A portable sensor rig for multi-sensor data aquisition in maritime environements

Emil Ludvig Martens

Supervisor: Annette Stahl

Spring 2022

NTNU | Department of Engineering Cybernetics

Abstract

Autonomous Surface Vessels require good situational awareness to operate. The standard method to provide this situational awareness is to use data-driven models capable of interpreting data from different types of sensors. Access to representative datasets for training and validation of these models is thus a prerequisite for research on autonomous surface vessels. Previous and ongoing projects, like the Autoferry project, have resulted in high-performing autonomous surface vessels capable of collecting this data. However, using these vessels, like any other ship, involves some amount of work. In this project, a new lightweight sensor platform has been designed and constructed. It is designed to be operable by a single human, making the threshold for going out and collecting data as low as possible. This report presents every step of the design and manufacturing process, to serve as a manual if anyone needs to construct a sensor rig.

Preface

Last fall I started my integrated Ph.D. here at NTNU, where I will research multi-sensor detection for autonomous surface vessels. As teaching duties and a complimentary course filled the schedule of the first semester, this pre-project is the first real project of my Ph.D. I wanted to do something that would be useful for the rest of my research period, and together with my supervisors, we found that having a device for collecting data would be very helpful later on.

With a background working primarily with software development, this project has been an opportunity to learn many new skills. I notably had little to no experience working with Computer-Aided Design (CAD), manufacturing, and embedded programming, which are skills I have always wanted to learn. In this report, I have tried to document everything I have learned throughout this project. Some readers might find parts of this to be trivial, but I hope readers sharing my background find it insightful.

Several people have helped me during this project. First of all, I want to thank my main supervisor, Annette Stahl, for reading through my draft of this thesis and giving useful feedback, as well as my cosupervisors Edmund Brekke and Rudolf Mester. The people at Omega Verksted have been very patient and helped whenever my hardware knowledge fell short, which was a regular event. Ivan Ivanov Gushkov and Hans Theodor Johnsen in particular have both played a significant role, as they have given feedback on my electrical and mechanical design and taught me how to use tools like the laser cutter and drill press without losing any fingers. Finally, I want to thank Tale Softing who has supported me along the way and allowed me to use our living room as a storage facility and makeshift lab for the sensor rig for the last couple of months.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The motivation behind this project is that I'm taking an integrated Ph.D. where I will research how combining data from different sensors, like lidars and special types of cameras, might improve the detection capabilities of Autonomous Surface Vessels (ASV). This work will involve machine learning; thus, there will be a need for a substantial amount of training data. As I will research *new* sensor combinations, it will not be possible to rely exclusively on existing data sets. Therefore a framework for acquiring new data is needed, preferably a framework that makes the threshold for collecting it as low as possible. Hence, the need for this sensor rig.

1.2 Design goals

The goal of this project is to design and build a human-operable sensor rig for multi-sensor data acquisition in maritime environments. Early in the design process, the following set of goals was specified to serve as a guide for the project.

Lightweight The sensor rig should be lightweight enough to be comfortable to carry by hand during data acquisition. This goal puts quite a few constraints on the design, as weight always has to be considered when deciding what materials and components to use. Another goal is to keep the platform's center of gravity close to the point of carrying to make the rig easy to operate.

Waterproof Operating in marine environments is in itself enough to warrant water protection to some degree. As future research most likely will benefit from datasets in adverse weather conditions, like rain and snow, it was decided that the platform should be fully waterproof, that is, having an IP67 rating or better [1].

Rigid The relative poses between the sensors must be known to efficiently combine sensor data from different sensors. The sensor rig should be rigid, such that when these poses are estimated through calibration, one should not have to redo the calibration later on.

Facilitate pose estimation Knowing the pose of the sensor rig is beneficial in many use cases. That is, knowing its position, orientation, velocity, and angular velocity, at any given time.

Automatic labeling of data is a motivating use case. If the camera's position is known for each image taken, then the position of stationary objects in the world frame can be found by labeling only a few images. This will enable automatic labeling of the rest of the images of that stationary object. This will also work on moving objects if their poses are known.

Knowing the pose of the sensor rig will also enable its use in research on Simultaneous Localization And Mapping (SLAM), as it will offer data that can be used as ground truth.

To facilitate accurate pose estimation, the sensor rig should offer high accuracy Inertial Measurement Unit (IMU) measurements and dual GNSS measurements with Real-Time Kinematic positioning (RTK). These measurements should be synchronized, and synchronization with cameras and other sensors should be possible.

Modular Modularity is desired as the sensor rig is a platform intended for research and not commercial production. It should be easy to mount and unmount different sensors on the sensor rig, and attaching the rig to various vessels should also be possible.

User-friendly As a considerable amount of time and effort is spent designing and manufacturing this sensor rig, I hope it will be used frequently. Therefore, the sensor rig should be as user-friendly as possible to facilitate its use. Among other things, this makes it essential to create easily maintainable software for the sensor rig.

1.3 Report outline

Several of the different components of the sensor ring has been developed simultaneously as most of them are interdependent. This made it challenging to structure this report chronologically, and it is therefore structured according to the different categories of work. First, all the mechanical designs are presented. Then a section on navigation sensor theory is followed by the presentation of the NavBox, which is the name given to the small box containing the navigation sensors of the platform, and a section on how communication with the NavBox is achieved. After this, the software and electronics related to the Xavier, the main computer, are presented. Finally, there is a discussion on the different tests that were conducted on the platform as well as a section describing partially finished and future work.

2 Mechanical design

2.1 Fusion 360

Two main options were considered when choosing CAD software; *Fusion 360* and *Solidworks*. *Fusion 360* was chosen as it is known for being more beginner-friendly while *Ssolidworks* is more targeted towards professionals [2].

The following subsections describe methods used to design new parts and create digital models of existing components in *Fusion 360*.

2.1.1 Design from parameters

Most of the designed parts are designed from parameters. That is defining, or measuring, the different dimensions of a part and making a digital model of it in *Fusion 360* with the equal measurements, as is shown in Figure 2a. A caliper was used to find the different measurements of my ordered parts that did not have an available digital model, as shown in Figure 1. The use of parametric design in *Fusion 360* made prototyping of new parts easier, as it lets you define variables – e.g., *tube_diameter*, that can be altered at the end of the design process.

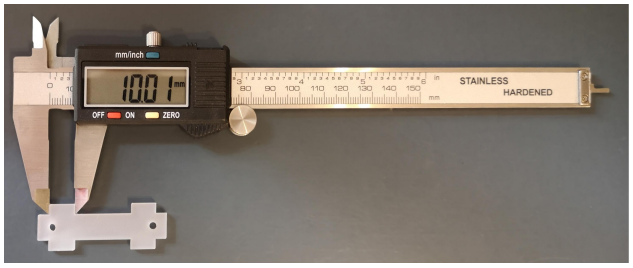


Figure 1: Measuring with a caliper.

2.1.2 Design from shape

Some parts, like the antennas, have non-primitive shapes without clearly defined measurements – e.g., curves and not lines or circles. To create models of these parts, pictures or design sketches were used to draw the shapes using splines, as is shown in Figure 2b.

2.1.3 Available models

A significant benefit of working with CAD software, like *Fusion 360*, is the availability of models. Several of the bought parts of the sensor rig, like the enclosure, had available design files that could be downloaded and used directly [3].

2.1.4 EAGLE files

For some of the circuit boards, there were available *.eagle* files, which is a file format used by the PCB software *Eagle* from *Autodesk*. These files can be opened in *Fusion 360* and exported to pseudo-3D-models as shown in Figure 2c. These pseudo-3D models had the right measurements, holes, and component placements, making it relatively simple to create 3d models of the boards.

2.1.5 Decals

It was helpful to add a photo of some parts as decals on their corresponding 3D models. This was particularly useful on the circuit boards as it made it easy to see where the different signal wires would be going. It also made it easier to create and place component bodies on the boards, like large capacitors and antenna connectors.

Some parts had images available online, while photos were taken of those that did not. As orthonormal images work best, scanning the parts using a photocopier was often the best option, as shown in Figure 2d.

2.2 Production methods

Lacking experience in manufacturing, there are many manufacturing methods I have had to learn throughout this project to be able to create the sensor rig. This section describes some of them.

2.2.1 3D printing

3D printing has become a popular technique of manufacturing for both hobbyists and the industry [4, p.2-3]. It works by depositing successive thin layers of material upon each other, producing a final three-dimensional product [5].

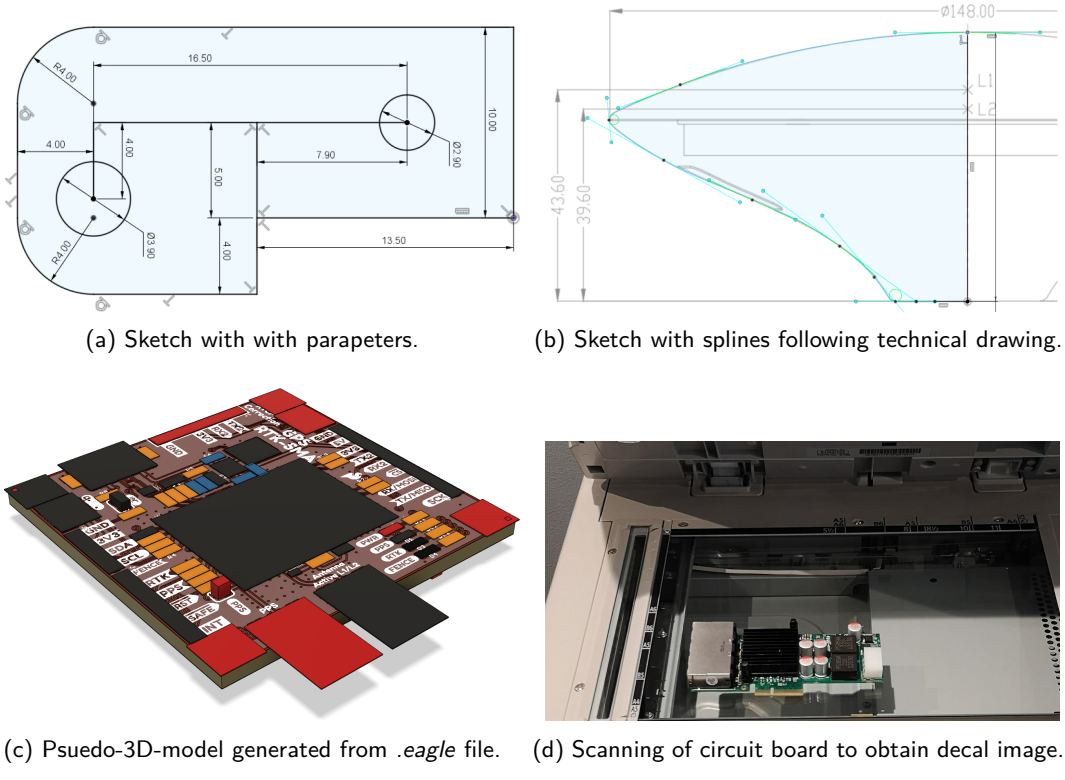
With a 3D printer, most 3D shapes can be made out of plastics within a couple hours [5] at a low price [6].

To print a product, a 3D model of it is needed[1]. In the case of this project, all the printed parts were designed in *Fusion 360*. The 3D models can then be exported as *.stl* files and imported into a program called a *slicer*. For this project, *PrusaSlicer* is used as the parts are printed on an *Original Prusa i3 MK3S+* 3D printer. The slicer finds how the print head should move and when and where the filament should be extruded to produce the part. In the slicer, it is possible to increase printing speed at the cost of quality, which was useful when designing prototypes.

2.2.2 Laser cutting

Laser cutting is a manufacturing method where, as the name suggests, a laser is used to cut different materials. In a laser cutter, mirrors guide a high-power laser to a cutting head that focuses the laser onto the material's surface. This burns a small part of the material away, creating a cut. The head is moved around by a 2D gantry controlled by a computer. Alongside the laser, a jet of air blows away burnt remains and keeps the surrounding area cool, preventing it from melting or catching fire. Figure 3 illustrates this process.

[1] Technically, a model is not required, only G-code



(a) Sketch with with parapeters.

(b) Sketch with splines following technical drawing.

(c) Psuedo-3D-model generated from *.eagle* file.

(d) Scanning of circuit board to obtain decal image.

Figure 2: Intermediate steps of different design processes in *Fusion 360*

Laser cutting is rapid, taking only minutes to cut out most parts, and it offers high precision and small kerf-width. The main disadvantage of laser cutting is that it can only cut 2D shapes out of material.

To cut out parts for this project, the *DXF for Laser* extension was used in *Fusion 360*. It lets you export surfaces of objects as *.dxf* files that can be imported on the computer controlling the laser cutter to cut out parts. A benefit of using *DXF for Laser* is that it compensates for the kerf resulting in cut parts having the exact dimensions as the corresponding 3D models.

The laser cutter used in this project belongs to Omega Verksted (OV). Its kerf width was estimated to be $0.16mm$ by cutting out a $10mm \times 10mm$ part without compensating for the kerf that measured $9.84mm \times 9.85mm$. However, this appears to vary slightly by where the material is placed in the laser-cutter. Time was spent calibrating the laser cutter to make the cut more consistent.

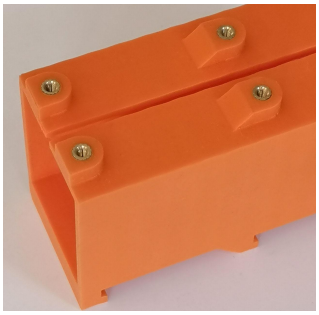
One limit of a laser cutter is that it can not cut through metal. For this project, however, acrylic sheets were considered strong enough. If that was not the case, the backup plan was to use two sheets side by side to double the thickness. Making parts out of aluminum using a *Wazer* water jet available at OV was also an option.

2.2.3 Melt inserts

It can be desirable to be able to screw bolts into 3D printed or laser-cut parts. In this project, melt-inserts were used to achieve this. Melt inserts can be described as threaded nuts with small fins on the outside, as shown in 4a. The melt inserts are pushed into slots in the parts with a soldering iron which heats up the insert causing the surrounding plastic to melt. When the soldering iron is removed, the plastic stiffens, and the melt insert is fixed in place.

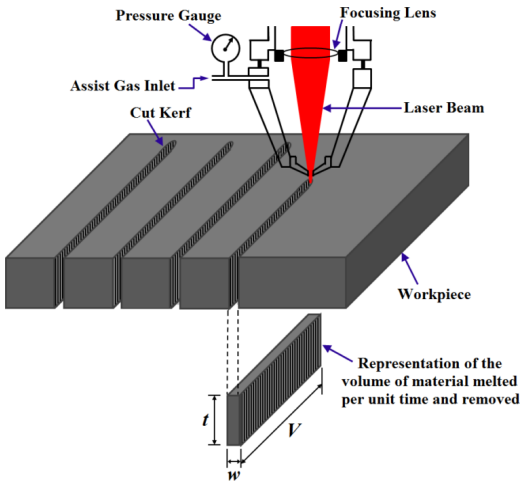
When it is not possible to use melt inserts, for instance, when a part is made out of metal or is missing the required holes, another option is to use a tap to create threads. A tap cuts away material, creating threads for bolts to be screwed into later.

(a) Close up [8].

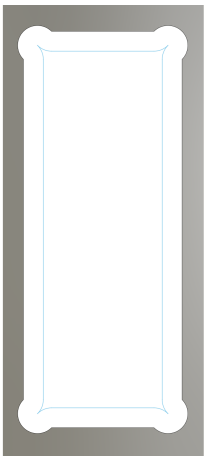


(b) Melt inserts used in

Figure 4: Melt inserts.



(a) Laser cutting illustration showing kerf width w [7].



(b) Cutout with dog bone fillets and exagurated kerf width. The blue line is followed by the laser.

Figure 3: Visualization of laser-cutting.

2.2.4 Glue

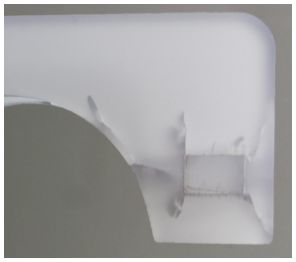
Three different types of glue, shown in Figure 5, were tested to find a good way of attaching laser-cut bodies together. First, a product called *plastlim* from *Clas Ohlson* was tested. This glue claimed to chemically melt the parts together and form a tight bond. The glue appeared to satisfy those claims but had the disadvantage of being really viscous, making it difficult to apply to some of the desired areas. It also almost always resulted in visible remains that were not cosmetically appealing.

Tamiya Extra Thin Cement is a glue containing acetone designed for working with plastic model kits [9]. The acetone chemically melts the plastic together forming a solid connection. One advantage of this glue is its low viscosity allowing the capillary effect to draw it into cracks from the outside. Unfortunately, the laser-cut acrylic parts tended to form cracks after being exposed to the glue or its vapor. Figure 5b shows the most severe case of this. A theory is that the rapid heating and cooling from the laser-cutter cause tension to build up in the cut surfaces, which the acetone releases in the form of cracks. If this is the case, then heating the parts to $\sim 104.85°$, the glass transition temperature of the acrylic sheets [2] [11], might remove the tension and allow the use of this type of glue. This has yet to be tested.

The last glue tested, *superlim*, was easier to apply than the *plastlim* and did not cause cracking like *Tamiya Extra Thin Cement*. Out of the three, this was the best option but was found to be unnecessary as interference fits solved the problem.

[2]The acrylic sheets are made of Poly(methyl methacrylate) (PMMA) [10]

(a) Three types of glue testes.



(b) Extreme case of cracking due to acetone exposure.

Figure 5: Construction useing glue.

2.2.5 Interference fit

Interference fit, also known as friction fit, relies on the parts fitting together so well that the friction in the interfaces is enough to hold them together. A nail holding two planks together is an example of this.

For the laser cut parts to hold together, it was necessary to compensate for the kerf-width of the laser, as described in Section 2.2.2. It was also required to add dogbone fillets, as shown in Figure 3b. They were added using the *Nifty Dogbone for Autodesk® Fusion 360™* extension [12] in *Fusion 360*.

A significant advantage of interference fits is that it removes the need for any other components than the acrylic sheet and allows rapid prototyping since it is no longer necessary to wait for the glue to cure.

To get a stronger interference fit, friction tape can sometimes be used. On the sensor rig, it is whenever something clamps onto something else, strengthening the clamping action.

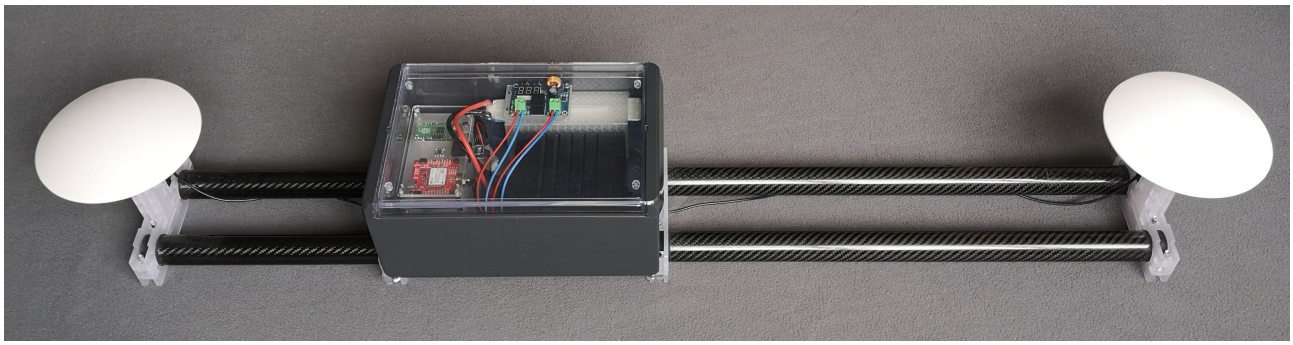


Figure 6: Photo of the final mechanical design of the sensor rig.

2.3 Design of the base

This section covers the mechanical design of the base, that is everything outside of the enclosure.

