University College Dublin

Boat Race Recording

Ciarán Nolan

The thesis is submitted to University College Dublin in part fulfillment of the requirements for the degree of Master of Engineering.

UCD School of Electronic & Electrical Engineering

Supervised by

Mr.Brian Mulkeen

30 April 2020

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Engineering, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

_____

Ciaran Nolan

30 April 2020

Acknowledgements

I would like to thank my supervisor Mr.Brian Mulkeen for his consistent support from day one of this project. His excellent organisation and enthusiasm along with open-mindedness allowed for this project to progress smoothly, for which I am extremely grateful. Similarly, I would like to thank the UCD School of Electronic & Electrical Engineering for enabling this project through the provision of funding. Finally I would like to thank my family and friends for their support throughout the year as this project progressed.

Appendix D: Hazard Identification / Risk Analysis

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

ME ELECTRICAL ENERGY ENGINEERING
or ME ELECTRONIC & COMPUTER ENGINEERING

Please complete and return **two copies** of this form to the School of Electrical & Electronic Engineering Office, Room 226 by 12 noon on Friday 27th September 2019 (c/o Damian Flynn or Mark Flanagan). One copy will be retained by the School. The other copy is retained by you as proof that your completed form has been received. At the end of the project you must include a copy of this form in your final report.

Student Name:	CIARAN NOLAN	Signature: Ciaran Nolan
Student Number:	15329936	
Supervisor:	Brian Mulkeen	Signature: Brian Mulkeen
Project Title:	BOAT RACE RECORDING	
Programme:	ME Electrical or Electronic & Computer Engineering: ME Electronic & COMPUTER ENGINEERING	

Identify any potential safety hazards for your project, e.g. dangerous voltages, rotating machinery, radiation, soldering fumes, sitting in front of a computer for prolonged periods

1) WATER - potential drowning hazards
2) Soldering - Exposing Electronic components to (salt) water
   of electronic components (fumes, splashes)
3) prolonged time seated at computer.
4) Exposure to maritime machinery (propellor engines, anchors)

Indicate how all identified safety hazards will be managed to provide a safe working environment

1) Ensure proper safety equipment: lifejackets, water proof clothing
   Enclosing electronic components in waterproof cases
2) Wear appropriate mask/goggles when operating any soldering irons
3) Ensure correct postures and take regular breaks
4) Be safe Apply correct procedures appliable to the specific vessel

Office Use Only

Date and Time Received:

UCD School of Electrical and Electronic Engineering [School Stamp] 26 SEP 2019

Received by:

Document Tracking

Lecturer:

List of Figures

List of Tables

Appendices

Contents

Abstract

At all levels of sailing, both amateur and competitive, there is an absence of dependable technology to determine a competitor's finish time. As of the time of writing of this thesis, finish times are recorded by teams of individuals, based on subjectivity and the use of the unassisted human eye. This thesis proposes a technological solution, using a combination of Real-time Kinematic (RTK) GNSS and a wireless communication protocol, based on Semtech's LoRa modulation, to calculate and communicate centimetre level positional accuracy used to determine a sailing vessel's finish time, in real-time.

Using a single band, multi-constellation RTK GNSS receiver (u-blox NEO-M8P), an Adafruit Feather M0 Express and an Adafruit RFM96W LoRa Radio Breakout board, finish times were calculated accurate to at minimum, the nearest tenth of a second, by examining captured video tests. The system relied on the centimetre level relative position accuracy of u-blox's moving baseline mode.

A radio link was created, implementing a hybrid Medium Access Control (MAC) protocol, based on the concept of Slotted ALOHA. Software based drivers and managers via AirSpayce's RadioHead library allowed for the implementation of the MAC protocol on the Feather M0.

Lay Abstract

When it comes to sailing in both the context of amateur and professional racing, finish line officiating is completed using the unassisted human eye. As a result, subjectivity and human error are introduced when calculating finish times.

This thesis aims to remove subjectivity from finish procedures, using precision positioning devices and radio communication devices to appropriately calculate a competitor's finish time, which is communicated to race officials. This thesis implements specific high precision positioning methods as well as automated communication procedures to efficiently calculate and communicate competitor finish times, completely removing the human factor from decision making.

Chapter 1: **Introduction**

1.1 Competitive Sailing

Competitive sailing events are run in many configurations, which have an important impact on how they are officiated. In some events, the use of tactics and the 'rules of sailing' are employed, to gain a legal advantage over your competitor.

1.1.1 Fleet Racing

In fleet racing, a competitor's primary focus is to complete the course and cross the line in the shortest time possible. Fleet racing can be either be:

- 'One-design' : All competing vessels are of the same class, with the same specifications. For example, all will use the same sail size. No vessels are considered to have any technical or physical advantage over the competitors due to their identical design.
- 'Handicap Racing' : There are varying classes in the same competing fleet. Since factors such as vessel design and sail size may result in a certain class having a technical advantage, each boat is assigned a handicap rating. The handicap is then applied to either the start or finishing time of each boat, to ensure a level playing field between all classes. This enables vessels with similar, but varying capabilities to compete against each other. In this event, the first boat across the line may not have the fastest time, after handicaps are applied.

1.1.2 Match Racing

In match racing, two boats compete head-to-head to cross the line first. This event relies more heavily on the use tactical maneuvers and the rules of sailing. Teams often attempt to cause their competitor to infringe a rule, resulting in some form of penalty. With these factors considered, steering the quickest course possible to the finish is not typically seen to be the optimal practice.

1.1.3 Team Racing

Team racing is somewhat similar to match racing. Instead of a head-to-head between two competing vessels, team racing comprises of two teams of either two, three or four vessels each. All vessels race at the same time on the same course and while using the low points scoring system (1st: 1 point, 2nd: 2 points etc.) aim to get a winning combination for their team. A winning combination in a fleet of 6, for example would be first, second and fourth.

1.2 Boat Race Recording Process

1.2.1 Background

Many sports which involve the use of a finish line, or require recording the time of a competitor's finish, incorporate some form of automated technology to achieve this goal. These systems are almost always accurate to the tenth or hundredth of a second, at the very least. In the context of sailing, the importance of correct and accurate timekeeping when recording results is apparent when dealing with events like handicap fleet racing. Fleets can consist of hundreds of boats fighting to have the fastest time. With handicaps, accurate time keeping is essential, since the first vessel to cross the line may not necessarily be declared winner.

With such a larger number of vessels in a handicap fleet race, margins between each competitor, after handicaps are applied, can come down to the second or less. Similarly, accurate result taking in both time and order with fleets of this size becomes a difficult process using the currently adopted processes, particularly when finish lines are crowded with finishing boats.

1.2.2 The Finish Process

At sailing events, the finishing and recording procedures are completed by a 'Race Committee'. Lead by the 'Race Officer' (RO), the race committee is responsible for all areas of organisation on the water, from setting up the race courses, to the recording and timekeeping of racing results. For this project, the area of focus is specifically on the finishing procedure of sailing events, which are defined in Section O of Race Management

Manual developed by World Sailing (2019).

A sailing line is constructed using two anchored floating objects, typically at minimum 50 metres, but can be up to several hundred metres apart. One end of the line is usually a vessel, which stations the race committee. The other end is commonly but not exclusively referred to as 'the pin-end', due to the shape of the buoy used. The committee vessel allows the race officer to be in an appropriate position at one end of the line to complete a finish sequence.

To record the finish sequence of a race, the race officer (RO), places himself between a fixed point on the committee boat and uses line of sight to create a finish plane with the pin-end. He or she then decides, subjectively, at what instant each boat crosses the line. In in the case of a head to head finish, the RO must differentiate between leading and trailing boats to award their order in the finish.

Finish times calculated to the nearest second, typically by another member of the committee, with the sole duty of time keeping. Like with the RO's line of sight estimation, finish times are read directly from a digital clock or stopwatch. In cases where there is a crowded finish line, competitors may be partially or fully obstructed from the race officer's view, the race officer must make best estimates, as to when it is determined a competitor crossed the line, and in what particular order.

1.2.3 Flaws in Current Finish Procedures

There a number of issues with this current procedure of boat race recording at every level of competitive sailing, of which this project aims to rectify:

1. Subjectivity: The Race Officer, who is stationed on a floating, non-stationary vessel is basing the finish times upon his or her own estimate of the position of the finish plane.

2. Human Error & Accuracy: While trained and perhaps quite experienced in conducting finish procedures, the Race Officer is making an estimate. In events with harsh conditions on the water, the committee boat can move considerably about its anchored position. For the RO, maintaining a suitable finish plane becomes notably difficult and human error is an unavoidable issue introduced. Combining the fact the timekeepers stationed on the vessel record times to the nearest second

only, this can have a huge impact on the decision of times in handicap racing.

3. Dependency on Line of Sight: As well as accounting for the committee vessel being non-stationary, there may arise cases along a cluttered finish line, where the Race Officer may not have line of sight of vessels crossing the line. Lack of suitable vision of vessels crossing the line equates simply to a 'best guess' attempt at the times and order of finish vessels.

1.3 Project Aims

1. Develop a system, capable of recording the order and finish times of competitive sailing boats accurately.
2. Configure a system such that results and order of competitors may be calculated in real-time.
3. Develop the system to be autonomous in order to remove the possibility of subjectivity and human error from the finishing process.

Chapter 2: **Literature Review**

2.1 Existing Finish Line Technology in Sport

2.1.1 Photo Finish & Image Processing

A photo finish is used in competitive sports when a clear winner cannot be clearly determined through observation. Photo finishes help declare a winner in cases where two competitors cross the line at extremely close intervals.

2.1.1.1 History of Photo Finish

Some of the first applications of the photo finish date back to horse racing in 1878. As demonstrated by O.D and A.R (1878); thin wires placed over the finish line are tripped by the leading horse, resulting in a series of simultaneous images being taken. Photo finish technology has improved considerably since its origin, with uses in sports such as athletics, motor and horse racing, with cameras capable of recording thousands of images per second.

2.1.1.2 Photo Finishes in Sporting Events

In the Olympic Games, the most recent photo finish cameras, developed by Omega have the capabilities of producing up to 10,000 images per second, as outlined in the overview document by Swiss Timing (2017). Dubbed as the 'SCAN O'MYRIA', this camera is used in both the summer and winter games to provide times to the nearest hundredth of a second.

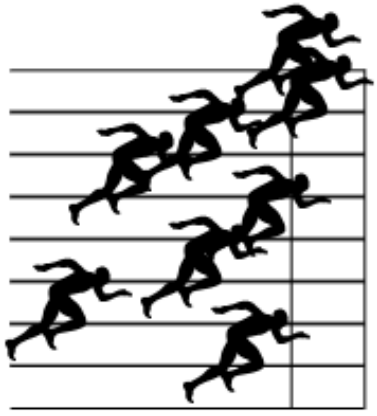
Consider a track event. Depending on the event and as outlined by International Amateur Athletic Federation (2015), a competitor is typically considered to have finished the line at the instant their **torso** crosses the line. As a result, there are a number constraints which must be applied to ensure that a correct judgement can be made from a photo finish.

- Inclination of the camera:

Depending on the sport, a minimum angle of inclination is required between the cam-

era's position and the furthest point along the finish line where a competitor may be positioned. This is to ensure that for any given variation of competitors approaching the finish, the part of the body, or sporting equipment responsible for instigating a finish, can be seen at the instant of crossing. For example, when using the 'SCAN O'STAR', another model of photo finish camera provided by Omega, the recommended angle of inclination between the camera and the furthest competitor in a track based event with human competitors, is approximately 25°to 30°as expressed by Swiss Timing (2012)'s manual.

To illustrate why such an angle of inclination is needed, consider the finish line scenario in Figure 2.1



(a) Finish Line of Athletic Event With Suitable Inclination

(b) Image of Competitors Without Inclination by Photo by Jonathan Chng on Unsplash

Figure 2.1: Comparison of Angles of Inclination For a Track Based Event

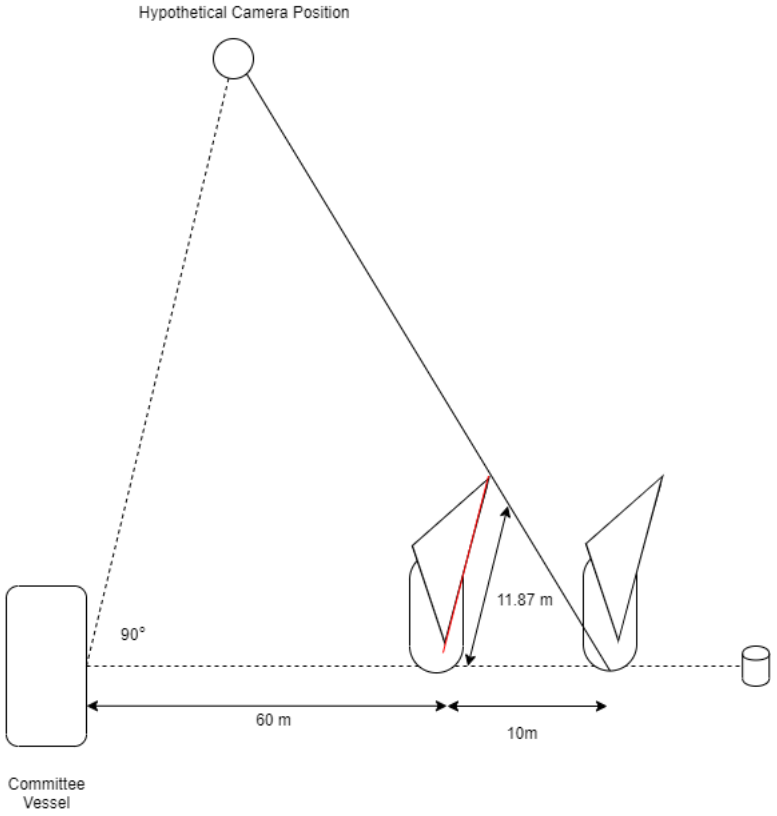
Clearly in Figure 2.1 (a), there is a sufficient angle of inclination to see the torso of all competitors crossing the line. Comparing this to image (b). While in image (b) the photo is not situated at a finish line, it gives a valid illustration of the impact of a lack of inclination. Clearly, the body of the nearest competitor is fully obstructing any view of the competitor in the track next to them.

Incorrect inclination can result in an athlete being obscured from view, making accurate result taking difficult. The specific quantification of 25°is as a result of testing and recommended guidelines for the SCAN O'STAR camera to ensure a sufficient view of all athletes. The required angle of inclination may vary with manufacturer.

Applying the concept of a sufficient angle of inclination to a sailing race, consider the basic situation in Figure 2.2. Note that this illustration assumes the surface of the ocean is, at the instant of line crossing, entirely flat, resulting in both masts being perpendicular to the finish line.

Note that the dimensions of the vessels in this example following the mast specifications of the 'FIRST 31.7', a boat found commonly in the sailing fleets of Dublin Bay, as illustrated by Sailboatdata (2020)

Figure 2.2: Illustration of Angle of Inclination of Finish Camera in a Sailing Race Context

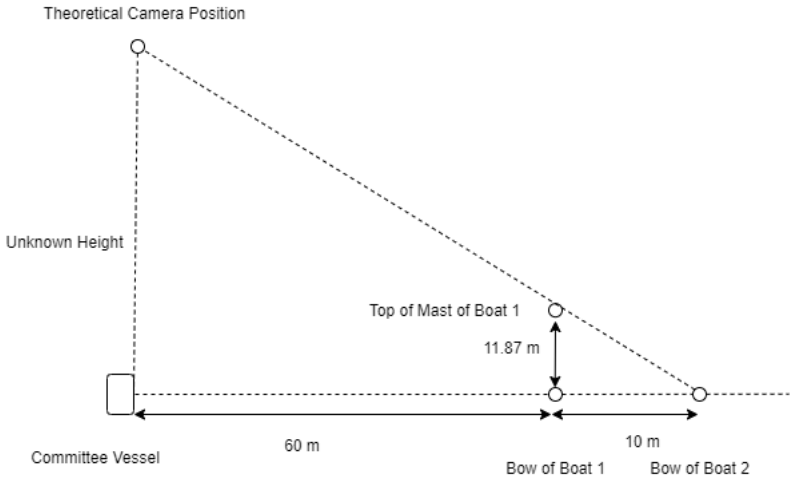


Note the intention of Figure 2.2 is to provide a three dimensional observation of the

placement of a finish camera with respect to the finish line.

Converting the image to a 2D concept:

Figure 2.3: 2D Illustration of Camera Position in a Sailing Race Context



Therefore, to calculate the altitude of the camera, the angle formed by the finish line and projection of the camera onto the bow of the further competitor must be determined. Defining this angle as θ and using the height of the mast of boat 1 and the bow separation between boat 1 and 2:

$$\theta = \arctan \frac{11.87}{10} = 49.887^{\circ}$$

Therefore, finding the required altitude of the camera to see the bow of boat 2:

$$\text{Altitude} = 70 \times \tan 49.887 = 83.089m$$

Therefore, in order for the camera to have a clear view of the bow of boat 2, it must be positioned at an altitude of approximately 83.08m. This implies that some form of device capable of maintaining an altitude of 80m, such as a drone or balloon is required.

- Static Positioning of Finish Camera:

In order for the finish line camera to produce accurate results, it is vital it remains stationary with respect to the finish plane for the duration of the finish sequence as

expressed in Swiss Timing (2012)'s finish camera manual. With land based events, a static position is easily achieved. A simple tripod or bolt fixture would be sufficient.

In the case of sailing, the committee vessel and finishing buoy, operate on a non-static body of water and rotate about their anchored position, responding to wind changes, swells and tides. Considering the unpredictable nature weather, implementing the traditional finish line technology would require additional stabilization and tracking capabilities, on top of the predetermined necessity of inclination.

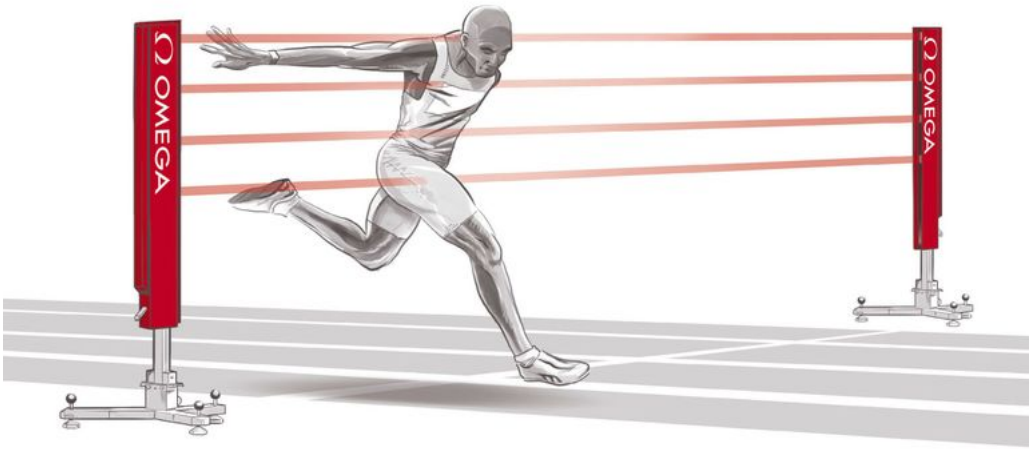
- Manual Image Recognition.

In nearly every application of a photo finish, a trained judge or adjudicator is required to identify the instance a competitor has crossed the line as outlined by LEFORT (2019)'s finish judge guide. As a result, a photo finish camera system alone is not a fully autonomous system, which is the end goal of this project. Addition of a computer vision system however would make such a concept possible.

2.1.2 Break Beam Light Gates

To understand the principle of operation of the break beam light gates, it is necessary to use Figure 2.4 to illustrate the concept.

Figure 2.4: Athlete in a Track Based Event Passing a Break Beam Timing Gate, by WATSON (2016)



As the illustration suggests, the timing system relies detecting beams of light broken by

the athlete. The basic technology behind the break detection is the use of photocells. While many other methods exist, such as photoconductors, phototransistors and photodiodes, many commercial athletic gate systems rely on infrared beams and photocells. Examples include the Smartspeed PT by Fusion Sport (2020), used by Australian and British Olympic committees.

A 'Photocell', in a basic sense, acts as a light sensor. The photocell's resistive values (in Ohms), change in response to a quantity of light incident on it. At a finish line, an array of light beams, typically infrared, are placed at one side of the beam gate, with beams of focused light centered on an equivalent array of photocells. When the beam is broken, the photocell's resistive value will making a sudden change, which can be identified as a passing competitor.

Like with the photo finish, this technology is subject to a number of constraints:

- Identification:

Break beam timing gates only informs officials that an object has passed through the gate. In events with only one participant this is not an issue. With events with more than a single participant in a head-to-head scenario, the light gate will be capable of detecting the leading competitor only. In this case, break beam systems are used only to augment photo-finishes to assist with accurate timing of the winner.

For a sailing race, applying a break beam system would not be appropriate as a standalone system, considering the number of competitors in any given race.

- False Detection:

In track events, where competitors are deemed to have finished an event when their torso crosses the line, light gates are susceptible to breakages from other body parts such as the head or hands. A simple light beam system would be insufficient to differentiate between body parts, so software augmentation would be required to fulfill this necessity.

For a sailing race, where boats may have a sail to their bow (a 'spinnaker' sail) approaching the line, difficulties would arise in differentiating between a vessels spinnaker sail and the bow of the hull.

2.1.3 Radio Frequency Identification (RFID)

Radio Frequency Identification is the term used to describe a system that transmits an identity, typically in the form of a unique serial number, of a particular entity (an object, human, animal). As the name suggests, the identity is transmitted using radio waves as explained by Violino (2008).

RFID tags are configured in two ways, which influence their potential use-cases:

2.1.3.1 Passive RFID Tags

RFID tags consist of a microchip and integrated radio coil antenna. RFID tags are often printed using labels. Passive RFID tags do not have their own power supply and rely on an RFID reader to generate power. The RFID reader sends a radio signal to the antenna of the tag. Due to the coiled shape of the antenna, the reception of a receivers RF wave creates an electromagnetic field and hence induces a high frequency voltage, enabling the antenna to transmit the identification information outlined by Tanim (2016).

2.1.3.2 Active RFID

Unlike passive RFID tags, active RFID tags have their own power supply, but the decision on when to transmit information is dependant on the use-case. Active RFID tags can acts as either:

- Transponders: Transmit a response upon reception of a request from a RFID reader.
- Beacons: Constantly report their location at set intervals to readers positioned about a monitored area. Used for services such as item location in warehouses.

2.1.3.3 Applications in Sport

In very large events, such as marathons, competitors placed closer to the back will be delayed in starting. In some cases, minutes will have passed before reaching the starting point. To rectify this, each athlete is provided with an RFID tag, which defines a start and finish time. The tag is identified by readers placed along the start and finish lines, as in Figure 2.5.

Figure 2.5: Example of Passive RFID Technology Embedded Along a Finish Line

2.1.3.4 Constraints

RFID experiences a number of constraining factors:

- Range:

Due the a passive RFID's lack of a power supply, the power generated from an incoming radio wave from the RFID reader is sufficient to provide a range of 4-6m when operating in the UHF frequency band, as specified by Srikant and Mahapatra (2010).

Active RFID is capable of longer distances, ranging up to 100m, but are susceptible to shorter battery lives, and noticeably high costs.

- Location: Locating a tag is possible, but requires a number of receivers positioned around the target area. Applying this to a water based event such as sailing would require technology to hold the readers in a fixed position through the finish line area.

2.1.4 Conclusion on Existing Finish Line Technologies

After critical review of each of the most common finish line technologies, it was clear that application of any would require significant modification, particularly in the case of a finish line camera. While, in theory devising a system to support the angle of inclination, stabilization and suitable computer vision is certainly possible, the time frame of this project limited the ability to solve all issues at once. As a result, expansion

into alternative positioning systems was made, to evaluate feasibility.

2.2 Geo-Positioning Using Satellite-Based Systems

While there are a number of well established finish-line technologies in operation for certain land and water based events, the requirements to satisfy accurate operation in sailing environments were not met for a number of existing solutions. It was necessary to investigate other technologies that could provide positioning services, which could in theory achieve the aims of this project.

2.2.1 Global Navigation Satellite System (GNSS)

A 'Global Navigation Satellite System', commonly abbreviated to GNSS, corresponds to a set of satellites, or 'constellation', that provide timing and positioning information. GNSS satellites propagate radio signals to Earth, allowing one's location to be determined on the Earth's surface. The process through which a user's location on Earth is calculated is referred to as 'trilateration', which is discussed in detail later in this section.

