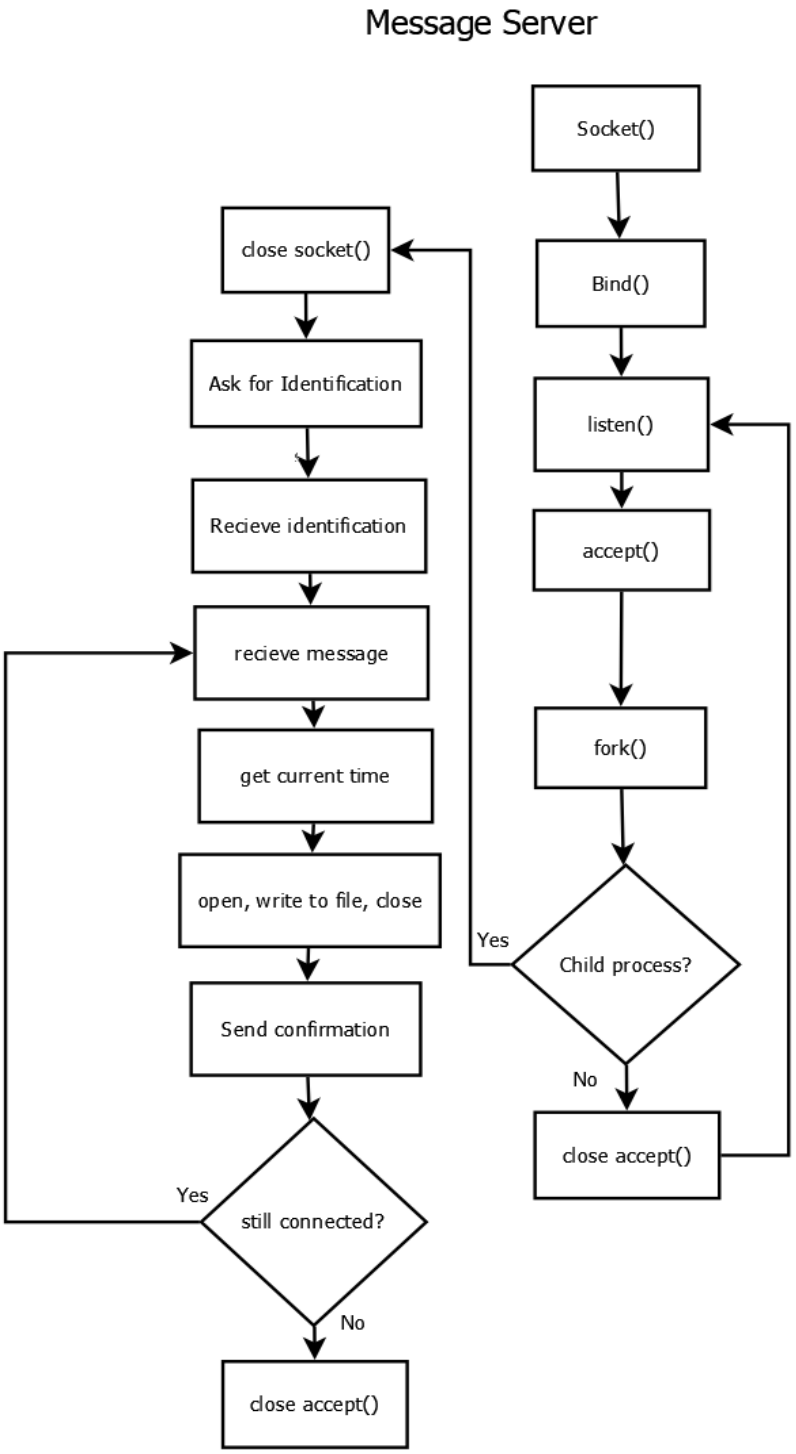


Figure 4.9 - Flowchart used for the message server software development

4.5.1 Socket Programming

A socket is an addressable entity used for communications between devices. They can be used between process on the same computer of across global networks. The primary mode of communication for sockets is TCP/IP. Socket programming is using these sockets to communicate

between them. There is always one socket that is passively listening, the server, while the other is actively looking for the connection, the client. To setup a socket the socket function needs to be used. The arguments for this function are domain, type of communication and the protocol. For this project the AF_INET was used for the domain, the type of communication is TCP and the protocol is IP which is signified as the integer 0. AF_INET is a macro that sets the domain to any IPv4 address. The bind function is then used to bind the created socket to a given address and a defined port number. The listen function is used to make the socket ready for a connection. When another process attempts to connect with the socket address the accept function is used to confirm the connection. The two nodes are now connected through the socket and read or write functions can be used to transfer or read data from the socket. When either node is done using the socket the close function will cancel it. Figure 4.8 and 4.9 show the flowcharts of each program using sockets to communicate between eachother. The clients of the message server won't need any software developed as they can connect directly with the telnet package.

4.5.2 Forking

The fork function creates a new process that continues through the code. This new process is called the child process, while the original is the parent. When fork is used, it returns an integer which shows which process is which. The parent fork will return a 1 and the child fork will return a 0. Fork will be used in this project to enable multiple clients to connect to the same server. This is shown in the flow diagram of the message server and the CTS server (figure 4.8 and 4.9) where the parent continues to listen for new connections while each child communicates with a client.

4.5.3 Time Structures

This project uses two different time includes, time.h and sys/time.h. Both of these header files use time structures that are based around the time since the start of the Epoch. This is defined as midnight on the first of January 1970. This day was picked arbitrarily as it was around the time when the Unix engineers were developing their code. By having a uniform start date for all time values, they can be stored as an integer and is easily parsed across different platforms. Time.h contains the time struct that is used to print the current time. It also has the clock() function which is used to calculate the computation time of the code. This is done by calling the clock() function before the part that is being measured and then calling it after the part. This gives the clock cycles taken to run that portion of the program. This is then divided by the macro CLOCKS_PER_SEC which is the speed of the CPU. This results in the time taken to run that part of the program. Sys/time.h contains the timeval structure that is used to transfer the time from the server to the client. This structure contains two "long" numbers, tv_sec is the number of seconds since the start of the Epoch and tv_usec which is the fraction of a

second measured in microseconds. This structure can be filled by calling the function gettimeofday() and time can be set by using settimeofday(). The ability to change the system clock can only be done with root or admin privileges which means the client devices will need to run the CTS script with these privileges.

4.5.4 NTP vs CTS

The NTP calculates the offset by using two timestamps from the server and two from the client. This gives a more accurate time offset which can account for changing delivery times. This project uses a wired local network which will cause the delivery time to be extremely short and consistent. It was decided when developing the CTS software that a simpler approach would be taken and only send the time that the client should change to. When this time is received, the client will add known time offset which is found earlier using the ping command. This command will give the average delivery time between the two devices which can be used as the offset time for the client.

4.5.5 Custom Time Synchronisation Server

The main goal of the CTS was to perform better than the standard NTP. The design of the CTS started with the development of the flowcharts in figure 4.8. Built around the socket and fork function so that it could communicate with multiple clients. Initially the server could only connect to one device while the time synchronising part was tested. The server starts by creating and binding to the port 9002. This port number was arbitrarily chosen and can be replaced by almost any other port number over 1000 as long as the client uses the same port. The socket is then set to listen for the client trying to connect. Once connected, the server will use the timeval structure to get the current time and send it to the client. It then waits for the reply which was a confirmation or sending back the time offset during testing. Once the time synchronising was performing adequatly, the code was modified so that it forked when a client connects to the socket. To test the performance of the software, various print functions were used to show the value of variables at different points in the code. The code used for this project can be seen in Appendix C.1.

4.5.6 Custom Time Synchronisation Client

When the socket was set up for the client side, the IP address of the server was specified in the server address section. Once the socket has been defined, it attempts to connect to the server. If the server isn't running or the IP or port number is incorrect, it will print an error to the terminal. Once successfully connected, the client enters a while loop where the sever time is recieved, the offset is added to it, the client time is set to the modified server time and then the difference is calculated. To

calculate the difference, the current client time is fetched before the new time is set and compared to the modified server time. The difference is converted to milliseconds from microseconds to improve readability. To make data collection easier, the time difference is printed to the terminal with the current time separated by a coma. This can then be directly copied into a .csv file. The amount to modify the server time by was calculated by using the ping command between the server and client. To check the amount of time elapsed from receiving the server time to setting it, the clock function was used. Similar to the server development, various print statements were used to troubleshoot different variable at different times in the code. Client CTS code can be found in Appendix C.2.

4.5.7 Message Sever and Client

The aim of the message server was to enable the users to record messages at each of the client devices which would be collected by the server and saved. Development of the message server started once the CTS server had finished and used it as a template. Where the two start to differ is once the child process has been created. After the fork, the server sends a message asking for an ID so that its messages can be tagged with it. This will enable users to look at all the messages and discern which device they came from. Once the ID has been set, the child process enters the while loop which wait for a message to be sent. When it recives a message, a text file is opened where the time, the ID and the message are appended to it. The file is then closed and it waits for another message. On the client side, the telnet package is used to connect to the message server. there are no scripts that need to be run, just a separate terminal from the CTS. The code for the message server can be found in Appendix C.3.

4.5.8 GPS Logger

The GPS logger uses the gpsd package to open up a socket to the incoming GPS data. This socket is then watched and each new packet that comes from the GPS is parsed. The components that are logged into a text file are the time, latitude and longitude. This software was largely based on demonstration of its use for the gpsd repository. The code is written in python and can be found in Appendix C.4.

4.6 The Switch and USB Hub

The switch has four USB 3.0 connections and an external power supply of 15W. As seen in figure 4.2, both USBs and the GPS are plugged into the hub. Theoretically, USB 3.0 speeds can reach 5Gb/s or 500MB/s but this will be limited by the 1Gigabit switch, which will cap any speed to roughly 110MB/s (Kingston, 2019). The RAID1 drive will also slow the transfer speeds as all data has to be

duplicated onto each drive. To set up the switch, much like the USB hub, plug it into power and attach the ethernet cables from each device.

5.0 Implementation and Testing

This section will cover how the software and hardware were implemented and how testing was performed. The challenges that were encountered during testing and implementation will also be discussed.

5.1 Harware Implementation

5.1.1 Raspberry Pi

During the set up phase of the Raspberry Pis several challenges were encountered. The first issue was when the Raspberry Pi server was power cycled while installing some of the packages. When it went to power on again, there was an error in the SD card which ended up with the whole card being corrupt and requiring replacement. After further research, this seemed to be a known issue with Raspberry Pis and cheaper SD cards from companies that might lack quality control. The next issue was encountered when setting up the RAID1 drives. The correct UUID wasn't recorded and the Raspberry Pi failed to boot. After attempting to resolve the issue, failling to safely remove the SD card from a windows PC ended up with another corrupt card. As both of these situations were in the middle of installing packages, no important data was lost, only time.