2.3.1 Carbon fiber tubes

The first part of designing the sensor rig was to decide what mechanical framework to use. The initial idea was to use standard $40mm \times 40mm$ aluminum profiles. They have the advantage of being easy to work with as they have built-in grooves for attachment. However, with a weight of $1.5kg/m$ [13], two one-meter profiles would add $3kg$.

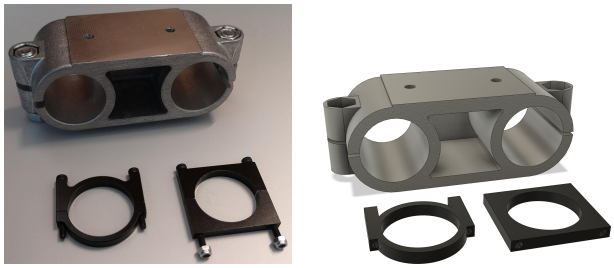
To achieve low weight, two $1m$ carbon fiber tubes with $30mm$ diameter and $3mm$ thickness are used, as they offer high strength and are more than five times lighter than the aluminum profiles at only $193.8g/m$ [14]. The tubes are depicted in Figure 6. The main disadvantage of this option is the lack of easy attachment options. The price is also higher but still small compared to the cost of industrial sensors.

2.3.2 Tube clamps

As the carbon fiber lacked the attachment groves on the aluminum profiles, it was necessary to find a way to mount them together and a way to mount components on them. Several readily available parts were tested, without success, before custom parts finally were designed.

Failed products Clamps from *Rose+Krieger* were tested as a way to keep the two tubes parallel. The type of clamp is depicted in the top half of Figure 7a and 7b. The clamps served their purpose to some degree. Unfortunately, as they had a clearance of only $0.1mm$ [15], they left scratch marks when threaded onto the carbon fiber rods. It was attempted, without success, to solve this by sanding the clamps with P240 sandpaper. The lack of clearance also offered no space for any friction tape, resulting in sub-optimal clamping action. It's also worth noting that the top mounting holes were not threaded when the clamps arrived. Thus they were threaded manually.

As the previous product did not work two new types of smaller clamps were ordered, depicted in the bottom half of Figure 7a and 7b. These clamps consisted of a top and bottom half and were quick and easy to attach anywhere. They appeared to work quite well, but they were also too tight to use with friction tape. They also presented no obvious way to rigidly keep the tubes parallel, and they had a build quality matching their low price.



(a) Photo (b) Drawing

Figure 7: Different clamping parts that were tested but not used.

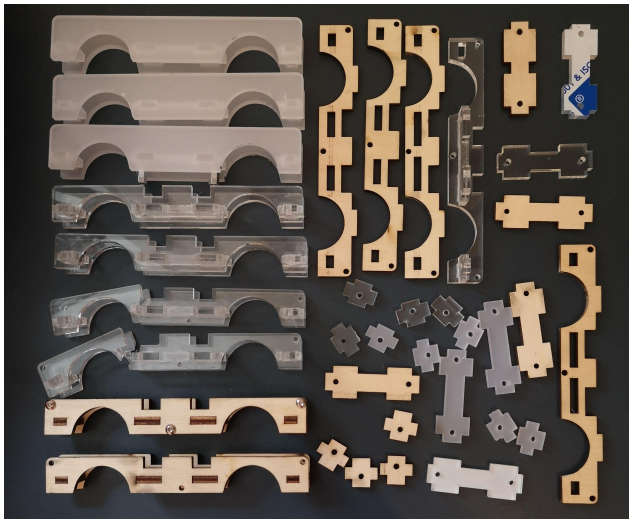
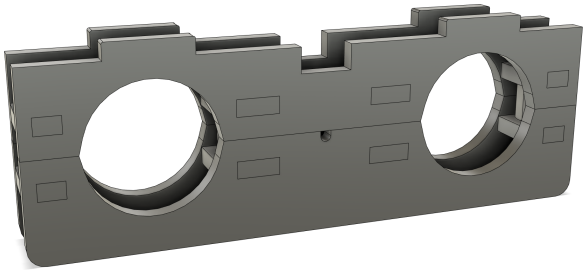
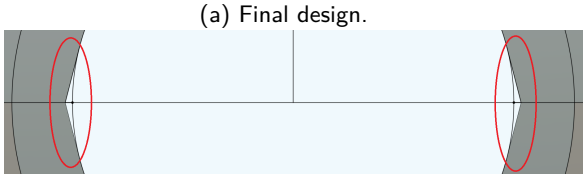


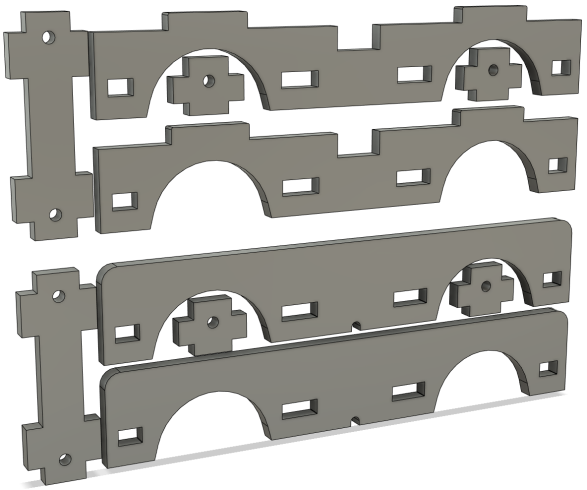
Figure 8: A selection of tube clamp prototypes.



(a) Final design.



(b) Indentations (encircled) in design for ease of use with friction tape..



(c) Bodies of final design.

Figure 9: Tube clamp

Final design Figure 9 shows the final design of the tube clamp. Even though the clamps discussed previously did not work, they offered inspiration to the final design. The final clamp design goes around both carbon fiber tubes, similar to the first clamp, and consists of a top and bottom half like the last ones.

Each half consists of five bodies laser-cut out of $4mm$ acrylic sheets. All the individual bodies are shown in Figure 9c. The protrusions on the top half fit into a mounting plate discussed later, and the notch is there for camera cables. The tiny hole in the middle is for the coaxial cable connected to theGNSS antennas.

The final design has tight enough tolerances to allow for interference fits between the bodies, removing the need for glue. The top and bottom half are tightened together using standoffs. The inner diameter of the clamp is $0.6mm$ wider than the carbon fiber tubes, giving just enough clearing for friction tape, and the $0.5mm$ indentations shown in Figure 9b solved the problem of tape getting stuck between the top and bottom half.

This part was the first one designed in Fusion 360 and manufactured from scratch; thus, several bad designs and products were made before the final result was reached. Roughly 15 prototypes were manufactured before ending up with the final result. Some of them are shown in Figure 8. The first set of prototypes was made out of wood to reduce costs.

2.3.3 GNSS antenna

The antenna model is drawn from its datasheet [16, p.2]. The two points in the upper half of Figure 11 show the phase centers of the internal L1 and L2 antennas, descibed in the F9P integration manual [17, p.114].

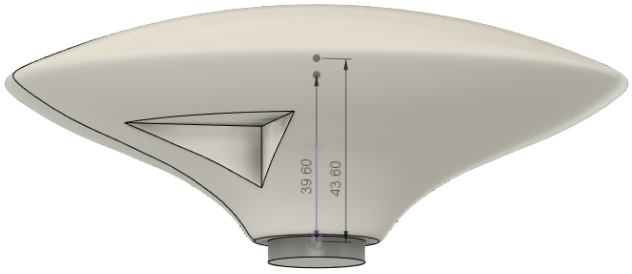


Figure 11: Antenna model.

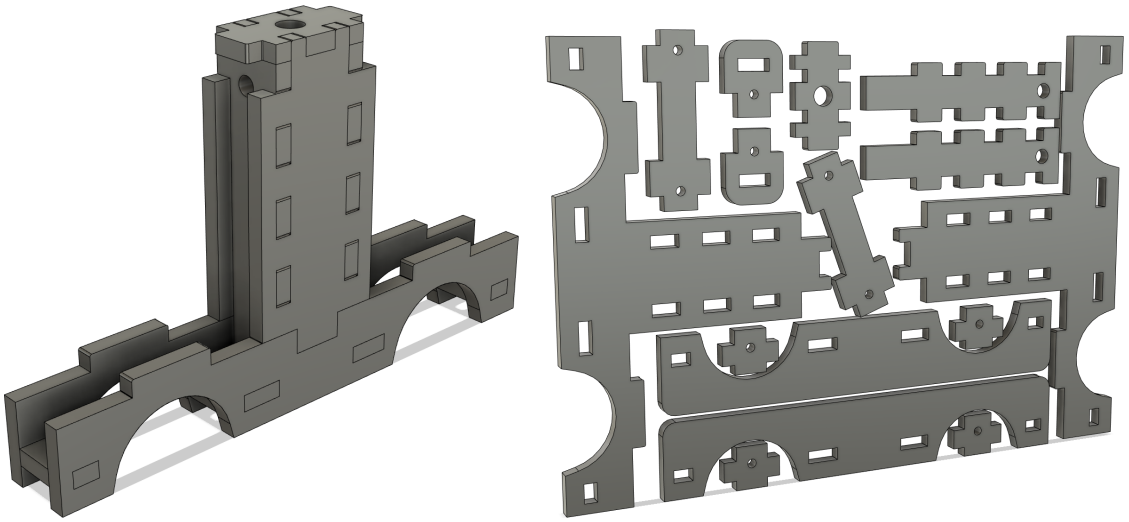
2.3.4 GNSS antenna mount

Using the *9.solutions Camera Kit*, shown in Figure 12, to attach the antennas was initially tested. These mounts have good build quality, fit nicely onto the carbon fiber tubes, and made the antennas easy to attach and detach. However, they were not tall enough for the camera to be mounted directly next to the antennas, so they were not used.

The finalGNSS antenna mount, shown in Figure 10 is based on the top half of the tube clamp and it is attached to the same type of bottom half as the tube clamp. The top part consists of 10 bodies, while the bottom half still consists of 5 bodies. Figure 10b shows all the bodies of theGNSS antenna mount.

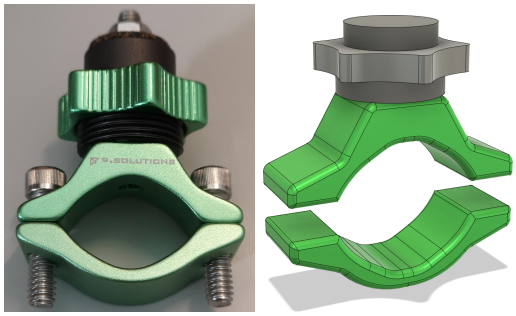
This part also relies on interference fits, and the *FingerJoints* extension [18] was used to generate the slots in the side. The antenna is attached to the top part of the mount with a $1/4$" bolt via a $1/4$" to $5/8$" adapter. The central part of the antenna mount has melt inserts to which this top part is attached.

The resulting construction was stiffer than anticipated and considering the variance in the GNSS measurements, the slight possible deflection in this antenna mount is negligible.



(a) Final design. (b) Bodies of final design.

Figure 10: GNSS antenn mount



(a) Photo (b) Drawing

Figure 12: Mounts from 9.solutions.

Figure 13: Photo of final anetnna mount.

2.3.5 Enclosure mount

The enclosure from *Boopla* is attached via two tube clamps, one on each side, via four small adapters, laser-cut out of $5mm$ acrylic sheets, as shown in Figure 14. This attachment method resulted in the enclosure touching the carbon fiber rods, thus making the center of gravity of the sensor rig as low as possible without having the rods on the outside of the box. The enclosure is placed off-center so the rig can be carried using one hand.

2.3.6 Future sensor mounts

The tube clamp is designed so it is easy to mount sensors to the sensor rig via mounting plates. Sensor mounts for mounting future cameras on the sides next to the GNSS antennas are designed, giving the cameras a baseline of almost $1m$. Figure 16 shows a 3D model of two *Triton* cameras, mounted next to one GNSS antenna on one of the sides of the sensor rig, allowing multiple focal lengths. The depicted cameras have an IP67 rating [19], removing the need for waterproof enclosures.

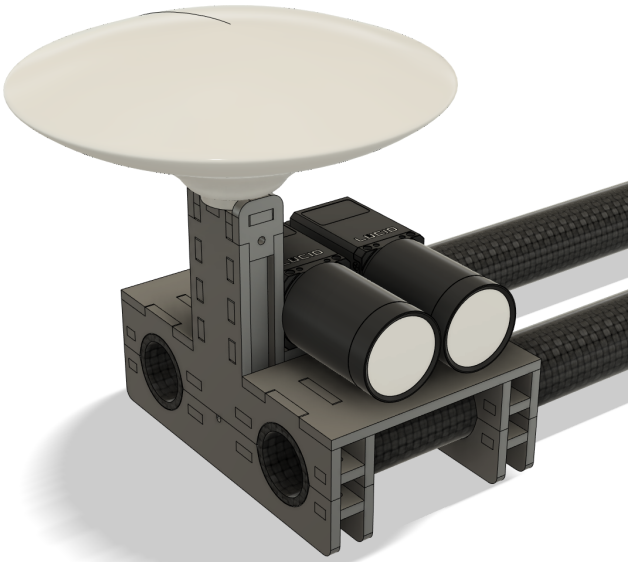
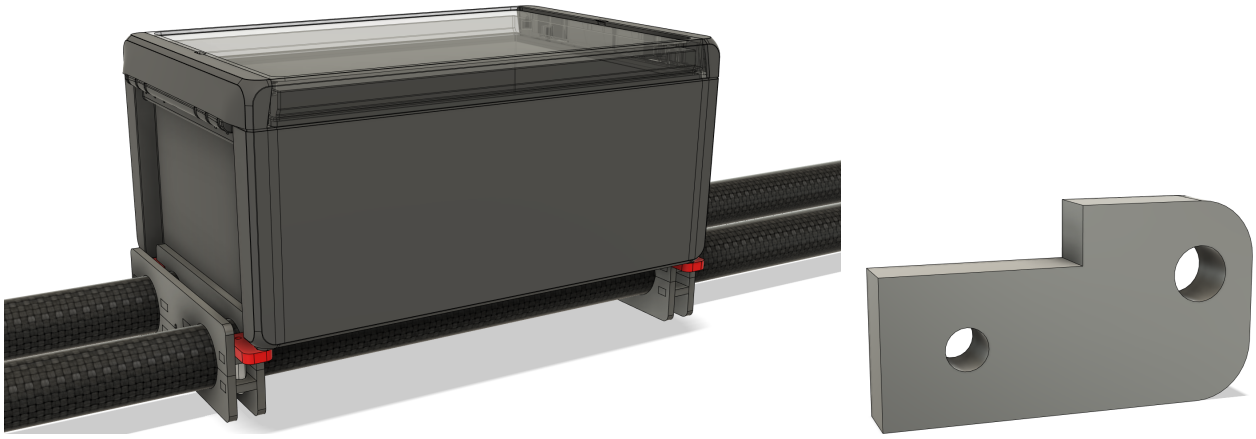


Figure 16: Sensor mount design.



(a) Final design.

(b) Adapter (red in 14b).

Figure 14: Enclosure mount.

2.4 Design of the central hub

This section discusses the mechanical design of the central hub and its internal components.

2.4.1 The enclosure

A waterproof enclosure was deemed necessary to house the different components to make a waterproof sensor rig. An enclosure called *Bocube* from *BOPLA* was chosen as it had a transparend lid and offered an IP68 rating when closed [20]. Variants existed in many different shapes and sizes and all had available 3D models. Before it was ordered, a 3D model of it with all the internal components was made to make sure everything would fit. The only downside with the product was the lack of threads in the attachment points, making it necessary to use a tap to create them manually. Screws could have been used instead of bolts, removing the need for threads, but they would probably have to be permanent.

2.4.2 Main mount plate

All the components inside the box are mounted on a shared laser-cut mounting plate, making it easy to take everything out of the enclosure. This shared mounting plate is srewed onto 6 melt inserts fixed in an intermediate laser-cut mounting bracket, which is screwed onto the threaded attachment points of the enclosure, as shown in Figure 17 Figure 15 shows an intersection view of the enclosure with its components, visualizing this attachment scheme.

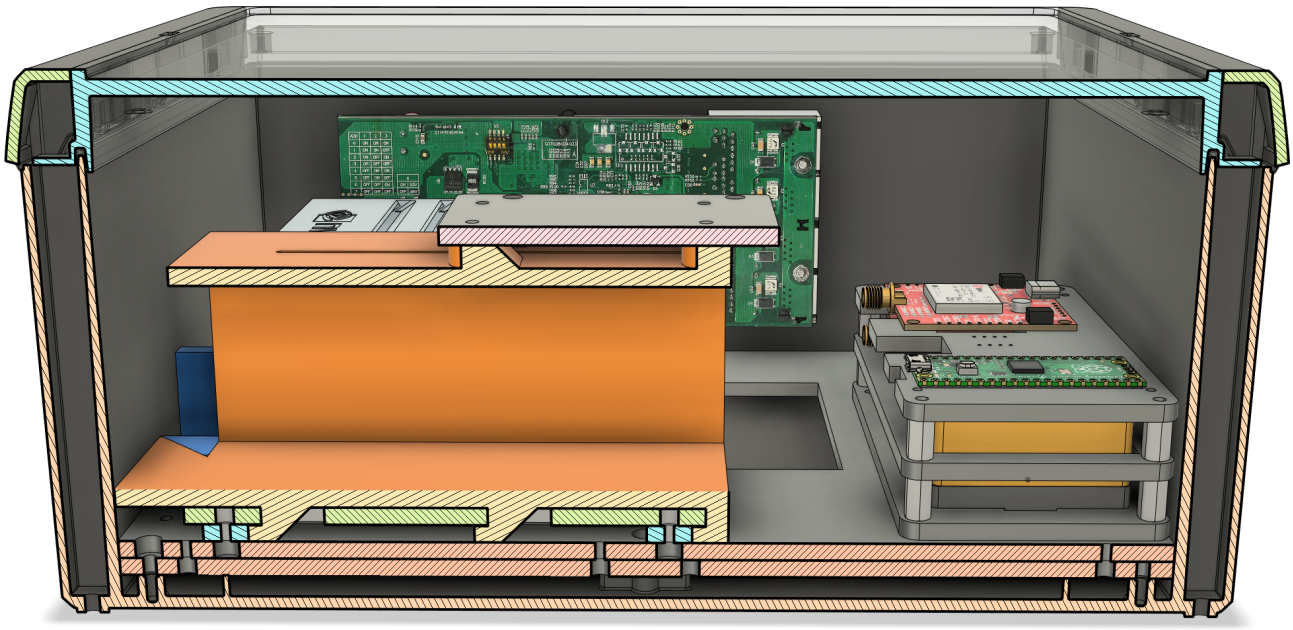


Figure 15: Intersection view of enclose.

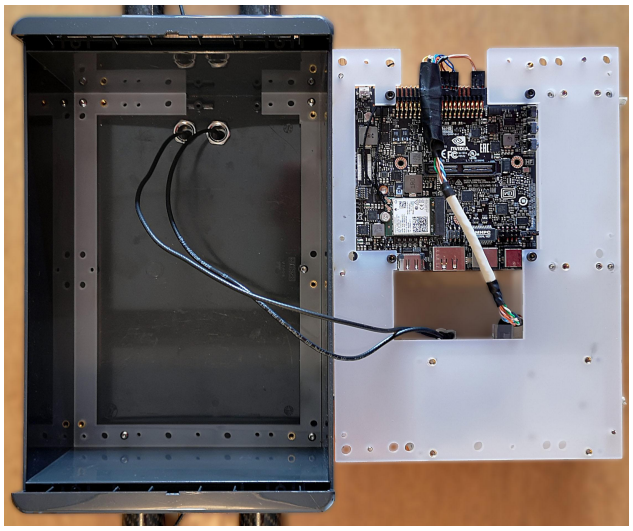


Figure 17: From left to right:
Intermediate mounting bracket attached inside enclosure.
Bottom view of shared mounting plate.