2.2.1.1 Active GNSS Constellations

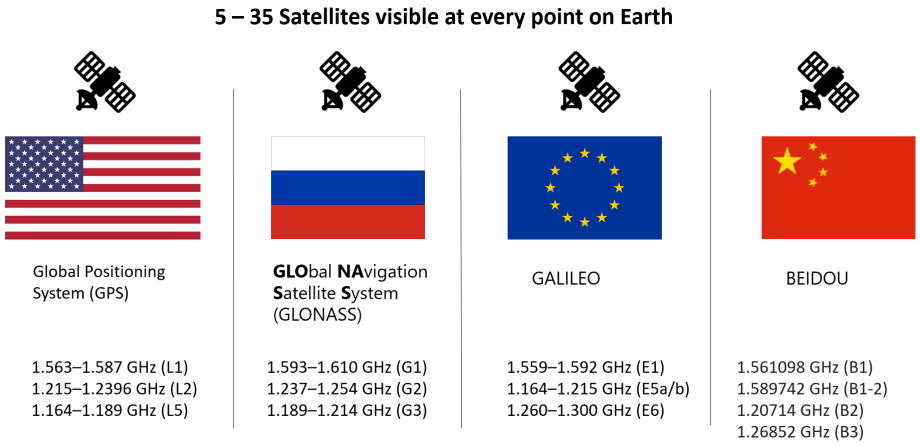
Constellations are configured such that at least four satellites are visible from any points on earth, a requirement for trilateration to be conducted in a typical receiver. The reason for this requirement is explained in Section 2.2.1.4. Examples of global systems can be seen in Figure 2.6:

Each satellite broadcasts on a set of frequency bands, unique to the GNSS constellation it belongs. For example, the American GPS system broadcasts on the L1, L2 and L5 bands. The designs applied to each of these constellations, is such that many individual satellite signals can coexist in the same frequency band. This allows for frequency bands of certain global systems to be shared. The challenge with shared frequency bands is the ability for GNSS signals of one system to be compatible with another.

2.2.1.2 Compatibility of GNSS Signals

Quoting Stupak and International Committee on GNSS (2015), as part of a 2015 presentation:

Figure 2.6: GNSS Constellations



> "Compatibility refers to the ability of global and regional navigation satellite systems and augmentations to be used separately or together without causing unacceptable interference and/or other harm to an individual system and/or service"

Steps towards cooperation between GNSS system providers have been made in recent years. As shown by Zaminpardaz, P. J. G. Teunissen and Nadarajah (2017), a notable move towards compatibility between GNSS systems was the decision in 2011 for the first GLONASS satellite to use a Code-Division Multiple Access (CDMA) access protocol, the same found in GPS satellites. Traditionally, GLONASS signals used Frequency-Division Multiple Access FDMA.

However, due to political tensions and disagreements, advancements are often delayed or halted completely. As discussed by Larsen (2015), the emerging conflict in Ukraine circa 2014, resulted in breakdown in cooperation between the US and Russia, with Russia closing all GPS monitoring stations located in the country. Since, as can be seen from Figure 2.6, the frequency bands of the GPS and GLONASS constellations do not suffer significant overlap, the ability to use both systems simultaneously, as of the time of writing of this report is not significantly affected.

Other systems, such as Europe's Galileo, operate using the same center frequency as GPS in their E1 and E5 bands (L1 and L5 equivalent). Since Galileo uses the same CDMA protocol, differentiating between a GPS and Galileo signal can be achieved.

The process through how this identification process is completed is described in the next section.

2.2.1.3 Identification of GNSS Signals at Receiver Level

Modern GNSS signals have seen significant modification in recent decades to enhance interoperability with other GNSS systems. The detailed modulation techniques which enhance their usability are outside the scope of this project. However, the common factor between all service types, with respect to traditional GNSS to be examined in this review, is the use of a very particular modulation type, using a 'Pseudo Random Code' (PRC). Newer generation satellites may employ variations to the traditional PRC, but in operation, the tracking principles remain the same. For example, the GPS Plan for the L1 band from Figure 2.6, can be seen in 2.7. Note the M-Code service, restricted for military use, has no accessible PRC.

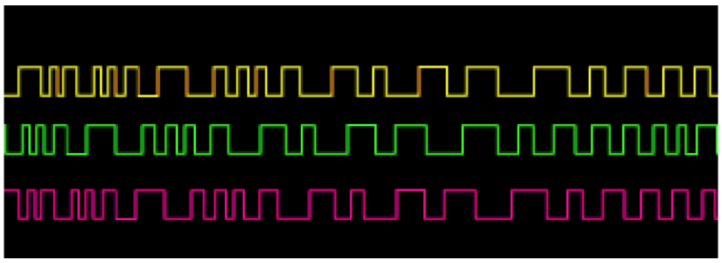
Figure 2.7: GPS Signal Plan from European Space Agency 2011

GNSS System	GPS	GPS		GPS	GPS
Service Name	C/A	L1C		P(Y) Code	M-Code
Centre Frequency	1575.42 MHz	1575.42 MHz		1575.42 MHz	1575.42 MHz
Frequency Band	L1	L1		L1	L1
Access Technique	CDMA	CDMA		CDMA	CDMA
Signal Component	Data	Data	Pilot	Data	N.A.
Modulation	BPSK(1)	TMBOC(6,1,1/11)		BPSK(10)	$BOC_{sin}(10,5)$
Sub-carrier frequency [MHz]	-	1.023	1.023 & 6.138	-	10.23
Code frequency	1.023 MHz	1.023 MHz		10.23 MHz	5.115 MHz
Primary PRN Code length	1023	10230		$6.19 \cdot 10^{12}$	N.A.

Since many constellations broadcast on the same frequency, this results in a considerably cluttered signal space. The aforementioned PRC is a complicated sequence of 1's and 0's. Due to the similarities between these sequences and the characteristics of random electrical noise, the signals are commonly referred to as Pseudo Random Noise (PRN) codes. For each satellite, this code is repeated at set intervals, the length of which depends on the generation of satellite and signal used and contains a unique fingerprint to identify the individual satellites. As outlined by Broumandan et al. (2012), sample PRNs of the American GPS satellite system are: 15, 16, 18, 21, 24. GNSS receivers are

programmed with a list of all PRNs they are capable of tracking.

Figure 2.8: PRN Examples by Avionic West (2020)



In order for a traditional GNSS receiver to determine its position, it must be capable of tracking multiple PRNs simultaneously. The receiver first generates a replica of each PRN it is capable of tracking. Next, since most receivers use a single antenna to accept all satellite communication, the device must make several identical copies of the received signals, from the set of visible satellites.

The number of copies depends on the design of the receiver and it's advertised accuracy. Each signal is digitized and a specific PRN from the list stored in the receiver is searched for. Should the PRN not be found, the receiver will move down the list until a valid PRN is found. In theory, since the receiver and satellite are both synchronised to GNSS time (the satellites to a considerably higher degree of accuracy), the received and generated codes would be expected to be identical. However, since the satellite signal must propagate to earth, the difference between the two signals represents the propagation time of the satellite signal.

Once a valid PRN is found, the receiver 'slews' the internally generated code, until the best possible correlation with the received PRN is found. The amount by which the replicated signal is slewed, signifies the time of travel from the transmitter. Once a given signal has been identified, a GNSS receiver can implement a tracking loop, to guarantee synchronisation with the incoming signal. As demonstrated by Vu and Andrle (2014) a common tracking method employed is through the use of a Phase Locked Loop (PLL).

It is important to note, that the receiver tracks this PRN using its local clock, which is orders of magnitude less accurate than the atomic clocks on board GNSS satellites. As explained by Novatel (2019), accuracy in newer rubidium based clocks are accurate to

approximately:

$$\pm 5 \times 10^{-11} seconds$$

Since the receiver correlates with a pseudorandom **code**, the positioning method which takes advantage of PRN tracking is also referred to as 'code-based positioning".

2.2.1.4 Code-Based Positioning Using GNSS - Trilateration

Code-based positioning relies on the phase tracking of the PRNs described in the previous section. When calculating the position of a GNSS receiver, the first required measurement is the propagation time of a GNSS signal from satellite to receiver. In addition to the PRN, each satellite signal is modulated with a low-rate broadcast navigation message, containing information relating to the satellites orbit. Using the approximated 'slew' of the replicated code described in the previous section, along with the time of arrival of the PRN at the receiver, a propagation time can be calculated:

$$\text{Propagation Time } (\tau) = \text{Time Signal Reached Receiver} - \text{Time Signal Left Satellite}$$

Knowing this result and, for the moment, assuming no error sources exist, the range to the satellite can be found by calculating the product of propagation time with the speed of light.

$$\alpha = c \times \tau,$$

Where:

$$\text{c} = \text{Speed of Light, } \alpha = \text{Range}$$

Using these ideal conditions, a vector is defined describing the position of the receiver and satellite. Since the receiver may be non-stationary, both the receiver and satellite are expressed as a function of time, with the receiver at the time of signal reception and satellite at the time of signal transmission. The vectors are presented in 'Earth-Centered, Earth-Fixed' (ECEF) frame formats, like the World Geodetic System described by National Geospatial-Intelligence Agency (2020).

Expressing the receiver and satellite positions:

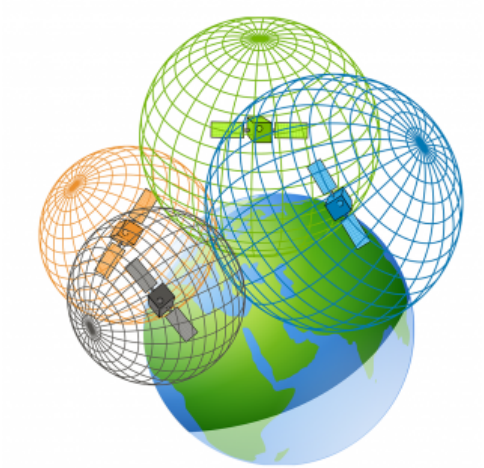
$$\text{Satellite Position: } s = (x_s(t-\tau), \quad y_s(t-\tau), \quad z_s(t-\tau))$$

$$\text{Receiver Position: } r = (x_r(t), \quad y_r(t), \quad z_r(t))$$

$$\alpha = \|r - s\|$$

Finding the range, α, for a single satellite, would inform the receiver that its location lies somewhere on a sphere with a centre of the satellite antenna, and radius α. Repeating for three satellites, the receiver's location will become the intersection of three spheres. Since the calculation is with respect to an ECEF frame, one of the points of intersection will appear on Earth, while the other will not. Intuitive elimination hence allows for the correct position to be determined. Otherwise, the addition of a fourth satellite would allow for the correct point to selected. This can be visualized in the Figure 2.9:

Figure 2.9: Trilateration Using Four Satellites from GISGeography (2016)



Representing the three satellite calculations using three equations.

$$\alpha_1 = \|s_1 - r\| = \sqrt{(x_r(t) - x_{s_1}(t-\tau))^2 + (y_r(t) - y_{s_1}(t-\tau))^2 + (z_r(t) - z_{s_1}(t-\tau))^2}$$

$$\alpha_2 = \|s_2 - r\| = \sqrt{(x_r(t) - x_{s_2}(t-\tau))^2 + (y_r(t) - y_{s_2}(t-\tau))^2 + (z_r(t) - z_{s_2}(t-\tau))^2}$$

$$\alpha_3 = \|s_3 - r\| = \sqrt{(x_r(t) - x_{s_3}(t-\tau))^2 + (y_r(t) - y_{s_3}(t-\tau))^2 + (z_r(t) - z_{s_3}(t-\tau))^2}$$

Since the satellite positions s_1, s_2, s_3, time of transmission at satellite, τ and time of reception, t are all known, this is a system of three linear equations in three unknowns. Solving this equation can be completed using a standard iterative linearisation approach. Once solved, the x, y, z co-ordinates of the receiver have been determined.

However, ideal conditions applied here only display the fundamental process of code-based positioning. As previously mentioned, the accuracy of the clock at the receiver is considerably less accurate than the atomic clocks found in the satellites. Furthermore, at power up, a receiver clock is not synchronised with any satellite. Since this calculation uses the speed of light, clock inaccuracies of the order of microseconds can result in positional errors in the kilometre range. Therefore, defining the receiver clock offset and then the induced error as a result of such:

$$\text{Clock Offset} : dt_r$$

$$\text{Clock Offset Induced error} : dt_r \times c$$

Where:

$$\text{c} = \text{Speed of Light},$$

Adding the error into the original range equation:

$$\alpha = \|r - s\| + cdt_r$$

Since a new unknown has been introduced, the system of linear equations now requires four satellites:

$$\alpha_1 = \sqrt{(x_r(t) - x_{s_1}(t-\tau))^2 + (y_r(t) - y_{s_1}(t-\tau))^2 + (z_r(t) - z_{s_1}(t-\tau))^2} + cdt_r$$

$$\alpha_2 = \sqrt{(x_r(t) - x_{s_2}(t-\tau))^2 + (y_r(t) - y_{s_2}(t-\tau))^2 + (z_r(t) - z_{s_2}(t-\tau))^2} + cdt_r$$

$$\alpha_3 = \sqrt{(x_r(t) - x_{s_3}(t-\tau))^2 + (y_r(t) - y_{s_3}(t-\tau))^2 + (z_r(t) - z_{s_3}(t-\tau))^2} + cdt_r$$

$$\alpha_4 = \sqrt{(x_r(t) - x_{s_4}(t-\tau))^2 + (y_r(t) - y_{s_4}(t-\tau))^2 + (z_r(t) - z_{s_4}(t-\tau))^2} + cdt_r$$

This system of equation can be solved using a number of different iterative methods involving linearisation, which vary with manufacturer, explained by P. J. Teunissen and Montenbruck (2014). This error highlights the need for four satellites in a trilateration process, and hence why GNSS has a minimum of 5 visible satellites at any point on earth.

2.2.1.5 Sources of Error for a GNSS Signal & Code-Based Receiver

Apart from the already mentioned receiver clock errors, there are a number of factors which considerably affect the propagation of a GNSS signal, as well as the positional accuracy experienced at a traditional receiver.

- Pseudocode Measurement Noise

Since the PRN is a combination of 1s and 0s, the receiver must compare the rising and falling edges of the transitions. Modern electronics are capable of measuring such an offset transition to approximately one percent of a bit pulse width. With the chip or pulse width of the L1 C/A signal equal to 293 metres, the one percent error at the receiver corresponds to 3 metres of error. For a sailing application, this is an intolerable error since centimetre level accuracy is desired. More accurate applications of GNSS systems are sought.

- Atmospheric Interference

As previously mentioned, GNSS data is broadcast using electromagnetic waves. As these waves enter the upper levels of the atmosphere, they experience refraction. There are two layers of atmosphere which are primarily responsible for this error source:

Ionosphere

In the ionosphere, two different effects are experienced, depending on the time of day. During the day, radiation from the sun, ionises neutral atoms, creating free electrons and

ions, increasing the electron density. The larger the density or total electron content (TEC), the larger the refraction and delay of the satellite signal. During the night, the lack of sun radiation results in the recombination of electrons with atoms, reducing the TEC and hence the delay the propagation of the GNSS signal. As expressed by P. J. Teunissen and Montenbruck (2014), the approximated affects of the TEC in the ionosphere is dependant on the the frequency of the transmitted signal:

$$I = 40.3\frac{TEC}{f^2}$$

Where:

$$I = \text{Magnitude of Code Phase Delay}, f = \text{frequency of transmitted signal}$$

So with an inversely proportional relationship, the higher the frequency, the lower the delay. Many models have been developed to approximate the ionospheric delays found across the globe, which are provided as part of a satellite's navigation message.

For single band, code-based receivers, computing corrections to these delays relies entirely on the provided model approximation, or through the use of post-processing, the later of which is not always a readily available option. The models are not perfect, an error is not completely avoidable. Unavoidable errors in any range above tens of centimetre result in a system unsuitable for a finish line in a sailing application.

Since the magnitude of the delay is a function of the signal frequency, a dual frequency receiver can reduce the effects of the ionosphere by comparing the relative delay from two frequencies.

Neutral Atmosphere - Troposphere

Outside of the ionosphere, exist 'neutral' atmospheric conditions, such as the troposphere. These layers still cause refraction, however are not a function of frequency. This layer is electrically neutral and filled with air and water vapour. Since the refraction is independent of any frequency, models exist to account for the delay in this layer.

Approximated contributions in a test case of the two layers of the atmosphere for

single band GPS signals can be seen in Table 2.1 through experimentation by Arbesser-Rastburg (2006):

Atmospheric Layer	Estimated Position Error
Ionosphere	4.0m
Troposphere	0.7m

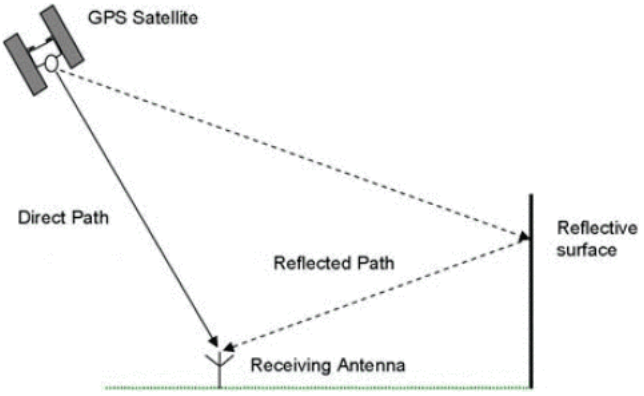
Table 2.1: Positioning Errors Induced by Unmodeled Atmospheric Conditions

Of course, these errors would be unacceptable for judging the finish line of a sailing race, where cm level accuracy is required. These errors alone make, single band, code-based receivers unsuitable for the application.

- Multipath Error

Multipath refers to when a satellite signal arrives to a GNSS receiver via more than one path. This occurs due to reflection of signals and results in certain signals not making the most direct path possible to the receiver. The position of a receiver greatly affects the influence of multipath. For example, urban areas with tall structures will increase the chance of receiving reflected signals. Figure 2.10 shows the effect multipath can have on a receiver. Modern receivers have algorithms capable of detecting and mitigating multipath error, using time domain processing techniques by Kos, Markezic and Pokrajcic (2010).

Figure 2.10: Example of Multipath Error in GNSS Receiver by Kos, Markezic and Pokrajcic 2010



2.2.2 GNSS Augmentation

Knowing the factors that limit the traditional GNSS systems, systems have been developed to overcome certain factors to improve accuracy. With some methods claiming accuracy down to the centimetre level, research was conducted on augmentation systems, as traditional GNSS does not provide sufficient accuracy for a finish line application.

2.2.2.1 Differential GNSS (DGNSS)

DGNSS applies the same code based measurements as described in Section 2.2.1.4. DGNSS uses a 'base-station' to rover approach. The configuration of a DGNSS base-rover is as follows:

- Base Station:

A DGNSS system has a base station in a fixed position, the location of which is known to a high degree of accuracy. Upon reception of GNSS signals, the base station will conduct code-based measurements, and will compare this calculated position to its known position. The difference between the two positions allows the base station to approximate for errors such as clock bias and atmospheric effects. These effects are the same as those explored in Section 2.2.1.5. These errors are transmitted, as correctional data via some communication link (e.g. UHF) to the rover for processing.

- Rover:

The rover is non-stationary, and conducts code-based measurements in the same way a traditional GNSS receiver would. However, the receiver is capable of receiving the correctional data transmitted from the base station, which can be applied to the raw measurements it receives at its own antenna. In order for this system to work, both the base and rover must share a common set of satellites, and therefore, as estimated by P. J. Teunissen and Montenbruck (2014), the distance between the two devices must remain under 1000km. This correction process effectively removes the impact of clock bias and atmospheric affects at the rover position

However, the DGNSS system still relies on code-based measurements, which, as in Section 2.2.1.5, are shown to have receiver errors in the metre range. While this is a significant improvement form the tens of metres experienced by traditional GNSS units,

code-based solutions remain insufficient to provide centimetre level accuracy needed at the finish line of a sailing race.

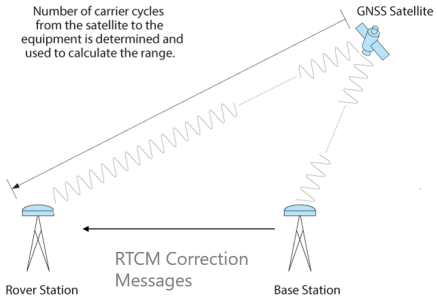
2.2.2.2 Real-Time Kinematic (RTK) GNSS

Real Time Kinematic GNSS is the most promising technology researched. RTK is itself a DGNSS concept. RTK Solutions, like DGNSS are configured in a base-rover configuration, where the base broadcasts correction data to the rover. For traditional RTK, like DGNSS, the base must be placed in a position of known accuracy.

The main difference lies in how satellite position measurements are made. Unlike traditional GNSS and differential GNSS, which use code-based measurements relating to the PRN, RTK relies on tracking the carrier signal of the GNSS transmission.

While it has been discussed in detail the concept of PRN modulation, as well as the inclusion of navigation data in a GNSS signal, the final component is the sinusoidal carrier wave, which takes the signal from the satellite to earth. Referring to Figure 2.11, we can see the centre frequency of the carrier for the L1 band is 1575.42MHz. RTK systems make use of this carrier wave by making a set of measurements. The process through which this is completed is discussed in detail in the next section.

Figure 2.11: Basic RTK Base-Rover Configuration by Novatel (2020)



2.2.2.3 Carrier Phase Measurements

As previously discussed, the noise of code-based measurement at the receiver was approximately 3m due to the wavelength of the PRN. The wavelength of the carrier signal is considerably shorter. Using Figure 2.7 the wavelength can be calculated by using:

$$\lambda = \frac{c}{f}$$

Where:

$$\lambda = \text{wavelength of signal}, c = \text{speed of light}, f = \text{frequency of signal}$$

This results in:

$$\lambda \approx 0.19m$$

Clearly the margin for error with a one percent bit pulse accuracy margin is in the millimeter range. Combined with the correctional data stream to mitigate atmospheric and clock bias affects, many RTK receivers advertised centimetre level accuracy, which is suitable to be used in a finish line setting.

To determine the range between the receiver and satellite, the task at hand is to measure the number of cycles of the carrier wave that exist between the satellite and receiver. However, a fundamental issue must be overcome. The moment a RTK receiver begins to track the satellite's carrier signal, it is extremely unlikely that the tracking starts precisely at the beginning of a wavelength's phase cycle. As defined by Van Sickle and Pennsylvania State University (2018), there are two factors at the first lock on of the carrier signal:

$$\alpha = \text{fractional initial phase}$$

$$N = \text{Number of integer phase cycles}$$

RTK GNSS bases its range calculation on the number of cycles of the satellite's carrier frequency, from time of transmission at the satellite, to reception at the receiver. The observations of the carrier phase enable centimeter level precision, however the difficulty lies obtaining the exact number of cycles directly. It should be noted, that the number of carrier cycles is always an integer, as a result, floating point approximations are a

degradation in accuracy rather than an increase in precision.

To gain access to the carrier phase observations, it is necessary to 'strip' the signal of its other components, namely the PRN and broadcast data. This can be achieved by several methods including integrating the Doppler shift of the signal.

2.2.2.4 Carrier Phase Analogy

Visualising the concept of carrier phases and the challenges in extracting the exact number of integer cycles can be challenging. Petovello, O'Driscoll and University of Calgary (2010) put forward an excellent analogy to understand the difficulty in generating position solutions using the carrier signal.

It is appropriate to visualise the carrier signal as a measuring tape that extends from the satellite antenna to receiver antenna. The measuring tape is marked at millimeter intervals. However, at the end of every wavelength (take L1, calculated above, for example), the tape resets to 0 millimeters. Using the L1 band, this corresponds to a reset every 19cm. So, this shows that while we can see the wavelength of the signal, the number of cycles that exist is not yet known, since the receiver cannot distinguish one carrier cycle from another.

2.2.2.5 Ambiguity Resolution

Since the receiver measures the fractional phase as previously mentioned, the best effort to be made is to track the change of phase. Carrier phase ambiguity resolution, aims to achieve the exact integer number of cycles of the carrier between the time of transmission and reception of the signal. It is a somewhat complex computation, whereby the intricate details of its composition are outside of the scope of this project.

What is necessary to understand, is the effect of failing to achieve an exact integer number of cycles in computation.There are a number of different methods developed, however often it is the intellectual property of RTK manufacturers as to how this task is completed. A common method used is the concept of Double-Differenced ambiguity resolution.