5.1.2 RTC and Clock Drift

To test the clock drift in each of the Raspberry Pis, CTS was used but the code that changes the time was commented out. Whichever Pi was acting as the server would get frequent clock updates from the NTP to keep its own clock drift from affecting the results. The results of these tests can be seen in figure 5.1 and 5.2. The data for these charts do not specify whether the clock is too fast or slow as the absolute difference was used.

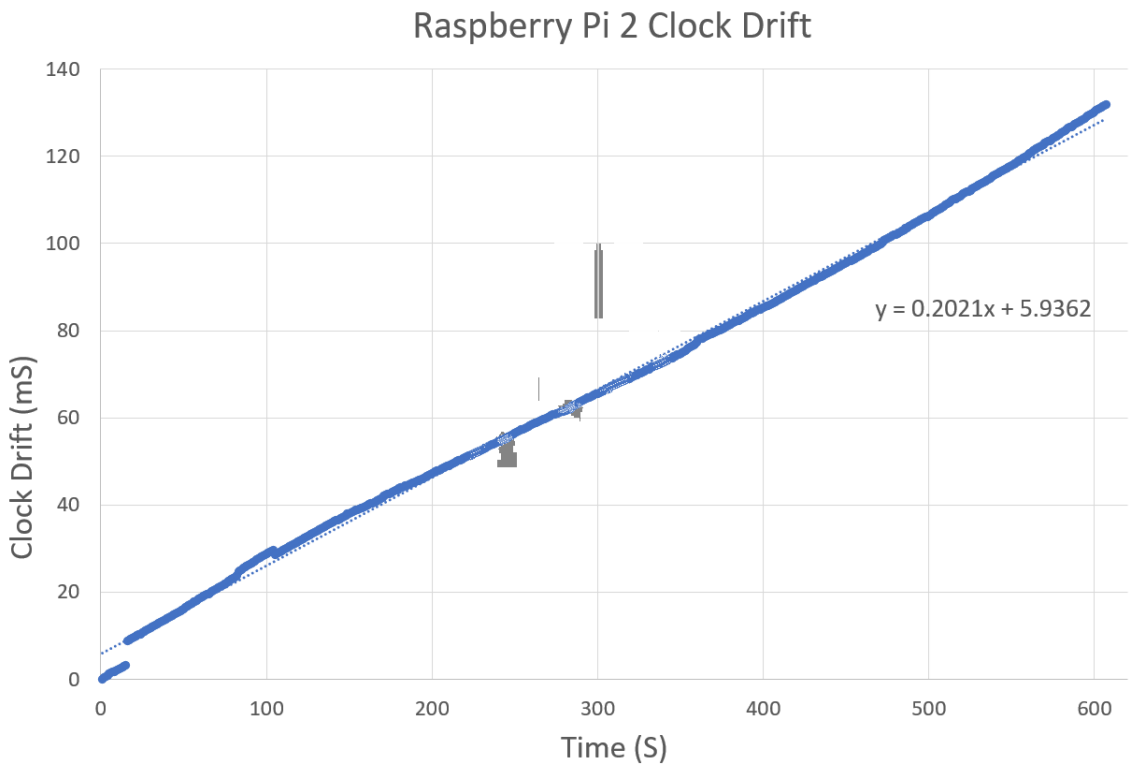


Figure 5.1 - Raspberry Pi 2 clock drift showing roughly 0.2mS drift per second.

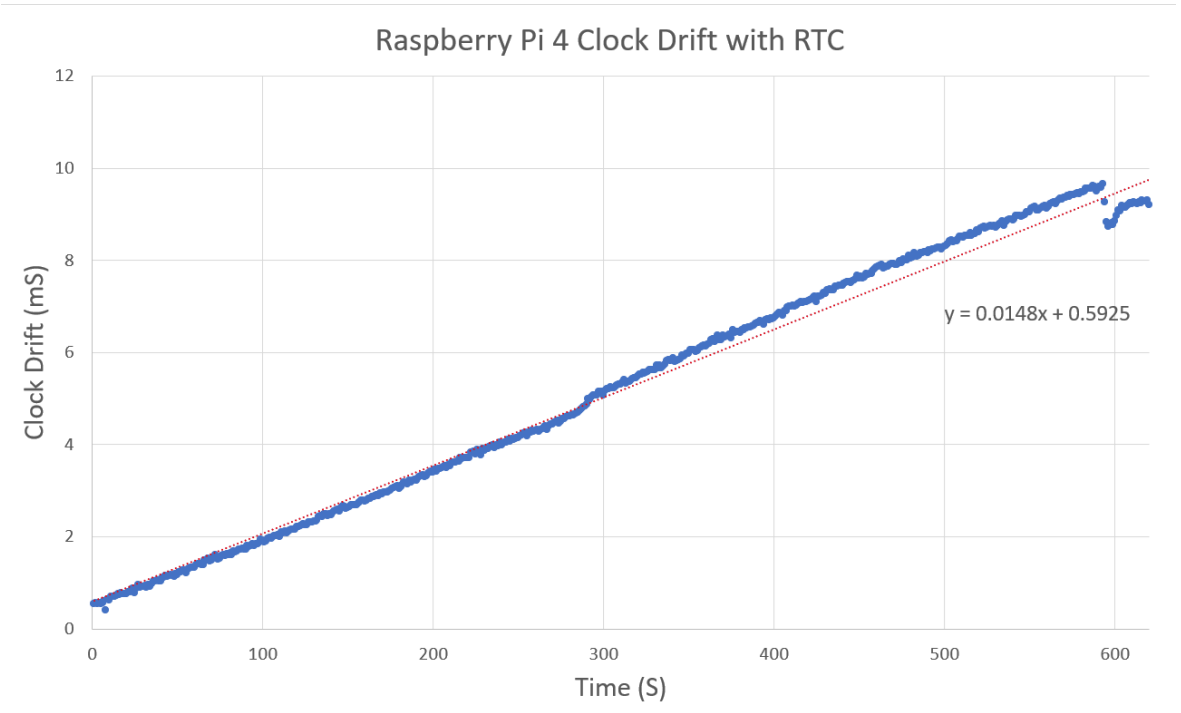


Figure 5.2 - Raspberry Pi 4 clock drift with the ds1307 RTC showing roughly 0.015mS drift per second.

The drift from the Raspberry Pi 4 and the RTC is as expected from the ds1307. This equates to a 15ppm acuraccy while the internal clock on the Raspberry Pi 2 has an error rate closer to 200ppm. This mean that if NTP sent updated time every minute, which is relatively standard, it would de-sync by roughly 12 milliseconds. During testing, it was discovered that there is no surface mounted battery for either Raspberry Pi to keep time. Each time the Raspberry Pi was booted, it would take a few minutes to resynchronise the clocks. This was shorter for the Raspberry Pi server as the RTC has a small capacitor to keep time. On an airborne mission where there is possibly no access to internet, the time differences between different devices may vary by an extreme amount.

5.1.3 Battery System

Setting up the battery system with the inverter and lead-acid battery worked as planned. Unfortunately the inverter was a modified sinewave inverter rather than the advertised pure sinewave inverter. There was no access to an oscilloscope to test the inverter but research into the model showed this. A modified sinewave inverter produces voltage that switches polarity instead of a smooth sinewave. This can affect electrical equipment that require a sinewave such as AC motors and sensitive medical equipment. As there is no equipment that requires efficiency of a pure sinewave in this project, the battery system was tested enough to show that it works. Most power measurements used a household powerpoint.

When measuring the amount of power the system uses, a smart home plug that can monitor power was used. No documentation could be found on the product but it measures power to one decimal place and updates every 10 seconds. The power level was taken every minute for a twenty minute period. The tests included: running the CTS between the server and client, running just NTP, doing nothing, transferring a file from client to server and running CTS while transferring a file. The results can be seen in figure 5.3.

From figure 5.3 , It can be seen that the power usage doesn't reach half of the max power draw. As seen in table 5.1, the maximum would be 43W. Both NTP and the CTS had almost no impact on the power draw. To test if more clients connecting to the server boosted the power usage, multiple terminals were opened on the client and connected to the server as seen in figure 5.3. Even with 12 clients all connected to the server delivering the time once per second, there was no significant change in power usage. As this power usage is minimal compared to the rated inverter load, the inverter efficiency will be low. If it's assumed that the maximum of 12W is sustained during use and that an ideal inverter (efficiency of 85%) is used, this prototype system could be sustained by the battery (7.2Ah) for over five hours (Pearsall, 2016).

Device	Maximum Rated Power (W)
Raspberry Pi 4	15
Raspberry Pi 3	10
USB Hub	15
Network Switch	3

Table 5.1 - Maximum rated power of devices in the system.

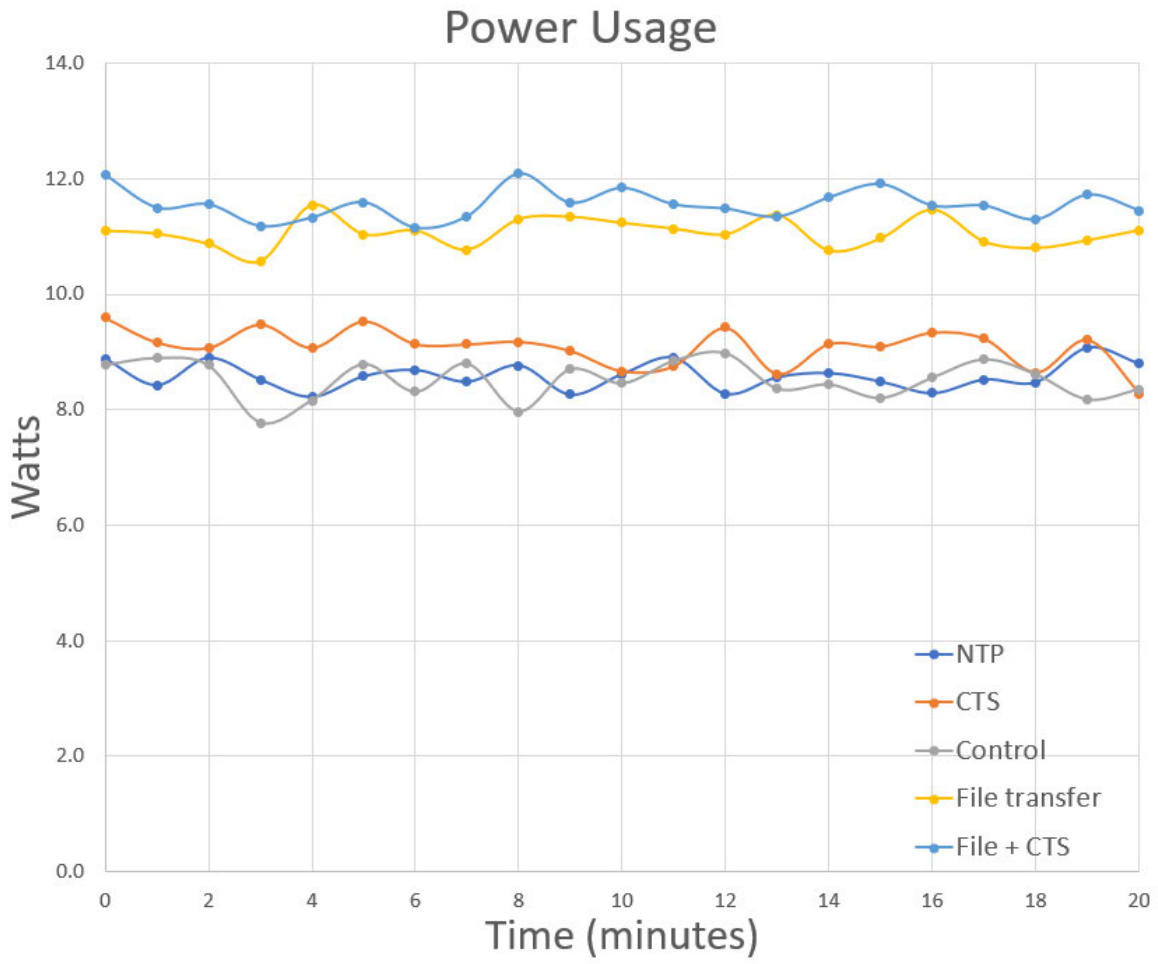


Figure 5.3 - Power usage of the management system with one client.