2.4.3 Xavier mount

The legs have been removed from the Xavier, and it is mounted on the shared mount plate, as shown on the right in Figure 17. Four $4mm$ spacers are added between the Xavier and the shared mount plate, and space has been cut out for the cable connecting the Xavier connector to the NavBox.

The model of the Xavier is available online [21], however, it contains several hundred individual bodies as every resistor and capacitor was modeled, making *Fusion 360* slow when the model is open. Fortunately, *Fusion 360* has a built-in *select-by-size* tool that makes it relatively easy to simplify the design by removing all the tiny bodies.

2.4.4 Battery mount

The 3D printed battery mount is designed to house the 4-cell LiPo battery discussed in Section 6.7, and fit between the Xavier and the enclosure wall. The walls are thin enough to flex a little bit, and the slit in the top makes it possible to pry them slightly apart, making the insertion of the battery easy. Friction tape is used to hold the battery in place without the need for any pressure. The battery holder is designed to slide and lock onto a mounting plate shown in Figure 18. A locking part, shown in blue in Figure 18 is screwed onto the shared mounting plate, preventing the battery holder from sliding off. This design makes it easy to attach and detach the battery from the sensor rig, which is helpful as it makes charging easier.

The bottom part of the lipo battery is not inside the enclosure. This is to make sure swelling of the battery is detected, as this is an indication that the battery should be disposed of and replaced with a new one [22].

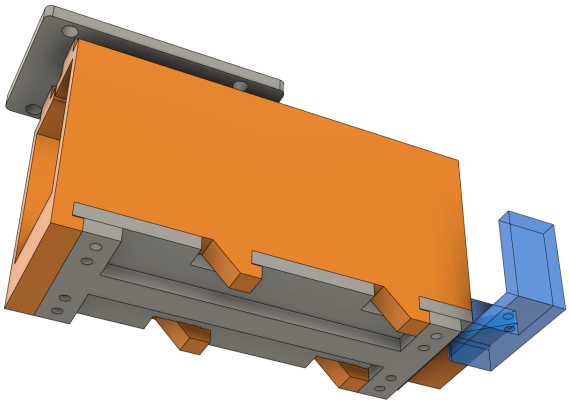


Figure 18: Battery mount design

2.4.5 NavBox

The NavBox is designed to work as a detachable module that can be mounted on the shared mount plate as shown in the lower right side of Figure 15. The NavBox is mounted in the corner between two anchoring points to be rigidly mounted. The STIM300 is attached to the bottom plate, while the Pico, the two F9Ps, and the *RS422 to RS232 converter* are attached to the upper plate. Space has also been cut out for the female RJ45 connector. On top of the NavBox, an acrylic plate is added for protection as can be seen in Figure 6. It is transparent to not obstruct the view of the indicator LEDs on the Pico and the top F9P.

2.4.6 Waterproofing

Several cables connect sensors outside the enclosure to components inside the enclosure. To keep the enclosure waterproof, these cables are connected through the walls of the enclosure using cable glands as shown in Figure 19. A challenge with normal cable glands is that connectors thicker than the cable itself might be possible to get through. This makes it necessary to make custom cables, like the custom terminated coaxial cables described in Section 4.1.5. A type of cable gland that can be retrofitted around thicker cables has been acquired to be used with future CAT-6 cables.

Figure 19: Cable glands on bottom side of enclosure.

3 Navigation sensors

3.1 GNSS theory

GNSS is a general term for satellite constellations providing positioning, navigation, and timing services [23]. Given the distance to three or more such satellites, the position of a receiver on earth can be calculated through triangulation as shown in Figure 20. In practice, at least four satellites are needed, as the clock offset of the receiver also has to be estimated [24, p.81]. How the position and clock offset is calculated, through the evaluation of pseudoranges, is well described in Section 6 of *Essentials of Satellite Navigation* [24, p.81-100].

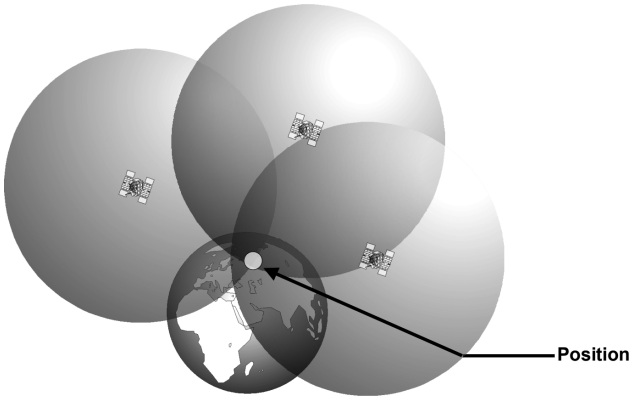


Figure 20: Visualization of GNSS navigation [24, p.16].

There are several sources ofGNSS error [24, p.90, 100–101]. Table 1 shows approximate error contributions from the different error sources. Fortunately, there are ways to improve the accuracy.

Error cause	Error without DGPS
Ephemeris data	$1.5m$
Satellite clocks	$1.5m$
Effect of the ionosphere	$3.0m$
Effect of the troposphere	$0.7m$
Multipath reception	$1.0m$
Effect of the receiver	$0.5m$
Total RMS value	$\mathbf{4.0}m$

Table 1: Error causes for GPS (typical ranges) [24, p.91]

3.1.1 Multi-frequancy GNSS

Using two different signal frequencies, e.g., L1/L2, the error caused by the ionosphere can be compensated for, as the ionosphere's effect on a signal depends on its frequency [24, p.102]. This is supported by GPS and GALILEO [24, p.102].

3.1.2 Differential GNSS

Several sources of error, like the satellite clocks, have the same effect on nearby receivers. Thus the relative position between receivers has better accuracy than the absolute position [24, p.103]. Working with relative positions is often called Differential GNSS (DGNSS) [25]. If both receivers are capable of measuring the phase of theGNSS signals, centimeter-level accuracy can be achieved for the relative position of the receivers [24, p.103].

Suppose the position of one receiver is accurately known, such as if it is stationary and has collected measurements over a long period. In that case, the relative position between it and a moving receiver can be used to find the position of the moving receiver with centimeter accuracy. For this to work in real-time, there must be a communication link between the receivers where correctional messages can be transferred from the base to the moving receiver. These correctional messages generally use the RTCM protocol discussed in Section 5.6. This type of positioning is called RTK.

The relative position between two moving receivers mounted on the same vehicle can be used to accurately find the heading of the vehicle [26, p.7]. This type of operation is called moving base operation [26, p.4]. It can be used together with RTK to accurately find both the position and heading of a vehicle in real-time [26, p.5].

3.2 The F9P sensor

The F9P is a GNSS receiver developed by *u-Blox*. It supports both multi-frequency and differential GNSS and is capable of achieving centimeter-level accuracy operating with RTK [27, p.4]. Because of this, its simple and well-documented user interface, and comparatively low price, it appears to have become the go-to product for many makers and researchers.

The F9Ps hve been configured, using the *u-center2* software from *u-blox*, for a moving base setup, inspired by the *Moving base applications* [26] application note. This is further described in Section 5.6. As shown in in Table 2, the top F9P outputs raw data (RAWX), global position data (NAV_PVT) as well as correctional RTCM messages. The bottom F9P configuration is similar, except that it outputs relative position data, rather than listens for RTCM messages rather than outputting them. The raw data is intended to be used for postprocessing, as discussed in Section 8.4.

	Keyname (Key ID)	Layer	Value
1	CFG-SPI-ENABLED	FLASH (2)	1
2	CFG-SPI-CPHASE	FLASH (2)	1
3	CFG-SPIOUTPROT-UBX	FLASH (2)	1
4	CFG-SPIOUTPROT-NMEA	FLASH (2)	0
5	CFG-SPIOUTPROT-RTCM3X	FLASH (2)	1
6	CFG-TP-POL_TP1	FLASH (2)	0
7	CFG-UART2-ENABLED	FLASH (2)	1
8	CFG-UART2-BAUDRATE	FLASH (2)	460800
9	CFG-RATE-MEAS	FLASH (2)	125
10	CFG-RATE-TIMEREF	FLASH (2)	0
11	CFG-MSGOUT-UBX_RXM_RTCM_SPI	FLASH (2)	1
12	CFG-MSGOUT-UBX_RXM_RAWX_SPI	FLASH (2)	1
13	CFG-MSGOUT-UBX_NAV_PVT_SPI	FLASH (2)	1
14	CFG-MSGOUT-RTCM_3X_TYPE4072_0_SPI	FLASH (2)	1
15	CFG-MSGOUT-RTCM_3X_TYPE1074_SPI	FLASH (2)	1
16	CFG-MSGOUT-RTCM_3X_TYPE1084_SPI	FLASH (2)	1
17	CFG-MSGOUT-RTCM_3X_TYPE1094_SPI	FLASH (2)	1
18	CFG-MSGOUT-RTCM_3X_TYPE1124_SPI	FLASH (2)	1
19	CFG-MSGOUT-RTCM_3X_TYPE1230_SPI	FLASH (2)	1
20	CFG-MSGOUT-RTCM_3X_TYPE4072_0_UART2	FLASH (2)	1
21	CFG-MSGOUT-RTCM_3X_TYPE1074_UART2	FLASH (2)	1
22	CFG-MSGOUT-RTCM_3X_TYPE1084_UART2	FLASH (2)	1
23	CFG-MSGOUT-RTCM_3X_TYPE1094_UART2	FLASH (2)	1
24	CFG-MSGOUT-RTCM_3X_TYPE1124_UART2	FLASH (2)	1
25	CFG-MSGOUT-RTCM_3X_TYPE1230_UART2	FLASH (2)	1

Table 2: Configuration of top F9P.

3.3 IMU theory

An Inertial Measurement Unit (IMU) is a type of interoceptive sensor, capable of measuring acceleration and angular velocity [28, p.169]. Sometimes they also include a magnetic or gyroscope compass but this is not the case for the IMU used in this project.

Most modern IMUs are based on Micro-Electro-Mechanical Systems (MEMS) technology [29]. They work by measuring the deflection of microscopic mass-spring-damper systems, caused by accelerating and centrifugal forces. From these deflections, the acceleration and angular velocity of the IMU can be estimated.

Generally speaking, IMU measurements contain two types of noise. One can be approximated as white noise and is often specified by how much its integral changes over time, e.g. velocity random walk for acceleration noise. The other type of noise is bias noise which is characterized by its stability, that is how much it statistically changes over time.

3.4 The STIM300 sensor

The STIM300 is an industry-grade IMU featuring an accelerometer, inclinometer, and gyroscope [30, p.1], offering very accurate measurements, as shown by the low random walk and bias instability values in Table 4. It was chosen based on its performance and the fact that the faculty had one available. It has been configured, over UART, with the configurations shown in Table 3.

Option	Value
Sample rate:	5 = External Trigger
Filter bandwidth:	4 = $262Hz$
Gyro output unit:	0 = Angular Rate $[^\circ/s]$
Acc. output unit:	0 = Acceleration $[g]$
Incl. output unit:	0 = Acceleration $[g]$
Gyro g-comp	7 = ACC
Datagram	7 = Gyro Acc Inc Temp
Bit-rate:	3 = $921600bits/s$
Datagram termination	1 = OFF

Table 3: Configuration of STIM300 used for the sensor rig [30, p.119].

Parameter	Min	Nom	Max	Unit
GENERAL				
Weight		55		g
Operating temperature	-40		85	°C
Supply voltage	4.5	5.0	5.5	V
Power consumption		1.5	2	W
Time to valid data		0.7	1	s
Sample rate			2000	SPS
Mechanical shock, any direction			1500	g
RS422 transmission bit rate			5.18	Mbit/s
Misalignment		1		mrad
GYRO				
Input range		±400[1)]		°/s
Non-linearity (condition: ±200 °/s)		15		ppm
Resolution		0.22		°/h
Bias instability		0.3		°/h
Angular random walk		0.15		°/√h
Bias error over temperature gradients		±10 [2)]		°/h rms
Linear acceleration effect				
Bias (no g-compensation)		7		°/h/g
Bias (with g-compensation)		1		
Scale factor (no g-compensation)		400		ppm/g
Scale factor (with g-compensation)		50		
Scale factor accuracy		±500		ppm
ACCELEROMETER				
Input range		±10 [3)]		g
Resolution		1.9		µg
Bias instability		0.04		mg
Velocity random walk		0.07		m/s/√h
Bias error over temperature gradients		±2 [2)]		mg rms
Scale factor accuracy		±200		ppm
INCLINOMETER				
Input range		±1.7		g
Resolution		0.2		µg
Scale factor accuracy		±500		ppm

1) Optional ranges are available 2) Condition: ΔT ≤ 1°C/min 3) Optional ranges: ±5 g, ±30 g and ±80 g

Table 4: Parameter specifications of the STIM300 [31].

4 Navbox

The module of the sensor rig responsible for collecting the data required for accurate pose estimation is called the NavBox. This section covers its electrical design, This section covers the theory behind its sensors, how the measurements are synchronized and how it communicates with the Xavier.

4.1 Electrical design

The complete wire diagram for the NavBox and its connection to the Xavier is shown in Figure 27. Every time something did not work, there were many potential sources of error. It could be drivers or software bugs on the Xavier, error in the C code on the Pico, a broken or shorted connection, or lack of power to one of the logic converters, to name a few. Consequently, the development was done iteratively and relied heavily on using an oscilloscope, as shown in Figure 21.

The following sections describe the different parts of the electrical design process chronologically.

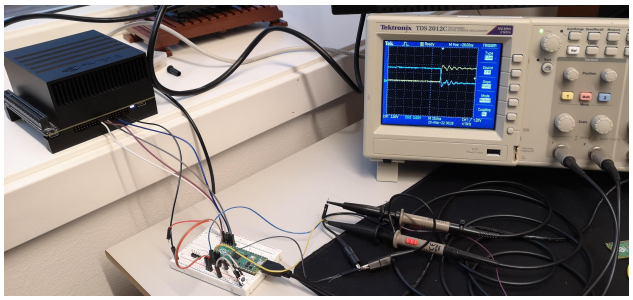


Figure 21: Photo of a temporary setup during development of the NavBox.

4.1.1 Custom STIM300 cable

When the STIM300 was acquired, it appeared to be broken at first, as it did not output any meaningful messages. After debugging, involving acquiring a second STIM300 for sanity checking, the culprit was found to be a broken interface cable. The previous user had taped the cable together and left it in the box without leaving a note.

As the interface cable was broken, a new one was made. The cable was cut before the damaged part to reuse the Micro-D connector needed to connect to the STIM300 [30, p.37] and terminated with a female RJ45 connector as shown in Figure 22. The final cable is shown in the bottom-left part of Figure 23 and its wiring is shown in the bottom-right corner of Figure 27.

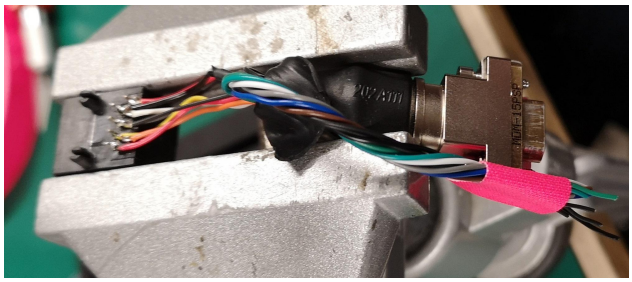


Figure 22: New cable for STIM300.

4.1.2 Communication with STIM300

As the STIM300 communicates over RS422 [30, p.1], a converter is used to translate the signal to RS232. RS422 is a more robust version of RS232 using differential signals [32, p.3], which is not directly supported by the Pico [33, p.4]. The DS8921A converter [34] was soldered onto a small breakout board and attached to the NavBox between the Pico and the top F9P by pressing PCB headers through cutout holes. The converter's intended input is $5V$ [34], but it was tested to work at $3.3V$ as well, as the Pico does not support $5V$ input [33, p.17].

The STIM300 is then connected to the rest of the NavBox by three detachable white JST connectors as shown in Figure 23. One connector for the data cables, one for power, and one for trigger and TOV. JST connectors were used rather than jumper wire connectors as the JST connectors have a lower profile.

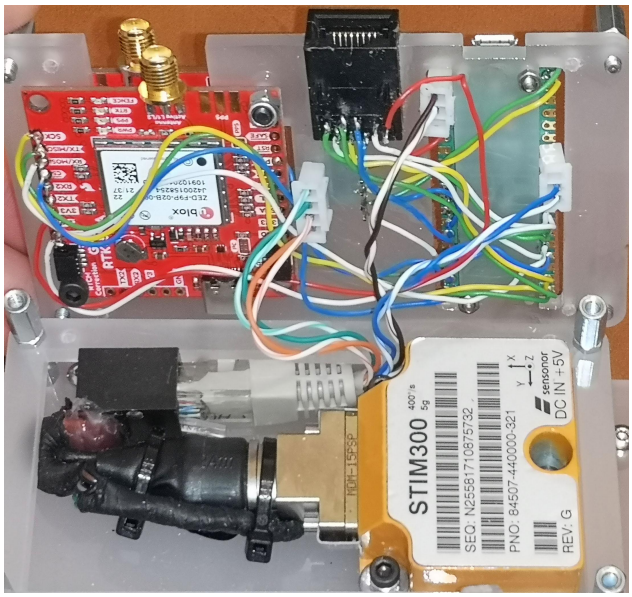


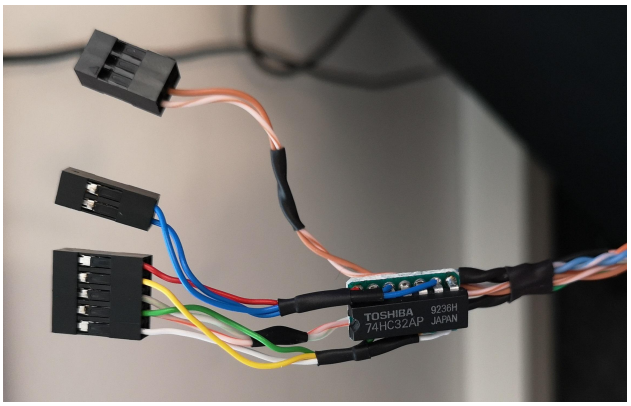
Figure 23: Photo of cable wiring inside the NavBox

4.1.3 Cable with analog signal buffering

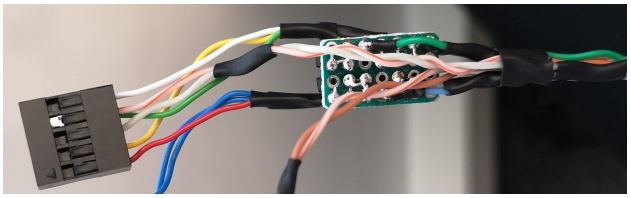
All the communication between Xavier and the NavBox, as well as the power, goes through on cable. The cable is terminated on the NavBox side with a male RJ45 connector, that plugs into a female RJ45 connector on the

NavBox shown in the top center of Figure 27. This makes the box easily attachable and detachable. On the Xavier side, the cable is terminated with female jumper wire crips that connect to pins on the 40-pin GPIO-header.

Inside the cable, a 74HC32 OR-gate [35] is used as an analog buffer. An analog buffer is a device that outputs a voltage that follows the voltage from an input source, but supplies drive current from another power source [36]. Section 4.1.4 explains why an analog buffer is needed. The 74HC32 provides four independent OR-gates [35, p.1] and is connected to to the cables via a protoboard as shown in Figure 24. The cable wiring is shown in the top part of Figure 27. The whole thing is wrapped in tape and connected to the Xavier as shown in Figure **??**.



(a) Top view.



(b) Bottom view.

Figure 24: Buffer cable.