After applying an ambiguity resolution process, an RTK receiver can have achieved one of two results:

- Fixed Ambiguity: The receiver has successfully calculated the **integer** number of carrier cycles between satellite and receiver. This allows for accuracy in the range of two centimetres or less, as advertised by many companies.
- Float Ambiguity: The receiver has not currently calculated the exact integer of carrier cycles, but has approximated to a floating point number. P. J. Teunissen and Montenbruck 2014.

In both cases, cm level precision is possible. Floating point carrier cycle estimations lose accuracy due to, intuitively, not representing the real number of cycles and hence the real range between the satellite and receiver. The error could simply be a fraction of a single cycle, however poor signal reception can push this to more. In the case of the L1 band, a float solution with a wavelength of 19cm, would at maximum be < 19 cm. Often the range is less than 5 cm. This is very much dependent on the quality of the satellite observations as explained by Novatel (2020). Clearly in the case of a fixed solution, the accuracy advertised is certainly sufficient.

2.2.2.6 Relative Positioning

A common application of RTK and carrier phase measurements is the ability to achieve centimetre level relative positioning. Relative Positioning with modern RTK receivers can be applied in two configurations:

- Static Baseline Relative Positioning

Static baseline positioning employs the same base-rover concept as both traditional RTK and DGNSS. The base is fixed in an accurately known position, and computes the same carrier phase measurements with ambiguity resolution. However, the position of the rover is not calculated using a globally defined reference frame like the World Geodetic System. Rather, a local coordinate frame such as North East Down (NED) offset frame, centred on the base position is the replacement.

Suddenly, the importance of locating the rover on the globe is not a priority, rather the vector describing the offset of the rover to the base. Still the base broadcasts its correction data as a result of the carrier phase observations it makes.

- Moving Baseline Relative Positioning

Moving baseline varies from traditional RTK and differential GNSS as, how the name suggests, in the fact that the base station does not remain stationary. Moving baseline is not capable of conducting centimetre level **absolute** positional measurements, and its primary purpose is for relative positioning applications. Moving baseline still conducts carrier phase measurements at receiver level, however the nature of corrections from the base will change.

Moving baseline relies on the fact that both the base and rover share the same set of visible satellites. With a set of common satellites, the majority of error sources will be constant in that particular locality. Since relative positioning relies on a **local** reference frame only, common errors across both the base and rover can be eliminated through applied correctional messages, allowing the rover to compute a relative vector to centimetre level accuracy.

2.3 Review Conclusion

With the review of existing finish line technology in the industry, a conclusion was drawn that the applicability of any of the presented systems to the finish line of sailing race was not apparent. What this system requires is to track all competitors down to the centimetres at which they cross the line. While some technologies, namely a finish line camera could achieve this, through the use of drone or balloon based configuration, with stabilization and computer vision, RTK seemed a more achievable goal for this project.

As a result, an alternate form of positioning system was researched. While traditional GNSS receivers were shown to not be sufficient in providing the level of accuracy needed, augmentation solutions certainly were proven to be capable. RTK GNSS was shown to be capable of centimetre level accuracy with both absolute and relative positioning. As a result, this project's finish line system is based on RTK technology.

Chapter 3: **Method**

3.1 Boat Race Recording Technology Decision

The research element of this thesis concludes that RTK GNSS is the technology that this project will be built on, due to its centimetre level accuracy in both absolute positioning (with respect to an ECEF global frame) and relative positioning (with respect to a local NED frame centred on a base station).

With the technology chosen, the next step was to evaluate the consumer grade, RTK ready receivers available on the market, to incorporate into the project. This technology yields receivers from prices ranging from less than one hundred euro, to several thousand. As a result there was a set of criteria established, to evaluate the best performance for price, within the monetary bounds of this project.

3.1.1 RTK Finish Line Overview

Before deciding on a product, it was pertinent to have a rough overview as to how RTK would be used at the finish line of sailing races. The concept is straightforward. The committee boat, pin mark and all competitors are fitted with an RTK receiver. Depending on the evaluation of absolute and positional accuracy, the base station may be the receiver fixed on the committee boat, or a known position a shore.

Knowing the location of the committee boat and the pin in absolute or relative terms allows for a hypothetical finish plane to be constructed. The positions of each competitor would then be monitored to evaluate when a line crossing occurs.

3.1.2 Product Decision Criteria

3.1.2.1 Price

The first factor to consider was the vast price range of RTK receivers. Considering this to be an academic project, the targeted price range would remain in the sub €1000 region. This would remove the strictly commercial applications, such as Swift Navigation (2020) Piksi Multi, which at the time of writing costs approximately €2,150 for a two piece

evaluation kit (Receiver, Antenna & Radio Link).

Looking in more detail at the 'consumer grade' products, there were a number of products available at the sub €1000 region. Most notable of which were manufactured by a Swiss company called 'u-blox'. In the past decade, u-blox have been carving a market in affordable precision location products, including RTK. They currently manufacture two main receiver chips.

- NEO-M8P: Single Band (L1/G1/B1) Dual-Constellation RTK Receiver

The NEO-M8P is u-blox's single band RTK receiver. Hence the description, it is capable of tracking the L1, G1 and B1 bands of GPS, GLONASS and Beidou respectively. Similarly it is capable of tracking two constellations simultaneously, for atmospheric and clock bias mitigation which were discussed in section 2.2.1.5. A single chip starts at approximately €100, with a further decrease for high volume customers. This solution was immediately considered as one of the most promising based purely on its price, hence its capabilities were more thoroughly examined in the next section

- ZED-F9P: Multi-Band (L1-L2/G1/B1) Multi-Constellation RTK Receiver

The ZED-F9P is u-blox flagship receiver. Hence this fact is reflected in the price. This receiver is capable of covering all four major GNSS systems as in Figure 2.6 simultaneously. As outlined by U-blox (2020b), it is similarly capable of concurrent reception of the L1 and L2 bands of GPS and all equivalent bands of the remaining three systems (G1/G2, B1/B2, E1/E5B). The ZED-F9P receiver chip was priced at €175 for a single unit.

Simply based on the value of these boards, combined with reputation of u-blox in the positioning industry, the choice had been narrowed to the NEO-M8P or ZED-F9P. It must be noted that these prices reflected the cost of the receiver chips alone, and did not account for evaluation boards, single-band/multi-band antennas and appropriate radio links to get an RTK GNSS setup. These factors are considered in the following sections.

3.1.3 Receiver Chip Accuracy Estimates

Clearly, from the initial specifications, the ZED-F9P would outperform the NEO-M8P. However, it was considered that the NEO-M8P would still be more than sufficient with

respect to accuracy.

Important specifications from U-blox (2020b) are compared in Table 3.1

RTK Performance Metric	NEO-M8P	ZED-F9P
Position Accuracy	0.025m	0.01m
Convergence Time (Ambiguity Resolution)	$<$ 60 sec	$<$ 10 sec
Acquisition Time (Cold Start)	26 sec	24 sec
Reacquisition Time	1 sec	2 sec

Table 3.1: RTK Specifications of NEO-M8P and ZED-F9P Receivers

Clearly, both both receivers are capable of centimetre level accuracy, as desired in this finish line system. ZED-F9P is noticeably more accurate, approach millimetre accuracy. In terms of convergence time, i.e. the integer ambiguity resolution discussed in Section2.2.2.5, the ZED-F9P is also noticeably faster, at under ten seconds. However, this level of speed is not required, since race preparation in sailing take minutes, at the very least.

The promising specifications pointed the decision in favour of the NEO-M8P, due to its price point over the ZED-F9P. An evaluation kit for NEO-M8P was researched.

3.1.3.1 Evaluation Kits

In order to evaluate the advertised accuracy of RTK out of the box, the decision was made to invest in an evaluation kit. With such a kit, minimal setup would be required, allowing for a conclusion on the suitable ability of this technology for a finish line to be quickly.

Referring back to the choice between the NEO-M8P and ZED-F9P, u-blox provides an evaluation kit for both.

- C94-M8P: NEO-M8P Evaluation Kit - €400 For a Set of 2

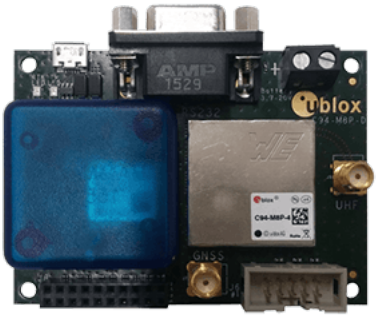
The C94-M8P is an breakout board and accessory pack, designed for evaluating RTK straight out of the box. The board contains both the NEO-M8P receiver, along with a preconfigured UHF radio-link, using the Irish License free 433MHz band as outlined by Commission for Communications Regulation (2020). The remaining contents of the kit included a single band GNSS antenna and micro-USB data cable.

- C099-F9P-1: ZED-F9P Evaluation Kit - €260 for a Single Board.

The C099-F9P serves the same purpose in allowing out of the box RTK evaluation. However, instead of a UHF radio link, WiFi and Bluetooth were incorporated for short range connectivity options. Short range connectivity was not the desired application sought, so an additional radio link would be required to achieve an appropriate distance to represent base to sailing vessel.

Considering the price for a ZED-F9P and the additional requirement of a longer range radio link, the decision was made to purchase the C94-M8P kit, as in Figure 3.1

Figure 3.1: The C94-M8P Application Board



3.2 Evaluation of the C94-M8P

Before building a system based around this technology, it was pertinent to endure the fundamental operation of these boards provided the accuracy desired. A number of tools and protocols are used to enable this task, the importance of each is outlined in detail to the operation of the hardware.

3.2.1 C94-M8P Hardware

For evaluating this board, it was important to determine what interfaces were accessible to provide the relevant information and to receive configuration instructions from the user.

Direct access to the NEO-M8P receiver was available through either a micro-USB (which doubled as a power supply), or a UART connection. UART or Universal Asynchronous

Receiver/Transmitter, is a hardware entity for transferring bytes of data using logic levels. UART was not used in the early evaluation and is described in more detail once it is incorporated into the system, found later in this section. USB connectivity was used initially due to its compatibility with a laptop without the need for a serial converter.

Access to the radio modem is through an RS-232 port. RS-232 or 'Recommended Standard' is another serial communication protocol. From this port, configuration to radio settings can be made using AT commands. AT commands are a defined set of commands to alter the configuration of a modem. Like UART, the radio modem remains unchanged from its default configuration for the period of initial evaluation.

3.2.2 C94-M8P Configuration using u-blox u-center Software and Communication Protocols

Configuring the boards for initial evaluation required the use of a number important tools and protocols.

3.2.2.1 u-blox UBX Protocol

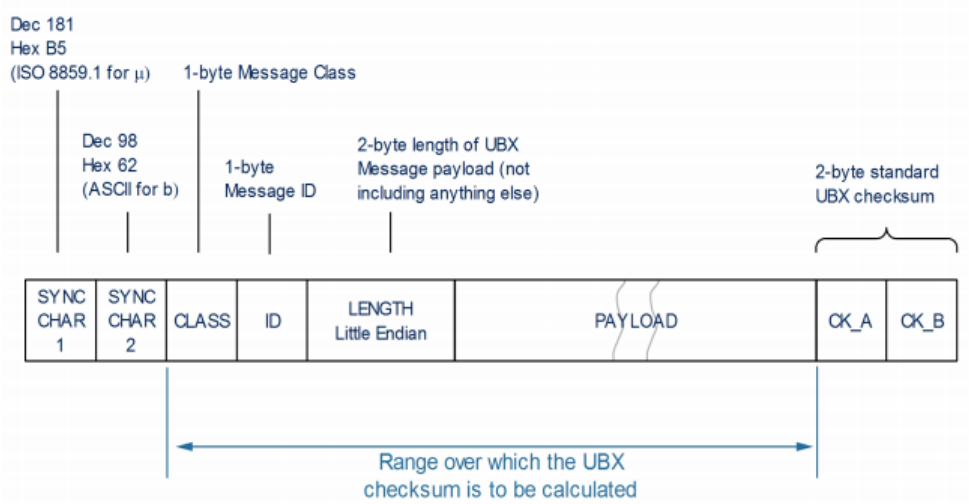
While the receiver supports the commonly used NMEA 0183 protocol, it is not used for the duration of this project and hence it is not explained in detail. The NMEA protocol is capable of some of the functionality provided by u-blox's UBX, but provides message outputs in string format.

An important feature of the u-blox receivers is the UBX proprietary radio protocol. The protocol enables users to communicate with the board using a host device, such as a computer, or to transfer information over a radio link. The structure of any given UBX message is given in Figure 3.2:

As per the M8 documentation provided by U-blox (2017), the UBX protocol uses 8 bit binary data. This protocol, is crucial for operating the receivers as:

1. Configuration of the receivers is completed using a list of UBX messages. This allows for the boards to be configured to the user's needs. Such examples include what navigation messages the receiver should transmit and accept, and over what particular interface.

Figure 3.2: The Structure of UBX Messages by U-blox 2017



2. Following from the previous item, the UBX protocol is similarly responsible for providing required navigation solutions from the receiver to the user. While in binary format, following the protocol description for any given message allows the information to be converted to a human interpreted format.

Any navigation information calculated by a u-blox M8 receiver is accessible to the user through the UBX commands outlined in the documentation by U-blox (2020a), through either the UART or USB interfaces described in 3.2.1. Therefore an understanding of the messages needed to evaluate the RTK performance was developed using the reference guides.

3.2.2.2 Radio Technical Commission for Maritime Services (RTCM) Protocol

With each evaluation of differential GNSS system, it was made clear the concept of 'correctional data' being sent from base to rover to enable cm level accuracy. It is important to mention the protocol used by u-blox is part of the RTCM radio protocol by Radio Technical Commission for Maritime Services (2020).

u-blox receivers use the RTCM protocol in both their absolute and relative positioning services to transmit correctional data. Each message used by the receiver serves a different purpose to enable the receiver to mitigate sources of error and bias. However, access

to the content of each message used in the RTCM requires purchasing the protocol, which totals more than €300.

Since a brief overview of the messages is available by Kalfus et al. (2010), an understanding of the messages used by u-blox, along with the general structure of RTCM messages in general was developed. A basic list of RTCM messages which a u-blox base station is capable of generating is found in Figure 3.3.

Figure 3.3: Example of C94-M8P Supported RTCM Messages by U-blox 2020a

Supported RTCM 3.3 Output Messages

Message Type	Cls/ID	Description
1005	0xF5 0x05	Stationary RTK reference station ARP
1074	0xF5 0x4A	GPS MSM4
1077	0xF5 0x4D	GPS MSM7
1084	0xF5 0x54	GLONASS MSM4
1087	0xF5 0x57	GLONASS MSM7
1124	0xF5 0x7C	BeiDou MSM4
1127	0xF5 0x7F	BeiDou MSM7
1230	0xF5 0xE6	GLONASS code-phase biases
4072, sub-type 0	0xF5 0xFE	Reference station PVT (u-blox proprietary RTCM Message)

Note that the class and ID field corresponds to the UBX messages which are used to enable/disable the output of these RTCM messages at receiver level.

3.2.2.3 u-blox u-center

'u-center' is a u-blox software tool, which grants the user a graphical interpretation of info contained in UBX messages, as well as to configure boards for RTK operation. Using the GUI allows for UBX configuration messages to be generated by the user using the click of a mouse, rather than a serial emulator with a string of bytes.

u-center displays data such as satellite orbit information and signal strength in a graphical manner, as can be seen in Figure 3.4

Since configurations can be saved to non-volatile memory on a receiver, this tool was key for configuring boards appropriately and easily.

Figure 3.4: Graphical User Interface (GUI) of u-blox 'u-center' by U-blox 2020a

3.2.3 C94-M8P Accuracy Evaluation

With the understanding of the underlying protocol used to generate base and rover positions, it was necessary to evaluate the accuracy of the u-blox boards in terms of absolute and positional accuracy. One note to make is that with a sailing application, a vessel is confined to the constructed race course. In reality, its position with reference to a global, ECEF frame is not as important as its position relative to the finish line. While this may be the case, the accuracy for both positioning methods is conducted, and compared to the provided accuracy estimates.

3.2.3.1 Absolute Positional Accuracy

The concept of absolute positional accuracy relies on the base station transmitting correctional data from a known, fixed location. In order for the base to reliably compare computed position measurements with its fixed position, the true location of the base must be known to a high degree of accuracy. Any error in approximation of the base's location will be carried into the navigation solutions for the rover as explained by U-blox (2017). There are a number of methods available to ensure this:

1. Use professional, survey grade equipment to find a highly accurate position for the

base.

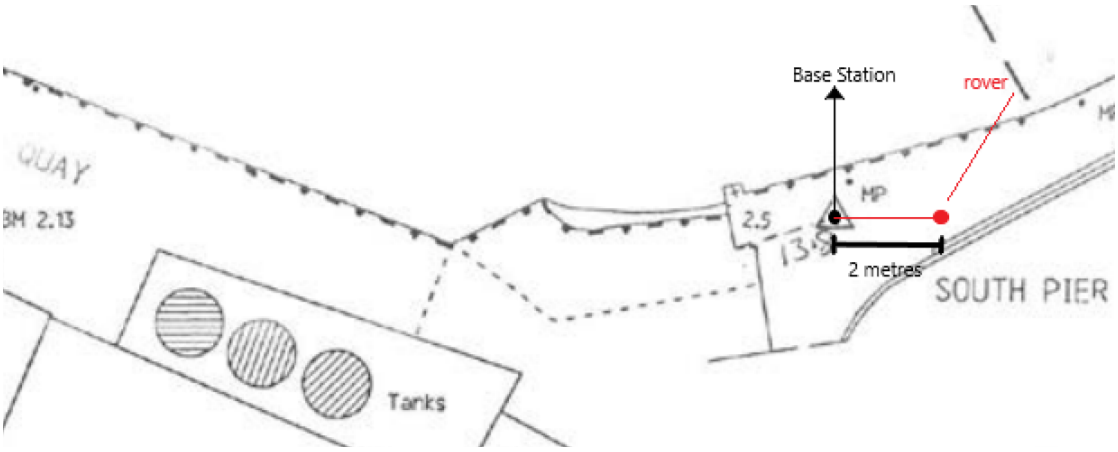
2. Use post-processing. Collect raw satellite observations and compare them to local government satellite observation data.

3. Use a reference point, with a location accurately known.

For a basic evaluation, and indeed the project as a whole, using surveying equipment is not a feasible option, due to cost. However both options two and three were examined together. Without a known reference point, determining whether or not the accuracy of post processed location is sufficient is not possible.

Ordinance Survey Ireland OSI created a number of reference points across the country, the accuracy of which was known to the centimetre level. Using a reference position in Arklow, Co.Wicklow, an experiment was set, using the configuration in Figure 3.5.

Figure 3.5: Absolution Position Experiment Setup. Adapted from Ordinance Survey Ireland (1995)



With the base set in a position of high accuracy, it would be assumed that the rover would receive appropriately accurate correction data, and hence would report a latitude and longitude which shows a distance of 2m from the base.

Taking a number of measurements and shown in Table 3.3

Clearly, with a maximum error of approximately 2.65 centimetres, there is confirmation in the advertised accuracy of the NEO-M8P. However, this experiment still required the use of an accurate base location, the nearest of which was found in Co.Wicklow. This

Base-to-rover Measured Distance (m)	Rover Reported Position (°)	Calculated Distance (m)	Error (m)
2.0m	52.7927217283°, -6.139450580°	1.98049m	0.01951m
2.0m	52.7927217297°, -6.139450607°	1.97347m	0.02653m

Table 3.2: Analysis of Absolute Position Accuracy of NEO-M8P

fact must be considered when designing the finish line system.

To evaluate the performance of post-processing, the reference point is again used. While placed at the reference point, the GNSS unit takes 40 minutes of raw satellite observation data. Again, using the resources of OSI, there is publicly available historical GNSS observations from stations around the country. With an observation station in Arklow, post processing can be applied. The tool used to conduct post-processing is know as 'RTKLib' developed by Tomoji Takasu, and is open source.

D161 Location	Post Processed Location	Error
52.79271561°, -6.139479904°	52.792725724°, -6.139467550°	1.3981m

Table 3.3: Analysis of Post-Processing to Determine Base Position

Clearly the post processed position reported by the tool suffers an intolerable error to the order of metres. Since this inaccuracy will be carried to the rovers position, it was concluded that using raw observations and post-processing would not guarantee and accurate base position. With this conclusion, the only remaining option to incorporate absolute positioning into the sailing application would be to use a professional survey.

3.2.3.2 Relative Positional Accuracy

Evaluating the relative position accuracy is a more straightforward process. Since u-blox advertises centimetre level accuracy in both a traditional base-rover and moving-baseline configuration, we can take advantage of the moving-baseline mode to relinquish the necessity for having an accurate, known position for the base. With this fact a simple experimental setup is created.

The base is placed at a fixed position, and using a measuring tape, the rover is placed at a set of measured distances away from the base. At each point, the UBX message which describes the relative position of the rover to the base in the NED local coordinate frame, "UBX-NAV-RELPOSNED", is examined. The RELPOSNED message provides a relative position in the form of North, East and Down offsets from the base in metres.

Since this frame is often used with aircraft, down is chosen over up for vertical offset. u-center allows for easy interpretation. The results of the experiment are found in Table. 3.4

Measured Distance (m)	North, East, Down Position (m)	Calculated Distance (m)	Error (m)
1.0m	[0.482m, -0.8619m, -0.0110m]	1.11m	0.11m
2.0m	[0.9368m, -1.7390m, -0.0148m]	1.975m	0.025m
2.0m	[0.9322m, -1.7463m , -0.0152m]	1.9842m	0.0158m
2.5m	[1.2318m, 2.1650m, 0.4294m]	2.527m	0.027m
3.0m	[1.4530m, -2.6272m, 0.4408m]	3.002	0.002m
30.0m	[-16.9444m, -24.7623m, 0.3783m]	30.007m	0.007m
60.0m	[-33.8421m, -49.5222m, 0.5476m]	59.9836	0.0164m

Table 3.4: Analysis of Relative Position Accuracy of NEO-M8P

Clearly, the relative positioning performs as advertised out of the box. A notable disparity was the measurement at 1 metre. There was a distinct increase in position error at ranges of 1 metre and less. While not a solution, specific positioning of a base station can prevent a base and rover (vessel) coming within 1m. To conclude, this was deemed to be a tolerable error, considering the centimetre level performance at all ranges $>$ 1m.

3.3 Finish Line System Design

With the conclusion that both absolute and relative positioning achieve the advertised centimetre level accuracy given the correct circumstance. A design could be built to facilitate autonomous race officiating.

3.3.1 Traditional RTK vs Moving Baseline RTK in a Sailing Application

The first consideration was of course the decision of using traditional RTK using a fixed base station, or the 'moving-baseline' using a non-stationary baseline.

3.3.1.1 Traditional RTK

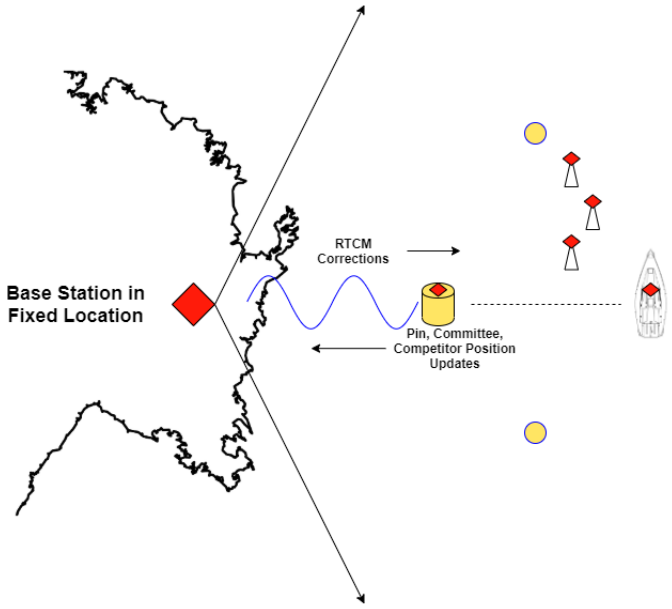
Emphasis has been placed on traditional RTK's requirement of a fixed point in an accurate location for either absolute or relative positioning. In a sailing application, the only guaranteed static location would be found ashore.

Considering the difficulty in achieving a reference point of sufficient accuracy for traditional RTK, this base location would require professional surveying to determine its position. While this would be possible in theory, this survey process would have to be repeated for **every** location this technology would be used.

Figure. 3.6 shows how a sailing race would use traditional RTK to officiate the finish line. The base ashore would broadcast RTCM correction data to all competitors, as well as the receiver on board both ends of the finish line (committee boat and finish pin).