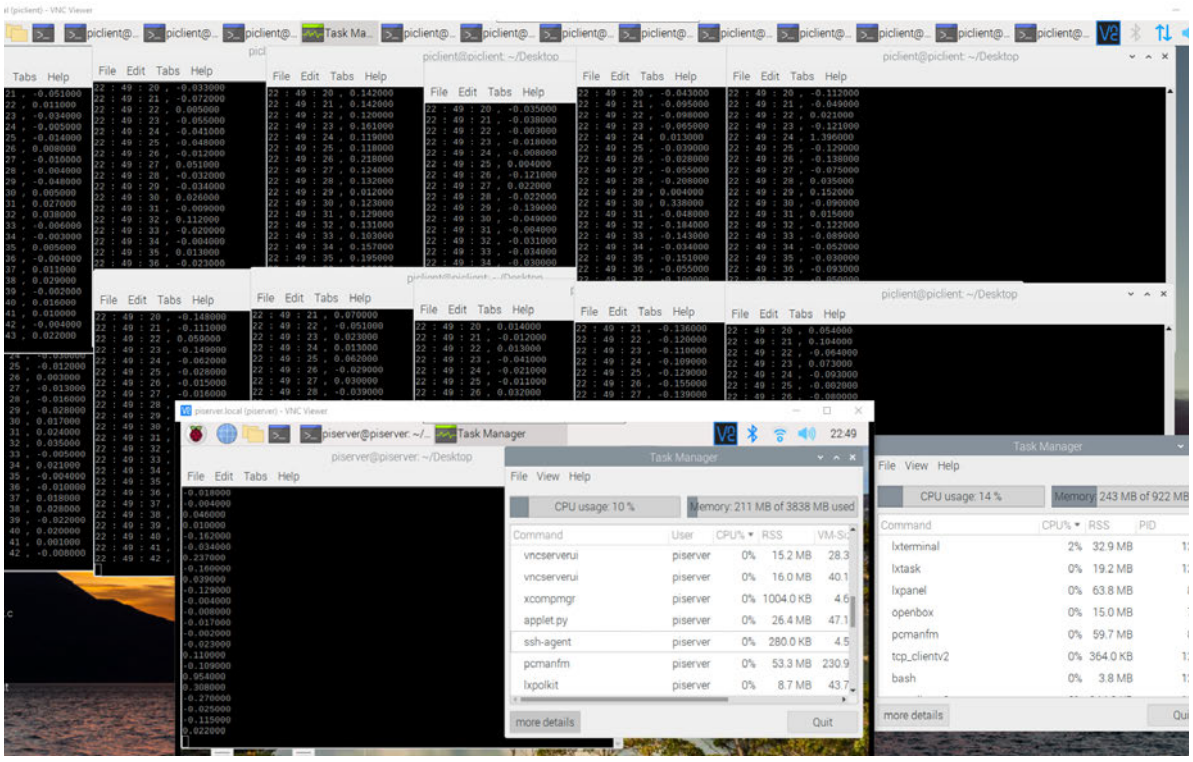


Figure 5.4 - Multiple terminals running the client software and connected to the server, all updating the clock.

5.1.4 GPS

The VK-172 GPS has an issue with reception. It is in a self contained plastic house with no external antenna. When powered, it has a red led that blinks green every second when receiving GPS information from satellites. To ensure connectivity, a USB extension cable was used to place the GPS just outside the window. With its inability to find a signal indoors, its performance inside an airplane will be minimal. When it was working, the latitude and longitude measurements were within the specification of ±0.5m. The read-out did fluctuate even when the GPS was held still.

5.1.5 NAS

Setting up the NAS resulted in some frustrating issues. The boot folder became corrupt when the word UUID was used when setting up the RAID1 drive. When it started working as intended, the transfer speeds achieved were less than expected, as seen in table 5.2. USB 2.0 have a max transfer speed of 43MB/s yet the Raspberry Pi USB 3.0 is peaking at 9.9MB/s (Vakuntula). To fix the issue, the drives were erased so that the set up could be done again. Once the reset was finished, the transfer speeds had not changed. After some more research, RAID1 will slow transfer write speeds to the drive and the CPU of the Raspberry Pi may be throttling the transfer as samba is very single threaded.

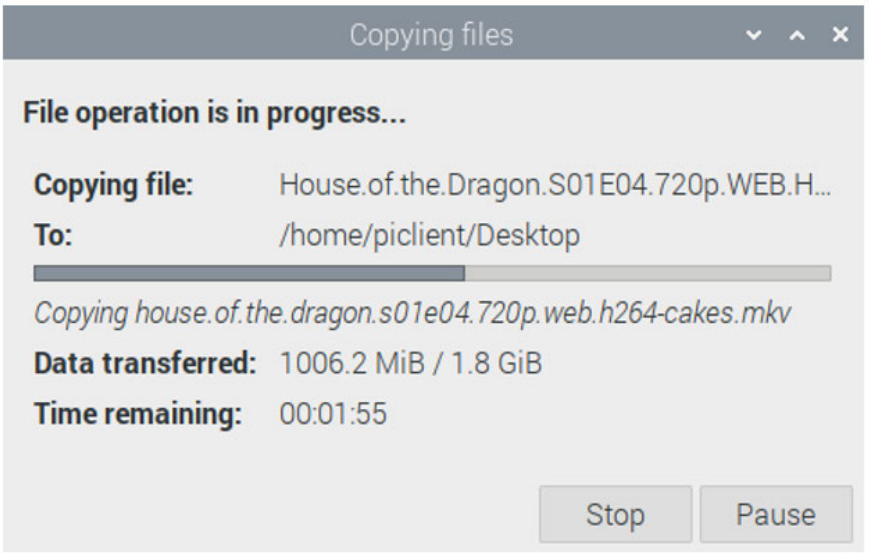


Figure 5.5 - Transferring file from server to client.

Size of file - MB	1863.31	
Path	seconds	MB/s
Client to Server	188	9.91
Server to Client	248	7.51
Server to Windows	496	3.76
Client to Server with CTS	540	3.45

Table 5.2 - Time to transfer and calculated MB/s

5.2 Software Implementation

5.2.1 CTS and NTP

When first implementing the CTS, working out how to properly use sockets was the first priority. Once it was discovered that a time structure could be directly transferred though the socket, getting the client to change time was the next step. After researching the sys/time.h include file, a server that sends a time to the client who then sets that time was achieved. From there, it was working out how to effectively find the difference in time. Using the clock() function to find out how long parts of the code was taking, the time from receiving the server time to setting it was minimised. Once the server was consistently updating the clients time, steps had to be taken to ensure no other packages were messing with the clients time. Using ufw, all outside connections were blocked and data could start being collected. The different iterations that were tested included:

1. Server getting updated time from NTP and GPS and setting the client with NTP
2. Server getting updated time from NTP and GPS and setting the client with CTS
3. Server not getting updated time and setting time of client with NTP
4. Server not getting updated time and setting the client with CTS

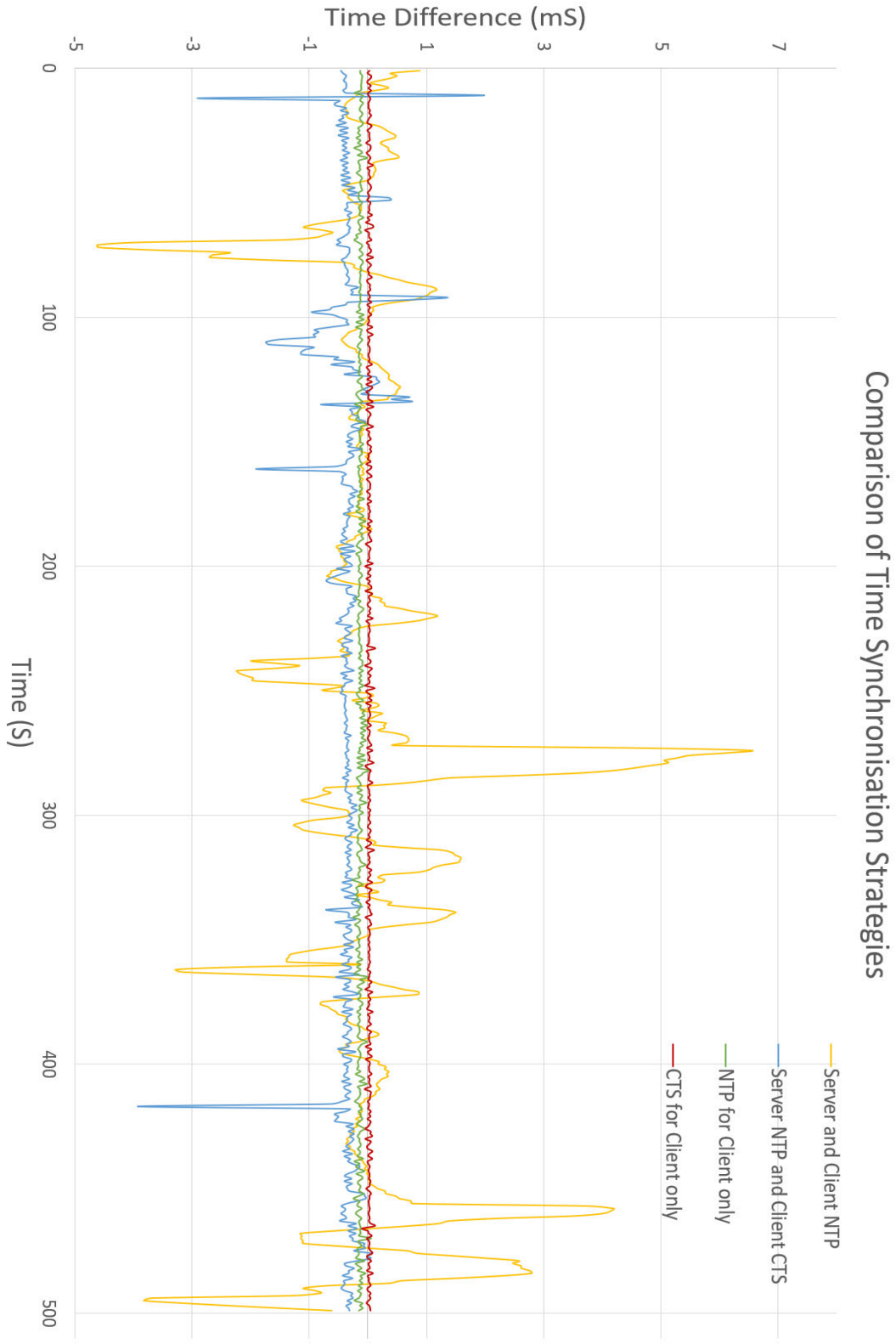


Figure 5.6 - Comparison of the different time synchronisation strategies.