4.1.4 Need for analog buffer

Occasionally the Serial Peripheral Interface (SPI)-signals from the Xavier started to oscillate rapidly when they applied voltage, as shown in Figure 25. The source of the bug was difficult to locate, as the bug would seem to solve itself but reappear later. After consulting the datasheet, the problem was identified as a lack of drive current from the Xavier. Several of the pins on the Xavier 40-pin GPIO-header only has merely $1\mu A$ drive current [37, p.26]. How this caused oscillations is unknown, but it appeared to depend on the impedance in the cables as it only occured when long cables were used, which explains why it sometimes disappeared. Using an analog buffer as described in Section 4.1.3 removed the oscillations and solved the problem.

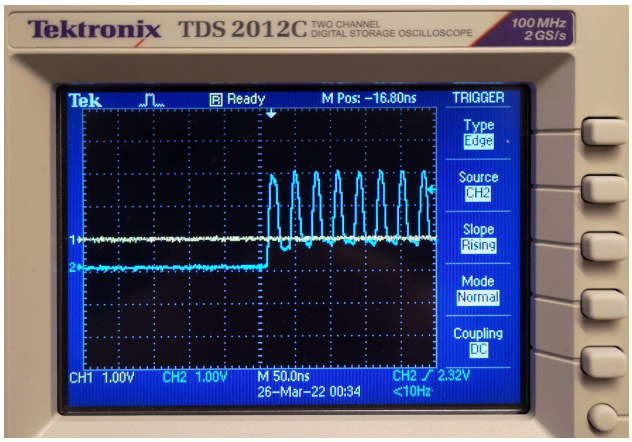
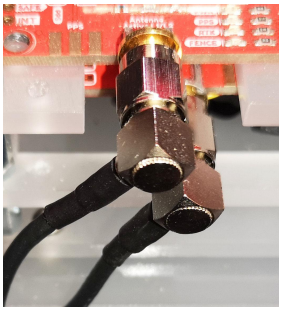


Figure 25: $\sim 33Mhz$ oscillations due to lack of drive current [37, p.26].

4.1.5 Custom RF cables

The GNSS-antenna has a female TNC-connectors [16, p.1] while the F9Ps breakout board has female SMA-connector [38]. Two custom male TNC to male SMA cable RF cables were created to connect the antennas to the F9Ps. Figure 26 shows the two types of connector terminations made. The SMA part had an available assembly guide [39] that was followed, while a video guide [40] was followed for the assembly of the TNC parts. Before attaching the SMA connectors, the cables were threaded through cable glands to be mounted on the enclosure. All parts have had a characteristic impedance of 50Ω [41] [42] [43] matching that of the antenna [16].

(a) TNC-connector.



(b) SMA-connector.

Figure 26: Terminations of custom RF cables

.

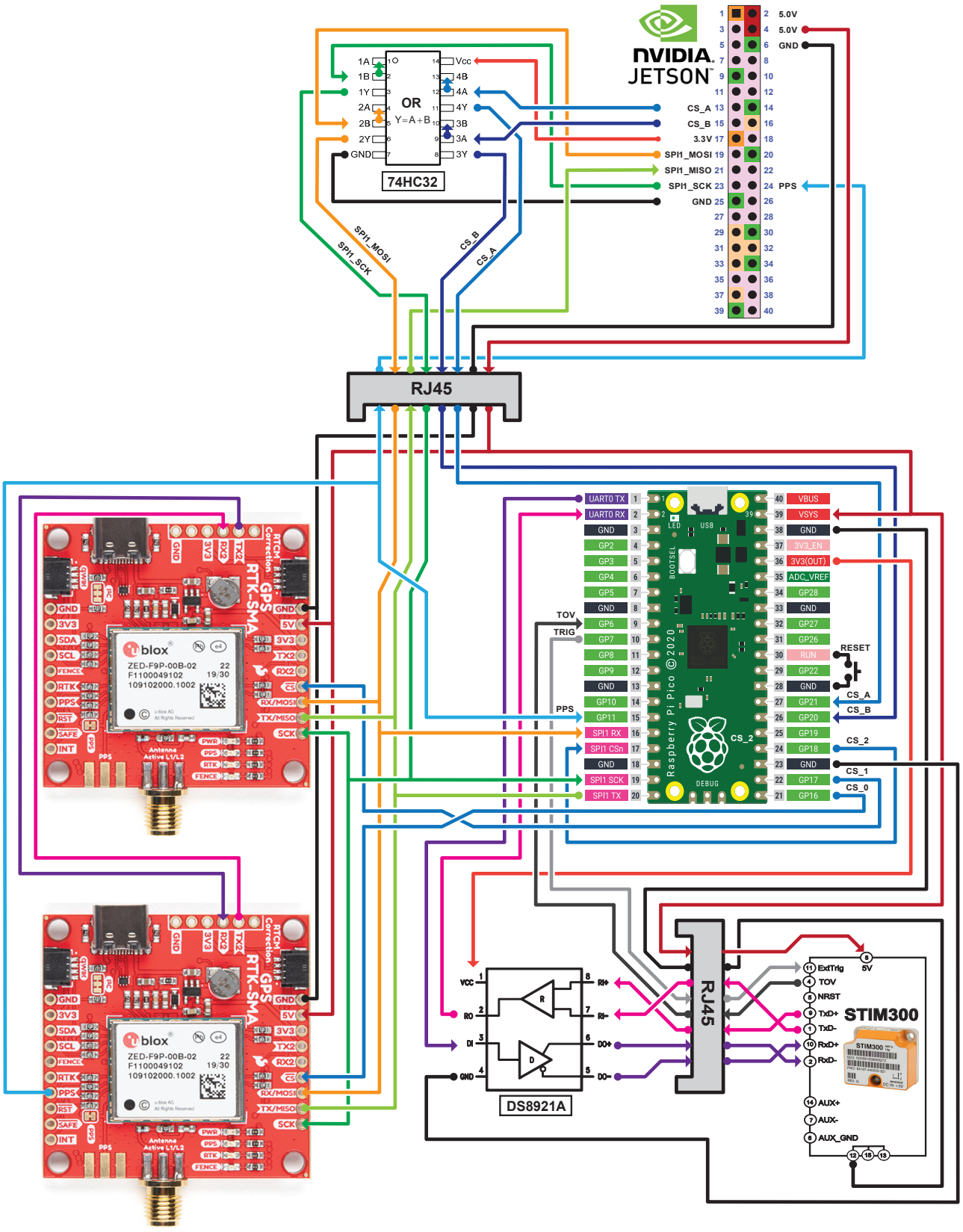


Figure 27: NavBox wiring diagram [35, p.1] [37, p.32] [38] [44, p.4] [34, p.3] [31, p.2].

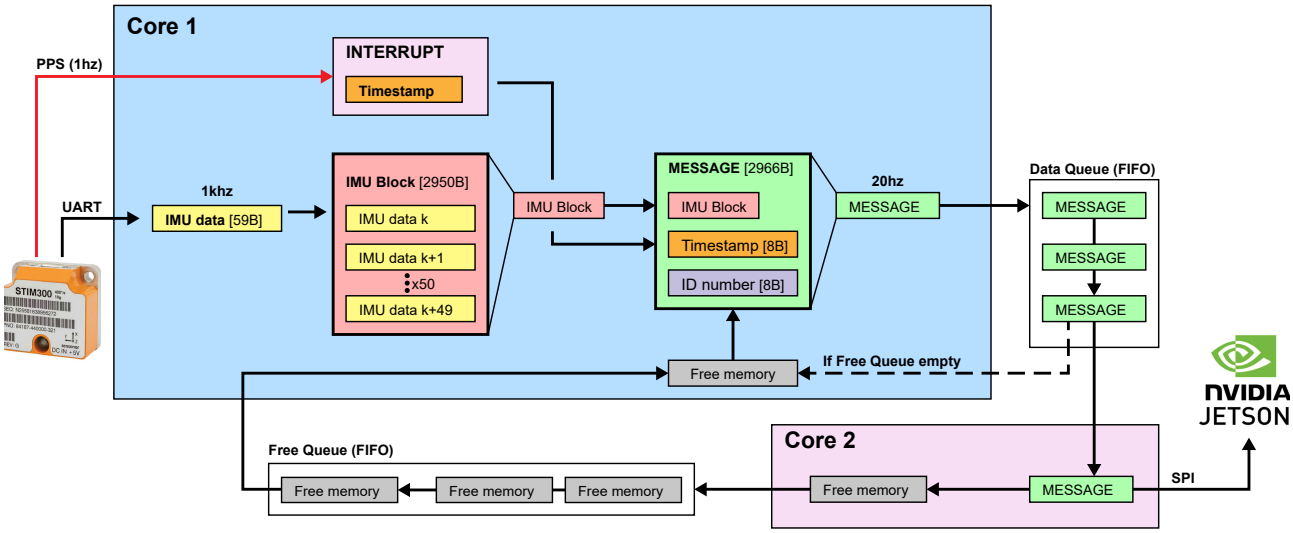


Figure 28: Data flow in Pico software.

4.2 Communication with the Xavier

The NavBox communicates with the Xavier over SPI. This enables the Xavier to read data from the two F9Ps and the and STIM300, via the Pico, over the same communication line. Following is a description of SPI and a presentation on how the Pico was programmed.

4.2.1 SPI

SPI allows high speed full-duplex communication between a controller and multiple peripherals [3] [46]. SPI was chosen rather than I^2C, as the F9Ponly support up to $400kb/s$ over I2C [47, p.48], wich is not enough. Up to $5.5Mb/s$ transfer rate is supported over SPI [27, p.16].

Mike Grusin has written a blog post on how SPI works, which is recommended alongside a video from Ben Eater [48] if the reader is unfamiliar with the protocol [46].

4.2.2 Pico data flow

The Pico is programmed in C. It receives measurements from the STIM300 over Universal Asynchronous Receiver-Transmitter (UART) and forwards them to the Xavier over SPI. Figure 28 shows how the measurement data from the STIM300 flow through the Pico to the Xavier. The main core receives IMU measurements from the STIM300 over UART, each consisting of 59 bytes of data. Blocks of 50 IMU measurements are packaged together as one block of data to be transferred in one continuous reading over SPI. The timestamp of the latest Pule Per Second (PPS) edge and the ID number of the first IMU measurement are sent alongside the IMU data to enable the calculation of time-stamps for all the measurements. The final message is added to the threadsafe First In First Out (FIFO) Data Queue to be read from the second core.

The second core reads a message from the Data Queue and makes them available for the Xavier over SPI. After the message has been fully transferred, its memory space is added to the Free Queue for reuse. Then the process is repeated.

Everything is done synchronously, which is why both cores are used. Asynchronous reading and writing over both UART and SPI is possible using Direct Memory Access Units (DMA) [44, p.92-102], but using two cores is a simpler solution and the power consumption is not a problem as discussed in Section 7.4.

There are 25 message slots allocated in memory, so the Pico can read data roughly once a second without missing any measurements. If the Free Queue is empty, the oldest message is discarded, and its memory space is used for the latest measurement.

4.2.3 SPI on the Pico

When using SPI on the Pico, by default, it requires the Chip Select (CS) signal to be pulled high between every byte transfer [49, p.534]. This is contrary to the F9P that expect continuous reading [47, p.52], and does not work with the CS setup implemented on the Xavier described in Section 5.2. Fortunately, the Pico supports continuous transfer when configured with Clock Phase (CPHA) set to 1 [49, p.535]. Thus all components communicating over SPI are configured with CPHA=1.

Another problem encountered is that the Pico never appears to let go of the CIPO line but pulls it low when idle. This can probably be solved elegantly, as there are a lot of low-level options related to the SPI [49, p.527-547], but with limited time and experience, such a solution was not found. Instead, it is solved by manually disabling the

[3] Controller-peripheral is the new term for master-slave [45].

CIPO pin when the Picos CS signal is high, i.e., the Pico is not selected, and reenabling it when the CS is low. This is done by the second core as part of the transfer process.

4.2.4 Line decoder using PIO

As ethernet connectors only support eight wires, using separate wires for all three CS signals is impossible with the current configuration. Instead, two wires and a line decoder are used to set the CS values on all three chips. A line decoder is a device that takes a set of signals representing a binary value and applies voltage to the wire with the corresponding number. Figure 29 shows a circuit diagram of a 2-to-4 line decoder.

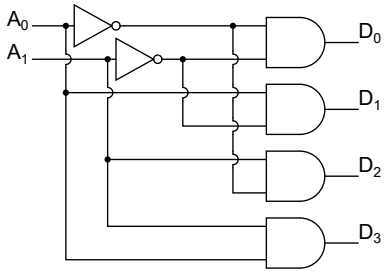


Figure 29: 2-to-4 line decoder circuit diagram [50, p.283]

A dedicated line decoder circuit was initially planned to be used, but another solution using the Pico was found. Programmable I/O (PIO) is a new type of hardware that has been developed for the Pico, that is intended to be used in implementing different types of Input and Output (I/O) [44, p.29]. PIO is a collection of state machines programmed using an assembly-like programming language with only nine different instructions [44, p.59]. One benefit of using PIO is that every instruction runs on exactly one clock cycle [4], enabling deterministic and high-speed glsio. Another benefit is that these state machines run separately from the cores on the Pico, enabling parallel execution. A small program, shown in Listing 1, was written to let six of the pins on the Pico behave like chip select controller. Running with a clock rate of $133Mhz$ and taking at most 13 cycles to loop, the maximal delay with this implementation is less than $10ns$

[4]The WAIT instruction takes more than one cycle [44, p.60].

```
1  .program selecter
2    set y,      0b1111 ; y  = ... 00001111
3    in  y,      4      ; isr= ... 00001111
4    set y,      0b0111 ; y  = ... 00000111
5    in  y,      4      ; isr= ... 11110111
6    in  null,   (32-8) ; isr=11110111 ...
7    mov y,      isr    ; y  =11110111 ...
8  .wrap_target
9    in  null,   32
10   in  pins,   2
11   mov x,      isr    ; x  = ... 000000AB
12   mov osr,    y      ; osr =11110111 ...
13 repeat:              ; lshift osr x+1
14   out null,   1
15   jmp x--     repeat ; jmp to repeat
16   out pins,   4      ; set CS pins
17 .wrap    ; jump to .wrap_target
```

Listing 1: Line decoder implementation in PIO assembly.

4.3 IMU-GNSS synchronization

To achieve accurate synchronization of the measurements, all the different time-deltas [5] must be accurately estimated. Figure 31 shows an overview of the different time-deltas.

A challenging factor is the presence of three different time frames. That is when events occur relative to Coordinated Universal Time (UTC) which is the true time of when something happens, and when they occur relative to the clock on the Pico and relative to the clock on the Xavier.

4.3.1 PIO trigger

The signal from the Pico used to trigger the sampling from the IMU is handled by one of the PIO state machines on the Pico, described in 4.2.4 [49, p.334]. Using the *WAIT* instruction [44, p.60], the main thread logs a time-stamp when triggering starts. This time-stamp is used to align the pico clock and the trigger pulses as shown in Figure 31. Thus the first trigger happens at $t_{pico} = 0$,the second at $t_{pico} = 0.001$ and so on. A counter assigns a number to each IMU message received in ascending order, i.e., message 0 corresponds to the first trigger, message 1 to the second, and so on.

4.3.2 IMU triggering

The STIM300 does not make a measurement exactly when a trigger signal is received [30, p.41]. In the STIM300, the sampling happens at regular intervals of $0.5\mu s$ [30, p.41]. When a triggering signal is, the most recent available measurement is sent as shown in Figure

[5]time-delta = difference in time between two events

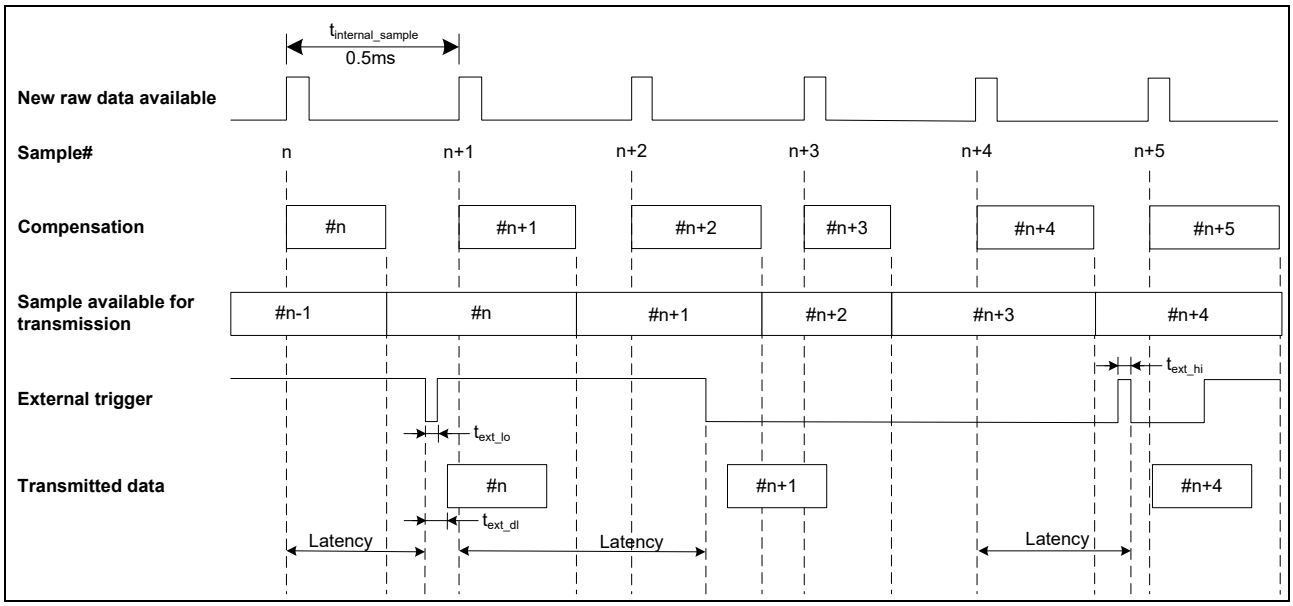


Figure 30: Timing of external trigger [30, p.34].

30. The time-delta between the measurement and the received trigger signal, called *Latency* in Figure 30 and Δm in Figure 31, is sent from the STIM300 together with the measurement data [30, p.41].

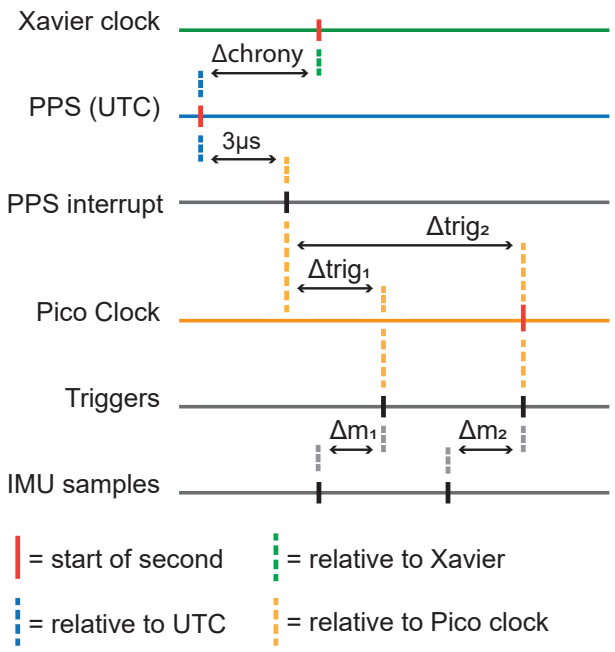


Figure 31: Navbox timing.

4.3.3 PPS interrupt

To synchronize the IMU messages to UTC, the PPS signal from one of the F9Ps is used. When a falling edge is detected on the PPS signal, an interrupt logs the time-stamp of the detected edge. As the timing of each trigger relative to the Pico clock is known, as discussed in Section 4.3.1, this time-stamp makes it possible to know the decimal value of the time-stamp of each trigger in UTC, corresponding to $\Delta trig + 3\mu s$ in Figure 31.

It was assumed that the interrupt code would run after only a couple clock cycles as described in the datasheet [49, p.74]. However, the testing described in Section 7.3 found the delay to be $\sim 3\mu s$. After consulting the forums, it appeared like some latency was normal [51]. Although it allegedly is possible to achieve the claimed latency by writing low-level code [51], the current implementation was deemed good enough as the delay was constant and could be compensated for.

