Power Efficient, Telemetry-Enabled Position Sensor for Animal Tracking

Bradley J. Panckhurst

A thesis submitted for the degree of Master of Science
at the University of Otago, Dunedin, New Zealand.

April 9, 2015
Abstract

This thesis describes the design, development and characterisation of an animal position sensor system. It focuses on the implementation of a telemetry system capable of retrieving position data wirelessly.

The position sensor electronics weigh only 10 g and utilise the 868 MHz frequency band for telemetry. Implementing a solar energy harvesting system, the sensors demonstrated the potential for an indefinite deployable lifetime. With over 300,000 location acquisitions, the static position sensors were found to have a Circular Error Probable (CEP) accuracy of <4 m. The telemetry system showed reliable data transmission out to 1 km. However, it is capable of a range of 2.4 km with a clear line of sight.

All aspects of the position sensors were developed to be power efficient. The development included details of the hardware circuit design and the software for both the position sensor devices and the base station. Also included is a customised telemetry protocol centred around power efficient strategies. Characterisation and testing of the position sensor included the evaluation of three key factors. The factors investigated were location accuracy, power consumption and the telemetry transmission performance.

Finally, future revisions to the position sensor system are detailed, with exciting possibilities from the hardware layer up to the telemetry protocol.
Acknowledgements

This thesis would not have been possible without the enormous help and support of many people. I would first and foremost like to thank my supervisor Tim Molteno. He has been extremely supportive throughout this project and under his guidance I have learnt a lot.

Thank you to Phil Brown for his expertise, help and instruction on everything hardware.

Thanks to all other members of the Electronics Research Group, past and present. You have given me the time of my life throughout this thesis. There have been ups and downs, but it has been fun.

Prinny, Mike, Isaac, LN, Leslie and Max: a flat above all others. Distractions aplenty, it has always been entertaining, as well as an adventure.

Thanks also goes to Andy, Hilary, Mike and Leslie for taking the time to proofread my thesis, and a special thanks goes to my ‘chief technical editor’ for her massive proofreading marathon, going above and beyond. A truly impressive feat and I will always appreciate the huge effort.

Finally, I would like to thank my family for years of encouragement and support, in particular Mum and Dad.
# Contents

List of Figures ix

Glossary xi

1 Introduction 1

1.1 Wireless Telemetry Technologies 3

1.1.1 Commonly-used Telemetry Systems 4

1.1.2 Telemetry Protocols 6

1.1.3 Telemetry Frequency 6

1.2 Energy Harvesting 9

1.3 Outline of this Thesis 10

2 Hardware Design 11

2.1 Telemetry System 12

2.1.1 Solar Energy Harvesting System 12

2.1.2 Telemetry System 13

2.2 CPU/GPS Module 15

2.2.1 CPU 15

2.2.2 GPS 16

2.3 Circuit Design 17

2.4 Sensor Mounting 19

3 Embedded Software Architecture 21

3.0.1 State Machine Design 21

3.1 Position Sensor Operation Consideration 25

3.1.1 Shutdown / Storage 25

3.1.2 Transmission Debounce 25

3.1.3 Data Management 25

3.1.4 GPS Acquisition Interval 26

3.2 Telemetry Configuration Options 27

3.2.1 Enhanced Wake-On-Radio Timing 27

3.2.2 Channel Monitoring 28

4 Communication Protocol 31

4.1 Packet Header 32

4.1.1 Sync Word 32
A Additional Position Accuracy Results
A.1 Position Sensor - 3609997124 ........................................... 81
A.2 Position Sensor - 3542264594 ........................................... 83
A.3 Position Sensor - 3542326081 ........................................... 85
A.4 Position Sensor - 3544034068 ........................................... 87

B Publication ........................................................................... 89

C Base Station Shield Designs .................................................. 97
C.1 Base Station Shield Schematic ........................................... 98
C.2 Base Station Shield PCB Layout ....................................... 99

D Position Sensor Telemetry Module Designs ......................... 103
D.1 Telemetry Module - Internal Antenna Schematic ............... 104
D.2 Telemetry Module - Internal Antenna Board Layout .......... 105
D.3 Telemetry Module - External Antenna Schematic ............. 108
D.4 Telemetry Module - External Antenna Board Layout .......... 109

E Position Sensor Software Overview ..................................... 113
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Position acquisition map</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>System and communication flow schematic diagram</td>
<td>2</td>
</tr>
<tr>
<td>1.3</td>
<td>Portable micro-cell GSM base station</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>Photo of the miniaturised ARGOS transmitter</td>
<td>5</td>
</tr>
<tr>
<td>1.5</td>
<td>ITU standards and regulations regions</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Block diagram of position sensor</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Photo of the telemetry board</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>Schematic of the radio frequency chain changes</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Photo of the CPU/GPS board</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Photo of the telemetry board with external and internal antennas</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>Photo of the telemetry board component layer</td>
<td>18</td>
</tr>
<tr>
<td>2.7</td>
<td>Photo of the full position sensor</td>
<td>19</td>
</tr>
<tr>
<td>2.8</td>
<td>Photo of the ear mounted livestock position sensor</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Position sensor’s state diagram</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Transceiver’s simplified state diagram</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>Battery measurement circuit schematic</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Diagram of the eWOR timing events</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Diagram of the eWOR transmit and receive, out of synchronisation</td>
<td>28</td>
</tr>
<tr>
<td>3.6</td>
<td>Diagram of the total receive mode current consumption</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>Schematic diagram of the packet format</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>Photo of the base station</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Flow-diagram of the base station software</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>Base station’s user control client interface</td>
<td>47</td>
</tr>
<tr>
<td>5.4</td>
<td>Photo of base station in its weather-proof enclosure</td>
<td>51</td>
</tr>
<tr>
<td>5.5</td>
<td>Photo of the base station shield mounted on a Raspberry Pi</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Overview of the web front end tag administration page</td>
<td>54</td>
</tr>
<tr>
<td>6.1</td>
<td>Photo of the stationary position accuracy test rig</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Horizontal position errors, position sensor - 3544165172</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Position errors normalised quantile plot, position sensor - 3544165172</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>Position errors autocorrelation, position sensor - 3544165172</td>
<td>58</td>
</tr>
<tr>
<td>6.5</td>
<td>Wild life position sensor antenna impedance matching</td>
<td>59</td>
</tr>
<tr>
<td>6.6</td>
<td>Livestock position sensor antenna impedance matching</td>
<td>60</td>
</tr>
</tbody>
</table>
6.7 Livestock position sensor with matching components antenna impedance matching .................................................. 61
6.8 Photo of a customised cycle helmet position sensor .......................................................... 62
6.9 On-line snapshot of a position sensor’s marker map ........................................................................ 63
6.10 Telemetry module power consumption results during a transmission phase ............................. 64
6.11 Telemetry module power consumption results during an eWOR operation .............................. 65
6.12 Telemetry module's power consumption results (zoomed) during an eWOR operation ............ 66
6.13 Photo of the solar test rig .............................................................................................................. 68
6.14 Battery voltages from the solar test position sensor .................................................................... 69
6.15 Estimated available solar energy .................................................................................................. 70
6.16 Solar test tag’s battery voltages for November, 2014 ................................................................. 71
6.17 Daily solar energy for November 2014 ......................................................................................... 71
6.18 Lithium-Ion battery constant current discharge curve ............................................................... 72
7.1 Photo of a single board livestock ear tag ....................................................................................... 74
A.1 Horizontal position errors, position sensor - 3609997124 ..................................................... 81
A.2 Position errors normalised quantile plot, position sensor - 3609997124 ...................... 82
A.3 Position errors autocorrelation, position sensor - 3609997124 .................................... 82
A.4 Horizontal position errors, position sensor - 3542264594 ................................................ 83
A.5 Position errors normalised quantile plot, position sensor - 3542264594 .................... 83
A.6 Position errors autocorrelation, position sensor - 3542264594 .................................... 84
A.7 Horizontal position errors, position sensor - 3542326081 ................................................ 85
A.8 Position errors normalised quantile plot, position sensor - 3542326081 ..................... 85
A.9 Position errors autocorrelation, position sensor - 35442326081 .................................. 86
A.10 Horizontal position errors, position sensor - 3544034068 .............................................. 87
A.11 Position errors normalised quantile plot, position sensor - 3544034068 ................... 87
A.12 Position errors autocorrelation, position sensor - 3544034068 .................................. 88
C.1 Base station shield board schematic diagram .............................................................................. 98
C.2 Base station shield board PCB layout ......................................................................................... 99
C.3 Base station shield board PCB layout: top layer ......................................................................... 100
C.4 Base station shield board PCB layout: bottom layer ............................................................... 101
D.1 Telemetry module board (internal antenna) schematic diagram ............................................ 104
D.2 Telemetry module board (internal antenna) PCB layout .......................................................... 105
D.3 Telemetry module board (internal antenna) PCB layout: Top layer .................................. 106
D.4 Telemetry module board (internal antenna) PCB layout: Bottom layer ............................ 107
D.5 Telemetry module board (external antenna) schematic diagram ............................................ 108
D.6 Telemetry module board (external antenna) PCB layout ....................................................... 109
D.7 Telemetry module board (external antenna) PCB layout: Top layer .................................. 110
D.8 Telemetry module board (external antenna) PCB layout: Bottom layer ............................ 111
Glossary

ADC  Analogue-to-Digital Converter.
ARGOS  Advanced Research and Global Observation Satellite.
CEP  Circular Error Probable.
CPU  Central Processing Unit.
CRC  Cyclic Redundancy Check.
DOP  Dilution Of Precision.
eWOR  enhanced Wake-On-Radio.
GPIO  General Purpose Input/Output.
GPRS  General Packet Radio Service.
GPS  Global Positioning System.
GSM  Global System for Mobile communications.
HDMI  High-Definition Multimedia Interface.
HDOP  Horizontal Dilution Of Precision.
HTTP  HyperText Transfer Protocol.
i.i.d  independent, identically distributed.
ICP  Integrated Ceramic Passive.
IEEE  Institute of Electrical and Electronics Engineers.
ISM  Industrial, Scientific and Medical.
ITU  International Telecommunication Union.
LDO  Low Drop-Out voltage regulator.
LQI Link Quality Indication.
PCB Printed Circuit Board.
RAM Random-Access Memory.
RF Radio Frequency.
RPC Remote Procedure Call.
RSSI Received Signal Strength Indication.
RTC Real-Time Clock.
SAW Surface Acoustic Wave.
SMA SubMiniature version A.
SMS Short Message Service.
SPI Serial Peripheral Interface.
SRD Short Range Device.
SSH Secure SHell.
USB Universal Serial Bus.
VHF Very High Frequency.
Chapter 1

Introduction

The research presented in this thesis centres around the design, development and characterisation of an animal position sensor system. It combines Global Positioning System (GPS) and Short Range Device (SRD) telemetry technology to provide a smaller, lighter and longer lasting tracking device.

This thesis focuses on the implementation of a solar powered and power efficient telemetry system. As the optimisation of power consumption is required across all protocol layers [1], this requires energy efficiency from the physical layer up to the application layer of the protocol. This was the basis for the telemetry system protocol, which was designed around low-power strategies.

Figure 1.1: A position acquisition map displaying over 4000 data points from a vehicle mounted position sensor. It demonstrates the sensor moving in and out of telemetry range of the base station, while still preserving position data.
Motivation

Satellite tracking of wildlife and livestock is well established \[2, 3, 4, 5, 6, 7\]. Whilst the species tracked are very different, the desired outcome from the studies are often similar; understanding an animal’s behaviour and group dynamics, as well as the impact they have on the environment. This helps to improve management practises, for both the animal and their environment \[3, 4, 8, 9\].

The majority of the location monitoring studies are accomplished with the use of GPS collars for gathering data. For larger animals, collars weighing between approximately 1 kg \[4\] and 2 kg \[6\] \[7\] are normally used. Recent advantages in sensor technology has allowed for smaller platforms to be developed, using GPS as the main platform sensor \[2, 10, 11\].

With this comes the necessity for the retrieval of location data and the ability to present it in a near real time fashion to the respective end user, be this farmer or researcher. The objective of this research is the development of a position sensor with telemetry that is readily deployable on wildlife and livestock. Figure 1.2 outlines the expected data communication flow of the system.

Figure 1.2: A schematic diagram illustrating elements involved in the position sensor system and communication flow. The ear mounted position sensor captures the animal’s location and once in range of a base station transmits the captured information. The base station then uploads this information to a cloud server, which is available for viewing and analysis via a web front end.
1.1 Wireless Telemetry Technologies

The position sensors are lightweight and solar-powered. These sensors are intended for long term use and are indiscriminately distributed throughout a deployment location. For a remote data transfer system to be a viable candidate for a telemetry module; it must be low-cost, low power and provide two way communication across a considerable distance at an appropriate data rate.

In previous research conducted by the Otago University Electronics Group [2], the use of a portable micro-cell Global System for Mobile communications (GSM) base station had a range between 300 and 700 m. The corresponding GSM module transmitted 8 GPS acquisitions with each Short Message Service (SMS) message sent. Each SMS message took approximately 6-7 seconds to transmit, or 1 GPS acquisition per 0.750-0.875 seconds.

Deployed on a hilltop on Great Barrier Island, New Zealand, the portable remote GSM base station weighed 632 kg. The majority of this weight is dedicated to the power supply system (see Figure 1.3). To continuously power the base station, the system implemented: 3x 220 W solar panels, a 400 W wind turbine and a battery bank consisting of 6x 12 V 40 Ah deep cycle batteries.

Figure 1.3: A trailer portable micro-cell GSM base station. It included a power system capable of providing continuous operation using 3x 220 W solar panels, a 400 W wind turbine and a battery bank. It was deployed on a hilltop on Great Barrier Island, New Zealand.
1.1.1 Commonly-used Telemetry Systems

With data retrieval being important in many other applications besides location tracking, there are a number of data retrieval systems already developed. This technology continues to advance with the reduction of size, weight and cost of the electronic components required [12]. Systems already in use are the Very High Frequency (VHF) beacon, GSM and satellite.

VHF “Pinger” Systems

In the VHF “Pinger” system, the VHF transmitter acts as both a directional finding beacon and a radio link for data transfer. They are commonly used in wildlife tracking as a directional beacon because of their simplicity, range of several kilometres and efficient power consumption. However, in this form a directional antenna must be staffed to determine the location of an animal, although data can be pulse-encoded into the VHF signal [13].

A pulse-encoded VHF signal only provides a data transfer in a single direction (from device to the base station). Ground-to-ground the signal can have a range of up to 5 km. A VHF transmitter is suited to real-time, small data-load transmissions, such as reporting a single current GPS location [14].

GSM-Based Systems

Initially conceived in 1982, Global System for Mobile communications (GSM) is a standard that describes protocols for second generation digital cellular networks used by mobile phones. There are two GSM technologies that have been deployed on animals; GSM/SMS and GSM/General Packet Radio Service (GPRS) services. It is possible for both systems to be utilised in a two-way data link, however if it is not correctly implemented it will negatively affect power consumption. An advantage of using GSM technology for data retrieval is that there is a ready-deployed network for the devices to use. However, a disadvantage appears in situations where the coverage zones of the network do not overlap an animal’s roaming zone. Data collection therefore can be intermittent or non-existent [2, 15]. Also of note, is that the coverage of the two GSM services may not be the same [14].

Another disadvantage of the GSM technologies is the high current draw they place on a power supply, with current spikes up to 2.5 A and an average current consumption of 438 mA [16]. They represent typical values in comparison to other GSM modems.

Satellite-Based Systems

Data recovery using satellite services is a popular choice especially where data needs to be accessed from remote locations [17]. There are two types of satellite systems that can be considered for data retrieval purposes.

Advanced Research and Global Observation Satellite (ARGOS) is a worldwide location and data collection system that is dedicated to environmental applications. It was established under an agreement with the French Space Agency - Centre National D’études Spatiales (CNES), National Oceanic and Atmospheric Administration
Figure 1.4: A bird mounted miniaturised ARGOS transmitter [20]. With the advancement in transmitter technology and satellite receiver sensitivity, the transmitters have become lighter (some weighing less than 5 grams) and require less power for transmission (approximately 150 mW).

(NOAA, USA) and National Aeronautics and Space Administration (NASA, USA) in 1978. With the advancement in transmitter technology and satellite receiver sensitivity, the ARGOS transmitters have become lighter (some weighing less than 5 grams) and require less power for transmission (approximately 150 mW). These features, plus worldwide availability, make the Argos tag a viable option for animals with a vast habitat. Figure 1.4 shows a solar powered bird mounted ARGOS tag.

Due to the polar orbit and limited number of the ARGOS satellites, the number of daily data collecting pass overs is around twenty-eight at the poles and eight at the equator, which last for approximately 10 minutes. The lack of satellite visibility can be alleviated with the use of a GPS receiver in combination with ARGOS for the location data retrieval [18].

As the ARGOS telemetry is a one-way data system, the transmitter is unaware of whether or not a receiving satellite is overhead. This means that a data stream must be re-transmitted to ensure reception, with most users allowed to transmit for a second every 90 or 200 seconds [19]. This is a major disadvantage where the density of tagged animals, such as livestock, is high. This increases the probability of an animal’s data not being received.

The other option is the use of commercial satellite-based telephone systems. Two such systems are Iridium and Globalstar [14]. All satellite options have the ability to provide two-way data link capabilities for use with a position sensor. However, they are a more expensive solution [21] and the addition of the two-way data link increases the battery consumption.
1.1.2 Telemetry Protocols

There are many different protocols available from standards to proprietary. One of the great limitations to wireless devices being used “anywhere at anytime” is their limited power supply. Significant power savings can result from the design of the network protocol by incorporating low-power strategies [22].

Below is a comparison of four protocols derived from different Institute of Electrical and Electronics Engineers (IEEE) standards [23]:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>IEEE spec.</th>
<th>Frequency band (GHz)</th>
<th>Max signal rate (Mbit s$^{-1}$)</th>
<th>Nominal range (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bluetooth</td>
<td>802.15.1</td>
<td>2.4</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>UWB</td>
<td>802.15.3a</td>
<td>3.1-10.6</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>ZigBee</td>
<td>802.15.4</td>
<td>868/915MHz; 2.4</td>
<td>0.25</td>
<td>10 - 100</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>802.11a/b/g</td>
<td>2.4; 5</td>
<td>54</td>
<td>100</td>
</tr>
</tbody>
</table>

The four protocols shown here are widely used for many indoor applications and are all operational using a battery power supply. However, the nominal ranges are not optimal for the position sensor’s telemetry system. Selecting a suitable protocol for a particular application is a critical issue. Many factors influence a decision, such as network reliability, roaming capability, recovery mechanism and cost [23].

1.1.3 Telemetry Frequency

In the modern era, there is a high demand for the use of the frequency spectrum. Due to this, a data retrieval telemetry system effectively utilising the available spectrum is fundamental. The choice of frequency and bandwidth impacts significantly on the maximum transmission range and the associated power consumption.

Free Space Propagation

Both wavelength and antenna gain influence telemetry range. The Friis Radiation Formula is an idealised ratio of power received at one antenna, when transmitted from another [24]. It represents transmission of a signal between two points, well removed from any effects of the earth’s surface. Equation 1.1 is the expression of the Friis Radiation Formula. The power available at the input of the receiving antenna, $P_r$ (W), is given by:

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2.$$  \hspace{1cm} (1.1)

Where $P_t$ is the Transmit Power (W), $G_t$ is the Transmit Antenna Gain, $G_r$ is the Receiver Antenna Gain, $R$ is the distance between the two antennas (m) and $\lambda$ is the wavelength (m).

For communication systems it is convenient to express power levels in relation to a 1 mW reference and in decibels. Equation 1.1 can be rewritten, where the power available at the input of the receiving antenna, $P_r$ (dBm), is given by:
\[ P_r = P_t + G_t + G_r - 20 \log_{10} \left( \frac{4\pi R}{\lambda} \right)^2. \]  \hspace{1cm} (1.2)

Where \( P_t \) is the Transmit Power (dBm), \( G_t \) is the Transmit Antenna Gain (dB), \( G_r \) is the Receiver Antenna Gain (dB), \( R \) is the distance between the two antennas (m) and \( \lambda \) is the wavelength (m).

It must be noted that Equation 1.2 assumes ideal conditions incorporating only free space loss. However, given set power levels, Equation 1.2 does provide an indication on the range possible for a frequency.

**Foliage Losses**

The Friis Radiation Formula (Equation 1.2) does not include a multitude of losses, such as antenna polarisation and atmospheric absorption losses, or losses incurred within the Radio Frequency (RF) chain (cables).

For an animal-mounted telemetry system, the most important loss to be incorporated into path loss calculations, is the attenuation of radio waves by trees. The Weissberger’s model is an improved empirical model, developed to mitigate predicted loss errors caused by propagation through a small grove of trees [25]. Below is the expression of Weissberger’s foliage path loss model. The foliage path loss, \( L \) (dB), is given by:

\[
L = \begin{cases} 
1.33f^{0.284}d^{0.588}, & 14 < d \leq 400 \\
0.45f^{0.284}d, & 0 < d \leq 14 
\end{cases}
\]  \hspace{1cm} (1.3)

Where \( f \) is the frequency (GHz) and \( d \) is the foliage depth (m)

The Weissberger model is only defined for foliage depth of up to 400 m. This is suitable for calculating the path loss due to shelter belt and small tree plots, which would be encountered by a livestock tracking sensor.

**Antenna Size**

While the wavelength affects telemetry range, it also affects the size of an antenna. An antenna must be tuned to the same frequency band as the radio system, otherwise transmissions will be impaired. The size of an antenna is relative to the wavelength of the desired frequency, where wavelength, \( \lambda \) (m), is given by:

\[
\lambda = \frac{c}{f}. \]  \hspace{1cm} (1.4)

Where \( c \) is the speed of light (m/s) and \( f \) is the desired frequency (Hz)

One of the most widely used antenna is a wire monopole. A common form is the quarter-wave monopole where the size of the monopole is a quarter of the wavelength. A quarter-wave monopole has a radiation pattern that is at full strength in the horizontal direction. The field strength monotonically declines to zero at the axis.
Regulations

The position sensor must comply with the applicable standards and regulations in the region of use. For the position sensor’s telemetry system, these are radio spectrum policies. They outline what a frequency range can be used for, plus the maximum amount of power allowed to be transmitted within that frequency. As the position sensor is being designed and developed in New Zealand, it must comply with the national legislation - the Radiocommunications Act 1989 and the Radiocommunication Regulations 2001 [26]. This legislation is managed by the Ministry of Business, Innovation and Employment (MBIE) in New Zealand. If the system is to be used internationally, it must comply with the International Radio Regulations. This is incorporated into New Zealand’s national legislation, as New Zealand is a signatory to the International Telecommunication Union (ITU) Telecommunications Convention. Figure 1.5 shows the ITU regions. The standards and regulations of the three international regions, while very similar, are not exactly the same. The system should be capable of being configured to achieve the desired frequency and power, within a specific region’s standards and regulations.

