Luleå University of Technology

Dept. of Computer Science, Electrical and Space Engineering

D7039E – Project in Industrial Computer Systems
E7025E – Project in Engineering Physics and Electrical engineering

Project *SailorAid*

Authors
Axelsson, Oskar,
Brolin, Daniel,
Eriksson, Kenny
Grape, Elias
Lundberg, Josef,
Sjölund, Johannes
Theolin, Henrik

Managing Editor
Brolin, Daniel

Supervisor
van Deventer, Jan

January 17, 2018

Abstract

Throughout history, sailing has been a key ingredient to our civilization. Whether for trading, fishing, exploring, or transporting, today's civilization would not have been the same if sailing was not invented. Today sailing has been developed more into a hobby from large sailing boats all the way down to one-man dinghies.

The *SailorAid* system was developed to help the boat commander (skipper) on dinghies to make better decisions. *SailorAid* can help novices skippers learn the ropes faster and experienced skippers perfect the skill. This could be achieved using inexpensive sensors to give the skipper feedback using graphics, sound, and vibrations through an Android phone application. Measurements were done on forces exerted on the centerboard, tilt of the dinghy, position, and velocity. The design was focused on being energy efficient, waterproof and easy to handle.

Contents

Glossary

ADC An Analog-to-digital converter (ADC) is a system that converts an analog signal into a digital signal. 22

AHRS An Attitude and Heading Reference System (AHRS) consists three axes sensors which provide attitude information, including roll, pitch and yaw. 20

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. 34

ARM The ARM architecture (ARM) is a family of reduced instruction set computing (RISC) architectures for computer processors. 16

CAD Computer-aided design (CAD) is a computer system which aids the creation and modification of some kind of design. 9

DCM Direction Cosine Matrix (DCM). 38

DMA Direct Memory Access (DMA) is a feature which allows hardware subsystems to access main system memory independent of the central processing unit (CPU). 17

GPS The Global Positioning System (GPS) is a radionavigation system owned by the United States government and operated by the United States Air Force. It uses sattelites for geolocation and time. 2

I^2C Inter-Integrated Circuit (I^2C), is a serial computer bus. 6

IMU Inertial Measurement Units (IMUs) are integrated circuits that can measure acceleration, rotational velocity and magnetic field strength. 2

INS Inertial Navigation System (INS). 36

IR Infrared radiation (IR) is electromagnetic radiation (EMR) with longer wavelengths than those of visible light. 14

LED A Light-emitting diode (LED), is a two-lead semiconductor light source. 16

LIDAR Light Detection and Ranging (LIDAR). 14

MAC Media access control address (MAC) is a unique identifier assigned to network interfaces. 33

MATLAB MATLAB (matrix laboratory) is a numerical computing environment. 21

MCU A Microcontroller Unit (MCU) is a single computer chip designed for embedded applications. 6

MEMS Microelectromechanical systems (MEMS), is the technology of microscopic devices. 19

NMEA The National Marine Electronics Association (NMEA) standard is a specification that defines the interface between various pieces of marine electronic equipment. 21

PCB A Printed Circuit Board (PCB) is the common acronym when referring to populated circuit boards.. 6

PNG Portable Network Graphics (PNG) is a raster graphics file format. 31

SEK Swedish Krona (SEK) is the currency in Sweden. 14

SINS Strapdown Inertial Navigation System (SINS). 36

SPI Serial Peripheral Interface Bus (SPI), is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. 17

ST STMicroelectronics (ST) is a French-Italian multinational electronics and semiconductor manufacturer. 16

UART Universal Asynchronous Receiver-Transmitter (UART) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It generally requires less power and is slower than its counterpart USART. 16

UML The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language. 23

USB Universal Serial Bus (USB), is an industry standard for cables, connectors and communications protocols for connection, communication, and power supply between computers and devices. 16

VIA A vertical interconnect access (via) is an electrical connection between layers in a physical electronic circuit.. 6

1 Introduction

Eriksson, Kenny (Sjölund, Johannes)

The art of sailing has been around for millennia. For much of human history it has been an absolutely vital part of civilization, providing efficient means of transporting goods all around the world. Today sailing has become a leisure activity enjoyed by millions of people around the world. Modern sailboats come in a large span of sizes, from large ships with crews of dozens down to small single-man dinghies.

While slipping across the waves out at sea with only the wind to drive you is a calming experience, it is not a simple thing to do. When you are alone on the water, you have to be in control of the tension of the sail, the attitude of the boat, the forces on the centerboard and more while deciding how to respond to all of these. The goal of Project *SailorAid* is to offload the decision-making from the sailor onto a compact, portable and simple system that will analyze these parameters and provide clear directions to the skipper.

1.1 Goals

The primary functional goals are the following:

- Boat attitude
  - Implementing appropriate Inertial Measurement Unit (IMU)s:
    - Accelerometer
    - Gyroscope
    - Magnetometer
  - Fusing the sensor output to get an accurate estimate of boat attitude
- Position tracking and velocity
  - Implementing a Global Positioning System (GPS) system
  - Fusing the GPS output with the accelerometer output for more accurate positioning and velocity
- Design a force measurement circuit for the centerboard
  - Design an appropriate mechanical externally mounted sensor
  - Implement an appropriate sensor
  - Implement a centerboard-depth sensor
- Feedback to the user of the system
  - Give the user information from the sensors
  - Display instructions to help improve the sailing experience depeding on the system state
  - Implement different ways for the user to retrieve information

2 The Physics of Sailing

Theolin, Henrik (Axelsson, Oskar)

2.1 Point of sail

A sailboat can increase velocity by catching the wind in the sail at different angles. This is called point of sail and the velocity is dependent on the dinghies displacement from the true wind direction, the wind experienced for a stationary object, where the velocity is a resultant of the force vector created by the wind depending on the alignment of the sail and the direction from the wind direction. There are five different states of point of sail that are divided into degrees away from the true wind origin. These are

Luffing (no propulsive force) angle between 0-30°

Close-hauled (lift) angle between 30-50°

Beam reach (lift) angle 90°

Broad reach (lift–drag) angle aound 135°

Running (drag) angle around 180°

and are represented in Figure 1. A sailor wants to prevent the sail from luffing, which is when the sail starts to flap in the wind and no propulsive force is achieved. When the dinghy is in the close-hauled and beam reach state, the sail produces lift force that is produced from the average pressure differences on the windward and leeward side of the sail where the pressure is higher on the windward surface thus acting like a wing, thus propelling the dinghy. When the dinghy is in the broad reach state both lift and also drag propels the dinghy. Drag acts like a parachute that catches the wind and propels the dinghy. The sideway force induced on the boat also introduces drift perpendicular to the relative bearing. This is counteracted by lowering a centerboard which also counteracts the dinghy from heeling.

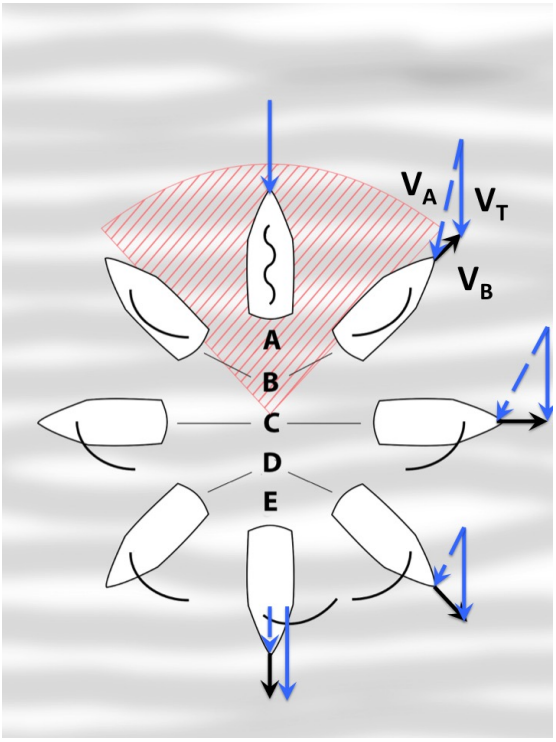


Figure 1: Points of sail.

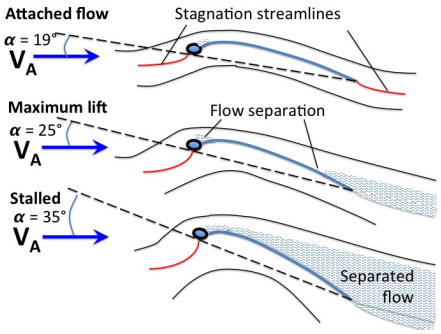


Figure 2: Sail angles of attack.

2.2 Velocity

The main goal in sailing is to maximize the efficiency at which the forces on the sail translates into the velocity of the dinghy. A layman might expect the fastest velocity to be achieved when the dinghy is parallel to the true wind, this is however not true. The apparent wind, which is the wind experienced from the dingy perspective, is what propels the dingy. When sailing parallel to the wind the dinghies speed can never exceed the speed of the wind[21]. By sailing upwind close-hauled the apparent wind is increased as the dingy accelerates

until the drag from the water exceeds the forward force created by the wind. To further increase the velocity, the dingy should not be heeling excessively. This is to prevent the centerboard from acting as a rudder and changing the bearing, introducing more drag from the stern rudder when compensating for the bearing changes. As mentioned earlier the centerboard helps to counteract heeling but also the sailor can prevent this by leaning out of the boat (hiking), to alter the center of mass for the dinghy. As a last resort the sailor must perform reefing to reduce the area of the sail to lower the center of effort from the sails.

3 Mechanical Sensors

Lundberg, Josef (Grape, Elias)

Since sailboats utilize wind to move, not fuel or electricity, optimization of the usage of surrounding forces are important. Measurements of these will provide the sailor with information needed to optimize the way to move and control the dinghy.

To make measurements on the dinghy, sensors are needed. As of now there are sensors measuring a wide array of items. This section describes; what they measure, where they were purchased, the requirements on the sensor, their features, drawbacks and also how they were implemented. Since the sensors have to work in this particular system, a casing was made for the sensors. The designs where made with the Computer-aided design (CAD) software Fusion 360[1]. The casings where manufactured with a 3D printer.

3.1 Force sensors

The function of the daggerboard is to compensate for the force that the wind is pushing on the sail, and keep the dingy from drifting in the water. The goal is to have a system that can measure the forces from the water that pushes on the daggerboard. By measuring this, an estimation of the exerted force on the sail can be made.

3.1.1 The implementation

Although different solutions where suggested the sensor and measurement method that was chosen his is shown in Figure 3. Important to know is that every solution is mandatory to be waterproof and sealed properly from the harsh environment that this system will operate in. With this implementation the daggerboard itself will not be disassembled or modified in any way; as solutions that require sensors to be mounted on the parts likely to be damaged in a crash was scratched.

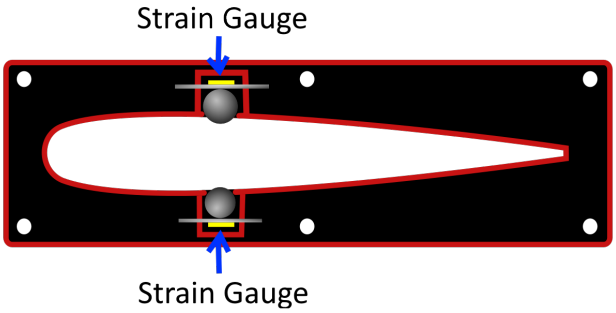


Figure 3: Function of first prototype.

3.1.2 The prototype

To integrate the gauges, a prototype was designed to show how the measurements would be made. The prototype shown in Figure 4 is a bit bigger than

the intended solution for this project but it is good to see how it would be constructed. The board goes on the outside and can easily slide up and down past the steel bead. The bead itself is kept inside a small space where it can move in and out.

The prototype of the pressure sensor was created in order to show the function of this sensor and to help the thought process involved in the improving of this design.

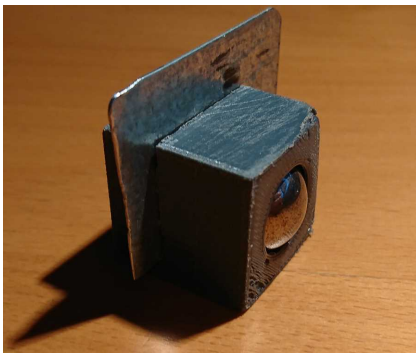


Figure 4: First prototype of a force sensor.

The force is measured at the back on the plate. The deflection of this plate will be used to measure the strain with strain gauges. The gauge itself will measure a small difference in resistance. This small difference is going to be difficult to measure without any amplifying circuit connected. With such a small signal the system might have issues with noise. Another problem is that the signal might drift, causing differences in the measurements during run-time. Because of this, with the measured values getting amplified, the drift differences may be off by unacceptable amounts.

In the second prototype load cells, which are sensors that utilizes strain gauges to measure forces, were chosen for this purpose. Rather than having a metal plate, this prototype can be built with a piece of plastic or rubber which can deform so the force is distributed directly to the sensor. By implementing load cells, a lot of time was saved in troubleshooting; and by having a sensor unit, the modified mounting plate will be easier to produce. An illustration of this setup can be seen in Figure 5.

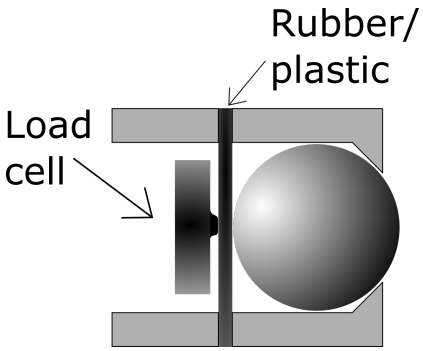


Figure 5: Function of second prototype.

3.1.3 Sensor

The force from the board on the mounting plate will be considerably large. The actual force is something that is not known for sure. The initial assumption was that a load cell with a 90.75 kg force range should be enough. If the sensor will be maxed out, the cell its rated for a 150% overload without causing some damage to the sensor.

The chosen sensor for this application is the compression load cell called *FX1901*. From the datasheet[31], the voltage readings of this part could be calculated. The maximum voltage difference is calculated to be around 180 mV, see Figure 6.

Figure 6: Load cell, *FX1901*.

3.2 Amplifier

With such a small signal, an amplifier is needed to get good measurements. A good measurement signal to the Microcontroller Unit (MCU) should be between 0-3.3 V. Since the maximum value from the load cell is 180 mV a signal of 3.3 V is achieved by an amplification gain of around 20 times. A suitable amplifier needs to be chosen from this implementation. Inspiration is taken from The University of Chicago[30] in an experiment where they use this exact load cell together with an instrumental amplifier called *INA125*.

In the same family of instrument amplifiers, the model *INA126* was selected as a less complicated and more power efficient solution. The *INA126* amplifier has a smaller power draw due to simpler functionality and lower precision than

the *INA125*, but is still sufficient for this application. The implementation is easy and the gain can easily be determined by connecting a resistor *Rg* between two pins on the amplifier, see Figure 7.

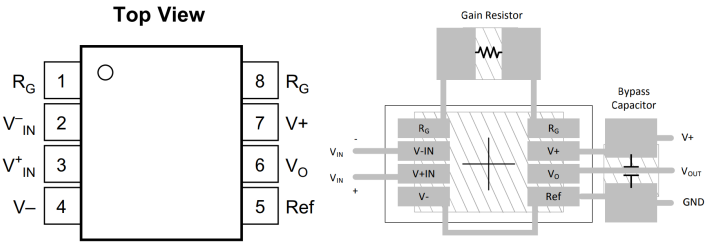


Figure 7: Amplifier *INA126* for the load cell signal.

The gain is calculated with the following equation.

$$\text{Gain} = 5 + \frac{80\ \text{k}\Omega}{R_G} \tag{1}$$

In order to adjust the gain of the amplifier, the resistor R_G is set to a fixed resistor in series with a potentiometer.

3.3 Height of dagger-board sensor

The height of the dagger-board is a crucial measurement in dinghy sailing. The dagger-board not only gives the dinghy a corresponding force to the sail, the board can also be a danger if you sail in running wind. One of the best implementations of a height sensor would be the use of a linear wire distance sensor. This sensor measures how far a wire is pulled, which gives a very accurate measurement. This solution can be completely watertight and concealed in the main center-plate[1]. Another solutions could be to use some type of light, Time of Flight (TOF) sensor.

3.3.1 Sensors

A suitable product was found, it is accurate and reliable, the *Micro epsilon mk30*; its physical volume in small enough with a size of $3x5$ cm. As the product have tight space constraints this would probably fit inside the case. The sensors price however was about 2000 kr, it was deemed to expensive. If no other solution would work as intended this might have been considered again.

[1]The plate on where the dagger-board is mounted.

Figure 8: Linear draw wire sensor, Micro epsilon MK30.

Light sensors have also been considered. This would be implemented with the use of a plate placed on top of the dagger-board and with the light being sent up to this panel, the height can be calculated. Initially, some Infrared radiation (IR)-sensors were considered, they will send the signal in a widespread arc which will make the distance measurement troublesome as this signal has just a small plate to bounce off.

With the use of a Light Detection and Ranging (LIDAR)ToF system, we can point our light signal at an exact spot and then get an exact measurement of the height. Many of the LIDAR systems found was both too large and expensive to be implemented in this project. But a product called "micro-lidar" could be used; as it has a small form factor it is easy to implement and is not to expensive. The chosen module is the *VL53L0X* from Adafruit[**?**], see Figure 9. It can measure heights up to one meter with great accuracy and communicates with Inter-Integrated Circuit (I^2C). By the fact that the sensor must be waterproof the signal has to go through a medium. The medium can be some type of transparent plastic or glass.

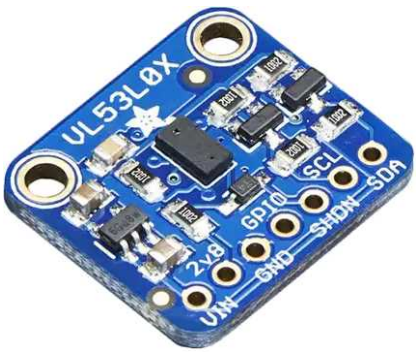


Figure 9: Time of flight, μLIDAR, distance sensor *Adafruit VL53L0X*.

All the information about this sensor and the usage of a cover widow can be found in the specific datasheet[39]. The maximum thickness of the cover window together with the air gap between the sensor and the window is 2 mm. That is the parameters that need to be considered. To reflect the signal a detection plate is constructed and fastened to the top of the dagger-board. This plate is level with the sensor and has a white bottom for best performance of the

sensor. In the case the sensor is mounted behind a thin epoxy window witch was to thick at the beginning but was sanded down to a suitable thickness. The final window was then carefully polished for the best possible measurements.

Figure 10: Height measurement model.

3.3.2 Results

The sensor works very well in the open air and can detect heights up to 1.5 m with great accuracy. It also works well when used behind a thin transparent plastic cover. With a 1 mm thin transparent plastic cover the maximum height measurement is decreased to about 1 m.

The performance of the sensor heavily depends on the the finish of the window, if there is small imperfections on the window such as scratches the sensor performance is greatly reduced or completely diminished due to the light spreading everywhere.

As there will always be water around and on the center plate the light that is sent might get scattered out off sight, even a small water droplet in between the sensor and the panel might cause a fault in the measurement.

This system can definitively be improved in the future with either a completely different measurement system or with a better implementation of the window cover. The window cover probably would work better if the cover was molded in a convex form, which could lead off the water droplets formed on the window surface. This in cooperation with a surface finish that repels water, such as a hydrophobic coating would be a great solution.

4 Main Sensorboard Design

Brolin, Daniel (Eriksson, Kenny)

This chapter will summarize most of the process of creating the PCB used to mount all the sensors and sending the data to the phone application. This circuit board will be the link between the physical forces and manipulated data sent to the handheld device. More on how this data is manipulated and how the circuit board is programmed in section 7.

The circuit board must minimally fulfill the following set of basic requirements to function properly:

- Circuit must fit within the casing, with dimensions extracted directly from the cad software[1], see Figure 67.
- Circuit must be easy to test for both hardware and software errors. All systems needs this, if nothing else only to test if they are correctly soldered.
- All sensors must fit on the PCB, both physically and also electronically on the ports of the MCU.
- Must be protected against all common problems, such as overcharge, undercharge, capacitive effects, and more.
- All components needs to be actively manufactured to ensure unit can keep being produced for several years in the future.