5 Reading data from the NavBox

The reading and processing of measurements, from the NavBox, on the Xavier is done in Python. In this section, the software and its development process are discussed. Figure 32 shows a flow chart of the final program.

5.1 Synchronous reading

Typically real-time applications working with I/O require some sort of asynchronous programming. Personal experience with AsyncIO made it the go-to option, but both the SPI library [52] and General Purpose I/O (GPIO) library [53] used did not support AsyncIO natively. The Threading library was also considered but also dopped, as the assumption that asynchronous programming was necessary was false.

A benefit of using SPI is having control of when data is received from the peripherals [46], removing the need for polling. With a transfer rate of $5Mb/s$, the time spent reading data is also small enough to be done synchronously. Thus the whole software could be synchronous, as shown in Figure 32, making it notably easier to develop and debug.

5.2 Chip select

As discussed in Section 4.2.4 the NavBox uses a line decoder to convert a signal from two wires to three intependant CS signals. The two wires are labeled CS_A and CS_B. Together they represent a binary number from 0 to 3 that can be expressed as:

$$CS = 2 \times CS\_A + CS\_B$$

CS_A and CS_B are 1 if they are pulled high and 0 otherwise. This CS number corresponds to which of the three CS wires is selected, i.e. pulled low:

$$\begin{aligned} CS = 0 &\iff CS_0 = 0, CS_1 = 1, CS_2 = 1 \\ CS = 1 &\iff CS_0 = 1, CS_1 = 0, CS_2 = 1 \\ CS = 2 &\iff CS_0 = 1, CS_1 = 1, CS_2 = 0 \\ CS = 3 &\iff CS_0 = 1, CS_1 = 1, CS_2 = 1 \end{aligned}$$

A challenging factor is that the Xavier cannot set CS_A and CS_B simultaneously using python. One is always set a couple microseconds after the first one. This is a challenge, as the Pico and the F9Ps both appear to lose a byte of data when CS is pulled low, then high, without any data being read. As it is impossible to go directly from $CS = 0$ to $CS = 3$, returning to $CS = 0$ between every read is not possible either. A reading sequence taking this into account had to be developed. Figur 32 shows this sequence, where CS_A and CS_B never are set simultaneously.

5.3 Reading from STIM300

The Pico was programmed to be as easy as possible to read from the STIM300. All messages are the same size, $2966bytes$, and have the same format. After reading the correct amount of bytes, the reader checks if the received data is empty, e.g., only zeros. That indicates that there is no available data from the STIM300 and the reading is complete.

If there is data, a CRC is performed to verify that the message is not corrupted. This is explained in detail in Section 5.5. If the data is valid, it is parsed to access the 50 measurements and give them the correct time-stamps. The parsing had to be implemented based on the sensors datasheet [30, p.34, 45–50].

After reading CS is pulled high, then low, for the Pico to make the next message available. The process is repeated until no data is received.

If the CRC fails, a warning is thrown. Then the Pico is flushed by pulling its CS signal low and 4096 bytes. This makes sure there are no partially read messages left on the Pico, which might happen if it is restarted during operation.

5.4 Reading from F9Ps

Reading from the F9Ps is a bit more challenging, as both send multiple different types of messages of different lengths. To make it as easy as possible, they have been configured to only use the UBX protocol [54, p.29], which uses data frames with the structure shown in Figure 33. UBX is chosen, rather than NMEA as it is more compact, using binary data rather than the string representation used by NMEA [54, p.5].

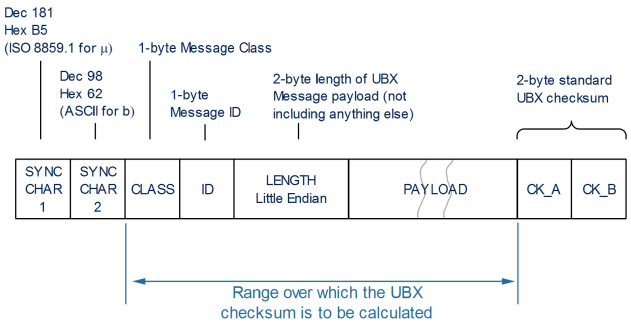


Figure 33: UBX frame structure [54, p.29].

The reading process starts by reading six bytes from the F9P. If no data is received, it indicates that no data is available and the reading is complete. Else, the data is added to a buffer, and RegEx is used to locate a group of six bytes matching the first 6 bytes of a UBX message, shown in Figure 33. If no match is found, this process repeats. A buffer is used as the message might not be aligned if the program is killed during execution and restarted. After the first message, the first six bytes should always be the first six bytes of a message.

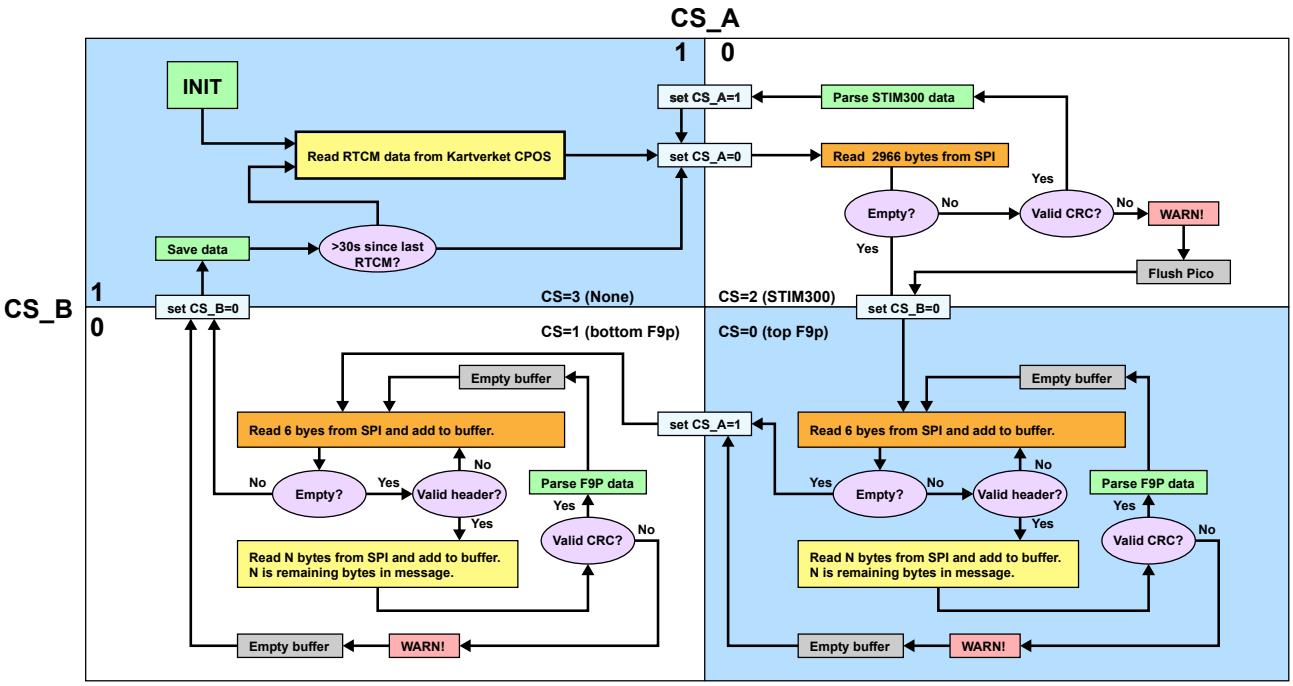


Figure 32: Flowchart of software on Xavier to read frm NavBox.

After finding a match, the number of remaining bytes is calculated from the two bytes shown in Figure 33 and the position of the match in the buffer. Then the remaining bytes are read, and a CRC is performed. If the CRC is valid, the data is parsed, the buffer is emptied, and the process repeats.

A failed CRC raises a warning, and the program continues.

5.5 CRC

Cyclic Redundancy Check (CRC) is a method used to verify data integrity. The sender calculates a checksum based on the message it's sending and adds that checksum to the sent data, as shown for the F9P, in Figure 33, where a 2-byte checksum is added. When the data is received, the receiver does the same calculation and checks that the same result is found. If the data is corrupted, the result will not match the checksum [6].

The calculation used in CRC is based on calculating the remainder of a polynomial division, where the coefficients of the polynomials belong to $\mathbb{F}_2$, the finite field of two elements. This operation can be formulated as follows:

$$p_{data} = \sum_{i=0}^{n-1} a_i x^i, \quad a_i \in \mathbb{F}_2$$

$$p_{div} = \sum_{i=0}^{m} d_i x^i, \quad d_i \in \mathbb{F}_2, m < n$$

$$p_{data} = p_{quotient} \times p_{div} + remain$$

$$\mathbf{crc}(p_n) = remain$$

Using coefficients belonging to $\mathbb{F}_2$, the finite field of two elements, makes the calculation efficient on computers. It makes the polynomials map to binary numbers and both the plus and minus operation becomes equivalent to the bitwise OR operation, which removes the need for carry-bits during division. Ben Eater has an excellent video explaining how this works [55].

Most implementations of CRC also use something called a seed, resulting in the following formulation:

$$seed = \sum_{i=0}^{m-1} s_i x^i, \quad s_i \in \mathbb{F}_2$$

$$\mathbf{crc}_{seed}(p_{data}) = \mathbf{crc}(seed \times x^n + p_{data})$$

There are many different variations of CRC, using different divisors, seeds and mappings from data to polynomials [56].

5.5.1 Implementation

The Python library used to parse the messages from the F9Ps has a built-in 16-bit CRC implementation [57, *ubx-helpers.py*: Ln.21-40] that is used on those messages.

[6] Corrupted data might have valid checksum, but this is very unlikely.

Verifying the messages from the STIM300 was more challenging. The data sheet states that is uses a 32-bit checksum with the following divisor and seed [30, p.35]:

$$p_{div} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

$$seed = \sum_{i=0}^{31} x^i$$

They can be represented by the following hexadecimal numbers respectively `0×104c11db7`, `0×ffffffff`. As there are several different versions of 32-bit CRCs [56], the webpage *crccalc.com* [58] was used to identify which one was used by the STIM300 as it is not specified in the datasheed [30, p.35]. By matching the results from the webpage, after giving it a message from the STIM300 as input, with the CRC-checksum of the message, it was found that the STIM300 uses the CRC-32/MPEG-2 variant. From the datasheet it is specified that the payload should be padded with zeros before the CRC is performed, making the number of bytes divisible by four [30, p.35].

A function to calculate this CRC was implemented in Python based on a forum post from Mark Adler [59]. The implementation is shown in Listin 2

```
1  def crc32mpeg2(message_padded):
2      poly = 0×104c11db7
3      crc = 0×ffffffff #seed
4      for value in message_padded:
5          crc ^= value << 24
6          for _ in range(8):
7              if crc & 0×80000000:
8                  crc = (crc << 1) ^ poly
9              else:
10                 crc <<= 1
11     return crc
```

Listing 2: CRC32-MPEG2 algorithm implemented in Python.

5.5.2 Speedup using lookup table

As Python is an interpreted language, rather than a compiled one, it is relatively slow. This is especially significant for the type of nested loops present in Listing 2. Performing this CRC on a thousand IMU messages every second using the above implementation was not possible on the Xavier as it resulted in dropped messages. This was only the case when running in low power mode and could be solved by increasing the clock speed on the Xavier, but that would increase its power consumption more than necessary. Thus effort was put into speeding up the CRC implementation.

From Listing 2 one can see the result after running the inner for-loop is determined by the eight first bits of value on line five. This can be leveraged by constructing a lookup table and referencing that lookup table for each new byte of data, rather than doing the bitshift one bit at a time. My implementation of this is shown in Listin 3. This implementation was almost three times as fast as the previous one.

```
1  def get_crc_table():
2      table = np.empty(256, np.uint32)
3      poly = 0×104c11db7
4      for i in range(256):
5          bflip = i << 24
6          for _ in range(8):
7              if bflip & 0×80000000:
8                  bflip = (bflip<<1)^poly
9              else:
10                 bflip <<= 1
11         table[i] = bflip
12     return table
13
14 TABLE_ARR = get_crc_table()
15
16 def crc32mpeg2_lookup(message_padded):
17     crc = 0×ffffffff
18     for val in message_padded:
19         crc = ((0×ffffff & crc) << 8
20                ^TABLE_ARR[val^(crc>>24)])
21     return crc
```

Listing 3: CRC32-MPEG2 algorithm using lookup table implemented in Python.

5.5.3 Acceleration with Numba

An effort was put into implementing an even faster version of the CRC. This turned into a project in the course *DT8117 Heterogenous & Cloud Computing*, where the algorithm was used as a benchmarking tool for different ways to speed up Python. A library called Numba, used to compile Python code, was tested as well as a library called Pybind11 that makes it possible to import functions written in C++ and CUDA.

Figure 34 shows some results from this project. It shows that basic Python (py) is roughly five hundred times slower than C++ (cpp), but that similar speeds can be achieved using Numba (jit). As the messages from the STIM300 are only 59 bytes long, parallelization is not beneficial. Based on results from that project, the implementation using a lookup table and Numba (py_lookup_jit) was chosen.

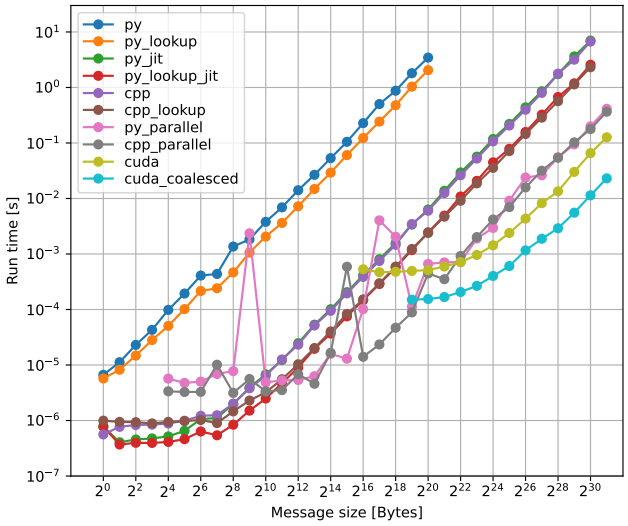


Figure 34: Runtime of different CRC-32 implementations as a function of message size on a *Dell XPS 15 9510*.

5.6 RTCM

As discussed in Section 3.1.2, relative positioning has major benefits. In this project, a moving base setup has been implemented similar to the one shown in Figure 35, based on the *Moving base application note* from *U-blox* [26].

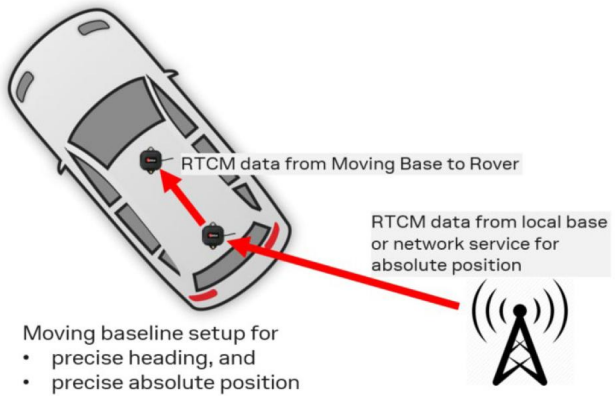


Figure 35: Illustrative figure of moving base setup [26, p.5]

As shown in Figure 35, two relative positions must be calculated; one from a local base or network service for absolute position and one from the moving base to the rover for heading.

5.6.1 Moving base to rover

The correction messages from moving base to the rover are transferred directly from one F9P to the other. The F9P has a separate UART tranceiver for built for exactly this purpose. In the bottom left side of Figure 27 the connection between the F9Ps is shown.

5.6.2 Kartverket CPOS

Instead of using a standard base station, corrections from Kartverket are used. They offer a positioning service called CPOS, enabling positioning with centimeter accuracy [60]. The CPOS system calculates a virtual reference station based on data from their permanent stations that supply correction messages to the user [60]. A significant advantage is that it removes the extra work of setting up a base station for every survey, which streamlines the process. Usually, the service comes as a paid subscription, but they offer it for free for educational purposes [60].

The downside, however, is that receiving the data is not straightforward. To receive data, a connection has to be established with Kartverket's servers over Networked Transport of RTCM via IP (NTRIP), which is a closed source protocol [61]. Unlike HTTP, which returns valuable headers that lets you debug your client, NTRIP does not return anything unless everything is perfect.

Four different sources were used to implement a client;

1. A document describing the old version, Rev1, of the protocol [7] [62].
2. An open-source implementation written in C [63].
3. A Python library [64].
4. A blog post on the difference between NTRIP Rev1 and Rev2 formats [65].

The open-source implementation was used to try to connect, but no data was received. After failing to get the python library to work, I contacted Kartverket by phone and managed to get hold of one of a developer. He confirmed that a connection had been made, with a timestamp corresponding to when the open-source implementation was tested. Unfortunately, he could not say why no data was received.

As the open-source implementation appeared to work somewhat, it was connected to a dummy server so that the transmitted data could be inspected. After comparing the data with the python library and the data from the other sources, the final problem turned out to be an invalid checksum in the message sent to position the virtual station. After fixing this, a working client was implemented.

The correction messages received are sent to the F9P working as moving base over SPI simultaneously as data is read.

[7] The first sentence is: *The material provided here is not the official RTCM documentation!*

6 Software and electronics

6.1 Jetson Xavier

The NVIDIA Jetson Xavier AGX Developer kit (Xavier) is a power-efficient computer with an 8-core ARM 64-but CPU and 512-core Volta GPU built for robotics. [66, p.8-10]. There are three main reasons it was chosen as the brain of the sensor rig. The first one is its 40-pin GPIO expansion header, used to power, and communicate with,the NavBox [37, p.25] Secondly, it has a PCIe slot on the side that can be used to connect a high speed network card, as discussed later in Section 8.7 [67, p.iii]. And finally, it has the computational power needed to handle the incoming data from future sensors.

Acquiring its newly-released successor, the Jetson Orin was initially planned, but the acquisition was postponed after the result from the heat dissipation test discussed in Section 7.5 indicated that the Jetson Orin's higher power consumption could be problematic.

6.2 Docker

Docker is an open-source platform that enables developers containerize their applications [68]. This can be described as packing the software and everything needed to run it, including operating system libraries, into one standalone container that can be executed from almost any computer [68].

This has countless advantages, but the most significant one for this project is that it makes it effortless to run the same code on the Xavier and my laptop. Normally this would not be the case, as the Xavier running a custom version of Linux, built for its ARM-based CPU, while the laptop is running Windows 11 on its Intel CPU.

Another advantage is that Docker significantly reduces the cost of trial and error. When something breaks during software development the latest changes can simply be reverted using Git and the container can be rebuilt. This will restore everything, leaving no corrupted driver or broken symbolic link behind.

6.3 SSH

Although the Xavier offers the necessary utilities to function as a desktop computer, developing code directly on the Xavier is not ideal. It is, for instance, difficult to connect to multiple monitors. Secure Shell or Secure Socket Shell (SSH) is a network protocol that makes it possible to access and control one computer from another computer through a network [69] and makes it possible to run code on the Xavier from a normal laptop. *Visual Studio Code* has a handy extension, *Remote -SSH* that makes it easy to view, edit and run code on the Xavier using SSH.

6.4 PPS on Jetson Xavier

To synchronize the clock on the Xavier with UTC, the PPS signal from one of the F9Ps is used. The PPS signal is an analog signal that has a falling edge at the beginning of every second of UTC [8] This signal can be connected to one of the GPIO pins of the Xavier to synchronize its clock.

6.4.1 Compiling Linux to enable PPS

As the precision required is on the order of microseconds, regular software cannot handle this, as its execution is controlled by the operating system's scheduler. Thus, to get accurate readings of the PPS signal, it was necessary to edit and recompile the Linux kernel running on the Xavier. This proved to be no easy task, as there was no guide on how to do it [9] for the Xavier. Fortunately, there is a well-written blog post on how to do it for a *Jetson Nano*, which is a product in the same family as the Xavier [70].