Figure 1.5: Global map showing the three different International Telecommunication Union (ITU) regions. While the international regions are very similar, the standards and regulations are not all exactly the same. Therefore, vigilance must be used when configuring any wireless device.
1.2 Energy Harvesting

Although a power-efficient telemetry system allows for the position sensors to live longer, they still have a limited life. The length of a device’s life is dependent upon the capacity of the energy source. Energy harvesting removes the power source limitation by allowing a device to exploit renewable energy resources within its environment [27].

Harvested energy can be utilised in two different ways: harvest-use and harvest-storage-use [28]. Harvest-use is where the harvesting system directly powers the device. However if the output power is not sufficient, the device will be disabled.

Harvest-storage-use is where the harvesting system uses an intermediary storage component, which allows for harvested energy to be stored for later use. How the harvested energy is best utilised is dependent on the device’s application. A device expected to operate continuously would require a harvest-storage-use method, which ensures a constant power supply.

The typical forms of energies extracted from the ambient environment are: solar energy, mechanical energy, thermal energy and RF energy [29]. Each energy source has different characteristics based around the criteria: controllability, predictability and magnitude [28]. If an energy source is controllable, the availability of the source is predictable. Table 1.1 outlines a few ambient energy harvesting sources available and their corresponding characteristics.

<table>
<thead>
<tr>
<th>Energy Source</th>
<th>Characteristics</th>
<th>Harvesting Technology</th>
<th>Amount of Energy Harvested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solar</td>
<td>Uncontrollable, Predictable</td>
<td>Solar Cells</td>
<td>15 mW/cm²</td>
</tr>
<tr>
<td>Wind</td>
<td>Uncontrollable, Predictable</td>
<td>Anemometer</td>
<td>1200 mWh/d</td>
</tr>
<tr>
<td>Indoor Vibrations</td>
<td>Uncontrollable, Unpredictable</td>
<td>Electromagnetic Induction</td>
<td>0.2 mW/cm²</td>
</tr>
</tbody>
</table>

Table 1.1

The amount of energy exploited by energy harvesting methods is relative to the size of the harvesting device. When choosing a suitable harvesting method, an important consideration is the density of exploitable energy relative to the design of the device, such as size and weight.

While research into wind energy harvesting is making it a viable option in the future, the current size and weight of the harvesting technology removes it as an energy harvesting method for animal tagging. However, both mechanical (not included in the ambient energy table above) and solar energy harvesting have been implemented in a wide range of animal tagging systems, with solar energy harvesting being implemented on bird tags as early as the late 1970s [30].
1.3 Outline of this Thesis

The remainder of this thesis is organised as follows:

Chapter 2 presents an outline of the position sensor’s hardware technical and design decisions. This includes the individual component selection and the layout of the PCB. This chapter concludes with a brief discussion on the practical mounting of the position sensors.

Chapter 3 gives an outline of the software technical and design decisions for telemetry integration. This discussion includes the general software design in the form of the position sensor’s state diagram. Finally, it covers special procedures and configuration considerations that allow for the correct operation of the position sensors.

Chapter 4 outlines the telemetry communication protocol. This includes the general packet format and a description of individual request and response packets. Also discussed is the interaction with the position sensor’s native hardware packet handling.

Chapter 5 addresses the base station development from hardware through to end-user interface. It begins with a general description of the software design. Then the hardware of the base station is explained. Finally, it briefly covers the user-interface developed for displaying the collected data.

Chapter 6 discusses the results of the position sensors. The key factors investigated include the location accuracy, power consumption and the telemetry transmission performance. Testing involved measuring power consumption and antenna matching of the telemetry module. With long-term experiments conducted, the position sensor system was tested as a whole to determine any performance bias in the devices.

Finally, Chapter 7 provides a summary of the work conducted in this thesis and discusses how the system could be improved with further revisions of the position sensors and base station.
Chapter 2

Hardware Design

This chapter outlines the telemetry and energy harvesting hardware decisions for the animal tracking devices. The telemetry module’s hardware is explained from the component selection, to the layout of the Printed Circuit Board (PCB).

The telemetry module’s hardware was developed around a pre-existing modular position sensor. It was introduced as a lightweight, low-power, flexible wireless solution as an alternative to Global System for Mobile communications (GSM). The telemetry module combines telemetry and solar energy harvesting systems onto a single PCB. This section will outline the components of the existing Central Processing Unit (CPU) / Global Positioning System (GPS) module and the telemetry module. It will also cover the telemetry module’s circuit design. Figure 2.1 shows the block diagram for the position sensor, developed as part of this thesis.

![Block diagram of position sensor modules and interfaces. The red dashed outline indicates the telemetry module hardware which includes the Short Range Device (SRD) module and solar charging module. The telemetry module maintains a separate battery connection to the CPU/GPS module, with the solar charging module controlling the SRD module’s regulator.](image-url)
2.1 Telemetry System

The telemetry board was designed and developed to provide a configurable wireless data link with the position sensors and meanwhile be as power efficient as possible. The board incorporates: Telemetry utilising the Texas Instruments CC1120 transceiver chip [31], Solar utilising the Texas Instruments BQ25504 nano-power management chip [32].

![Telemetry Board](image1)

Figure 2.2: Telemetry Board. Figure 2.2a shows the top side and Figure 2.2b shows the bottom side of the Telemetry board. It is a two layered board with the bottom side containing all of the components apart from the solar cells, which are mounted on the top side. The circuitry combines both the solar (Texas Instruments BQ25504) and telemetry (Texas Instruments CC1120) components, which are split respectively in the layout of the module. The New Zealand ten cent coin provides a size comparison (bottom side only). The dimensions of the module are 44 mm long by 19 mm wide and weighs 1.42 g.

2.1.1 Solar Energy Harvesting System

The solar section of the telemetry board allows for an extended deployment of the tags. This is achieved by removing the limiting constraint of the battery run time. Depending on the application of the tags, small batteries can be used reducing weight. The first major component in this section is the Texas Instruments BQ25504 that is an ultra low power, highly efficient boost converter/charger [32].
The Texas Instruments BQ25504 implements a battery-good flag designed to signal an attached microprocessor, when the voltage on the battery has depleted below a set voltage [32]. Chapter 3 outlines the power management software solution the tag implements. It is conceivable that the battery may be excessively drained during a long microprocessor sleep period. The solar telemetry board re-purposes the battery-good flag, as a final hardware battery management telemetry cutoff solution. The battery good flag is connected directly to the telemetry section’s Low Drop-out voltage regulator (LDO) enable pin.

The second major component of the solar chain is the solar panels. These are mounted on the top side of the telemetry board shown in Figure 2.2a. In standard formation the telemetry board consists of two panels. Concessions have been made to allow for the operation of a single panel to be implemented, with the addition of a jumper. The panels chosen are the IXYS KXOB22-12X1. They are made of monocrystalline with a cell efficiency of typically 22% measured at a wafer level [33].

Each panel weighs 0.5 grams with dimensions of 22x7x1.8 mm, enabling two to easily fit side by side on the telemetry board. They are arranged in a series configuration to provide the Texas Instruments BQ25504 energy harvester with a typical voltage of one volt, at maximum power point.

Another reason for the selection of the KXOB22-12X1 solar panels is the very good response over a wide wavelength range and the ability to extend run time even in low light conditions. This complements the Texas Instruments BQ25504 capability to begin operating on only microwatts of power.

2.1.2 Telemetry System

The main component in the telemetry section is the Texas Instruments CC1120 transceiver chip. It was chosen, as it is a cost effective fully integrated single-chip radio transceiver, which has low power consumption while maintaining high performance. The chip operates in the Industrial, Scientific and Medical (ISM) and SRD radio bands with available frequency ranges of 164–192 MHz, 274–320 MHz, 410–480 MHz, and 820–960 MHz.

The 820-960 MHz band was chosen based on a compromise between the theoretical range (both line of sight and through foliage) and the antenna length required for use with this frequency range. New Zealand Radiocommunication Regulation General User License for Short Range Devices states, “A general user licence (GUL) provides for certain classes of radio transmitters to be used without the need for the owner to obtain a licence in their own name” [34]. The intersection between the capabilities of the transceiver and the General User License spectrum is a frequency band from 864 - 868 MHz that satisfies the requirements of the tags.

The frequency range of 820-960 MHz and the configurable software allows for the tags to be deployed in similar global ISM and SRD radio bands.
The other significant component in the telemetry pathway is the Integrated Ceramic Passive (ICP) component. This is a match balun filter that simplifies the Radio Frequency (RF) chain by removing thirteen passive components in the original schematic from Texas Instruments. This not only reduces the total component but also frees up space on the board. Figure 2.3 shows the components that are removed.

Figure 2.3: The blue line outlines all the passive components replaced by using the matched ICP component [35]. This simplifies the telemetry module’s radio frequency chain by removing 13 passive components and replacing it with a single balun filter.
2.2 CPU/GPS Module

The CPU/GPS module is the main control module of the position sensor. The modular design allows for rapid prototyping of the overall position sensors, as well as future expansion. Figure 2.4 shows the latest iteration of the CPU/GPS module around which this thesis was developed. It incorporates a CPU utilising an STMicroelectronics STM32F103 microprocessor [36] and a GPS unit, a Navman Jupiter F2 [37].

Figure 2.4: The top side of the completed CPU/GPS module. It is a four layer board with one side containing all of the components. The circuitry combines the CPU (STM32F103 microprocessor) and GPS (Navman Jupiter F2 GPS module) components. The length of the module is 44 mm by 19 mm wide and weighs 2.56 g.

2.2.1 CPU

The STM32F103 microprocessor is a capable 32 bit microprocessor with 128kB of Flash memory and 20kB of Random-Access Memory (RAM). It has the ability to perform up to 72 million operations per second [36]. It is a lot more powerful than required for the position sensor, however, this allows for future development of smarter processing algorithms and extra sensors to be implemented.

The microprocessor accepts a relatively wide application voltage supply, between 2.0 - 3.6 V and possesses three low power modes. The selection of the low power mode is dependent on which processor resources are required for correct operation of the position sensor. Further more, a direct power supply to the on-board Real Time Clock (RTC) and backup registers allows for these resources to be utilised fully in the lowest power mode.
2.2.2 GPS

The Navman Jupiter F2 GPS module was selected based on size, weight and technical specifications, along with the availability of comprehensive documentation. The Navman Jupiter F2 GPS module is implemented using the SiRFstarIV chipset, providing quicker and more accurate tracking. It is a low power module consuming only 14 µA in hibernation mode, while 13 mA when receiving [37].

The Navman Jupiter F2 is designed to operate under low signal level conditions, with tracking down to -163 dBm. To achieve this selectivity, the module utilises a band pass Surface Acoustic Wave (SAW) filter centred on GPS L1 signal - 1.575 GHz and a low noise amplifier. This provides enough gain for the receiver to be used in conjunction with a passive antenna.

The GPS module supports three different time-to-first-fix methods: hot, warm and cold start. Due to the hot start requiring a continuous current draw of between 50 - 500 µA for operation with the position sensors, this is considered to be too high.
2.3 Circuit Design

The telemetry module combines telemetry and solar energy harvesting systems on to a single PCB. Early circuit designs and layout were performed in Eagle version 6 free edition software. Later revisions of the telemetry module’s circuit designs and layout were migrated to KiCad, an open source software suite for Electronic Design Automation [38] (see Appendix D).

The modular circuit board stack was adopted from previous development on the position sensor. This significantly simplified the development, debugging and construction. As part of the modular stack an inter-board bus connector provides an interface between the boards. The telemetry module repurposed the connections that were employed by the GSM module. This allowed the modules to be interchangeable via a software configuration change. With the telemetry module implementing solar energy harvesting, a single inter-board bus connector is used on the bottom layer of the module.

The first production telemetry module was designed with an external antenna. The dimensions matched the form factor of the CPU / GPS board with which it is designed to interface: 18 mm by 44 mm, with 3 mm radius corners. This is shown in Figure 2.4. The transceiver manufacturer, Texas Instruments, provides a miniature PCB helical antenna reference design [39]. This, combined with the compact circuitry layout allowed for an internal antenna by only increasing the telemetry module board width to 20 mm. Matching components are required because the impedance is far from 50 ohms. The reference design PCB board thickness is 0.8 mm that is the same as the telemetry module board. This has an optimal antenna match with a series 1.0 pF capacitor and a shunt inductor of 12 nH for operation at 868 MHz [39]. The telemetry module circuit board is a 0.8 mm thick, 2 layer board and weighs 1.42 g (external antenna) - 1.47 g (internal antenna). Both bare telemetry module boards are shown in Figure 2.5.

Figure 2.5: Shown here are the two types of telemetry modules developed. On the left is the internal antenna board. On the right is the external antenna variant of the telemetry module. The internal antenna board contains extra matching components (pi filter) and is 2 mm wider than the external antenna board.
The solar cells are located on the top of the telemetry module board. The remaining components are located on the bottom of the board. These components are separated into two discrete sections: solar charging and radio (see Figure 2.6).

The respective schematics provided from the manufacturer were followed with the appropriate component values substituted. During testing, the telemetry module was observed to have current ‘leaks’ of approximately 300uA. While debugging this issue it was discovered that the microprocessor lost its General Purpose Input/Output (GPIO) pin state once it had entered its STANDBY sleep mode. This created a voltage difference between the transceiver and the microprocessor. This generated the leakage currents. To correct this issue, suitable pull up and pull down resistors were incorporated into the design on the telemetry module board.

The circuit boards were designed with the largest tracks and vias possible. They had to pass a 0.108 mm clearance design rules check with track widths 0.2 mm or larger. This clearance tolerance was set by the precision of the PCB manufacturer. Blind or buried vias were not allowed and drill sizes were 0.3 mm or greater. Both boards provided a 50 ohm impedance radio frequency trace, required by the ICP component for the antenna connector. To achieve this trace impedance, a coplanar waveguide with ground plane was utilised. The transceiver proved insensitive to the minimal noise created by the solar charging circuitry. This allowed for a common ground plane between the two sections to be used. The ground plane is heavily stitched with a number of 0.3 mm vias, lowering the ground connection impedance.

Figure 2.6: The component side of the telemetry module. The red line encloses the radio components and the light blue line the solar components. The two sections share a common ground. The only other connections shared, are the power line and the battery-good line that control the radio section’s LDO (regulated power to the radio section). This is operated by the solar controller in reference to the battery level.
2.4 Sensor Mounting

This project further developed the position sensor’s animal tagging capability. This included the ability to monitor livestock as well as wildlife. For livestock monitoring the packaging of the position sensor was adapted to fit the common livestock ear tag. Retaining the modular stack design, the battery had extended leads added that allowed it to be located next to the position sensor. This is shown in Figure 2.8 and permits the packaging to be slimmer in profile. This will conceivably lower excessive damage caused by the animals. Due to the constraints of the packaging dimensions of the ear tag, the wire antennas had to be shaped to fit the available space.

![Figure 2.7: The above figure shows a complete position sensor with whip antennas in its avian wildlife tag form. The bottom figure shows a close up of the same position sensor. The position sensor is a three high modular stack. On the base of the stack is a 120 mAh battery, in the middle the GPS/CPU module and on the top is the new solar telemetry module. The total electronic stack including battery weighs 10g. Passive whip antennas are used for both telemetry and GPS. After previous testing they are made from fine, nylon-coated, braided stainless steel cable, designed for use in shark fishing traces [2].]
Figure 2.8: This figure shows an ear mounted livestock position sensor. This is an adaption on the layout of the modular wildlife tags. The battery leads have been extended allowing for the battery to be repositioned giving an overall lower profile. It is housed in a 3D printed case, designed for correct and consistent layout of the major components and antennas. This is fastened directly to a standard livestock ear tag adding an additional 25 g to the total weight of the tag.
Chapter 3

Embedded Software Architecture

This chapter outlines the device software for the animal tracking system. It provides a description of the general software design in the form of the position sensor’s state diagram. This includes special procedures and configuration considerations.

The software has been developed on an existing code platform designed for operation with the Global Positioning System (GPS)/Global System for Mobile communications (GSM) tag [2]. The initial software design concept was to allow interchangeable GSM and telemetry modules, via a simple configuration selection that would occur during programming of the Central Processing Unit (CPU). This concept was abandoned in the final phases of the software development, as different functional abilities of the GSM and telemetry modules determined that separate software development pathways were preferable. An overview of the source code, written in C++, is summarised in Appendix E.

3.0.1 State Machine Design

The software is written, around a state machine (see Figure 3.1). The transitions between states of the state machine and interaction with the transceiver’s internal state machine (see Figure 3.2) are described below:

- SHELL_WAIT: With limited hardware control resources and the complexity of handling multiple modules, the SHELL_WAIT state determines all major next state decisions. The validity of the GPS acquisition is the most important task. A GPS acquisition Horizontal Dilution of Precision (HDOP) value, not within a set range will cause a change in state to HDOP_WAIT state. This possible loop is restricted by a maximum of five attempts, after which it is considered a failure. A GPS acquisition success and failure will result in a move to the FIX_WAIT state.
The SHELL_WAIT state is also responsible for the final handling of a GPS Acquisition Request that is received via the RADIO_INPUT state. This does not change the internal state. However, it interacts with the telemetry state machine shown in Figure 3.2 to transmit the appropriate packet. The only other transition is via a correct UART interrupt, which triggers a move to SHELL_INPUT.

- **SHELL_INPUT**: Available within the first 30 seconds from power on or reset. This state supports changing of initial configuration, displaying stored data (fix and log) and testing different functions by a user via a Desktop Computer. Transitioning from SHELL_INPUT state to FIX_WAIT state requires a user to issue a quit command in the control terminal.

- **FIX_WAIT**: This state utilizes the microprocessor’s lowest power STANDBY mode, consuming a maximum of 5 \( \mu \text{A} \) [40] and only accepts certain interrupts. STANDBY mode setup requires storing the expected position acquisition time into a back up register. The microprocessor does not provide an indication flag to determine which interrupt was activated. Comparison between Real Time Clock (RTC) time and the stored expected acquisition time provides a workaround. An RTC timer interrupt will cause a change in state to NAV_WAIT, while a telemetry interrupt will change to RADIO_INPUT state. The setup also includes placing the telemetry board into either SLEEP state or the enhanced Wake-On-Radio (eWOR) mode. This is a state transition in the telemetry state machine, shown in Figure 3.2.

- **RADIO_INPUT**: The RADIO_INPUT state is only available via the FIX_WAIT state. Depending on the request received, the communication operations carried out in this state are outlined in the Communication Protocol Chapter 4. Most communication requests are resolved within this state, then a state change will be issued to the FIX_WAIT state. A GPS acquisition request will cause a transition to the NAV_WAIT state for an acquisition attempt.

- **NAV_WAIT**: This state disables all external interface interrupts. The telemetry radio is placed in a low power state, where it is neither transmitting nor receiving. A GPS setup is attempted. If it fails, the state is exited to SHELL_WAIT state. If successful, the microprocessor is placed into STOP mode where all peripherals continue to operate [40]. A NAV interrupt from the GPS module, signifies either a GPS acquisition has been achieved or a fix timeout alarm has been triggered. This indicates that the maximum allotted acquisition time has been exceeded. Both situations will trigger a transition into SHELL_WAIT state.

- **HDOP_WAIT**: The microprocessor is set to STOP mode. A pre-established timeout allows the GPS module an increased time window to improve the HDOP parameter. This is an accuracy estimate of the GPS acquisition Latitude and Longitude. The HDOP alarm will trigger a transition back to SHELL_WAIT for another GPS acquisition attempt.
Figure 3.1: This figure shows the microprocessor’s state diagram for the position sensor. From START the sensor drops into a SHELL_WAIT state, which is the main controller state. The HDOP_WAIT and NAV_WAIT states control GPS operations. SHELL_INPUT provides an initial configuration state for a user. The sensor spends most of its time in FIX_WAIT state. This is when the sensor is in its lowest power orientation, waiting for the next GPS acquisition timeout. The RADIO_INPUT state manages all telemetry communications, which is only able to be accessed via the FIX_WAIT state.
Figure 3.2: This is the transceiver’s simplified state diagram for the telemetry module [41]. The microprocessor interacts with the state diagram through commands issued over the Serial Peripheral Interface (SPI) interface. The SLEEP state is critical to the battery consumption performance of the position sensor.
3.1 Position Sensor Operation Consideration

Due to the nature of both the hardware and software considerations, there are a number of specific protocols for the correct operation of the tags.

3.1.1 Shutdown / Storage

The correct shutdown and storage of a sensor is complicated. The hardware which disconnects and shuts down the CPU / GPS module is not available for use with the transceiver. This is due to the continuous operation of the transceiver when the sensor is active. To circumvent this issue, the transceiver is placed into its lowest power mode. This increases the total shutdown current consumption by a maximum 1 µA [31]. Triggering a power reset allows the tag to complete an initial start up procedure into FIX_WAIT state. This resets the RTC, which places the transceiver into the required low power mode. The magnetic switch is reactivated leaving the transceiver in a non-transmitting, low power mode and the CPU / GPS board deactivated.

3.1.2 Transmission Debounce

Transmission debounce is when the position sensors ignore/reject incoming packets. The position sensors constantly receive broadcast packets from the base station. The sensor’s microprocessor processes each packet, a very power intensive procedure. The original concept, used a list appended to the constant status request ping from the base stations. This list instructs the position sensors to ignore the request and return to a low power mode. This still required the sensor to process the packet each time.

To lower the power consumption of this intense operation, the sensor utilises two available broadcasting channels and address filtering [41]. This transfers the decision making process from the microprocessor to the transceiver. One channel is used as an optional data response channel. The other is used as a compulsory response channel where the list is utilised.