4.1 Method

Brolin, Daniel (Eriksson, Kenny)

The main circuit board is built using the open source electronic design software *KiCad*[4]. It is completely free to use and the source code is open for any modifications. Designing circuit boards differs slightly between what software you are using. *KiCad* uses a four step process as follows:

1. Draw schematic:
   (a) Place and connect all components electronically.
   (b) (Optional:) All non-standard components must be custom designed, this is referred to as a"symbol". Most sensors requires custom built symbols.
   (c) Annotate schematic and run Electronic Rule Check (ERC) to identify simple electronic violations.
2. Create net-list:
   (a) (Optional:) All components with non-standard (or simply not standard enough) footprints must be custom designed, these include all sensors and the Bluetooth (BT) module.
   (b) Associate all components with whatever physical footprint are to be used.
   (c) Generate net-list.
3. Design PCB layout:
   (a) Import net-list to PCB editor.
   (b) Set global and net-specific design rules.
   (c) Place and connect all components.

(d) Run Design Rule Check (DRC) to identify all specified design rule violations.

4. Generate manufacturing (gerber/drill) files.

The initial required circuitry, see Figure 59, was copied from earlier works with the STMicroelectronics (ST) ARM architecture (ARM) MCU's but modified to meet our needs. The power section, also found in Figure 59, is designed to only allow for exactly +5 V. This design is credited to, and derived from *Maxim Integrated*, Application note 760[37].

Non-standard library entries include following components:

Name	Type	Library	Used in last rev.[a]
A2235-H	GPS	a2235-h.lib	**YES**
FT232RL	USBtoSerial converter	ftdi.lib[36]	**YES**
HTS221	Humidity sensor	Custom_sensors.lib	**YES**
LM1117ADJ	Voltage Regulator	PowerSupply.lib[34]	**YES**
LPS22HB	Pressure sensor	Custom_sensors.lib	**YES**
LSM303AGR	Acc.[b]/Mag.[c] IMU	Custom_sensors.lib	**YES**
LSM303Cx	Acc./Mag. IMU	Custom_sensors.lib	**NO**
LSM6DSL	Acc./Gyro.[d] IMU	Custom_sensors.lib	**YES**
LSM9DS1	Acc./Gyro./Mag.	Custom_sensors.lib	**NO**
MCDTS2-4N	Tactile switch button	Custom_Switches.lib	**YES**

Table 1: Non-standard library entries, custom or third-party designed.

[a] revision
[b] Accelerometer
[c] Magnetometer
[d] Gyroscope

In total three revisions of the PCB was designed. The first revision was designed well before any components arrived and was meant exclusively as a prototype, while both the second and third revisions were supposed to have full functionality. All revisions following the first were initially supposed to be designed as consecutive debugged states, but complete redesigns of component placings and tracing was eventually necessary.

4.1.1 First revision

First revision included the main functionality and test points on basically every pin in the system. It did not follow any space- or requirements and was riddled with small problems. The *USB-to-Serial* converter was broken, the I^2C line was short circuited somewhere, the custom footprint for the GPS was mirrored, the BT worked, but poorly due to interference from other circuitry and so on and so fourth. Most of this was probably because we had to design the PCB and draw all custom made footprints before actually receiving the components. This revision was also meant as a prototype, and not enough time was put to make sure all isolations and trace widths had well thought out values. High-speed switching logic had been designed for the wakeup-signal to the GPS, but this

was not used. The schematic of revision one has not been included in this report, as there is little to learn from it; the PCB can however be seen as Figure 11.

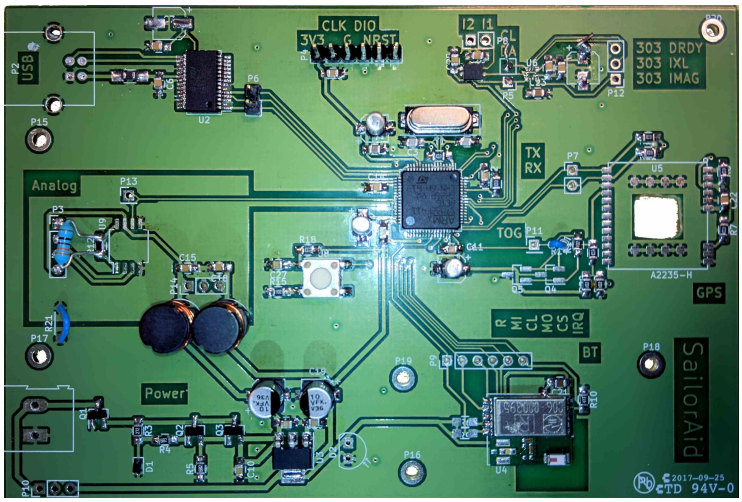


Figure 11: First (1) revision circuit board.

4.1.2 Second revision

The second revision fixed all known problems, followed the space requirements and added a lot of customization. The option to toggle power source and what sensors were currently powered was added. This way the effect from individual modules could be measured. Also since it was still unclear if a combined chip for all IMUs would be added or not, space for all of them and an evaluation module was added. However, some wires were lost in the transition from revision one to two, as all net-labels in the schematic were redrawn; also the micro-Universal Serial Bus (USB) housing was to small for continuous use and was ripped of by our (way to strong) programmers; all of this was hotfixed and eventually the circuit board worked as expected. As with revision one, the schematic of revision two has not been included in this report, as there is little to learn from it; the PCB can however be seen as Figure 12.

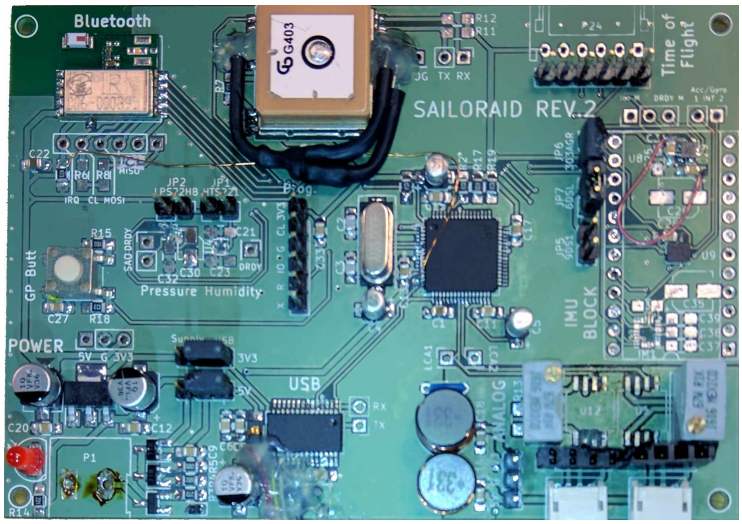


Figure 12: Second (2) revision circuit board.

4.1.3 Third revision

Almost all problems were solved in the second revision; what remained was some hotfixing for an unexpected wiring error in the GPS, interference between I^2C clock and data line, minimizing much to large decoupling ground loops, unwanted antennas, ground islands of missing copper and a KiCad problem where disconnected vias lose their net property. The option to pull power from either USB or batteries remained in the last revision, as this makes programming and testing much easier. A four pin connector to measure the battery level through I^2C was added; and since the power level of this depends slightly on the power left in the batteries, logic level converters were designed to allow for variable voltage levels[35], the electronic circuitry can be see in Figure 60. For the third revision all decisions have been made and the schematic can be seen as Figure 59, 60 and 61, section A.1.

Following communicative devices will used and act as "module areas", i.e. these units will be the central units of the PCB layout. This was introduced to the second revision PCB, but is much more prominent in the last revision (Figure 14).

- STM32F411RET - Microcontoller unit
- FT232RL - USBtoSerial converter (USART)
- A2235-H - GPS unit (UART)
- SPBTLE-RF - BT connection unit (SPI)
- Sensors (I^2C):
  - LSM303AGR - Accelereometer & Magnetometer
  - LSM6DSL - Accelerometer & Gyroscope
  - HTS221 - Humidity
  - LPS22HB - Pressure
- Sensors, connectors only (I^2C):
  - Battery level

  - VL530x - Time of flight
- INA128 - Analog load cell amplifiers (ADC)
- MCDTS2-4N - Tactile push buttons (Pull-up/Pull-down)

The layout of the components are such that all I^2C communicative devices were moved to one spot ensuring the shortest possible I^2C data line, these were placed to the top right of the PCB. The power was placed in the bottom left, since this is as far as possible from everything else. USB was placed next to the power, as power can be opted to be drawn from USB rather than the battery. Analog plane with the load cells were put right next to the power to ensure the star-ground[2] to be as close as possible to the power-source allowing for as little noise as possible. BT communication devices are rather fragile, it was therefore placed as far as possible from everything in the top left corner of the PCB. This BT module, *SPBTLE-RF*, requires Serial Peripheral Interface (SPI) communication and in total 6 isolated traces were to be drawn between the MCU and the BT unit. To minimize crossover the GPS is placed in the top-mid of the PCB. Lastly the buttons whose placing is of low importance were placed in the middle-left, between the power source and the BT unit because this is where there was space left. Ground vias were added around communication traces to prevent unwanted interference, ground vias were also added to prevent antennas[3]. The third, and last, revision of the PCB can be seen in Figure 13, a simple explanation of the layout can be seen in Figure 14.

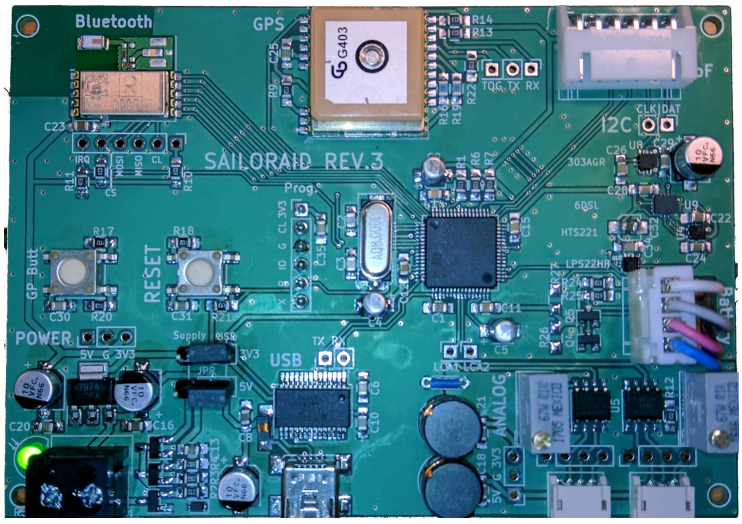


Figure 13: Third (3) revision circuit board.

[2] A common node ground connecting all grounds together to ensure the same set working level

[3] Antennas appear by letting isolated thin grounds be connected on only one end, a via can therefore be added to the other end to prevent this from happening.

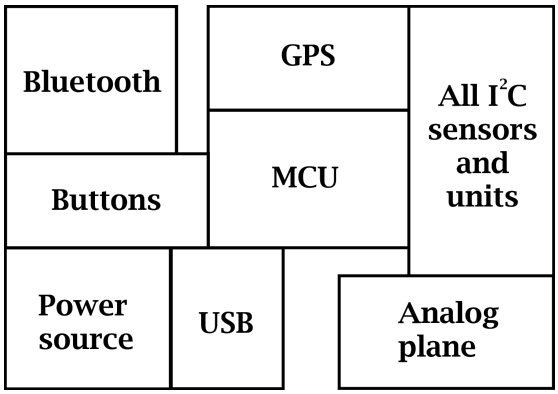


Figure 14: Simple model of third (3) revision circuit board areas.

4.2 Component choices

Eriksson, Kenny (Brolin, Daniel)

In order to choose what components would be used for the system, the first consideration was the central controller of the system. Because the system was designed to work in conjunction with an app on a smartphone it was agreed that there is no need for heavy computation on the on-board module. This meant that a MCU would be sufficient. Because several of the project members had previous experience with the ST brand *STM32F411RE* and it has a significant number of GPIO ports, useful peripherals (I^2C, SPI, timers, etc) and a high clock speed of up to 100 MHz it was chosen as the core of the on-board module.

The *STM32F411RE* utilizes the Serial Wire Debug (SWD) protocol for programming and debugging, the programmer in this case being the programming section of a *NUCLEO-F411RE* development board configured as a programmer. The SWD header also provides a Universal Asynchronous Receiver-Transmitter (UART) channel to communicate with the MCU. However, in the interest of convenience for the developers a *FT232RL* chip was added. The *FT232RL* is a *USB-to-UART* converter that also allows for the USB to act as a power source for the board. This eliminated the need for the prototypes to be connected to a power supply and the programmer at times when they were not necessary, giving the more convenient alternative of a common USB cable. The FT232RL was configured only to provide the communication conversion and the power. There are more features available to the FT232RL but no use was found as the on-board module is not connected to a USB port for the majority of its operation.

In order to communicate with the smartphone in a convenient way the on-board module used a Bluetooth module, the SPBTLE-RF. It was chosen as a compact complete module, eliminating the need for soldering inconvenient components and designing PCB antennas, and because ST Microelectronics provide official libraries for the *STM32F411RE-to-SPBTLE-RF* interface.

One of the primary functions that the system had to provide was velocity. The velocity is derived from the position of the boat as measured with GPS. This necessitated a GPS module. The project group from the previous year had thought the same thing, and among the remains of their project was a *A2235-H*

module. The *A2235-H* module fit the requirements of the project, as it provided circa 1Hz update rate with sub-meter precision in reasonable conditions, all within a compact form factor. It did however have some requirements on how it could be initialized, otherwise risking corruption of the GPS modules firmware.

The second primary function the board was supposed to provide was the ability to tell the boats orientation. The common Madgwick method discussed later relies on the use of an accelerometer, magnetometer and gyroscope in conjunction. Because the MCU had already been decided we looked for compatible development boards that had the required components and ideally also support libraries. Having found the *NUCLEO-1KS01A2* that satisfies this, the components were chosen from the development board to be able to reuse already written code.

For magnetometer, the *LSM303AGR* was what was on the development board. It incorporates both an accelerometer and a magnetometer and testing the development board showed it performed good enough for the intended purpose. For the gyroscope, the *LSM6DSL* had once again proved good enough for our purpose, and it too incorporated an accelerometer, possibly allowing for additional filtering possibilities in the future.

During the course of the project we considered two changes to the choice of components. One was to change from the *LSM303AGR* to the *LSM303C* because of apparent lack of availability, but we found another place to get the already chosen part. The second change we considered was to replace the *LSM6DSL* and *LSM303AGR* with the *LSM9DS1*, which incorporates the magnetometer, accelerometer and gyroscope in a single package to minimize the need to solder inconvenient packages, but because we could not find any official libraries we decided to not change over, so as to not have to rewrite the entire library by ourselves.

In addition to the IMU sensors, the development board also provided two additional sensors: the *HTS221*thermometer and relative humidity sensor, and the *LPS22HB* pressure sensor. These were included in the final design to give room for expansion.

4.3 Results, Discussion and Future Work

Brolin, Daniel & Eriksson, Kenny

The final sensor board works as expected. Had it not been for the fact that one of the sensors was broken it would have been soldered without any problem. No interference problems or antennas has been found on the last revision PCB and no hot fixes were necessary. Calibrating the potentiometers for the analog load cell amplifiers is possibly the most uncertain process, as wrongly tuning these will offset all values in the feedback. This is however necessary until the last prototype for the casing is complete, see section 6, since small differences in the casing will require different calibrations. The system is estimated to run for about 24 hrs, more about power management in section 5.

All components in the last revision prototype is of a rather large form factor considering the relatively low effect acting on the components, making soldering and testing easier but taking up more space. As it is now the system can very likely be minimized further. Taking away the general purpose button, set the general form factor to the industry standard 0402 (approximately 1 mm x

0.5 mm) which is still easily soldered by hand, use 4-layer design could probably shrink the system to about half of the size if no more devices are added to the design. Alternatively more sensors or communication with an external sensor board could be supported in future updates without the need of to much remodeling.

When placing test points, consider convenience. We did not have access to a bed-of-nails test bed and creating one within the span of the project, time- and budget constraints did not seem reasonable. In order to make debugging and testing easier, we placed test points at points of interest like communication buses. The test points were in the form of vias, however it would have been more convenient that they had been made larger, since the debugging tool was often an oscilloscope. The probe points did not fit the holes and they were inconveniently close together. This could have been improved.

Another lesson from the last revision was to use the tool for the job. Debugging the I^2C via the MCU is inconvenient simply due to the overhead associated with reprogramming, and possibly limiting unless the debug code is extensive and general. In this case it was far more convenient to acquire a logic analyzer. It provides multi-protocol communication and monitoring and is faster to setup and modify. The point being that a *test engineer* is an entire profession onto itself and we could have spent more time early on considering how to more efficiently test and debug to save more time during the rest of the project.

The most significant part of assembling each prototype was soldering the Land Grid Array (LGA) packages (the IMUs). Because the pins on these packages were on the underside and not accessible by means of a soldering iron. The only convenient tool for soldering these were to either get it right the first time in the Surface Mount Device (SMD) oven, or to use a hot-air station. Even when using these tools, the soldering was not trivial. To start with, the small size of the packages make them hard to place. Next, the packages were more likely to short due to the small margin for error. Third, even when the packages appear properly soldered, a few times they were not; while not being shorted they would still not function. In order to make the mounting easier, the footprint pads were extended beyond the edge of the package. This aided in finding shorts by giving easy probe points, and it provided some capacity to handle excess solder.

During the debugging of the final revision of the PCB there was a case of note. While soldering a new LSM6DSL to the board, we discovered a technique to consistently get good soldering results. Assuming that the board has already been stenciled and baked, there should be fresh solder on the pads. Apply generously with flux (better too much than too little) to clean the area and help flow. Using a hot-air station, preheat the board slowly from below for circa a minute and preheat the component towards the end (just flip the board and continue heating from the top while holding the component in the air flow). This prevents the solder from setting fast and making the soldering harder. When the solder visibly melts, place the package on the pads carefully and lightly push the component flat against the board. This forces any residual dirt away from the pads, and the extended footprints control where the displaced solder goes to avoid shorts. When the component has been released, lightly push it from the side. If the component springs back from surface tension it is a good chance that it has been soldered correctly, provided the component was pushed down before.

In terms of future work, most likely the first thing should be wind direction and/or intensity sensing. Because the direction of the wind has such a significant impact on the performance of the boat. It could possibly be implemented as a rotary encoder, or it might be possible to implement it through a forced-convection sensor[2]. Other things that could be interesting to measure would the either the state of the rudder or sail, although the latter one especially would probably be a significant task, there is the consideration of what the final scope of the product should be; if it should actually compete as an extensive set with all features the sailor could want or if it should try to go for the lower end of products and do that very well instead.

5 Battery Management

Grape, Elias (Lundberg, Josef)

5.1 Introduction

Lithium-ion batteries packs large amounts of energy in a small package and can power systems for long periods before a recharge is needed. Because of the high energy density they have to be handled with care. Beyond protection a BMS can be used to report battery percentages and remaining capacity. The specific type of battery used in this project is *ICR18650*.

5.2 Lithium-ion Battery Composition

Lithium-ion batteries are similar to other batteries in the way that they work by the same electrochemical principles. Though there are differences that has to be taken into account when developing a battery management system.

A lithium-based battery does not have any metallic lithium in it, instead, lithium ions are intercalated in the electrode material. Due to the highly reversible properties of the intercalation, Lithium-ion Batteries (LIB) have greater electrode stability and cycle life compared to other electrochemical processes. They have high coulombic efficiency in all states of charge. Self-discharge is also slower in LIB batteries which makes them useful in situations where they may not be used for extended periods and are expected to work once the system turns on.

Memory effects present in other chemistries which causes gradual loss of capacity if not discharged properly are not present in lithium-ion batteries and makes the usage more flexible. Trickle charging is a common method of determining the State of Charge (SoC), but lithium-ion batteries lead to overcharging and may damage the batteries.

Water-based electrolytes are common in most batteries but as electrolysis begins at two volts in water and the voltages present in lithium cells are higher than that, that is not possible. Organic solvents are instead used. These solvents are flammable and have high vapour pressures. The increased performance of LIB batteries comes with higher severity if something would go wrong, so they have to be handled with extra care.

One of the most common form factors is the cylindrical cell referred to as 18650. That is 18 mm in diameter and 65 mm in length[50, p. 26-27]. Figure 15 displays the parts that build up such a cylindrical cell.

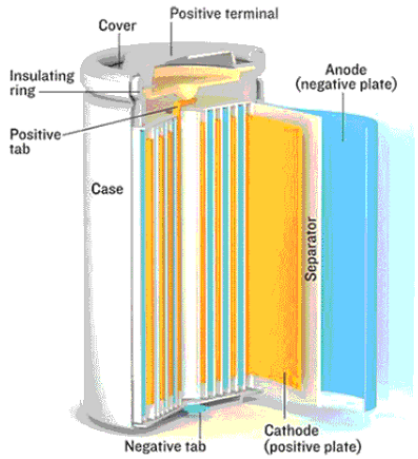


Figure 15: LIB 18650 battery structure.