To download the source code and flash the Xavier, a software called *Jetson SDK Manager* is needed [71]. Attempts were made to get the available docker image [71] to run in *Docker Desktop* on Windows without success. It was possible to detect and connect to the Xavier over USB, Using the *usbipd-win* software [72], but network errors appeared to prohibit flashing with new firmware. Ubuntu was installed on my old laptop to solve the issue.

Following Sadowski's guide [70], the files were downloaded and compiled, and PPS was enabled in the *.config* file. Then an answer on the Nvidia forum [73] explained how to correctly modify the device tree on the kernel to set which pin on the Xavier should listen for the PPS signal. To find the correct correspondence between GPIO-number and pin, the *Pinmux Configuration Template* [74, p.2, row 136] was consulted as well as the *tegra194-gpio.h* file [75] and another forum post [76]. The device tree was configured for the Xavier to detect a PPS signal on a falling edge. The kernel could finally be recompiled, and the Xavier was flashed. Using *pps-tools*, it was possible to confirm that the PPS signal was correctly detected in the Xavier [70].

6.4.2 Using Chrony to synchronize clock

After enabling PPS, the Xavier could detect exactly when a PPS signal was detected, but a method was needed to synchronize its clock. Initially, the Network Time Protocol daemon (NTP) [77] was used in an attempt at synchronizing the clock on the Xavier to UTC using the PPS signal, like describet in Tomologan501's post [73]. Unfortunately, it appears this method required NMEA messages over UART to function properly [10].

[8] It can also be rising [47, p.67]

[9] No guide was found after hours searching

[10] The PPS signal alone cannot tell which second it belongs to. For it to work, another source, like stamped NMEA messages fromGNSS, is needed to synchronize the clock to the correct sec-

Instead of using NTP, a software called *Chrony* [78] was used. This software appears to be better suited for the application as it works better with systems that are not always turned on [79]. The installation process is simple [80], and it was possible to work using only the real-time clock on the Xavier, and PPS signal [81]. Adding the following line to *chrony.conf* made *Chrony* update the real time clock when it had interned access.

```
rtcsync
```

After installing, it was possible to confirm that the PPS signal was used using the *chrony sources* command [82].

6.4.3 GPIO and SPI on the Xavier

Enabling SPI on the Xavier was more straightforward than initially anticipated [83]. SPI is accessed in python using the *spidev* package [84]. To avoid having to manually enable SPI every time the Xavier was rebooted [85], the following line was added to *etc/modules* [86].

```
spidev
```

Enabling GPIO was also an easy task, as it could be done by simply installing the *Jetson.GPIO* package using *pip* [87].

6.4.4 JTOP

When working with the *Jetson* products from *Nvidia*, JTOP [88] is a really usefull tool. It includes a performance monitor where memory, CPU, and GPU utilization are displayed alongside other metrics like the energy consumption and temperature of the different components of the computer. It also makes it easy to adjust the power mode of the Xavier.

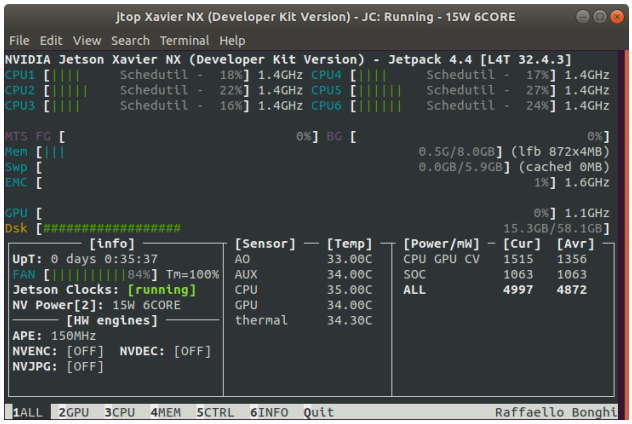


Figure 36: A window in *jetson-stats* [88].

6.5 Network

As discussed in Section 6.3, working on the Xavier over a network is often a better option than using it as a desktop computer. During normal development, the Xavier is connected via ethernet to the same local network as the laptop. Its IP is found by briefly connecting it to a monitor and operating it in desktop mode.

6.5.1 Connectin to Wi-Fi

However, as the sensor rig is intended for wireless use outdoor, another connection method is needed. The initial solution was to connect the Xavier to a phone over USB and share the network from the Phone. This worked but was not considered a clean solution.

The Xavier does not have built-in Wi-Fi, so a *Intel AC8265 Bluetooth, WiFi Wireless Adapter* is used to enable wireless communication. The adapter can be seen in Figure 17, as a white square on the lower left side of the Xavier. The datasheet does not specify the adapter antenna connector type [89], but it looked like U.FL connectors. After several failed attempts at connecting the antennas, it was clear that it was not U.FL connectors but MHF4 connectors, which are almost identical. After a new antenna was received and connected, it was possible to follow a forum post [90] and connect the Xavier to Wi-Fi. The antenna was mounted on the side of the Xavier, as shown in Figure 37.

Changes were made to */etc/network/interfaces* for ethernet to be preferred when connected to multiple networks with Wi-Fi and ethernet [91].

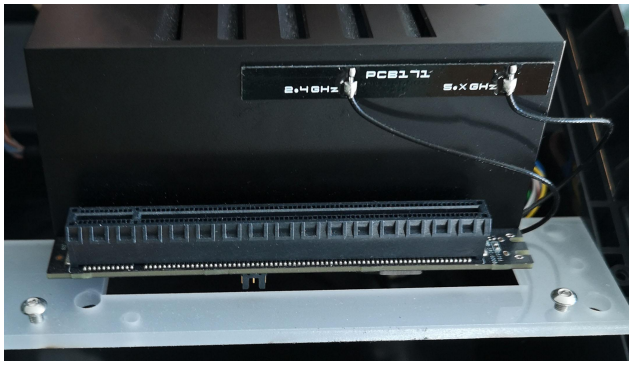


Figure 37: Wi-Fi antenna mounted on Xavier.

6.5.2 Juice SSH

Bringing a laptop into the field to control the sensor rig during missions might be challenging. The proposed solution is to operate the sensor rig from the phone. Using the *hotsopot* functionality available on most smartphones [92] the Xavier can connect to the phone over Wi-Fi. This gives it access to the internet everywhere, as long as the phone is connected to 4G or other similar networks. Being connected to the same network, it is possible to control the sensor rig from the phone using an app called *JuiceSSH* [93]. Another benefit of using this setup is that it makes it easy to find the IP of the sensor rig whenever it is connected to a network over ethernet, as you can access the shell from the phone over Wi-Fi and run *ifconfig*.

ond.

6.6 Expanded storage

It is necessary to expand the storage of the Xavier for it to be used in collecting datasets, as it only has 32GB built-in storage [66, p.2]. Fortunately the Xavier has a *M.2 Key M Connector* [37] where a SSD hard drive can be connected. To get permanent access to the extended storage space, modifications were made to the */etc/fstab* file, as described in a blogpost on *CodinGame* [94].

6.7 Battery

A four-cell $5000mAh$ Lithium Polymer (Li-Po) battery is used to power the sensor rig. This battery should roughly hold $3.7 \times 4 \times 5 = 7.4Wh$, which should be enough for several hours of operation. A larger battery could be used if a more extended recording is desired. The advantage of using Li-Po batteries is their reduced weight and high capacity [95].

The downside is that they are more challenging to charge, requiring a dedicated charger [95]. Such a charger has been acquired and is powered by a retrofitted power supply, as shown in Figure 38. They also pose a fire risk if not handled properly and should be stored in a fireproof safety bag. This is why the battery mount is designed to be easily detachable, as described in Section 2.4.4

A *HCW-M635* [96] battery protection circuit, shown in Figure 39, is used to ensure the the voltage never drops below $3.0V$ per cell, as that causes permanent degradation of the battery. It is also useful as it visually indicates the voltage level of the battery through the lid of the enclosure.

Using a five-cell Li-Po with a nominal voltage of $18.5V$ might seem better, as the Xavier take up to $20V$ of supply voltage [37, p.3], when fully charged each cell holds $4.2V$ [95], which would cause damage.

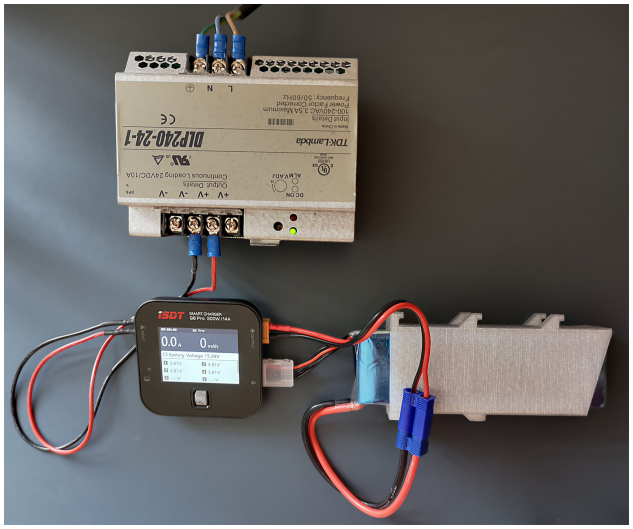


Figure 38: Li-Po battery charging.

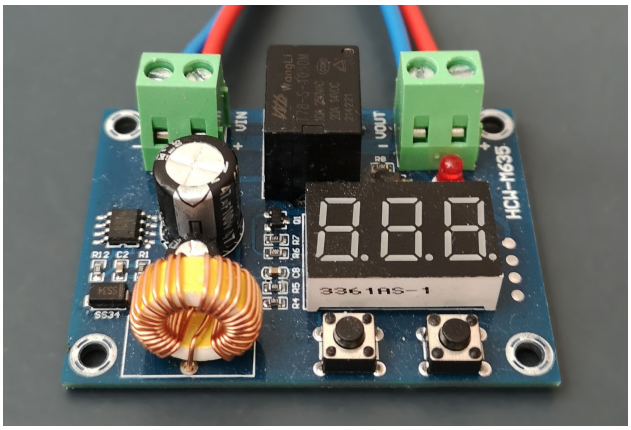


Figure 39: Battery protection circuit.

7 Testing

7.1 Mechanical strength

In the absence of a more scientific test to examine the mechanical strength of the sensor rig, a simple experiment was conducted; Holding the sensor rig with two hands and shaking it forcefully. It appears to not have suffered any damage from this, and it is concluded that it is pretty solid. The test was conducted over a soft mattress so that parts would not be damaged if they broke and fell.

7.2 NavBox data output

The sensor rig can simultaneously collect, verify and parse messages from the STIM300 at $1000Hz$ and the F9Ps at $8Hz$.

It receives raw data from both F9Ps, RTCM messages, position data from the top, and relative positions from the bottom. Disconnecting the antenna from the top F9P results in no position data from it and no relative positions from the bottom F9P, as it stops receiving correction data. The position messages from the top F9P indicate the correction messages from Kartverket are used when available. This is also shown by a lower noise estimate when receiving corrections.

Changing one bit of the received data is always caught by the corresponding CRCs.

7.3 IMU-UTC synchronization

A simple test was conducted to test if the software was written to synchronize the STIM300 to UTC works as intended. The PPS signal was disconnected from the Pico, and the trigger signal used to trigger the STIM300 was connected in its place. With this configuration, every message should be $0\mu s$ behind the previous PPS signal as there is no delay between the trigger signal and itself. However, every message was $3\mu s$ delayed. This is how it was found that the interrupt on the Pico has a $3\mu s$ delay When this compensation was added, the $0\mu s$ delay was achieved. The synchronization is therefore accurate down to $0.5\mu s$.

If better synchronization is wanted, the Time Of Validity (TOV) signal from the STIM300 could be used together with PIO, but this was considered necessary.

7.4 Power consumption

The NavBox is powered from the $5V$ voltage source on the Xavier. As JTOP can display information about power usage on the Xavier, it was used to see how much power the NavBox requires. It was found to draw $2240mW$ on average, as shown in Figure 40. The Xavier can supply $1A$ on the $5V$ bus [37, p.26], i.e., $5W$, which leaves room for more devices to be connected to the bus in the future.

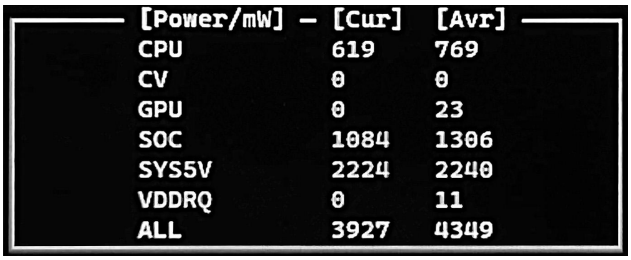


Figure 40: Output from JTOP during operation.

7.5 Enclosure heat dissipation

The lack of heat dissipation through the enclosure walls was a concern in the development process. To test this, a temperature test was conducted. The goal of the test was to find the terminal temperature in the box when $30W$ of heat was produced inside it, as that is the maximal power consumption of the Xavier.

Twelve 600Ω heat resistors were connected in parallel and placed inside the box. A glass fiber plate was placed between the enclosure bottom and the resistors to avoid damage from the heat. The Thevenin equivalent of the resistor would be a single 50Ω resistor; thus, when $39V$ is applied, $30W$ of power is produced. The temperature was measured using a thermistor and multimeter as shown in Figure 41. After roughly half an hour, the temperature settled at just below $70°C$. This is problematic as Li-Po batteries should not operate at this temperature. Thus, it is currently impossible to use the Xavier, with the lid closed, at max capacity. Especially since the other components also produce heat.

Fortunately, Figure 40 clearly shows that the Xavier also can operate at lower power. Data acquisition might be possible without altering the design, but this has to be verified in the future. Section 8.5 discusses how the cooling problem might be solved in the future.

Figure 41: Photo of setup under temperature test.

8 Partially finished and future work

There are several tasks remaining that need to be accomplished for the sensor rig to serve its purpose. This project is a part of my integrated Ph.D. made it possible to postpone this work. This has allowed me to spend more time learning new skills and making a higher quality product, which would have been possible if everything was to be completed within the time of this project thesis.

Even though there has not been time to finish the following tasks, varying amounts of time have been spent working on them.

8.1 GigE interface

A PCIe network card has been acquired to connect to future cameras and other sensors. Acquiring a suitable network card was difficult as several requirements had to be met;

1. Support four connections with $1Gb/s$ bandwidth each for multi-sensor setup.
2. Support Power over Ethernet (PoE) to power the sensors, removing the need for extra cables.
3. Support Jumbo frames to reduce CPU load.
4. Be Linux compatible.
5. Be a small form factor to fit inside the enclosure.

With the current chip shortage, no network card matching the specs appeared available until late 2022. But after extensive searching, a product on *Alibaba* that appears to satisfy the requirements was found. The card can be seen in Figure 2d as well as in Figure 43.

Using a multimeter, it has been verified that the card outputs the correct PoE voltage when connected to the Xavier. No other testing has been conducted, and it remains to see if it works as intended. The testing has been delayed as it was considered risky to connect it to the expensive polarization camera without testing it on a less expensive camera first. Figure 43 shows how the card is mounted on the Xavier.

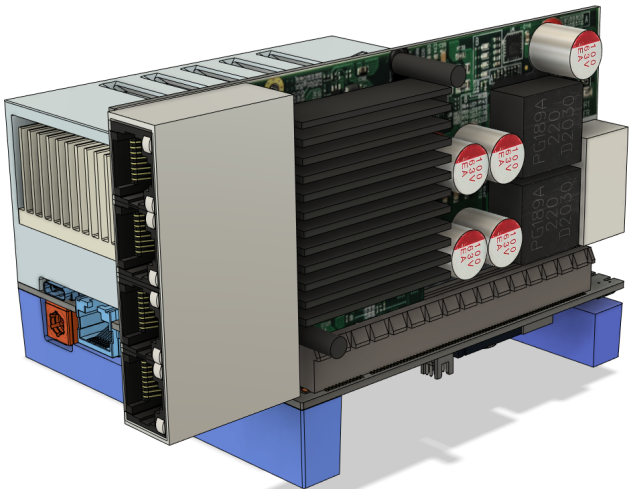


Figure 43: A model of the network card mounted in the PCIe slot on the Xavier.

8.2 Polarization camera

The first sensor to be tested on the sensor rig is the polarization camera from *Lucid Vision Labs* [97]. The polarization camera captures both polarization and color information using built-in polarization filters. Figure 44 shows how each pixel is built up of sixteen subpixels to achieve this. As reflected light is polarized, this type of camera can remove reflections [98, p.1], which is expected to be a significant benefit for maritime applications. Reflection removal can be seen in Figure 42 where the reflection from the sun on the top of the laptop is almost removed in the bottom left image.

Arena SDK, a software used to capture data from the camera, has successfully been installed and tested in a docker container on the Xavier. Currently, a Python class capable of reading images has been implemented, but Precision Time Protocol (PTP) things like synchronization over PTP and adaptive control of shutter speed and exposure gain remain to be implemented.

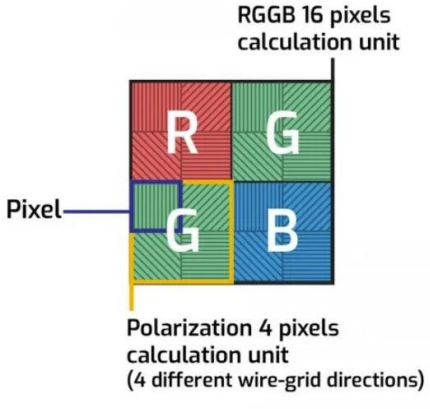
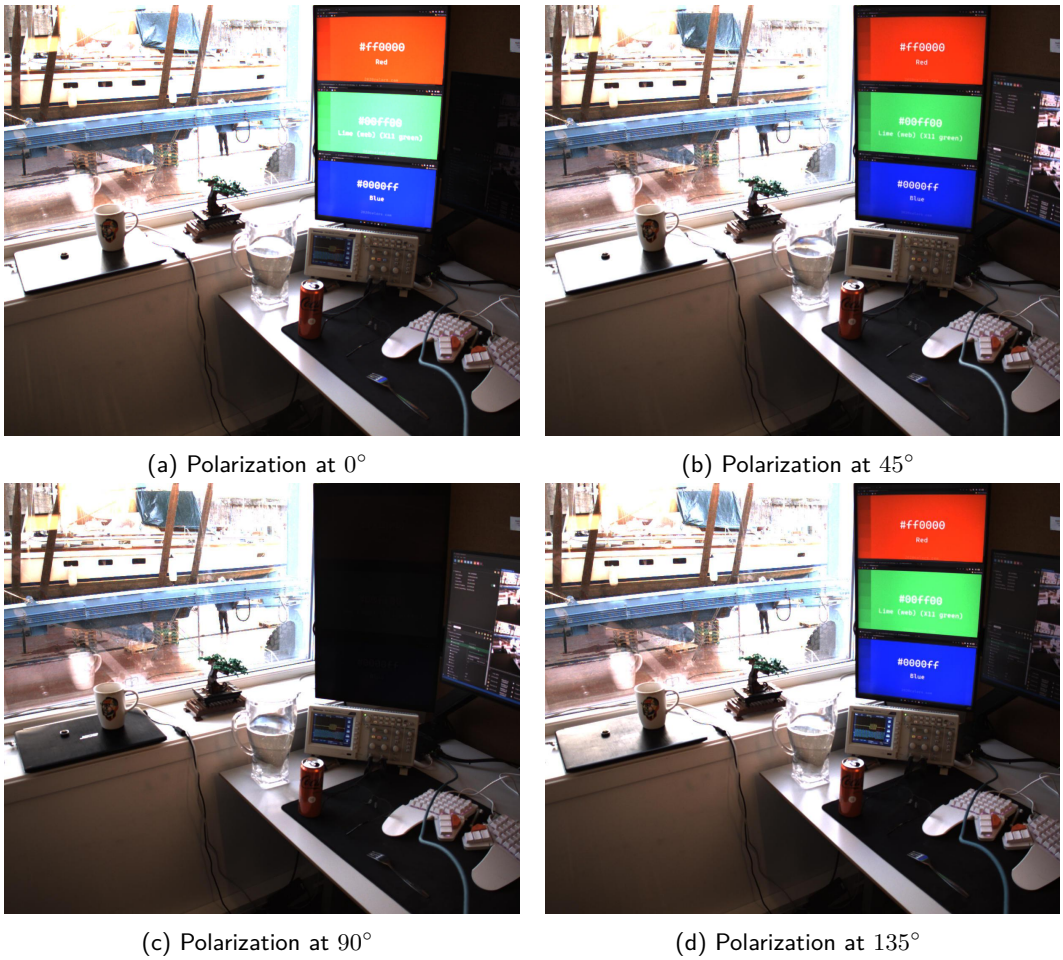


Figure 44: Visualization of pixel in polarization camera [98]



(a) Polarization at 0°

(b) Polarization at 45°

(c) Polarization at 90°

(d) Polarization at 135°

Figure 42: Photo taken with polarization camera, separated into its four polarization components.