3.1.3 Data Management

As part of the position sensor’s data management, fix data is stored internally in Flash memory before it is transmitted to a base station. Discovered early in testing, fix data was erased automatically by tags. This occurred because their queue became full. Erasing of the microprocessor’s Flash is completed either via bulk or by individual pages [42]. To maintain data integrity, the whole queue is erased as it becomes full.

To avoid loss of data, two schemes were introduced. The first scheme inserted an option whether or not to erase the queue, when it is full. This requires a decision during programming, if the start or finish of a tracking path is of greater importance. Selecting not to erase will mean that the start of the path will be saved until it is transmitted to a base station. Selecting to erase will provide the finish of a path. This is vulnerable to loss of data, with the potential for the queue to be emptied prior to establishing contact with a base station.
The second scheme provides a configuration queue reset length variable. This may be adjusted from the default setting, via the Change Configuration Request packet outline in Section 4.3.6. The fix queue is erased once it has exceeded the value of this variable, so long as all stored fixes have been transmitted. This circumvents the potential loss of data when the queue fills to capacity.

3.1.4 GPS Acquisition Interval

The GPS acquisition interval is implemented through a basic algorithm within the sensor’s software. Measurement of the battery voltage is compared to configurable battery thresholds. This multiplies the initial interval by the appropriate amount, which is by powers of two for each threshold up to thirty two. After this the interval is multiplied by thirty two as well as twice the number of aborted GPS fixes, due to a low battery voltage. The simplicity of this algorithm makes it vulnerable to the battery measurements modifying the interval unnecessarily.

Unexpected short-term fluctuations between neighbouring battery voltages are shown in Figure 6.14 in Section 6.3. The battery data during night time hours show the variability in the battery measurement chain.

Figure 3.3 shows the schematic of the battery measurement circuit implemented on the CPU / GPS module. Connected via the BATT_SENSE line, an Analogue-to-Digital Converter (ADC) on this pin measures and calculates a battery voltage (VPP) in millivolts.

![Battery Measurement Circuit](image)
This measurement circuit contains two features, which cause minor inaccuracies in measuring the battery voltage. Firstly, when a voltage reading is not required, the measurement circuit incorporates a transistor (T1) allowing it to be deactivated, preserving battery use. An error is introduced, as the measured voltage equals the true battery voltage (VPP) minus the saturation voltage of the transistor. The manufacturer’s data sheet [43] states that the saturation of the transistor at the current used in the measurement chain will vary the calculated battery voltage by 0.2 mV/°C.

Secondly, the voltage measurement is subject to sampling noise, created by inaccuracies of the ADC reading the measurement chain (BATT\_SENSE). The STM32 ADC accuracy [40] shows a total unadjusted error of typical ±2 with a maximum of ±5 LSB. The LSB is 659.18 μV for the ADC setup on the tag. This means the calculated battery will typically be within ±1.77 mV, with a maximum error of ±4.43 mV.

The majority of the fluctuations between neighbouring voltage measurements, is produced by the intermittent operation of the sensor during GPS acquisitions. This leads to the occurrence of the “recovery effect” where the battery can recover during idle periods [44]. This will affect the measured battery level (VPP), giving the appearance of an increasing voltage.

Fluctuations experienced in the battery measurement chain only become a factor on the boundary cases, in relation to the configurable battery thresholds. These boundary cases are where the GPS acquisition interval may needlessly increase, reducing the number of fixes.

3.2 Telemetry Configuration Options

The standard telemetry radio configuration is a preset file included in the compiled code. This file is created with the assistance of the Texas Instruments SmartRF Studio software [45]. This application is useful for generating the configuration register values. The configuration values represent the considerations to meet the radio frequency regulations and maximise range performance, while minimising the telemetry module’s battery consumption.

Also included in the design for the tag’s radio configuration, is select configurable configuration values. The registers able to be reconfigured are the radio frequency, radio address and the eWOR sleep time. These are stored in Flash by the tag and are able to be changed via telemetry transmission. This wireless change is outlined in the Change Configuration Request Section 4.3.6. These configuration values take priority over the preset configuration file.

3.2.1 Enhanced Wake-On-Radio Timing

The enhanced Wake-On-Radio (eWOR) is a key feature in supplementing the low power nature of the sensor. Figure 3.4 outlines the eWOR timing events. The idle and receive components of the timing diagram dominate the majority of the power consumption. The idle component length is preset in register setup and the length of the receive mode timeout is discussed in the Channel Monitoring 3.2.2.
The radio chip typically consumes 0.5 µA in sleep mode with the resistor-capacitor oscillator running. Maximising the sleep component of the eWOR will decrease the overall power consumption of the transceiver. The sleep time is able to be adjusted wirelessly via the communication protocol outlined in Chapter 4. The sleep time is limited to a maximum interval of approximately 65 seconds.

Transmissions between a base station and position sensor will ideally be synchronous. Nonetheless, the receive and transmit mode transmissions can shift out of synchronisation, outlined in Figure 3.5. A solution is to have the radio performing the receive mode function, adjust its timing in order to resynchronise the communication. The receive mode function occurs on the sensor requiring microprocessor intervention, which increases power usage. To avoid this, the base stations poll at a faster offset rate, increasing the probability of periodic resynchronisation. The trade off for increasing the sleep time (decreasing power consumption) on the sensors, is miscommunication. This is because the probability of a base station synchronising and waking a sensor decreases.

\[\text{Figure 3.5: eWOR transmit and receive, out of synchronisation [46]. The receive mode function can adjust its timing in order to resynchronise the communication. This requires microprocessor intervention, which increases power usage.}\]

### 3.2.2 Channel Monitoring

Channel Monitoring is achieved using the receive sniff mode, a low power mode, where the transceiver searches for radio frequency activity. This is implemented by operating the eWOR in combination with receive mode termination, an event which stops the transceiver receiving [41]. The transceiver is equipped with three different forms of receive mode termination: receive termination timer, carrier sense threshold and preamble quality threshold. Figure 3.6 illustrates the three cases.
The case A and B are receive mode terminations, based upon carrier sense and preamble quality sense respectively. Case C is based on the carrier sense, but with a timeout.

The preamble quality threshold is immune to unwanted signals on the same frequency, therefore more power efficient in noisy environments.

In low radio frequency activity environments, the position sensor’s employ termination based on carrier sense. The carrier sense threshold requires less up-time in receive mode without the presence of noise. This makes it the most power efficient receive mode termination. Setting the optimal threshold is a trade-off between sensitivity and current consumption [46].

Figure 3.6: Total Receive Mode Current Consumption in Presence of Noise [46]. The case A and B are receive mode terminations, based upon carrier sense and preamble quality sense respectively. Case C is based on the carrier sense, but with a timeout. The selection of the receive mode termination is important as it directly affects the current consumption of the transceiver.
Chapter 4

Communication Protocol

This chapter outlines the telemetry communication protocol conducted between the base station and position sensors. Designed as an extension of the native packet handling of the Texas Instruments transceiver, it outlines the packet format which was created. This covers the packet header and footer. It also outlines the individual request and response packets.

For ease of control, base stations initiate and control all communication between themselves and surrounding tags. It is achieved through the use of request and response packets, transmitting specific packets dependent on a base station’s current use case. They all follow the same packet format outlined by Figure 4.1 (the most significant bit of each field is transmitted first) and the implemented packet types are summarised in Table 4.1.

Figure 4.1: This figure depicts the schematic diagram of a packet format. The header and footer fields are handled by the telemetry module’s transceiver, simplifying the packet handling.
Table 4.1: Summary of implement packet types as part of the communication protocol.

<table>
<thead>
<tr>
<th>Requests</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Status</td>
</tr>
<tr>
<td>Fix</td>
<td>Fix</td>
</tr>
<tr>
<td>Instant GPS</td>
<td>No Fix</td>
</tr>
<tr>
<td>Log</td>
<td>Fixes Received</td>
</tr>
<tr>
<td>Configuration</td>
<td>Log</td>
</tr>
<tr>
<td>Change Configuration</td>
<td>End Log</td>
</tr>
</tbody>
</table>

4.1 Packet Header

The header fields of the packet format are inserted on creation of the packet. The Sync Word is used to distinguish an incoming packet by the recipient. The length field, address field and packet type assist in the filtering process once received.

4.1.1 Sync Word

The radio will continuously calculate a Sync Word qualifier value, to distinguish the Sync Word from background noise. In transmit mode, these bits are automatically inserted at the start of the packet by the modulator. In receive mode the demodulator uses the Sync Word to find the start of the incoming packet. [41]

4.1.2 Length Field

Position sensors and base stations are configured to allow packets of variable length. Therefore, the packets must have the Length Field implemented as the first byte, after the Sync Word. The length is defined as the length of the data fields including the address field and packet type. If the received Length Field has a larger value than this, the packet is discarded and the radio controller will either restart receive mode or go to IDLE. [41]

4.1.3 Address Field

The Address Field is set up at the creation of a packet being transmitted. Each tag has an address as part of its default configuration that can be changed dependent on use. Base stations are all assigned the Address Field 0xFE. There are also two broadcast addresses 0x00 and 0xFF, which are utilised in the communications protocol. If the address match fails, the packet is discarded and the radio controller will either restart receive mode or go to IDLE. [41]
4.1.4 Packet Type

A single byte communication flag interpreted in software allows for a range of options to be sent and received, such as the retrieval of the error log data from the tags. The Packet Types are split into requests and responses.

4.2 Packet Footer

The footer fields of the packet format are appended to the packet, once it is received by the intended recipient. The combination of RSSI and LQI, provides information about the quality of the connection between the base stations and position sensors. This is important because it is more useful for a strong connection from a position sensor transmitting to a base station. This will provide a higher probability of receiving all data packets.

4.2.1 Received Signal Strength Indication

As part of the radio, the automatic gain control module returns an estimate on the signal strength received at the antenna called Received Signal Strength Indication (RSSI) [41].

4.2.2 Link Quality Indication

The Link Quality Indicator (LQI) gives an estimate of how easily a received signal can be demodulated. LQI is best used as a relative measurement of the link quality (a low value indicates a better link than what a high value does), since the value is dependent on the modulation format. [41]
4.3 Request Types

Apart from Status and Change Configuration Requests packets, request packets all follow a similar data fields format comprising of the packet header followed by a mandatory tag ID field. This allows the base station to restrict its communication to individual tags and retrieving specific response data from intended tags.

4.3.1 GPS Fixes Request

- Packet Type = 0x02

When a base station transmits a Global Positioning System (GPS) Fixes request, it expects to receive a multiple of Fix Responses or a No Fix response from a tag with the appropriate tag ID.

4.3.2 Log Request

- Packet Type = 0x03

When a base station transmits a Log request, it expects to receive a multiple of Log Responses followed by an End of Log Response from a tag with the appropriate tag ID.

4.3.3 Configuration Request

- Packet Type = 0x04

When a base station transmits a Configuration request, it expects a Configuration Response from a tag with the appropriate tag ID.
4.3.4 Current GPS Location Request

- Packet Type = 0x06

When a base station transmits a Current GPS Location request, it expects a GPS Response or a No Fix Response from a tag with the appropriate tag ID. This request has a long time out period as it must wait for the tag to attempt a GPS capture.

4.3.5 Status Request

- Packet Type = 0x01

The Status Request has an optional tag ID data array. It informs the respective tags that the base station is not communicating with them and therefore, they should ignore the packet and continue normal operation. The Ignore tag array is utilised, when a base station is transmitting either on the 0x00 broadcast address or a selected address. This avoids excess battery consumption by continually waking a tag up.

4.3.6 Change Configuration Request

Combining the reconfiguration ability with appropriate database management, the tags develop into a completely reusable system. This functionality permits tags to be remotely customised and optimised to particular applications after initial programming.

- Packet Type = 0x05

Change configuration requests have a mandatory tag ID field, however it also contains extra data referring to the tags’ new configuration setup. There are three different configuration packets, Radio configuration, GPS configuration, Battery configuration. For simplicity reasons, the configuration data trailing a configuration command is compulsory, meaning that redundant data is also transmitted. The redundancy is offset by less complexity in processing the packet once received by the tag.
Radio Configuration

<table>
<thead>
<tr>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tad ID</td>
</tr>
<tr>
<td>Configuration Command</td>
</tr>
<tr>
<td>Radio Frequency P2</td>
</tr>
<tr>
<td>... Reset Threshold</td>
</tr>
</tbody>
</table>

- Configuration Command = 0x01

  - Tag Address:
    * While currently not fully utilised, this 8 bit address field will in future developments allow for isolation of tags.
  
  - Wake on Radio Sleep time
    * A 16 bit value, giving a sleep period of approximately 65 seconds. This field is valuable for balancing the communication availability of the tags against battery usage.

  - Radio Frequency (Separated into three bytes)
    * Due to the complexity of calculating the required three register values for a given frequency on the tags, the calculation is done on the base station side, then transmitted as three 8 bit values.

  - Fix Queue Reset Length
    * The tags storage fix queue when full, will continue by erasing all previous data in Flash. This field provides flexibility between battery usage for resetting the queue and data lost, due to fix queue overflow.

  - Queue Unsent Threshold
    * With the normal base station operation mode constantly trying to wake surrounding tags, this field contains a threshold time. This enables the tags change to a different broadcasting address until they are ready to change back. This allows the tags to still be accessible without constant interruption.
GPS Configuration

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>23</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Tad ID | Configuration Command | GPS fix ...
| Data Fields |

- **Configuration Command = 0x02**
  - **GPS Fix Interval**
    - This field represents the initial GPS fix interval. From this, it will increase relative to the thresholds set out in the battery configuration.

Battery Configuration

<table>
<thead>
<tr>
<th>0</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>23</th>
<th>24</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Tad ID | Configuration Command | GPS Battery Threshold | GPS Battery Threshold x2 | GPS Battery Threshold x4 | GPS Battery Threshold x8 | GPS Battery Threshold x16 | GPS Battery Threshold x32 | Telemetry Battery Threshold
| Data Fields |

- **Configuration Command = 0x03**
  - **GPS Battery Threshold**
  - **GPS Battery Threshold x2**
  - **GPS Battery Threshold x4**
  - **GPS Battery Threshold x8**
  - **GPS Battery Threshold x16**
  - **GPS Battery Threshold x32**
  - **Telemetry Battery Threshold**

The Battery Configuration packet provides access to all the battery thresholds. This is significant. Depending on the type of deployment for the tag dictates how these thresholds are spaced.
**Total Configuration**

<table>
<thead>
<tr>
<th>Configuration Command</th>
<th>Tag Address</th>
<th>Radio Sleep Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio Frequency P2</td>
<td>Radio Frequency P1</td>
<td>Radio Frequency P0</td>
</tr>
<tr>
<td>... Reset Threshold</td>
<td>Queue Unsent Threshold</td>
<td>GPS ...</td>
</tr>
<tr>
<td>... Fix Interval</td>
<td>GPS Battery ...</td>
<td></td>
</tr>
<tr>
<td>... Threshold x4</td>
<td>GPS Battery Threshold x8</td>
<td>GPS Battery ...</td>
</tr>
<tr>
<td>... Threshold x16</td>
<td>GPS Battery Threshold x32</td>
<td>Telemetry ...</td>
</tr>
<tr>
<td>... Battery Threshold</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Configuration Command = 0x04

The Total Configuration packet provides access to all the configuration settings outlined by the other configuration packets. This eases the mechanism for bulk reconfiguring of multiple tags.
4.4 Response Types

As part of the communication protocol, all response packets begin with a single byte Version number. This allows for the base stations to handle different versions of the position sensor’s software. This is followed by a 4 byte tag ID uniquely identifying each tag.

4.4.1 Status Response

- Packet Type = 0x20

The status response packet contains basic information about the tag. It is transmitted by the tag, in response to any initial communication from the base station. The base station uses this response to identify individual tags in the area that have the appropriate information. This is done without waking other tags in the vicinity.

- Build Reference
  - Identifies the date the tag software was compiled. Comparison against the main code repository allows for debugging of software versions.

- Battery Millivolts
  - When a status response is generated or a tag has a successful GPS fix, the battery voltage is constantly monitored and transmitted along with the packet. This also helps in the remote debugging of the behaviour of the tags.
4.4.2 Fix / GPS Response

- Packet Type = 0x21 / 0x25 (Fix / GPS)

Each Fix / GPS response contains information obtained from a single GPS fix taken by the tag. When a fix request is received, a tag will continuously transmit all of its unsent stored fixes to the base station as fix responses. When a GPS request is received from a base station the tag will endeavour to perform a GPS capture. If successful, the tag will transmit this as a single GPS response.

<table>
<thead>
<tr>
<th>Message</th>
<th>Message Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGA (GPS Fix Data)</td>
<td>Time, position, and fix related data</td>
</tr>
<tr>
<td>GSA (GPS Dilution of Precision (DOP) and Active Satellites)</td>
<td>GPS position fix mode, space vehicles used for navigation and DOP values</td>
</tr>
<tr>
<td>RMC (Recommended Minimum Specific GPS Data)</td>
<td>Coordinated Universal Time, status, latitude, longitude, speed over ground, date and magnetic variation of the position fix</td>
</tr>
</tbody>
</table>

Table 4.2: NMEA Message Summary [47]

Table 4.2 describes the NMEA-0183 message set that is used by the tags to generate the information contained in the Fix / GPS response. The processing of the messages on the tag, has been designed to limit the amount of unnecessary data that needs to be sent. For example the Time Field has fractions of a second removed.
4.4.3 No Fix Response

- Packet Type = 0x22

Comprising of the default response packet, the No Fix Response is transmitted by the tag either when it has no fix data, or once the stored unsent data has already be sent.

4.4.4 Fixes Received Response

- Packet Type = 0x23

The Fixes Received Response is transmitted by the base station in response to a No Fix Response from a tag.

- (Fix IDs)*
  - The Fixes Received packet will contain at least one Fix ID that was sent from a tag. A Fix ID with a length of 4 bytes means an upper limit of 29 Fix IDs per packet. This is formulated to reduce variability in set up of the radio between different packet types.

- Packet Number / Total Packet Number
  - To overcome the limitation of only 29 fix IDs sent at a time, multiple packets may be sent. The comparison between Packet Number and Total Packet Number allows the tag to decide on the next course of action.
4.4.5 Log Response

- Packet Type = 0x24

The Log Response packet contains a single log entry from a tag. The Entry RTC represents the time and date at which the log entry was created. The Log String contains the entry. The Log String—has a fixed length of 32 bytes. Zero padding is added to those log entries which are not long enough.

4.4.6 End Log Response

- Packet Type = 0x30

Comprising of the default response packet, the End Log Response is transmitted by the tag when it has finished transmitting all of the stored log data.
### 4.4.7 Configuration Response

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Address</td>
<td>6</td>
<td>Radio Sleep Time</td>
</tr>
<tr>
<td>Radio Frequency p1</td>
<td>7-8</td>
<td>Radio Frequency p1</td>
</tr>
<tr>
<td>Radio Frequency p0</td>
<td>15-16</td>
<td>Radio Frequency p0</td>
</tr>
<tr>
<td>Queue Reset Threshold</td>
<td>23-24</td>
<td>Queue Reset Threshold</td>
</tr>
<tr>
<td>GPS Fix Interval</td>
<td></td>
<td>GPS Fix Interval</td>
</tr>
<tr>
<td>GPS Battery Threshold</td>
<td></td>
<td>GPS Battery Threshold</td>
</tr>
<tr>
<td>GPS Battery Threshold x2</td>
<td></td>
<td>GPS Battery Threshold x2</td>
</tr>
<tr>
<td>GPS Battery Threshold x4</td>
<td></td>
<td>GPS Battery Threshold x4</td>
</tr>
<tr>
<td>GPS Battery Threshold x8</td>
<td></td>
<td>GPS Battery Threshold x8</td>
</tr>
<tr>
<td>GPS Battery Threshold x16</td>
<td></td>
<td>GPS Battery Threshold x16</td>
</tr>
<tr>
<td>GPS Battery Threshold x32</td>
<td></td>
<td>GPS Battery Threshold x32</td>
</tr>
<tr>
<td>Telemetry Battery Threshold</td>
<td></td>
<td>Telemetry Battery Threshold</td>
</tr>
</tbody>
</table>

A Configuration Response is transmitted by the tag when it either receives a configuration request or a change in configuration request. Unlike the change in configuration request that is separated into three requests, the response packet contains all of the remotely configurable fields that the tag has available.
Chapter 5

Base Station Design

This chapter outlines the technical and design decisions for the creation of a telemetry base station. It provides a description of the general software design, via a program flow diagram. The base station’s hardware is also explained from the component selection to the layout of the Printed Circuit Board (PCB). Finally, it covers the user-interface developed for data collected for individual position sensors.

The simplest implementation of the telemetry system, requires a standalone base station. This base station is capable of communicating with all the individual tags, while providing the received information to the respective end users for analysis. The base station provides a two-way communication link with the position sensors allowing for the expansion of the information, which is able to be collected from the tags. The procedures to conduct the information transfer are outlined in Communication Protocols Chapter 4.

Figure 5.1: Base Station
5.1 Software Design

The base station software is written in Python 2.7 and operates as an automated process under the Raspbian OS of the Raspberry Pi. The base station utilises the multiprocessing Python package that supports spawning processes and shared memory, sharing data between processes. This allows for the separation between the user - base station control and the base station telemetry polling loop. The flow chart in Figure 5.2 gives an overview of the processes and the top level procedures contained within each. During the start procedure, the base station checks and if necessary creates a SQLite3 database. This database is used in the base station process to store each position sensor’s telemetry data.

Figure 5.2: The flow-diagram of the base station software. A multi-process design utilising resource sharing to avoid unexpected interruptions from user interactions.
5.1.1 Client Process

The client process provides user control of the base station telemetry polling without interfering with any ongoing telemetry operation. Figure 5.3 displays the operations available to the end user via a command line. When an operation is selected, it changes the shared data value defining the telemetry method. Once any ongoing telemetry operation has been completed, the base station will then select and implement this method.

Connection to the client process is served up using a json Remote Procedure Call (RPC) server, allowing a client program to easily connect and disconnect remotely. It is currently restricted to use via localhost network only and the base station network is accessed by a Secure Shell (SSH) connection.