For figure 5.6, data was collected each second and the graph shows a line to make it easier to see each of the results. Each test was run separately so any correlation is coincidence.

From figure 5.6, it can be seen that all of these time synchronising strategies keep the devices within ten milliseconds of the server time. The delivery time is found beforehand using the ping command and there is roughly 12uS between the client receiving the server time and setting it. With these two sources of error, the data from figure 5.6 should have error bars that are no bigger than ± 0.1mS. When comparing the results, it needs to be noted that the NTP and CTS for only the client results may not have accurate time just very precise time. Any clock drift in the Raspberry Pi server would be directly transferred to the clients. When the client is kept synchronised with the NTP, there seems to relatively large changes in the time which are then corrected. When compared to the CTS result, it can be seen how the NTP dampens any abrupt change in time. The CTS has many sharp changes while the NTP has relatively smooth peaks and valleys. Looking at the more precise results, the NTP seemed to have an offset of 0.1mS while the CTS seems centred around the baseline. A contributing factor of this offset for the NTP is most likely a result of the clock drift of the client. Previously it showed at 0.2 millisecond drift per second. This make the CTS result questionable as it maintains the accuracy even with the drift.

This clock drift issue was encountered after compiling the data for figure 5.6, similar results could not be obtained from the CTS. No matter what code was reverted or what aspects of the NTP were turned off and on, the accuracy around the baseline were unrepeatable. Figure 5.7 shows four separate tests that used the same server and client scripts. This is a closer view showing only 200 data points of each test to see the small differences between them. The red plot along the baseline is the same as the red plot in figure 5.6. The other three tests were performed after the “perfect” result. The offset found using the ping command directly before performing the test all gave the same result of 0.45mS average ping. Though it was later realised, no matter what the offset was, the current time would always be compared to server time plus the offset. These differences in performance may be due to delivery time it was also notices that the performance of the Raspberry Pi 2 client seemed to have declined. During tests, where the client wasn’t receiving any time updates, the first time difference when CTS started would be relatively large. To test this theory that the software clock of the client had degraded further, a clock drift test was performed and the results can be seen in figure 5.8. As before, the clock drift is an absolute value so they can be more easily comparable. The result was surprising at first, but when seeing that the drift is around 0.36 milliseconds per seconds and the results from figure 5.7 show a 0.2 - 0.6 millisecond offset, the knowledge that there wasn’t an oversight in the software was reassuring. This discovery explains what was causing the offset and may be be the start of larger hardware issues in the older Raspberry Pi.

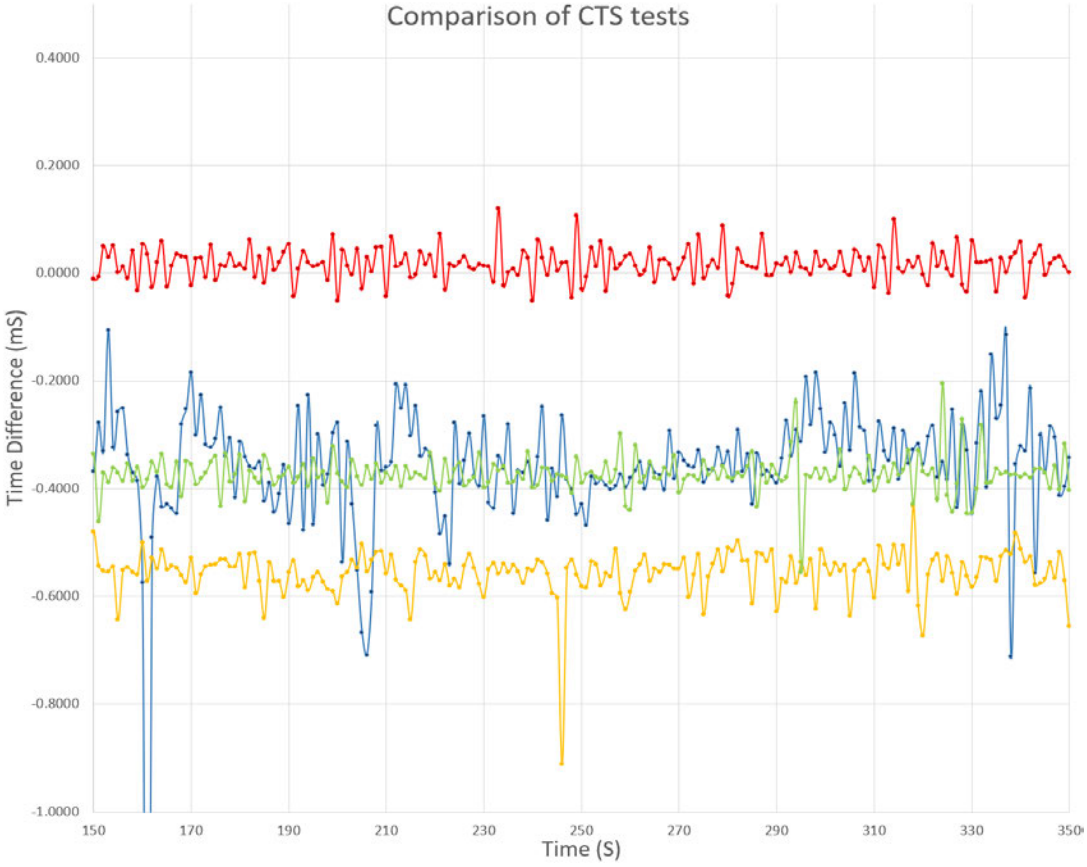


Figure 5.7 - Comparison of four separate CTS sessions.

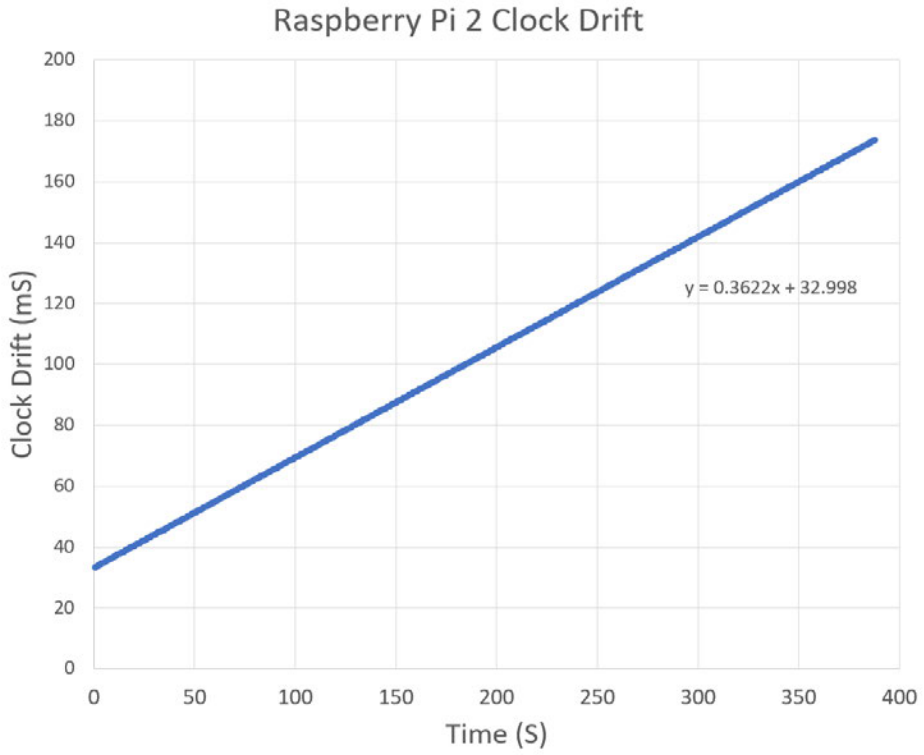


Figure 5.8 - Clock drift of Raspberry Pi 2 client after a noticeable drop in performance.

While the perfect result is still unexplained, later results show that the clocks of even the most uncooperative clients can be kept to within a millisecond of the server. The NTP can also achieve similar results with an arguably easier time to set up. Either process of keeping the time syncronised will work well for an airborne observation mission.

5.2.2 Message server

The results of the message server were mostly successful. One of the issues that arose was with the message variable not being erased after each use. Any message that was shorter that the last from the same process, printed a message that contained the short message overlapped on the long message. this was easily fixed by resetting the memory of the variable with the memset() function. The last issue, which is still occurring, is that some characters seem to print gibberish to the text file. This was mainly with question marks but now they seem to work fine. As seen in figure 5.9, when a client connects to the server, an HTS symbol is printed below the message and when an exclamation mark is used in a message odd characters are shown as can be seen on line 77. At first it was thought there might be a memory overflow issue but the message variable is now reset and is currently 2046 characters long. Otherwise the message server works as expected. The time is printed as well as the identifier the clients choose. If this is being used to take a quick note of something that just happened suring the airborne mission, this could work quite well. If a more experienced C programmer inspects the code, there is most likely an easy fix.