8.3 Calibration

The calibration process is planned to be separated into two parts; First, all the sensors will be calibrated individually, then the extrinsic between them will be estimated. It is not yet decided precisely how the calibration should be done, in terms of what software to use and how much of it should be done using custom implementations, but the following is a default plan.

8.3.1 Individual calibration

First, the intrinsics of all the camera-lens setups have to be found. This is a relatively standard process that can be done using software like Kalibr [99].

The noise parameters of the STIM300 can be found by collecting data over a more extended period while the STIM300 is lying still. The Allan-variance plot for each of the nine internal sensors can then be made, and the random and bias walk parameters can be estimated. The axis alignment and scale factor can be considered negligible, as the STIM300 has built-in compensation [30, p.46]. Still, verifying this might be beneficial and done as a part of the visual-inertial calibration.

The precision of the GNSS likely depends on the surrounding environment, so a static noise parameter estimate should not be used. The F9P outputs estimated accuracy, and studying the relationship between this estimated and measured variance will be interesting.

8.3.2 Relative calibration

An advantage of having modeled the entire sensor rig in *Fusion 360* is that an excellent initial guess of the relative position of the sensors is available, which should make this calibration process easier.

After all the sensors have been calibrated individually, the relative position between the sensors can be calibrated. The plan is to start by finding the relative position between the cameras using an April grid and then doing visual-inertial calibration to get the STIM300's position and orientation close to the cameras. Both these calibrations can be done in Kalibr [99]. Finally, the position of the GNSS antenna relative to the cameras will be estimated. This is done by filming the April grid outside while changing the orientation of the sensor rig. Some sort of least square optimization will be used to find the best relative position of the antennas.

8.4 Pose estimation

Having good sensor data from the STIM300 and the F9Ps is a good start, but there is still much work remaining to get accurate pose estimation.

Using the measurements from the F9Ps directly is an option, but postprocessing the raw data is expected to yield better more accurate position estimates. This will probably be done using *RtkLib* [100] or *GnssTk* [101].

Similarly, fusing the GNSS data and IMU data will probably benefit from being done in post rather than on the fly. Earlier this semester, I participated in a project in the course *TK8102 - Nonlinear State Estimation* where we compared the performance of different pose estimations techniques on a simulated system with an IMU and one GNSS receiver. ESKF, two factor-graph-based filters and one factor-graph-based post-processer were compared. Unsurprisingly the post-processing technique performed the best, as indicated by the red line being the lowest in the top plot in Figure 45 from the project.

A factor graph-based implementation is a go-to option based on experience from that project, but this is to be decided in the future.

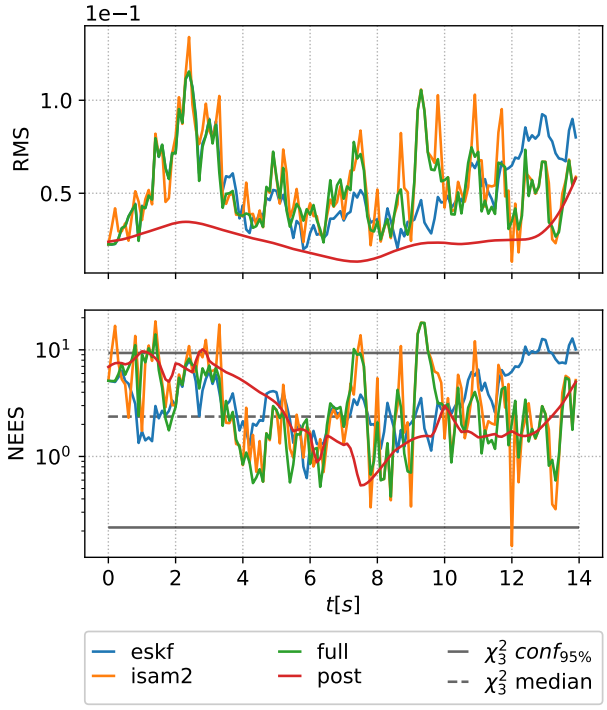


Figure 45: Position RMS and NEES from one experiment in a semester in *TK8102*.

8.5 Cooling

From the head dissipation test, discussed in Section 7.5, it appears like a better cooling solution is needed if the GPU on the Xavier is to be used at total capacity. The current idea is to use water cooling to cool the inside of the enclosure. From the design files of the Xavier and its thermal design guide, [103] it appears relatively straightforward to remove the current cooling fan and mount a water block on the thermal block, shown in blue in Figure 47. Using water cooling, the radiator can be placed outside the enclosure to release the heat outside. If a fan is used on the radiator, it might suffer from exposure to saltwater, but replacing it occasionally is not costly.

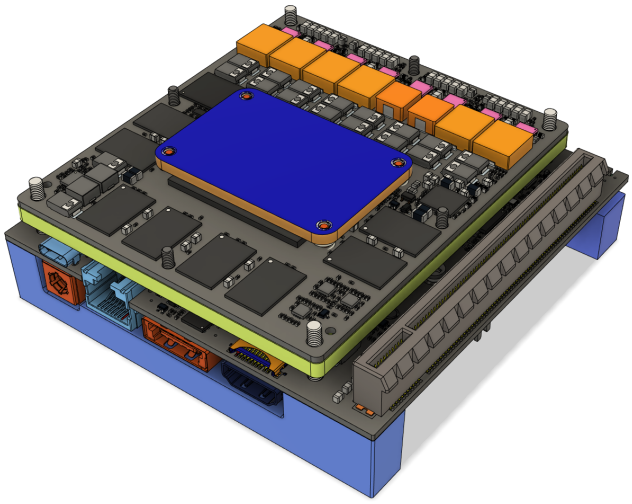


Figure 47: Mounting survace (blue) for custom cooling on Xavier.

8.6 Compression

Industrial cameras, like the polarization cameras from Lucid Vision Labs, can produce more than $1Gb/s$ of data [11] [97]. Raw data from these sensors should not be stored directly as the datasets would be huge. Thus some sort of compression is required. Fortunately, the Xavier has a multi-standard video encoder that can compress the video on the fly [66, p.16-17].

Leveraging this video encoder and comparing the supported encoding standards, HEVC, H.264, and VP9, has to be done in the future. Compressing the data from the polarisation camera is probably not trivial as standard encodings are not designed with polarization video in mind.

8.7 Graphical interface

Currently it is possible to control the sensor rig from the phone over SSH using *JuiceSSH* [93]. This can be used to start and stop recording, but there is currently no visual output.

When cameras are added, it would be beneficial to be able to see the video feed from the phone. From experience, the *web_vidwo_server* pachage for ROS [104] is a simple way to acheive this. However, it might be beneficial not to rely on ROS and use a framework that can do more than display a video feed. *Plotly's Dash* appears to

[11] $12bit \times 5MP \times 22FPS > 1Gb/s$

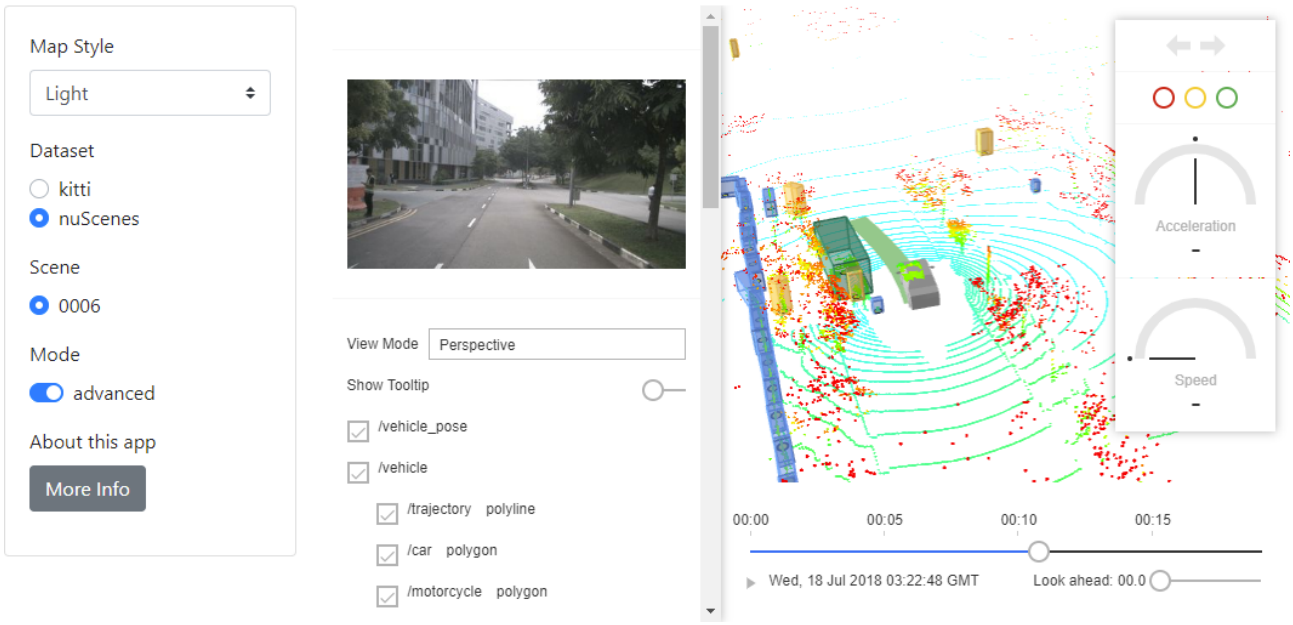


Figure 46: Screenshot of the graphical inerface of the Dash Autonomous Driving Demo [102].

be the best option, as it lets you design interactive web apps for data visualization in Python, like their *Dash Autonomous Driving Demo* [102] shown in Figure 46. *Dash* is built on top of *Flask*, allowing near limitless customization [105]. Some effort has been put into making a web app with *Dash*, but much work remains.

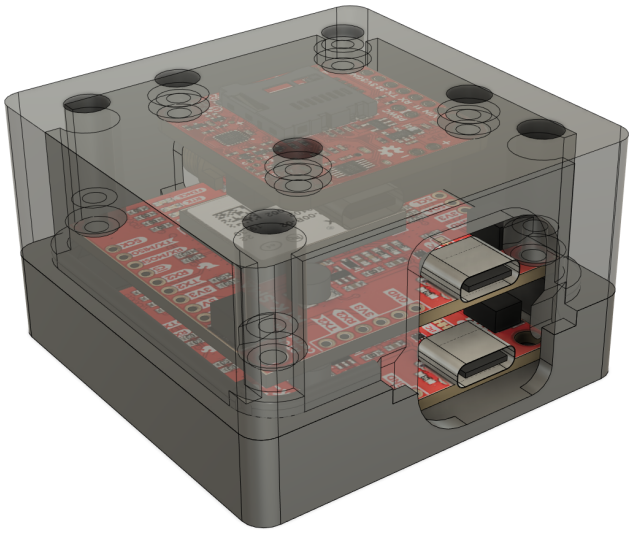


Figure 48: Enclosure design for trackin units, holding one OpenLog Artemis (top) and two F9Ps (bottom).

8.8 Tracking boxes

A final goal, related to the sensor rig, is to make small tracking boxes that can be mounted on moving objects to automatically label data. A prototype for the mechanical design of these boxes is shown in Figure 48. The box is intended to include two F9Ps and one OpenLog Artemis [106]. OpenLog Artemis, shown in Figure 49, makes logging from the F9Ps very simple, as they can be connected using Qwiic connectors, requiring no soldering, and log the data directly to an SD-card [106]. The OpenLog Artemis also has a built-in IMU running at $100Hz$ [106] that will help estimate the moving targets' pose. It is also simple to power, as it can be powered directly from a single cell Li-Po battery and serve as a charger for the battery [106]. Three OpenLog Artemis boards have been acquired and tested with the F9Ps from the sensor rig.

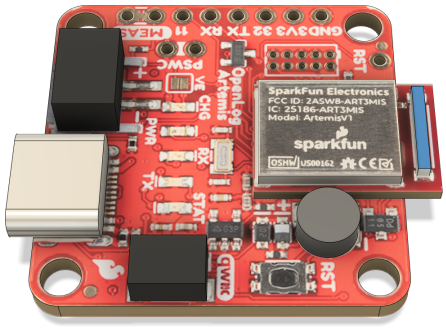


Figure 49: Digital model of the OpenLog Artemis

9 Conclusion

In this report, I have documented how I have designed and built a human-operable sensor rig for multi-sensor data acquisition in maritime environments. It has been a practical-oriented multideplicinary process where I have had the opportunity to learn several new skills. The project has involved designing and manufacturing both mechanical and electrical components as well as writing software for real-time data acquisition in Python, C and Pico assembly for different platforms. Some challenges were unforeseen, notably compiling the Linux kernel from source, reverse engineering the NTRIP protocol and acquiring a suiting network interface card during the global chip shortage.

Looking back it is clear that I miscalculated the amount of work needed to create a working product. My respect for hardware and electrical engineers has increased even more and I have yet again been reminded to never forget that Things Take Time (TTT).

The plan for the road ahead is to start mounting sensors on the sensor rig, calibrate them and start collecting data. This work will start this summer and be completed in my master thesis, where I intend to collect stereo-video from two polarization cameras, using the sensor rig, and research how the sensors can be used to improve detection capabilities for ASVs.

Hopefully, this sensor rig will serve as a useful and frequently used tool throughout my Ph.D.

Glossary

AsyncIO	Python library for asynchronous programming
CUDA	API used to program NVIDIA GPUs
JTOP	System monitoring utility for for Jetson products
Kartverket	The Norwegian Mapping Authority
NavBox	The box containing the navigation sensors
Numba	Python library for just in time compilation
Pybind11	Python library for binding C++ functions
RegEx	Regula expression, a type of text search
STIM300	An industrial grade IMU

Acronyms

ASV	Autonomous Surface Vessel
CAD	Computer-Aided Design
CIPO	Controller In Pheropheral Out
CPHA	Clock Phase
CRC	Cyclic Redundancy Check
CS	Chip Select
DGNSS	Differential GNSS
DMA	Direct Memory Access Unit
ESKF	Error State Kalman Filter
F9P	ZED-F9P GNSS module
FIFO	First In First Out
GNSS	Global navigation satellite systems
GPIO	General Purpose I/O
I/O	Input and Output
IMU	Inertial Measurement Unit
IP	Internet Protocol
Li-Po	Lithium Polymer
MEMS	Micro-Electro-Mechanical Systems
NTP	Network Time Protocol daemon
NTRIP	Networked Transport of RTCM via IP
OV	Omega Verksted
Pico	Raspberry Pi Pico microcontroller
PIO	Programmable I/O
PMMA	Poly(methyl methacrylate)
PoE	Power over Ethernet
PPS	Pule Per Second
PTP	Precision Time Protocol
ROS	Robot Operating System
RTCM	Radio Technical Commission for Maritime Services
RTK	Real-Time Kinematic positioning
SLAM	Simultaneous Localization And Mapping
SPI	Serial Peripheral Interface
SSH	Secure Shell or Secure Socket Shell
TOV	Time Of Validity
TTT	Things Take Time
UART	Universal Asynchronous Receiver-Transmitter
UTC	Coordinated Universal Time
Xavier	NVIDIA Jetson Xavier AGX Developer kit

References

[1] S. IEx, "Degrees of Protection." [Online]. Available: http://www.sourceiex.com/Catalogs/IP%20Degress%20Testing%20Details.pdf.

[2] L. Carolo. "SolidWorks 2022 vs Fusion 360: The Differences," All3DP. (Aug. 26, 2021), [Online]. Available: https://all3dp.com/2/fusion-360-vs-solidworks-cad-software-compared-side-by-side/ (visited on 05/18/2022).

[3] BOOPLA. "Enclosure," BOPLA. (), [Online]. Available: https://www.bopla.de/en/enclosure-technology/product/bocube/pc-ul-94-v0-crystal-clear-lid/b-261712-pc-v0-g-7024.html (visited on 05/17/2022).

[4] formlabs. "The 2022 3D Printing Applications Report." (), [Online]. Available: https://formlabs.drift.click/74c0506e-fddb-4221-b325-3fdaa037a281 (visited on 05/18/2022).

[5] M. Attaran, "The rise of 3-D printing: The advantages of additive manufacturing over traditional manufacturing," *Business Horizons*, vol. 60, no. 5, pp. 677–688, Sep. 1, 2017, ISSN: 0007-6813. DOI: 10.1016/j.bushor.2017.05.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007681317300897 (visited on 05/18/2022).

[6] O. Verksted. "Price of PLA filament 10 gram at Omega Verksted." (), [Online]. Available: https://www.omegav.ntnu.no/komp?search=FDM-3D-Printer+filament-PLA+10+gram (visited on 05/18/2022).

[7] B. Spennberg. "What is Kerf? - Parts Badger - Your Online Machine Shop," Parts Badger. (Dec. 13, 2019), [Online]. Available: https://parts-badger.com/whats-a-kerf/ (visited on 05/18/2022).

[8] N. F. N. Store. "M2 M3 100pcs Insert Knurled Nuts Brass Hot Melt Inset Nuts Heating Molding Copper Thread Inserts Nut Free Shipping - Nuts - AliExpress," aliexpress.com. (), [Online]. Available: //www.aliexpress.com/item/4001258499799.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp= (visited on 05/18/2022).

[9] TAMIYA. "Tamiya Extra Thin Cement w/Brush 40ml 300087038 - Craft accessories - Accessories - Categories - www.tamiya.de." (), [Online]. Available: https://www.tamiya.de/tamiya_en/categories/accessories/craft-accessories/tamiya-extra-thin-cement-wbrush-40ml-300087038-en.html (visited on 05/18/2022).

[10] biltema. "Akrylplast, opalhvit." (), [Online]. Available: https://www.biltema.no/bygg/platematerialer/akryplast/akrylplast-opalhvit-2000043756 (visited on 05/18/2022).

[11] polymerdatabase. "Poly(methyl methacrylate)." (), [Online]. Available: https://polymerdatabase.com/polymers/polymethylmethacrylate.html (visited on 05/18/2022).

[12] E. S. LLC. "Nifty Dogbones for Fusion 360," Ekins Solutions, LLC. (), [Online]. Available: https://ekinssolutions.com/product/nifty-dogbones-f360/ (visited on 05/16/2022).

[13] R. Components. "RS PRO Aluminium Strut, 40 x 40 mm, 8mm Groove , 1000mm Length | RS Components." (), [Online]. Available: https://no.rs-online.com/web/p/tubing-and-profile-struts/7613319 (visited on 05/16/2022).

[14] E. Composites. "30mm (27mm) Woven Finish Carbon Fibre Tube; 1m, 2m - Easy Composites." (), [Online]. Available: https://www.easycomposites.co.uk/30mm-woven-finish-carbon-fibre-tube (visited on 05/16/2022).