To generate a finish line, an algorithm would have to be developed to monitor the ECEF position of both the committee boat and finish pin to generate a finish plane between their locations.

Figure 3.6: Configuration of Traditional RTK



The issue with this solution is its rigidness with respect to portability. This solution has to be tailored to suit every new location it is applied. Considering the moving baseline does not require a fixed base position, it can be placed on the committee boat itself. This allows for the solution to be more portable and excludes the need for surveying an accurate base position.

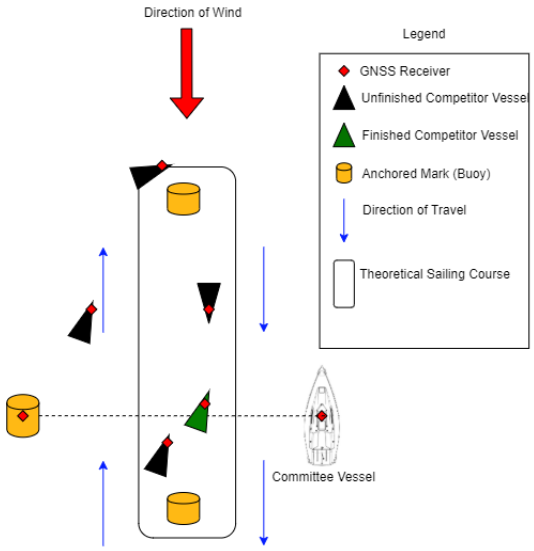
3.3.1.2 Moving Baseline RTK

The same conclusion can be drawn from the results from Section 3.2.3.2 as with Traditional RTK. It can be concluded that no disadvantage would be induced by choosing 'moving-baseline' operation. The system would rely purely on relative positioning. However, in the confines of a sailing race, this is appropriate, as it's only required to know a boat's position relative to the finish line, rather than a global ECEF frame.

3.3.2 The RTK Moving Baseline Finish Line

It was necessary to devise a system, using the capabilities of u-blox NEO-M8P. To illustrate, Figure 3.7 shows a typical 'windward-leeward' configuration for a sailing race. It is assumed that, for the scope of this project at least, the finish detection system will be situated on the committee boat, with intermediate data processing on the pin mark and competitors.

Figure 3.7: Basic Windward Leeward Sailing Course



The name windward-leeward describes the position of the two floating marks which define the course. Windward refers to the mark nearest the wind, with leeward referring to the mark downwind. Differentiating the two marks can be completed by using the 'Direction of Wind' arrow. However, what is important for this system is not the course configuration, but the run of operations at the finish line. The finish line can be created

using the vector describing the offset of the pin to the base. This is marked by the dotted line in Figure 3.7

With the relative position of the pin, combined with the relative position of any competitor, there were a number of options available to ascertain whether or not a boat has finished the race. A major design decision of this finish system, was where the computation of finish times were completed. Two options were available:

1. The competitor receiver, using its own UBX-NAV-RELPOSNED messages, which describes its position relative to the base, would determine whether or not it had crossed the line. Of course, the receiver would need to know the location of the pin also.

2. The base station will receive constant relative relative position updates from each competitor over the radio link, using the same UBX message. In this case, the base is the only receiver required to know the pin location.

Both methods would require a custom radio protocol, the design of which is discussed in a later part of this section. To evaluate the potential of both solutions, it was necessary to determine a number of line detection methods that could used with the relative position messages.

3.4 Detecting a Line Crossing

Since there is access to two boards only, a number of constraints were employed:

- Fixed Finish Line:

In theory in a sailing race, the line will not be fixed in terms of its relative position vector. A finish buoy is non-static. In ideal circumstances, more boards would allow for constant monitoring of the position of the finish buoy relative to the committee vessel, and hence updates to the position of the finish line.

- C94-M8P Moving Baseline Navigation Update Frequency Limit:

An important parameter of the moving baseline configuration is its limit on navigation updates over a wireless radio link. As per U-blox 2020a documentation, navigation updates, including the UBX-NAV-RELPOSNED message are limited to 1Hz over a

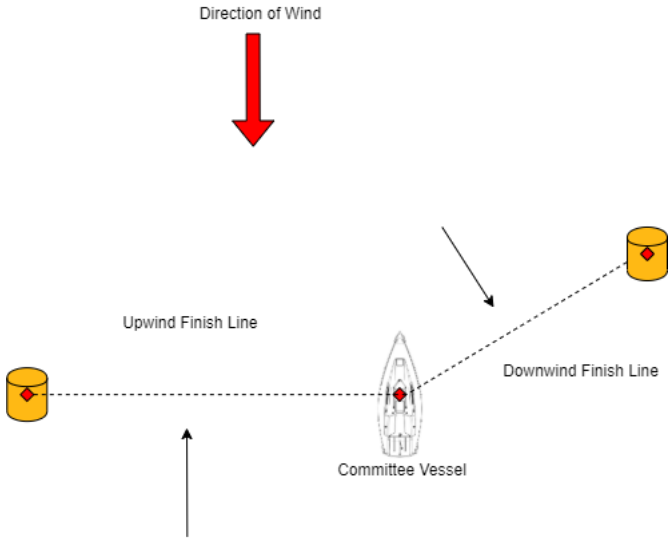
radio link. Therefore, it is pertinent to note, that these finish detection methods are capable of telling **if** a vessel has crossed the finish line, but further work, which is discussed later in this section, is required to determine the time instant the boat crossed the line.

With the finish line being fixed, it was possible to test the developed algorithms in both a simulated and real world environment.

3.4.1 Format of Finish Line

The orientation of the finish line in a sailing race is important when developing the finish line algorithms. Since many sailing clubs may employ local rules, different from each other, the orientation of the finish line may not be the same. This can be illustrated in Figure 3.8.

Figure 3.8: Alternate Finish Lines



Each finish line algorithm evaluated must account for varying finish line orientations.

3.4.2 Algorithms

The algorithms were evaluated based on the retention of accuracy from the obtained RTK receiver data. Since the UBX relative position message provided NED coordinates

as signed integers, maintaining integer arithmetic was desired. Pure integer arithmetic would avoid potential rounding errors when dealing with floating point numbers.

3.4.2.1 Angle Between Pin and Competitor Relative Position Vectors

To calculate the angle between the finish line and competitor, consider two UBX relative position vectors $\mathbf{u}$ and $\mathbf{v}$:

Knowing:

- Dot Product: $$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$
- Cross Product: $$\|\mathbf{u} \times \mathbf{v}\| = \|\mathbf{u}\|\|\mathbf{v}\| \sin\theta$$

$$\tan\theta = \frac{\|\mathbf{u} \times \mathbf{v}\|}{\mathbf{u} \cdot \mathbf{v}}$$

$$\theta = atan2(\|\mathbf{u} \times \mathbf{v}\|, \mathbf{u} \cdot \mathbf{v})$$

Knowing the angle between the vector describing a boat approaching the line, and the finish line itself allows for a finish line detection, as the angle experiences a 'zero-crossing'. The issue with a zero crossing arises when dealing with varying finish orientation.

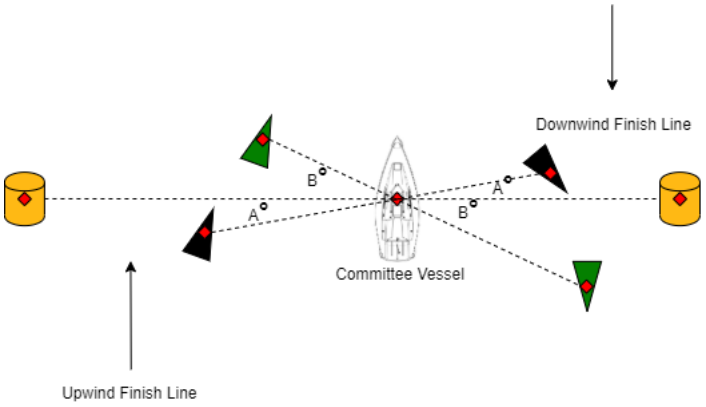
Note the atan2 function is an adaptation of a traditional arctan function, to overcome undefined radian angles such as $\frac{\pi}{2}$. Similarly, it allows for differentiation between points which are directly opposite each other in a 2D plane. The atan2 conditions can be seen below:

$$atan2(y,x) = \begin{cases} arctan(\frac{y}{x}), & \text{if } x > 0 \\ arctan(\frac{y}{x}) + \pi, & \text{if } x < 0 \text{ and } y \geq 0 \\ arctan(\frac{y}{x}) - \pi, & \text{if } x < 0 \text{ and } y < 0 \\ +\frac{\pi}{2}, & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2}, & \text{if } x = 0 \text{ and } y < 0 \\ undefined, & \text{if } x = 0 \text{ and } y = 0 \end{cases}$$

As mentioned in Section 3.4.1, this algorithm's design depends on the orientation of the finish line. Using Figure 3.9, and assuming the Bow of the committee boat is pointing

West, differences in angle values can be made using generated values.

Figure 3.9: Variation of Finish Line with Angle Algorithm



For the upwind finish line assume (ignoring the down component of the relative vector):

1. Unfinished Boat NED = -40m, 10m
2. Finished Boat NED = -30m, -20m

Therefore the angles A and B for the upwind finish:

$$A = atan2(10, -40) = 2.8966 \text{ radians}$$

$$B = atan2(-20, -30) = -2.3562 \text{ radians}$$

Note how the 'zero crossing' is from a positive radian angle to a negative radian angle.

For the downwind finish assume:

1. Unfinished Boat NED = 30m, -10m
2. Finished Boat NED = 30m, 30m

$$A = atan2(-10, 30) = -0.3218 \text{ radians}$$

$$B = atan2(30, 30) = 0.785 \text{ radians}$$

Note how the 'zero crossing' is from a negative radian angle to a positive radian angle.

Hence, this detection method, if implemented, would need know the variation of the finish line, to ensure the correct zero crossing is accounted for.

3.4.2.2 Point Above Or Below a Line

With a relative position message describing the location of the finish buoy, it is straightforward to define the equation of the finish line as $y = mx + C$. Since the intercept for both x and y will always be the origin, $y = mx$. The slope of the line can be found by division of the northern component of the relative vector, by the eastern component:

$$y = \frac{E_{finish}}{N_{finish}} x$$

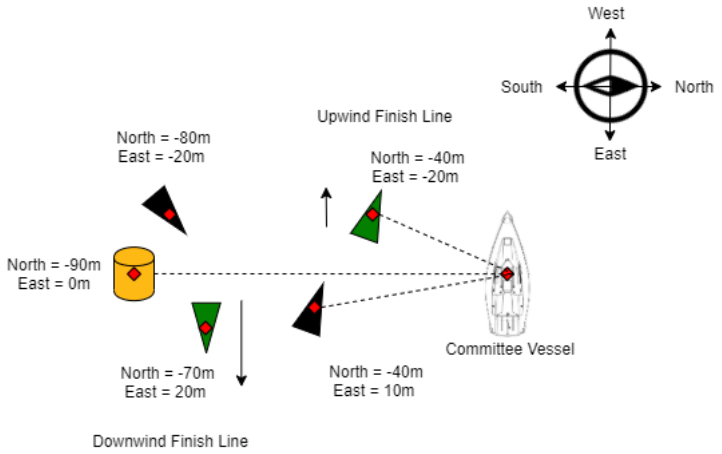
A point describing a boat's relative position (N_{boat}, E_{boat}) is deemed to be above a line, such that:

$$E_{boat} > \frac{E_{finish}}{N_{finish}} N_{boat}$$

$$E_{boat} N_{finish} > E_{finish} N_{boat}$$

As with Section 3.4.2.1, this algorithm must account for the variation orientation of a finish line. Figure 3.10 illustrates this implication.

Figure 3.10: Variation of Finish Line with Point Algorithm



Using the relative offsets provided, we can check the validity of the equation for the downwind finish on the left and upwind on the right.

1. Downwind Finish

Calculating for the Unfinished Boat:

$$E_{boat}N_{finish} > E_{finish}N_{boat}$$

$$(-20)(-90) > 0$$

Clearly, this statement is true. Calculating for the finished boat:

$$(20)(-90) > 0$$

Conversely, the statement is false for the finished boat.

1. Upwind Finish

Repeating the same process results in the following observations:

- Unfinished Boat = False
- Finished Boat = True

Clearly, the orientation of the inequality in this equation depends on orientation of the finish line. Therefore both:

$$E_{boat}N_{finish} > E_{finish}N_{boat}$$

$$E_{boat}N_{finish} < E_{finish}N_{boat}$$

Are valid. Therefore, it will be part of the design to account for both cases appropriately. This algorithm proved to be the only available to maintain purely integer arithmetic, and is planned to be utilized as the primary detection method.

3.5 Interpolation

As emphasised in Section 3.4, the line crossing algorithms are capable of detecting if a boat has crossed a line, but not the exact time instant. This is mainly due to the 1Hz navigation update limit for the relative position messages. Unless one of the algorithms which quantifies proximity to the line reports a result which is infinitely close to the finished side of the line, it is not possible to determine the exact time the boat crossed the line.

Therefore, a method of interpolating results is required. The structure of this interpolation method would remain the same in the case of either the competitors making local finish time calculations, or the base receiving positioning messages from each rover to conduct a central interpolation process. As defined in the UBX protocol, the UBX relative position message contains a timestamp, along side its NED vector.

Since sailing vessels can move quickly, it would not be uncommon for noticeable changes of speed as well as heading to occur within a one second time period. Since the method in Section 3.4.2.2 provides a boolean result (above/below line), it is not possible to interpolate. Hence, depending on the interpolation technique, a number of relative positions, and their associated timestamp each side of a finish must be stored, to conduct a successful interpolation process.

Since the NED vector itself does not quantify proximity to the line, the shortest distance method below is used for experimentation and visualisation.

3.5.1 Shortest Distance From Boat to Finish Line

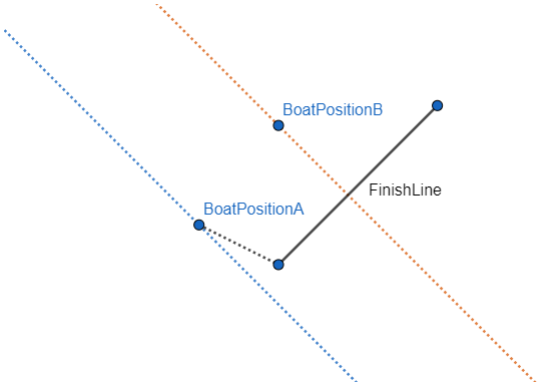
Since the finish line is a finite line segment, the shortest distance from any point on the race course to the finish line may not necessarily be the perpendicular distance, as can be seen in Figure 3.11

Defining the location of boats:

$$A(A_x, A_y), B(B_x, B_y)$$

The finish line is defined as the line segment from [0,0] to the pin location, defined as

Figure 3.11: Sample of Shortest Distances To A Sailing Finish Line



F_x, F_y :

$$[(0,0),(F_x,F_y)]$$

The parametric equation of the line which passes through the finish line segment is given by;

$$v = \begin{bmatrix} F_x t \\ F_y t \end{bmatrix}$$

Where t describes how far along the line a point lies. Finding the squared distance between boat B and the line and solving for t:

$$t = -\frac{-B_x F_x + (-B_y)F_y}{(F_x)^2 + (F_y)^2}$$

In the case where $1 \leq t \leq 0$, then it can be confirmed that the projection from the boat to the finish line, lies within the segment, making the perpendicular distance the shortest distance. If t lies outside this range, as it will with Boat A, the shortest distance will be one of two end-points of the line segment.

Perpendicular Distance from Boat B to Finish Line:

$$d = \frac{|F_x(-B_y) - F_y(-B_x)|}{\sqrt{(F_x)^2 + (F_y)^2}}$$

The implementation of this shortest distance method in Python, as can be seen in

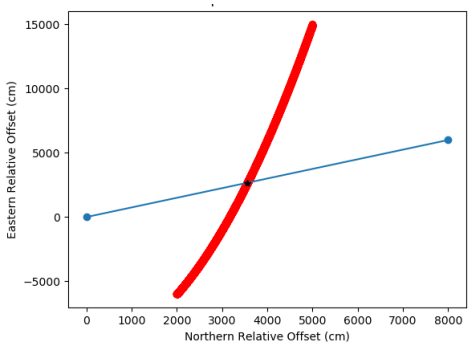
Appendix 7.2 uses a variation of set of formulae by Kessen (2020).

3.5.2 Linear Interpolation

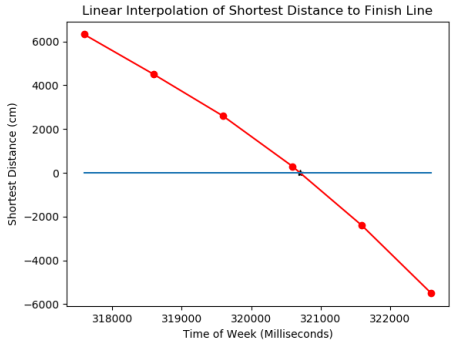
The benefit of linear interpolation is the necessity to store only a single position message either side of a finish line crossing. Creating a linear function to describe the direct path between the two points and finding the point of intersection with the finish line at that interval allows an approximation for the timestamp.

The main draw back of this method is the processes assumption that the boat will travel in a completely linear path between the two position updates. While the time interval is a single second, vessels have the ability to make course corrections at considerable speeds. As a result it is import that both non-linear and linear interpolation and examined.

Figure 3.12 gives an example of both the simulated course of a boat as it crosses a finish line, as well as a linear interpolation calculation, using the shortest distance method and the embedded UBX message timestamp.



(a) Example of Simulation of a Finish Line Crossing using NED Vector



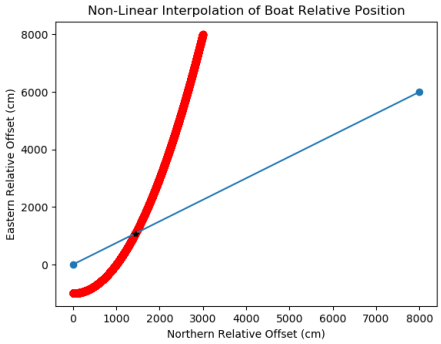
(b) Example of Linear Interpolation Calculation Using Shortest Distance & GNSS Receiver Timestamps

Figure 3.12: A Sample Linear Interpolation Simulation

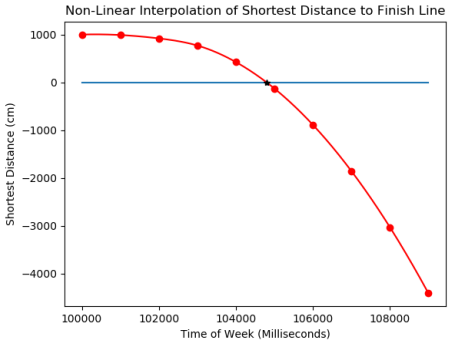
3.5.3 Non-Linear Interpolation

Non-Linear Interpolation aimed to more closely account for the non-linear nature of a boat's movements as it approaches the finish line. As a result, a spline curve was chosen to fit the data. As previously mentioned, to conduct non-linear interpolation multiple

data points are required. Spline approximations need a minimum of 4 points to create the polynomial used. However, the more data points available, the more closely it would be expected the interpolated curve would represent the approach of the finishing boat.



(a) Example of Simulation of a Finish Line Crossing using NED Vector



(b) Example of Non-Linear Interpolation Calculation Using Shortest Distance & GNSS Receiver Timestamps

Figure 3.13: A Sample Non-Linear Interpolation Simulation

3.5.4 Linear vs Non-Linear Interpolation Test Cases

In order to compare the feasibility of both linear and non-linear interpolation as a means of determining a vessels finish time, a number of experiments were created. Like in Figure 3.12 and Figure 3.13, a set of course headings for a boat crossing a finish line were created. With thousands of data points, the actual time of crossing the line is known to the nearest millisecond. A sample of RELPOSNED messages is selected such that a 1Hz update rate exists.

While linear interpolation requires only two points, the same subset of data is applied in both linear and non-linear test cases for visualization purposes. Test cases are created to test the performance of the interpolation methods and as a result use tactics which may be uncommon when approaching a finish line in a real race. The implementation of the interpolation methods to determine the following results can be found in Appendix 7.3.

Clearly, with three out of four cases, linear interpolation performed better than its non-linear counterpart. Calculating the mean error of each interpolation method:

True Finish Time (ms)	Linear Interpolation (ms) (ms)	Non-Linear Interpolation (ms)
120965.3695	120977.577	120983.766
104797.171	104790.885	104796.855
577936.860	577937.126	570892.7514
302885.604	302867.414	302878.579

Table 3.5: Linear vs. Non-Linear Interpolation Simulations

Linear Interpolation Error(ms)	Non-Linear Interpolation Error(ms)
12.2	18.397
6.286	0.316
0.266	44.109
11.81	22.975

Table 3.6: Linear vs. Non-Linear Interpolation Simulation Errors

$$\text{Mean of Linear Interpolation Error} = 7.64ms$$

$$\text{Mean of Non Linear Interpolation Error} = 21.449ms$$

As a result of the performance of linear interpolation along with the requirement of only two relative position points, each side of the finish line, linear interpolation is chosen to be implemented.

3.5.5 Practical Testing of Finish Detection and Interpolation

With the choice of both finish detection method and interpolation method, it was relevant to combine both methods to determine the overall accuracy of the detection system. To do so, a practical test was created. A land based, real-time test was devised.

3.5.5.1 Experimental Setup

Constrained to only two RTK receivers, a fixed finish line must be implemented. To compute this, the rover is placed stationary, at the end of the line and the corresponding UBX-RELPOSNED messages is extracted and stored.

All computation is completed on the rover side. Since the radio protocol has not been developed during the proof of concept phase, the most straightforward method of conducting this experiment was to follow a set of steps:

1. The laptop is connected to the rover via USB.

2. Using the pre-configured radio link, the base broadcasts RTCM messages at 1Hz.
3. At the rover, the pre-configured link accepts RTCM messages and communicates them to the receiver, without the need for external changes.
4. Imitation of a boats approach to the finish line is made by a human, combining walking, jogging and running.
5. Using the USB link and the simulation code found in [LINK TO APPENDIX PY FILE], the UBX-NAV-RELPOSNED message is enabled and extracted to the laptop processor.
6. The Python script computes the finish detection and interpolation processes as outlined in Sections 3.4 and 3.5. Results are displayed on the laptop screen rather than transmitted over the radio link

Once finish times are approximated by the interpolation process, they are compared with the captured crossing on a 60 frame per second recording device, Note results are hence accurate to the nearest frame.

3.5.5.2 Experimental Results

Test	Camera Finish Time (hh:mm:ss)	Interpolation Finish Time	Error (s)
1	09:49:23.418	9:49:23.313	0.105
2	09:54:30.0	09:54:30.011413	0.011413
3	09:57:14.729	09:57:14.70434	0.02466
4	10:03:42.12	10:03:42.1351	0.0151
5	10:08:29.12	10:08:29.179	0.059

Table 3.7: Video Captured vs Interpolated Finish Times

Table 3.7 showcases the performance of linear interpolation in a real-time application. Interpolated times are compared with camera calculated results. Camera results are interpreted to the nearest frame, using a 60 frame per second device.The average error across the tests was 0.043 seconds. This confirms that the full end-to-end system is capable, at the least, of determining finish times to the nearest tenth of a second.

3.5.6 Finish Detection & Interpolation Conclusion

It is important to note, that the finish detection calculations were conducted at the rover, using a laptop. In reality, using a such a large device to conduct finish line detection

and processing is not feasible. Similarly, the results calculated are stored locally, on the laptop at the rover. In the case of a real race, if the calculations were being conducted locally on the rover, the base would need to be notified via the radio link.

Therefore, a decision on radio link protocols is discussed in the next section, to ascertain in what configuration the finish detection methods will be applied.

3.6 Communication Link Design

With a proof of concept validated in the practical tests found in Section 3.5.5, a suitable radio link is required to successfully transmit and receive the information required to run a completely autonomous system.

3.6.1 Location of Finish Detection Processing

As mentioned in the previous section, the interpolation and finish techniques are developed to cover two potential cases:

3.6.1.1 Base Finish Processing

In this case, the base station conducts all finish detection and interpolation processes locally. In order to complete this task, the base station requires relative position updates from **every** vessel once per 1Hz transmission cycle, as well as the location of the pin end buoy. The design of the protocol may dictate that a vessel only transmits once it is within a certain proximity, by examining its own UBX message. In any case, the base may be required to receive the relative position of tens of vessel each second.