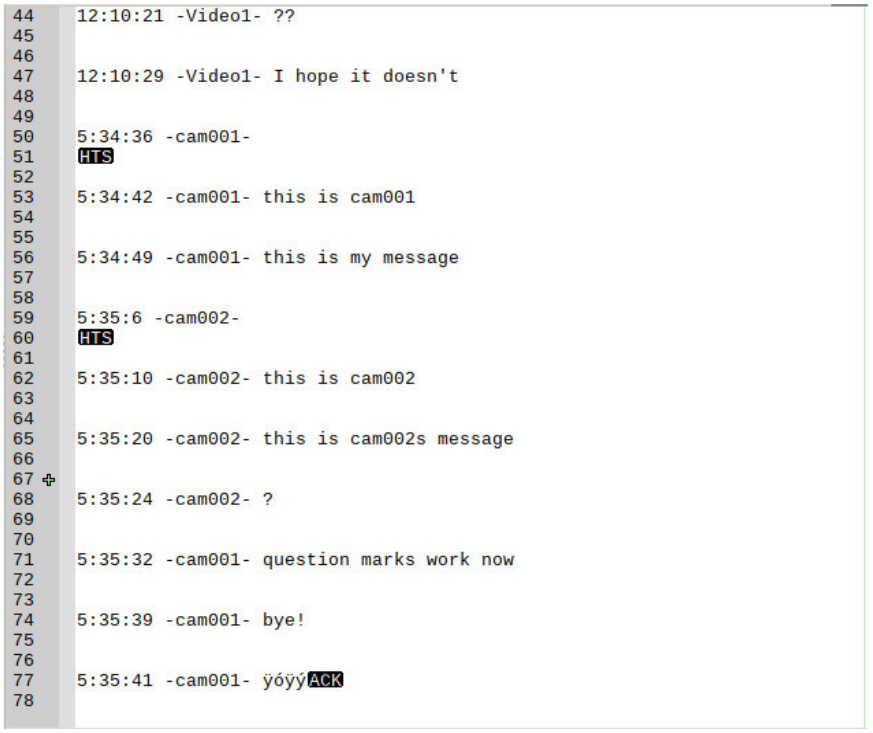


Figure 5.9 - Message log text file

5.2.3 GPS logger

When researching how to read the GPS data using C programming language, many examples were using a GPS that used the GPIO pins. These example used the serial include to read the data stream and parse the latitude and longitude from it. This was an issue for this project as the GPS chosen uses a USB interface. Eventually the the GPS logger was found researching the gpsd package for linux. It is used as an example of python code that can parse the GPS data that gpsd is looking at. Some time was spent learning and adding some extra code that printed the time, latitude and longitude to a GPS log text file.

6.0 Conclusions and Further Work

6.1 Conclusions

6.1.1 Battery System

The battery system created for this project achieved the aims that were set out in the project specification. The Raspberry Pi 4 server has a maximum rated power usage of 15W but in testing, it never came close to that. It was only when transferring files to and from the NAS storage that there was a noticeable increase in usage from the system. While there was no testing that used laptops, this battery systems goal was to prove that it can work. If a this type of system is used for an airborne

observation mission, the recommendation is to use a UPS or an inverter that has the option to charge the batteries when connected to mains. A battery backup would reduce the needed battery capacity. If battery power is required for the mission, 12V lithium batteries would be ideal for there large capacity, smaller form factor, consistent discharge voltage and lighter weight. (Powersonic, 2022)

6.1.2 RTC

From the results, the RTC seemed to improve the Raspberry Pi 4 clock by more than a factor of 10. Later in the testing, the Rasbperry Pi 2 clocked deteriorated and there was no way to know if these performance issues happen before this project started. Therefore, the clock drift results can't be compared. However it was discovered that Raspberry Pis don't have a hardware clock. Consequently, any project that uses a Raspberry Pi where there is any chance that internet connection is inconsistant, should use an RTC.

6.1.3 GPS

For the use on an airborne mission, the VK-172 GPS would not work. To receive GPS data, the dongle must have direct line of sight with the sky for accurate results. While it can work indoors, reception is inconsistent. The GPS does not have an option to attach an antenna which some similar models do. After further research of the VK-172, it was found that it has guidelines on how well it performs in the air. It's maximum horizontal velocity is 100m/s but has a "large" maximum position deviation. The highest altitude it works at won't be an issue as its 50km. While this particular GPS wouldn't be appropriate for an airborne mission, a higher grade GPS with better antennas would help with keeping the the time accurate while the NTP or CTS kept the high time precision.

6.1.4 NAS

The NAS caused the most issues for this project and was unable to perform as expected. While it did work, the transfer rate was underwhelming. There are serveral users that have a similar performance issue with the Raspberry Pi and samba NAS but none of the solutions seemed to work. If used on an airborne mission, the recommendation would be to transfer over the network one device at a time after the data has been collected. Depending on the amount of data that needs to be transferred, this could take several hours. If the devices aren't needed before the plane lands, then this may not be an issue.

6.1.5 CTS

The CTS program developed is not perfect but it seems to perform on par with NTP. As the CTS doesn't calculate the roundtrip time or the offset delay automatically, it would not work effectively in a system where these values are changing. This also means the offset delay needs to be manually calculated for each device and entered into the code. Once CTS is set up, it can keep the clients time to within 1 millisecond of the server time even if the client has extreme clock drift. There is room to improve the efficiency of the code and many parts of it could be put into subroutines. When it comes to the style of iteration that was used, the project was more CFI than DFI. During development, each new objective was built on what was already there rather than factorizing and streamlining the design. While this isnt a major issue for such a small piece of software, the ability to add features is diminished.

6.1.6 Message Server

The messaging server performs well but still has some issues with the characters that are accepted. With more time dedicated to improving the code, these issues can be overcome. This code suffers from the same design issues as the CTS and would be difficult for it to be integrated into another system.

6.1.7 Project Objectives and Outcomes

6.1.7.1 Objective 1

The has been shown to support up to 12 processes that are updating time. While the battery system and the switch could not support this many devices, they can be easily upgraded so that they can. The only concern raised from this project is transferring data from the clients to the NAS. Given enough time, this transfer speed may not be an issue but when important data is involved the slow transfer rate could be stressful.

6.1.7.2 Objective 2

From the beginning of the project, the Raspberry Pi was the clear choice of microcontroller. It has an excellent cost for performance ratio and has a large community that provide troubleshooting and packages for almost project. Its performance during this project proved that it is up to the task. The only aspect of the Raspberry Pi that may have an issue is the CPU speed that might be throttling transfer speeds using samba.

6.7.1.3 Objective 3

Both NTP and the CTS were successfully used during this project and the performance of the CTS shows that it can match NTP in most situations on a wired network.

6.1.7.4 Objective 4

While the use of a GPS helped keep the time accurate, its inability to receive a signal indoors shows that it would perform poorly in an airborne observation mission. Other GPSs may be better suited but that can be convered in future work.

6.1.7.5 Objective 5

The Raspberry Pi used as the client had terrible clock drift which got worse as more tests were performed. However, once NTP or CTS were used, the clients time was kept within a millisecond of the server time.

6.1.7.6 Objective 6

With the battery system, it was decided to use an inverter so that non standard voltages were not an issue. This would increase the losses in the system but increase its useability. If the system is reliant on the batteries for power, two average car batteries or one 12V lithium ion battery should be used for every six devices connected.

6.1.7.7 Objective 7

When choosing the size of storage, it was decided to use USBs as a proof of concept. The Raspberry Pi server and switch are both 1Gb/s capable and the two USB 3.0 ports are theoretically capable of 4.8Gb/s. In practice these speeds were far from being reached. Further work is needed on why it didn't reach acceptable speeds and different software solutions that could achieve these speeds

6.1.7.8 Objective 8

RAID1 was used for the NAS which mirrors each drive. This would provide whole drive redundancy and improve reliability.

6.2 Future Work

6.2.1 Single program

Work can be done joining each of the software packages into one complete package. This would improve usability and would require improved code design.

6.2.2 Screen

A screen for the Raspberry Pi server would provide immediate information on how each client is synchronised and possibly the amount of battery left in the battery system.

6.2.3 GPS and RTC

Further work should be done the use of a better GPS that has an improved antenna. If GPS coordinates aren't needed, a highly accurate RTC could be used to keep time accurate while the NTP or CTS keeps it precise.

6.2.4 EMS

The electromagnetic connectivity of the entire system should be tested. This is especially true for systems that are to be used on a plane.

6.2.5 Wireless

The convenience of a system with only power cable could be realised with a wireless network rather than a wired one.

6.2.6 NAS

As mentioned, further work should be done on improving the speed of the NAS. Currently the average speed tops out at 9MB/s which is significantly worse than theoretical speeds.

6.2.7 Microprocessor

The use of a different microprocessor could be researched or even the use of an FPGA.

References

IEEE Standard for a precision clock synchronization protocol for networked measurement and control systems, 2009, IEEE, 2-8318-1026-4.

Aamir, M, Ahmed Kalwar, K & Mekhilef, S 2016, 'Review: Uninterruptible Power Supply (UPS) system', Renewable & sustainable energy reviews, vol. 58, pp. 1395-410.

Austin 2021, Millisecond accurate Chrony NTP with a USB GPS, viewed 1/09/22 2022, <https://austinsnerdythings.com/2021/09/29/millisecond-accurate-chrony-ntp-with-a-usb-gps-for-12-usd/>.

Bell, K 2018, History of Airborne Astronomy at NASA, NASA, <https://www.nasa.gov/feature/history-of-airborne-astronomy-at-nasa>.

Blattau, N & Hillman, C 2004, 'Failure Mechanisms in Electronic Products at High Altitudes', CALCE Electronic Products and Systems Center.

Chao, C, Huang, S & Hung, H 2009, 'Embedded System on NTP', paper presented to 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology, 24-26 Nov. 2009.

Chen, PM, Lee, EK, Gibson, GA, Katz, RH & Patterson, DA 1994, 'RAID: High-performance, reliable secondary storage', ACM Computing Surveys (CSUR), vol. 26, no. 2, pp. 145-85.

Curnow, R 2021, Chrony, viewed 1/09/22 2022, <https://chrony.tuxfamily.org/>.

Cynthia Stackpole, S 2013, A User's Manual to the PMBOK Guide, Fifth Edition, Wiley.

die.net 1994, telnet - user interface to the TELNET protocol, viewed 1/9/22 2022, <https://linux.die.net/man/1/telnet>.

---- 2010, mdadm - manage MD devices aka Linux Software RAID, <https://linux.die.net/man/8/mdadm>.

Dinar, AE, Merabet, B & Ghouali, S 2021, 'NTP Server Clock Adjustment with Chrony', in Applications of Internet of Things, Springer, pp. 177-85.

Dueck, M, Schloesser, M, Kaparaki, M, Srivastava, S, Waasen, Sv & Schiek, M 2014, 'Raspberry Pi based testbed verifying TrueTime network model parameters for application in distributed active turbulent flow control', paper presented to 2014 Proceedings of the SICE Annual Conference (SICE), 9-12 Sept. 2014.

Dueck, M, Schloesser, M, van Waasen, S & Schiek, M 2015, 'Ethernet based time synchronization for Raspberry Pi network improving network model verification for distributed active turbulent flow control', Control theory and technology, vol. 13, no. 2, pp. 204-10.

Fairbanks, G 2022, 'Two Kinds of Iteration', IEEE Software, vol. 39, no. 1, pp. 114-7.

gpsd 2021, gpsd — a GPS service daemon, viewed 10/10/22 2022, <https://gpsd.io/index.html#documentation>.