5.3 Charging

Charging can be either controlled by a separate charger or by the management system itself. Whichever implementation is used, these are the tasks the battery management system must perform[50, p.111]:

- Charging rate
- Life of the batteries
- Charge termination

The preferred way of charging LIB batteries is Constant Current, Constant Voltage (CCCV)-charging. The charge cycle begins with a constant current after the specification of the battery in use. This current is supplied continually until the battery has reached its end-of-charge-voltage. Then charging is switched over to constant voltage mode and the current will taper off as the cell reaches its full capacity[41]. Charging is then terminated when the current goes below a predefined value, Figure 16 displays this process. As the voltage curve is steep when the battery reaches its full capacity precise regulation is needed to prevent overcharging[50, p.111-112].

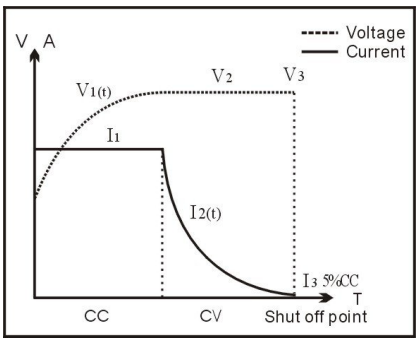


Figure 16: Behaviour of CCCV-charging.

5.4 Balancing

Unlike other types of batteries that have a lower coulombic efficiency, lithium-ion batteries do not self-balance during a charge cycle. Without the ability to transfer charge, capacity will be limited to the battery with the lowest capacity in a series configuration. Knowing the capacity in real-time increases the efficiency of the balancing process. Cell balancing is used for one or more of these purposes[50, p.183-185]:

- Minimize charge difference
- Maximize available battery power
- Maximize available battery energy

Figure 17 Shows the variations that can occur between cells in a series stack.

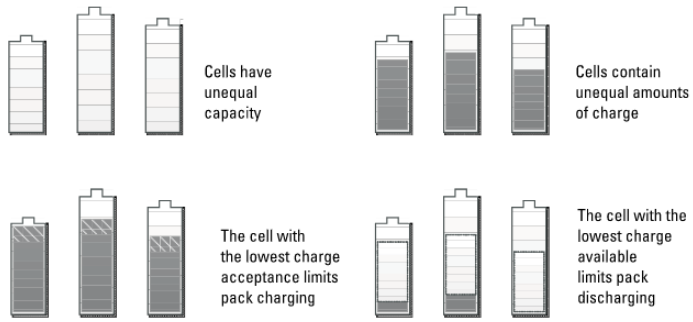


Figure 17: Series stack cell variations.

5.5 18650 Basic Safety

There are both protected and unprotected 18650 batteries. Protected ones include three basic safety features[33]:

- Positive Temperature Coefficient (PTC)
- Current Interrupt Device (CID)
- PCB

A PTC is a resistor that increases resistance when temperature rises, thus protecting against over temperatures. It will automatically reset when the temperature goes down. CID or pressure valve disables the battery permanently if the vapor pressure inside the cell becomes too high. Preventing further swelling and possible fire. The PCB protects against over discharge, overcharge, and overcurrent. Depending on the design, it will be reset when there is no fault condition any more or when it is placed in a charger.

Figure 18 is an illustration of a protected LIB battery. The pressure valve on the left side also breaks the connection between the internal and external positive pole which is why it is called CID or current interrupt device.

Figure 18: A protected 18650 cell.

5.6 Safety and Abuse Conditions

Due to the differences between LIB batteries and other batteries, their higher capacity makes them more susceptible to faults when abused. A list of abuse conditions and their implications follows[50, p.35-38].

- Overcharge occurs when SoC is greater than 100%. Cell voltage will begin to rise fast and may damage load or monitoring circuitry. Irreversible degradation inside the cell will occur. Both single large events and many small events cause degradation. Unlike in other chemistries, small currents causes this. Leading to thermal runaway, cell swelling, and venting. Begins somewhere between 3.75 V and 4.2 V depending on the cell.
- Over-discharge occurs when SoC reaches below 0%, the voltage decreases fast and can even be reversed if discharge current is too high. A reversed voltage may result in damaged monitoring circuitry. Internal damage includes dissolution of the negative terminal foil. Self-over-discharge cannot be prevented. Minimum voltage before over discharge occurs is between 1.8-2.5 V depending on the cell.
- High temperatures increase cell degradation which in turn causes thermal runaway further increasing inside temperatures. Excessive temperatures activate exothermic chemical reactions in the organic solvents. Releasing high amounts of energy that leads to venting of cell contents, fire or explosion.
- Low temperatures can cause plating of metal lithium onto the electrodes which leads to irreversible capacity loss and possibly metallic "dendrite" growth. Dendrites can penetrate the separator leading to internal short circuits. Low temperatures also increase cell impedance thus limiting discharging capabilities. Charging below 0° C is not recommended, although some cells allow temperatures below −10° C.
- Overcurrent, both during dis- and re-charging can cause the same type of problems as with general over- and under-charge conditions, at the same time high currents increases temperature. The maximum current allowed for is a function of SoC and temperature. Even though temperature would be held at acceptable levels, the amount of current will still be limited by the number of ions the negative terminal can receive at any given moment.
- Age is a major factor when it comes to Li-ion batteries, the risk of failure increases with age. Monitoring and estimating remaining battery lifetime can be done and is expressed in State of health (SoH).

5.7 Determining BMS Specifications

A flow chart was used to determine the power requirements of the system powered by the batteries, as is seen in Figure 19. The particular MCU operates at 3.3 V and the sensors that were chosen operates at $3.3V$ and $5.5V$ respectively, so those requirements where definite. There were thoughts about adding a display that might have needed 12 V, therefore the light blue block in Figure 19. As the project continued it was omitted and 12 V would not be needed. Therefore two batteries in series are used. Resulting in a voltage range of 8.4 V fully charged and down to 6 V fully depleted. This voltage is stepped down to 5 V and supplied to the system. The level of current required by the system was unknown, but it was estimated that the capacity of the batteries of 2.2 Ah would be enough to power the system for several hours. In a worst-case scenario, the system would require more than 500 mA which would result in 4 h and 24 min of usage. Safety is of high priority as concluded earlier. So a second flowchart was created to determine the specifications of the monitoring of the two batteries in a series configuration. Figure 20 displays these specifications.

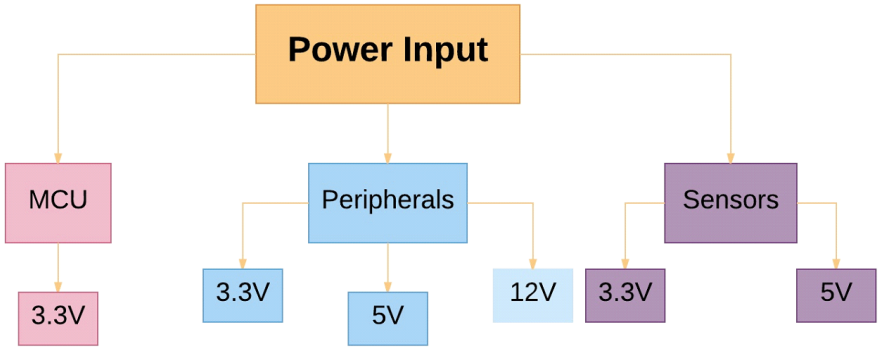


Figure 19: Power input flowchart.

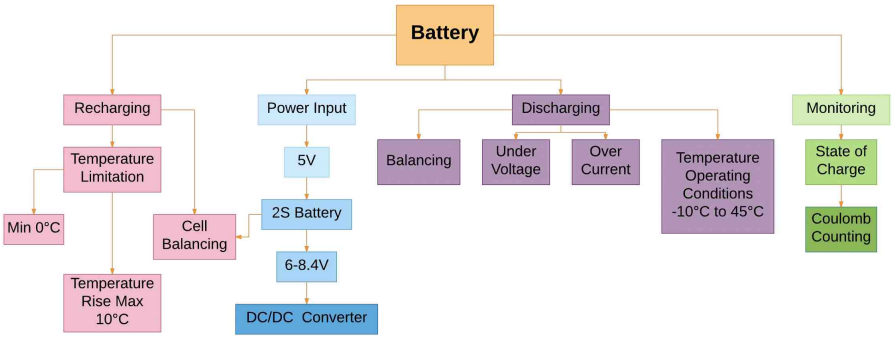


Figure 20: Battery management flowchart.

Three Integrated Circuit (IC)s where chosen for the BMS. The *BQ28Z610* is a gas gauge IC for Li-ion batteries from Texas Instruments Inc. (TI), including

these features:

- Communication of status to host I^2C.
- Balancing of cells, both during charge and use.
- Temperature monitoring and regulation.
- Under and overvoltage protection.
- Overcurrent protection.
- SoC reporting.

The *BQ24133*, also a TI IC, is chosen for charging purposes, making use of the following features:

- 1-3 series cells.
- Temperature monitoring and regulation.
- Power Path, load can safely be connected during charging.

The output voltage is managed by the TI IC *LM2596* step-down converter.

During development, a *BQ28Z610* development board was used together with a *Luxorparts* variable voltage regulator utilizing the *LM2596* and a generic LIB Two cells in series (2S) charger.

The *BQ28Z610*'s software was configured using a windows computer, TI's software *bqStudio*[38] and the *EV2300* Personal Computer (PC) interface board. A Printed Wire Board (PWB) containing the three IC's mentioned above was designed using KiCad[4]. The designs were made according to the typical application of each IC. Typical applications are found in data sheets.

5.8 Current State

Figure 21 displays how the BMS is currently set up. Up left on top of the main PCB is the *BQ28Z610* development board, and below that is the voltage regulator. Charging is enabled via the charging port visible to the right, on the lid.

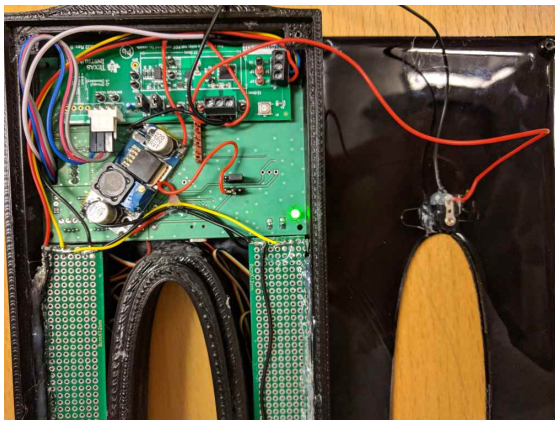


Figure 21: Current setup of the battery management.

Charging is done by turning the device upside down and connecting a 2S LIB charger, See Figure 22.

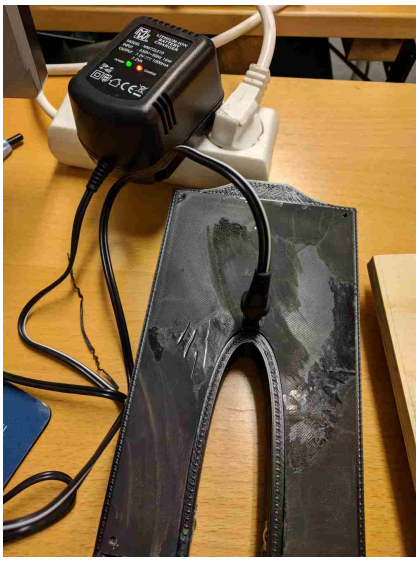


Figure 22: How to charge.

A connection diagram for this configuration can be seen in Figure 23. The charger is connected to the input of the voltage regulator. This configuration has all the features of that implemented on the PWB except the charger is external, charges with 1 A instead of 2 A and there is no power path, meaning that the device has to be turned off to make charging safely.

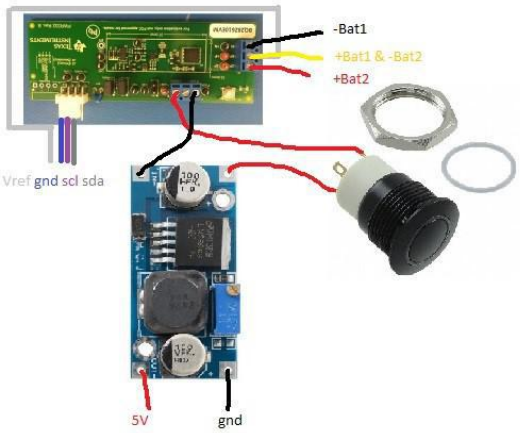


Figure 23: Connection diagram.

Figure 24 shows the PWB that when soldered and tested would fulfill all requirements. It is supposed to be the entire BMS. For detailed view of the schematic, see section A.2.

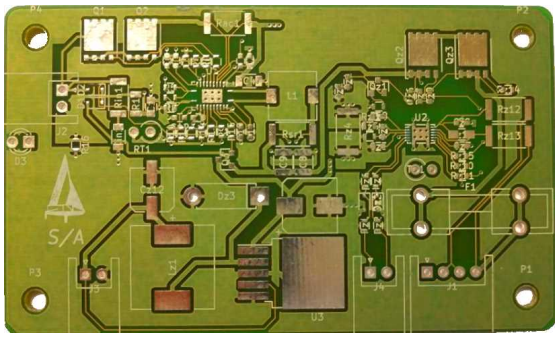


Figure 24: Battery management PWB.

6 Casing

Lundberg, Josef (Sjölund, Johannes)

The *SailorAid* system has some different types of components and sensors, which all need to be housed in a watertight casing for safety and robustness. Some design ideas for this part was to make it easy to mount, small physical footprint, all the electronics in the same enclosure and watertight. The case was revised and worked on over the whole length of the course, redefining and remodeling the construction over time. The cases were constructed with the use of the program *Fusion 360*[1], which is a free to use CAD software. They were then produced by using a 3D printer, which is a fast and easy way of try out different ideas and models.

6.1 First revision

First, a suitable solution was needed, with no thoughts about physical limitations, strength, material or manufacturability.

The idea was modeled around the original mounting plate which is already fastened to the boat, which has the function to hold the board in place and make the dagger-board stable in the boat. The model had the same dimension as the original plate, only the height was different to accommodate the size of the pressure sensors. By making a mounting plate with the same width and length of the original mounting plate the implementation would be neat and space saving. This one had some problems, one of them was that the case was too big for any of the 3D-printers available. Another problem with this instance of the model was that for the circuit boards to fit inside they had to be very small and have some odd shapes.

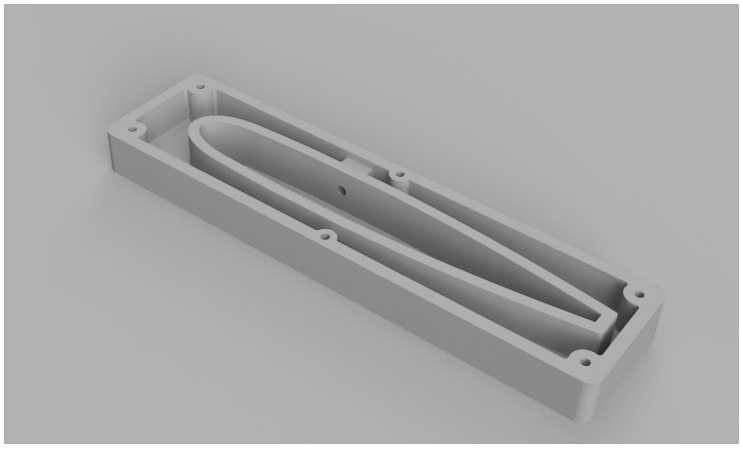


Figure 25: First (1) revision of the case.

6.2 Second revision

The next revision was like the first one, it had the same shape and idea as the first one by having the same size as the original plate, but this time it only

consisted of the front part that holds the sensors and all the electronics. This version could thus be made on a 3D-printer. It still had the problem of having very little space for all the components.

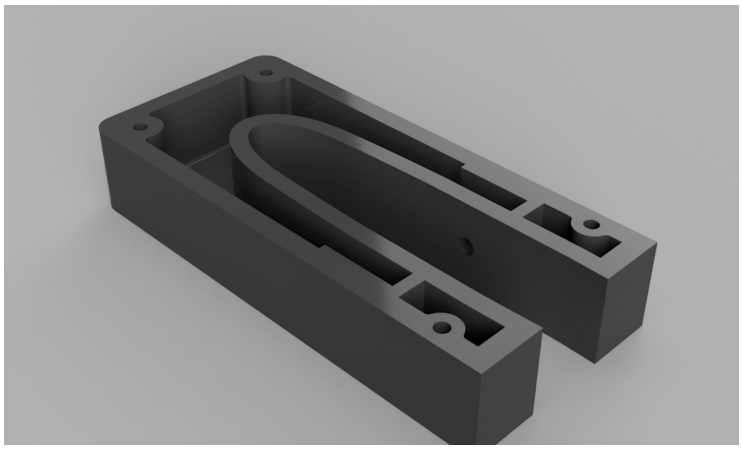


Figure 26: Second (2) revision of the case.

6.3 Third revision

With all the prior knowledge, the next version was modeled. This time it had a much larger casing to accommodate for the PCB. The actual size of the PCB was set by the inner dimensions of this part, see Figure 27.

This case still had some problems, it did not include a holder for the distance sensor nor was it watertight. But this version fulfilled some requirements. The size of the piece was enough to accommodate for the PCB, batteries and all the sensors. The piece was created using a 3D-printer to have some progress to show, check for design faults and to determine improvements for the next version.

Due to the fundamental workings of a 3D-printer the size and dimension on the real produced part differed some from the CAD model. When construction with a 3D-printer the model needs to be compensated for this. One thing that needed to be revised was the implementation of the steel beads.

When building the third version on the 3D-printer the problems with the model could be more easily determined than only looking at a design in a CAD software. The process of improving the design is both about the physical limitations, and the issues due to the manufacturing process.

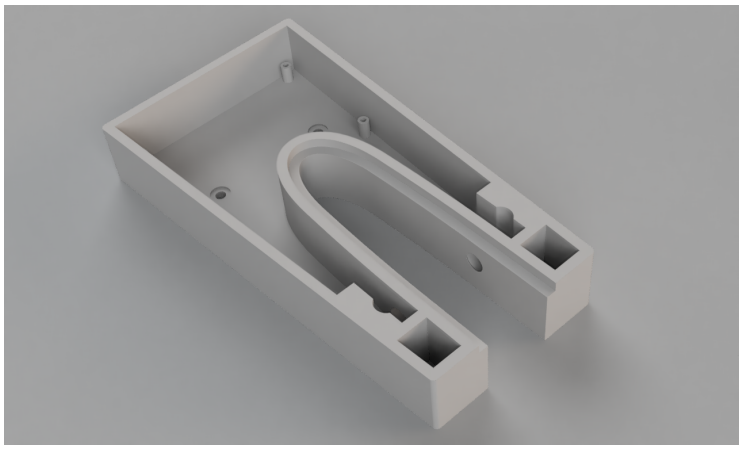


Figure 27: Third (3) revision of the case.

6.4 Fourth and final revision

The fourth revision had all the features and could hold all the electronics. It was designed to house the power switch in the front, the Time of Flight sensor on the top, the load cells, the batteries and both the PCBs in the same enclosure. The access panel was placed at the bottom of the model to get a better water seal. The access panel was a big plate covering most of the bottom of the model. To ensure a watertight case, a seal between the panel and the casing itself could be added.

Due to the material cost and time it takes for the 3D printer to produce models the manufactured part has just some thin layers in the bottom part of the print. To get the some strength in the model epoxy resin was mixed and poured in to both get the stability needed and a clear window for the ToF sensor, see Figure 28 for reference.

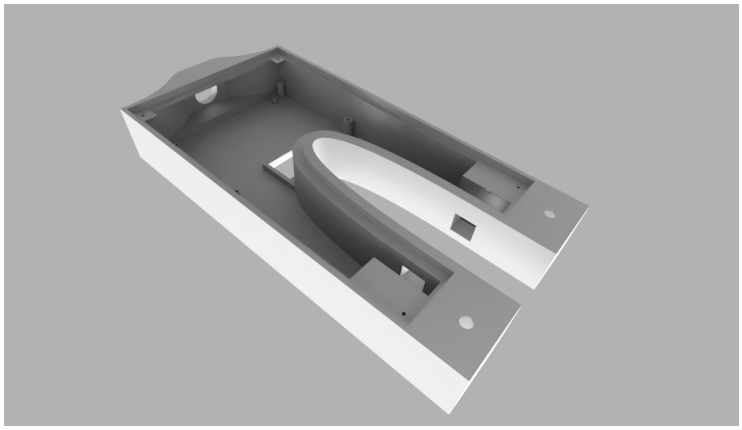


Figure 28: Fourth (4) and final version of the case.