With each position message received, the processor, which, given the availability of space on the committee boat can be a laptop, which must run a finish detection process. If a boat is found to be finished, linear interpolation for the finished boat must be executed to find a finish time. As part of the protocol design, it would be expected that the committee boat would acknowledge a vessel's finish by sending an addressed acknowledgement message.

The main consideration with this concept is the complexity in data transmission. If every vessel is transmitting their location at a 1Hz rate, the chance of packet collisions,

even with a suitable access protocol, is going to be high. With a potential for tens to hundreds of boats to be involved in a race, maintaining an organised system to accept tens of position messages and in turn issue acknowledgments to each sender, in a 1Hz cycle would prove to be a complex system.

3.6.1.2 Rover Finish Processing

Alternatively, in this case, finish detection and interpolation are completed locally on the rover. With limited space on a rover, the use of a microcontroller would be required to handle processing and calculation. The relative position of a rover is generated locally by their respective receivers, so access to the data is as soon as it is available.

In order to compute finish detection, the location of the pin, and hence the plane describing the finish line is required. Once a finish time has been calculated on board the rover, it is necessary for the information to be transmitted back to the base at the committee boat, in order for the committee to accumulate a list of results. In comparison to sending a relative position once per second, this method requires only a single message and acknowledgment per boat for any given race. Assuming no collisions, there is a considerable decrease in radio link traffic, and as a result, a decrease in the potential for packet collision.

3.6.1.3 Finish Detection Processing Conclusion

With the reduced complexity in message traffic and medium access, it was decided to pursue a rover based finish detection solution. While this increases processing requirements, it reduces the overall traffic in the system, and hence the overall probability of packet collisions

3.6.2 Radio Link Hardware Considerations

In order to develop a suite of software functionality, the overall requirements of the radio link to act as an autonomous finish line must be evaluated. The hardware design considerations of the protocol to run the radio link is dependant on two of key factors:

1. The required rate of data transmission, or bit rate when dealing with binary information is dictated by the nature of the message sent over the radio link.

Along with the expected RTCM messages, custom messages from both the base and rover will be required, influencing the overall suitable bit rate.

2. An internal packet engine capable of sending packetized, information. Radio Packets allowing for embedding information like addressing and broadcast addressing.

The first step in understanding the data rate requirements is to evaluate the data size of all message types used on the link.

3.6.2.1 RTCM Correction Stream

As outlined, the centimetre level relative positioning accuracy of RTK in a moving baseline application is dependent on a consistent stream of RTCM corrections. As limited by the C94-M8P, all navigation updates are limited to 1Hz, including RTCM.

Since access to the RTCM protocol is not available, experimentation was conducted to determine the minimum and maximum observed message sizes for each message used in the finish line system. Referencing Figure 3.3, the messages required for moving baseline operation, using GPS and GLONASS, as outlined by U-blox (2017), are: 1077, 1087, 1230, 4072.0.

Examining the size of each message over 1000 transmission cycles, Table 3.8, shows the minimum and maximum payload size of each message.

Message Number	Minimum Payload Size (bytes)	Maximum Payload Size (bytes)
1077	80	124
1087	80	139
1230	6	6
4072.0	132	132

Table 3.8: Minimum & Maximum Observed RTCM Payload Sizes for 1000 1Hz Cycles

It is important to note, as outlined by Kalfus et al. (2010), that while the payload contents are unknown, it is known that 3 bytes of overhead and 3 bytes of CRC exist for each message. Taking a maximum size scenario, accounting for overhead, the base would be required to transmit approximately 401 payload bytes and 24 header bytes, over four messages, once per second.

3.6.2.2 Committee, Pin & Competitor Custom Messages

The remaining traffic on the radio link is a combination of status messages from the committee boat, location updates from the pin buoy, finish time notifications from competitors and acknowledgements from the base.

- Committee Vessel Status Message

The purpose of the status message of the committee boat is to both provide information to the competitors on the current status of a given race, as well as to provide the location of the finish pin. As outlined in Figure 3.22 the pin location is addressed to the base. As a result the base will then broadcast to all competitors within range. Figure 3.14 outlines the structure of the status message. Note that the structure of the components of the relative position of the pin remain the same as the UBX protocol.

Committee Status Byte 1	Committee Status Byte 2	Pin Northern Offset Component 32 bit Signed Integer 4 bytes	Pin Eastern Offset Component 32 bit Signed Integer 4 bytes

Figure 3.14: Structure of Committee Boat Status & Pin Message

Status bits contain boolean (0 for No/ 1 For Yes or similar) information. Examples such as:

1. Is there an active race?
2. Is there a valid finish line constructed? i.e. Is there a valid pin position being broadcast.
3. What is the orientation of the finish? (Downwind/ Upwind)

The status bits required may vary from use case to use case, and as a result, 2 bytes are made available, totalling, at minimum, 16 boolean notifications.

- Competitor Finish Time Message

The only message transmitted from the rover for the duration of a race, is to inform the committee boat of its finish time. Figure 3.15 shows the structure of the **payload** of the finish time message. Using this message, the base station on board the committee boat can identify each boat and their respective finish time, creating a list of results

autonomously.

It is not necessary for a vessel to transmit its finish time in the same second it finishes. As long as it remains within transmission and reception range of the committee boat, the competitor arguable has perhaps minutes to send.

Competitor ID Byte 1	Competitor ID Byte 2	Competitor Finish Time 32 bit Unsigned Integer 4 bytes

Figure 3.15: Competitor Finish Time Message

The maximum case scenario, although very highly improbable is the event that all boats in the race finish at the same time instant. Since the actual number of vessels in a race will vary the exact number of simultaneous boats finishing is not possible to predict. Assuming a case where 100 boats finish and transmit their time in the same transmission cycle, given a sufficient medium access protocol ensure no packet collisions, there is potentially 600 bytes transmitted.

- Pin Location Update

As the name may suggest, the purpose of this message is to update the base with the most recent location of the pin. The base is then capable of broadcasting the updated pin location in the next transmission cycle, allowing the competitors to conduct up to date finish detection and interpolation. Figure 3.16 shows the payload of a pin update message. Note the North and east relative offset components maintain their structure as in the UBX protocol specification.

Pin ID Byte 1	Pin ID Byte 2	Pin Northern Offset Component 32 bit Signed Integer 4 bytes	Pin Eastern Offset Component 32 bit Signed Integer 4 bytes

Figure 3.16: Structure of Pin Location Update Message

This message is only transmitted once per second, adding a payload of 10 bytes per transmission cycle.

3.6.2.3 Message Size Conclusion

Combining the size of all potential traffic on the radio link; the radio link, in theory, must be capable of handling 1035 payload bytes per second. Accounting for the header and trailers applied by the modulation technique any hardware module, the amount of bytes could increase considerably depending on the number of unique messages. Therefore, as a minimum requirement, a module with bit rate capabilities with at least 20,000 bits per second, was considered. While this is more than a potential worst case scenario, considerable 'leeway' is given, when considering the implementation of a MAC protocol and the addition of header bytes with a hardware packet engine.

3.6.2.4 Point to Multi-point Network Capabilities & Packetization of Radio Link Traffic

With a wide variety of messages on the radio link, a method of addressing, when applicable was required.

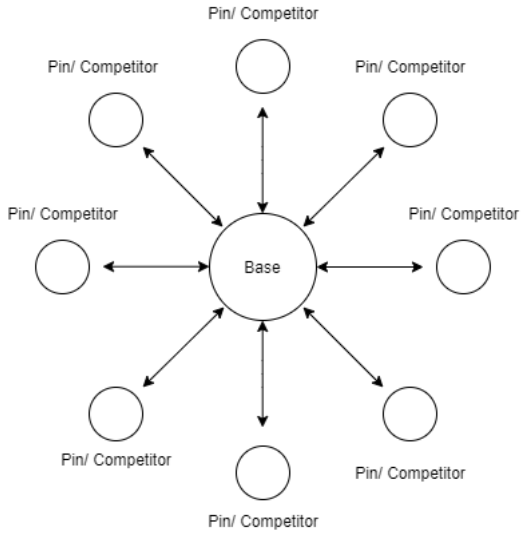


Figure 3.17: Topology of Required Network

Figure 3.17 shows the topology required to handle all communication traffic on the radio link. All competitors, as well as the pin, should be capable of transmitting and receiving information from the base station system on the committee boat only. As a result, competitors must be capable of differentiating messages they are required to receive such as RTCM and status byte updates.

Therefore, a module capable of packetizing messages, with the inclusion of source and destination addressing, including broadcast message is required. Since addressing can be software based, many modules can handle this requirement. Since many of the RTCM messages are often greater then 130 bytes, a module with a packet engine to support such a payload is required.

3.6.2.5 Implication of Radio Link Range on RTK Receivers

Another important consideration in choice of hardware is the range at which the competitors can receive messages. This is not vital for finish updates, since the competitor will be within about 100 metres of the committee boat.

However, for RTCM correction data it is more important. In order for an RTK receiver to maintain an RTK fix, it is required to have a stream of correction data, even in moving-baseline operations. If a stream of correction data is lost, requisition time for centimetre level accuracy is about 10 seconds as outlined by U-blox (2017). Many sailing yachts are capable of travelling up and greater than 15 knots, or approximately 8 metres per second. In a ten second time frame, a competitor could travel up to 80 metres before acquiring centimetre level accuracy. As a result, ranges up to 500 metres plus were desired,

3.6.3 Existing C94-M8P Radio Link

The existing radio link on the C94-M8P board uses a HM-TRP radio by HOPERF (2020) module with custom firmware. The firmware is based on an open source design, by ArduPilot (2018). This firmware was modified by ublox to be used in the C94-M8P evaluation kit. This firmware however supports point-to-point communication only, hence the boards being sold as a preconfigured pair.

In order to utilize the radio modules to work in a multi-point configuration, modification to the existing ublox firmware or the creation a completely new firmware was required. Since there was no access to the source code of the modified firmware, developing a new firmware was the only remaining option. In light of this, alternative radio modules were sought, with sufficient documentation.

3.6.4 Adafruit RFM96W LoRa Radio Transceiver

In light of the decision on the existing radio link, the Adafruit RFM96W breakout board was chosen as the module of choice. The RFM96W is a HOPERF chip based on the Semtech SX1276 chip. The module operates in the 433MHz band, and is based on Semtech's LoRa modulation technique.

3.6.4.1 LoRa

LoRa, short for long range, is a modulation technique developed by Semtech (2020), based on the concept of chirp spread spectrum CSS technology. The basis for LoRa is a long range, low power application as is desired in the context of the finish line system. With LoRa, symbols of information are modulated as chirps, as can be seen in Figure 3.18.

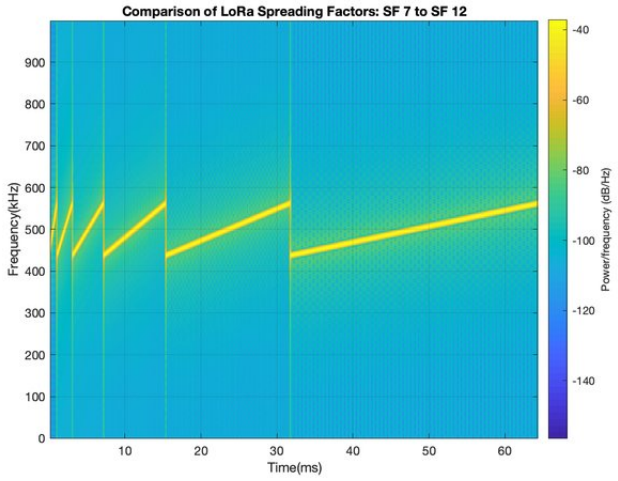


Figure 3.18: Length of Chirps with Varying Spreading Factor by D.-H. Kim, Lee and J. Kim (2019)

LoRa supports lower data rates than many other radio systems, to prioritise low power usage and long range requirements. The physical limit of most LoRa enabled modems is 37.5kbps. The data rate capabilities of LoRa depend on several factors:

- Spreading Factor: The Spreading factor is an indication of the length of each chirp or symbol of information in a LoRa modulated transmission. Therefore, the shorter the chirp, the faster the data rate. LoRa modules can vary spreading

factors from 6-12

- Code Rate: The code rate refers to the proportion of bits in a symbol/chirp that carry information to the number of parity bits for error correction. The larger the code rate, the faster the data rate, with a reduction in the probability of recovery in the case of error.
- Bandwidth: The bounds of frequency which the chirp occupies. LoRa modulation offers between 7.8-500KHz. A larger bandwidth implies more bits per chirp in this case, hence an increase in data rate.

The data rate of a LoRa modem can be calculated using :

$$DataRate = SF \times \frac{[\frac{4}{4+\mathrm{CR}}]}{[\frac{2^{SF}}{BW}]} \times 1000$$

Where SF = Spreading Factor, CR = Coding Rate and BW = Bandwidth

Therefore, SF = 6, CR = 1 and BW = 500KHz gives a data rate of 37.5kbps.

3.6.4.2 Hardware Specification Requirements

When comparing the specifications in the manual by HOPERF (2020) to the requirements outlined in Section 3.6.2 :

1. Packet Engine: The module supports packets up to 256 bytes with its internal packet engine, which sufficiently covers even the largest RTCM message.
2. Data Rate: The maximum data rate of the module, when using the LoRa modulation technique is 37.5kbps, which is more than sufficient to cover the 20kbps rate specification in the hardware requirements.
3. Packetization allows for the development of software based addressing, hence allowing this module to support a point-to-mulitpoint topology as required.
4. Communication with the module is completed over a serial peripheral interface (SPI) protocol, which is common to many microcontroller chip sets and development boards

3.6.5 Microcontroller Considerations

The main requirements for the microcontroller relate to suitable interfaces for connectivity with both the RTK receiver and radio module. With the decision on the radio module made, an equivalent microcontroller is required to interface with the SPI interface of the radio module, as well as the UART interface of the C94-M8P for receiving and delivering RTCM and UBX messages. The microcontrollers on board both the base and competitors are responsible for message processing from the SPI radio, finish detection and timing requirements as per the MAC protocol defined in Section 3.6.7.

3.6.5.1 Universal Asynchronous Receiver Transmitter (UART)

UART is a physical circuit, found on many devices, with the sole purpose of transmitting and receiving serial data. Figure 3.19 illustrates a connection between two UART devices, which in this scope of this project would be the connection between the UART connection on the C94-M8P and the microcontroller.

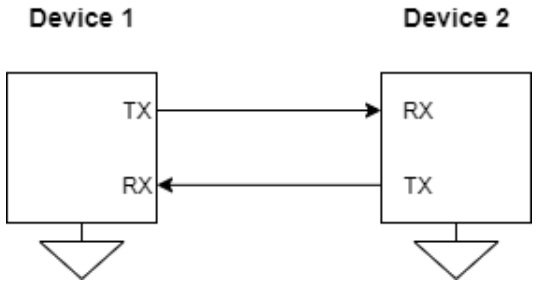


Figure 3.19: Example of Connection Between Two UART Circuits

The speed of data transfer, i.e. serial bits, of a UART is expressed by the baud rate. For example, the maximum baud rate of the C94-M8P over the UART as defined by the manual, is 115200 baud measured in bits per second. UART, as the name suggests is asynchronous, and does not rely on a common clock. Instead, start and stop bits are added to data packets during transfer. UART only requires two wires for connectivity

3.6.5.2 Serial Peripheral Interface

Unlike UART, SPI can send data in a continuous stream, rather than in packets. SPI is one of the most commonly used interfaces between microcontrollers and peripherals. In the case of this project, a microcontroller and the LoRa module. The protocol works

in a 'Master' 'Slave' configuration.

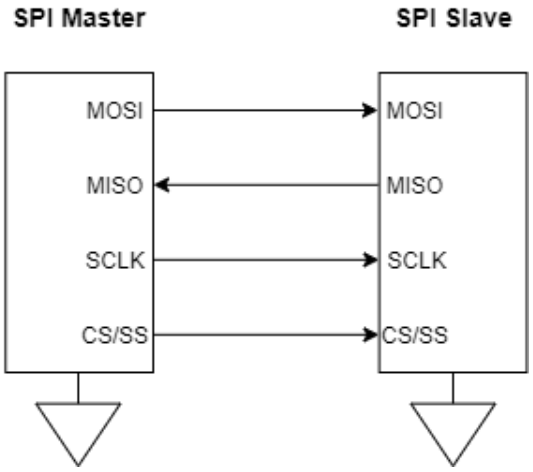


Figure 3.20: Example of Connection Between SPI Master & Slave

Figure 3.20 shows the connectivity between a SPI master (the microcontroller) and slave (LoRa Radio Module).

- MOSI: Master Output, Slave Input
- MISO: Master Input, Slave Output
- SCLK: Clock for transmission synchronisation
- SS/CS : Slave Select/ Chip Select. Used to select a slave in the case where a master has more than one slave.

While SPI uses four wires, unlike UART's two, the ability to stream data continuously allows for considerably faster transmission speeds. SPI is capable of up to 10Mbps.

3.6.5.3 Adafruit Feather M0 Express

The Feather M0 by Adafruit was the perfect option due to the following specifications:

1. UART Interface for communication with C94-M8P.
2. SPI Interface for LoRa Radio Communication.
3. USB port for power and debugging. The USB port also functions as a Virtual COM port for Serial Data transfer if required.

4. Supports an important library used in the handling of the radio module at software level. The details of which are discussed in Section 3.6.6

3.6.5.4 Feather M0 UART Buffer & Introduction of Laptop Processing at the Base Station

As examined in Section 3.6.2.1, the number of RTCM correction bytes seen at the instance of a new second, indicated by the receiver timepulse may exceed 400 bytes. A decision was made, to simplify the extraction of each individual RTCM message by using the RTK's USB port and a Python script running on a laptop. The main reason for this is due to the limitations of the buffer experienced on the Feather M0. When reading RTCM data, the buffer experienced overwriting issues.

As a result, Figure 3.21 shows hardware configuration is found at the base:

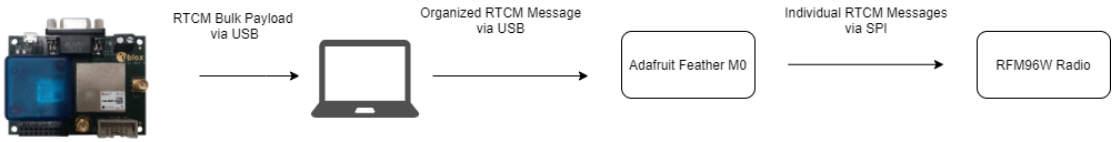


Figure 3.21: Hardware Configuration at the Base

As a result, there is an intermediate step added. While this introduces a short amount of latency, this effect works favourably in the context of the medium access protocol, explained in Section 3.6.7.

3.6.6 AirSpayce RadioHead Driver Library

An important tool used in the development of handling of the SPI radio modules at a software level is the use of the RadioHead Driver Library by AirSpayce (2020). The RadioHead library is designed for a subset of radio and microcontroller modules, of which both the Adafruit Feather M0 and RFM96W is included.

The library provides an object-oriented approach to the transmission and reception of packetized, addressed messages. The library provides classes for driver and manager functionality, the details of which are described below.

3.6.6.1 RFM95 Driver

The 'driver' is responsible for low-level interaction with the SPI radio module, to provide functionality such as:

- Initializing the radio with the desired specification (Transmit Power, Frequency)
- Adjust the mode of operation in the modem (Transmit/ Receive)
- Interaction with First in First Out Buffer.

Note that the name RFM95, applies to the variety of HOPERF radios which use the same LoRa modulation techniques, but at varying frequencies. RFM96W is for the 433MHz band for example.

This functionality alone can be used to develop a custom driver, capable of packetized message communication. However, RadioHead has developed managers to handle addressed and acknowledged message functionality, which this project takes advantage of. As a result, Appendices 7.5, 7.7 and 7.6 uses the manager functionality at the top level.

3.6.6.2 Reliable Datagram Manager

The 'Reliable Datagram' manager provides functions to interact with the radio driver to create a network of addressed packet transmissions with the option for acknowledgments. In order to create a software based addressing scheme, the reliable datagram manager uses 4 of the 256 available bytes to create a set of software headers. As a result only 252 bytes are available as intended payload, which is still well exceeding the maximum observed size of an RTCM message. The four header bytes are:

1. TO: The address of the receiver to which the message is intended. Setting the value of this byte to hexadecimal 0xFF indicates a broadcast message, to which all recipients in the network will accept
2. FROM: The address of the sender
3. ID: A distinct Message ID, for each message sent from a particular sender.
4. FLAGS: A reserved byte for RadioHead status bits

In the case of acknowledgment messages, the manager module sends the same set of overhead bytes, along with a single character byte "!" to indicate to the receiver their

message has been accepted. This function allows the base station to acknowledge the finish times of competitors.

3.6.7 Medium Access Control

With a chosen radio, microcontroller and software library, creating an radio-link access protocol was the final task to ensure reliable delivery of RTCM corrections, pin location updates and competitor finish times.

3.6.7.1 RTK Receiver Timepulse

As has been mentioned numerously, the RTK receivers operate at a 1Hz navigation update rate. To ensure the frequency of message is maintained at 1Hz, the receivers use their on board clocks which are all synchronised with the atomic clocks of the same set of satellites.

As a result, both RTCM at the base receiver and the UBX-NAV-RELPOSNED message at the competitor receiver will be generated at the same time instant, i.e. the rising edge of the clock signifying the beginning of the next 1Hz transmission cycle.

With the introduction of the brief latency at the base when processing RTCM and transmitting over the radio module, this gives both the competitors and the pin receiver sufficient time to accept the UBX relative position message over UART in preparation for processing, before returning to accept RTCM corrections.

Similarly, this timepulse is available as an external output on the RTK receiver, which vitally allows the microcontroller to sync with the receiver at the beginning of each transmission cycle. This is completed by connecting the timepulse output to a digital IO pin on the microcontroller, and inducing an Interrupt Service Routine in the microcontroller process, to inform about the beginning of a new transmission cycle.

For example, the competitor controller will then be ready to first handle its relative position message over UART, then prepare for incoming RTCM data over SPI.

3.6.7.2 Message Flow Overview

Figures 3.22 and 3.23 show the data exchanges over time, for both a base station and rover.