Figure 5.3: This is the base station’s user control client interface. It is implemented on the command line to provide access through an SSH tunnel and is asynchronous to the main base station process.
5.1.2 Base Station Process

The base station process is a more complicated process. The top level procedures initiated are, uploading of tag sensor data, base station telemetry configuration and operation of each telemetry polling method.

The communication with telemetry is achieved over Serial Peripheral Interface (SPI). The base station process uses a Python package SPI-Py, which is a C extension for Python.

The data upload and telemetry configuration procedures are not called constantly in the base station process. Data upload is initiated on a time basis while the telemetry configuration is completed on the number of base station loops.

Data Upload

The data upload procedure moves data stored in the base station’s local SQLite3 database and uploads it to a central PostgreSQL database stored on an Amazon Web Server. This upload is attempted on the process initialisation and will be repeated approximately hourly. There will be slight variation in the time, dependent on the base station process.

The data transfer was originally completed using a direct link to the central database. The ports used for this transfer are not consistently open by firewalls. This is circumvented by using a Hypertext Transfer Protocol (HTTP) database transfer on port 80, which is not normally closed.

The base station software uses PycURL. This is a Python interface to LibCurl, a free and easy-to-use client-side URL transfer library. PycURL supports the necessary POST request method, including user and password authentication.

Initially, each fix was sent individually to the central database. This was changed to sending packets containing up to a hundred fixes for each tag at a time. This reduced the time consuming process of authenticating each packet and assigning or creating appropriate tag data.

Once the fix data has been confirmed to be uploaded to the central database, the corresponding local fix data is deleted. This restrains the growth of the local database.

Telemetry Configuration

As the base stations will be easily accessible, any changes to the telemetry configuration can be made directly to the configuration file. Consequently, unlike the configuration in the position sensors, the base stations do not include an extra configurable settings structure. Also, as the base station is not restricted by any power consumption constraints, a number of power saving features are disabled to optimise the base station for their role of polling tags.
Telemetry Polling Method

The polling method is the same for all the requests carried out by the base station, which is a simple transmit - receive poll.

Control of the telemetry hardware is carried out over SPI. The polling method also utilises the General Purpose Input/Output (GPIO) available from the Raspberry Pi header to indicate when packets have been sent and received. The base station uses a Python package Raspberry-GPIO-Python, to control the GPIO on a Raspberry Pi.

As this package and the process runs under the Linux kernel on the Raspberry Pi, the real time performance for controlling the telemetry is unsuitable. This is alleviated by using the inbuilt interrupt detection in conjunction with threaded callbacks. This is only critical for use when the telemetry is receiving packets. It means that packet handling callback functions can run at the same time as the main program, in immediate response to an edge from the GPIO.

The receive packet handling callback function confirms that a valid packet has been received. The callback function then appends the packet to a global queue for further handling by the main program and resets the telemetry back into receive mode. As packets are added to the global queue, the main polling method withdraws them and applies the applicable packet formatting method. During transmit - receive, the formatted tag data is cached. Once the polling times out, the cache data is then entered into the local SQLite3 database.

With the acceptance of valid packets, the initial receive mode timeout is increased. This allows for more packets to be received, while avoiding a possible infinite loop problem.
5.2 Hardware Design

With the base station having to perform a multitude of functions ranging from data retrieval to providing a user-interface, a Raspberry Pi has been utilised as the base building block. On top of this, is the base station shield. This board takes advantage of the header connector. It brings out communication pins from the processor, providing the tag communication telemetry as well as an optional DC-DC converter.

5.2.1 Raspberry Pi

“The Raspberry Pi is a credit-card sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects and for many of the things that your desktop PC does, like spreadsheets, word-processing and games. It also plays high-definition video.” [48]

A Raspberry Pi provides many different options via Universal Serial BUS (USB) wireless connections or the High-Definition Multimedia Interface (HDMI). This eases the implementation complexity of the user-interface for the base station. Figure 5.4 shows an implementation of a GSM connected base station. Behind the telemetry antenna is the USB GSM modem, which provides an automatic link for data to be uploaded or base station program settings to be changed.

5.2.2 Shield Board

Figure 5.5 shows the shield board placed on the Raspberry Pi. The complete configuration of the shield board contains telemetry for the tags and a power supply DC-DC converter (for circuit design and board layout see Appendix C).

Telemetry

The telemetry hardware on the base station is the same as the telemetry hardware on the position sensors, which is outlined in Chapter 2. However, slight differences have been made, based on design constraints that are not present with the base station.

With the base station continually operating and powered by a constant stable power supply. The first design change lowers component count, by removing the pull up and down resistors required on the position sensors to stop leakage currents.

The base station is not restricted by PCB area or weight, so the second design decision incorporated the addition of an SubMiniature version A (SMA) connector. This is shown in Figure 5.5. It is the gold plated connector near the centre of the base station board. This allows the base station to utilise a wide range of purpose built antennas.
Figure 5.4: Contained within an IP66/IP67 enclosure, the base station consists of a shield mounted on top of and interfacing with an embedded Linux computer. This shield has a telemetry section similar to the solar telemetry module of the tags. It also has a wide range DC-DC converter, providing flexibility in the power supplied to the base station. The telemetry antenna is positioned at the front and top of the enclosure. This is a commercial WRT Series antenna that is centred around 868 MHz. Behind this is a Global System for Mobile communications (GSM) module, which provides external access. It allows the base station to upload information received from the tags, to a central database.
Base Station DC Power

The DC-DC converter provides the base station with the ability to be powered by any DC power supply ranging from seven to thirty six volts. This flexibility allows for the base station to be implemented in a wide range of situations from mobile to fixed. Where a stable five volt input voltage is able to be supplied for operation of the Raspberry Pi and telemetry, the DC-DC converter can be removed.

In Figure 5.5, the DC-DC converter circuitry forms an L-shape around the outside of the shield board. It connects to the five volt header pin of the Raspberry Pi. The five volt pin is directly connected to the micro-usb conventional power supply. For this reason when the converter is implemented, the micro-usb power connector is not used.

The main component of the converter is the Texas Instruments PTN78020. This a wide-input adjustable output switching regulator that is capable of step-down voltage conversion for loads up to six amps [49].

To decrease the noise level / ripple of the input and output, a higher order filter is used. Based upon schematic designs outlined in the Texas Instruments PTN78020 data sheet, a pi filter is included in series with input and output. By including this filter in the circuit, the data sheet claims a noise / ripple reduction of 20dB [49].

In a further attempt to limit noise between the DC-DC converter and the telemetry circuitry, a 0 ohm isolation resistor separates the two grounds. Allowances have been made in the design of the PCB to place this resister in different locations, depending on the level of noise disruption experienced by the telemetry radio frequency output.
5.3 Web-based Interface / Data Viewing

The user-interface was originally developed as a means of displaying data collected by the position sensors, to assist in debugging logic errors. The usability and layout of the user-interface lead to further development to provide respective end users with secure access to their tag data.

Developed as a web front end, the user-interface provides interaction with the user’s position sensors. Selecting a specific tag, moves the user to its individual page containing the detailed data. All the data from a tag is overlaid on Google Maps. The acquisition markers are selectable and once clicked show the more comprehensive information about the acquisition. This data is also available in a large table, which allows easy access to the complete data set. From the user’s tag deployments page, it is possible to download all the individual tag data into separate .csv files.

The user is also able to select multiple tags and overlay all of them on Google Maps. This allows for the data to be compared visually, for similar paths or overlay of territories. Figure 5.6 shows an example of the user’s tag deployment page.

From the debugging development, the user-interface includes a battery voltage graph. This provides informative feedback to the user about battery consumption decisions made for each tag.

The web front end client-side has been developed using AngularJS. The client is responsible for the processing and displaying of the information. The web front end was designed to alleviate the access load on the server-side, but also to make the data display more interactive and informative. The server-side of the system is a combination of NodeJS/Express and PostgreSQL. Express is a minimal and flexible NodeJS web application framework. It provides a connection between the web front end and the PostgreSQL database. All data which is transferred is compressed to decrease the bandwidth used by the user-interface.
Figure 5.6: This page contains the position sensors deployed by the user logged on. It presents an overview of the administration information of individual tags. It provides ordering based upon the last GPS acquisition stored in the main database. From here, a user can move to view the complete data of an individual sensor, display multiple sensors on a map overlay or download the data into a .csv file.
Chapter 6

Testing and Results

This chapter outlines the testing and results of the three key factors investigated of the position sensor system. These factors are the position accuracy, telemetry performance and power consumption. Also detailed in this chapter are the results of long-term experiments conducted to test the position sensor system as a whole, determining any performance bias in the devices.

6.1 Characterising Position Accuracy

The Global Positioning System (GPS) module used in the position sensor has a quoted horizontal position accuracy of <2.5 m. Position accuracy is a measure of the distribution of differences between the location determined by the GPS module and its actual location. The quoted horizontal position accuracy from the GPS module’s data sheet is based upon Circular Error Probable (CEP), which is the probability that 50% of the fixes will be contained within a certain radius.

The position sensor’s GPS performance was quantified by the measure of the position accuracy. This was achieved using a stationary test rig designed to hold six sensors. The test rig comprised of three sensors with patch antennas mounted on the top of the sensors and three sensors with whip antennas. Two of these whip antennas were mounted in an approximate north-south direction, with the third in an east-west orientation. The different antennas were positioned separately across the rig to avoid interference and bias from the location of their mounting. To ensure an adequate data set, the sensors were provided with a common stable power supply set to approximately 4 volts, which avoided the power consumption algorithm.

Figure 6.1 shows the test rig that is fixed underneath a skylight. This gives a clear sky view to the rig for maximum satellite exposure. The data gathered by the sensors was transmitted via the telemetry module. To promote the upload of each sensor’s data every hour, the sensors were configured accordingly. The GPS acquisition interval was set to 4 s and the enhanced Wake-On-Radio (eWOR) sleep time was shortened to 100 ms.
Figure 6.1: The stationary GPS position accuracy test rig. Constructed of 3D print parts around a PVC pipe, it is suspended underneath a skylight. This provides it with a clear sky view that maximises satellite exposure.

Early during the experiment, position sensor - 3544164930 stopped responding. To avoid interrupting the other sensors, the defective sensor was left in place. This sensor uses a whip antenna in an approximate north-south orientation.

Data from the five remaining position sensors was analysed separately to determine if there was a correlation in the accuracy of the location data collected and the antenna used. Due to differences in time-to-first-fix and the transmission time of collected data, the sensors yielded between 320447 and 411914 data points.

Figure 6.2 is a plot of the horizontal position errors of the location data gathered by position sensor - 3544165172, which utilised a patch antenna. Empirically determined the sensor has a CEP of a 4 m radius. This is greater than the theoretical best of <2.5 m from the GPS modules’ data sheet; however it is consistent with the other sensors’ CEP shown in Table 6.1 (see Appendix A for additional position accuracy results). The CEP values were resolved empirically because the formula used to calculate them is vulnerable to outliers in the data [50]. As a result of this, the magnitude of the error scope in Figure 6.2 has been restricted to a ±150 m square. This displayed a bias in the position sensors with a greater latitude error in the location data. This is persistent across each position sensor, irrespective of the antenna used.

Very common is the use of Gaussian noise models [51]. It provides significance to the mean and covariance of the data. Figures 6.3 and 6.4 show the relationship between the latitude and longitude errors and an independent, identically distributed (i.i.d) Gaussian model. According to the normalised quantile plots shown in Figure 6.3, the horizontal errors cannot be modelled as a Guassian distribution. The plots suggest long tails at both ends of the data distribution, as well as outliers in the data.
Figure 6.2: Horizontal position errors of fixes taken from position sensor - 3544165172, which utilises a patch antenna. The position sensor - 3544165172 shows a 50% probability $CEP$ of fixes within a radius of 4 metres. It shows a 95% probability (R95) within a radius of 14.7 metres. Both radii were determined empirically.

<table>
<thead>
<tr>
<th>Tag id</th>
<th>Antenna Type</th>
<th>CEP (m)</th>
<th>R95 (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3544165172</td>
<td>patch</td>
<td>4</td>
<td>14.7</td>
</tr>
<tr>
<td>3544034068</td>
<td>patch</td>
<td>2.7</td>
<td>10.1</td>
</tr>
<tr>
<td>3542326081</td>
<td>whip</td>
<td>3.8</td>
<td>15.3</td>
</tr>
<tr>
<td>3542264594</td>
<td>patch</td>
<td>3.8</td>
<td>14.6</td>
</tr>
<tr>
<td>3609997124</td>
<td>whip</td>
<td>3.5</td>
<td>14.5</td>
</tr>
</tbody>
</table>

Table 6.1

Figure 6.4 shows the autocorrelation of the latitude and longitude errors taken from position sensor - 3544165172. A characteristic of i.i.d Gaussian noise is that the theoretical autocorrelation is zero after $dt$, where $dt$ is the sampling period. As the sampling period of the position sensors fluctuate during operation, data used for the autocorrelation, was taken from the measured data at five minute intervals. Compared to the sampling period, any deviation from this period would be small. The latitude and longitude error autocorrelations clearly did not reach zero after the five minute sample period. Instead it took approximately an hour for the autocorrelation to reach zero.
If a Gaussian model is used to fit the GPS position measurements to account for outliers, it will have to have an artificially large variance. This will still not account for the observed autocorrelations.

While not within the scope of this thesis, the GPS position uncertainties were analysed. Alternative noise models were suggested and published in the Enhanced Noise Models for GPS position paper [52], refer Appendix B.

Figure 6.3: Normalised quantile plot of the latitude (left) and longitude (right) position errors taken from position sensor - 3544165172, which utilises a patch antenna. The distribution of the horizontal errors cannot be modelled as a Gaussian distribution, according to the normalised quantile plot. The plot suggests long tails at both ends of the data distribution as well as outliers in the data.

Figure 6.4: Autocorrelation of latitude (left) and longitude (right) position errors taken from position sensor - 3544165172, which implements a patch antenna. The sampling period of the position sensor was 8 seconds, although during operation the period can fluctuate. The data used for the autocorrelation was taken from the measured data at 5 minute intervals. Compared to the sampling period any deviation from this period would be small. This data is taken from a data set of ∼2300 hours but the time axis has been cut off at 10 hours for readability.
6.2 Characterising Telemetry Performance

The characterisation of the telemetry performance is important because it directly determines the maximum range of the position sensor system. This section investigates the performance of the position sensor’s Radio Frequency (RF) chain and the real world application of the position sensors.

6.2.1 Antenna Impedance Matching

The initial telemetry module of the position sensor was designed with a simple whip antenna. The wildlife version trails the antenna out the back of the tag. To avoid damage, the livestock ear tag version had the antennas positioned internally in the case. This is outlined in the Telemetry Module Hardware Chapter 2. In order to test the effectiveness of both the antennas, a test telemetry module was connected to a radio frequency network analyser (Agilent E5071B). Using a reflection measurement (S11), which is where the input and output ports are the same, the return loss of the antennas was measured. The higher the return loss, the more effective the antenna is at dispersing the power of the signal. As part of testing the telemetry’s whip antenna and RF trace, the magnitude of the return loss was measured, as well as the characterisation of the impedance. This is displayed using a Smith Chart. The impedance of the antenna and trace is important because the Integrated Ceramic Passive (ICP) component used in the RF chain (see Section 2.1), relies on a 50 Ω impedance match to the antenna pin. An oversight in the design of the initial telemetry module meant that no matching circuitry was included in the RF chain.

![Graphs showing return loss and Smith Chart](image)

**Figure 6.5:** Shown here is the logarithmic magnitude of the return loss (left) and the Smith Chart (right) of the wildlife position sensor. It is a whip antenna constructed of a short, solid copper wire (proposed for weatherproofing), soldered to a braided shark-resistant leader line (providing resilience from damage). With an intended operating frequency of 868 MHz, the return loss of the antenna is -18.041 dB and the Smith Chart shows a 38.057 Ω real and a 731.26 mΩ complex impedance.
Figure 6.6: Shown here is the logarithmic magnitude of the return loss (left) and the Smith Chart (right) of the livestock ear tag position sensor. It has a whip antenna constructed of a short, solid copper wire that is bent to conform to the shape of the ear tag. At the intended operating frequency of 868 MHz, the return loss of the antenna is -10.461 dB and the Smith Chart shows a 79.183 Ω real and a -26.927 Ω complex impedance.

Figure 6.5 shows the wildlife position sensor’s telemetry antenna’s return loss measurements. It is a whip antenna constructed of a short, solid copper wire (proposed for weatherproofing), soldered to a braided shark-resistant leader line (providing resilience from damage). With an intended operating frequency of 868 MHz, the return loss of the antenna is -18.041 dB and the Smith Chart shows a 38.057 Ω real and a 731.26 mΩ complex impedance.

Figure 6.6 shows the livestock position sensor’s telemetry antenna’s return loss measurements. It has a whip antenna constructed of a short solid copper wire, which is bent to conform to the shape of the ear tag. At the intended operating frequency of 868 MHz, the return loss of the antenna is -10.461 dB and the Smith Chart shows a 79.183 Ω real and a -26.927 Ω complex impedance.

Both Figures 6.5 and 6.6 measurements were conducted with the same position sensor components. The only difference being, the antenna construction and shape used. The livestock antenna has a lower Q-factor with the desired 868 MHz centre frequency return loss being approximately 8 dB less with a wider bandwidth. This is supported by the corresponding Smith Chart results showing a 79.183 Ω real and a -26.927 Ω complex impedance. This does not closely match the required 50 Ω impedance. The bending of the antenna to correspond to the shape of the livestock ear tag, plus moving the antenna in relationship to the ground plane, contributes to the effectiveness of the antenna [53]. In this case there is a mismatch for the required input.
Figure 6.7: Shown here is the logarithmic magnitude of the return loss (left) and the Smith Chart (right) of the livestock ear tag position sensor. It has a whip antenna constructed of a short, solid copper wire that is bent to conform to the shape of the ear tag. Matching components were added on the radio frequency trace, a 10 nH shunt inductor and a 1.2 pF capacitor. At the intended operating frequency of 868 MHz the return loss of the antenna is -10.915 dB and the Smith Chart show a 38.406 Ω real and a -23.168 Ω complex impedance.

Figure 6.7 shows the results of the livestock position sensor with matching components provisionally added to the RF chain, a 10 nH shunt inductor and a 1.2 pF capacitor. At the intended operating frequency of 868 MHz, the return loss of the antenna is -10.915 dB. The Smith Chart shows a 38.406 Ω real and a -23.168 Ω complex impedance. It reduces the bandwidth, in comparison to position sensor without matching circuitry. However, there was no improvement in the return loss at the centre frequency.

These experiments were conducted without the use of cases. This is not applicable to the wildlife position sensor, which trails the antenna out of the rear of the case. However, the loading effects of the plastics used in the case outlined in Section 2.4, will shift the resonant frequency. The frequency change is easily compensated for by adjusting the length of the antenna [54].

Attempts to measure the necessary changes with the use of the RF network analyser were impeded by the introduction of outside effects, such as any pressure contact applied to the testing cable from the case. Therefore the adjustments of the livestock antenna were achieved experimentally. This was with the use of a base station reporting the Received Signal Strength Indication (RSSI) and Link Quality Indicator (LQI) of the test packets as changes were made, until an adequate signal performance was reached.
6.2.2 Telemetry Transmission Range

To provide a more realistic test of the position sensor’s telemetry system, a position sensor was deployed on a cyclist. Figure 6.8 shows the position sensor attached to the cyclist’s helmet, with a clear sky view. As it was mainly to be used off-road (downhill), the position sensor packaging was custom designed to be located within the helmet structure in order to avoid compromising the safety of the helmet. The position sensor incorporated two 300 mAh batteries to increase the life of the device because solar charging was not implemented on this test telemetry board. The device was programmed to attempt a GPS acquisition every eight seconds, with a five minute timeout on the telemetry transmission to avoid saturating the network.

Figure 6.8: A customised position sensor attached within a cyclist’s helmet. It is split into two PLA 3D printed cases. The case located in the top of the helmet, is the position sensor and the case at the back is the battery pack and power control. The battery pack consists of two 300 mAh batteries. The sensor GPS acquisition is configured to every eight seconds, with a five minute timeout for telemetry transmissions.

Figure 6.9 provides an on-line snapshot of a marker map containing location and telemetry data from a test position sensor. The red markers indicate GPS fixes, while the green arrows represent successful base station-position sensor communication. The base station is represented by the yellow star in the cluster of GPS fixes. The data displayed on the marker map is the combination of multiple different rides. The green arrows represent a select number of successful transmissions. The solid blue circle indicates the farthest direct line of sight transmission of 2.4 km and has an RSSI of -84 dBm and LQI of 129. The two dashed yellow circles are transmissions where the line of sight is impeded, the farthest is 1 km and has an RSSI of -98 dBm and LQI of 145.

The telemetry transmissions were only considered successful if the sensor was capable of transmitting its complete set of stored data. The use of the off-road cycle helmet was a more realistic test towards the deployment of the position sensors on wildlife and livestock. This is because it moved in and out of foliage and interacted with geographical features, which impact on the transmission of location data from the position sensor.
To accommodate intermittent transmission connections between the base station and position sensors, the sensors are required to maintain a copy of the data, until a fixes received packet transmission is received from the base station. This is further outlined in Chapters 3 and 4. Although not precisely displayed due to the saturation of data points, the helmet mounted test position sensor confirmed that the sensors are capable of the required storage behaviour.

Figure 6.9: On-line snapshot of a marker map containing location and telemetry data from a test position sensor. The red markers indicate GPS fixes, while the green arrows represent successful base station-position sensor communication. The base station is represented by the yellow star in the cluster of GPS fixes. The solid blue circle indicates the farthest direct line of sight transmission of 2.4 km and has an RSSI of -84 dBm and LQI of 129. The two dashed yellow circles are transmissions where the line of sight is impeded, the farthest is 1 km and has an RSSI of -98 dBm and LQI of 145.
6.3 Characterising Power Consumption

The main feature of the position sensors is the low power consumption required for it to operate. This section characterises the power used by the telemetry module as well as the effectiveness of the solar energy harvesting.