[15] R. Components. "Rose+Krieger Connecting Component, Parallel Clamp, strut profile 30 mm | RS Components." (), [Online]. Available: https://no.rs-online.com/web/p/connecting-components/4489615/?sra=pmpn (visited on 05/16/2022).

[16] STOTON, "TOP106 GNSS L1/L2 Multiband Antenna." [Online]. Available: https://cdn.sparkfun.com/assets/b/4/6/d/e/TOP106_GNSS_Antenna.pdf (visited on 05/26/2022).

[17] *ZED-F9P Integration manual*.

[18] F. Pommerening, *Finger Joints*, Apr. 8, 2022. [Online]. Available: https://github.com/FlorianPommerening/FingerJoints (visited on 05/16/2022).

[19] L. V. Labs. "Triton Industrial GigE Camera | LUCID Vision Labs." (), [Online]. Available: https://thinklucid.com/triton-gige-machine-vision/ (visited on 05/17/2022).

[20] BOPLA. "Bocube enclosure," BOPLA. (), [Online]. Available: https://www.bopla.de/en/enclosure-technology/product/bocube/pc-ul-94-v0-crystal-clear-lid/b-261712-pc-v0-g-7024.html (visited on 05/18/2022).

[21] M. Toygar. "NVIDIA Jetson AGX Xavier | 3D CAD Model Library | GrabCAD." (), [Online]. Available: https://grabcad.com/library/nvidia-jetson-agx-xavier-1 (visited on 05/17/2022).

[22] Eric. "Puffed Lipo Battery: Why they swell and what to do about it ▪ LearningRC," LearningRC. (Mar. 15, 2017), [Online]. Available: http://learningrc.com/puffed-lipos/ (visited on 05/19/2022).

[23] "GPS.gov: Other Global Navigation Satellite Systems (GNSS)." (), [Online]. Available: https://www.gps.gov/systems/gnss/ (visited on 05/21/2022).

[24] u-blox, "GPS Essentials of Satellite Navigation," GPS-X-02007-D, 2009. [Online]. Available: https://content.u-blox.com/sites/default/files/gps_compendiumgps-x-02007.pdf.

[25] GMV. "Differential GNSS - Navipedia." (Jul. 23, 2018), [Online]. Available: https://gssc.esa.int/navipedia/index.php/Differential_GNSS (visited on 05/21/2022).

[26] u-blox, "ZED-F9P Moving base applications," Datasheet UBX-19009093 - R02. [Online]. Available: https://content.u-blox.com/sites/default/files/ZED-F9P-MovingBase_AppNote_%28UBX-19009093%29.pdf (visited on 05/02/2022).

[27] u-blox, "ZED-F9P-04B Data sheet," p. 25,

[28] E. Brekke, *Fundamentals of Sensor Fusion Target Tracking, Navigation and SLAM*, Third edition. Aug. 2, 2021.

[29] W. G. Wong. "MEMS IMUs: The Ultimate in Sensor Fusion," Electronic Design. (Dec. 1, 2017), [Online]. Available: https://www.electronicdesign.com/technologies/test-measurement/article/21805896/mems-imus-the-ultimate-in-sensor-fusion (visited on 06/07/2022).

[30] Safran, "STIM300 Datasheet," Datasheet TS1524 rev.28. [Online]. Available: https://d29ykr7lqkqqrr.cloudfront.net/media/5z5lv25o/ts1524-r28-datasheet-stim300.pdf (visited on 05/02/2022).

[31] Safran, "STIM300 Product brief." [Online]. Available: https://d29ykr7lqkqqrr.cloudfront.net/media/vl4jeya3/2022-04-06-product-brief-stim300-a4_high-quality-print.pdf.

[32] J. Sonnenberg, "Serial Communications RS232, RS485, RS422," p. 6, 2018.

[33] R. P. T. Ltd, "Raspberry Pi Pico Datasheet," 150df05-clean, Nov. 4, 2021. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf.

[34] T. Instruments, "DS8921x Differential Line Driver and Receiver Pair," Datasheet SNLS374D –MAY 1998–, Jan. 2015. [Online]. Available: https://www.digikey.no/en/products/detail/texas-instruments/DS8921AMX-NOPB/366600 (visited on 05/12/2022).

[35] D. incorperated, "74HC32 datasheet," Datasheet DS35324 Rev. 3 - 2, Jan. 2013. [Online]. Available: https://www.diodes.com/assets/Datasheets/74HC32.pdf.

[36] B. Schweber. "What does an analog driver/buffer do?" Analog IC Tips. (Jun. 6, 2017), [Online]. Available: https://www.analogictips.com/what-does-an-analog-driverbuffer-do/ (visited on 05/23/2022).

[37] NVIDIA, "NVIDIA Jetson AGX Xavier Developer Kit Carrier Board," SP-09778-001_v2.1, Jun. 2020.

[38] Sparkfun. "SparkFun GPS-RTK-SMA Breakout - ZED-F9P (Qwiic) - GPS-16481 - SparkFun Electronics." (2022), [Online]. Available: https://www.sparkfun.com/products/16481 (visited on 05/21/2022).

[39] Telegartner, "Montageanweisung / Assembly Instruction C04xx." [Online]. Available: https://docs.rs-online.com/cc0c/0900766b80154af1.pdf (visited on 05/26/2022).

[40] juan aragon, director, *RG59 Making a TNC connection*, May 16, 2020. [Online]. Available: https://www.youtube.com/watch?v=M8Zl1cUe_3Q (visited on 05/26/2022).

[41] A. Wire, "Alpha Wire Unterminated to Unterminated Coaxial Cable, RG174/U, 50 \Omega, 30m," 2013. [Online]. Available: https://docs.rs-online.com/1380/0900766b8139013d.pdf (visited on 05/26/2022).

[42] A. RF. "T1121A1-ND3G-1-50 | Amphenol RF, TNC Connector | RS Components." (), [Online]. Available: https://no.rs-online.com/web/p/coaxial-connectors/1943235 (visited on 05/26/2022).

[43] Telegartner. "J01150A0069 | Telegartner 50\Omega Right Angle Cable Mount, SMA Connector , Plug | RS Components." (), [Online]. Available: https://no.rs-online.com/web/p/coaxial-connectors/1938984 (visited on 05/26/2022).

[44] R. P. T. Ltd, "Raspberry Pi Pico C/C++ SDK," Datasheet 150df05-clean, Nov. 4, 2021, p. 375. [Online]. Available: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf (visited on 05/02/2022).

[45] O. S. H. Association. "A Resolution to Redefine SPI Signal Names," Open Source Hardware Association. (), [Online]. Available: https://www.oshwa.org/a-resolution-to-redefine-spi-signal-names/ (visited on 05/23/2022).

[46] M. Grusin. "Serial Peripheral Interface (SPI) - learn.sparkfun.com." (), [Online]. Available: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all (visited on 05/23/2022).

[47] u-blox, "ZED-F9P Integration manual," Datasheet UBX-18010802 - R11, p. 119. [Online]. Available: https://content.u-blox.com/sites/default/files/ZED-F9P_IntegrationManual_UBX-18010802.pdf (visited on 05/02/2022).

[48] Ben Eater, director, *SPI: The serial peripheral interface*, Sep. 4, 2021. [Online]. Available: https://www.youtube.com/watch?v=MCi7dCBhVpQ (visited on 05/26/2022).

[49] R. P. T. Ltd, "RP2040 Datasheet," 150df05-clean, Nov. 4, 2021. [Online]. Available: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf.

[50] T. R. Kuphaldt, "Lessons In Electric Circuits," vol. Volume IV – Digital, p. 517, Nov. 1, 2007. [Online]. Available: https://www.allaboutcircuits.com/assets/pdf/digital.pdf.

[51] triss64738. "Signal delays between components in the RP2040 - Raspberry Pi Forums." (Jun. 16, 2021), [Online]. Available: https://forums.raspberrypi.com/viewtopic.php?t=325355 (visited on 05/21/2022).

[52] S. Caudle, *Python Spidev*, May 3, 2022. [Online]. Available: https://github.com/doceme/py-spidev (visited on 05/27/2022).

[53] B. Croston, *RPi.GPIO: A module to control Raspberry Pi GPIO channels*, version 0.7.1, Feb. 6, 2022. [Online]. Available: http://sourceforge.net/projects/raspberry-gpio-python/ (visited on 05/27/2022).

[54] u-blox, "ZED-F9P Interface Description," Datasheet UBX-18010854 - R07. [Online]. Available: https://cdn.sparkfun.com/assets/f/7/4/3/5/PM-15136.pdf (visited on 05/02/2022).

[55] Ben Eater, director, *How do CRCs work?* Apr. 28, 2019. [Online]. Available: https://www.youtube.com/watch?v=izG7qT0EpBw (visited on 05/27/2022).

[56] "9. Different CRC algorithms," How does it work? Automatics, computers, etc... (), [Online]. Available: https://pidlaboratory.com/9-rozne-algorytmy-crc/ (visited on 05/16/2022).

[57] *Pyubx2*, SEMU Consulting, May 20, 2022. [Online]. Available: https://github.com/semuconsulting/pyubx2 (visited on 05/28/2022).

[58] "Online CRC-8 CRC-16 CRC-32 Calculator." (), [Online]. Available: https://crccalc.com/ (visited on 05/16/2022).

[59] M. Adler. "Answer to "How can calculate mpeg2/crc32 in Python?"." (Sep. 27, 2021), [Online]. Available: https://stackoverflow.com/a/69340177 (visited on 05/16/2022).

[60] Kartverket. "Guide to CPOS," Kartverket.no. (), [Online]. Available: https://www.kartverket.no/en/on-land/posisjon/guide-to-cpos (visited on 05/02/2022).

[61] R. S. C. NO.104. "RTCM 10410.1 Standard for Networked Transport of RTCM via Internet Protocol (Ntrip) Version 2.0 with Amendment 2, January 12, 2021," Radio Technical Commission for Maritime Services. (), [Online]. Available: https://rtcm.myshopify.com/products/rtcm-10410-1-standard-for-networked-transport-of-rtcm-via-internet-protocol-ntrip-version-2-0-with-amendment-1-june-28-2011 (visited on 05/02/2022).

[62] "Networked Transport of RTCM via Internet Protocol," 1, 2004. [Online]. Available: https://gssc.esa.int/wp-content/uploads/2018/07/NtripDocumentation.pdf.

[63] E. Software. "RTCM-Ntrip." (), [Online]. Available: https://software.rtcm-ntrip.org/ (visited on 05/02/2022).

[64] jcmb. "NTRIP/NtripClient.py at master · jcmb/NTRIP," GitHub. (), [Online]. Available: https://github.com/jcmb/NTRIP (visited on 05/28/2022).

[65] SNIP. "An RTCM 3 message cheat sheet," SNIP Support. (Nov. 12, 2020), [Online]. Available: https://www.use-snip.com/kb/knowledge-base/an-rtcm-message-cheat-sheet/ (visited on 05/02/2022).

[66] NVIDIA, "NVIDIA Jetson AGX Xavier Series System-on-Module," DS-09654-002_v1.6, 2021. [Online]. Available: https://developer.nvidia.com/jetson-agx-xavier-series-datasheet.

[67] NVIDIA, "Jetson AGX Orin Series and Jetson AGX Xavier Series Interface Comparison and Migration," DA-10655-001_v1.1, Mar. 2022.

[68] I. C. Education. "What is Docker? | IBM." (Jun. 23, 2021), [Online]. Available: https://www.ibm.com/cloud/learn/docker (visited on 06/07/2022).

[69] P. Loshin. "What is SSH (Secure Shell) and How Does it Work? Definition from TechTarget," SearchSecurity. (), [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Secure-Shell (visited on 05/19/2022).

[70] M. Sadowski. "Enabling PPS on Jetson Nano," msadowski blog. (Apr. 28, 2020), [Online]. Available: https://msadowski.github.io/pps-support-jetson-nano/ (visited on 05/19/2022).

[71] NVIDIA, *NVIDIA SDK Manager*, Apr. 11, 2019. [Online]. Available: https://developer.nvidia.com/nvidia-sdk-manager (visited on 05/20/2022).

[72] F. van Dorsselaer, *Usbipd-win*, May 18, 2022. [Online]. Available: https://github.com/dorssel/usbipd-win (visited on 05/19/2022).

[73] Tomlogan501. "Enabling PPS on Xavier AGX - Jetson & Embedded Systems / Jetson AGX Xavier," NVIDIA Developer Forums. (Dec. 18, 2020), [Online]. Available: https://forums.developer.nvidia.com/t/enabling-pps-on-xavier-agx/147762/16?u=emil.martens (visited on 05/20/2022).

[74] NVIDIA, "NVIDIA Jetson AGX Xavier Developer Kit Pinmux," 1.4, Jan. 28, 2022.

[75] *Tegra194-gpio.h source code [linux/include/dt-bindings/gpio/tegra194-gpio.h] - Woboq Code Browser*, version v5.1-rc2, Mar. 29, 2019. [Online]. Available: https://code.woboq.org/linux/linux/include/dt-bindings/gpio/tegra194-gpio.h.html (visited on 05/20/2022).

[76] shgarg. "HOW to Increase GNNS timing module on Xavier? - Jetson & Embedded Systems / Jetson AGX Xavier," NVIDIA Developer Forums. (Jan. 2, 2020), [Online]. Available: https://forums.developer.nvidia.com/t/how-to-increase-gnns-timing-module-on-xavier/107409/8 (visited on 05/20/2022).

[77] *Network Time Protocol daemon - ArchWiki*, May 16, 2022. [Online]. Available: https://wiki.archlinux.org/title/Network_Time_Protocol_daemon (visited on 05/20/2022).

[78] *Chrony – Introduction*, Dec. 16, 2021. [Online]. Available: https://chrony.tuxfamily.org/ (visited on 05/20/2022).

[79] admin. "CentOS / RHEL 7 : Chrony V/s NTP (Differences Between ntpd and chronyd) – The Geek Diary." (), [Online]. Available: https://www.thegeekdiary.com/centos-rhel-7-chrony-vs-ntp-differences-between-ntpd-and-chronyd/ (visited on 05/20/2022).

[80] M. Todorov. "How to Install and Use Chrony in Linux." (), [Online]. Available: https://www.tecmint.com/install-chrony-in-centos-ubuntu-linux/ (visited on 05/20/2022).

[81] Alexis. "Answer to "Creating an isolated NTP server with only 1PPS and no external "time" input (No GPS)"," Unix & Linux Stack Exchange. (Jan. 27, 2022), [Online]. Available: https://unix.stackexchange.com/a/688159 (visited on 05/20/2022).

[82] Fedora, *13.3.5. Checking if chrony is Synchronized*. [Online]. Available: https://docs.fedoraproject.org/en-US/Fedora/22/html/System_Administrators_Guide/sect-Checking_if_chrony_is_synchronized.html (visited on 05/20/2022).

[83] kangalow. "SPI on Jetson - Using Jetson-IO," JetsonHacks. (May 4, 2020), [Online]. Available: https://jetsonhacks.com/2020/05/04/spi-on-jetson-using-jetson-io/ (visited on 05/20/2022).

[84] V. Thoms, *Spidev: Python bindings for Linux SPI access through spidev*, version 3.5. [Online]. Available: http://github.com/doceme/py-spidev (visited on 05/20/2022).

[85] ShaneCCC. "Is there any spidev*.* in Jetson AGX with native JetPack 4.6? - Jetson & Embedded Systems / Jetson AGX Xavier," NVIDIA Developer Forums. (Sep. 29, 2021), [Online]. Available: https://forums.developer.nvidia.com/t/is-there-any-spidev-in-jetson-agx-with-native-jetpack-4-6/190355/3?u=emil.martens (visited on 05/20/2022).

[86] C. Ltd. "Ubuntu Manpage: /etc/modules - kernel modules to load at boot time." (2019), [Online]. Available: http://manpages.ubuntu.com/manpages/bionic/man5/modules.5.html (visited on 05/20/2022).

[87] NVIDIA, *Jetson.GPIO - Linux for Tegra*, NVIDIA Corporation, May 12, 2022. [Online]. Available: https://github.com/NVIDIA/jetson-gpio (visited on 05/20/2022).

[88] R. Bonghi, *Jtop · rbonghi/jetson_stats Wiki*. [Online]. Available: https://github.com/rbonghi/jetson_stats (visited on 05/20/2022).

[89] I. Corporation, "Intel® Dual Band Wireless-AC 8265," 334064-005, 2016. [Online]. Available: https://docs.rs-online.com/4813/A700000006829287.pdf.

[90] chili555. "Answer to "Automatically connect to a wireless network using CLI"," Ask Ubuntu. (Jan. 29, 2014), [Online]. Available: https://askubuntu.com/a/412394 (visited on 05/20/2022).

[91] Gnat. "Answer to "Changing the metric of an interface permanently"," Unix & Linux Stack Exchange. (Dec. 25, 2017), [Online]. Available: https://unix.stackexchange.com/a/413036 (visited on 05/20/2022).

[92] B. Nadel. "How to use a smartphone as a mobile hotspot," Computerworld. (), [Online]. Available: https://www.computerworld.com/article/2499772/how-to-use-a-smartphone-as-a-mobile-hotspot.html (visited on 05/20/2022).

[93] juiceSSH. "JuiceSSH - Free SSH client for Android." (), [Online]. Available: https://juicessh.com/ (visited on 05/20/2022).

[94] player_one. "Permanent mounts - Linux Filesystems 101 - Block Devices," CodinGame. (), [Online]. Available: https://www.codingame.com/playgrounds/2135/linux-filesystems-101---block-devices/permanent-mounts (visited on 05/11/2022).

[95] B. Schneider. "A Guide to Understanding LiPo Batteries," Roger's Hobby Center. (Oct. 9, 21), [Online]. Available: https://rogershobbycenter.com/lipoguide (visited on 05/29/2022).

[96] "Digital Low Voltage Protector Disconnect Switch Cut Off 12V Over-Discharge Protection Module for 12-36V Lead Acid Lithium Battery : Patio, Lawn & Garden." (), [Online]. Available: https://www.amazon.com/Digital-Battery-Low-Voltage-Protection/dp/B07929Y5SZ (visited on 06/06/2022).

[97] L. V. Labs. "Triton 5.0MP Polarization Camera, Sony's IMX264MZR / MYR CMOS | LUCID Vision Labs." (), [Online]. Available: https://thinklucid.com/product/triton-5-0-mp-polarization-model-imx264mzrmyr/ (visited on 05/30/2022).

[98] L. V. Labs, "LUCID-Going-Polarized-White-Paper," 2018.

[99] *Ethz-asl/kalibr*, ETHZ ASL, May 27, 2022. [Online]. Available: https://github.com/ethz-asl/kalibr (visited on 05/30/2022).

[100] "RTKLIB: An Open Source Program Package for GNSS Positioning." (), [Online]. Available: http://www.rtklib.com/ (visited on 05/30/2022).

[101] *SGL-UT/gnsstk*, Space and Geophysics Laboratory of ARL:UT at Austin, May 29, 2022. [Online]. Available: https://github.com/SGL-UT/gnsstk (visited on 05/30/2022).

[102] "Autonomous Vehicle Visualization." (), [Online]. Available: https://dash.gallery/dash-avs-explorer/ (visited on 05/30/2022).

[103] NVIDIA, "Jetson AGX Xavier Series Thermal Design Guide," TDG-08981-001_v1.3, Jun. 2021. [Online]. Available: https://developer.nvidia.com/embedded/dlc/jetson-agx-xavier-series-thermal-design-guide.

[104] M. Wills. "Web_video_server - ROS Wiki." (), [Online]. Available: http://wiki.ros.org/web_video_server (visited on 05/30/2022).

[105] T. Birchard. "Integrate Plotly Dash Into Your Flask App," Hackers and Slackers. (Dec. 10, 2018), [Online]. Available: https://hackersandslackers.com/plotly-dash-with-flask/ (visited on 05/30/2022).

[106] Sparkfun. "SparkFun OpenLog Artemis - DEV-16832 - SparkFun Electronics." (), [Online]. Available: https://www.sparkfun.com/products/16832 (visited on 05/30/2022).