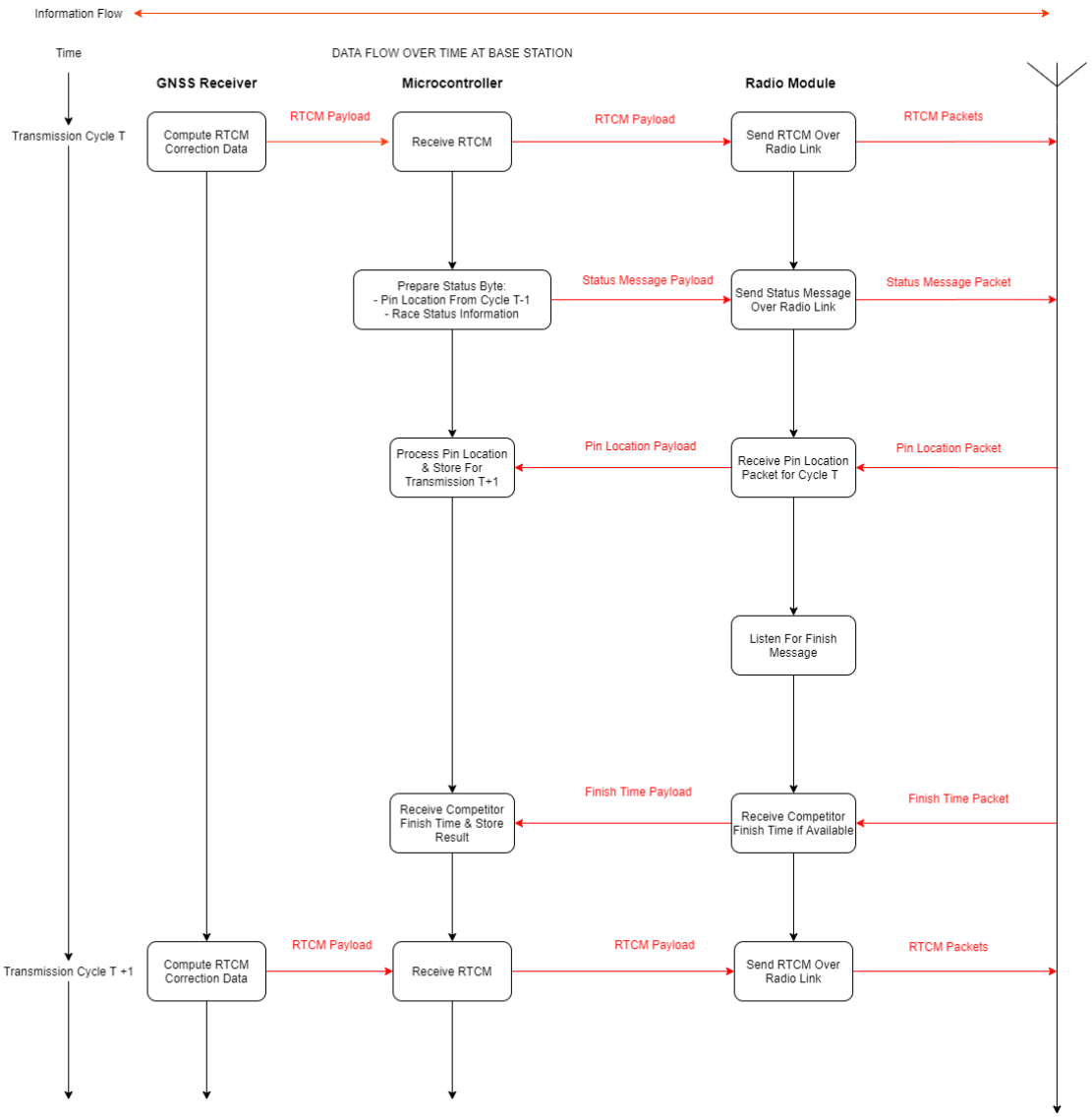


Figure 3.22: Flow Diagram for Transmission Cycle At the Base

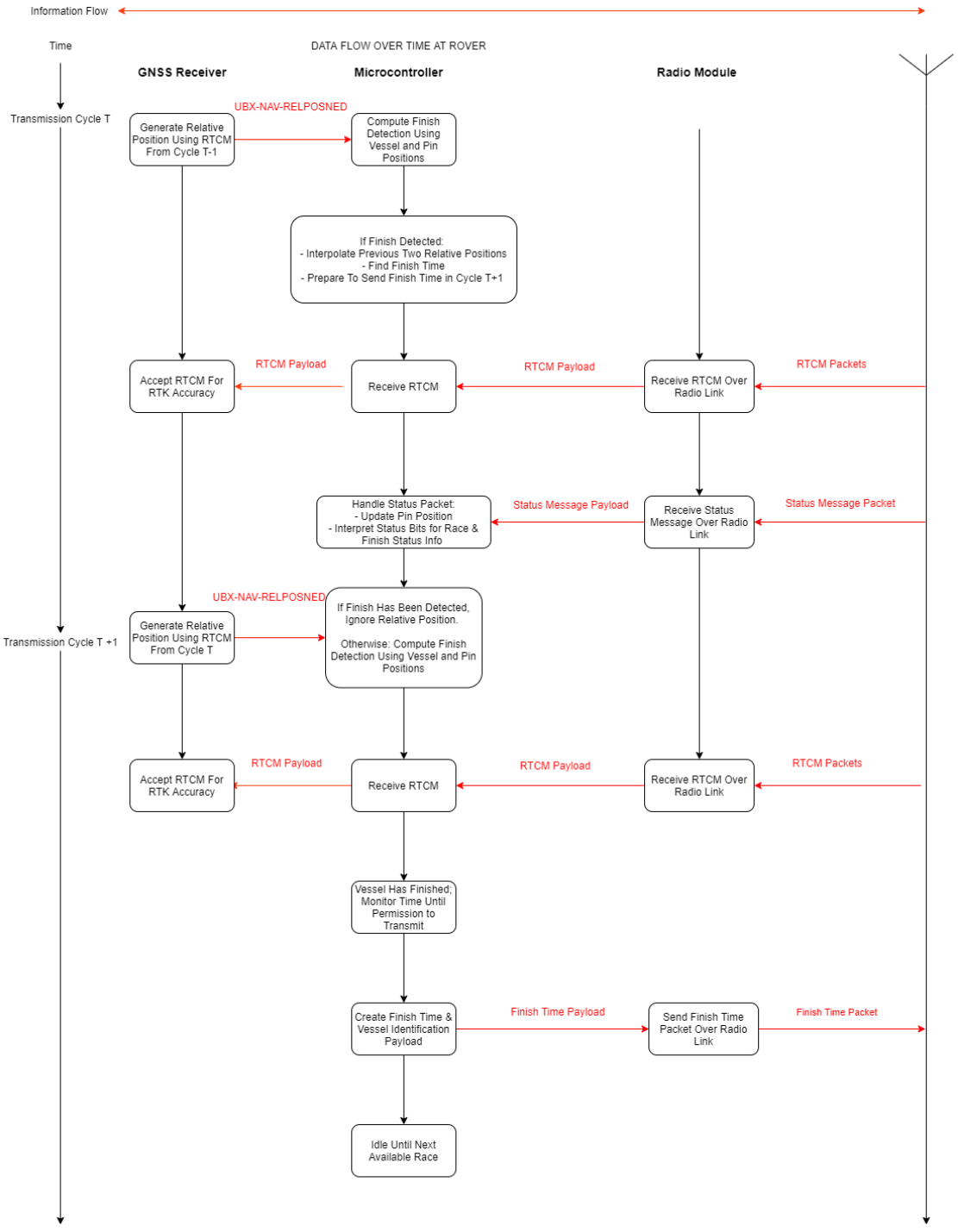


Figure 3.23: Flow Diagram for Transmission Cycle At the Rover

As a result of this flow of traffic, there is an introduction of a hybrid MAC protocol dependent on the time of the transmission cycle. There are two components to each transmission cycle:

1. A reserved time period at the beginning of each transmission cycle, where radio link traffic is restricted to RTCM and 'Base Status' Messages only.
2. A 'Slotted ALOHA' access protocol for competitor finish times.

3.6.7.3 Reserved Base Transmission Period

Due to the nature of the arrival of serial data from the RTK receivers at the start of every transmission cycle, it was sensible to reserve the first portion of the transmission cycle to handle the broadcast of RTCM and status information to all competitors and pin mark, as well as the pin message location update to the base.

Using the default initialized values when using the RadioHead manager module, gives a data-rate of 21.875kbps with the LoRa radio.

Assuming a maximum case scenario: 4 RTCM Messages (Max 425 Bytes) and 1 status message (Max 10 bytes) w/ 4 Bytes of RadioHead overhead. Therefore 435 payload bytes and 20 overhead bytes:

Calculated time to transmit with default data rate:

$$TransmissionTime = \frac{455 \times 8}{21,875} \approx 166ms$$

With the latency experienced due to the intermediate laptop processing, an additional buffer was added, providing the base 250ms to transmit. The reason for this decision is due to the fact that, in reality, only a small number of vessels reporting their position once per second will occur, hence a very large portion of the transmission cycle is not mandatory.

3.6.7.4 Reserved Pin Transmission Period

Like with the Base and RTCM, the pin must also transmit its position a guaranteed once per second. Once the Pin has received the RTCM corrections over the radio link,

and communicated such with the RTK receiver, it has priority to transmit its 8 byte position payload. Since the manager uses addressing, the address of the pin is already handled in the 4 byte overhead, this can be seen in line 310 of Appendix 7.5. Therefore for a 12 byte message:

$$TransmissionTime = \frac{12 \times 8}{21,875} \approx 4.389ms$$

Therefore a decision was to assign initially with a transmission slot of 10ms, including 2ms guard intervals, as illustrated in Figure 3.24

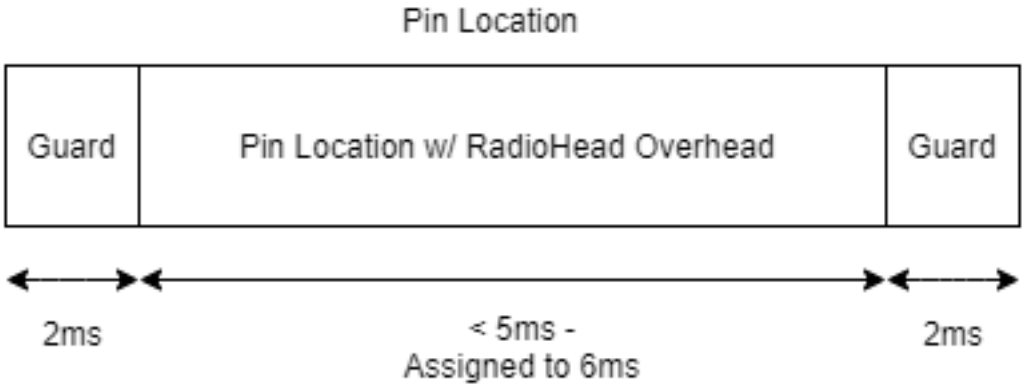


Figure 3.24: Timeslot Composition for Pin Transmission

3.6.7.5 Slotted ALOHA Medium Access

Slotted ALOHA is a pseudorandom access protocol, based on the concept of time windows for transmission. Finished competitors will not know when other finished competitors plan on transmitting their results. As a result of this, the concept of pseudorandom timeslot allocation is employed.

Given N timeslots, a presumed true random number generator and two competitors, the probability of both competitors transmitting in the same time slot is simply $\frac{1}{N}$. Therefore, the more timeslots, the decrease in the probability of simultaneous packet transmission.

Given a finish message consists of a 32 bit unsigned integer as well as RadioHead's 4 header bytes:

$$TransmissionTime = \frac{8 \times 8}{21,875} \approx 2.925ms$$

Therefore, since each finish time transmission expects an immediate acknowledgement from the base, a decision was to assign initially with a transmission slot of 20ms, including 2ms guard intervals, as illustrated in Figure 3.25

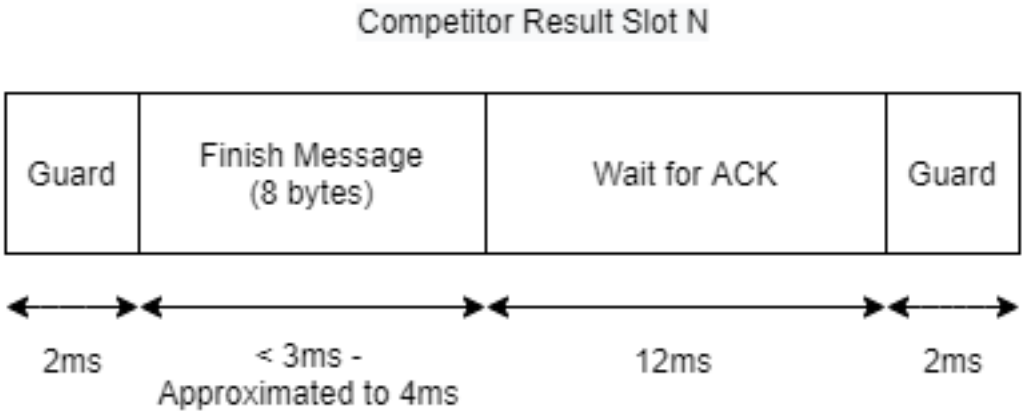


Figure 3.25: Timeslot Composition for Competitor Finish Transmission

Therefore with timeslots of 20ms, the full 1Hz transmission cycle can be illustrated in 3.26

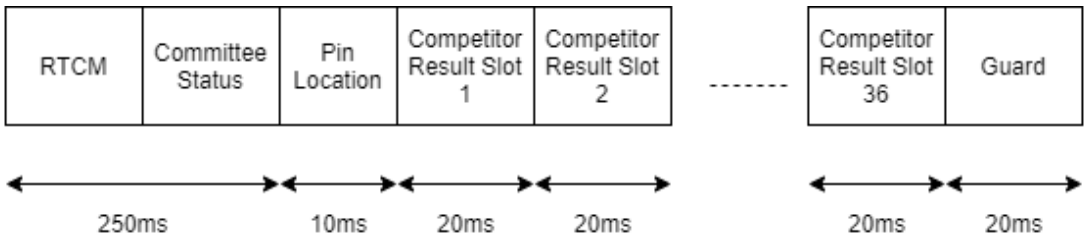


Figure 3.26: Timeslot Structure Full 1Hz Transmission Cycle

Appendices 7.5, 7.7 and 7.6 show the implementation of the medium for the base, pin and rover on the Feather M0. In the case where two vessels transmit in the same timeslot and packet collision occurs, the vessels simply assign a new timeslot in the next transmission cycle until an acknowledgement is found. Allowing the reliable reception of all finish times and hence the full list of results on the committee boat.

Chapter 4: **Discussion & Conclusion**

4.1 Implication of Current Working Environment

It is worth briefly mentioning the impact the current global situation had on this project.

4.1.1 Radio Protocol Testing

In order for RTK receivers to achieve centimetre level accuracy, a fundamental requirement is a clear view of the horizon in all directions. In any publicly accessible places this is achievable, however due to movement restrictions from March in depth testing of the radio protocol in response to accurate finish crossings was not possible.

The protocol was built and basic transmission and reception of transmission times completed, but in a practical sense more testing would in theory be completed. The same concept applies to a full practical set of tests, to prove the feasibility of the end-to-end system. As a result, future work will be required to validate the radio protocol using practical experiments.

4.2 Positioning Accuracy in the Context of Sailing Races

Experimentation using RTK receivers in this thesis, proved, in more ideal circumstances that finish detection, capable of at the worst, times nearest to a tenth of a second, and often times nearest to one hundredth of a second. The results are extremely promising, especially considering this project used only a single band RTK receiver.

Applications which take advantage of dual band receivers like the ZED-F9P, could prove to be even more successful in terms of generating more accurate finish times.

4.2.1 Moving Baseline Mode

Moving baseline mode was hugely successful in the context of not only positional accuracy, but the idea of portability for the system as a whole. Without the requirement of a fixed base station, this system could be moved to any location and used immediately.

4.3 Conclusion

This project lays the ground for a GNSS based system to be incorporated into the officiating of races at all levels of sailing. Further work is required to determine the feasibility as a whole, however the results show the potential for a commercial grade implementation of such a system in the future.

The current price of the technology may not be feasible for a number of levels of the sport. However with the cost of precision positioning, in particular RTK, dropping considerably over time, such a system may be used even at amateur level.

Chapter 5: **Impact of Research**

5.1 Provision of Previously Non-Existent Service in The Sailing Industry

In the sailing world, there is no finish line technology available at a commercial level. That is, clubs, committees and competitors alike all rely on the subjectivity of human interpretation to provide a service.

This finish line technology, with the appropriate future work completed, provides a service to ensure the sport of sailing is officiated autonomously, without bias or subjectivity. This technology will greatly enhance the quality of race officiating in the sport.

Likewise, with the accuracy of the technology proven, the system can help reduce protests made from competitors in which they disagree with an officials decision. Since may protests exist on the basis of the committees subjectivity, the number of valid claims will reduce significantly.

5.2 Potential Commercialization

As mentioned in the previous section; with further testing, the opportunity for a commercialization of such a system is evident. As of the time of writing of this thesis, there are no known companies or startups who seem to be making developments to provide such a facility to mariners globally.

This service could be particularly attractive as the price of precision RTK drops, particularly as consumer grade equipment from companies like u-blox grow in popularity.

Chapter 6: **Future Work**

6.1 In Depth Radio Protocol Testing

6.1.1 Overview

Due to the current global health situation, testing of the protocol in an area where centimetre level accuracy can be achieved with the RTK receivers, to be used in conjunction with the developed protocol was not possible.

As a result, simulated packet environment tests were the only experiment available to be conducted. This means that any small errors not found in simulations could be detected and rectified in a practical scenario.

6.1.2 End-to-End Testing With Multiple RTK Receivers

As this project was completed, for the majority, using only two RTK receivers, practical tests are required to more closely represent a real finish line. That is, an RTK receiver for both the pin and committee ends of the finish line, as well as a least two receivers to represent multiple competitors approaching the line. This test aims to evaluate:

1. The performance of finish detection and interpolation in a multi-competitor environment
2. The performance of the Medium Access protocol to handle time slotting, and recovery in the case of packet collision.

6.2 Expansion of the 'Committee Status Message' & Additional Committee Boat Services

6.2.1 Committee Status Message

As explained in Section 3.6.2, the committee boat is provided with a number of status bytes, to inform competitors about particular race configurations. Ideally, this status

message would have been expanded during testing to handle more cases as they became apparent.

In particular, vital messages such as the indication of the orientation of the finish line. Without this information in a real race, the competitor systems would not know when they are finished, or may determine they are finished when they are not.

6.2.2 Graphical User Interface - GUI

While the finish officiating itself is entirely autonomous, the competitors are required to know information about the configuration of the race, which must be specified by the committee. Items such as the start of a race, or the orientation of the finish line. As a result, a GUI capable of providing the committee with a method of communication with the fleet, or even a particular vessel could be completed through a graphical user interface on board the committee boat.

6.3 Embedded System Design

The final process would be to combine the set of technologies used here; a mix of evaluation kits and breakout boards, into a single PCB capable of all required functionality on the water.

Bibliography

AirSpayce (2020). *RadioHead Packet Radio Library*. URL: https://www.airspayce.com/mikem/arduino/RadioHead/ (visited on 03/03/2020).

Arbesser-Rastburg, B (2006). 'The GALILEO single frequency ionospheric correction algorithm'. In: *3rd European Space Weather Week, Brussels* December 2006.

ArduPilot (2018). *SiK Firmware*. URL: https://github.com/ArduPilot/SiK (visited on).

Avionic West (2020). *How GPS Works*. URL: http://www.avionicswest.com/Articles/howGPSworks.html (visited on 10/04/2020).

Broumandan, Ali et al. (2012). 'GNSS spoofing detection in handheld receivers based on signal spatial correlation'. In: *Record - IEEE PLANS, Position Location and Navigation Symposium* April, pp. 479–487. DOI: 10.1109/PLANS.2012.6236917.

Commission for Communications Regulation (2020). *Radio Frequency Plan for Ireland*. Tech. rep. May 2017. URL: https://www.comreg.ie/publication/radio-frequency-plan-for-ireland-6.

European Space Agency (2011). *GPS L1 Signal Plan*. URL: https://gssc.esa.int/navipedia/index.php/File:Chapter%7B%5C_%7D2%7B%5C_%7DTable%7B%5C_%7D1.png%7B%5C#%7Dfilehistory (visited on 01/04/2020).

Fusion Sport (2020). *SMARTSPEED PT SYSTEM OVERVIEW*. URL: https://learn.fusionsport.com/help/article/system-overview/ (visited on 14/11/2019).

GISGeography (2016). *Trilateration vs Triangulation – How GPS Receivers Work*. URL: https://gisgeography.com/trilateration-triangulation-gps/ (visited on).

HOPERF (2020). *HM-TRP Module*. URL: https://www.hoperf.com/modules/lora/HM-TRP.html (visited on 20/02/2020).

International Amateur Athletic Federation (2015). 'IAAF Photo Finish Guidelines'. In: June. URL: https://www.worldathletics.org/download/download?filename=4423f7ca-84bd-401d-b326-c074cebc4800.pdf%7B%5C&%7Durlslug=IAAF%20Photo%20Finish%20Guidelines.

Kalfus, Rudy et al. (2010). 'RTCM Overview: :RTCM SC-104 Enabling Standards that Support Emerging Positioning and Related Technologies'. In: URL: http://www.geopp.com/pdf/gppigs06%7B%5C_%7Drtcm%7B%5C_%7Df.pdf.

Kessen, Malcolm (2020). 'Shortest Distance From Point To Line'. In: URL: http://www.fundza.com/vectors/point2line/index.html.

Kim, Dong-Hoon, Eun-Kyu Lee and Jibum Kim (2019). 'Experiencing LoRa Network Establishment on a Smart Energy Campus Testbed'. In: *Sustainability* 11, p. 1917. DOI: 10.3390/su11071917. URL: https://www.researchgate.net/publication/332151302%7B%5C_%7DExperiencing%7B%5C_%7DLoRa%7B%5C_%7DNetwork%7B%5C_%7DEstablishment%7B%5C_%7Don%7B%5C_%7Da%7B%5C_%7DSmart%7B%5C_%7DEnergy%7B%5C_%7DCampus%7B%5C_%7DTestbed.

Kos, T., I. Markezic and J. Pokrajcic (Sept. 2010). 'Effects of multipath reception on GPS positioning performance'. In: *Proceedings ELMAR-2010*, pp. 399–402.

Larsen, Paul B. (2015). 'International Regulation of Global Navigation Satellite Systems'. In: *Journal of Air Law and Commerce* Volume 80.2, p. 59. URL: https://scholar.smu.edu/jalc/vol80/iss2/4?utm%7B%5C_%7Dsource=scholar.smu.edu%7B%5C%%7D2Fjalc%7B%5C%%7D2Fvol80%7B%5C%%7D2Fiss2%7B%5C%%7D2F4%7B%5C&%7Dutm%7B%5C_%7Dmedium=PDF%7B%5C&%7Dutm%7B%5C_%7Dcampaign=PDFCoverPages.

LEFORT, Michel (2019). *Practical Guide For The Finish Judge*, p. 17.

National Geospatial-Intelligence Agency (2020). *World Geodetic System (WGS 84)*. URL: https://www.nga.mil/ProductsServices/GeodesyandGeophysics/Pages/WorldGeodeticSystem.aspx (visited on 15/01/2020).

Novatel (2019). *Basic GNSS Concepts*. URL: https://www.novatel.com/an-introduction-to-gnss/chapter-2-basic-gnss-concepts/step-1-satellites/.
– (2020). *Real-Time Kinematic*. URL: https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/real-time-kinematic-rtk/ (visited on).
O.D, Munn and Brach A.R (1878). 'A Horse's Motion Scientifically Determined'. In: *Scientific American* 39.16, pp. 241–241. ISSN: 0036-8733. DOI: 10.1038/scientificamerican10191878-241b.
Ordinance Survey Ireland (1995). *D161 Arklow Refernce Station*. URL: https://www.osi.ie/wp-content/uploads/2017/07/D.-161-Arklow.pdf (visited on 01/03/2019).
Petovello, Mark, Cillian O'Driscoll and University of Calgary (2010). 'Carrier phase and its measurements for GNSS'. In: *Inside GNSS* July/August 2010, pp. 18–22.
Radio Technical Commission for Maritime Services (2020). *Radio Technical Commission for Maritime Services (RTCM)*. URL: https://www.rtcm.org/ (visited on).
Sailboatdata (2020). *FIRST 31.7 (BENETEAU) SPECIFICATIONS*. (Visited on 13/01/2020).
Semtech (2020). *What is LoRa?* URL: https://www.semtech.com/lora/what-is-lora (visited on 01/03/2020).
Srikant, Satya Sai and Rajendra Prasad Mahapatra (2010). 'Read Range of UHF Passive RFID'. In: *International Journal of Computer Theory and Engineering* June, pp. 323–325. ISSN: 17938201. DOI: 10.7763/ijcte.2010.v2.160.
Stupak, Grigory and International Committee on GNSS (2015). 'Compatibility and interoperability of GNSS . Further discussions .' In: pp. 1–14. URL: https://www.unoosa.org/pdf/icg/2015/icg10/wg/wga12.pdf.
Swift Navigation (2020). *Piksi Multi Evaluation Kit*. URL: https://www.swiftnav.com/store/piksi-multi-evaluation-kit/piksi-multi-evaluation-kit-2.4ghz (visited on).
Swiss Timing (2012). *SCAN ' O ' VISION STAR CAMERA*. November, p. 4. URL: https://www.swisstiming.com/fileadmin/Resources/Instruction%7B%5C_%7DManuals/3434.505.02.pdf.
– (2017). *SCAN ' O ' VISION MYRIA CAMERA*. URL: https://www.swisstiming.com/fileadmin/Resources/Data/Datasheets/DOCM%7B%5C_%7DMS%7B%5C_%7DMyria%7B%5C_%7DCamera%7B%5C_%7D1017%7B%5C_%7DEN.pdf.
Tanim, M M Zaman (2016). 'How a Passive UHF RFID System Operates .' In: January. DOI: 10.13140/RG.2.2.12361.34402.
Teunissen, Peter J.G. and Oliver Montenbruck (2014). *Springer Handbook of Global Navigation Satellite Systems*. Vol. 18. 6. Springer International Publishing, pp. 7–11. ISBN: 9783319429267. DOI: 10.1007/978-3-319-42928-1.
U-blox (2017). 'C94-M8P u-blox RTK Application Board Package - User Guide'. In: (visited on 25/12/2019).
– (2020a). *u-blox 8 / u-blox M8 Receiver Description with Protocol Specification*. Tech. rep., pp. 1–425. URL: https://www.u-blox.com/sites/default/files/products/documents/u-blox8-M8%7B%5C_%7DReceiverDescrProtSpec%7B%5C_%7D%7B%5C%%7D28UBX-13003221%7B%5C%%7D29.pdf (visited on 12/05/2019).
– (2020b). *ZED-F9P Summary*. URL: https://www.u-blox.com/sites/default/files/ZED-F9P%7B%5C_%7DProductSummary%7B%5C_%7D%7B%5C%%7D28UBX-17005151%7B%5C%%7D29.pdf (visited on).
Van Sickle, Jan and Pennsylvania State University (2018). *Components of the Carrier Phase Observable*. URL: https://www.e-education.psu.edu/geog862/node/1729 (visited on 20/02/2020).
Violino, Bob (2008). 'What is RFID'. In: pp. 1–9. DOI: 10.1007/978-1-84800-153-4_1.
Vu, B. N. and M. Andrle (2014). 'The code and carrier tracking loops for GPS signal'. In: *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*, pp. 569–574.