6.3.1 Telemetry Module Power

The transceiver on the telemetry module was chosen to provide a low power continuous transmission link. To confirm the current values used for telemetry power consumption calculations are correct, a position sensor was connected to the Keithley Model 2308 Portable Device Battery/Charger Simulator. This provided two analog output terminals, which output a voltage based on the measured current. This connected to an oscilloscope, which displayed the output of 10 mA/V (each volt out represents 10 mA). Both the transmit and receive (eWOR) modes of the transceiver were measured from this output. The results were compared to the values provided from the transceiver’s data sheet and those set via the sensor’s configuration.

Figure 6.10: The power consumption of the telemetry module during a transmission phase. This shows the position sensor transmitting a status response that takes approximately 274 ms.

This time corresponds accurately to the data packet length (41 bytes, including 12 bytes preamble, 4 bytes sync word and 2 bytes Cyclic Redundancy Check (CRC)) divided by the transmit data rate (150 B/s): 41 B / 150 B/s = 0.2743 s = 273.3 ms. During this phase it draws an average of 46 mA. (Vertical axis scale: 10 mA per division.)
Figure 6.10 shows the oscilloscope capture of the position sensor transmitting a status response. During this transmission phase, the position sensor draws an average of 46 mA, which includes the CPU module. The length of this current draw directly corresponds to the length of the data packet, divided by the transmit data rate. For the status response packet the oscilloscope capture shows the transmission takes approximately 274 ms. This compares with the expected time for the status response: data packet length (41 bytes, including 12 bytes preamble, 4 bytes sync word and 2 bytes Cyclic Redundancy Check (CRC)) divided by the transmit data rate (150 B/s) equals, \( \frac{41 \text{ B}}{150 \text{ B/s}} = 0.2743 \text{s} = 273.3 \text{ ms} \). Consequently, the result confirms the transceiver’s data sheet values and the position sensor’s configuration settings for transmitting.

The receive mode (eWOR) is the essential low power attribute, which allows for the continuous operation of the transceiver with minimal power consumption. The configuration of the eWOR is discussed in the Telemetry Software Chapter 3.

Figure 6.11: The power consumption of the telemetry module during an enhance Wake On Radio operation. The transceiver is in low power sleep mode, until it wakes to check for a radio signal before dropping back to sleep. Cursors A and B indicate the time positions of the receive phase of the eWOR, which is outlined further in Figure 6.12. The difference between the two is 2 seconds, which matches exactly the position sensor’s sleep configuration value of 2000 ms. (Vertical axis scale: 10 mA per division.)
Figure 6.11 shows the oscilloscope capture of the eWOR operation. The transceiver stays in low power sleep mode until it wakes to check for a radio signal. If the correct signal is not discovered, then the transceiver places itself back into sleep mode. In Figure 6.11 the cursors A and B outline the position of the receive phase. The measured difference between the two separate phases is 2 seconds, which exactly matches the sleep interval value set in the sensor’s configuration.

Figure 6.12: The power consumption of the telemetry module during an enhance Wake On Radio RX phase. The two main phases of the operation shown are: a = Transceiver Idle mode, b = Transceiver high performance RX mode. The transceiver high performance RX mode takes approximately 1.56 ms. During this phase it draws an average of 24 mA with a peak of 28 mA. (Vertical axis scale: 10 mA per division.)

Figure 6.12 shows the receive phase of the eWOR in greater detail. The operation consists of two main phases, the transceiver’s idle mode and high performance receive mode. These are indicated by a and b respectively on the oscilloscope capture. The transceiver’s high performance receive mode takes approximately 1.56 ms, during which time, the average current draw is 24 mA with a peak of 28 mA.
Combining the results from Figures 6.11 and 6.12, the approximate average eWOR current consumption can be calculated:

<table>
<thead>
<tr>
<th>Phases</th>
<th>Current (mA)</th>
<th>Time (ms)</th>
<th>Current Consumption (mAms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>0.0005</td>
<td>2000</td>
<td>1</td>
</tr>
<tr>
<td>Idle</td>
<td>3</td>
<td>0.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Receive</td>
<td>24</td>
<td>1.56</td>
<td>37.44</td>
</tr>
</tbody>
</table>

| Total  | 2002.06      | 38.94     |

\[
eWOR \text{ average current consumption} = \frac{38.94}{2002.06} = 0.01945 \text{ mA}.
\]

The average eWOR current consumption is added to the overall standby consumption of the position sensor. This includes leakage and quiescent currents. The Keithley Battery/Charger Simulator’s inbuilt long integration current mode shows a difference of 0.023 mA when the transceiver is operating in eWOR mode over a 1 minute integration time. This is different to the calculated consumption using the approximate current averages for the oscilloscope captures. This may be explained by the rudimentary approximation of the current consumption. However, the current measurements obtained from the oscilloscope captures do reinforce the values from the transceiver’s data sheet and the position sensor’s configuration settings. As the time of the idle and receive phases are predetermined, the sleep phase heavily influences the transceiver’s eWOR average current consumption. This allows for the position sensor’s power consumption characteristics to be accurately calculated prior to deployment.
6.3.2 Solar Charging Performance

The addition of a solar charging circuit outlined in Telemetry Hardware Chapter 2, was designed to allow for a lower GPS acquisition interval. This is because the solar charging would be able to replenish the battery, provided that the interval is of a suitable length.

Figure 6.13: The perspex weatherproof enclosure on the roof of the Otago University Physics building was used as the solar test environment.
To test the effectiveness of the solar charging circuitry and the acquisition interval algorithm discussed in the Telemetry Software Chapter 2, a test position sensor was placed on the roof of the Physics building at Otago University. Figure 6.13 shows the position sensor, which was located in a weatherproof perspex enclosure on the roof providing it with a clear sky view. This sensor stack had a larger 300 mA battery, chosen for no other reason than availability. It was packaged in an experimental 3D print ABS plastic case, which had an opaque top section.

Figure 6.14: The top plot is a graph of the battery voltage of a stationary test tag starting the 5th of September, 2014 and ending the 11th of January, 2015. Clearly apparent is the time dependent oscillation of the voltage. The period of this oscillation is equal to one day-night cycle. The two bottom plots show evidences of the adaptive data acquisition algorithm. The algorithm adjusts the length of fix interval in accordance with the battery voltage until it is able to reach a sustainable charge cycle. In the period between the 27th of November, 2014 and the 11th of January, 2015 this test tag sustained a charge cycle finding an equilibrium between 4.0 and 4.168 volts.
This solar test was a long term experiment with optimal conditions for the sensor. The clear sky view meant it was in direct sunlight for solar charging and was exposed to the greatest GPS satellite coverage possible. The test sensor was configured with a 30 minute GPS acquisition interval that was short enough for the GPS module to maintain valid almanac and ephemeris information. This also provided the sensor’s solar energy harvesting with enough time to replenish the battery. The telemetry on the sensor was configured to a 10 hour timeout, which avoided the sensor being woken repeatedly.

The battery voltage is recorded alongside each GPS acquisition. Figure 6.14 shows a plot of the test sensor’s battery voltage from the 5th of September, 2014, to the 11th of January, 2015. Clearly apparent is the time dependent oscillation of the voltage. The period of this oscillation is equivalent to a single day-night cycle. The adaptive data acquisition algorithm is also evident. The algorithm adjusts the length of fix interval in accordance with the battery voltage, until it is able to reach a sustainable charge cycle. In the period between the 27th of November, 2014 and the 11th of January, 2015 this test tag sustained a charge cycle finding an equilibrium between 4.0 and 4.168 volts.

The sustainable charge cycle voltage level is proportional to the available solar energy. Figure 6.15 shows the estimated available solar energy [55]. The median horizontal position coordinates from the GPS accuracy experiment (see Section 6.1) provide the location for the calculation of the estimate. This combines an image of the local landscape with irradiance data from the nearest climate station. The daily cumulative solar energy shown in Figure 6.15, is highest during the summer months around December and lowest around June during winter. This is corroborated by the solar test sensor’s battery voltages shown in Figure 6.14.
Closer observation of the battery voltage from the solar test position sensor lead to the discovery that when relatively fully charged, the sensor provides an indication of the cloud cover of a day.

Figure 6.16 provides the battery voltage readings for November 2014 from the position sensor. Days 19-21 show a drop in battery voltage because as the sensor’s solar energy harvesting is incapable of matching the power consumption of the sensor.

Figure 6.16: Plot of the battery voltage of a stationary solar test tag for November, 2014. The days between the 19\textsuperscript{th} - 21\textsuperscript{st} show a drop in battery voltage and again three days later. This is when the sensor’s solar energy harvesting is incapable of matching the power consumption of the sensor.

Figure 6.17: Daily Solar Energy for November 2014. The data provided from the weather station located on the roof of the Science 3 Building, University of Otago Campus, Dunedin, New Zealand.
Figure 6.17 is the corresponding daily solar energy for November 2014. This data was collected by the Otago University, Department of Physics weather station. It shows that on the 19th - 21st of November, daily solar energy was less than surrounding days. This is a result of cloud cover blocking the sun. From this comparison, it can be concluded that below approximately 12MJ/m$^2$ of daily solar energy, the sensors are not capable of a 30 minute GPS acquisition interval.

Of note is November 2nd. The daily solar energy was 10MJ/m$^2$, however, this does not appear as a battery voltage drop event in the solar test sensor’s data. Figure 6.18 shows the constant current discharge curve for a Lithium-Ion battery, which demonstrates similar characteristics as the battery used in the solar experiment. As the battery approaches voltages around the nominal voltage, the voltage change becomes more gradual, taking longer to diminish with power consumption. Therefore, while it is likely to be discharging at a similar rate, it is not as apparent in the sensor’s voltage data, as when the battery is relatively fully charged.

Figure 6.18: Constant current discharge curve for a Lithium-Ion battery [56]. This discharge curve demonstrates similar characteristics as the battery used in the solar test experiment. This discharge curve has a nominal voltage of 3.7 V and a discharge cut-off voltage of 3.0 V.
Chapter 7
Summary and Future Development

The ultimate aim of this work was to produce a wireless configurable position sensor capable of being deployed on wildlife and livestock in remote areas. This device must present data collected to an end-user in near real time.

During the course of this research, a telemetry system was designed to create a position sensor capable of collecting and transmitting location data. This involved the design, development and testing of a telemetry module plus a corresponding base station. For the telemetry system to be considered successful it had to out-perform a previously deployed micro-cell Global System for Mobile communications (GSM) base station. This base station had a transmission range of 300-700 m and a data rate of a single Global Positioning System (GPS) acquisition every 0.750-0.875 seconds [2].

The position sensor’s performance was characterised through laboratory and field testing to investigate three areas. These were position accuracy, telemetry transmission performance and power consumption. With over 300,000 location acquisitions, the static position sensors were found to have a Circular Error Probable (CEP) accuracy of <4 m, independent of the GPS antenna chosen.

The helmet mount device demonstrated reliable data transmission out to 2.4 km with a clear line of sight. Approximately every 0.387 s, the position sensor transmitted each location packet. This packet included extra location error information for post analysis.

As part of this research, refinements were made to the position sensor’s software to increase its overall power-efficiency. This involved removing unnecessary timing loops and improving sleep procedures. These refinements, along with the use of the adaptive acquisition interval algorithm, solar charging and a sensible starting interval, provide the position sensors with the capability of an indefinite deployable lifetime. This was demonstrated with a starting GPS acquisition interval of 30 minutes and a clear sky view.
In comparison to a micro-cell GSM base station, the performance results of the telemetry system can be concluded to be a successful telemetry option with a greater range and faster data rate. However, the location acquisition data would be too infrequent for modelling instantaneous behaviour of animals, such as feeding or drinking. It does provide a mechanism for the profiling of an animal’s long-term behaviour.

Further improvement to the position sensor’s hardware and software would be focused towards improving the power consumption performance. This would decrease the location acquisition interval, enabling more animal characteristics to be investigated.

7.1 Future Development

This section reviews enhancements of the hardware and software of the position sensors and base station with respect to future development.

7.1.1 Position Sensor Enhancements

Combining Modules

The modular design has proved to be significantly beneficial in the development and debugging phases of this project. As the separate modules have advanced and become more reliable, the integration into a single platform is desirable as it reduces the overall cost of each device. Figure 7.1 shows a combined livestock ear tag platform, which is currently in development. This combines all of the modules and internal antennas into a single board platform with the dimension of a standard livestock ear tag.

Figure 7.1: A single board livestock ear tag that combines all of the modules and internal antennas.
Changing Microprocessors

This research utilised the STM32F103 microprocessor. It was chosen to provide a common development architecture, which met the requirements of all the projects in which it was incorporated. This microprocessor is more powerful than required to meet the processing demands of the position sensor. In future iterations of the position sensor, a smaller, cheaper, more power-efficient processor could be employed in the design.

Additional Sensors

The position sensors currently only implement a GPS module. Further development of the sensor by adding additional sensors will extend its applicability. This will provide more information about the tagged animal. The inclusion of sensors such as an accelerometer and pressure sensor would provide additional information between GPS acquisitions about an animal’s movement and the environmental conditions to which the animal is exposed.

Limit Transmission Confirmation Number

Designed as part of the communication protocol (see Chapter 4), the position sensors will only attempt to transmit the complete set of GPS acquisition data on request of the base station. The sensor then waits to receive confirmation of the fix identity that the base station has received. This worked in stationary tests and for the most part throughout tracking testing. However, issues of miscommunication did arise during tracking. If the sensor has stored a large fix queue, it takes an extended amount of time to complete all of the transmissions, but in some cases this will not occur. A solution is to limit the number of packets transmitted in each sequence. This would allow the queue to be emptied gradually, with multiple base station - sensor handshakes. This means the position sensors would avoid power consuming re-transmits.

7.1.2 Base Station Enhancements

Additional Microprocessor

The Raspberry Pi provides an excellent main board on which to build the base station platform. However, it runs an operating system where the predictability of the response to interrupts from the transceiver is uncertain. An alternative solution that would provide true real-time performance and predictability is implementing an intermediary microprocessor. This would be a similar operation to the current implementation of the position sensor, which currently implements a STM32 microprocessor. This solution would allow the microprocessor to complete an interruptible operation. This would free the Raspberry Pi’s CPU to process the result of the interrupt at another more convenient time.
Separate Receive and Transmit

The continuous polling operation of the base station is limited only by the time it takes for the transceiver to transition between receive mode and transmit mode. Separating the receive and transmit functions on the radio on the shield board would remove this limitation. This could be achieved by using two transceivers. One would continually poll position sensors, while the other would be set to constantly receive. To avoid crosstalk between the transmit and receive transceivers, the receive transceiver can be configured to ignore all transmit polling packets. This is achieved by utilising the inbuilt packet-handling address filter.

Software Improvements

The focus for the software improvements on the base station is the flow of data from the position sensors to the end-user. With the current design there is duplication created by the intermittent transmission of data from the sensors and lag in the end-user experience. The lag is created when there is a vast number of location acquisitions available.

Future revisions of the base station’s software would work on the removal of all duplication from an individual base station level. Consequently, this duplicated data would not be propagated into the main online database.

From the main database to the end-user, the experience can be improved by reducing the query size, which will provide more modest data loads. This would supply the end-user with a more enhanced interface and with instantaneous presentation of tracking data.

7.2 Epilogue

Satellite tracking of wildlife and livestock is well established. With advancements in technology, the numbers and species of animals tagged are increasing.

To this end, the ultimate objective of this research focused on designing and producing a robust and flexible telemetry system for the position sensors. The design carried out, maintained the key design philosophy of light weight and power efficient sensors. The overall system presents the end-user with a near real time display of the tagged animals’ location and allows for these position sensors to be configured wirelessly.

The design and development of any such practical research project is an ongoing process. Consequently, with exciting possibilities from the hardware layer up to the telemetry protocol, the opportunities for this research to be furthered seem limitless.
Bibliography


[16] Power considerations for 2g & 3g modules in mid designs. Technical report.


Appendix A

Additional Position Accuracy Results

The following pages in this Appendix include the additional position accuracy results from the stationary, clear sky view Global Positioning System (GPS) test.

A.1 Position Sensor - 3609997124

Figure A.1: Horizontal position errors of fixes taken from position sensor - 3544165172, which utilises a patch antenna. The position sensor - 3609997124 shows a 50% probability Circular Error Probable (CEP) of fixes within a radius of 4 metres. It shows a 95% probability (R95) within a radius of 14.7 metres. Both radii were determined empirically.
Figure A.2: Normalised quantile plot of the latitude (left) and longitude (right) position errors taken from position sensor - 3609997124, which utilises a whip antenna. The distribution of the horizontal errors cannot be modelled as a Gaussian distribution, according to the normalised quantile plot. The plot suggests long tails at both ends of the data distribution as well as outliers in the data.

Figure A.3: Autocorrelation of latitude (left) and longitude (right) position errors taken from position sensor - 3609997124, which implements a whip antenna. The sampling period of the position sensor was 8 seconds, although during operation the period can fluctuate. The data used for the autocorrelation was taken from the measured data at 5 minute intervals. Compared to the sampling period any deviation from this period would be small. This data is taken from a data set of ~2300 hours but the time axis has been cut off at 10 hours for readability.
### A.2 Position Sensor - 3542264594

Figure A.4: Horizontal position errors of fixes taken from position sensor - 3544165172, which utilises a patch antenna. The position sensor - 3542264594 shows a 50% probability CEP of fixes within a radius of 4 metres. It shows a 95% probability (R95) within a radius of 14.7 metres. Both radii were determined empirically.

![Horizontal Position Errors](image1)

Figure A.5: Normalised quantile plot of the latitude (left) and longitude (right) position errors taken from position sensor - 3542264594, which utilises a patch antenna. The distribution of the horizontal errors cannot be modelled as a Gaussian distribution, according to the normalised quantile plot. The plot suggests long tails at both ends of the data distribution as well as outliers in the data.

![Quantile Plot](image2)
Figure A.6: Autocorrelation of latitude (left) and longitude (longitude) position errors taken from position sensor - 3542264594, which implements a patch antenna. The sampling period of the position sensor was 8 seconds, although during operation the period can fluctuate. The data used for the autocorrelation was taken from the measured data at 5 minute intervals. Compared to the sampling period any deviation from this period would be small. This data is taken from a data set of ∼2300 hours but the time axis has been cut off at 10 hours for readability.
A.3 Position Sensor - 3542326081

Figure A.7: Horizontal position errors of fixes taken from position sensor - 3544165172, which utilises a patch antenna. The position sensor - 3542326081 shows a 50% probability CEP of fixes within a radius of 4 metres. It shows a 95% probability (R95) within a radius of 14.7 metres. Both radii were determined empirically.

Figure A.8: Normalised quantile plot of the latitude (left) and longitude (right) position errors taken from position sensor - 3542326081, which utilises a whip antenna. The distribution of the horizontal errors cannot be modelled as a Gaussian distribution, according to the normalised quantile plot. The plot suggests long tails at both ends of the data distribution as well as outliers in the data.
Figure A.9: Autocorrelation of latitude (left) and longitude (longitude) position errors taken from position sensor - 3542326081, which implements a whip antenna. The sampling period of the position sensor was 8 seconds, although during operation the period can fluctuate. So the data used for the autocorrelation was taken from the measured data at 5 minute intervals. Compared to the sampling period any deviation from this period would be small. This data is taken from a data set of \(\sim 2300\) hours but the time axis has been cut off at 10 hours for readability.
A.4 Position Sensor - 3544034068

Figure A.10: Horizontal position errors of fixes taken from position sensor - 3544165172, which utilises a patch antenna. The position sensor - 3544034068 shows a 50% probability CEP of fixes within a radius of 4 metres. It shows a 95% probability (R95) within a radius of 14.7 metres. Both radii were determined empirically.

Figure A.11: Normalised quantile plot of the latitude (left) and longitude (right) position errors taken from position sensor - 3544034068, which utilises a patch antenna. The distribution of the horizontal errors cannot be modelled as a Gaussian distribution, according to the normalised quantile plot. The plot suggests long tails at both ends of the data distribution as well as outliers in the data.
Figure A.12: Autocorrelation of latitude (left) and longitude (longitude) position errors taken from position sensor - 3544034068, which implements a patch antenna. The sampling period of the position sensor was 8 seconds, although during operation the period can fluctuate. The data used for the autocorrelation was taken from the measured data at 5 minute intervals. Compared to the sampling period any deviation from this period would be small. This data is taken from a data set of \(~2300\) hours but the time axis has been cut off at 10 hours for readability.
Appendix B

Publication

This Appendix includes the publication, Enhanced Noise Models for GPS positioning, which involves the position sensor system.
Enhanced Noise Models for GPS positioning

Andrew W.R. Soundy, Bradley J. Panckhurst, Tim C.A. Molteno
Elec Research
Department of Physics
University of Otago
tim@elec.ac.nz

Abstract—We analyze GPS position recordings, and show that they have long term autocorrelation functions. The traditional approach of assuming Gaussian uncertainties is therefore potentially problematic. We suggest some alternative noise models such as the Ornstein-Uhlenbeck process or autoregressive process, that can be used for state estimation.

I. INTRODUCTION

Positions reported by satellite tracking systems, such as GPS, contain uncertainties in the measured position. Typically the process is modeled by assuming that the GPS device reports the true position \( X_0 \) with additive noise in the form of a stochastic process, i.e.,

\[
X_t = X_0 + N_t,
\]

where \( N_t \) is the stochastic process representing the position error. The measurement error \( N_t \) is usually taken to be independent and normally distributed with zero mean and variance of \( \sigma^2 \).

In this paper we analyze some GPS position records from known positions. We then characterize the error signal \( N_t \), and propose noise models that fit the data better than the independent, identically distributed (i.i.d) Gaussian model in common use. Previous work has recognised the problems with fitting an i.i.d Gaussian noise model to GPS measurements\cite{1}. Two models have been proposed to model the measurement noise more accurately. One is a moving average (MA) processes, the other is an autoregressive (AR) model\cite{15}. We will analyse both to determine which model fits the GPS errors best.

The autocorrelation (or autocovariance) between \( X_t \) and \( X_s \) for some process \( \{X_t\} \) is given by \cite{4}

\[
\text{cov}(X_t, X_s) = E[(X_t - E(X_t))(X_s - E(X_s))]
\]

where \( E[X] \) is the expectation \( \int x f(x) \, dx \) where \( x \in X \). One of the ‘pleasant’ characteristics of i.i.d Gaussian noise is that the theoretical autocorrelation is zero after lag of \( dt \), where \( dt \) is the sampling period. Figure 1 shows the autocorrelation of GPS noise, taken from a GlobalSat BU-353S4 GPS unit on the L1 frequency with sampling period of 1 second. Clearly the autocorrelation of the error is not zero after 1 second, indeed it takes approximately an hour for the autocorrelation to reach zero.