Grinstead, J, Jenniskens, P, Cassell, A, Albers, J & Winter, M 2011, 'Airborne observation of the Hayabusa sample return capsule re-entry', paper presented to 42nd AIAA Thermophysics Conference.

Guerrero, JM, Luis Garcia de, V, Matas, J, Castilla, M & Miret, J 2005, 'Output impedance design of parallel-connected UPS inverters with wireless load-sharing control', IEEE transactions on industrial electronics (1982), vol. 52, no. 4, pp. 1126-35.

Hertel, CR 1994, Samba Documentation, viewed 2/9/22 2022, <https://www.samba.org/samba/docs/>.

Ho, C-C, Chang, Y-M, Chang, Y-H & Kuo, T-W 2018, 'An SLC-Like Programming Scheme for MLC Flash Memory', ACM transactions on storage, vol. 14, no. 1, pp. 1-26.

Horvath, TJ, Cagle, MF, Grinstead, JH & Gibson, DM 'Remote observations of reentering spacecraft including the space shuttle orbiter', 2013.

Im, S & Shin, D 2011, 'Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD', IEEE transactions on computers, vol. 60, no. 1, pp. 80-92.

Integrated, M 2002, DESIGN CONSIDERATIONS FOR MAXIM REAL-TIME CLOCKS, viewed 1/10/22 2022, <https://www.maximintegrated.com/en/design/technical-documents/app-notes/5/504.html>.

---- 2015, DS1307 - 64 x 8, Serial, I2C Real-Time Clock, viewed 10/09/22 2022, <https://datasheets.maximintegrated.com/en/ds/DS1307.pdf>.

Intergrated, M 2015, DS3231 RTC, Maxim Intergrated.

Jenniskens, P 2016, Cygus Shallow re-entry observation campaign, SETI institute, <http://atv5.seti.org/cygnus/>.

Jolles, JW 2021, 'Broad‐scale applications of the Raspberry Pi: A review and guide for biologists', Methods in ecology and evolution, vol. 12, no. 9, pp. 1562-79.

Kingston 2019, What’s the Difference in USB 3.1 Gen 1, Gen 2 and USB 3.2?, viewed 10/10/2022 2022, <https://www.kingston.com/en/usb-flash-drives/usb-30>.

Leandro 2021, Raspberry Pi RAID NAS Server Setup.md, <https://gist.github.com/leandrofilipe/f9636be272f97d414652ce1f21e6b1f4.js>.

Litos, G, Zabulis, X & Triantafyllidis, G 'Synchronous Image Acquisition based on Network Synchronization', 2006.

Löhle, S, Zander, F, Lemmens, S & Krag, H 2017, 'Airborne Observations of Re-entry Break-up Results and Prospects', paper presented to Proceedings of the 7th European Conference on Space Debris.

Lombardi, M 2000, 'Computer time synchronization', National Institute of Standards and Technology, pp. 1-2.

Matson, J 2013, 'Choosing the correct Time Synchronization Protocol and incorporating the 1756-TIME module into your Application', Rockwell Automation, May.

Mielke, N, Goodwin, K, Harris, R, Kumar, A, Lin, E, Parekh, V, Zhang, B & Zweig, M 2015, 'Accelerated Testing of Radiation-Induced Soft Errors in Solid-State Drives', IEEE Transactions on Device and Materials Reliability, vol. 15, no. 4, pp. 552-8.

Mills, D 2006, Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI, 2070-1721.

Mills, DL 1981, DCNET internet clock service, 2070-1721.

---- 1991, 'Internet time synchronization: the network time protocol', IEEE Transactions on communications, vol. 39, no. 10, pp. 1482-93.

---- 2006, 'Network time protocol version 4 reference and implementation guide', Electrical and Computer Engineering Technical Report, pp. 06-.

Pearsall, N 2016, The performance of photovoltaic (PV) systems: modelling, measurement and assessment, Woodhead Publishing.

Pereira, RIS, Dupont, IM, Carvalho, PCM & Jucá, SCS 2018, 'IoT embedded linux system based on Raspberry Pi applied to real-time cloud monitoring of a decentralized photovoltaic plant', Measurement : journal of the International Measurement Confederation, vol. 114, pp. 286-97.

Pi, R 2022, raspberrypi.com, <https://www.raspberrypi.com/>.

PiHut, T 2015, Adding a Real Time Clock to your Raspberry Pi, <https://thepihut.com/blogs/raspberry-pi-tutorials/17209332-adding-a-real-time-clock-to-your-raspberry-pi>.

Powersonic 2022, The complete guide to lithium vs lead acid batteries, viewed 10/10/22 2022, <https://www.power-sonic.com/blog/lithium-vs-lead-acid-batteries/>.

Richards, M 2018, Raspberry Pi Buster – GPS Dongle as a time source with Chrony & Timedatectl, 2022, <https://photobyte.org/raspberry-pi-stretch-gps-dongle-as-a-time-source-with-chrony-timedatectl/>.

Rizvi, SS & Chung, T 2010, 'Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems', paper presented to 2010 2nd International Conference on Computer Engineering and Technology, 16-18 April 2010.

Rybaczyk, P 2005, Expert Network Time Protocol An Experience in Time with NTP, 1st ed. 2005. edn, Expert's Voice, Apress, Berkeley, CA.

Shrivastava, AR & Gadge, J 2017, 'Home server and nas using raspberry PI', paper presented to 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 13-16 Sept. 2017.

Slayman, C 2011, 'Soft error trends and mitigation techniques in memory devices', paper presented to 2011 Proceedings - Annual Reliability and Maintainability Symposium, 24-27 Jan. 2011.

Snively, JB, Taylor, MJ & Jenniskens, P 2011, 'Airborne imaging and NIR spectroscopy of the ESA ATV spacecraft re-entry: instrument design and preliminary data description', International Journal of Remote Sensing, vol. 32, no. 11, pp. 3019-27.

Strom, BD, Lee, S, Tyndall, GW & Khurshudov, A 2007, 'Hard Disk Drive Reliability Modeling and Failure Prediction', IEEE Transactions on Magnetics, vol. 43, no. 9, pp. 3676-84.

Tack, S, Horvath, T, Tomek, D, Verstynen, H & Shea, E 2010, 'Cast Glance Near Infrared Imaging Observations of the Space Shuttle During Hypersonic Re-entry', paper presented to 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition.

Tjoa, R, Chee, KL, Sivaprasad, PK, Rao, SV & Lim, JG 2004, 'Clock drift reduction for relative time slot TDMA-based sensor networks', paper presented to 2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754), 5-8 Sept. 2004.

---- 2004, 'Clock drift reduction for relative time slot TDMA-based sensor networks', paper presented to 2004 IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE Cat. No.04TH8754), 5-8 Sept. 2004.

Tovar, JC, Hoyer, JS, Lin, A, Tielking, A, Callen, ST, Elizabeth Castillo, S, Miller, M, Tessman, M, Fahlgren, N, Carrington, JC, Nusinow, DA & Gehan, MA 2018, 'Raspberry Pi–powered imaging for plant phenotyping', Applications in plant sciences, vol. 6, no. 3, pp. e1031-n/a.

Watt, ST, Achanta, S, Abubakari, H, Sagen, E, Korkmaz, Z & Ahmed, H 2015, 'Understanding and applying precision time protocol', paper presented to 2015 Saudi Arabia Smart Grid (SASG).

Xue, M & Zhu, C 2009, 'The Socket Programming and Software Design for Communication Based on Client/Server', paper presented to 2009 Pacific-Asia Conference on Circuits, Communications and Systems, 16-17 May 2009.

Zander, F, Buttsworth, DR, Birch, B, Noller, L, Wright, D, James, CM, Thompson, M, Apirana, S, Leis, J & Lobsey, C 2021, 'Australian rapid-response airborne observation of the Hayabusa2 reentry', Journal of Spacecraft and Rockets, vol. 58, no. 6, pp. 1915-9.

Appendix A – Project Specification

ENG4111/4112 Research Project

Project Specification

For: Robert Schuster

Title: Airborne Observation Controller

Major: Electrical and electronic engineering

Supervisor: John Leis

Enrolment: ENG4111 – EXT S1, 2022

ENG4112 – EXT S2, 2022

Project Aim: To design and build a portable management system for aircraft-borne observations. This system should be able to synchronise the time of all connected devices, record GPS data, record aircraft logs as needed, and manage data collected from connected devices. As this data is critical, battery and file storage redundancies should be used to enhance data storage reliability and ensure the integrity of all information stored.

Programme: Version 4, 22 March 2022

1. Determine appropriate requirements to support a network of instruments onboard the aircraft for an observation mission.
2. Determine an appropriate processor speed and memory size and review currently available microcontrollers to use as the core of the management system.
3. Review the network time protocol to design code that uses a custom time protocol for the chosen microcontroller to synchronise connected devices.
4. Investigate the use of GPS hardware and determine the feasibility of incorporating it into the microcontroller with the custom time protocol.
5. Test the microcontroller and connected devices for synchronisation accuracy.
6. Investigate the current and voltage needed to support the network of instruments and the ability to deal with non-standard voltages. Review these parameters and determine a suitable battery size to ensure reliability and redundancy.
7. Review the amount of data that is to be collected from devices and choose appropriate file storage approaches as well as the capability of ethernet and USB transfer rates on the microcontroller.
8. Investigate the use and implementation of a redundant array of independent disks (RAID) to add a level of redundancy and reliability to the chosen storage approach.

If time and resources permit

9. Implement a GUI and interface to control and observe the microcontroller directly.
10. Test the portable management system for radio frequency and electromagnetic interference.