WATSON, STEPHEN (2016). *Olympic Timekeeping Is a Seriously Hi-Tech Affair*. URL: https://www.esquire.com/style/g2891/omega-olympics-rio/ (visited on 14/01/2020).

World Sailing (2019). 'Race Management Manual'. In: July. World Sailing. Chap. O. URL: https://www.sailing.org/tools/documents/RaceManagementManualJuly2019-[25256].pdf.

Zaminpardaz, Safoora, Peter J G Teunissen and Nandakumaran Nadarajah (2017). 'GLONASS CDMA L3 ambiguity resolution and positioning'. In: *GPS Solutions* 21.2, pp. 535–549. ISSN: 1521-1886. DOI: 10.1007/s10291-016-0544-y. URL: https://doi.org/10.1007/s10291-016-0544-y.

Chapter 7: **Appendices**

7.1 A. Implementation of Finish Detection Algorithms

```
import math
import ubx_messages
import distance


## A variety of finish detection methods evaluated during system
    develpment

# used for the angle between two vectors
def angle_between(vector1, vector2):
    dot_prod = vector1[0] * vector2[0] + vector1[1] * vector2[1]
    determinant = vector1[0] * vector2[1] - vector1[1] * vector2[0]
    # use atan2 to compute angle
    angle = -math.atan2(determinant, dot_prod)
    return angle


# Real-time processing of base/rover angles to detect a line crossing
def has_crossed_line_angle(base_ned, ser):
    # using boolean variable for better visualisation purposes
    crossed_line = False

    while not crossed_line:
        # check for relative position message
        ubx_messages.isrelposned(ser)
        # poll a relative position message
        boat_ned = ubx_messages.ubxnavrelposned(ser)
        # calculate angle
```

```
28          angle = angle_between((base_ned[0], base_ned[1]), (boat_ned[0],
                boat_ned[1]))
29          if angle < 0:
30              # line crossing has occurred, exit.
31              crossed_line = True
32      return
33
34
35  ## Line crossing based on shortst distance
36  def has_crossed_line_dist(base_ned, ser):
37      crossed_line = False
38
39      while not crossed_line:
40          # check for relative position message
41          ubx_messages.isrelposned(ser)
42          #extract relative position message
43          boat_ned = ubx_messages.ubxnavrelposned(ser)
44          # calculate perpendicular distance
45          dist = distance.pnt2line(boat_ned, base_ned)
46          if dist < 0:
47              crossed_line = True
48      return
49
50  ## Line crossing using point above or below to slope of the finish line
51  def has_crossed_slope(base, boat_ned):
52      # detect whether the boat is above or below the finish_line
53      if boat_ned[1] * base.position[0] > base.position[1] * boat_ned[0]:
54          return 0
55      else:
56          return 1
```

7.2 B. Shortest Distance From Point to Line Segment Python Script

```
# This Python Script is an adaptation of the methods employed
# by Malcolm Kesson
# This Script Calculates the shortest distance from a point
# (NED Offset) to a line Segment (Finish Line)
# The result will return one of three answers:
# The location of the boat is closest to one of the two endpoints of the
# finish line,
# or the shortest distance is the perpendicular distance.
# A full explanation can be found in the report.
# Note the variable conventions are inline with Python 2.x

# As per Malcolm Kessen Instructions:

# Given a line with coordinates 'start' and 'end' and the
# coordinates of a point 'pnt' the proc returns the shortest
# distance from pnt to the line and the coordinates of the
# nearest point on the line.
#
# 1  Convert the line segment to a vector ('line_vec').
# 2  Create a vector connecting start to pnt ('pnt_vec').
# 3  Find the length of the line vector ('line_len').
# 4  Convert line_vec to a unit vector ('line_unitvec').
# 5  Scale pnt_vec by line_len ('pnt_vec_scaled').
# 6  Get the dot product of line_unitvec and pnt_vec_scaled ('t').
# 7  Ensure t is in the range 0 to 1.
# 8  Use t to get the nearest location on the line to the end
#    of vector pnt_vec_scaled ('nearest').
# 9  Calculate the distance from nearest to pnt_vec_scaled.
# 10 Translate nearest back to the start/end line.
# Malcolm Kesson 16 Dec 2012
```

```
31
32 import math
33
34
35 # Calculate the dot product of two 2D vectors v, w
36 def dot(v, w):
37     # Unpack the vectors into thei x and y coordinates
38     x, y = v
39     X, Y = w
40     return x * X + y * Y
41
42
43 # find the length of a vector from the origin
44 def length(v):
45     x, y = v
46     return math.sqrt(x * x + y * y)
47
48
49 # find the difference between two vectors
50 def vector(b, e):
51     x, y = b
52     X, Y = e
53     return (X - x, Y - y)
54
55
56 # find the unit length of the vector
57 def unit(v):
58     x, y = v
59     mag = length(v)
60     return x / mag, y / mag
61
62
63 # find the distanec between two points
```

```
64 def distance(p0, p1):
65     return length(vector(p0, p1))
66
67
68 # Scale a vector by scaling fact sc
69 def scale(v, sc):
70     x, y = v
71     return x * sc, y * sc
72
73
74 # compute vector addition
75 def add(v, w):
76     x, y = v
77     X, Y = w
78     return x + X, y + Y
79
80
81 def pnt2line(boat_data, base_data):
82     pnt = boat_data[0:2]
83     start = [0, 0]
84     end = base_data[0:2]
85     line_vec = vector(start, end)
86     pnt_vec = vector(start, pnt)
87     line_len = length(line_vec)
88     line_unitvec = unit(line_vec)
89     pnt_vec_scaled = scale(pnt_vec, 1.0 / line_len)
90     t = dot(line_unitvec, pnt_vec_scaled)
91     if t < 0.0:
92         t = 0.0
93     elif t > 1.0:
94         t = 1.0
95     nearest = scale(line_vec, t)
96     dist = distance(nearest, pnt_vec)
```

```
    nearest = list(add(nearest, start))

    return dist, boat_data[2]

# Basic function to test script functionality
def test_pnt():
    dist, time = pnt2line([0, -3600, 10000], [6000, 8000])
    print(dist)


if __name__ == "__main__":
    test_pnt()
```

7.3 C.Interpolation Techniques

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import splprep, splev, interp1d, UnivariateSpline
import tools
import distance
from intersection import intersection
import finish_detection
# note this script employs external libraries such as:
# scipy, matplotlib and numpy. These must be installed via pip or IDE
# for proper usage


def linear_interpolation_relposned(base, boat_history):
    np.unique(boat_history, axis=0)
    x_pts, y_pts = tools.separate_x_y_coords(boat_history)
    N = 1000
```

```
18      x = np.linspace(0, base[0], N)
19      y = np.linspace(0, base[1], N)
20      x_intersect, y_intersect = intersection(x_pts, y_pts, x, y)
21
22
23  # nonlinear interpolation of relative positions
24  # Only for visualization purposes.
25  def nonlinear_interpolation_b_spline(base, boat_history):
26      # ensure no duplicate points
27      np.unique(boat_history, axis=0)
28
29      # separate x and y points for non linear interpolation
30      x_pts, y_pts = tools.separate_x_y_coords(boat_history)
31
32      # conduct spline interpolation using scipy package
33      tck, u = splprep([x_pts, y_pts], u=None, s=0.0, per=0)
34      u_new = np.linspace(u.min(), u.max(), 10000)
35      x_new, y_new = splev(u_new, tck, der=0)
36
37      # points describing finish line for graphical representation
38      x = np.linspace(0, base.position[0], 1000)
39      y = np.linspace(0, base.position[1], 1000)
40
41      # Find intersection
42      x_intersect, y_intersect = intersection(x_new, y_new, x, y)
43
44      # plotting
45      fig, ax = plt.subplots()
46      ax.plot(x_pts, y_pts, 'ro')
47      ax.plot(x_new, y_new, 'r-')
48      ax.plot([0, base.position[0]], [0, base.position[1]], marker='o')
49      ax.plot(x_intersect, y_intersect, "*k")
```

```
50      ax.title.set_text('Non-Linear Interpolation of Boat Relative
            Position')
51      ax.set_xlabel('Northern Relative Offset (cm)')
52      ax.set_ylabel('Eastern Relative Offset (cm)')
53
54      return
55
56
57  def linear_interpolation_shortest_distance(base, boat_history,
        full_set):
58      np.unique(boat_history, axis=0)
59      # lists to hold interpolation requirements
60      # distances to line
61      distances = []
62      # time stamp of generated relative positions
63      timestamp = []
64      # status of each point (above/below line)
65      line_status = []
66
67      crossed_time = []
68      crossed_dist = []
69
70      for i in range(len(boat_history)):
71          if i == 0:
72              distances.append(distance.pnt2line(boat_history[i],
                    base.position)[0])
73              line_status.append(finish_detection.has_crossed_slope(base,
                    boat_history[i]))
74              timestamp.append(boat_history[i][2])
75
76          else:
77              timestamp.append(boat_history[i][2])
```

```
78              boat_distance = distance.pnt2line(boat_history[i],
                    base.position)[0]
79              above_below = finish_detection.has_crossed_slope(base,
                    boat_history[i])
80
81              # line crossing detected
82              if above_below != line_status[i - 1]:
83                  # full set contains thousands of points, and has runtime
                        issues with interpolation plotting
84                  # as a result, this only plots the one point either side
                        of the finish line
85                  if full_set:
86                      # negate the distance to show a line crossing
87                      boat_distance = -1 * boat_distance
88                      # add the distance of the point to line after BEFORE
                            crossing
89                      crossed_dist.append(distances[i - 1])
90                      # add the newly calcated point
91                      crossed_dist.append(boat_distance)
92                      # add the two relevant timestamps
93                      crossed_time.append(timestamp[i - 1])
94                      crossed_time.append(boat_history[i][2])
95                      break
96
97                  # smaller subset of points extracted from full set
98                  boat_distance = -1 * boat_distance
99              # if boat has already crossed the line, check the value of
                    the shortest distance
100             # if its < 0 maintain this until a new line crossing is
                    detected
101             elif distances[i - 1] < 0:
102                 boat_distance = -1 * boat_distance
103
```

```
104             distances.append(boat_distance)
105             line_status.append(above_below)
106
107     #find intercept of interpolated data
108     intercept_x = np.linspace(timestamp[0], timestamp[len(timestamp) -
            1], len(timestamp))
109     intercept_y = [0] * (len(timestamp))
110
111     # plot results depending on whther or not a full set or subset of
            data is used.
112     # Note this was to prevent run time issues
113     if (full_set):
114         x_intersect, y_intersect = intersection(np.array(crossed_time),
                np.array(crossed_dist), np.array(intercept_x),
115                                                 np.array(intercept_y))
116         print("Linear Intercept: ", x_intersect[0], y_intersect[0])
117         fig, ax = plt.subplots()
118         ax.plot(x_intersect, y_intersect, '*k')
119         ax.plot(crossed_time, crossed_dist, '-ro')
120         ax.plot(intercept_x, intercept_y)
121         ax.title.set_text('Linear Interpolation of Shortest Distance to
                Finish Line')
122         ax.set_xlabel('Time of Week (Milliseconds)')
123         ax.set_ylabel('Shortest Distance (cm)')
124
125     else:
126         x_intersect, y_intersect = intersection(np.array(timestamp),
                np.array(distances), np.array(intercept_x),
127                                                 np.array(intercept_y))
128
129         print("Linear Intercept: ", x_intersect[0], y_intersect[0])
130         fig, ax = plt.subplots()
131         ax.plot(x_intersect, y_intersect, '*k')
```

```
132         ax.plot(timestamp, distances, '-ro')
133         ax.plot(intercept_x, intercept_y)
134         ax.title.set_text('Linear Interpolation of Shortest Distance to
                Finish Line')
135         ax.set_xlabel('Time of Week (Milliseconds)')
136         ax.set_ylabel('Shortest Distance (cm)')
137
138
139 ## Non Linear Interpolation of Data Set Points Using Shortest Distance
        To Finish Line
140 def nonlinear_interpolation_shortest_distance(base, boat_history):
141     # remove duplicate rows
142     np.unique(boat_history, axis=0)
143     # unique distance to line
144     distances = []
145     # time stamp of each relative position message
146     timestamp = []
147     # shows if each point is above or below the finish line
148     line_status = []
149
150     # process the relative position messages into shortest distance for
            illustration
151     for i in range(len(boat_history)):
152         if i == 0:
153             distances.append(distance.pnt2line(boat_history[i],
                    base.position)[0])
154             line_status.append(finish_detection.has_crossed_slope(base,
                    boat_history[i]))
155             timestamp.append(boat_history[i][2])
156         else:
157             timestamp.append(boat_history[i][2])
158             boat_distance = distance.pnt2line(boat_history[i],
                    base.position)[0]
```

```
159             above_below = finish_detection.has_crossed_slope(base,
                    boat_history[i])
160             # if line crossing has been detected, negate the distance
161             if above_below != line_status[i - 1]:
162                 boat_distance = -1 * boat_distance
163             # if previous value is negative line crossing has already
                    been detected, negate distance
164             elif distances[i - 1] < 0:
165                 boat_distance = -1 * boat_distance
166
167             distances.append(boat_distance)
168             line_status.append(above_below)
169
170     # compute interpolation
171     tck, u = splprep([timestamp, distances], u=None, s=0.0, per=0)
172     u_new = np.linspace(u.min(), u.max(), 1000)
173     time_new, short_new = splev(u_new, tck, der=0)
174
175     # find intercepts and point of intersection
176     intercept_x = np.linspace(timestamp[0], timestamp[len(timestamp) -
            1], len(timestamp))
177     intercept_y = np.zeros(len(timestamp))
178     x_intersect, y_intersect = intersection(time_new, short_new,
            intercept_x, intercept_y)
179
180     print("Non-linear Intercept: ", x_intersect[0], y_intersect[0])
181     # plot results
182     fig, ax = plt.subplots()
183     ax.plot(timestamp, distances, 'ro')
184     ax.plot(time_new, short_new, 'r-')
185     ax.plot(intercept_x, intercept_y)
186     ax.plot(x_intersect, y_intersect, '*k')
```

```
187     ax.title.set_text('Non-Linear Interpolation of Shortest Distance to
            Finish Line')
188     ax.set_xlabel('Time of Week (Milliseconds)')
189     ax.set_ylabel('Shortest Distance (cm)')
190 
191     return x_intersect[0], y_intersect[0]
192 
193 
194 # Non Linear Interpolation of Data Set Points Using Shortest Distance
        To Finish Line
195 def nonlinear_interpolation_shortest_distance_real_time(base,
        boat_history):
196     np.unique(boat_history, axis=0)
197     distances = []
198     timestamp = []
199     line_status = []
200 
201     for i in range(len(boat_history)):
202         if i == 0:
203             distances.append(distance.pnt2line(boat_history[i])[0])
204             line_status.append(finish_detection.has_crossed_slope(base,
                    boat_history[i]))
205             timestamp.append(boat_history[i][2])
206 
207         else:
208             timestamp.append(boat_history[i][2])
209             boat_distance = distance.pnt2line(boat_history[i])[0]
210             # boat crossing detected, negate distance
211             if boat_history[i][3] != boat_history[i - 1][3]:
212                 boat_distance = -1 * boat_distance
213             # boat crossing previously dete
214             elif  boat_history[i - 1][3] < 0:
215                 boat_distance = -1 * boat_distance
```

```
216
217             distances.append(boat_distance)
218     # interpolate
219     tck, u = splprep([timestamp, distances], u=None, s=0.0, per=0)
220     u_new = np.linspace(u.min(), u.max(), 1000)
221     time_new, short_new = splev(u_new, tck, der=0)
222
223     # find intercept and point of intersection of interpolated values
            with zero crossing
224     intercept_x = np.linspace(timestamp[0], timestamp[len(timestamp) -
            1], len(timestamp))
225     intercept_y = np.zeros(len(timestamp))
226     x_intersect, y_intersect = intersection(time_new, short_new,
            intercept_x, intercept_y)
227
228     print("Non-linear Intercept: ", x_intersect[0], y_intersect[0])
229     # plot
230     fig, ax = plt.subplots()
231     ax.plot(timestamp, distances, 'ro')
232     ax.plot(time_new, short_new, 'r-')
233     ax.plot(intercept_x, intercept_y)
234     ax.plot(x_intersect, y_intersect, '*k')
235     ax.title.set_text('Non-Linear Interpolation of Shortest Distance to
            Finish Line')
236     ax.set_xlabel('Time of Week (Milliseconds)')
237     ax.set_ylabel('Shortest Distance (cm)')
238
239     return x_intersect[0], y_intersect[0]
```

7.4 D. RTCM Streaming from Laptop to Feather M0

```
# Python Script for accepting bulk RTCM and creating an organized message for
# Adafruit Feather M0 to handle all messages at once
# Process:
# 1: Accept RTCM Messages at the beginning of each one second transmission cycle
# 2: Extract the payload size of each message
# 3: Add an array of markers at the beginning of the full RTCM message,
# to indicate where each RTCM message starts and ends, reducing the
# necessity for bitwise operation at the MCU
# 4: Include 2 bytes representing the total size of the message to allow
# all data to be read into memory without risk of overwrites
# 5: Transmit organized packet over Feather M0 USB virtual COM port

import serial
from bitstring import BitArray

# function to open a com port.
# NOTE: Ensure COM Port number is correct before running (Varies from computer)
def createserialcommunication(com_port, baud):
    ser = serial.Serial() # open serial port
    ser.port = com_port
    ser.baudrate = baud
    ser.open()
    ser.inter_byte_timeout = 0.1
    return ser


# Function to extract the index where each message starts and ends
def extract_indexes(data_string):

    # count the number of messages
```

```
31      marker_count = 0
32      # placeholder to hold the current marker position. Starts at first
            message (0)
33      marker = 0
34      # byte array to hold hex value representing marker position
35      marker_bytes = b''
36      # extract the data size of the first message using bitwise operation
37      data_size = int(BitArray(data_string[marker+1:marker+3]).bin[6:], 2)
38
39      # increment the new marker position. Note the increment of 4 is to
            account for
40      # 3 bytes of CRC and to move the marker to the next preamble (if it
            exists):
41      # the increment of two is to account for the reserved and data size
            bits
42      marker = marker + 2 + data_size + 4
43      # If only one message arrives, exit
44      if marker >= len(data_string):
45          return marker_bytes, 0
46
47      else:
48          # 211 represents 0xD3 in binary, the preamble for each RTCM
                message
49          while data_string[marker] == 211:
50              # a preamble has been found.
51
52              # convert the position of the marker into a hex value
53              marker_bytes += marker.to_bytes(1, 'big')
54              # increment the byte counter
55              marker_count += 1
56              # extract the data size of the proceeding message
57              data_size = int(BitArray(data_string[marker + 1:marker +
                    3]).bin[6:], 2)
```

```
58              # increment the marker position to the start of the new
                    message (if exists)
59              marker = marker + 2 + data_size + 4
60              # marker now exceeds the size of the string, no more
                    messages, break from the loop
61              if marker >= len(data_string):
62                  break
63          # return the byte array representing each of the markers, as
                well as the number of marker
64          return marker_bytes, marker_count
65
66
67  def stream_rtcm():
68      # open serial ports for GNSS receivers and Feather M0
69      s_in = createserialcommunication("COM24", 115200)
70      s_out = createserialcommunication("COM21", 115200)
71
72      while True:
73          # read in the max amount of RTCM data possible, timeout after
                1ms if no further bytes appear
74          data_buffer = s_in.read(4120)
75
76          # find the length of the entire message
77          payload_length_int = len(data_buffer)
78
79          # if no data read, wait until available data
80          while len(data_buffer) == 0:
81              data_buffer = s_in.read(4120)
82
83          # use the extract function to generate markers
84          markers, markers_int = extract_indexes(data_buffer)
85
```

```
86          # increase the payload length by the number of markers to be
                added, so Feather M0
87          # is aware of the amount of data to read
88          payload_length_int += len(markers)
89          # convert payload length to two hex bytes
90          payload_length = payload_length_int.to_bytes(2, 'big')
91
92          # generate byte array for overhead (payload length and markers)
93          overhead = payload_length + markers
94
95          # add overhead to data buffer
96          payload = overhead + data_buffer
97          # Write organized message to Feather via USB
98          s_out.write(payload)
99
100 if __name__ == "__main__":
101     stream_rtcm()
```

7.5 E. Feather M0 Base Protocol

```
1    /*Begining of Auto generated code by Atmel studio */
2  #include <Arduino.h>
3
4  /*End of auto generated code by Atmel studio */
5
6  // Radio Protocol using the Airspayce Packet Radio Library
7
8  // Include Headers: Note Reliable Datagram inherits most of regular
       datagram so this is
9  // suitable for use
10 // Airspayce Packet Radio Library
11 #include <RHReliableDatagram.h>
```

```
12 #include <RH_RF95.h>
13 ///
14 #include <SPI.h>
15 //Beginning of Auto generated function prototypes by Atmel Studio
16 void toggle();
17 void handle_competitor_finish(uint8_t len);
18 void stream_and_send_rtcm();
19 void send_status_byte();
20 //End of Auto generated function prototypes by Atmel Studio
21
22
23
24 // Some global definitions for Base Address and Pin Mappings
25 #define SERVER_ADDRESS 1
26 #define RFM95_CS 5 // SPI Chip/ Slave Select
27 #define RFM95_RST 6 // SPI Reset
28 #define RFM95_INT 10 // SPI Interrupt
29
30 // Some other flag vars for ISR etc.
31 //Define the max buffer size = 255 bytes
32 uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
33 const uint8_t external_interrupt = 13;
34 int rtcm_flag = 0;
35 unsigned long flag_time;
36
37
38 /// PIN and Status ///
39 // array to
40 uint8_t pin[8] = {0};
41 // array to hold 16 bits which make the two byte status
42 uint8_t status_bytes[16] = {0};
43
44 uint8_t status_message[10];
```

```
45 ////////////////
46
47 ///// RTCM Variables /////
48
49 int marker = 0;
50 int data_size = 0;
51 byte markers[3];
52 int num_messages = 0;
53 byte preamble = 0xD3;
54 // an array to hold the size of each message
55 uint8_t sent_sizes[4] = {0,0,0,0};
56 // a var to hold the size of the current message to be sent
57 uint8_t size_to_send = 0;
58 // 2 byte array to collect the first two received bytes, which tell the
       size of the payload
59 char payload_size[2];
60 // Since the RTCM protocol is not freely available to identify each
       message as they arrive
61 // They are simply held in "message" arrays"
62 // The max theoretical size of a message is:
63 // 2^10 = 1024 byte payload
64 // 3 bytes of header (8 bits Preamble, 6 Reserved bits, 10 bit Data
       Size)
65 // 24 bit CRC
66 // Each Message: Max 1030 bytes
67 // Maximum RTCM = 4*1030 = 4120 bytes
68 char rx_data[4120];
69 uint8_t message_1[1030];
70 uint8_t message_2[1030];
71 uint8_t message_3[1030];
72 uint8_t message_4[1030];
73
74 /////// END RTCM Variable ////////////////////////////////////////////
```

```
75
76 /// SPI Radio Driver and Manager Instances //////
77 // Singleton instance of the radio driver
78 RH_RF95 driver(RFM95_CS, RFM95_INT);
79 // Class to manage message delivery and receipt, using the RH_RF95 drive
80 RHReliableDatagram base(driver, SERVER_ADDRESS);
81
82
83 //// END SPI Instances ////////////////////////////////////////////////
84
85
86
87 // Interrupt Service Routine for GPS timepulse
88 void toggle(){
89     // set the flag to one so the controller knows to read from usb
        serial
90     rtcm_flag = 1;
91 }
92
93 // send finish to laptop for processing
94 void handle_competitor_finish(uint8_t len){
95   Serial.write(buf, len);
96
97   }
98
99 //funciton to stream serial form the
100 void stream_and_send_rtcm(){
101     // set the number of received messages to 0
102     num_messages = 0;
103     // reset the flag in preparation for the next time pulse
104     rtcm_flag = 0;
105
```