We present experimental data and generate models for the uncertainties in GPS position measurements. To determine which noise model is the ‘best’ we propose using the Akaike Information Criterion (AIC) which uses Maximum Likelihood Estimation to determine the likelihood of a set of parameters and penalises the number of adjustable parameters to safeguard against over-determining solutions.

II. AKAIKE INFORMATION CRITERION

For a statistical model of the GPS positioning error process, with \( k \) parameters, the Akaike Information Criterion \cite{2} is given by

\[
A_k = 2k - \log(L),
\]

where \( L \) is the maximum likelihood for the model given the process data. This is the probability of the positioning data given the parameter values.

The Akaike Information Criterion (AIC) is based in information theory and provides a balance between how well the model fits the data and the number of parameters in the model, to guard against overfitting. It can be shown that the AIC is asymptotically optimal in selecting the model with the least mean square error under the assumption that the exact ‘true’ model is not in the candidate set \cite{3}.

However it should be noted that the AIC does not say anything about the absolute probability of a model being
correct or incorrect, it only determines relative probabilities between models. Therefore the AIC will give no indication if none of the candidate models fit the data well.

We will now briefly go through the method of determining the AIC value of a noise model for the noise models that we are considering, starting off with the commonly used i.i.d Gaussian noise model.

A. AIC for Gaussian Models

The simplest noise model is a two parameter model that treats each successive measurement as statistically independent and normally distributed.

\[ \Delta P \sim N(\mu, \sigma) \]

We therefore maximize the likelihood

\[ L(\mu, \sigma|x) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \]

over the \( \mu - \sigma \) parameter space. This is easiest to do using the log-likelihood, as we can find the maximum likelihood of a single measurement \( x \) from \( \ln(L) = \ln\left(\frac{1}{\sigma \sqrt{2\pi}}\right) - \frac{(x-\mu)^2}{2\sigma^2} \).

For all measurements, the log-likelihood becomes

\[ \ln(L) = \ell = -\frac{N}{2} \ln(2\pi) - \frac{N}{2} \ln(\sigma^2) - \sum_i (x_i - \mu)^2 \frac{1}{2\sigma^2}, \]

The maximum likelihood parameters also maximize the log-likelihood [2]. The log-likelihood is maximized when \( \sum_i (x_i - \mu)^2 \) is minimized.

To obtain the maximum likelihood parameters we can simply take the partial derivative of the log-likelihood like so:

\[ \frac{\partial \ln(L)}{\partial \mu} = 0 = \frac{1}{\sigma^2} \sum_i (x_i - \mu) \]

This gives the simple expression for our most likely estimate of \( \mu \) which is \( \frac{1}{N} \sum_i x_i \). Once this is done the same procedure can be applied for the \( \sigma \) term to get the following equation.

\[ \frac{\partial \ln(L)}{\partial \sigma} = 0 = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_i (x_i - \mu)^2 \]

We can then rearrange to get the most likely estimate of \( \sigma \) which is \( \sigma^2 = \frac{1}{N} \sum_i (x_i - \mu)^2 \), where \( \mu \) is the maximum likelihood estimate of \( \mu \).

B. Ornstein-Uhlenbeck process

An Ornstein-Uhlenbeck (O-U) process satisfies the stochastic differential equation

\[ dx_t = \theta(\mu - x_t)dt + \sigma dW_t \]

where \( \theta, \mu \) and \( \sigma \) are parameters of the process and \( W_t \) is the Wiener process.

\[ W_t \sim N(0, t) \]

From our point of view the Ornstein-Uhlenbeck process has several desirable properties, it is mean-reverting, it can have a long autocorrelation time and periodic structure in the autocorrelation function and its parameters are not determined by the sampling rate of the device.

We simulated an Ornstein-Uhlenbeck process with the Euler-Maruyama method [8] and then plotted its autocorrelation function, as seen in Figure 2. Note the similarities between the the simulated Ornstein-Uhlenbeck process in Figure 2 and the empirical data in Figure 1.

The maximum likelihood estimates for the Ornstein-Uhlenbeck process provides useful descriptions of the noise and its behaviour. The \( \mu \) is the ‘true’ position of the device which we seek to find and which the process mean-reverts towards, the \( \theta \) determines the autocorrelation function with the autocorrelation time \( \propto e^{-\theta t} \) where \( dt \) the sampling period of the GPS device. The \( \sigma \) parameter then determines the noise between each measurement.

The O-U process is a continuous-time analogue of an autoregressive process of order 1.

1) Maximum Likelihood Estimation of O-U Process: The log likelihood function of an Ornstein-Uhlenbeck process is given below,

\[ \ln(L) = \ln(p(X_0)) - \frac{(N-1)}{2} \ln(2\pi) - \frac{(N-1)}{2} \ln(\delta^2) - \sum_{i=2}^{N} \frac{(x_i - \bar{x})^2}{2\delta^2}, \]

where \( \delta^2 \) and \( \bar{x} \) are defined like so,

\[ \delta^2 = \frac{\sigma^2(1-e^{-2\theta t})}{2\theta} \quad \text{and} \quad \bar{x} = X_{t-1}e^{-\theta t} + \mu(1-e^{-\theta t}) \]

Ignoring the first component, the parameters that maximize the log likelihood can be obtained explicitly by

\[ \hat{\theta} = -\delta^{-1} \log(\beta_1), \quad \hat{\mu} = \beta_2 \quad \text{and} \quad \hat{\sigma}^2 = \frac{2\beta_3}{\delta^{-1}} \]

where

\[ \beta_1 = \sum_{i=1}^{n} X_i X_{i-1} - n^{-1} \sum_{i=1}^{n} X_i \]

\[ \beta_2 = \sum_{i=1}^{n} X_i^2 - n^{-1} \left( \sum_{i=1}^{n} X_i \right)^2 \]

\[ \beta_3 = \sum_{i=1}^{n} X_i X_{i-1} - n^{-1} \left( \sum_{i=1}^{n} X_i \right)^2 \]

Fig. 2: Autocorrelation of an Ornstein-Uhlenbeck process showing similar structure to the GPS position errors.
Technique for parameter estimation for M, p autoregressive processes of comparable order v The simplest processes tend to be considerably more difficult than for processes not observable parameter estimation for moving average Gaussian process v Unfortunately because the lagged terms and so a moving average process of order zero reduces to

\[ E. Moving average processes \]

\[ \beta_2 = \frac{n^{-1} \sum_{i=1}^{n} (X_i - \beta_1 X_{i-1})}{1 - \beta_1} \]

\[ \beta_3 = n^{-1} \sum_{i=1}^{n} (X_i - \beta_1 X_{i-1} - \beta_2 (1 - \beta_1))^2 \]

C. Autoregressive processes

A Gaussian autoregressive process of order p (AR(p)) is defined as,

\[ X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \epsilon_t \]

where \( \epsilon_t \sim N(0, \sigma^2) \). The process therefore has \( p + 2 \) parameters. Due to the dependence on the previous measurement the process has an autocorrelation function that exhibits exponential decay and, like the O-U process, can exhibit periodic behaviour[4].

Because the autoregressive process is linear and only dependent on the previous \( p \) measurements, a system of linear equations can be easily expressed as a matrix, which can be inverted to give the parameters of the AR processes. This process, known as solving the Yule-Walker equations, is a computationally efficient method of parameter estimation.

D. Maximum likelihood for autoregressive processes

The exact maximum likelihood function for the autoregressive process has some slight complications. However if the maximum likelihood function is conditioned on the first \( p \) measurements the conditional likelihood is much easier to calculate.

The conditional likelihood of \( X_t \) given measurements \( X_{t-1}, \ldots, X_{t-p} \) is given by,

\[ f(X_t | X_{t-1}, \ldots, X_{t-p}, \theta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^{p} (X_i - \mu - \sum_{i=1}^{p} \phi_i X_{i-1})^2 \right\} \]

Therefore the conditional log likelihood of the set of data, ignoring measurements 1, 2, \ldots, \( p \) is given by:

\[ \ln(L(\theta)) = \frac{-(N - p)}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=p+1}^{N} (X_i - \mu - \sum_{i=p}^{N} \phi_i X_{i-1})^2 \]

E. Moving average processes

A gaussian moving average process of order q (MA(q)) is

\[ X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q} \]

where \( \epsilon_t \sim N(0, \sigma^2) \). Thus the process has \( q + 2 \) parameters, and so a moving-average process of order zero reduces to a Gaussian process. Unfortunately because the lagged terms are not observable parameter estimation for moving average processes tends to be considerably more difficult than for autoregressive processes of comparable order. The simplest technique for parameter estimation for MA(q) processes is the Kalman filter[6]. While an impressive use of the Kalman filter, this approach is unfortunately very slow relative to the solution of the Yule-Walker equations seen in the autoregressive process.

It should be pointed out that for a moving average process of order \( q \) the autocorrelation function is zero after a lag of \( qdt \), where \( dt \) is the sampling period.

Therefore a pure moving average process would require a cumbersome large order to explain the autocorrelation functions that were observed in the data. However a combined autoregressive moving average process would deal with this problem much more elegantly.

F. ARMA processes

The ARMA process is a combination of autoregressive and moving-average, defined as

\[ X_t = \mu + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q} \]

As the ARMA(p,q) process is just the linear combination of an autoregressive process of order \( p \) with a moving average process of order \( q \) the process predictably takes on the best and worst properties of both processes. Due to the autoregressive terms an ARMA process is able to have a non-zero autocorrelation time after lag \( pdt \) or \( qdt \).

However, as it has the unobservable terms from the moving average process, we also need to run a Kalman filter through the data in order to maximise the likelihood function and so the estimation time is considerably longer than the autoregressive process alone.

III. Experiment

To determine the most appropriate model for GPS noise, data was collected from an array of GPS units positioned at a stationary point for an extended period of time. To ensure that it was the errors in the GPS signal and not errors related to the device, a commercial GPS placed using a different antenna was also used at each test site. Four tag-styiee GPS units were used in conjunction with a commercially available GPS unit with a different antenna.

One of the factors that cause degradation of GPS accuracy is the geometric distribution of satellites. For GPS to function effectively the satellites in the sky should be widely distributed rather than all satellites arranged in one section of the sky. This is why the altitude errors in GPS measurements are typically much larger than the errors in either the latitude or longitude directions [7]. As GPS works on line of sight the satellites being used are all above the unit, whereas for longitude, for example, there could easily be a broad arrangement of satellites to the east and west of the unit. However, given that the GPS and GALLILEO constellations both operate at and inclination of around 55° (55° for GPS and 56° for GALLILEO), i.e. at a latitude of more than 56° (north or south) there are only satellites towards the equator from your position.

Our initial measurements were made from the University of Otago, which is at a latitude of approximately 45.52° South.
To ensure that our noise model was not biased by an imperfect satellite arrangement measurements were made at latitudes of 27.25°S (North Stradbroke Island), 16.30°S (Port Douglas) and 10.35°S (Thursday Island). Each measurement period was for approximately 48 hours. Measurements were made with one GlobalSat BU-353S4 GPS unit on the L1 frequency, sampling once a second and with four tag-style GPS units with lower-quality aerials, also using the L1 frequency, sampling at an average of 18 seconds per fix. However the tag-style GPS units take fixes from a warm start instead of a hot start and so while the average sampling period is around 18 seconds it can be considerably longer. To deal with this the data used for calculation was taken from the measured data at 5 minute intervals. Therefore any deviation in sampling period would be small in comparison to the sampling period.

IV. RESULTS

Figure 3 shows the horizontal errors found at all four locations where measurements were made.

![Fig. 3: Horizontal positions from Dundedin (a), North Stradbroke Island (b), Port Douglas (c) and Thursday Island (d).](image)

Note how the position outliers tend to go on ‘walks’, where the current position estimate does not immediately go back towards the mean, but instead stays considerably removed from the ‘true’ position.

The following tables are laid out with the AIC values for the respective positions (latitude, longitude, altitude) for the data taken from the various sites. Keeping in mind that the model with the lowest AIC value is the best fit to the data. The data taken from the GlobalSat BU-S353 GPS unit with the higher quality aerial is titled ‘GlobalSat’ while the data gathered from the tag-style GPS units with lower quality aerials is title ‘Tags’ for simplicity. The top acronyms denote the noise model being dealt with, i.e. i.i.d Gaussian: Gauss, Ornstein-Uhlenbeck process: O-U, autoregressive process of order $p$: AR($p$), moving average process of order $q$: MA($q$) and mixed autoregressive moving average process of order $p, q$: ARMA($p,q$). Occasionally the statistics package was unable to model either the moving average (MA) or mixed autoregressive moving average (ARMA) processes due to the moving average parameters not being invertible. In these cases we have given the highest order processes available and denoted non-invertible processes as DNC.

1) North Stradbroke Island (27.25°S) data set:

<table>
<thead>
<tr>
<th></th>
<th>GlobalSat</th>
<th>Gauss</th>
<th>O-U</th>
<th>AR(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat</td>
<td>61099.42</td>
<td>4099.02</td>
<td>846880.76</td>
<td></td>
</tr>
<tr>
<td>Lon</td>
<td>584039.85</td>
<td>-57217.30</td>
<td>807603.39</td>
<td></td>
</tr>
<tr>
<td>Alt</td>
<td>991468.56</td>
<td>401359.30</td>
<td>-638245.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MA(1)</td>
<td>MA(3)</td>
<td>ARMA(3,3)</td>
<td></td>
</tr>
<tr>
<td>Lat</td>
<td>417771.97</td>
<td>160424.77</td>
<td>27297.91</td>
<td></td>
</tr>
<tr>
<td>Lon</td>
<td>369863.67</td>
<td>67178.09</td>
<td>DNC</td>
<td></td>
</tr>
<tr>
<td>Alt</td>
<td>783784.45</td>
<td>549365.25</td>
<td>272510.80</td>
<td></td>
</tr>
</tbody>
</table>

Nort Stradbroke Island data set from GPS units with lower quality aerials. Note that for brevity the tag AIC values have been taken and averaged for the given data sets.

<table>
<thead>
<tr>
<th></th>
<th>Tags</th>
<th>Gauss</th>
<th>O-U</th>
<th>AR(9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat</td>
<td>4524.69</td>
<td>4454.52</td>
<td>4343.39</td>
<td></td>
</tr>
<tr>
<td>Lon</td>
<td>4337.74</td>
<td>4295.58</td>
<td>4225.97</td>
<td></td>
</tr>
<tr>
<td>Alt</td>
<td>5085.63</td>
<td>4295.58</td>
<td>4225.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MA(1)</td>
<td>MA(3)</td>
<td>ARMA(3,3)</td>
<td></td>
</tr>
<tr>
<td>Lat</td>
<td>4487.33</td>
<td>4426.07</td>
<td>4342.59</td>
<td></td>
</tr>
<tr>
<td>Lon</td>
<td>4312.76</td>
<td>4292.28</td>
<td>4271.80</td>
<td></td>
</tr>
<tr>
<td>Alt</td>
<td>5062.48</td>
<td>5045.75</td>
<td>5025.82</td>
<td></td>
</tr>
</tbody>
</table>

2) Port Douglas (16.30°S) data set: Note that the higher order MA and ARMA models were not invertible for the GlobalSat GPS unit and so the highest terms were 1 and 1, respectively.
3) Thursday Island (10.35°S) data set: Unfortunately power was lost for around three minutes and so the data set for the GlobalSat GPS unit from Thursday Island has been split into two. The tag-style GPS units appear to have retained enough charge in their capacitors for this short loss of power to not affect them. This is the first data set for the GlobalSat GPS unit.

And the second data set from Thursday Island. As in the data set from Port Douglas the parameters for higher order MA and ARMA processes were not invertible and so the highest order terms obtained were 1 and 1,1, respectively.

Data set from low-quality aerial GPS units. Note that one of the tags was damaged in transport and so the data set from Thursday Island is averaged over three GPS units, not four as is the case for the other data sets.

V. Analysis

It is clear from Figure 4 that the long-term autocorrelation times observed in Dunedin are not a result of the poor satellite arrangement at larger latitudes. Of these the autocorrelation time of the data gathered at Thursday Island has a slightly different pattern to those observed at the other locations. This could either be because of a temporary loss of power experienced at Thursday Island which essentially forced the data to be split into two sets of data of around 24 hours each.

What we see in the data is that there is a general trend that appears across the regions where the noise model with the highest AIC (and so the worst fit to the data) is the i.i.d Gaussian. After that is typically a lower-order moving average process, followed by the Ornstein-Uhlenbeck model and then either the higher order autoregressive models or the mixed autoregressive moving average model.

It is worth noting that getting the parameters that maximise the likelihoods for the Ornstein-Uhlenbeck and autoregressive models is a trivial matter where either the analytical solution is known or it is simply a case of inverting a matrix which can be done numerically and very quickly. Conversely estimating the parameters for the moving average and mixed autoregressive moving average models is considerably more involved and typically requires running a Kalman filter over the data set with a given set of parameters, calculating the log likelihood of that particular case and then doing the same thing again with a slightly different set of parameters until the log likelihood is maximised. This approach is much more computationally expensive. Also if the process is non-stationary then the moving average parameters are not invertible and further measures must be taken to solve for the MLE parameters. This is why the autoregressive models are taken up to ninth order whereas, at most the MA and ARMA models are only taken up to third order and even then we experienced invertibility problems.

VI. PROPOSED NOISE MODELS

From our analysis of the data we have seen that the autoregressive processes are considerably more likely for the more accurate device and the estimation of the parameters is far more computationally efficient.

The higher order AR processes have lower AIC values, indicating that, from the point of view of modelling the data most precisely it is better to have a higher order AR process as the likelihood maximization outweighs the increase in complexity due to the extra terms.

However for the less accurate GPS units the higher order AR processes are only slightly more likely than the AR(1)
If using the AR(1) process we would advise using the discretized form of the Ornstein-Uhlenbeck process. This has several advantages over simply estimating the autoregressive parameter $\phi$. For one, the sampling period of the device is explicit in the formulation of the Ornstein-Uhlenbeck process and so for cases when there is a nonconstant sampling period this is clearly more useful. Also the other parameters of the O-U process give clear insight into the characteristics of the model. The $\theta$ parameter describes the rate at which random measurements will revert towards the mean $\mu$ while the $\sigma$ parameter indicates the amount of noise present between each measurement.

VII. INCORPORATION INTO STATE ESTIMATION

As mentioned previously the Kalman filter is a versatile algorithm that combines sequential measurements and provides asymptotically optimal estimates of the state while minimizing the state covariance. One of the practically useful aspects of the Kalman filter is that it is recursive and all of the information from the previous measurements is contained in the current state estimate, thus significantly easing the computational load over each iteration.

Unfortunately all of our proposed noise models do require an expansion of the state estimate and an expansion of the propagation matrices within the Kalman filter. Incorporation into an existing state estimate is much simpler for the autoregressive processes (O-U, AR($p$)) than it is for the moving average processes (MA($q$), ARMA($p,q$)).

Consider, for example, an Ornstein-Uhlenbeck noise model of position $x$ with O-U parameters $\theta, \sigma$ and $\mu$. The true position is $\mu$ and so we wish to estimate it. The measurement variance is simply $\sigma^2$ and the state propagation $F(k)$ and measurement matrices $H(k)$ are given below. It is also necessary to include the current measurement $z(k)$ into the state estimate $\hat{x}(k)$. To construct a functioning Kalman filter

the system noise covariance matrix $Q(k)$ must, likewise, be defined. However in the case of a stationary system there is no system noise and so the system noise covariance is simply zero. There exists some computational reasons for not having the matrix $Q(k)$ being exactly zero, however setting $Q(k)$ to some small value, e.g. $10^{-6}$, deals with these satisfactorily.

$$
\hat{x}(k) = \begin{bmatrix} \hat{z}(k) \\ \hat{\theta}(k) \end{bmatrix}, \quad R(k) = \begin{bmatrix} \sigma^2 \\ 0 \end{bmatrix}, \quad Q(k) \approx [0],
$$

$$
F(k) = \begin{bmatrix} 1 & \theta \\ 0 & \exp(-\theta dt) \end{bmatrix}, \quad H(k) = [\theta \exp(-\theta dt)],
$$

Figure 5 shows the result of using an Ornstein-Uhlenbeck noise model with a Kalman filter to produce an estimate of the latitude from empirical data with the parameters derived from maximum likelihood estimation.

VIII. CONCLUSION

The use of independent Gaussian noise models is very common [6]. However we have shown that for GPS position measurements there are other models that fit the data better. If a Gaussian model is used, it will have to have an artificially large variance to account for the outliers, and will still not account for the observed autocorrelations. We have found that higher order autoregressive noise processes provide a far better fit over the observed data. Moving average and mixed autoregressive moving average models also provide a lower AIC value, however the calculation of the specific parameters is somewhat more involved and incurs a considerably higher computational expense.