6.5 Results

The final version is the product of many revised and improved steps. The model holds all the electronics and sensors in the same enclosure and takes up as small physical space as possible. The model is not water tight, the only attempt to correct this is the rubber seal between the load cell and the steel beads used to measure the forces and the power button in the front.

7 Software Design

Sjölund, Johannes (Brolin, Daniel)

The software has been divided into two parts, the firmware for the ARM MCU with associated sensors, and an Android application which can display sensor data. These two parts utilize a Bluetooth connection to communicate their current states; for example, when the IMU calculates a new orientation, this data should be processed by the firmware and the resulting calculations sent to the Android application over Bluetooth to be displayed to the user.

7.1 ARM firmware

Sjölund, Johannes (Brolin, Daniel)

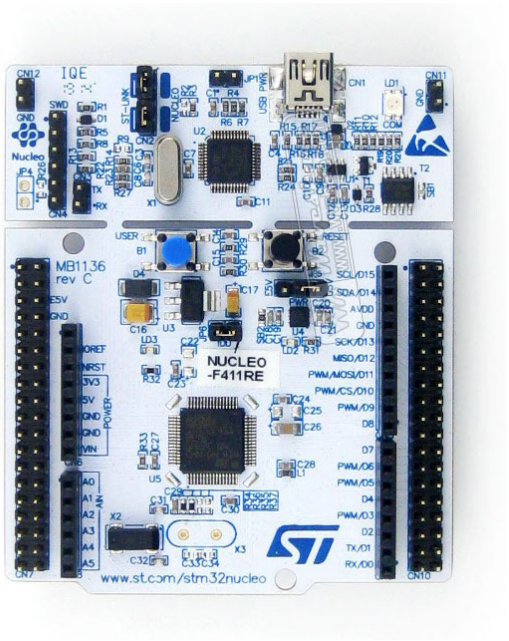


Figure 29: *NUCLEO-F411RE* development board for the ARM Cortex-M microcontroller *STM32F411RE*.

For rapid prototyping and firmware development purposes, the *NUCLEO-F411RE* development board seen in Figure 29 was used. This board contains break-out pins for the ARM Cortex-M microcontroller *STM32F411RE*, an UART to USB bridging circuit and general purpose Light-emitting diode (LED)s and buttons. It is compatible with various Arduino shields[4] as well as expansion boards developed by ST.

In order to speed up firmware development, the *STM32CubeMX* [3] initialization code generator was used to set up a basic working system. This application, developed by ST, can generate C language code for setting up MCU

[4]Shields are boards that can be plugged on top of the development board, extending its capabilities.

clocks, peripherals, interrupts and similar. It is controlled by a graphical interface (section A.3) for setting MCU options and controlling the previously mentioned code generation.

The main challenge in working with this type of code generation is integrating it with external software libraries directly not built for it. If the library interferes with generated code by overriding settings and register values, the software may enter an undefined state and stop working. Care therefore had to be taken to only use the parts of the libraries which did not interfere. Frequent testing of any newly added functionality had to be done in order to find interfering parts.

Three libraries produced by ST were used, one for the Bluetooth module, the IMU chips and the range (ToF) sensor.

7.1.1 Bluetooth

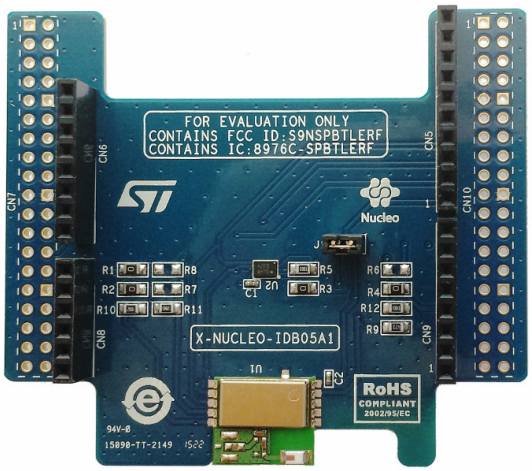


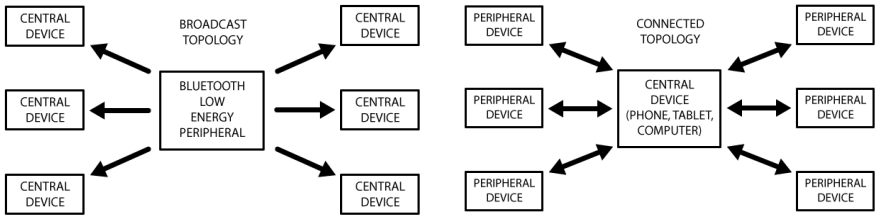
Figure 30: *X-NUCLEO-IDB05A1* Bluetooth Low Energy evaluation board for the *STM32 Nucleo*.

For prototyping, the Bluetooth evaluation board *X-NUCLEO-IDB05A1* [8] seen in Figure 30 was used, which could be stacked on top of the Nucleo board. The pins on the evaluation board connected it to an SPI port on the MCU.

To avoid having to implement the Bluetooth stack from scratch, the firmware package called *X-CUBE-BLE1* [5] developed by ST was used. It consisted of several parts; MCU and Bluetooth evaluation board device definitions such as named pins and ports, functions for manipulating them, a Bluetooth GATT server implementation as well as several demo applications showing usage examples. Additionally an Android demo application for displaying sensor data from Bluetooth was available from the Google Play platform, called BlueNRG[7]. The library code was integrated into the code generated by *STM32CubeMX*, added as an external library and statically linked.

While ST included example code for communicating with the Bluetooth module over SPI through interrupt based Direct Memory Access (DMA) transfer, this code was relatively complex. Eventually blocking SPI communication

were used, since this was simpler to get working. The reasoning was that since the module supported a baud rate of up to 10 Mbit/s, this would be fast enough to cause minimal interference with other parts of the firmware code.



(a) GAP topology (b) GATT topology

Figure 31: Bluetooth topologies.

As mentioned previously, the library implemented the Bluetooth GATT protocol. This protocol supports bidirectional communication from a single central device, in this case an Android cell phone, to several peripheral devices, such as the embedded system in this project. The library also supported the Bluetooth GAP protocol, which is a unidirectional communication protocol allowing one peripheral device to broadcast to multiple central devices. Figure 31 illustrates the topological differences between these protocols.

For this project, the GATT protocol was chosen. The reasoning was that enabling the Android app to send commands to the embedded system could be useful for controlling functionality. This meant that only a single phone could be connected to the system at any time, as opposed to the GAP protocol, which would allow multiple phones to listen to the Bluetooth broadcasts. Since the embedded system is designed to be used on a small dinghy with space for a maximum of two people, this seemed like a reasonable trade-off. If the system was to be used on a larger sail boat, the GAP protocol might be more useful, since it would allow multiple passengers to listen to broadcasted sensor data.

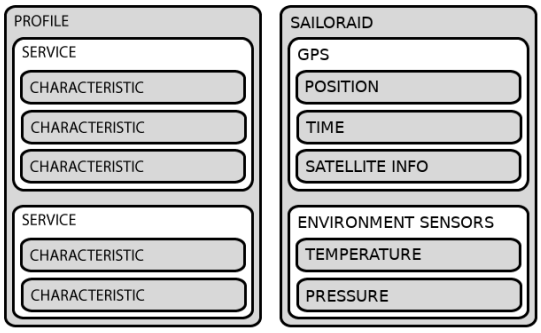


Figure 32: Bluetooth GATT transaction profile with usage example.

The GATT protocol performs transactions by nested structures called Profiles, Services and Characteristics. An example of this structure can be seen in Figure 32. These structures were already implemented in the *X-CUBE-BLE1* and updated by simple function calls. When new sensor data was received from

e.g. the GPS or IMU devices, these functions were called at regular intervals which pushed the data to the Android app. Each profile were given a unique identifier which allowed the app to recognize which type of data was received.

7.1.2 IMU

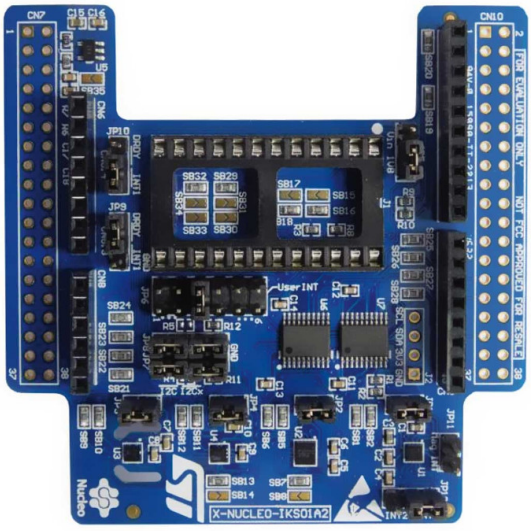


Figure 33: *X-NUCLEO-IKS01A2* motion Microelectromechanical systems (MEMS) and environmental sensor expansion board for the *STM32 Nucleo*.

In order to measure the various time dependent spatial features such as orientation, acceleration and velocity, an IMU device was used. More specifically, the *X-NUCLEO-IKS01A2* [9] evaluation board, see Figure 33, developed by ST was chosen for rapid prototyping purposes. This board included the *LSM6DSL* 3D accelerometer/gyroscope, the *LSM303AGR* 3D accelerometer/magnetometer, the *HTS221* humidity and temperature sensor as well as the *LPS22HB* pressure sensor.

To interface the firmware with the board, the *X-CUBE-MEMS1* [6] library developed by ST was used. This library implemented the I^2C communication protocol used by the previously mentioned IMU devices in the form of simple function calls, which saved a lot of development time. It was quite simple to integrate with the code generation from *STM32CubeMX*, only a few source definitions had to be modified. Like with the Bluetooth library (section 7.1.1) blocking communication was chosen to simplify the code, even though the MCU supported interrupt based DMA transfers. The I^2C operated in fast mode at 400 kHz which was thought to cause minimal interference with the rest of the system in blocking transfer mode.

An important use case for the IMU was to determine the current orientation of the dinghy. To accomplish this, a type of sensor fusion algorithm called Madgwick Attitude and Heading Reference System (AHRS) was used, more about Madgwick in section 7.1.3.

7.1.3 Madgwick AHRS

Madgwick AHRS is a type of sensor fusion algorithm which calculates the current orientation in space from three dimensional vectors of acceleration, angular velocity and magnetic field strength. It was developed in 2010 by Sebastian O.H. Madgwick as a better performing alternative to the Kalman filter approach[10]. It basically integrates the angular velocity from the gyroscope, while using the accelerometer and magnetometer to create a reference to the horizontal plane. Earth's magnetic poles provides a horizontal vector which lies on the plane, while the gravitational acceleration is the plane's normal vector. This is then used by the algorithm to compensate for drift in gyro integration. The algorithm stores orientation in quaternions[5], but can be converted to Euler angles, which is simpler to manipulate.

The mathematical background of this algorithm is quite complicated. See the official report[11] for a detailed explanation about how the algorithm works.

7.1.4 GPS

Figure 34: Maestro *A2235-H* GPS module with built-in antenna.

The GPS module used in this project, *A2235-H* by Maestro [12] could communicate with the MCU through either I^2C or UART. Both protocols require only two pins to operate, but UART communication was determined to be easier to implement in code. The UART baud rate of the GPS module was set to 4800 Hz by default. While this could be changed by software there was no reason to do so. The low baud rate did however mean that blocking transmissions might cause problematic interruptions in the firmware code. To prevent this, interrupt based communication through DMA was implemented, using the same type of queuing system as the USB UART, see section 7.1.6.

Data from the GPS was formatted according to the National Marine Electronics Association standard (NMEA). It is used by nearly all GPS devices internally, but is quite hard for a human to read. For example,

```
GPGSA,A,3,03,22,31,23,01,06,09,11,,,,,1.9,1.2,1.5*33
GPRMC,152053.000,A,6537.0389,N,02208.0160,E,0.17,264.54,240917,,,A*6A
GPGGA,152054.000,6537.0389,N,02208.0160,E,1,08,1.2,14.0,M,25.0,M,,0000*68
```

contains three so called sentences. *GPGSA* contains data about the number of active satellites and positional accuracy. *GPRMC* and *GPGGA* both encode longitude, latitude, current time and date, as well as other data.

[5] rotation vectors with four elements

Several NMEA parsing libraries are freely available on the web. The one chosen for this project was called Libnmea[13] and allowed the sentences to be automatically recognized, parsed and stored into easy to use C[6] structures.

7.1.5 Range sensor

In order to communicate with the *VL53L0X* range finder sensor, another software library called *X-CUBE-53L0A1*[14] developed by ST was used. This software was written for another Nucleo expansion board called *X-NUCLEO-53L0A1* which integrated three range sensor units to perform gesture recognition. It was however possible to modify the code to work with the single sensor used in this project. The library implemented a leaky integrator algorithm (basically a low-pass filter) to reduce measurement noise and improve accuracy.

Since the sensor measures the time-of-flight of a laser beam, it was obvious that having the firmware code block while waiting for measurement data was not a realistic option. Instead, the code repeatedly alternated between instructing the sensor to initiate a measurement, and reading back the data from the last measurement. Since the module was meant only to measure the height of the dinghy dagger-board a low measurement frequency of 1 Hz was used.

7.1.6 USB UART

Since this project involved analyzing sensor data for developing sensor fusion algorithms, for example combining GPS and accelerometer for accurate positioning and measuring water wave properties, it was important to be able to log data at a reasonably high frequency; for more about sensor fusion, see section 8. Transferring serial commands between a computer and the MCU also helped in debugging the code. To this end, a hardware UART-over-USB chip was used, the *ST-LINK/V2-1* on the Nucleo board, and *FTDI FT232R* on the custom project board.

It was determined that send and receive should both be interrupt based using DMA transfers to minimize the impact on system resources. Reception of data like key presses from a computer was handled one character at a time. The characters were appended as a string until the enter key was detected. At this point the string was matched against a list of valid commands, and the appropriate task performed, such as sending current sensor values. Sending was implemented as a simple circular buffer which could be transferred to the UART peripheral registers using DMA.

To connect to the USB-UART, a baud rate of 230400 npb should be used, with eight data bits, no parity and one stop bit. Commands can be entered with the the keyboard and executed by pressing the enter key. Valid commands and outputs are listed in Table 2. The rate at which the serial outputs sensor values can be adjusted in the main firmware source file.

[6] C programming language

Command	Serial output
`imu`	Orientation in euler angles
`gpsraw`	Raw NMEA GPS coding
`gps`	Formatted GPS data from the NMEA coding
`adc`	Load cell output
`env`	Environment humidity, pressure and temperature
`range`	Range sensor measurement in centimeters
`matlab`	Complete sensor state in binary format

Table 2: Serial connection commands and outputs.

In order to log sensor values for later analysis a simple Matrix Laboratory (MATLAB) script was developed for listening to sensor data over the UART serial port. By inputting a serial command, the embedded system starts sending live sensor data at at constant rate. The MATLAB script listens to this and logs it to a table structure which can be used for analysis of sensor data.

7.1.7 Firmware design pattern

In order to meet the design requirements, information such as dinghy orientation, position, velocity and center board height had to be measured continually in order to be useful. It followed that the most important task of the firmware was to measure these sensor data through external peripherals, perform parsing and calculations on them, then report them to the Android app over Bluetooth in a timely fashion.

Certain measurements were required to be performed more frequently than others. For example, the Madgwick AHRS algorithm required frequent measurements at a fixed time interval of the acceleration, angular velocity and magnetic strength in order to work as intended.

The center board height required regular but not very frequent measurements, as did the load cells through Analog-to-digital converter (ADC), while the GPS unit passively reported the position and velocity of the receiver automatically after being initialized.

In order to save battery power, it was decided that not all sensor data needed to be reported through Bluetooth at the same rate they were measured. Similarly, the USB debug output was rate limited. Table 3 shows the system sample and output rates of the sensor data from different peripheral devices. The main firmware program loop keeps track of the timing for these tasks and performs device polling and interrupt based requests based on these rates.

Device	Sample rate	BT output rate	USB output rate
IMU	100	30	60
Load cell	10	5	10
Range sensor	10	5	10
GPS	≈ 1	≈ 1	≈ 1

Table 3: System input/output rates in Hz. BT is short for Bluetooth and refers to the rates at which GATT transactions are pushed to the Android app. USB refers to the rate at which serial UART values are output to the terminal.

A hardware timer with microsecond resolution was used to calculate the timing of these device communications. For the devices which used polling, a timeout of a few milliseconds was used in case of communication failure on the I^2C and SPI data buses. Problems such as noise or framing errors and were handled by discarding any buffered data, clearing hardware error status bits and re-initializing devices if necessary.

7.2 Android application

THEOLIN, HENRIK (AXELSSON, OSKAR)

The main reason for this project is to give a sailor qualitative feedback and help in clearing the mind of the techniques required to achieve a smooth sail experience so that the sailor can focus on the joy of sailing. A crucial part is to display the data in a manner that is easy to interpret and provide the help that the skipper needs.

To further increase the flexibility of the design, several different user layouts are implemented, that the user can switch between while running the application. This was determined to be a good way of increasing the chances that the user would find a preferable layout.

It was also determined that not only visual representation was enough, since the sailor needs to be in constant motion to counteract the forces applied on the ship by the wind and currents. This will make watching a screen to retrieve information somewhat difficult. Other ways of representing data were implemented using text-to-speech where the sailor will get important information by sound as well as text Vibration is also implemented with different vibration sequences depending on different states of the ship.

7.2.1 Software design

A Unified Modeling Language (UML)[25] model of the system, see Figure 35, was developed for an overview of the system implementation. The android system uses activities[24] to display content to the device screen. These activities have a life-cycle (Figure 36) that determine how data is accessed and displayed. What should be understood is that the application starts in a main activity, and switching to another activity is done by sending an intent that spawns as a child activity. While the application is in the child activity the main activity is paused but the state is stored and when switching back from the child all data from the previous state is being accessed. When the user returns from the child activity, that activity is destroyed and all data is erased. Sending data between activities is done using intents, to send an intent to a child activity is done by adding a bundle with a data object along with the intent to spawn the child activity. Sending data back to the child activity is done by calling the method "*startActivityForResult*", this allows the child to send a data packet as a result back to the parent.

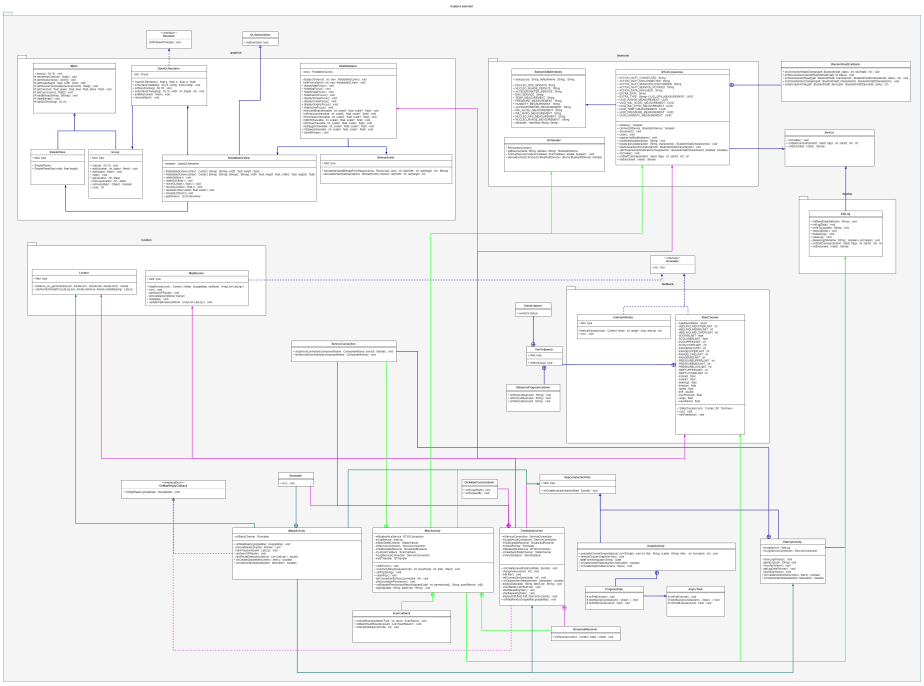


Figure 35: UML model.