```
106     // message have not yet arrived, usually due to latency from GPS ->
            Computer -> MCU via USB
107     // Exit Function and try again
108     if(!Serial.available()){
109       return;
110       }
111
112     // There is data available
113     // Read the first two bytes to get the data size
114     Serial.readBytes(payload_size, 2);
115     //convert two byte uint8_t data to 32bit integer
116     data_size = (int)payload_size[0]*256 + (int)payload_size[1];
117
118
119     // some debgging messages which can be read from FTDI serial
            converter
120     Serial1.print("Data to Read: ");
121     Serial1.println(data_size);
122
123     // integer for debugging number of received bytes
124     int count = 0;
125     count = Serial.readBytes(rx_data, data_size);
126
127     //clear buffer in any noisy data has arrived
128     while(Serial.available()){
129       Serial.read();
130       }
131
132     // Need to find out how many messages exist in the payload.
133     // The start of each message is marked by a marker byte, for easy
            collection
134     // The markers appear at the front of the payload.
```

```
135     // Search until the RTCM preamble (0xD3) is found, all bytes before
            the preamble will be a marker
136     // Number of messages is the number of markers + 1, since the start
            of the first message starts at
137     // 0 and does not need to be tracked.
138     while(rx_data[num_messages] != 0xD3){
139       // add the marker to a dedicated array for handling the individual
              messages
140       markers[num_messages] = rx_data[num_messages];
141       // increase the number of messages
142       num_messages++;
143     }
144
145     // Iterate for the nuber of available messages
146     // In theory, the SPI radio should receive at minimum, 3 messages
            per one second cycle
147     // The conditionals below handle all cases from 1 message to 4
148     // <= rather than < allows for the first message which does not
            have a marker to be accounted
149     // for.
150     for(int i = 0; i <= num_messages; i++){
151       // If the current message is the finsl message in the stream then
              the end marker is simply the
152       // size of the data
153       if(i == num_messages){
154           if(i==0){
155             for(int j = 0; j<data_size;j++){
156               // pull message from full rxbuffer and place in relevant
                      message array
157               message_1[j] = rx_data[j+num_messages];
158             }
159             // Find the size to be sent
160             size_to_send = data_size + 1;
```

```
161             // Send over SPI
162             base.sendto((message_1), size_to_send, RH_BROADCAST_ADDRESS);
163             Serial1.println("Sent");
164             break;
165           }
166           else if(i==1){
167             for(int j = markers[0]; j<data_size; j++){
168               message_2[j-markers[0]] = rx_data[j+num_messages];
169           }
170             size_to_send = (data_size + 1) - markers[0];
171             base.sendto(message_2, size_to_send, RH_BROADCAST_ADDRESS);
172             Serial1.println("Sent");
173             break;
174           }
175           else if(i==2){
176             for(int j = markers[1]; j<data_size;j++){
177               message_3[j-markers[1]] = rx_data[j+num_messages];
178           }
179
180             size_to_send = (data_size + 1) - markers[1];
181             base.sendto(message_3,(data_size+1)-markers[1],
                    RH_BROADCAST_ADDRESS);
182             Serial1.println("Sent");
183             break;
184           }
185           else if(i==3){
186             for(int j = markers[2]; j<data_size;j++){
187               message_4[j-markers[2]] = rx_data[j+num_messages];
188           }
189
190             size_to_send = (data_size + 1) - markers[2];
191             base.sendto(message_4,size_to_send, RH_BROADCAST_ADDRESS);
192             Serial1.println("Sent");
```

```
193             break;
194           }
195         }
196       // the current message is not the final message, need to use the
          markers to collect the
197       // correct amount of data
198       else{
199         if(i==0){
200           for(int j = 0; j<markers[0];j++){
201             message_1[j] = rx_data[j+num_messages];
202           }
203           size_to_send = markers[0];
204           base.sendto(message_1, size_to_send, RH_BROADCAST_ADDRESS);
205           Serial1.println("Sent");
206           continue;
207         }
208         else if(i==1){
209           for(int k = markers[0]; k<markers[1];k++){
210             message_2[k-markers[0]] = rx_data[k+num_messages];
211         }
212           size_to_send = markers[1]-markers[0];
213           base.sendto(message_2, size_to_send, RH_BROADCAST_ADDRESS);
214           Serial1.println("Sent");
215           continue;
216         }
217         else if(i==2){
218           for(int j = markers[1]; j<markers[2];j++){
219             message_3[j-markers[1]] = rx_data[j+num_messages];
220         }
221           sent_sizes[i] = data_size-markers[1];
222           base.sendto(message_3, markers[2]-markers[1],
                RH_BROADCAST_ADDRESS);
223           Serial1.println("Sent");
```

```
224             continue;
225           }
226           else if(i==3){
227             for(int j = markers[2]; j<data_size;j++){
228               message_4[j-markers[2]] = rx_data[j+num_messages];
229           }
230             size_to_send = (data_size+1)-markers[2];
231             base.sendto(message_4, size_to_send, RH_BROADCAST_ADDRESS);
232             Serial1.println("Sent");
233             continue;
234           }
235       }
236     }
237
238     Serial1.println("RTCM Collection Complete: ");
239     Serial1.println(num_messages);
240     markers[0] = 0;
241     markers[1] = 0;
242     markers[2] = 0;
243     marker = 0;
244     num_messages = 0;
245   }
246
247 void setup()
248 {
249   // Define timemode pin as an input
250   pinMode(external_interrupt, INPUT);
251   //initiate ISR on rising edge of pin
252   attachInterrupt(digitalPinToInterrupt(external_interrupt), toggle,
        RISING);
253
254   // Open USB Serial Port (Only need USB on Base)
255   Serial.begin(115200);
```

```
256   // FTDI Debugging only
257   Serial1.begin(19200);
258   // Wait for serial port to be available
259   while (!Serial) ;
260
261   //Initialise manager class and hence the radio with default settings
262
263   ////////////////// Defaults /////////////////
264   /// Frequency 434MHz                   ///
265   /// Tx Power: 13dBm                    ///
266   /// Bandwidth: 125KHz                  ///
267   /// Coding Rate: 4/5                   ///
268   /// Spreading Factor: 128 chips/ Symbol ///
269   /// CRC On                             ///
270   ////////////////////////////////////////////
271
272
273   // Note HOPERF RFM96 module uses a PA_BOOST transmitter pin
274   // Power Range: 5 to 23 dBm:
275   // Example base.setTxPower(23, false);
276   if (!base.init())
277     Serial1.println("init failed");
278   else
279     Serial1.println("init succesful");
280
281 }
282
283
284 void send_status_byte(){
285    // bitwise operations here (Future Work)
286    //memcpy(status_message, status_byte)
287    memcpy(status_message+2, pin, 8*sizeof(uint8_t));
288
```

```
289     base.sendto(pin, 8, RH_BROADCAST_ADDRESS);
290 
291 }
292 
293 void loop() {
294     //timepulse has cause ISR, need to send RTCM corrections
295     if(rtcm_flag == 1){
296         // debugging to UART
297         Serial1.println("RTCM Interrupt");
298         stream_and_send_rtcm();
299         send_status_byte();
300         //send_status_packet();
301         rtcm_flag = 0;
302       }
303      else{
304         uint8_t len = sizeof(buf);
305         // Address from which the message originated
306         uint8_t from;
307         if(base.recvfromAck(buf, &len, &from)){
308             // recieved a finish time, need to handle it
309             // Acknowledge who its from, hold results
310             if(from == 2){
311                 Serial1.write("Pin Updated");
312                 memcpy(pin, buf, 8*sizeof(uint8_t));
313               }
314             else{
315                 handle_competitor_finish(len);
316               }
317             // debug message
318             Serial1.write(buf, len);
319           }
320 
321       }
```

```
322 }
```

7.6 F. Feather M0 Rover Protocol

```
1    /*Begining of Auto generated code by Atmel studio */
2  #include <Arduino.h>
3
4  /*End of auto generated code by Atmel studio */
5
6  // Header file for the manager
7  #include <RHReliableDatagram.h>
8  #include <RH_RF95.h>
9  #include <SPI.h>
10 #include <math.h>
11 //Beginning of Auto generated function prototypes by Atmel Studio
12 void toggle();
13 void handle_message(uint8_t len);
14 bool retrieve_relposned();
15 bool check_if_finished();
16 void self_assign_timeslot();
17 //End of Auto generated function prototypes by Atmel Studio
18
19
20
21 // The rover must know its own address and the address of the base
       station
22 // The client address must ALWAYS be > 2 ( 1 - Base, 2 - Pin)
23 #define CLIENT_ADDRESS 3
24 #define BASE_ADDRESS 1
25
26 // Some global definitions for Base Address and Pin Mappings
27 #define RFM95_CS 5 // Chip Select / Slave Select
```

```
28 #define RFM95_RST 6 // SPI Radio Reset
29 #define RFM95_INT 10 // SPI Radio Interrupt
30
31 // Frequency of SPI radio
32 #define RF95_FREQ 434.0
33
34 // External interrupt Pin
35 const uint8_t external_interrupt = 13;
36
37 // FLAGS //
38
39 ///// RTCM ////////////////////////////////////////////////////
40 // this flag signifies that the receiver must prepare to receive
41 // RTCM correction data
42 int rtcm_flag = 0;
43 int rtcm_transmission_time = 500;
44 ////////////////////////////////////////////////////////////////
45
46
47 ///// RELPOS / FINISH //////////////////////////////////////////
48 bool check_for_finish = false;
49 int check_relpos = 0;
50 // this flag signifies that there are TWO valid positions stored
51 // if there is only one valid position, we cannot interpolate
52 int valid_positions = 0;
53 // the boat finish in this period, this flag prepares for the result
54 // to be sent in the next period
55 int boat_finished = 0;
56 // this flag permits the result to be sent in the CURRENT period
57 int can_send_finish = 0;
58 // the timeslot in which the boat can transmit its finish time
59 int timeslot_number = 0;
60 // time in milliseconds of the width of a finish time transmission slot
```

```
61 int finish_time_slot = 30;
62 int guard_interval = 5;
63
64 // used for when finish time is ackowledged, so just wait for next race
65 int idle = 0;
66
67 int current_timestamp = 0;
68 int current_rover_north = 0;
69 int current_rover_east = 0;
70
71 int prev_timestamp = 0;
72 int prev_rover_north = 0;
73 int prev_rover_east = 0;
74 ////////////////////////////////////////////////////////////////
75
76 ////// OTHER FLAGS ////////////////////////////////////////////
77
78 // this value signifies the time at which the timepulse arrived
79 // relative the the number of seconds since MCU program start
80 unsigned long flag_time;
81
82 ////////////////////////////////////////////////////////////////
83
84
85 /// Relative Position & finish Time Buffers /////////////////
86 // the previous relative positoin location
87 uint8_t previous_pos[12];
88 // the most recent relative position location
89 uint8_t current_pos[12];
90 // buffer to store full UBX-NAV-RELPOSNED Message
91 char current_relposned[48];
92 // integer to hold finish time
93 int finish_time = 0;
```

```
94
95  ////////////////////////////////////////////////////////////
96
97  /// Status Message Buffer ////////////////////////////////
98  // full message (2 byte Status, 8 byte Pin)
99  uint8_t size_of_status_message = 8;
100 uint8_t status_buffer[size_of_status_message];
101 uint8_t status_bytes[2];
102 uint8_t pin_location[8];
103 ////////////////////////////////////////////////////////////
104
105 //Sample data
106 uint8_t data[] = "I have finished!";
107
108 //////// SPI Radio ////////////////////////////////////////////////
109 //SPI Radio Buffer - holds all incoming spi radio messages
110 uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
111
112 // Singleton instance of the radio driver
113 RH_RF95 driver(RFM95_CS, RFM95_INT);
114
115 // Class to manage message delivery and receipt, using the driver
       declared above
116 RHReliableDatagram rover(driver, CLIENT_ADDRESS);
117
118 ////////////////////////////////////////////////////////////////////
119
120 // ISR for external interrupt
121 void toggle(){
122     // the boat has finished, no longer cares about RTCM
123     if(boat_finished == 1){
124         rtcm_flag = 0;
125         can_send_finish = 1;
```

```
126         boat_finished = 0;
127       }
128
129     // RTCM flag is set, to preapre for reception over serial
130     else{
131       rtcm_flag = 1;
132       check_relpos = 1;
133     }
134
135     // Set time of flag using the millis() function
136     flag_time = millis();
137   }
138
139 void setup()
140 {
141   // set the mode of the external interrupt pin
142   pinMode(external_interrupt, INPUT);
143   // Attach interrupt
144   attachInterrupt(digitalPinToInterrupt(external_interrupt), toggle,
        RISING);
145
146   //USB for debug
147   Serial.begin(38400);
148   //USART TX and RX pins for sending Received RTCM data to GPS
149   Serial1.begin(115200);
150
151   Serial1.setTimeout(1);
152   while (!Serial) ; // Wait for serial port to be available
153
154   //Initialise manager class and hence the radio with default settings
155
156   ////////////////// Defaults /////////////////
157   /// Frequency 434MHz                    ///
```

```
158   /// Tx Power: 13dBm                    ///
159   /// Bandwidth: 125KHz                  ///
160   /// Coding Rate: 4/5  (1)                 ///
161   /// Spreading Factor: 128 chips/ Symbol(7) ///
162   /// CRC On                             ///
163   ////////////////////////////////////////////
164
165
166   // Note HOPERF RFM96 module uses a PA_BOOST transmitter pin
167   // Power Range: 5 to 23 dBm:
168   // Example rover.setTxPower(23, false);
169   if (!rover.init())
170     Serial.println("init failed");
171   else
172     Serial.println("init succesful");
173 }
174
175
176
177 // function to handle a message (either RTCM or Status)
178 void handle_message(uint8_t len){
179   Serial.print("message");
180   if(buf[0] == 0xD3){
181     Serial1.write(buf, len);
182   }
183   else if(len == size_status_message){
184       memcpy(pin_location, len);
185       Serial.println("PIN");
186     }
187 }
188
189 char preamble;
190
```

```
191 bool retrieve_relposned(){
192     // following the UBX protocol
193     // UBX-NAV_RELPOSNED is 46 Bytes long, the rover is concerned with
            three elements of this message only:
194     // The Northern Offset of the pin from the committee boat
195     // The Eastern Offseet of the pin from the committee boat
196     // The timestamp in ms, generated by the GNSS unit
197
198     if(Serial1.available() > 0){
199
200       preamble = Serial1.read();
201       while(preamble != 0xB5)
202         preamble = Serial1.read();
203       // this is to ensure that there are two valid positions before
              assigning previous_pos a value
204       if(valid_positions == 1){
205          prev_timestamp = current_timestamp;
206          prev_rover_north = current_rover_north;
207          prev_rover_east = current_rover_east;
208       }
209
210       // read the new relative position
211       Serial1.readBytes(current_relposned, 47);
212
213       // extract relevant info from RELPOSNED message
214       int validity_check_north = current_relposned[13] | (
              current_relposned[14] << 8 ) | ( current_relposned[15] << 16 )
              | ( current_relposned[16] << 24 );
215       int validity_check_east = current_relposned[17] | (
              current_relposned[18] << 8 ) | ( current_relposned[19] << 16 )
              | ( current_relposned[20] << 24 );
216       //if North and East are both zero, the position is invalid
217       if(validity_check_north != 0 && validity_check_east != 0){
```

```
218         // only update if position is valid
219          current_timestamp = current_relposned[9] | (
                 current_relposned[10] << 8 ) | ( current_relposned[11] << 16
                 ) | ( current_relposned[12] << 24 );
220          current_rover_north = validity_check_north;
221          current_rover_east = validity_check_east;
222        }
223
224        //Debug Messages
225        Serial.println(current_rover_north);
226        Serial.println(current_rover_east);
227        Serial.println(current_timestamp);
228        //for(int i=0; i<48;i++) Serial.print(current_relposned[i], HEX);
229        Serial.println();
230
231        check_relpos = 0;
232        return true;
233      }
234      else
235        return false;
236    }
237
238  bool check_if_finished(){
239
240     int pin_north = pin_location[0] | ( pin_location[1] << 8 ) | (
           pin_location[2] << 16 ) | ( pin_location[3] << 24 );
241     int pin_east = pin_location[4] | ( pin_location[5] << 8 ) | (
           pin_location[6] << 16 ) | ( pin_location[7] << 24 );
242
243
244     // boat has crossed the line, time to get finish time
245     if(current_rover_east*pin_north > current_rover_north*pin_east){
246
```

```
247         // Here, some calcualtions are required to interpolate for the
                finish time
248         float finish_line_slope = pin_east/pin_north;
249         float rover_slope = (current_rover_east -
                prev_rover_east)/(current_rover_north - prev_rover_north);
250
251         float point_of_intersection_north = (-rover_slope *
                prev_rover_north + prev_rover_east)/(finish_line_slope -
                rover_slope);
252         float point_of_intersection_east =
                (rover_slope)*(point_of_intersection_north - prev_rover_north)
                + prev_rover_east;
253
254         // to find finsh time, see how far alongthe line segment
                (Previous, Next) the point of intersection is. Since the time
                between both
255         // RELPOSNED messages is exactly 1 second, a approximate finish
                time to the nearest millisecond can be calculated.
256         float factor_previous = hypot((point_of_intersection_north -
                prev_rover_north), (point_of_intersection_east -
                prev_rover_east));
257         // find the length between the two relative positions
258         float factor_current = hypot((current_rover_north -
                prev_rover_north), (current_rover_east - prev_rover_east));
259
260         // UBX reports time in milliseconds, so to find the factor between
                to 1 second reports, multiply by 1000 ms
261         finish_time = prev_timestamp +
                (factor_previous/factor_current)*1000;
262         Serial.println(prev_timestamp);
263         return true;
264       }
265     }
```

```
266
267
268 void self_assign_timeslot(){
269   timeslot_number = random(20);
270   boat_finished = 1;
271 }
272
273 void loop() {
274     if(idle == 1){
275        // wait until new race has started (How????)
276       }
277     // time allocated time slot for inbound RTCM is finished, set the
          flag to 0
278     if(rtcm_flag == 1 && (millis() - flag_time > 500)){
279       // Debug message
280       Serial.println("End RTCM");
281       rtcm_flag = 0;
282     }
283     // flag is 1, epect to receive corrections
284     if(rtcm_flag == 1){
285       // first check for RELPOSNED Message
286         if(check_relpos == 1){
287           if(retrieve_relposned()){
288             if (valid_positions == 0){
289               valid_positions = 1;
290             }
291             else{
292               check_for_finish = true;
293               }
294           }
295         }
296         // RTCM flag is one but timeslot has not expired
297         uint8_t len = sizeof(buf);
```

```
298         uint8_t from;
299         if(rover.recvfrom(buf, &len))
300           handle_message(len);
301       }
302       // Boat has crossed the line on the previous period, time
303       // to inform the committee
304       else if(can_send_finish == 1){
305         if(millis() - flag_time >= (flag_time +
              (timeslot_number*finish_time_slot) + guard_interval))
306         // timeslot has expire, finish time has been determined
307         // boat have crossed the line, needs to inform committee
308
309         if(rover.sendtoWait(data, sizeof(data), BASE_ADDRESS)){
310           Serial.println("Ack Received");
311            idle = 1;
312            can_send_finish = 0;
313         }
314         else{
315             // set this to zero so the rover must wait unitl next time
                  period
316             can_send_finish = 0;
317             // reassign a new timeslot
318             self_assign_timeslot();
319           }
320     }
321      else if(check_if_finished()){
322            self_assign_timeslot();
323         }
324
325 }
```

7.7 F. Feather M0 Pin Protocol

```
  /*Begining of Auto generated code by Atmel studio */
#include <Arduino.h>

/*End of auto generated code by Atmel studio */

// Protocol to enable the pin to transmit its relative position to the
    base station
// At the moment this message is unacknowledged
// The relative position is

#include <RHReliableDatagram.h>
#include <RH_RF95.h>
#include <SPI.h>
//Beginning of Auto generated function prototypes by Atmel Studio
void toggle();
void handle_message(uint8_t len);
void compute_relposned();
//End of Auto generated function prototypes by Atmel Studio



#define CLIENT_ADDRESS 2
#define BASE_ADDRESS 1

#define RFM95_CS 5
#define RFM95_RST 6
#define RFM95_INT 10
#define RF95_FREQ 434.0

const uint8_t external_interrupt = 13;

//Globals flags
```

```
32 int rtcm_flag = 0;
33 unsigned long flag_time;
34 int can_send_relposned = 0;
35
36
37 //// Data Buffers ///////////////////
38 uint8_t data[] = "I have finished!";
39 char relposned[46];
40 // array to hold pin position and timestamp
41 uint8_t pin_position[12];
42 // buffer for received messages
43 uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
44 ////////////////////////////////////
45 /////// END BUFFERS ///////////////
46
47
48 // Singleton instance of the radio driver
49 RH_RF95 driver(RFM95_CS, RFM95_INT);
50
51
52 // Class to manage message delivery and receipt, using the driver
       declared above
53 RHReliableDatagram pin(driver, CLIENT_ADDRESS);
54
55 void toggle(){
56     rtcm_flag = 1;
57     flag_time = millis();
58   }
59
60 void setup()
61 {
62   // set the pin mode of the external interrupt
63   pinMode(external_interrupt, INPUT);
```

```
64  // set up interrupt for timepulse
65  attachInterrupt(digitalPinToInterrupt(external_interrupt), toggle,
        RISING);
66
67  //USB for debug
68  Serial.begin(38400);
69  // USART TX and RX pins.
70  // TX for sending Received RTCM data to GPS
71  // RX for receiving the UBX-NAV-RELPOSNED message describing the pins
        location
72  Serial1.begin(115200);
73
74  while (!Serial) ; // Wait for serial port to be available
75
76  //Initialise manager class and hence the radio with default settings
77
78  ////////////////// Defaults /////////////////
79  /// Frequency 434MHz                   ///
80  /// Tx Power: 13dBm                    ///
81  /// Bandwidth: 125KHz                  ///
82  /// Coding Rate: 4/5                   ///
83  /// Spreading Factor: 128 chips/ Symbol ///
84  /// CRC On                             ///
85  ////////////////////////////////////////////
86
87  // Note HOPERF RFM96 module uses a PA_BOOST transmitter pin
88  // Power Range: 5 to 23 dBm:
89  // Example rover.setTxPower(23, false);
90
91  if (!pin.init())
92    Serial.println("init failed");
93  else
94    Serial.println("init succesful");
```

```
95  }
96
97
98  // send RTCM corrections to GNSS Unit
99  void handle_message(uint8_t len){
100   //only send rtcm to the GNSS
101   if(buf[0] == 0xD3){
102     Serial1.write(buf, len);
103     Serial.write(buf, len);
104   }
105 }
106
107 //retrive relevant info from UBX message
108 void compute_relposned(){
109     // following the UBX protocol
110     // UBX-NAV_RELPOSNED is 46 Bytes long, the pin is concerned with
          three elements of this message only:
111     // The Northern Offset of the pin from the committee boat
112     // The Eastern Offseet of the pin from the committee boat
113     // The timestamp in ms, generated by the GNSS unit
114     if(Serial.available()){
115       Serial.readBytes(relposned, 46);
116       memcpy(pin_position, relposned+8 ,12*sizeof(uint8_t));
117     }
118
119 }
120 void loop() {
121     // RTCM period has timed out, set flag to zero
122     if(rtcm_flag == 1 && (millis() - flag_time > 350)){
123       // Debug
124       Serial.println("End RTCM");
125       rtcm_flag = 0;
126       // pin can now send its position
```

```
127       can_send_relposned = 1;
128     }
129    if(rtcm_flag == 1){
130         uint8_t len = sizeof(buf);
131         uint8_t from;
132         if(pin.recvfrom(buf, &len))
133           handle_message(len);
134       }
135      else if(can_send_relposned == 1){
136         compute_relposned();
137         if(pin.sendto(pin_position, sizeof(pin_position), BASE_ADDRESS))
138           Serial.println("Pin Position Transmitted.");
139         //pin has attempted to send position, wait until next cycle
140         can_send_relposned = 0;
141
142       }
143
144
145
146
147
148
149 }
```