REFERENCES


Appendix C

Base Station Shield Designs

This Appendix includes the circuit schematic and PCB layout diagrams for the base station shield.
C.1 Base Station Shield Schematic

Figure C.1: Schematic diagram of base station shield board.
C.2 Base Station Shield PCB Layout

Figure C.2: Base station shield board PCB layout. Showing from top to bottom: top silkscreen layer, top layer, bottom layer, bottom silkscreen. Circuit board measures 56x66mm. Diagram not to scale.
Figure C.3: Base station shield board PCB layout: top layer. Clearly showing the separation between the power supply and telemetry areas, outlined by different ground planes. Circuit board measures 56x66mm. Diagram not to scale.
Figure C.4: Base station shield board PCB layout: bottom layer. The bottom layer consists of signal tracks as well as power supply and telemetry ground planes. Circuit board measures 56x66mm. Diagram not to scale.
Appendix D

Position Sensor Telemetry Module Designs

This Appendix includes the circuit schematic and PCB layout diagrams for the telemetry module, including both the internal and external antenna versions.
D.1 Telemetry Module - Internal Antenna Schematic

Figure D.1: Schematic diagram of telemetry module board with an internal antenna.
D.2 Telemetry Module - Internal Antenna Board Layout

Figure D.2: Telemetry module board (internal antenna) PCB layout. Showing from top to bottom: top silkscreen layer, top layer, bottom layer, bottom silkscreen. Circuit board measures 20x44mm. Diagram not to scale.
Figure D.3: Telemetry module board (internal antenna) PCB layout: Top layer. The top layer consists of signal tracks and the shared ground plane. Circuit board measures 20x44mm. Diagram not to scale.
Figure D.4: Telemetry module board (internal antenna) PCB layout: Bottom layer. Clearly showing the separation between the solar charging and telemetry areas. Circuit board measures 20x44mm. Diagram not to scale.
D.3 Telemetry Module - External Antenna Schematic

Figure D.5: Schematic diagram of telemetry module board with an external antenna.
D.4 Telemetry Module - External Antenna Board Layout

Figure D.6: Telemetry module board (external antenna) PCB layout. Showing from top to bottom: top silkscreen layer, top layer, bottom layer, bottom silkscreen. Circuit board measures 18x44mm. Diagram not to scale.
Figure D.7: Telemetry module board (external antenna) PCB layout: Top layer. The top layer consists of signal tracks and the shared ground plane. Circuit board measures 18x44mm. Diagram not to scale.
Figure D.8: Telemetry module board (external antenna) PCB layout: Bottom layer. Clearly showing the separation between the solar charging and telemetry areas. Circuit board measures 20x44mm. Diagram not to scale.
Appendix E

Position Sensor Software Overview

The pages in this Appendix include the software overview for the position sensor.
Here is a list of all files:

- backup_registers.h
- config.h
- fix.cpp
- fix.h
- FlashList.h
- gizmo_debug.cpp
- gizmo_debug.h
- gizmo_led.cpp
- gizmo_led.h
- gizmo_math.cpp
- gizmo_math.h
- gizmo_string.cpp
- gizmo_string.h
- gizmo_string_tb.cpp
- gizmo_test.cpp
- gizmo_test.h
- gizmo_time.cpp
- gizmo_time.h
- gizmo_usart.cpp
- gizmo_usart.h
- gizmo_util.cpp
- gizmo_util.h
- gps_time.cpp
- gps_time.h
- gpshandler.cpp
- gpshandler.h
- hardware_def.h
- NMEA_parser.cpp
- NMEA_parser.h
- packet_formats.cpp
- pichandler.cpp
- pichandler.h
- queue.cpp
- queue.h
- srd_register_config.h
- srd_registers.h
- srd_util.cpp
- srdhandler.cpp
- srdhandler.h
- state_transition.cpp
- state_transition.h
- tag.cpp
- tag.h
File Documentation

0.1 backup_registers.h File Reference

Include dependency graph for backup_registers.h:

```
backup_registers.h
  libopencm3/stm32/f1/bkp.h
```

Macros

- #define state BKP_DR1
- #define bkp_queue_front BKP_DR2
- #define bkp_basestation_address BKP_DR3
- #define bkp_next_fix_msb BKP_DR4
- #define bkp_next_fix_lsb BKP_DR5
- #define bkp_general_broadcast_msb BKP_DR6
- #define bkp_general_broadcast_lsb BKP_DR7
- #define bkp_voltage_abort_count BKP_DR8
- #define gps_fix_failures BKP_DR9

0.2 config.h File Reference

Macros

- #define ISM 121
- #define ISMCOW 122
- #define HDOP_SLEEP_TIME 1
- #define GPS_FIX_WAIT_TIME 90
- #define GPS_WARM_START 0
- #define BASESTATION_TEST_MODE 0
- #define BASESTATION_GPS_TEST_MODE 0
- #define BASESTATION_SRD_TX_TEST_MODE 0
- #define BASESTATION_SRD_RX_TEST_MODE 0
- #define SHELL_MODE_TIMEOUT 1
- #define DEBUG 1
- #define DEBUG_LOG 1
- #define FAKE_FIXES 0
- #define ERASE_QUEUE_ON_FULL 1
- #define ERASE_QUEUE_ON_STARTUP 0
- #define TAG_CONFIG_PAGE_NUM 30
- #define FIX_QUEUE_FIRST_PAGE 31
- #define FIX_QUEUE_LAST_PAGE 110
- #define DEBUG_LOG_FIRST_PAGE 111


- `#define DEBUG_LOG_LAST_PAGE 127`
- `#define FLASH_MAIN_MEMORY_BASE 0x08000000UL`
- `#define FLASH_PAGE_SIZE 0x400UL`
- `#define ERASED_FLASH 0xFFFFFFFF`
- `#define TAG_CONFIG_PAGE_ADDR (FLASH_MAIN_MEMORY_BASE + TAG_CONFIG_PAGE_NUM*FLASH_PAGE_SIZE) /* 0x08009800 */`

### 0.3 fix.cpp File Reference

Include dependency graph for fix.cpp:

![Dependency Graph for fix.cpp](dependency_graph_fix.cpp)

### 0.4 fix.h File Reference

Include dependency graph for fix.h:

![Dependency Graph for fix.h](dependency_graph_fix.h)
Classes

- class Fix

Macros

- #define FIX_TYPE_3D 3

0.5 FlashList.h File Reference

Include dependency graph for FlashList.h:

Classes

- class FlashList<T>

Macros

- #define FLASH_BASE_ADDRESS 0x08000000UL
- #define ELEMENT_FLAG 0xDEADBEEF
- #define FLASH_WORD_SIZE 4

0.6 gizmo_debug.cpp File Reference

Classes

- class LogEntry
- class LogFlashList

Functions

- void print_debug_log ()
- uint32_t get_debug_entry_rtc (uint32_t index)
- char * get_debug_entry_str (uint32_t index)
- int count_debug_log ()
- void initialize_debug_log ()
- void endl ()
- void tag_debug_print_nocr (const char *str)
- void tag_debug_print (const char *str)
- void tag_debug_print_num (uint32_t num)
• void tag_debug_print (const char *str, const uint32_t num)
• void tag_debug_print (const char *str, const int32_t num)
• void tag_print (const char *str, const uint32_t num)
• void tag_print (const char *str, const int32_t num)
• void tag_debug_print (char c)
• void tag_debug_print_fix (const Fix *pos)
• void tag_print_nocr (const char *str)
• void tag_print_num (uint32_t x)
• void tag_print (const char *str)
• void tag_print_fix (const Fix *pos)
• void tag_print_fix (const Fix &pos)
• void tag_debug_print_fix (const Fix &pos)

Variables

• LogFlashList flash_debug_log

0.7  gizmo_debug.h File Reference

Include dependency graph for gizmo_debug.h:

```
+-----------------------------+
| gizmo_debug.h               |
|                             |
| ./config.h                  |
| ./fix.h                     |
| ./gizmo_string.h            |
| ./gizmo_math.h              |
| libopencm3/cm3/common.h     |
+-----------------------------+
```

Functions

• void endl ()
• void initialize_debug_log ()
• void print_debug_log ()
• uint32_t get_debug_entry_rtc (uint32_t index)
• char * get_debug_entry_str (uint32_t index)
• int count_debug_log ()
• void tag_debug_print (const char *str)
• void tag_debug_print_nocr (const char *str)
• void tag_debug_print_num (uint32_t num)
• void tag_debug_print_fix (const Fix *pos)
• void tag_debug_print_fix (const Fix &pos)
• void tag_debug_print (const char *str, const int32_t num)
• void tag_debug_print (char c)
• void tag_print_nocr (const char *str)
• void tag_print_num (uint32_t x)
• void tag_print (const char *str)
• void tag_print (const char *str, const int32_t num)
• void tag_print_fix (const Fix *pos)
• void tag_print_fix (const Fix &pos)

0.8 gizmo_led.cpp File Reference

Include dependency graph for gizmo_led.cpp:

Functions

• void tag_led_setup ()
• void tag_led_on (void)
• void tag_led_off (void)
• void tag_led_toggle (void)
• void tag_led_flash (const int &secs)

0.9 gizmo_led.h File Reference

Functions

• void tag_led_setup (void)
• void tag_led_on (void)
• void tag_led_off (void)
• void tag_led_toggle (void)
• void tag_led_flash (const int &secs)
0.10  gizmo_math.cpp File Reference

Include dependency graph for gizmo_math.cpp:

```
gizmo_math.cpp
./gizmo_math.h
```

Functions

- int gizmo_math_pow (int x, int y)

0.11  gizmo_math.h File Reference

Functions

- int gizmo_math_pow (int x, int y)

0.12  gizmo_string.cpp File Reference

Include dependency graph for gizmo_string.cpp:

```
gizmo_string.cpp
./gizmo_util.h
./gizmo_string.h
libopencm3/cm3/common.h ./gizmo_math.h
```

```
libopencm3/cm3/common.h
```

```
./gizmo_math.h
```

```
./gizmo_math.h
```

```
Functions

- int32_t char_to_int (const char c)
- gizmo_string uint32_t_to_string (uint32_t num)
- gizmo_string uint32_t_to_hex (const uint32_t &num)
- gizmo_string int32_t_to_string (int32_t num)

0.13 gizmo_string.h File Reference

Include dependency graph for gizmo_string.h:

![Include dependency graph]

Classes

- class gizmo_string_base< N >

Macros

- #define DEBUG_ASSERT(x, y, z) {}
0.14  gizmo_string_tb.cpp File Reference

Include dependency graph for gizmo_string_tb.cpp:

Macros

• #define DEBUG_ASSERT()
• #define ASSERT(fun, val)

Functions

• int main ()

0.15  gizmo_test.cpp File Reference

Macros

• #define GPS_TEST_TIME 8*10

Functions

• void tag_test_gps ()
• void fake_fix ()

0.16  gizmo_test.h File Reference

Functions

• void fake_fix ()
• void tag_test_gps (void)
• void tag_test (int tag_test_counter)
Macros

- `#define SEC_PER_DAY 86400`
- `#define SEC_PER_HOUR 3600`
Classes

• class gizmo_time

0.19  gizmo_usart.cpp File Reference

Include dependency graph for gizmo_usart.cpp:

Functions

• void usart_write_char (uint32_t usart, uint8_t byte)

• void shell_interrupt_enable (void)

• void shell_interrupt_disable (void)

• void usart_put_string (uint32_t usart, const char *str)

• void usart_put_string (uint32_t usart, const int32_t &num)

• void usart_put_string (uint32_t usart, const uint32_t &num)

• gizmo_string usart_recv_line (uint32_t usart)

• void gizmo_usart_set_baudrate (uint32_t usart, uint32_t baud)
0.20  gizmo_usart.h File Reference

Include dependency graph for gizmo_usart.h:

```
Include dependency graph for gizmo_usart.h:

```

Functions

- void usart_write_char (uint32_t usart, uint8_t byte)
- void usart_put_string (uint32_t usart, const char *str)
- void usart_put_string (uint32_t usart, const uint32_t &num)
- void usart_put_string (uint32_t usart, const int32_t &num)
- gizmo_string usart_recv_line (uint32_t usart)
- void gizmo_usart_set_baudrate (uint32_t usart, uint32_t baud)
- void shell_interrupt_enable (void)
- void shell_interrupt_disable (void)

0.21  gizmo_util.cpp File Reference

Classes

- struct tag_config_s

Typedefs

- typedef struct tag_config_s tag_config

Functions

- void tag_setup_system_clock (void)
- void wait_aux (uint32_t systick_reload)
- void tag_wait_ms (int time_ms)
- void tag_wait (int time_sec)
- void tag_wait_us (int time_us)
- void tag_flashing_wait (int time)
- gizmo_string tag_get_time ()
- void alarm_disable ()
- void set_alarm_time (int absolute_time)
- void set_alarm_delay (int delay_seconds)
- void tag_config_set_defaults (void)
- void tag_config_copy (void)
- void tag_config_write (void)
• uint32_t get_gps_fix_interval ()
  void set_gps_fix_interval (uint32_t t)
• uint32_t get_tag_id (void)
• uint16_t get_gps_battery_threshold (void)
  void set_gps_battery_threshold (uint16_t t)
• uint16_t get_gps_battery_threshold_x2 (void)
  void set_gps_battery_threshold_x2 (uint16_t t)
• uint16_t get_gps_battery_threshold_x4 (void)
  void set_gps_battery_threshold_x4 (uint16_t t)
• uint16_t get_gps_battery_threshold_x8 (void)
  void set_gps_battery_threshold_x8 (uint16_t t)
• uint16_t get_gps_battery_threshold_x16 (void)
  void set_gps_battery_threshold_x16 (uint16_t t)
• uint16_t get_gps_battery_threshold_x32 (void)
  void set_gps_battery_threshold_x32 (uint16_t t)
• uint16_t get_telemetry_battery_threshold (void)
  void set_telemetry_battery_threshold (uint16_t t)
• uint16_t get_radio_sleep_time (void)
  void set_radio_sleep_time (uint16_t t)
• uint8_t get_tag_addr (void)
  void set_tag_addr (uint8_t addr)
• uint8_t get_radio_channel_p0 (void)
  void set_radio_channel_p0 (uint8_t channel)
• uint8_t get_radio_channel_p1 (void)
  void set_radio_channel_p1 (uint8_t channel)
• uint8_t get_radio_channel_p2 (void)
  void set_radio_channel_p2 (uint8_t channel)
• uint16_t get_queue_reset_length (void)
  void set_queue_reset_length (uint16_t length)
• uint16_t get_queue_unsent_threshold (void)
  void set_queue_unsent_threshold (uint16_t threshold)
• int get_next_fix_time ()
  void tag_enter_sleep_mode (void)
  void tag_enter_stop_mode (void)
• bool set_bkp_general_broadcast (void)
• bool tag_enter_standby_mode (uint32_t v)
• bool gpio_check (uint32_t gpioport, uint16_t gpio)
  void gpio_config_output (uint32_t gpioport, uint16_t gpio)
• void gpio_config_input_float (uint32_t gpioport, uint16_t gpio)
• void gpio_config_input_pullup (uint32_t gpioport, uint16_t gpio)
• void gpio_config_input_pulldown (uint32_t gpioport, uint16_t gpio)
• void tag_adc_setup (void)
• uint32_t tag_get_battery_voltage ()

Variables
• uint32_t SYSTICK_RELOAD_MS
• volatile int wait_bsy
• tag_config ram_config
• bool tag_config_changed
• uint32_t batt_v
Include dependency graph for gizmo_util.h:

Classes

- class gizmo

Functions

- void tag_setup_system_clock (void)
- void tag_wait (int time)
- void tag_wait_ms (int time_ms)
- void tag_wait_us (int time_us)
- void tag_flashing_wait (int time)
- gizmo_string tag_get_time ()
- void alarm_disable ()
- void set_alarm_time (int absolute_time)
- void set_alarm_delay (int delay_seconds)
- int get_next_fix_time ()
- void tag_enter_sleep_mode ()
- void tag_enter_stop_mode ()
- void tag_enter_standby_mode (uint32_t v)
- void gpio_config_output (uint32_t gpioport, uint16_t gpio)
- void gpio_config_input_float (uint32_t gpioport, uint16_t gpio)
- void gpio_config_input_pullup (uint32_t gpioport, uint16_t gpio)
- void gpio_config_input_pulldown (uint32_t gpioport, uint16_t gpio)
- bool gpio_check (uint32_t gpioport, uint16_t gpio)
- void tag_adc_setup (void)
- uint32_t tag_get_battery_voltage ()
- void set_bkp_general_broadcast (void)
- bool get_bkp_general_broadcast (void)
- void tag_config_set_defaults (void)
- void tag_config_copy (void)
- void tag_config_write (void)
- uint8_t get_tag_receive_broadcast (void)
- void set_tag_receive_broadcast (uint8_t broadcast)
• uint32_t get_gps_fix_interval()
• void set_gps_fix_interval(uint32_t t)
• uint16_t get_gps_battery_threshold(void)
• void set_gps_battery_threshold(uint16_t t)
• uint16_t get_gps_battery_threshold_x2(void)
• void set_gps_battery_threshold_x2(uint16_t t)
• uint16_t get_gps_battery_threshold_x4(void)
• void set_gps_battery_threshold_x4(uint16_t t)
• uint16_t get_gps_battery_threshold_x8(void)
• void set_gps_battery_threshold_x8(uint16_t t)
• uint16_t get_gps_battery_threshold_x16(void)
• void set_gps_battery_threshold_x16(uint16_t t)
• uint16_t get_gps_battery_threshold_x32(void)
• void set_gps_battery_threshold_x32(uint16_t t)
• uint32_t get_tag_id(void)
• void set_tag_id(uint32_t t)
• uint16_t get_telemetry_battery_threshold(void)
• void set_telemetry_battery_threshold(uint16_t t)
• uint16_t get_radio_sleep_time(void)
• void set_radio_sleep_time(uint16_t t)
• uint8_t get_tag_addr(void)
• void set_tag_addr(uint8_t addr)
• uint8_t get_radio_channel_p0(void)
• void set_radio_channel_p0(uint8_t channel)
• uint8_t get_radio_channel_p1(void)
• void set_radio_channel_p1(uint8_t channel)
• uint8_t get_radio_channel_p2(void)
• void set_radio_channel_p2(uint8_t channel)
• uint16_t get_queue_reset_length(void)
• void set_queue_reset_length(uint16_t length)
• uint16_t get_queue_unsent_threshold(void)
• void set_queue_unsent_threshold(uint16_t threshold)

0.23  gps_time.cpp File Reference

Include dependency graph for gps_time.cpp:
0.24  gps_time.h File Reference

Include dependency graph for gps_time.h:

```
gps_time.h
gps_time.h
libopencm3/cm3/common.h
```

Classes

- class gps_time

0.25  gpshandler.cpp File Reference

Macros

- #define GPS_DEBUG 1
- #define GPS_DEBUG_PRINT(x) tag_debug_print(x)

Functions

- void gps_clock_setup ()
- void gps_usart_setup ()
- void gps_gpio_pulldown ()
- void gps_gpio_setup ()
- void gps_exti_enable ()
- void gps_exti_disable ()
- char gps_usart_recv (uint32_t timeout)
- void gps_USART_enable ()
- void gps_USART_disable ()
- void gps_exti_setup ()
- void gps_nvic_setup ()
- void gps_setup ()
- bool gps_get_system_on ()
- void gps_power_pulse ()
- int gps_power_on ()
- void gps_power_off ()
- int gps_get_fix (Fix ∗out)
- bool gps_has_a_fix ()
- void gps_init_warm_start (const Fix &fix)

Variables

- uint32_t fix_start_time
0.26 gpshandler.h File Reference

Include dependency graph for gpshandler.h:

Macros

- `#define SUCCESS 0`
- `#define FAIL 1`

Functions

- `void gps_clock_setup ()`
- `void gps_usart_setup ()`
- `void gps_gpio_setup ()`
- `void gps_exti_setup ()`
- `void gps_exti_enable ()`
- `void gps_exti_disable ()`
- `void gps_nvic_setup ()`
- `void gps_setup ()`
- `void gps_usart_enable ()`
- `void gps_usart_disable ()`
- `int gps_power_on ()`
- `void gps_power_off ()`
- `void gps_init_warm_start (const Fix &fix)`
- `int gps_get_fix (Fix *out)`
- `bool gps_has_a_fix ()`
Include dependency graph for hardware_def.h:

```
Hardware_def.h
  |- libopencm3/stm32/gpio.h
  |   |- ./config.h
```

Macros

- `#define SRD_ON_OFF GPIO11` /* Main Function */
- `#define SRD_ON_OFF_PORT GPIOA`
- `#define SRD_CS GPIO12`
- `#define SRD_CS_PORT GPIOA`
- `#define SRD_GPIO0 GPIO0`
- `#define SRD_GPIO0_PORT GPIOA`
- `#define SRD_GPIO0_EXTI EXTI0`
- `#define SRD_GPIO0_IRQ NVIC EXTI0_IRQ`
- `#define SRD_GPIO2 GPIO9`
- `#define SRD_GPIO2_PORT GPIOA`
- `#define SRD_GPIO2_EXTI EXTI9`
- `#define SRD_GPIO2_IRQ NVIC EXTI9_5_IRQ`
- `#define SRD_GPIO3_GPIO10`
- `#define SRD_GPIO3_PORT GPIOA`
- `#define SRD_GPIO3_EXTI EXTI10`
- `#define SRD_GPIO3_IRQ NVIC EXTI15_10_IRQ`
- `#define SRD_RESET_GPIO8` /* WAS B3 Alternate REMAP */
- `#define SRD_RESET_PORT GPIOA`
- `#define BAT_SENSE_GPIO1`
- `#define BAT_SENSE_PORT GPIOB`
- `#define BAT_SENSE_CNFG GPIO CNFG_INPUT_ANALOG`
- `#define BAT_TEST_EN GPIO0`
- `#define BAT_TEST_EN_PORT GPIOB`
- `#define BAT_TEST_EN_CNFG GPIO CNFG_OUTPUT_PUSH_PULL`
- `#define GPS ON OFF GPIO2` /* Main Function PB2/BOOT1 */
- `#define GPS ON OFF PORT GPIOB`
- `#define GPS ON OFF CNFG GPIO CNFG_OUTPUT_PUSH_PULL`
- `#define GPS NAV GPIO4` /* Main Function */
- `#define GPS NAV_PORT GPIOA`
- `#define GPS NAV EXTI EXTI4`
- `#define GPS NAV_IRQ NVIC EXTI4_IRQ`
- `#define GPS_TX GPIO2` /* Main Function */
- `#define GPS_TX_PORT GPIOA`
- `#define GPS_TX_USART GPIO_USART2_TX` /* Alternate */
- `#define GPS_TX_CNF GPIO CNFG_OUTPUT_ALTFN_PUSH_PULL`
0.28 NMEA_parser.cpp File Reference

Include dependency graph for NMEA_parser.cpp:

Functions

- int32_t NMEA_to_decimal_lat (const gizmo_string &x)
- int32_t NMEA_to_decimal_long (const gizmo_string &y)
- gizmo_string NMEA_get_column (const gizmo_string &nmea_msg, uint8_t col)
- gizmo_string NMEA_round (const gizmo_string &str_val)
- gizmo_string NMEA_round_10 (const gizmo_string &str_val)
- uint8_t NMEA_to_satellites_used (const gizmo_string &gga)
- gizmo_string NMEA_calculate_checksum (const gizmo_string &msg)
Macros

- `#define GGA_TIME 1`
- `#define GGA_LATITUDE 2`
- `#define GGA_N_S 3`
- `#define GGA_LONGITUDE 4`
- `#define GGA_E_W 5`
- `#define GGA_SAT 7`
- `#define GGA_HDOP 8`
- `#define GGA_ALTITUDE 9`
- `#define RMC_SPEED_K 7`
- `#define RMC_DATE 9`
- `#define GSA_FIX_TYPE_COLUMN 2`
- `#define GSA_PDOP 15`
- `#define GSA_HDOP 16`
- `#define GSA_VDOP 17`

Functions

- `int32_t NMEA_to_decimal_lat (const gizmo_string &x)`
- `int32_t NMEA_to_decimal_long (const gizmo_string &y)`
- `gizmo_string NMEA_get_column (const gizmo_string &nmea_msg, uint8_t col)`
- `gizmo_string NMEA_round (const gizmo_string &str_val)`
- `gizmo_string NMEA_round_10 (const gizmo_string &str_val)`
- `uint8_t NMEA_to_satellites_used (const gizmo_string &gga)`
- `gizmo_string NMEA_calculate_checksum (const gizmo_string &msg)`
0.30 packet_formats.cpp File Reference

Functions

- void sendStatusResponse ()
- void sendGetResponse ()
- void sendNoFixResponse ()
- void sendLogFile ()
- void sendGPSPLoc ()
- void sendConfigConfirmation ()

0.31 pichandler.cpp File Reference

Include dependency graph for pichandler.cpp:

Functions

- void pic_gpio_pulldown ()
- void pic_gpio_setup ()
- void pic_usart_setup ()
- char pic_usart_recv (uint32_t timeout)
- void pic_usart_enable ()
- void pic_usart_disable ()
- void pic_setup ()

0.32 pichandler.h File Reference

Include dependency graph for pichandler.h:
Functions

- void `pic_usart_setup` ()
- void `pic_gpio_setup` ()
- void `pic_setup` ()
- void `pic_usart_enable` ()
- void `pic_usart_disable` ()
- char `pic_usart_recv` (uint32_t timeout)

0.33 queue.cpp File Reference

Include dependency graph for queue.cpp:

Classes

- class `FixQueueElement`
- class `FixFlashList`

Functions

- int `fixQueue_get_unsent` (int index)
- int `get_first_unsent` (void)
- Fix ∗ `fixQueue_get` (int index)
- void `fixQueue_mark_sent` (uint32_t queue_index)
- void `fixQueue_pushback` (Fix ∗element)
- int `fixQueue_number_unsent` ()
- int `fixQueue_number_stored` ()
- void `fixQueue_print_fixes` ()
- void `fixQueue_erase_fixes` ()
- uint32_t `fixQueue_max_length` ()

Variables

- FixFlashList `flash_fix_list`
0.34 queue.h File Reference

Include dependency graph for queue.h:

```
queue.h
       ./fix.h
            ./gizmo_string.h
                 libopencm3/cm3/common.h
                        ./gizmo_math.h
```

Functions

- void fixQueue_mark_sent (uint32_t queue_index)
- Fix * fixQueue_get (int index)
- int fixQueue_get_unsent (int index)
- void fixQueue_pushback (Fix *element)
- int fixQueue_number_unsent ()
- int fixQueue_number_stored ()
- void fixQueue_print_fixes ()
- void fixQueue_erase_fixes ()
- uint32_t fixQueue_max_length ()

0.35 srd_register_config.h File Reference

Classes

- struct registerSetting_t

0.36 srd_registers.h File Reference

Macros

- #define CC112X_IOCFG3 0x0000
- #define CC112X_IOCFG2 0x0001
- #define CC112X_IOCFG1 0x0002
- #define CC112X_IOCFG0 0x0003
- #define CC112X_SYNC3 0x0004
- #define CC112X_SYNC2 0x0005
- #define CC112X_SYNC1 0x0006
- #define CC112X_SYNC0 0x0007
• \#define CC112X_SYNC_CFG1 0x0008
• \#define CC112X_SYNC_CFG0 0x0009
• \#define CC112X_DEVIATION_M 0x000A
• \#define CC112X_MODCFG_DEV_E 0x000B
• \#define CC112X_DCFILT_CFG 0x000C
• \#define CC112X_PREAMBLE_CFG1 0x000D
• \#define CC112X_PREAMBLE_CFG0 0x000E
• \#define CC112X_FREQ_IF_CFG 0x000F
• \#define CC112X_IQIC 0x0010
• \#define CC112X_CHAN_BW 0x0011
• \#define CC112X_MDMCFG1 0x0012
• \#define CC112X_MDMCFG0 0x0013
• \#define CC112X_DRATE2 0x0014
• \#define CC112X_DRATE1 0x0015
• \#define CC112X_DRATE0 0x0016
• \#define CC112X_AGCRF 0x0017
• \#define CC112X_AGCS THR 0x0018
• \#define CC112X_AGCGAIN_ADJUST 0x0019
• \#define CC112X_AGCCFG3 0x001A
• \#define CC112X_AGCGFG2 0x001B
• \#define CC112X_AGCGFG1 0x001C
• \#define CC112X_AGCGFG0 0x001D
• \#define CC112X_FIFO_CFG 0x001E
• \#define CC112X_DEV_ADDR 0x001F
• \#define CC112X_SETTLING_CFG 0x0020
• \#define CC112X_FS_CFG 0x0021
• \#define CC112X_WORCFG1 0x0022
• \#define CC112X_WORCFG0 0x0023
• \#define CC112X_WOREVENT0_MSB 0x0024
• \#define CC112X_WOREVENT0_LSB 0x0025
• \#define CC112X_PKT_CFG2 0x0026
• \#define CC112X_PKTCFG1 0x0027
• \#define CC112X_PKTCFG0 0x0028
• \#define CC112X_RFENDCFG1 0x0029
• \#define CC112X_RFENDCFG0 0x002A
• \#define CC112X_PACFG2 0x002B
• \#define CC112X_PACFG1 0x002C
• \#define CC112X_PACFG0 0x002D
• \#define CC112X_PKTLEN 0x002E
• \#define CC112X_IFMIX_CFG 0x2F00
• \#define CC112X_FREQOFF_CFG 0x2F01
• \#define CC112X_TOCCFG 0x2F02
• \#define CC112X_MARCSPARE 0x2F03
• \#define CC112X_ECG_CFG 0x2F04
• \#define CC112X_SOFT_TX_DATA_CFG 0x2F05
• \#define CC112X_EXTCTRL 0x2F06
• \#define CC112X_RCCAL_FINE 0x2F07
• \#define CC112X_RCCALCOARSE 0x2F08
• \#define CC112X_RCCALOFFSET 0x2F09
• \#define CC112X_FREQOFF1 0x2F0A
• \#define CC112X_FREQOFF0 0x2F0B
• \#define CC112X_FREQ2 0x2F0C
• \#define CC112X_FREQ1 0x2F0D
• \#define CC112X_FREQ0 0x2F0E
• \#define CC112X_IFADC2 0x2F0F
• #define CC112X_IF_ADC1 0x2F10
• #define CC112X_IF_ADC0 0x2F11
• #define CC112X_FS_DIG1 0x2F12
• #define CC112X_FS_DIG0 0x2F13
• #define CC112X_FS_CAL3 0x2F14
• #define CC112X_FS_CAL2 0x2F15
• #define CC112X_FS_CAL1 0x2F16
• #define CC112X_FS_CAL0 0x2F17
• #define CC112X_FS_CHP 0x2F18
• #define CC112X_FS_DIVTWO 0x2F19
• #define CC112X_FS_DSM1 0x2F1A
• #define CC112X_FS_DSM0 0x2F1B
• #define CC112X_FS_DVC1 0x2F1C
• #define CC112X_FS_DVC0 0x2F1D
• #define CC112X_FS_LBI 0x2F1E
• #define CC112X_FS_PFD 0x2F1F
• #define CC112X_FS_PRE 0x2F20
• #define CC112X_FS_REG_DIV_CML 0x2F21
• #define CC112X_FS_SPARE 0x2F22
• #define CC112X_FS_VCO4 0x2F23
• #define CC112X_FS_VCO3 0x2F24
• #define CC112X_FS_VCO2 0x2F25
• #define CC112X_FS_VCO1 0x2F26
• #define CC112X_FS_VCO0 0x2F27
• #define CC112X_GBIAS6 0x2F28
• #define CC112X_GBIAS5 0x2F29
• #define CC112X_GBIAS4 0x2F2A
• #define CC112X_GBIAS3 0x2F2B
• #define CC112X_GBIAS2 0x2F2C
• #define CC112X_GBIAS1 0x2F2D
• #define CC112X_GBIAS0 0x2F2E
• #define CC112X_IFAMP 0x2F2F
• #define CC112X_LNA 0x2F30
• #define CC112X_RXMIX 0x2F31
• #define CC112X_XOSC5 0x2F32
• #define CC112X_XOSC4 0x2F33
• #define CC112X_XOSC3 0x2F34
• #define CC112X_XOSC2 0x2F35
• #define CC112X_XOSC1 0x2F36
• #define CC112X_XOSC0 0x2F37
• #define CC112X_ANALOG_SPARE 0x2F38
• #define CC112X_PA_CFG3 0x2F39
• #define CC112X_IRQ0M 0x2F3F
• #define CC112X_IRQ0F 0x2F40
• #define CC112X_WOR_TIME1 0x2F64
• #define CC112X_WOR_TIME0 0x2F65
• #define CC112X_WOR_CAPTURE1 0x2F66
• #define CC112X_WOR_CAPTURE0 0x2F67
• #define CC112X_BIST 0x2F68
• #define CC112X_DCFILTOFFSET_I1 0x2F69
• #define CC112X_DCFILTOFFSET_I0 0x2F6A
• #define CC112X_DCFILTOFFSET_Q1 0x2F6B
• #define CC112X_DCFILTOFFSET_Q0 0x2F6C
• #define CC112X_IQIE_I1 0x2F6D
• #define CC112X_IQIE_I0 0x2F6E
• #define CC112X_IQIE_Q1 0x2F6F
• #define CC112X_IQIE_Q0 0x2F70
• #define CC112X_RSSI1 0x2F71
• #define CC112X_RSSI0 0x2F72
• #define CC112X_MARCSTATE 0x2F73
• #define CC112X_LQI_VAL 0x2F74
• #define CC112X_PQT_SYNC_ERR 0x2F75
• #define CC112X_DEM_STATUS 0x2F76
• #define CC112X_FREQOFF_EST1 0x2F77
• #define CC112X_FREQOFF_EST0 0x2F78
• #define CC112X_AGC_GAIN3 0x2F79
• #define CC112X_AGC_GAIN2 0x2F7A
• #define CC112X_AGC_GAIN1 0x2F7B
• #define CC112X_AGC_GAIN0 0x2F7C
• #define CC112X_SOFT_RX_DATA_OUT 0x2F7D
• #define CC112X_SOFT_TX_DATA_IN 0x2F7E
• #define CC112X_ASK_SOFT_RX_DATA 0x2F7F
• #define CC112X_RNDGEN 0x2F80
• #define CC112X_MAGN2 0x2F81
• #define CC112X_MAGN1 0x2F82
• #define CC112X_MAGN0 0x2F83
• #define CC112X_ANG1 0x2F84
• #define CC112X_ANG0 0x2F85
• #define CC112X_CHFILT_I2 0x2F86
• #define CC112X_CHFILT_I1 0x2F87
• #define CC112X_CHFILT_I0 0x2F88
• #define CC112X_CHFILT_Q2 0x2F89
• #define CC112X_CHFILT_Q1 0x2F8A
• #define CC112X_CHFILT_Q0 0x2F8B
• #define CC112X_GPIO_STATUS 0x2F8C
• #define CC112X_FSCAL_CTRL 0x2F8D
• #define CC112X_PHASE_ADJUST 0x2F8E
• #define CC112X_PARTNUMBER 0x2F8F
• #define CC112X_PARTVERSION 0x2F90
• #define CC112X_SERIAL_STATUS 0x2F91
• #define CC112X_RX_STATUS 0x2F92
• #define CC112X_TX_STATUS 0x2F93
• #define CC112X_MARC_STATUS1 0x2F94
• #define CC112X_MARC_STATUS0 0x2F95
• #define CC112X_PA_IFAMP_TEST 0x2F96
• #define CC112X_FSRF_TEST 0x2F97
• #define CC112X_PRE_TEST 0x2F98
• #define CC112X_PRE_OVR 0x2F99
• #define CC112X_ADC_TEST 0x2F9A
• #define CC112X_DVC_TEST 0x2F9B
• #define CC112X_ATEST 0x2F9C
• #define CC112X_ATEST_LVDS 0x2F9D
• #define CC112X_XOSC_TEST1 0x2F9E
• #define CC112X_XOSC_TEST0 0x2F9F
• #define CC112X_RXFIRST 0x2FD2
• #define CC112X_TXFIRST 0x2FD3
• #define CC112X_RXLAST 0x2FD4
• #define CC112X_TXLAST 0x2FD5
• #define CC112X_NUM_TXBYTES 0x2FD6
• #define CC112X_NUM_RXBYTES 0x2FD7
• #define CC112X_FIFO_NUM_TXBYTES 0x2FD8
• #define CC112X_FIFO_NUM_RXBYTES 0x2FD9
• #define CC112X_SINGLE_TXFIFO 0x003F
• #define CC112X_BURST_TXFIFO 0x007F
• #define CC112X_SINGLE_RXFIFO 0x00BF
• #define CC112X_BURST_RXFIFO 0x00FF
• #define CC112X_LQI_CRC_OK_BM 0x80
• #define CC112X_LQI_EST_BM 0x7F
• #define CC112X_SRES 0x3F30
• #define CC112X_SFSTXON 0x3F31
• #define CC112X_SXOFF 0x3F32
• #define CC112X_SCAL 0x3F33
• #define CC112X_SRX 0x3F34
• #define CC112X_STX 0x3F35
• #define CC112X_SIDLE 0x3F36
• #define CC112X_SWOR 0x3F38
• #define CC112X_SPWD 0x3F39
• #define CC112X_SFRX 0x3F3A
• #define CC112X_SFTX 0x3F3B
• #define CC112X_SWORRST 0x3F3C
• #define CC112X_SNOP 0x3F3D
• #define CC112X_AFC 0x3F37
• #define CC112X_STATE_IDLE 0x00
• #define CC112X_STATE_RX 0x10
• #define CC112X_STATE_TX 0x20
• #define CC112X_STATE_FSTXON 0x30
• #define CC112X_STATE_CALIBRATE 0x40
• #define CC112X_STATE_SETTLING 0x50
• #define CC112X_STATE_RXFIFO_ERROR 0x60
• #define CC112X_STATE_TXFIFO_ERROR 0x70

0.37 srd_util.cpp File Reference

Macros

• #define VCDAC_START_OFFSET 2
• #define FS_VCO2_INDEX 0
• #define FS_VCO4_INDEX 1
• #define FS_CHP_INDEX 2

Functions

• void srd_power_enable ()
• void srd_power_disable ()
• void spi_init ()
• void srd_gpio_setup ()
• void srd_gpio_setup_off ()
• void srd_gpio_exti_setup ()
• void srd_gpio_nvic_setup ()
• void srd_gpio0_exti_enable ()
• void srd_gpio0_exti_disable ()
• void srd_gpio2_exti_enable ()
• void srd_gpio2_exti_disable ()
• void srd_gpio3_exti_enable ()
• void srd_gpio3_exti_disable ()
• uint8_t srd_transaction (uint8_t accessType, uint16_t address, uint8_t ∗data, uint16_t len)
• void registerConfig (uint8_t_t powerLevel)
• void manualCalibration ()
0.38  srdhandler.cpp File Reference

Functions

- void sleepRX()
- void srdSleep()
- void fullPowerRunRx()
- void runRX()
- int srd_wakeup()
- void txtest()
- void rxtest()
- void modeTx(uint8_t* txBuffer, uint16_t N)
- void radioRxTxISR()
- void changeConfig(uint8_t data[])

Variables

- volatile uint8_t packetSemaphore

0.39  srdhandler.h File Reference

Macros

- #define BURST_ACCESS 0x40
- #define CMD 0x3F
- #define READ 0x80
- #define WRITE 0x00
- #define ISR_ACTION_REQUIRED 1
- #define ISR_IDLE 0
- #define RX_FIFO_ERROR 0x11
- #define MAX_RXBUFFER 128
- #define BASESTATION 0xFE /* Iron Basestation address */
- #define STATUS_REQ 0x01 /* status request */
- #define GPS_REQ 0x02 /* gps request */
- #define LOG_REQ 0x03 /* log request */
- #define CONFIG_REQ 0x04 /* config request */
- #define CHANGE_CONFIG_REQ 0x05 /* change config request */
- #define CUR_GPS_REQ 0x06 /* gps request */
- #define STATUS_RES 0x20 /* status response */
- #define GET_FIX_RES 0x21 /* Get fix response */
- #define GET_LOG_RES 0x22 /* Get Log file response */
- #define GET_GPS_RES 0x23 /* Get GPS response */
- #define GET_CONFIG_RES 0x24 /* Getconfigs response */
- #define END_LOG_RES 0x25 /* End Log response */
- #define TAG_ADDR 0x51
- #define TAG_ID 0x52
- #define RADIO_SLEEP_MS 0x53
- #define RADIO_CHANNEL_P2 0x54
- #define RADIO_CHANNEL_P1 0x55
- #define RADIO_CHANNEL_P0 0x56
- #define GPS_FIX_INTERVAL 0x57
- #define GPS_BATTERY_THRESHOLD 0x58
- #define GPS_BATTERY_THRESHOLD_X2 0x59
- #define GPS_BATTERY_THRESHOLD_X4 0x60
- #define GPS_BATTERY_THRESHOLD_X8 0x61
• #define GPS_BATTERY_THRESHOLD_X16 0x62
• #define GPS_BATTERY_THRESHOLD_X32 0x63
• #define TELEMETRY_BATTERY_THRESHOLD 0x64
• #define RX_FULL_POWER 0x01
• #define RX_LOW_POWER 0x02
• #define RX_DEFAULT_POWER 0x03
• #define RADIO_CONFIG 0x01
• #define GPS_CONFIG 0x02
• #define BATTERY_CONFIG 0x03
• #define ALL_CONFIG 0x04

Functions
• template<class T >
  void write_buff (uint8_t *buff, int index, T value)
• void srd_gpio_setup ()
• void srd_gpio_exti_setup ()
• void srd_gpio_nvic_setup ()
• void spi_init ()
• void srd_power_enable ()
• void srd_power_disable ()
• void srd_gpio_setup_off ()
• void manualCalibration ()
• void registerConfig (uint8_t powerLevel)
• uint8_t srd_transaction (uint8_t accessType, uint16_t address, uint8_t *data, uint16_t len)
• void radioRxTxISR ()
• void fullPowerRunRx ()
• void runRX ()
• void sleepRX ()
• void srdSleep ()
• void srd_first_time ()
  int srd_wakeup ()
• void modeTx (uint8_t *txBuffer, uint16_t N)
• void txtest ()
• void rxtest ()
• void srd_gpio0_exti_enable ()
• void srd_gpio0_exti_disable ()
• void srd_gpio2_exti_enable ()
• void srd_gpio2_exti_disable ()
• void srd_gpio3_exti_enable ()
• void srd_gpio3_exti_disable ()
• void sendStatusResponse ()
• void sendGetResponse ()
• void sendNoFixResponse ()
• void sendLogFile ()
• void sendGPSLoc ()
• void changeConfig (uint8_t rxBuffer[])
• void sendConfigConfirmation ()

0.40 state_transition.cpp File Reference

Enumerations
• enum state_type {
  SHELL_WAIT, FIX_WAIT, NAV_WAIT, HDOP_WAIT,
  SHELL_INPUT, SRD_INPUT }

Functions

- void enter_hdop_wait_state()
- void enter_shell_wait_state()
- void enter_shell_input_state()
- void enter_fix_wait_state()
- void enter_nav_wait_state()
- void gps_fix_complete()
- void gps_fix_failed()
- void fix_wait_alarm()
- void fix_timeout_alarm()
- void shell_timeout_alarm()
- void hdop_alarm()
- void srd_service_routine()
- void state_handler_gps_nav_isr()
- void state_handler_usart_isr()
- void state_handler_alarm_isr()
- void handle_shell_input(char c)
- void state_handler_main()

Variables

- volatile int hdop_count
- int do_telemetry
- int srd_gps_req
- int do_gps
- char send_buffer[100]
- int buf_ptr
- char prev_char

0.41 state_transition.h File Reference

Functions

- void state_handler_alarm_isr()
- void state_handler_usart_isr()
- void state_handler_gps_nav_isr()
- void state_handler_main()

0.42 tag.cpp File Reference

Functions

- void tag_clock_setup()
- void tag_rtc_setup()
- void tag_exti_setup()
- void tag_nvic_setup()
- void tag_debug_usart_setup(void)
- void tag_flash_setup()
- void tag_setup()
- int tag_get_gps_fix()
- void handle_shell_command(char temp)
- void exti4_isr()
- void rtc_alarm_isr()
- void usart3_isr()
- void exti9_5_isr()
- int main()
0.43 tag.h File Reference

Include dependency graph for tag.h:

```
libopencm3/stm32/f1
/bkp.h ./fix.h ./config.h
libopencm3/cm3/common.h
./gizmo_string.h
./gizmo_math.h
```

Functions

- void tag_clock_setup()
- void tag_nvic_setup()
- void tag_rtc_setup()
- void tag_gpio_setup()
- void tag_exti_setup()
- void tag_usart_setup()
- void tag.flash_setup()
- void tag_setup()
- void tag_led_on()
- void tag_led_off()
- void tag_led_toggle()
- void tag_led_flash(const uint32_t &secs)
- void tag_alarm_isr()
- void tag_gps_fix_isr()
- void tag.usart_isr()
- void handle_shell_command(char temp)
- void tag_setup_shell_entry_mode()
- void tag_setup_fix_mode()
- void tag_leave_telemetry_mode()
- int tag_get_gps_fix()
- void tag_test_gps()
- void tag_test()