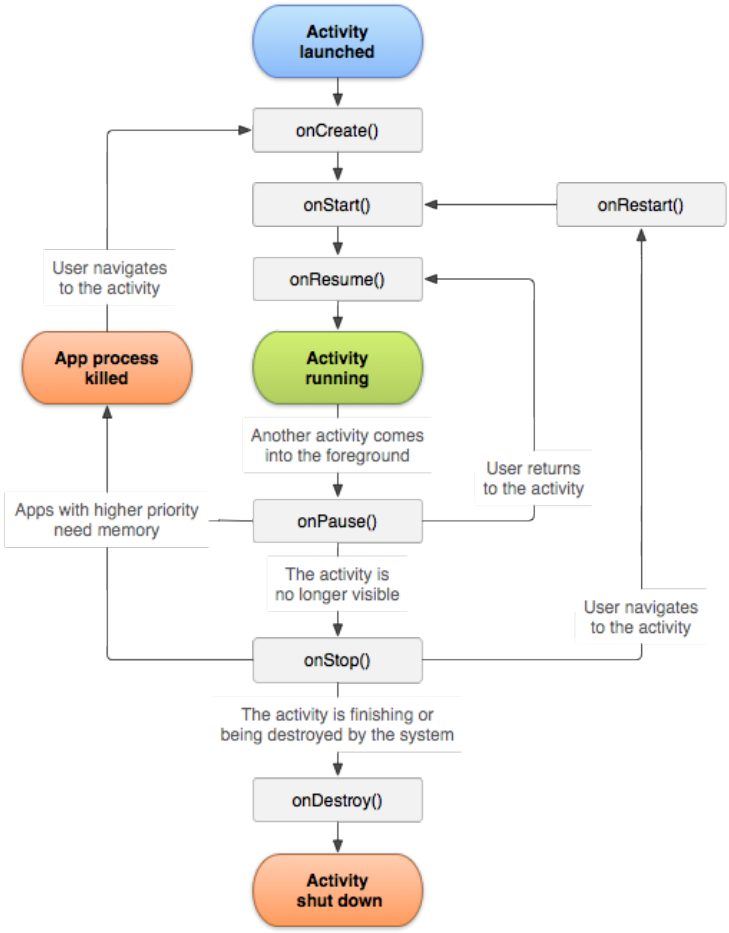


Figure 36: Activity lifecycle.

The Bluetooth connection and data logging class is implemented as a services[15]. A service can be started by an activity and continue running until it is explicitly called to stop. This allows several activities to share resources and perform long-running operations in the background. The lifecycle of a service is seen in Figure 37.

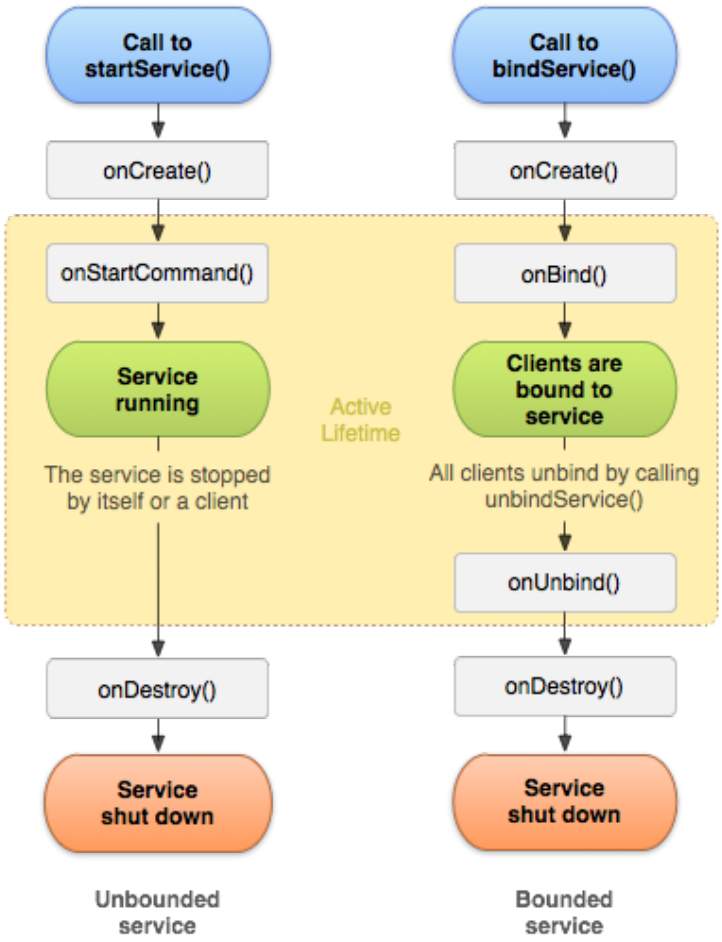


Figure 37: Activity lifecycle.

7.2.2 Feedback

For the feedback states in the application, there are some approximations, and simplifications made for easier implementation. As explained in section 2, the chapter about Physics of sailing, there are certain parameters that can be evaluated in different states of the dinghy. An extensive approximation is that there is no water movement in the implementations. The states implemented are

Clear - The dinghy holds good velocity and not heeling or drifting and with a good amount of pressure on the dagger-board

Drift - The dinghy is drifting perpendicular to the bearing while the dagger-board is in an upper position

Heel - The dinghy has high heel angle

Reefing - The dinghy has high heel angle and the pressure on the dagger-board is high

Wrspeed - The dinghys speed is above 65.45 kn

Lrspeed - The dinghys speed is above 16.8 kn

Hike - The dinghy has more then moderate heel angle and the pressure on the dagger-board is between medium and high

Keelhaul - The dinghy has an above moderate heel angle and the dagger-board is not in the lowest position

Runninghigh - The dinghy is sailing directly windwards with dagger-board high and low heel angle.

Runninglow - The dinghy is sailing directly windwards with dagger-board high and mid heel angle.

Landcrab - The dinghys speed is low.

Given these states, appropriate feedback can then be provided to the user. For handling states, a class called *StateChecker* was implemented. This stores sensor values and check against limits defined at an interval that is also defined in the class. Depending on these states this class also determines what feedback that should be given.

7.2.3 Visual Feedback

It was determined that the visual feedback provided to the user was to include very little text information and would consist mainly of figures that changed position based on sensor data to give a good representation of what was going on with the dinghy. Because of different personal preferences, the ability to switch between different layouts is implemented. This is done by a simple swipe on the screen to toggle the view to the next layout. All layouts consists of a subset of views from the complete set including

38 **Incline** displays a ships relative incline against and artificial horizon.

39 **Pressure** moves a pin along a colored bar to represent high of low pressure applied on the dagger-board.

40 **Bearing** rotates a compass to show the ship relative bearing against true north.

41 **Map** displays current location of the ship.

42 **Speed** displays a speedometer from a classical Swedish vehicle[22] with a movable bar representing speed over ground.

43 **Drift** is represented with a colored bar containing two arrows that moves to the relative drift direction to show the sailor if the ship was holding its set navigational reference.

44 **Feedback** a text that changes values based on the current state of the boat.

45 **Height** of the dagger-board is represented with a visual dagger-board moving up and down along a graphical ruler.

46 **Wave frequency** displays a wave moving towards a ship at a speed representing different period of the waves.

Figure 38: Incline feedback view

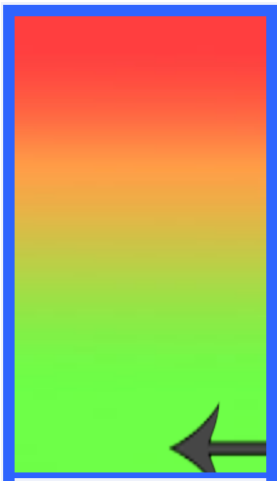


Figure 39: Pressure feedback view.

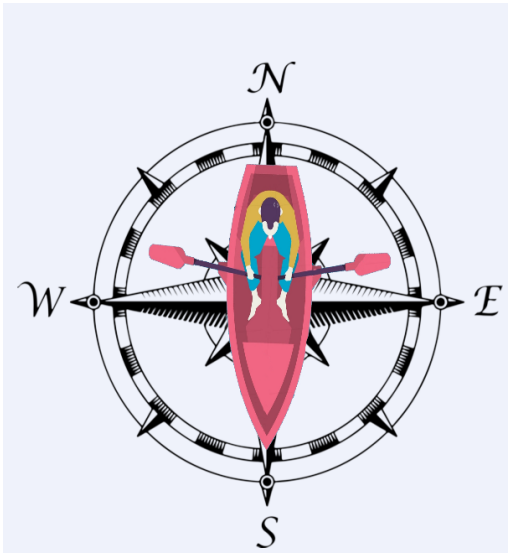


Figure 40: Bearing feedback view.

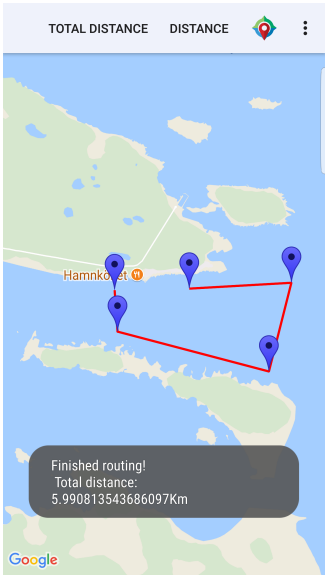


Figure 41: Map feedback view.

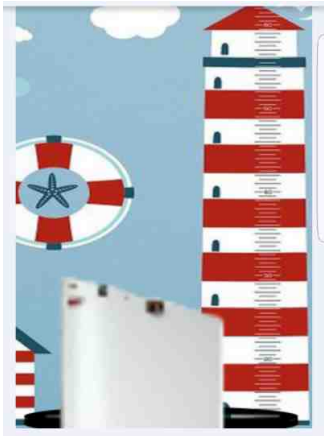


Figure 42: Dagger-board height view.

Figure 43: Drift feedback view.

Figure 44: Text feedback view.

Figure 45: Speed over ground view.

Figure 46: Wave period feedback view.

The figures were chosen at a development stage and improvements can easily be made by adding different Portable Network Graphics (PNG)[23] files in android studio. They are implemented as *GLSurfaceView*[26] so that updating the figures is be done by calling a *requestRender* function, this improves the performance of the application when only updates to the figures are done when

new sensor readings are received. To make the figure fit the application the *GLSurfaceView* is extended into a *RotatableGLView* that has the wanted features for rotation and positioning. Using these figures four different layouts are developed to give multiple choices for the user and is seen in figure 47. Switching between these layouts a *swipeTouchListener* is implemented which allows the user to swipe across the screen to change the layout. Changing view the figures needs to be redrawn onto a new view matching the current layout, this is handled in the *ViewDisplayer* class that contained all *RotatableGLViews*.

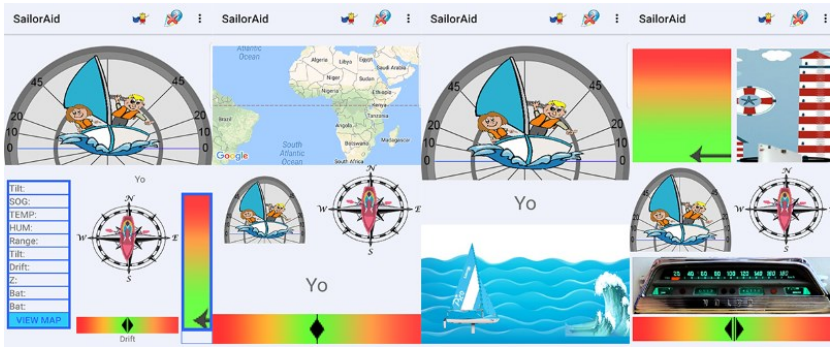


Figure 47: Different layouts

7.2.4 Audio feedback

Based on values from the sensors, different states are implemented and for each state, a text is read out to the user. The feedback is implemented such that the audio feedback would continue even if the device is put into sleep mode. Handling the text to speech feature a *Text-to-Speech*[27] class is implemented as an inner class in *StateChecker*. To prevent the speech to be interrupted preemptively an *UtteranceProgressListener*[28] is used.

7.2.5 Haptic feedback

Based on the same states as the audio feedback vibration is also implemented. The frequency and length of the vibrations are implemented in such a way each state has a unique signature that can be recognized by the sailor after some practice with the system. Handling the interval an *IntervalVibrator* is implemented and run by the *StateChecker*.

7.2.6 Log

The ability to analyze the sailing trip was determined to be an important feature for the user, so a logging function is implemented so data can be stored in the device internal storage. This is handled by the *SailLog* class and these files can be read or deleted at a later stage, by the user. Storing a log is implemented such that the sailor would need to start a logging session after Bluetooth connection has been established to the ship. After the log is started it creates a file on the device internal storage and writes sensor data to the file at the same frequency as the data is transmitted by the system. While logging is active the device would continue to store information even if the device is put in sleep mode so that the

sailor could choose to only log data and not view the information displayed on the screen. After the sailing trip is finished the sailor can read the saved log file and receive a summary of the trip (Figure 48). More information from the log can be analyzed by reading graphs (Figure 49) where the sensor data is shown with respect to time. These graphs are implemented such that the user is able to choose what graphs to be displayed on the device for improved comparability.

Avg Incline: -0.32980242
Max Incline: 36.411453
Max Drift: 1.4229604
Total Drift: 103.87611
Avg SOG: 3.5639582
Top SOG: 4.92632
Max pressure: 1032.4
Avg Pressure: 1032.2876

Figure 48: Log data summary example.

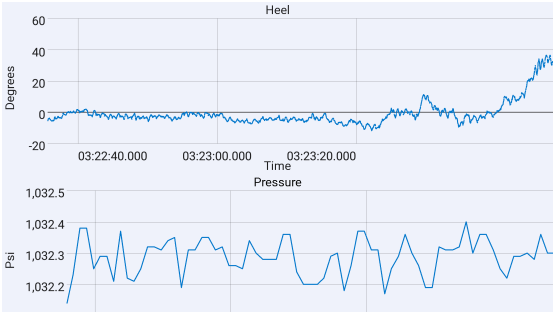


Figure 49: Log graphs examples.

7.2.7 Bluethooth connection

A list of all devices in the nearby area is displayed when the user decides to connect to the system, this adds flexibility to the user and allows for connection between multiple systems with different Media access control address (MAC). Scanning for nearby devices is done in the *MainAcivity* and handled by the *BTHandler* class. The application uses Bluetooth low energy technology to match the system implementation. This allows the device to receive notifications when the data is altered on the system and updates occurs in different frequencies for different types of data. To ensure a stable connection between the system and the device the connection class is implemented as an android service[15]. This allows the device to keep the connection alive between different views and even when the device is put into sleep mode. All functionality regarding connecting and receiving data is implemented in the *BTLEConnection* class and known characteristics and the determined unique GATT service identifiers are stored in the *SampleGattAttributes* class.

7.2.8 Map

For the sailor to get more information about location two maps are implemented, one that is embedded into a layout in the *FeedbackActivity* and the functionality of this is handled in the *MapRunner* class, the other map is implemented in *MapsActivity*. This could be improved by using the same class for both maps to achieve better readability and modifiability of the code. Both these implementations utilized the Google maps Application Programming Interface (API)[29], which can be used freely and the map service is highly accurate and suits our system well. Interaction with the maps can also be made quite easy which helped to make development faster. The sailor has the ability to see a path over the trip and the total distance traveled. Waypoints can be placed if

the sailor wants to plan a certain trip in advance. The distance of the trip and the distance currently traveled is displayed to give the skipper information on how long the trip is and how much of the trip is left to be sailed. A path to the nearest waypoint is shown to help navigation. When the sailor is close to the first waypoint the path to the next waypoint is shown. The waypoint route can also be viewed in the map of the *FeedbackActivity* layout.

7.2.9 Drift

Calculating leeward drift is done by measuring the distance from an estimated destination point and the position received from the GPS. By using the GPS position and the ships bearing from the systems magnetometer a *rhumb line*[17] is derived and the new estimated position is calculated with

$$\begin{aligned}
&\delta = d/R \\
&\varphi_2 = \varphi_1 + \delta \cdot \cos\theta \\
&\Delta\psi = \ln\left(\tan\frac{n}{4} + \frac{\varphi_2}{2} \Big/ \tan\frac{n}{4} + \frac{\varphi_1}{2}\right) \\
&q = \frac{\Delta\varphi}{\Delta\varphi} \\
&\Delta\lambda = \delta \cdot \sin\frac{\theta}{q} \\
&\lambda_2 = \Delta\lambda_1 + \Delta\lambda
\end{aligned}$$

where R is the Earth's radius, d is the distance traveled, θ is ships bearing, λ_1 and φ_1 is the point of origin in longitude and latitude. This new estimated position λ_2 and φ_2 is then compared to the next positional data from the GPS and the distance between these points is calculated using *Equirectangular projection*[18].

$$\begin{aligned}
x &= \Delta\lambda \cdot \cos\frac{\Delta\varphi}{2} \\
y &= \Delta\varphi \\
d &= R \cdot \sqrt{x^2 + y^2}
\end{aligned}$$

where $\Delta\lambda$ and $\Delta\varphi$ are the differences in longitude and latitude for two location points. This function has high performance gain but is less accurate over large distances then, e.g., the *Haversine formula*[19]. Since for this function, only small changes in distance are calculated the accuracy is more than adequate. A problem with this way of calculating leeward drift is the accuracy of the GPS readings and to get a good bearing the idea was to make use of the magnetometer though the accuracy of that is also far from perfect. This combined with the fact that a small deviation of a couple of degrees results in a large drift resulted in large errors in leeway drift. The solution was to use the directional data sent from the GPS module which calculates direction based on last positional data. This solution works well for progress in the forward direction and when fast sideways motion is detected the estimated position will be along the same vector

as the previous direction. However, for constant drift sideways this approach will display that no drift was taking place. This calculation is handled by the *Locator* class.

7.3 Results

Theolin, Henrik (Axelsson, Oskar)

As mentioned the system consists of two parts, the ARM MCU firmware and the Android application. Reading the sensor data in the MCU is done with different frequencies depending on the type of sensor. This data is computed so that is easily interpreted by the Android application.

Communication is done by the Bluetooth GATT protocol which allows the data to be sent with unique service ids to identify what type of data is sent, this also makes it possible for the data to be sent at different frequencies matching the sensor readings. The Android application is designed for simple usage and can only be connected to a device named *SailorAid*.

For more flexible usage the feedback from the sensor reading can be acquired in multiple ways. The feedback can be received visually, by audio and by vibrations, it is also possible to close the screen and still be able to receive feedback. The logging of data is done at the same frequency as it is received by the Bluetooth protocol.

The resulting system is created for an easy to use experience that would effortlessly be turned on and connected.

8 Inertial Navigation System and Kalman Filter

Axelsson, Oskar (Theolin, Henrik)

8.1 Sensor theory

Sensor fusion can be observed everywhere e.g., living animals uses all of its senses to survive daily, an animal cannot hunt using its eyes only, it has to combine its sense of smell, eyes, and hearing to hunt the prey[47].

Sensor fusion theory is found not only in the living species but also in cars, planes, computers and so on; this to enhance performance and accuracy. In this project, a sensor fusion will be designed to enhance the accuracy of the dinghy's position and velocity. The fusion will be a global positioning system, GPS and an Inertial Navigation System (INS). The INS uses a low priced IMU.

The GPS's accuracy is not uniform since there might be building reflections, atmospherics delays or clock bias errors[43]. Using the only information provided by a Strapdown Inertial Navigation System (SINS) is not sufficient either since the IMU sensor will drift after time, but using the information provided by the INS for short time intervals will give more accurate results.

8.2 Inertial Navigation System

In navigation there exist different techniques on how to navigate using an IMU, a typical technique is using a SINS. A SINS consists of an IMU which is mounted on the dinghy so it measures the acceleration and rotational rate that the dinghy encounters. The concepts of inertial navigation are to determine the position and velocity of the dinghy from a known starting point, using only measurements from the IMU. The IMU, in this case, consists of a three-axis gyroscope, a three-axis accelerometer and a three-axis magnetometer.

Measurement from the gyroscope is to determine the angular motion of the dinghy, from that its heading relative to a reference frame can be derived. By measuring specific forces acting on the dinghy using the accelerometer, then resolve the specific force measurements into the reference frame using the knowledge derived from the information provided by the gyroscope. The resolved specific force measurements are integrated to obtain its velocity and position[46].

8.3 Sensor fusion

A popular filter to use when applying sensor fusion is to use a Kalman filter. The Kalman Filter is a recursive filtering method for discrete data, the algorithm was developed by a Hungarian mathematician *Rudolf (Rudi) Emil Kalman* in 1960, its popular to use due to its efficiency when calculating predictions[40].

8.4 Navigation Frames

Navigation algorithm involves various coordinate frames and therefore transformation between frames is a must. In this case, four different frames is used.

The Inertial frame denoted i-frame for future notation is defined such that its origin is at the center of Earth and its axes X_i, Y_i and Z_i is non-rotating with respect to the stars. Z_i is coincident with the Earth's polar axis, i.e., North.