Appendix B – Project Resources

ENG4111/4112 Research Project

Project Resources

For: Robert Schuster

Title: Airborne observation controller

Major: Electrical and electronic engineering

Supervisors: John Leis

Enrolment: ENG4111 – EXT S1, 2022

ENG4112 – EXT S2, 2022

Equipment that will be needed:

- Microcontroller – Raspberry Pi 4 model B.
- TP-Linknk 8 port Gigabit switch 3W.
- Vk 172 Gmouse USB GPS receiver u-blox7
- DS1307 Real-time clock.
- Multiple devices to test time synchronisation – Max Number of devices that can connect to the switch is 8.
- LAN modules – laptops and some microcontrollers may not have a LAN connector built-in which will require a LAN to USB or mini-USB converter.
- Lan cables to connect devices to switch – Cat 5e
- Lead-acid battery – 12V 7.2Ah
- 1000W Inverter

Appendix C - Raspberry Pi Server time synchronisation

```
#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <signal.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#include <sys/time.h>

#define PORT 9002

#define SLEEP_TIME 1

/*

When this code is run, clients can connect to the server using its IP address.

Once connected the server will immiediately start sending the client server time.

the difference should be replied by the client. As this process forks when a client

connects, max number of connections should be dependant on CPU availability. While

it is not neccesary, testing was done by running this code with root privelages

Variables

Ints

- sock is the socket used to connect to clients

- client_len is the length of the client_address struct

- isRecv and isSend are the integers used to indicate successfull send and recieve
```

```
- client_socket indicates the connection status with the client
Structs
- server_address is the socket address of the server
- client_address is the socket address of the client
- now and diftime are tival structs used to get the time and show the time difference
    between client and server
- ticks is used to get the current time and is the number of seconds since the start
    of the Epoch.
*/
int main()
{
    int sock, client_len, isSend, isRecv, client_socket;
    time_t ticks;
    struct sockaddr_in server_address, client_address;
    struct timeval now, diftime;

    //create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        printf("error creating socket\n");
        exit(1);
    }
    //Fill in server address info
    memset(&server_address, '0', sizeof(server_address));
    server_address.sin_family = AF_INET;
```

```
server_address.sin_port = htons(PORT);
server_address.sin_addr.s_addr = htonl(INADDR_ANY);

//Bind port number to socket
if (bind(sock, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
{
   printf("error binding socket\n");
   exit(2);
}
//set socket to listen
listen(sock, 5);
printf("server listening...\n");
/*
while loop for the parent process. Accepts a client joining the socket and forks.
Child process closes socket and enters second while loop. In this loop, current time
is fetched and sent to the client, the message sent back is the time struct containing
the difference between the server and client time. In this case the difference is
printed in milliseconds and starts the loop again after SLEEP_TIME seconds
*/
while (1)
{
   client_len = sizeof(client_address);
   client_socket = accept(sock, (struct sockaddr *)&client_address, &client_len);
   if (client_socket < 0)
   {
      printf("connection error\n");
```

```
    exit(3);
}
//splits into parent and child process
if (fork() == 0)
{
    close(sock);
    printf("new child (pid%d) using descriptor %d\n", getpid(), client_socket);
    while (1)
    {
        //get time of day and send to client
        gettimeofday(&now, NULL);
        isSend = send(client_socket, &now, sizeof(now), 0);
        if (isSend < 0)
        {
            printf("send error\n");
        }
        isRecv = recv(client_socket, &diftime, sizeof(diftime), 0);
        if (isRecv < 0)
        {
            printf("recieve error\n");
        }
        //converts usecs to msecs and prints, used for checking the return message
        float dif_usec = diftime.tv_usec;
        dif_usec = dif_usec / 1000;
        printf("%f\n", dif_usec);
        if (client_socket < 0)
```

```
            {
                exit(4);
            }
            sleep(SLEEP_TIME);
        }
        printf("pid%d disconnected", getpid());
        close(client_socket);
    }
    else
    {
        //close parent
        close(client_socket);
    }
  }
  close(sock);
  return 0;
}
```

Appendix D - Client time synchronisation

```
#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#include <sys/time.h>

#define OFFSET 450 //in uSecs, average of the ping command between client and server

#define USECS_2_MSECS 1000.0

#define SLEEP_TIME 5

#define IP_ADDRESS "169.254.213.88" //IP address should be confirmed before running

/*

When this code is run, if the server is not already running it will return an error.

This code needs to be run with root privileges to change the time.

If the server is running, it will start updating the time every SLEEP_TIME seconds

and print the diffences between the server time and client time as comma seperated values

Variables

Ints

- sock is the socket to connect to the server

- isRecv and isSend are the integers used to indicate successfull send and recieve

- client_socket indicates the connection status with the client
```

Structs

- server_address is the socket address of the server
- timeval structs to hold the current client time (oldtime) server time (newtime) difference in time (diftime) and the time offset representing the delivery time of messages
- ticks and tm are used to display the time in the commandline outputs

Float

- diff_msec is the calculated millisecond time difference between server and client

Long

- the message delivery time between server and client found by using the ping command

*/

```
int main()
{
    int sock, isRecv, isSend, client_socket;
    struct sockaddr_in server_address;
    struct timeval oldtime, newtime, offset, diftime;
    time_t ticks;
    struct tm tm;
    float diff_msec;
    long ping_time = OFFSET;
    /*
    // this section was used to test the code to ensure the time could be changed this way. In this case
    // it would change the time to the 1st of Jan 2022 12am and print the current time
    ticks = time(NULL);
    newtime.tv_sec = 1640959200;
    settimeofday(&newtime, NULL);
    printf("weird time is: %s", ctime(&ticks));
```

```
*/
//pass the time between server and client (the ping time) to the offset time struct
offset.tv_usec = ping_time;
//setup the socket and set the memory for the server_address
sock = socket(AF_INET, SOCK_STREAM, 0);
memset(&server_address, '0', sizeof(server_address));

// specify an address for the socket to connect to. In this case the server IP is known
server_address.sin_family = AF_INET;
server_address.sin_port = htons(9002);
server_address.sin_addr.s_addr = inet_addr(IP_ADDRESS);
// connects to the socket and returns -1 if failed
client_socket = connect(sock, (struct sockaddr *)&server_address, sizeof(server_address));
if (client_socket == -1)
{
   printf("there was an error making a connection to the remote socket \n");
   return -1;
}
/*
while loop recieves server time and writes it to the newtime struct, gets the current client time
uses the known offset to configure the server time and sets the client time to the modified server time
compares the server time to client time to find the difference and sends it back as a timeval strcut
*/
while (1)
{
```

```
        isRecv = recv(sock, &newtime, sizeof(newtime), 0);
        if (isRecv < 0)
        {
           printf("recieve error\n");
        }
        gettimeofday(&oldtime, NULL);
        newtime.tv_usec = newtime.tv_usec + offset.tv_usec;
        //print statements to confirm what the code is doing
        //printf("%ld.%06ld vs %ld.%06ld\n",
newtime.tv_sec,newtime.tv_usec,oldtime.tv_sec,oldtime.tv_usec);
        settimeofday(&newtime, NULL);
        diftime.tv_sec = oldtime.tv_sec - newtime.tv_sec;
        diftime.tv_usec = oldtime.tv_usec - newtime.tv_usec;
        diff_msec = (oldtime.tv_usec - newtime.tv_usec) / USECS_2_MSECS;
        diff_msec = diff_msec + diftime.tv_sec * USECS_2_MSECS;
        ticks = time(NULL);
        tm = *localtime(&ticks);
        //print statement that has time and comma seperated difference to easily transform into .csv
        printf("%d : %d : %d , %f \n", tm.tm_hour, tm.tm_min, tm.tm_sec, diff_msec);
        //send the diftime struct back to the server
        isSend = send(sock, &diftime, sizeof(diftime), 0);
        if (isSend < 0)
        {
           printf("send error\n");
        }
        //sleeps for SLEEP_TIME seconds
```

```
        sleep(SLEEP_TIME);
    }
    //close socket
    close(sock);
    return 0;
}
```

Appendix E - Raspberry Pi Message Server

```
#include <netdb.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <signal.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#include <sys/time.h>

#define PORT 9007

/*

When this code runs, other users on the network can connect to this using "telnet"

which enable messages to be collected by the server. In this case the server can be

connected by typing telnet piserver.local (port number above).

   Ints

   - sock is used for the socket

   - client_socket is used for the accepted connection to the client

   Structs

   - server_address is the socket address of the server

   - client_address is the socket address of the client

   - timer is the time struct used to get the current time
```

```
    - ticks is also used to get the current time using seconds since the start of the Epoch
    Chars
    - msg is used to tranfer messages from the client
    - greeting sends the first message to the client asking for an indentifier
    - filename determines the name of the text file to record the messages.
    - ident is used by the child to better identify the device the messages are coming from.
    */
int main()
{
    int sock, client_socket, client_len;
    struct sockaddr_in server_address, client_address;
    struct tm timer;
    time_t ticks;
    unsigned char msg[2048];
    FILE *file;
    char greeting[] = "please enter 6 character identifier\n";
    char filename[] = "msglog.txt";
    char ident[6];
    memset(&msg, "0", sizeof(msg));
    //set up socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
    {
        printf("error creating socket\n");
        exit(1);
    }
```

```
//set up socket address
server_address.sin_family = AF_INET;
server_address.sin_port = htons(PORT);
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
//bind socket
if (bind(sock, (struct sockaddr *)&server_address, sizeof(server_address)) < 0)
{
   printf("error binding socket\n");
   exit(2);
}
//set socket to listen
listen(sock, 5);
printf("server listening...\n");
/*
while loop for the parent process. when it connects to the client it splits into
child and parent, child enters next while loop where messages are sent to the server
and entered into the text file. Parent closes client_socket and waits for another
to connect.
*/
while (1)
{
   //wait for client to connect and then fork
   client_len = sizeof(client_address);
   client_socket = accept(sock, (struct sockaddr *)&client_address, &client_len);
   int child = fork();
   if (child == 0)
```

```
    {
        //only childs in this, request identifier for logs
        close(sock);
        printf("connection to pid%d successful\n", getpid());
        write(client_socket, greeting, sizeof(greeting));
        read(client_socket, ident, sizeof(ident));
        //opens the .txt file and puts the current time and identifier before each message
        while ((count = read(client_socket, msg, sizeof(msg))) > 0)
        {
            ticks = time(NULL);
            timer = *localtime(&ticks);
            file = fopen(filename, "a");
            fprintf(file, "\n%d:%d:%d -%s- %s\n", timer.tm_hour, timer.tm_min, timer.tm_sec, ident,
msg);
            write(client_socket, "recorded\n", sizeof("recorded\n"));
            memset(msg, 0, sizeof(msg));
            fclose(file);
        }
        //dissconnection print
        printf("pid%d disconnected\n", getpid());
        close(client_socket);
    }
    else
    {
        //close parent
        close(client_socket);
```

```
        }
    }
    close(sock);
    return 0;
}
```

Appendix F GPS Logger

```
from gps3 import agps3

import time

#I take no ownership to most of this code. It is part of the gps3 python package

#found here https://github.com/wadda/gps3.git and has been widely used in

#other raspberry pi gps projects


#gets the gpsd socket where the GPS is transmitting data

#gets the data that the GPS is streaming

#connects and watches the socket

gps_socket = agps3.GPSDSocket()

data_stream = agps3.DataStream()

gps_socket.connect()

gps_socket.watch()


#loops through the socket and prints to the terminal the time, lat and lon

#writes to the file gps_log.txt the time and location. Records every 2 sec

for new_data in gps_socket:

    if new_data:

        log_file = open("gps_log.txt","a")

        data_stream.unpack(new_data)

        log_file.write("%s "%  data_stream.time)

        log_file.write("Lat = %s "%  data_stream.lat)

        log_file.write(" Lon = %s \n"%  data_stream.lon)

        print(data_stream.time)

        print('Latitude = ', data_stream.lat)
```

```
print('Longitude = ', data_stream.lon)

log_file.close()

time.sleep(2)
```

Appendix G NAS server setup

To follow these instructions, mdadm and samba need to be installed.