The Earth navigation frame denoted e-frame for future notation is fixed with respect to Earth and has its origin at the center of the Earth. The frame is defined as X_e, Y_e and Z_e, with Z_e along Earth's polar axis. Axis X_e and Y_e lies along the intersection of the plane of the Greenwich meridian with the Earth's equatorial plane. The e-frame rotates at a constant rate with respect to the i-frame and is denoted ω_e.

The Navigation frame denoted n-frame for future notation is a local frame and has its origin located in the navigation system, in this case point P, see Figure 50, and its axes aligned with the directions of north, east and down, denoted NED. The turn rate of the navigation frame with respect to Earth's fixed frame, ω_{en}^{n}, is directed by the motion of point P with respect to the Earth and is referred to the transport rate.

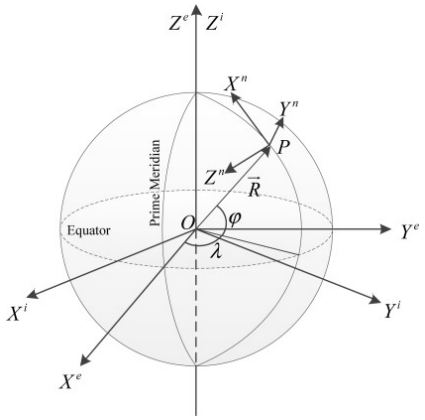


Figure 50: Inertial, Earth and Navigation frame.

The Body frame denoted b-frame is the sensitive axes of the IMU's sensors, which are made to coincide with the axes of the moving body in which the IMUs are mounted on. The body, in this case, is referred to the dinghy, see Figure 52.

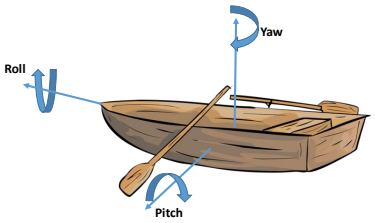


Figure 51: Body frame.

- Yaw (ψ): Is an imaginary line running vertically through the ship and through its center of gravity. A yaw motion is a side-to-side movement of the bow and stern of the dinghy [44].
- Pitch (θ): Is an imaginary line running horizontally across the ship and through the center of gravity. A pitch motion is an up-or-down movement of the bow and stern of the dinghy [44].

- Roll (ϕ): Is an imaginary line running horizontally through the length of the ship, through its center of gravity, and parallel to the waterline. A roll motion is a side-to-side or port-starboard tilting motion of the superstructure around this axis[44].

8.5 Transformation Between Frames

Referring to Figure 50 it can be observed that its possible to align the n-frame with the e-frame, this is done by rotating the n-frame by $(\lambda-90)$-degrees around its X-axis (east-direction) and $(-\phi-90)$-degrees about its Z-axis, (downward direction)[45]. Where λ and φ is the latitude and longitude, respectively. Then the transformation between the two frames can be done using the Direction Cosine Matrix (DCM)[45], which is defined as

$$C_n^e = R_z(-\lambda - 90)R_x(\varphi - 90) \tag{2}$$

Where C_n^e should be interpreted as moving from n-frame to e-frame, R_x and R_z are the rotation matrices around its axis, respectively. Expanding equation (2)

$$C_n^e = \begin{bmatrix} cos(-\lambda-90) & sin(-\lambda-90) & 0 \\ -sin(-\lambda-90) & cos(-\lambda-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi-90) & sin(\varphi-90) \\ 0 & -sin(\varphi-90) & con(\varphi-90) \end{bmatrix} \tag{3}$$

$$C_n^e = \begin{bmatrix} -sin(\lambda) & -cos(\lambda) & 0 \\ cos(\lambda) & -sin(\lambda) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & sin(\varphi) & -cos(\varphi) \\ 0 & sin(\varphi) & sin(\varphi) \end{bmatrix} \tag{4}$$

$$C_n^e = \begin{bmatrix} -sin(\lambda) & -sin(\varphi)cos(\lambda) & cos(\varphi)cos(\lambda) \\ cos(\lambda) & -sin(\varphi)sin(\lambda) & cos(\varphi)sin(\lambda) \\ 0 & cos(\varphi) & sin(\varphi) \end{bmatrix} \tag{5}$$

Exploring the orthogonality its possible to transform from e-frame to n-frame by taking the inverse of the equation above, i.e.

$$(C_n^e)^{-1} = C_e^n \tag{6}$$

Using equation (6) its now possible to move from e-frame to n-frame.

Transforming from n-frame to b-frame is done in the same way, i.e. using rotation matrices. The DCM, moving from b-frame to n-frame is given by

$$C_b^n = R_z(-\psi)R_y(-\theta)R_x(-\phi), \tag{7}$$

where

$$R_z(-\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$R_y(-\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{9}$$

$$R_x(-\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{10}$$

Solving equation (7) with (8), (9) and (10)

$$C_b^n = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{11}$$

$$C_b^n = \begin{bmatrix} cos(\theta)cos(\psi) & -cos(\phi)sin(\psi) + sin(\phi)sin(\theta)cos(\psi) & sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi) \\ cos(\theta)sin(\psi) & cos(\phi)cos(\psi) + sin(\phi)sin(\theta)sin(\psi) & -sin(\phi)cos(\psi) + cos(\phi)sin(\theta)sin(\psi) \\ -sin(\theta) & sin(\phi)cos(\theta) & cos(\phi)cos(\theta) \end{bmatrix} \tag{12, 13}$$

Where ψ, θ and ϕ are the Euler angles as described earlier. Expressing the matrix above in a more compact form.

$$C_b^n = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \tag{14}$$

The Euler angles is a transformation from one coordinate frame to another and is defined by three successive rotations about the different axis. Its possible to see the three angles as a set of mechanical gimbals. The problem with Euler angles is that when rotating about each axis, its possible to drive two of the three gimbals so they appear parallel to each other this implies that a degree of freedom is lost, this is called a gimbal-lock[45]. To resolve this issue its possible to use quaternion attitude representation instead, as Euler angels the representation allows transformation between one coordinate frame to another, but the difference is that the transformation is done in a single rotation, instead of three about a vector defined in the reference frame. The DCM using quaternion is defined such that

$$C_b^n = \begin{bmatrix} (q_1^2 + q_2^2 - q_3^2 - q^4) & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 - q_1q_4) & (q_1^2 - q_2^2 + q_3^2 - q^4) & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & (q_1^2 - q_2^2 - q_3^2 + q^4) \end{bmatrix} \tag{15}$$

Where q_1, q_2, q_3 and q_4 can be derived using the Euler angles from equation (14).

$$\begin{aligned} q_1 &= \frac{1}{2}(1 + c_{11} + c_{22} + c_{33})^{0.5} \\ q_2 &= \frac{1}{4q_1}(c_{32} - c_{23}) \\ q_3 &= \frac{1}{4q_1}(c_{13} - c_{31}) \\ q_4 &= \frac{1}{4q_1}(c_{21} - c_{12}) \end{aligned}$$

The angular velocity of the e-frame with respect to the i-frame projected onto the e-frame is given as

$$\bar{\omega}_{ie}^{e} = \begin{bmatrix} 0 & 0 & \omega_e \end{bmatrix}^T \tag{16}$$

Where ω_e is the angular velocity of the Earth and has a value of 7.2921158×10^{-5} rad/s[45]. Using the projection matrix equation (2) its possible to project $\bar{\omega}_{ie}^{i}$ onto the n-frame

$$\bar{\omega}_{ie}^{n} = C_e^n \bar{\omega}_{ie}^{e} = \begin{bmatrix} \omega_e cos(\varphi) & 0 & -\omega_e sin(\varphi) \end{bmatrix}^T \tag{17}$$

$\bar{\omega}_{en}^{n}$ represents the turn rate of the n-frame with respect to the e-frame its called the transport rate and may be expressed as the rate of change of latitude and longitude as follows

$$\bar{\omega}_{en}^{n} = \begin{bmatrix} \dot{\lambda} cos(\varphi) & -\dot{\varphi} & -\dot{\lambda} sin(\varphi) \end{bmatrix}^T \tag{18}$$

Where $\dot{\lambda} = v_e/(R_N + h)cos(\varphi)$ and $\dot{\varphi} = v_n/(R_E + h)$, here we assume the Earth to be an ellipsoid and that there are no variations in Earth's gravitation depending on where the user is on the ellipsoid. R_N is the meridian radius of curvature and defined as $R_N = R_0(1 - e^2)/(1 - e^2 sin^2\varphi)^{3/2}$[45]. R_E is the transverse radius of curvature and defined as $R_E = R_0/(1 - e^2 sin^2\varphi)^{1/2}$. Where R_0 is the length of the semi-major axis and has a constant value of 6356752.3142 m, e is the major eccentricity of the ellipsoid and has a constant value of 0.081819, φ is the current latitude in radians, v_N and v_E is the velocity in north and east direction, respectively and h is the height above the surface of the Earth. Then Rewriting equation (18) with the new expressions

$$\bar{\omega}_{en}^{n} = \begin{bmatrix} v_e/(R_E + h) & -v_n/(R_N + h) & -v_e tan(\varphi)/(R_N + h) \end{bmatrix}^T \tag{19}$$

Now the turn rate of the n-frame with respect to i-frame, $\bar{\omega}_{in}^{n}$ can be obtained by adding equations (17) and (19) together.

$$\bar{\omega}_{in}^{n} = \begin{bmatrix} \omega_e cos(\varphi) + v_e/(R_E + h) & v_n/(R_N + h) & -\omega_e sin(\varphi) + v_e tan(\varphi)/(R_N + h) \end{bmatrix}^T \tag{20}$$

Now equation (20) is a function dependent both on velocity and position.

8.6 Inertial Navigation Equation

Describing the position of the dinghy in the n-frame is done by

$$\bar{r}^n = \begin{bmatrix} \varphi & \lambda & h \end{bmatrix}^T \tag{21}$$

Since velocity is described as the rate of change of its position with respect to a frame of reference and is a function of time, the velocities in the north, east and down can be expressed as[45]

$$\begin{bmatrix} v_N \\ v_E \\ v_D \end{bmatrix} = \begin{bmatrix} (R_E + h) & 0 & 0 \\ 0 & (R_N + h)cos(\varphi) & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \dot{\varphi} \\ \dot{\lambda} \\ \dot{h} \end{bmatrix} \tag{22}$$

Where $\dot{-}$ symbolizes the first derivative with respect to time. Thus, $\dot{\varphi}$, $\dot{\lambda}$ and $\dot{h}$ can be derived by rewriting equation (22) as

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\lambda} \\ \dot{h} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{(R_E+h)} & 0 & 0 \\ 0 & \frac{1}{(R_N+h)cos(\varphi)} & 0 \\ 0 & 0 & -1 \end{bmatrix}}_{D^{-1}} \begin{bmatrix} v_N \\ v_E \\ v_D \end{bmatrix} \tag{23}$$

Thus the dynamics describing the system can be expressed as

$$\dot{\bar{r}}_n = D^{-1} v_n \tag{24}$$
$$\dot{\bar{v}}_n = C_b^n \bar{f}^b - (2\omega_{ie}^n + \omega_{en}^n) \times \bar{v}_n + \bar{g}^n \tag{25}$$
$$\dot{C}_b^n = C_b^n (\Omega_{ib}^b - \Omega_{in}^b) \tag{26}$$

Where $\bar{f}^b$ is the specific force vector defined as the difference between the true acceleration in space and the acceleration due to gravity and $\bar{g}^n$ is the gravity vector. Where Ω is the skew matrix of ω, more precise Ω_{ib}^b the skew-matrix of the outputs of the strapdown gyroscopes and Ω_{in}^n is the skew-matrix of equation (20). Skew-matrix is defined as the cross product between the vectors.

8.7 INS mechanization

Since the INS has to work in discrete time domain, equations (24), (25) and (26) has to be discretized. The sampling time should be seen as a changing variable since it may not be constant instead it will fluctuate around 100 Hz, this means that $\Delta t_k = \Delta k_{k-1} - t_k$, this to improve the accuracy. The discrete dynamics are [46], Using equations (24), (25) and (26).

$$\bar{r}_{k+1}^n = \bar{r}_k + 0.5 D^{-1} (\bar{v}_k^n + \bar{v}_{k+1}^n) \Delta t \tag{27}$$
$$v_{k+1}^n = \bar{v}_k + \Delta \bar{v}_{k+1}^n \tag{28}$$
$$\dot{C}_b^n = C_b^n (\Omega_{ib}^b - \Omega_{in}^b) \Delta t \tag{29}$$

where

$$\Delta \bar{v}_{k+1}^n = \Delta \bar{v}_f^n - (2\omega_{ie}^n + \omega_{en}^n) \times \bar{v}_n \Delta t + \bar{\gamma} \Delta t \tag{30}$$

where $\gamma = [0 \quad 0 \quad 9.8123]$.

The Navigation frame Inertial Navigation System can now be seen in Figure 52, Where its possible to examine all the steps which were derived earlier, from the IMU to the output.

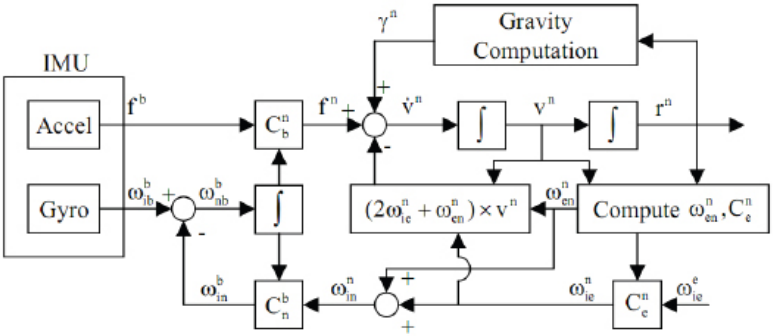


Figure 52: The Inertial Navigation Frame.

9 Implementation of Sensor Fusion Using a Kalman Filter.

9.1 Perturbation Analysis

Since the differential equations describing velocity, position, and the attitude, DCM are non-linear, utilize a perturbation series to minimize the errors [32].

$$\hat{r}^n = r^n + \delta r^n \tag{31}$$

$$\hat{v}^n = v^n + \delta v^n \tag{32}$$

$$\hat{C}_b^n = (I - E^n)C_b^n \tag{33}$$

Where E^n is the skew matrix and given as

$$\begin{bmatrix} 0 & -e_D & e_E \\ e_D & 0 & -e_N \\ -e_E & e_N & 0 \end{bmatrix} \tag{34}$$

Where $\hat{}$ is the computed values, δ and e_{NED} are the errors in North, East and upward direction, respectively.

9.1.1 Error Dynamics Position

Perturbing equation (31), the linearized position is obtained since the position dynamics is a function of both position and velocity, the position error dynamics is obtained as

$$\delta\dot{\bar{r}}^n = F_{rr}\delta\bar{r}^n + F_{rv}\delta\bar{v}^n \tag{35}$$

Where

$$F_{rr} = \begin{bmatrix} 0 & 0 & \frac{-v_N}{(M+h)^2} \\ \frac{v_E sin(\varphi)}{(N+h)cos^2(\varphi)} & 0 & \frac{-v_E}{(N+h)^2 cos(\varphi)} \\ 0 & 0 & 0 \end{bmatrix} \tag{36}$$

and

$$F_{rv} = \begin{bmatrix} \frac{1}{(M+h)} & 0 & 0 \\ 0 & \frac{1}{(N+h)cos(\varphi)} & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{37}$$

9.1.2 Error Dynamics Velocity

The same is used when deriving the perturbation for the velocity, equation (32), here, velocity is described by forces and gravity acting on the dinghy. Using equation (25), and perturbation equation (32), the following is obtained[45]

$$\delta\dot{\bar{v}}^n = \bar{v}_n \times \underbrace{(2\delta\bar{\omega}^n_{ie} + \delta\bar{\omega}^n_{en})}_{I} + \delta\bar{g}^n - \underbrace{(2\bar{\omega}^n_{ie} + \bar{\omega}^n_{en})}_{II} \times \delta\bar{v}^n + (\bar{f}^n\times)e^n + C^n_b\delta\bar{f}^n \tag{38}$$

where II is expressed as follow, using equation (20)

$$2\bar{\omega}^n_{ie} + \bar{\omega}^n_{en} = \begin{bmatrix} 2\omega_e cos(\varphi) + \frac{v_E}{(N+h)} \\ \frac{-v_N}{(M+h)} \\ -2\omega_e sin(\varphi) - \frac{v_E tan(\varphi)}{(N+h)} \end{bmatrix} \tag{39}$$

Now perturbing I with II, using equations (18) and (20), the following is obtained

$$2\delta\bar{\omega}^n_{ie} + \delta\bar{\omega}^n_{en} = \delta\Omega_r\delta\bar{r}^n + \delta\Omega_r\delta\bar{v}^n \tag{40}$$

where

$$\delta\Omega_r = \begin{bmatrix} -2\omega_e sin(\varphi) & 0 & \frac{-v_E}{(N+h)^2} \\ 0 & 0 & \frac{v_N}{(M+h)^2} \\ 2\omega_e cos(\varphi) - \frac{v_E}{(N+h)cos^2(\varphi)} & 0 & \frac{v_E tan(\varphi)}{(N+h)^2} \end{bmatrix} \tag{41}$$

and

$$\delta\Omega_v = \begin{bmatrix} 0 & \frac{-1}{(N+h)} & 0 \\ \frac{-1}{(M+h)} & 0 & 0 \\ 0 & \frac{-tan(\varphi)}{(N+h)} & \frac{v_E tan(\varphi)}{(N+h)^2} \end{bmatrix} \tag{42}$$

Then calculating the first term on the left side of equation (38), using equations (41) and (42)

$$\bar{v}^n \times (2\delta\bar{\omega}^n_{ie} + \delta\bar{\omega}^n_{en}) = \bar{v}^n \times \delta\Omega_r\delta\bar{r}^n + \bar{v}^n \times \delta\Omega_v\delta\bar{v}^n \tag{43}$$

Then expressing equation (38) as

$$\delta\dot{\bar{v}}^n = F_{vr}\delta\bar{r}^n + F_{vv}\delta\bar{v}^n + (\bar{f}^n\times)\bar{e}^n + C^b_n\delta\bar{f}^b \tag{44}$$

Expressing F_{vr} and F_{vv} in matrix notation

$$F_{vr} = \begin{bmatrix} -2v_E\omega_e cos(\varphi) - \frac{v_E^2}{(N+h)cos^2(\varphi)} & 0 & \frac{-v_N v_D}{(M+h)^2} + \frac{v_E^2 tan(\varphi)}{(N+h)^2} \\ 2\omega_e(v_N cos(\varphi) - v_D sin(\varphi)) + \frac{v_E v_N}{(N+h)cos^2(\varphi)} & 0 & \frac{-v_E v_D}{(N+h)^2} - \frac{v_N v_E tan(\varphi)}{(N+h)^2} \\ 2v_E\omega_e sin(\varphi) & 0 & \frac{v_E^2}{(N+h)^2} + \frac{v_N^2}{(M+h)^2} - \frac{2\gamma}{R+h} \end{bmatrix} \tag{45}$$

and

$$F_{vv} = \begin{bmatrix} \frac{v_D}{(M+h)} & -2\omega_e sin(\varphi) - 2\frac{v_E tan(\varphi)}{(N+h)} & \frac{v_N}{(M+h)} \\ 2\omega_e sin(\varphi) + \frac{v_E tan(\varphi)}{(N+h)} & \frac{v_D + v_N tan(\varphi)}{(N+h)} & 2\omega_e cos(\varphi) + \frac{v_E}{(N+h)} \\ -2\frac{v_N}{(M+h)} & -2\omega_e cos(\varphi) - 2\frac{v_E}{(N+h)} & 0 \end{bmatrix} \tag{46}$$

9.1.3 Error Dynamics Attitude

The output from the INS can be expressed using equation (26).

$$\hat{\dot{C}}_b^n = \hat{C}_b^n(\hat{\Omega}_{ib}^b - \hat{\Omega}_{in}^b) \tag{47}$$

Then relate the above equation with equation (33)

$$-\dot{E}^n C_b^n = (I_E^n)C_b^n(\delta\Omega_{ib}^b - \delta\Omega_{in}^b) \tag{48}$$

Expressing $\dot{E}^n$ by is self in left-hand side, the equation above can be expressed as

$$\dot{E}^n = -C_b^n(\delta\Omega_{ib}^b - \delta\Omega_{in}^b) \tag{49}$$

or in vector form

$$\dot{\bar{e}}^n = -C_b^n(\delta\omega_{ib}^b - \delta\omega_{in}^b). \tag{50}$$

Since we want to express the equation above in its error equation $\delta\bar{\omega}_{in}^b$, the following change can be done $\hat{\bar{\omega}}_{in}^b = \hat{C}_n^b\hat{\bar{\omega}}_{in}^n$ This can be expanded into

$$\bar{\omega}_{in}^b + \delta\bar{\omega}_{in}^b = C_b^n(I + E^n)(\bar{\omega}_{in}^n + \delta\bar{\omega}_{in}^n). \tag{51}$$

Using vector notation for the skew matrix, the equation above can be written as

$$\delta\bar{\omega}_{in}^b = C_n^b[\delta\bar{\omega}_{in}^n + (e^n\times)\bar{\omega}_{in}^n]. \tag{52}$$

Substituting equation (52) into equation (50), the following is obtained

$$\dot{\bar{e}}^n = \delta\bar{\omega}_{in}^n - (\bar{\omega}_{in}^n\times)\bar{e}^n - C_b^n\delta\bar{\omega}_{ib}^b \tag{53}$$

Then expressing the first term on the right-hand side into the position and velocity error terms explicitly, using equations. (23) and (39). The error attitude dynamics is then obtained and written as

$$\dot{\bar{e}}^n = F_{er}\delta\bar{r}^n + F_{ev}\delta\bar{v}^n - (\bar{\omega}_{in}^n\times)\bar{e}^n - C_b^n\delta\bar{\omega}_{ib}^b. \tag{54}$$

Where F_{er} and F_{ev} is

$$= F_{er} = \begin{bmatrix} -\omega_e sin(\varphi) & 0 & \frac{-v_E}{(N+h)^2} \\ 0 & 0 & \frac{v_N}{(M+h)^2} \\ \omega_e cos(\varphi) - \frac{v_E}{(N+h)cos^2(\varphi)} & 0 & \frac{v_E tan(\varphi)}{(N+h)^2} \end{bmatrix} \tag{55}$$

$$= F_{ev} = \begin{bmatrix} 0 & \frac{1}{(N+h)} & 0 \\ \frac{-1}{(M+h)} & 0 & 0 \\ 0 & \frac{-tan(\varphi)}{(N+h)} & 0 \end{bmatrix} \tag{56}$$

9.2 Implementing the Fusion Kalman Filter

This system uses a feedback method to control the drift errors in the IMU, the model can be seen in Figure 53.

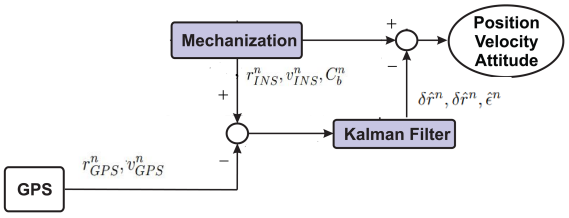


Figure 53: The feedback-method which is used in the system.

Implementing a continuous Kalman Filter using integration between Inertial Navigation System and a Global Positioning System.

$$\dot{\hat{\bar{x}}} = F\bar{x} + G\bar{u} \tag{57}$$

Where F is describing the dynamics of the system, $\bar{x}$ is the state vector, G is the design matrix, $\bar{u}$ is the input matrix, i.e. forces acting on the dinghy recorded by the IMU and $\hat{\bar{x}}$ is the estimated state vector.

Where F is a 9×9 matrix and contain equations (36), (37), (45), (46), (55), (56), (20), and $\bar{x}$ is a 9×1 is given by

$$F = \begin{bmatrix} F_{rr} & F_{rv} & 0 \\ F_{vr} & F_{vv} & (\bar{f}^n\times) \\ F_{er} & F_{ev} & -(\bar{\omega}^n_{in}\times) \end{bmatrix}, \qquad \bar{x} = \begin{bmatrix} \delta\bar{r}^n \\ \delta\bar{v}^n \\ \delta\bar{e}^n \end{bmatrix} \tag{58}$$

G is a 6×6 matrix and $\bar{u}$ is a 6×1 vector and is defined as.

$$G = \begin{bmatrix} -C^n_b & 0 \\ 0 & C^n_b \end{bmatrix}, \qquad \bar{u} = \begin{bmatrix} \delta\bar{\omega}^b_{ib} \\ \delta\bar{f}^b \end{bmatrix} \tag{59}$$

The elements of $\bar{u}$ is assumed to be white noise with zero mean, thus

$$Q = diag\begin{bmatrix} \sigma^2_{ax} & \sigma^2_{ay} & \sigma^2_{az} & \sigma^2_{gx} & \sigma^2_{gy} & \sigma^2_{gz} \end{bmatrix}^T \tag{60}$$

where $\sigma^2_{a(x,y,z)}$ and $\sigma^2_{g(x,y,z)}$ is the variance for the accelerometer and gyroscope in every direction, respectively.

Since a computer does not use continuous time domain the equation has to be transformed into discrete domain. The state equation will then be expressed as

$$\hat{\bar{x}}_{k+1} = \Phi_k\bar{x}_k + \bar{w}_k \tag{61}$$

Here Φ_k is the state transition matrix in discrete time domain, in order to obtain a discrete state transition matrix the inverse Laplace transform is performed on the continuous state transition matrix, i.e.

$$\Phi_k = \mathcal{L}^{-1}\left[(SI - F)^{-1}\right] \tag{62}$$

Since the sample interval, Δt is very small in this case equation (62) can be approximated using Taylor approximation

$$\Phi_k = e^{F\Delta t} \approx I + F\Delta t \tag{63}$$

From equation (61), $\hat{w}_k$ is the driven response t_{k+1} due to the input white noise. White noise is uncorrelated between sample periods, i.e the noise between t_k and t_{k+1} is uncorrelated [48]. Then the covariance matrix which is associated with $\bar{w}_k$ is[48]:

$$\mathbb{E}[\bar{w}_i \bar{w}_j^T] = \begin{cases} Q_k & i = j \\ 0 & i \neq j \end{cases} \tag{64}$$

and Q_k can then be approximated using a first order of the discrete transition matrix[49]:

$$Q_k \approx \Phi_k G Q G^T \Phi_k^T \tag{65}$$

If equation (60) is analyzed, by increase the norm of Q_k the Kalman Filter trusts the measurements more than the system, which will make the output more noisy due to noise induced from the measurements, the advantages with a large norm is that the time lag will decrease. If the norm is small the measurements will be less induced by measurement noise but time lag will increase, which means that we do not trust the measurements. Determining Q_k can be done by testing several different settings and from that make an assumption, but a good assumption should be that the trajectory of the output should follow the GPS data when the GPS is connected to several satellites.

The Kalman filter is a linear quadratic estimator which is recursive and the estimator is unbiased and has minimum variance. The algorithm starts with a random process model, i.e. equation (61) and the following observation matrices [49].

$$z_k = H_k \bar{x}_k + \bar{e}_k \tag{66}$$

where z_k is the measurement vector and e_k is random measurement noise, with following characteristics [48].

$$\mathbb{E}[\bar{e}_i \bar{e}_j^T] = \begin{cases} R_k & i = j \\ 0 & i \neq j \end{cases} \tag{67}$$

The Kalman Filter can be seen as a two-step filter with a prediction update and a correction update. In the latter case, the Kalman gain is first calculated by

$$K_k = P_k^- H^T (H P_k^- H^T + R_k)^{-1} \tag{68}$$

then the state vector is updated

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{69}$$

the last step is updating the covariance matrix

$$P_k = (I - K_k H) P_k^- \tag{70}$$

When correction update is done the algorithm makes a prediction update this is done by

$$\hat{x}_k^- = \hat{\hat{x}}_k \Phi_k \tag{71}$$

then updating its covariance

$$P_k^- = \Phi_k P_k \Phi_k^T + Q_k \tag{72}$$

where $(\bar{}_k)$ should be inferred such as calculating the prediction at time k given time $k-1$.

The measurement vector z_k is containing the difference of velocity and position from the INS and GPS

$$z_k = \begin{bmatrix} \lambda_{INS} - \lambda_{GPS} \\ \varphi_{INS} - \varphi_{GPS} \\ h_{INS} - h_{GPS} \\ v_{N_{INS}} - v_{N_{GPS}} \\ v_{E_{INS}} - v_{E_{GPS}} \\ v_{d_{INS}} - v_{d_{GPS}} \end{bmatrix} \tag{73}$$

The measurement matrix H_k is defined such

$$H_k = \begin{bmatrix} I_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{74}$$

Where $I_{3\times3}$ is the identity matrix of size 3×3. The measurement noise matrix R_k is defined such as

$$R_k = diag(\sigma_{r_N}^2 \quad \sigma_{r_E}^2 \quad \sigma_{r_d}^2 \quad \sigma_{v_N}^2 \quad \sigma_{v_E}^2 \quad \sigma_{v_d}^2) \tag{75}$$

where $\sigma_{r_{(N,E,d)}}^2$ and $\sigma_{v_{(N,E,d)}}^2$ is the variance of the position and velocity in all direction, respectively.

The Kalman Filter is invoked every time the GPS is updated, i.e. $1Hz$, but since the IMU and the GPS updates at different frequencies a problem arises. The problem is such that at $t_{GPS}(k)$ there will not be a value to read from the IMU, since discrete time domain. To solve this problem a linear interpolation is done between $t_{imu}(k)$ and $t_{imu}(k+1)$, where $t_{imu}(k) \leq t_{GPS}(k) \leq t_{imu}(k+1)$, see Figure 54.

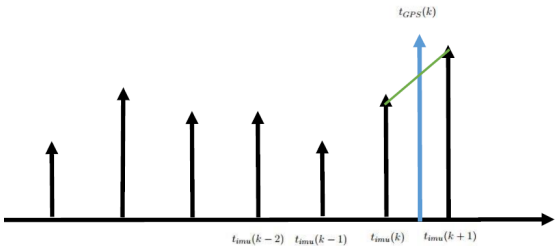


Figure 54: Different update sequences IMU and GPS

The following linear interpolation equation is used for calculating the positions at $t_{GPS}(k)$

$$r_n(t_k(GPS) = r_n(t_{imu}(k)) + \frac{r_n(t_{imu}(k+1)) - r_n(t_{imu}(k))}{t_{imu}(k+1) - t_{imu}(k)}(t_{GPS}(k) - t_{GPS}(k)) \tag{76}$$

and the equations for the velocities

$$v_n(t_k(GPS) = v_n(t_{imu}(k)) + \frac{v_n(t_{imu}(k+1)) - v_n(t_{imu}(k))}{t_{imu}(k+1) - t_{imu}(k)}(t_{GPS}(k) - t_{GPS}(k)). \tag{77}$$

9.3 Result

The data provided by the Inertial Navigation System is working better than just using the raw GPS data, on some occasions. If the goal is to estimate the position, then it is better to use the INS connected to a Kalman Filter, this can be seen in Figure 55. As seen, the Filtered data is inaccurate in the beginning but after some time converges to the true value, true value as in the case of wanted value, and at some points even catches the wanted value. In comparison to the raw GPS data, the INS is a better estimator for the user's position.

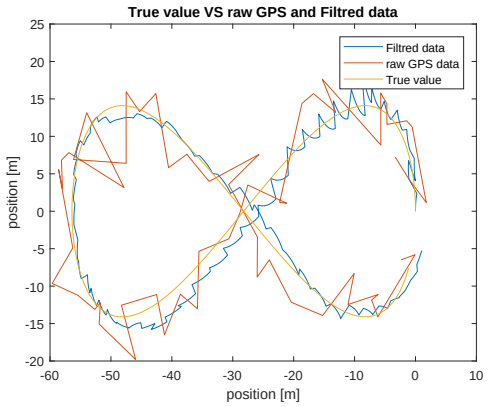


Figure 55: True VS. Filtered and raw GPS data.

The Mean Squared Error, MSE for the position in XY plane, can be seen in Figure 56 for both INS and raw GPS data we can see that the error is lower for the INS data, in both cases.

Figure 56: Mean Squared Error XY-plane separately.

When estimating the height, i.e. Z-direction the result is poor for the INS, the estimated value from the INS is diverging from the true value, this can be seen in Figure 57. In this case, the GPS data is more accurate.

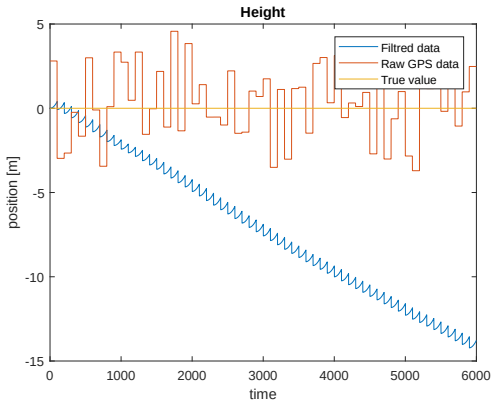


Figure 57: Height. True VS. Filtered and raw GPS data.

The velocities in the XY plane does not represent the true system in a good manner either this can be seen in Figure 58. In this case, the GPS data is more accurate as well.

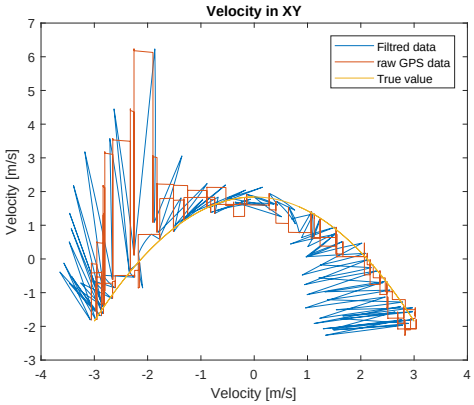


Figure 58: The velocities in XY plane.

9.4 Discussion/Future Work

Depending on what the user wants to measure, e.q. position in two dimensions, height or velocity, different sensors should be used. In the firts case, the INS should be used in the case of the two latter the GPS should be used.

The Matlab code that exists, as right now, can read from the Nucleo Board using a serial read (assuming that the Nucleo Board uses the same firmware that was used by us), use stored data, (data created by our group) and also generate new data, with the IMU and GPS wanted characteristics.

Also, there is a static calibration program for the accelerometer. To see how the program works there exists a *readme.txt* file in the Matlab folder.

Implement a working real-time system for the sensor fusion between the GPS and INS, there exits C-code for this already, but there is some serious bug/bugs which makes the error stored in $\hat{X}$ diverge exponentially. A serious attempt finding the problem was conducted and is narrowed down to a problem when calculating the Kalman Gain, K_k. Also, figure out why the height position diverges from the wanted value.

Improvements in the INS/Kalman Filter implementation can be done as well, here are some thoughts that might be seen as guidelines/starting points to improve the system. To obtain much more accurate information from the INS one should consider more advanced techniques for calibrating the IMU. In this case, a static calibration technique was studied. This is done by keeping the IMU in a fixed position to observe the effect of gravity. Consider implementing a more advanced technique, e.g. using this three-stage process.

- Coarse checking or evaluation using very simple test, such as a single stationary test on a bench, to establish the variance and standard deviation of the IMU, (gravity).
- Static testing and studying the effects of natural phenomenon on the IMU.
- Dynamic testing where the IMU is subjected to motions that should resemble the conditions out on the sea, e.g. constant waves hitting the sides of the boat.

The purpose of this more advanced calibration technique is to determine the following parameters[46]

- scale factors
- scale factor linearity
- null bias error
- axis alignment error, (might not be perfectly perpendicular)

Also, the dinghy's body may flex due to rough sea conditions, to solve this, implement one master INS and one slave INS, consider literature as Strap Down Inertial Navigation Technology, D.H. Titterton and J.L. Weston as a starting point for these different calibration methods.

A more advanced interpolation technique between IMU data and GPS data, here referencing to literature like "Fundamentals of Scientific Computing", *Bertil Gustafsson*, e.g., Lagrange polynomial can be used to capture the true value better than using a linear interpolation.

One should also try different types of control methods to see if one can achieve more accurate results or not, e.g. feed-forward method.
Weighted parameters if one believes the INS more than the GPS or vice-versa, this question arises when e.g. the GPS is connected to many/few satellites or if the user is having a large/small velocity or sensor anomalies.

References

[1] https://www.autodesk.com/products/fusion-360/overview

[2] ltu.diva-portal.org/smash/get/diva2:1039147/FULLTEXT01.pdf

[3] http://www.st.com/en/development-tools/stm32cubemx.html

[4] KiCad open source electronics design, kicad-pcb.org/

[5] http://www.st.com/en/embedded-software/x-cube-ble1.html

[6] http://www.st.com/en/embedded-software/x-cube-mems1.html

[7] http://play.google.com/store/apps/details?id=com.st.blunrg

[8] http://www.st.com/en/ecosystems/x-nucleo-idb05a1.html

[9] http://www.st.com/en/ecosystems/x-nucleo-iks01a2.html

[10] http://x-io.co.uk/open-source-imu-and-ahrs-algorithms

[11] http://x-io.co.uk/res/doc/madgwick_internal_report.pdf

[12] http://update.maestro-wireless.com/GNSS/A2235-H/Maestro_GPS_Evaluation_Kit_EVA2235_H_User_Manual_V01.pdf

[13] http://github.com/jacketizer/libnmea

[14] http://www.st.com/content/st_com/en/products/ecosystems/stm32-open-development-environment/stm32-nucleo-expansion-boards/stm32-ode-sense-hw/x-nucleo-53l0a1.html

[15] https://developer.android.com/reference/android/app/Service.html

[16] https://www.movable-type.co.uk/scripts/latlong.html

[17] https://en.wikipedia.org/wiki/Rhumb_line

[18] https://en.wikipedia.org/wiki/Equirectangular_projection

[19] https://en.wikipedia.org/wiki/Haversine_formula

[20] https://en.wikipedia.org/wiki/Sailing

[21] http://newt.phys.unsw.edu.au/~jw/sailing.html

[22] https://sv.wikipedia.org/wiki/Volvo_Amazon

[23] https://sv.wikipedia.org/wiki/PNG

[24] https://developer.android.com/reference/android/app/Activity.html

[25] https://en.wikipedia.org/wiki/Unified_Modeling_Language

[26] https://developer.android.com/reference/android/opengl/GLSurfaceView.html

[27] https://developer.android.com/reference/android/speech/tts/TextToSpeech.html

[28] https://developer.android.com/reference/android/speech/tts/UtteranceProgressListener.html

[29] https://developers.google.com/maps/

[30] https://edg.uchicago.edu/tutorials/load_cell/

[31] http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=FX19&DocType=DS&DocLang=English

[32] https://en.wikipedia.org/wiki/Perturbation_theory

[33] http://www.lygte-info.dk/info/battery%20protection%20UK.html

[34] KiCad library entry for voltage regulator component lm1117 by *ObKo*, 2012, https://github.com/ObKo/kicad-libraries/blob/master/libraries/lm1117.lib.

[35] Bi-directional Logic Level Converter, *Sparkfun*, Aug 4, 1997, https://cdn.sparkfun.com/tutorialimages/BD-LogicLevelConverter/an97055.pdf.

[36] KiCad library entry for USBtoSerial component FT232RL by *jbaker0428*, Oct 3, 2010 https://github.com/jbaker0428/Kicad-Libraries/blob/master/library/ftdi.lib.

[37] Overvoltage and Reverce-voltage Protection in Automotive Systems, Application note 760, *Maxim integrated*, Apr 02, 2002, https://www.maximintegrated.com/en/app-notes/index.mvp/id/760.

[38] Battery Management Studio (bqStudio) utility tool for TI Battery management products, http://www.ti.com/tool/bqstudio.

[39] ST, Application note: AN4907, *VL53L0X ranging module cover window guidelines*, http://www.st.com/content/ccc/resource/technical/document/application_note/group0/9d/93/be/33/13/be/46/19/DM00326504/files/DM00326504.pdf/jcr:content/translations/en.DM00326504.pdf

[40] Duygun, M., Kutlu, L. & Sickles, R.C., Journal of Productivity Analysis, 2016, 46: 155, https://doi.org/10.1007/s11123-016-0477-z

[41] Power Guy, *Constant Current Constant voltage*, May 26, 2016, San Diego, USA, https://www.us.tdk-lambda.com/media/292143/tdk-lambda-blog-052616.pdf.

[42] D.L. Hall and J. Llinas, *An introduction to multisensor data fusion*, Proceedings of the IEEE, 85(1):6 –23, jan 1997.

[43] B.R. Grover H, Patrick, *Introduction to random signals and applied Kalman filtering*, 1997.

[44] Society of Naval Architects and Marine Engineers (SNAME), *Principles of Naval Architecture*, 1989, Vol. III, *SNAME*.

[45] Dr. Oliver Nelles, `nonlinear system identification`.