The following instructions on setting up a NAS server with RAID1 is based on the instructions found at: https://gist.github.com/f9636be272f97d414652ce1f21e6b1f4.git (Leandro, 2021)

1. connect the drives to the device, RAID1 uses two storage drives.
2. Unmount the storage drives using the file explorer.
3. partition the drive using: sudo fdisk/dev/sda
    a. type m
    b. type o
    c. type n
    d. type p
    e. type 1
    f. enter
    g. enter
    h. type w
4. format the drive using: sudo mkfs.ext4 /dev/sda1
5. repeat steps 2- 4 for the second drive using sdb and sdb1
6. find the mount points of each drive by using “blkid” in the terminal
7. to create a RAID1 with the two drives enter into the terminal: sudo mdadm -Cv /dev/md0 -l1 -n2 /dev/sd[ab]1
8. save the RAID array by following these commands:
    a. sudo -i
    b. mdadm --detail --scan >> /etc/mdadm/mdadm.conf
    c. less /etc/mdadm/mdadm.conf
    d. exit
9. The file system for the RAID1 array needs to be created by entering: sudo mkfs.ext4 -v -m .1 -b 4096 -E stride=32,stripe-width=64 /dev/md0
10. next the directory where it will be mounted: sudo mkdir [your choice of directory]
11. mount the drive to the chosen directory: sudo mount /dev/md0 [your choice of directory]
12. use “blkid” to find to UUID
13. add the UUID to fstab so that the RAID array is started at boot up. If this step isn’t done properly, the operating system may have to be reinstalled.
    a. sudo nano /etc/fstab
    b. add to new line: UUID=[your UUID] [your directory] ext4 defaults,noatime 0 0

The next steps are how to setup Samba so that the drives can be accessed over the network.

1. edit the samba config file (sudo nano /etc/samba/smb.conf) and enter this text at the bottom of the file:
    a. # NAS Share Block
    b. [NAS]
    c. path = [your directory]
    d. comment = RPI4 RAID1 NAS Server
    e. volume = NAS-Server
    f. valid users = [pi user name]
    g. read only = NO
    h. guest ok = YES
    i. public = YES
    j. writable = YES
    k. browsable = YES
    l. ### -rwxr--r--
    m. create mask = 0744
    n. ### -rwxr-xr-x
    o. directory mask = 0755
    p. ### All hosts on the [your subnet] subnet allowed:
    q. hosts allow = [your subnet]
2. restart samba with: sudo service smbd restart
3. done!

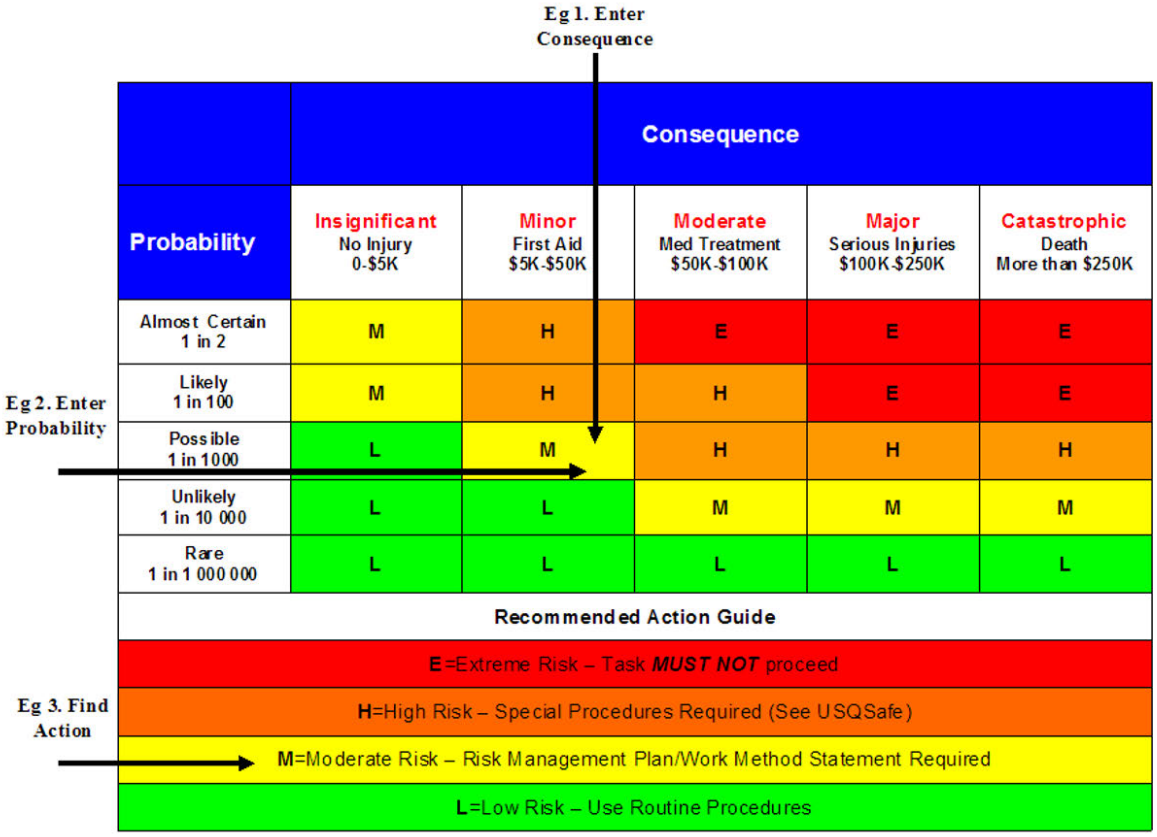
University of Southern Queensland **Offline Version**

USQ Safety Risk Management System

Note: This is the offline version of the Safety Risk Management System (SRMS) Risk Management Plan (RMP) and is only to be used for planning and drafting sessions, and when working in remote areas or on field activities. It must be transferred to the online SRMS at the first opportunity.

Safety Risk Management Plan – Offline Version			
Assessment Title:	Airborne Observation Controller	Assessment Date:	4/04/2022
Workplace (Division/Faculty/Section):		Review Date:(5 Years Max)	25/05/2022
Context			
Description:			
What is the task/event/purchase/project/procedure?	To assemble and code microcontroller that will act as a portable management system for aircraft-borne observations. Assemble and use a battery management system to power the devices that are connected to the microcontroller.		
Why is it being conducted?	Final year dissertation		
Where is it being conducted?	From home		
Course code (if applicable)	ENG4111 and ENG4112	Chemical name (if applicable)	
What other nominal conditions?			
Personnel involved	Robert Schuster		
Equipment	Rasperry pi, RTC and GPS modules, network switch, laptops, LAN cables, batteries, harddrives, computers		
Environment	Home office		
Other			
Briefly explain the procedure/process	Research and code raspberry pi to manage connected devices through a LAN connection. Asses and test battery usage to power devices for a predetermined amount of time.		
Assessment Team - who is conducting the assessment?			

Assessor(s)	John Leis
Others consulted:	



Step 1 (cont)	Step 2	Step 2a	Step 2b	Step 3			Step 4				
Hazards: From step 1 or more if identified	***The Risk:*** What can happen if exposed to the hazard without existing controls in place?	***Consequence:*** What is the harm that can be caused by the hazard without existing controls in place?	***Existing Controls:*** What are the existing controls that are already in place?	***Risk Assessment:*** Consequence x Probability = Risk Level			***Additional controls:*** Enter additional controls if required to reduce the risk level	***Risk assessment with additional controls:***			
				Probability	**Risk Level**	**ALARP? Yes/no**		**Consequence**	**Probability**	**Risk Level**	**ALARP? Yes/no**
Example											
Working in temperatures over 35° C	Heat stress/heat stroke/exhaustion leading to serious personal injury/death	catastrophic	Regular breaks, chilled water available, loose clothing, fatigue management policy.	possible	high	No	temporary shade shelters, essential tasks only, close supervision, buddy system	catastrophic	unlikely	mod	Yes
Extended period of time on the computer	eye strain, possible RSI in the wrist and hands	Minor	Regular breaks, hand and finger stretches	Possible	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
mishandle a battery	fire, explosion, minor to major person injury, possible damage to surrounding devices and room.	Catastrophic	Battery management policy - keep away from sharp objects and sources of heat. Keep at nominal voltages when not in use. Do not overcharge or overdrain batteries.	Unlikely	Moderate	No	Store batteries in a battery safe container or bag. Use a battery management system to charge.	Catastrophic	Rare	Moderate	No
Project deadlines aren't met	Stress, delayed acedemic deliverables, late project, worload bleed into sem 2	Minor	Project plan, updates with supervisor.	Possible	Moderate	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Heat and fumes from soldering	possible burns and smoke inhilation.	Minor	Safety protection when soldering, exhaust fans to remove fumes. Lead-free solder	Unlikely	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
Voltage spikes when assembling	Voltage spikes could damage components and shock someone.	Minor	Protection diodes, Fuses, check polarities before powering.	Unlikely	Low	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No

Step 1 (cont)	Step 2	Step 2a	Step 2b	Step 3			Step 4				
***Hazards*:** From step 1 or more if identified	***The Risk:*** What can happen if exposed to the hazard without existing controls in place?	***Consequence:*** What is the harm that can be caused by the hazard without existing controls in place?	***Existing Controls:*** What are the existing controls that are already in place?	***Risk Assessment:*** Consequence x Probability = Risk Level			***Additional controls:*** Enter additional controls if required to reduce the risk level	***Risk assessment with additional controls:***			
				Probability	**Risk Level**	**ALARP? Yes/no**		**Consequence**	**Probability**	**Risk Level**	**ALARP? Yes/no**
Example											
Working in temperatures over 35° C	Heat stress/heat stroke/exhaustion leading to serious personal injury/death	catastrophic	Regular breaks, chilled water available, loose clothing, fatigue management policy.	possible	high	No	temporary shade shelters, essential tasks only, close supervision, buddy system	catastrophic	unlikely	mod	Yes
No availability of parts	Unable to complete or delayed project	Minor	Order parts early. Order extra parts for redundency	Possible	Moderate	Yes		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
		Select a consequence		Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
				Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
				Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
				Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
				Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No
				Select a probability	Select a Risk Level	Yes or No		Select a consequence	Select a probability	Select a Risk Level	Yes or No

Step 5 - Action Plan (for controls not already in place)			
Additional controls:	*Resources:*	*Persons responsible:*	*Proposed implementation date:*
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.
			Click here to enter a date.

Step 6 - Approval					
Drafter's name:				Draft date:	Click here to enter a date.
Drafter's comments:					
Approver's name:	John Leis		Approver's title/position:	Hons project supervisor	
Approver's comments:	Have discussed with Robert				
I am satisfied that the risks are as low as reasonably practicable and that the resources required will be provided.					
Approver's signature:				Approval date:	1 May 2022