[46] Noureldin., Karamat T.B., Georgy J., 2013, `Basic Navigational Mathematics, Reference Frames and the Earth's Geometry. In: Fundamentals of Inertial Navigation, Satellite-based Positioning and their Integration`, Springer, Berlin, Heidelberg.

[47] D.L. Hall and J. Llinas. *An introduction to multisensor data fusion*, Proceedings of the. IEEE, 85(1):6 –23, jan 1997.

[48] Henry Stark, John W Woods, *Probability and Random Processes with Applications to Signal Processing*, 4/E 2012, Pearson Higher Education.

[49] Donald E. Catlin, *Estimation, Control, and the Discrete Kalman Filter*, 1989.

[50] Weicker, P., *A Systems Approach to Lithium-ion Battery Management*, Boston, Artech House, 2014.

Appendices

A Large Figures

A.1 Schematic of sensorboard

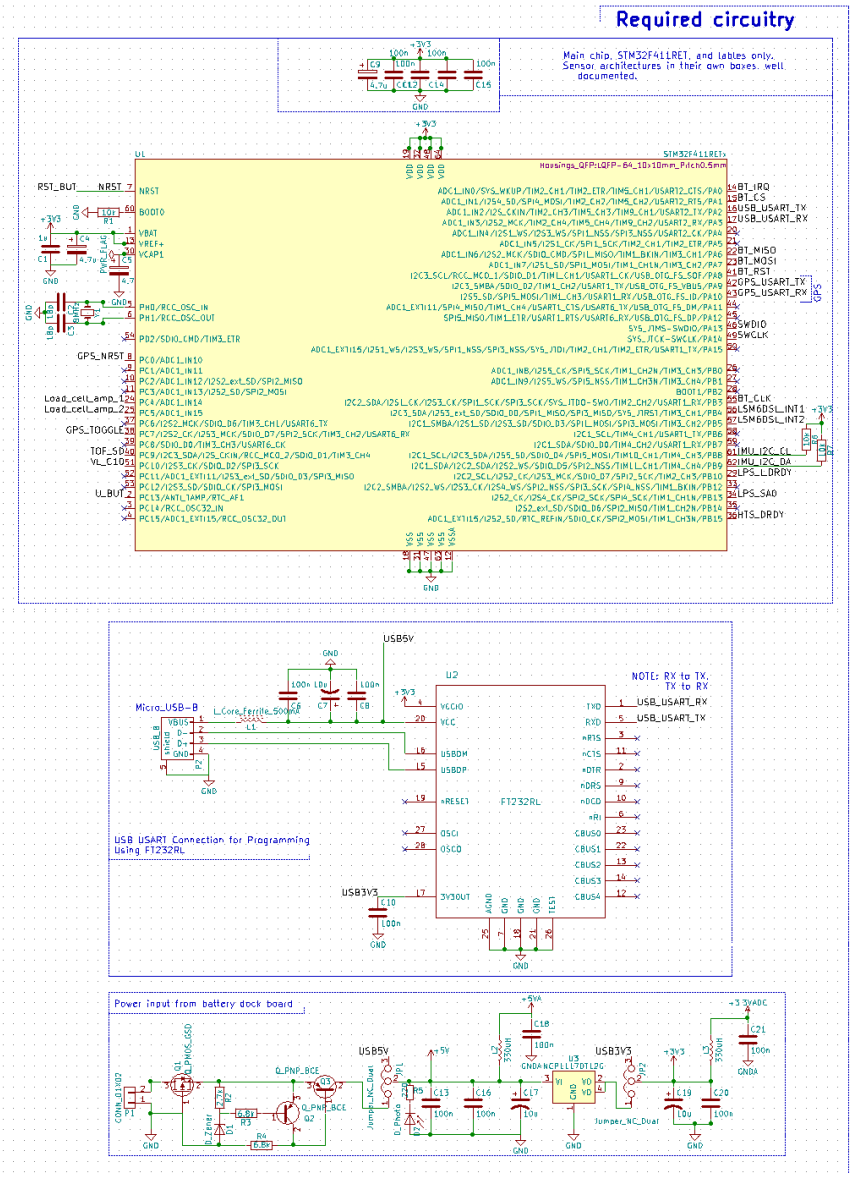


Figure 59: Schematic of revision 3, part a: Required circuitry.

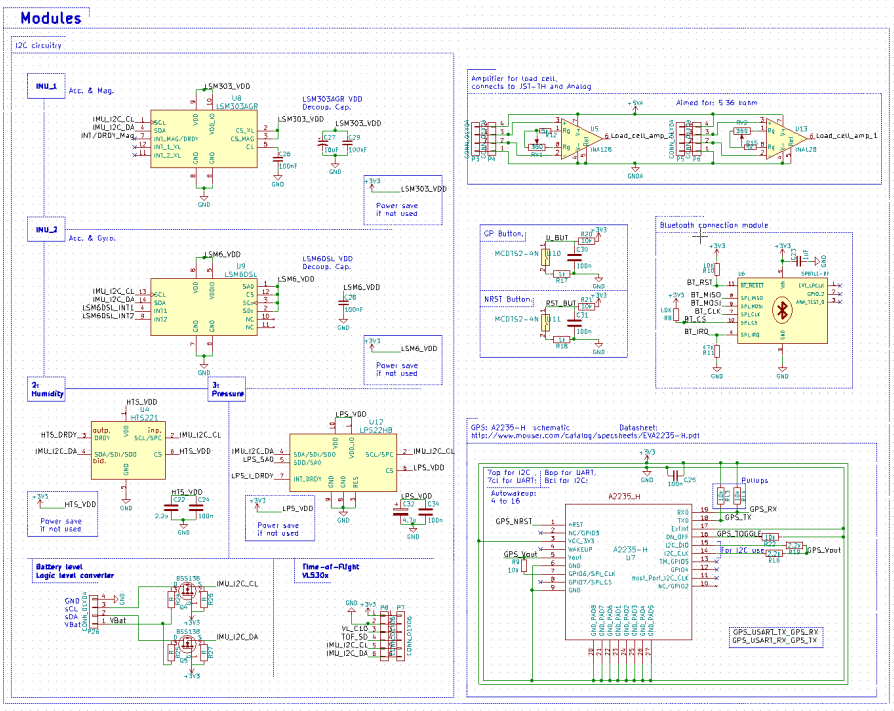


Figure 60: Schematic of revision 3, part b: Modules.

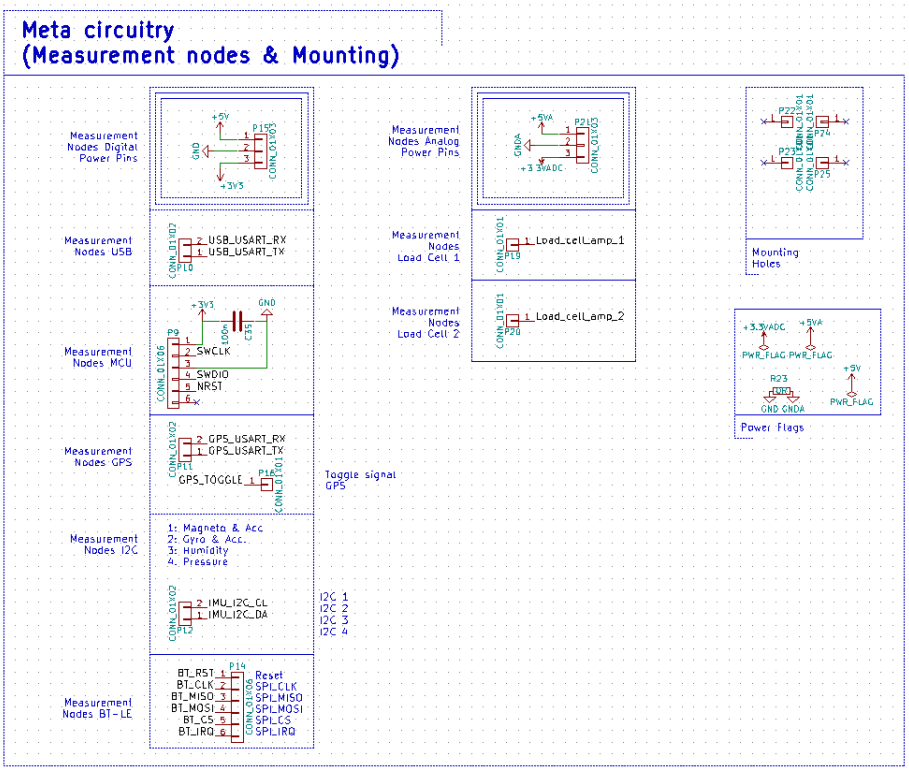


Figure 61: Schematic of revision 3, part c: Meta circuitry.

A.2 Schematic of BMS

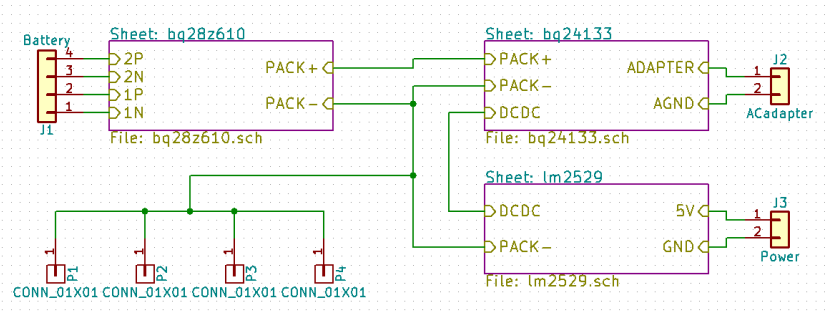


Figure 62: Root schematic of BMS circuitry.

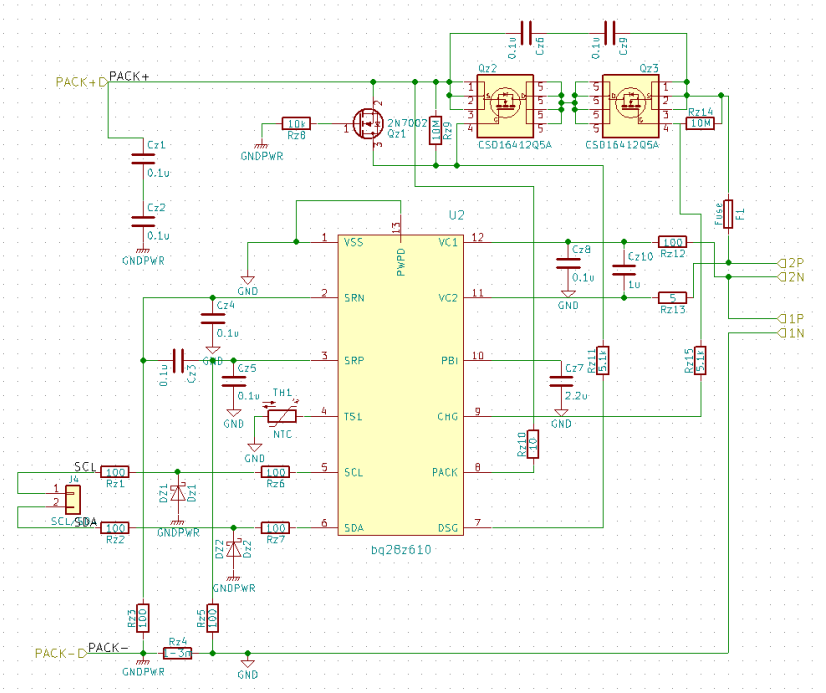


Figure 63: Schematic of BMS, part a: bq28z610.

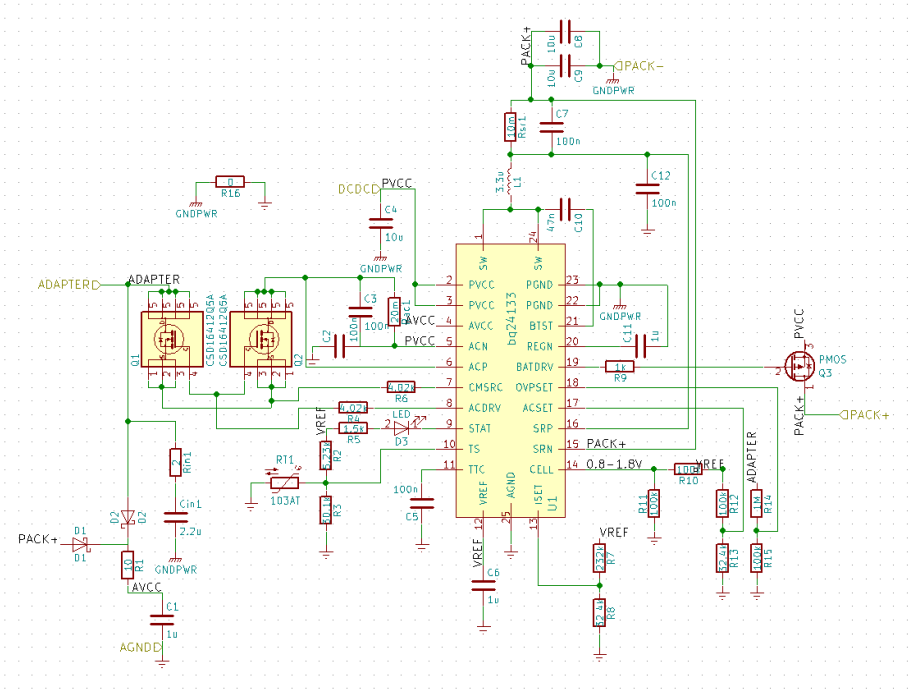


Figure 64: Schematic of BMS, part b: bq24133.

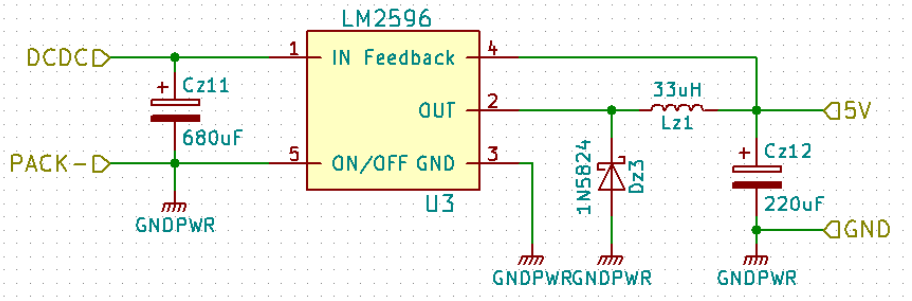


Figure 65: Schematic of BMS, part c: lm2596.

A.3 CubeMX software

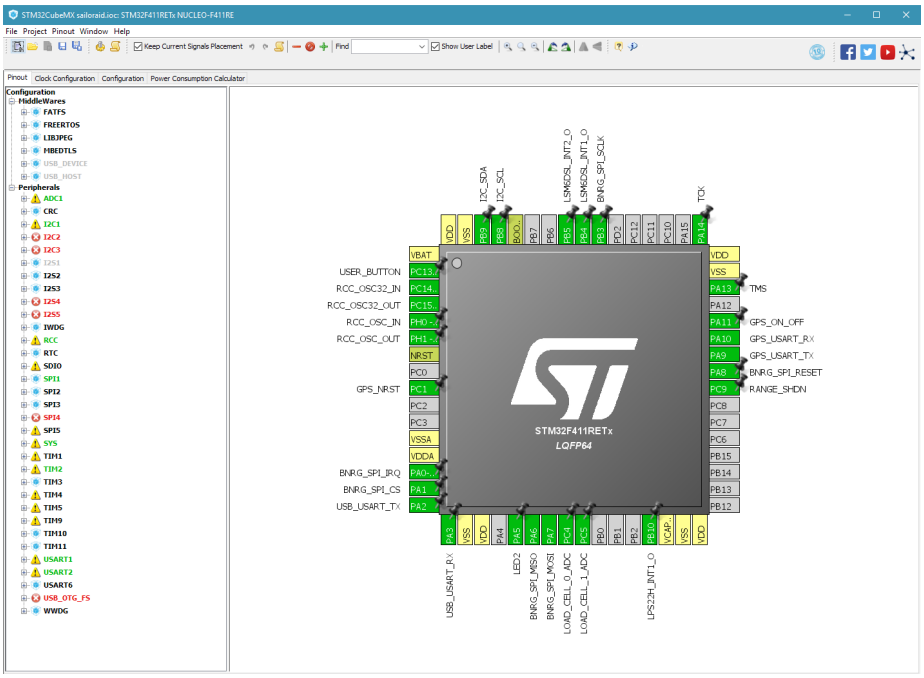


Figure 66: CubeMX software

B Other Figures

B.1 Casing Dimensions

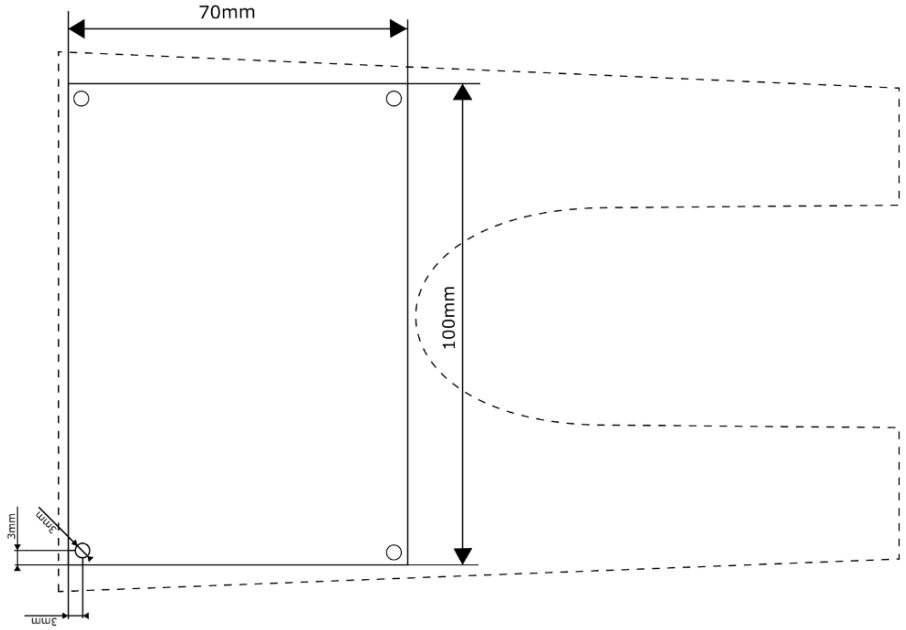


Figure 67: System enclosure shape and dimensions

C Lists

C.1 Bill of Materials: Main PCB

Main Board Components	**Package**	**Quantity**
Capacitors:		
18p	0805	2
100n	0805	21
1u	0805	2
2.2u	0805	1
4.7u	0805	4
10u	Electrolytic SMD 5x5.3	6
Resistors:		
220	0805	1
1k	0805	2
1k	potentiometer	2
2.2k	0805	2
2.7k	0805	1
5k	0805	2
6.8k	0805	2
10k	0805	11
47k	0805	1
Inductors:		
1.5u	1206	1
330u	Radial 200mil Pin Pitch	2
Integrated Circuits:		
STM32F411RE	LPQF64	1
SPBTLE-RF	Custom	1
A2235-H	Custom	1
FT232R	SSOP-28	1
LSM6DSL	LGA-14L	1
LSM303AGR	LGA-12	1
HTS221	HLGA-6L	1
LPS22HB	HLGA-10L	1
INA128	SOIC-8	2
LM1117-3.3	SOT-223	1
Semiconductors:		
P-MOS PMV65XP	SOT-23	1
PNP BJT FMMT718	SOT-23	2
Zener Diode, 5.6V	SOD-323	1
Green LED	ø2.5mm 100mil pitch	1
Others:		
Tactile SMD Button		2
8MHz Crystal		1
Micro USB-B		1
Straight 6-Pin Header	100mil pitch	1
Straight 3-Pin Header	100mil pitch	2
Pin Jumper		2

C.2 Bill of Materials: Battery Management PCB

Battery Management Circuit	**Package**	**Quantity**
Capacitors:		
47n	0603	1
100n	0603	13
1u	0603	4
2.2u	0603	2
10u	1206	3
220u	Electrolytic SMD 6.3x5.8	1
680u	Electrolytic SMD 6.3x5.8	1
Resistors:		
0	0603	1
1-3m	2512	1
10m	2512	1
20m	2512	1
2	2010	1
5	2512	1
10	0603	1
10	1206	1
100	0603	7
100	2512	1
1k	0603	1
1.5k	0603	1
4.02k	0603	2
5.1k	0603	2
5.23k	0603	1
10k	0603	1
30.1k	0603	1
32.4k	0603	2
100k	0603	4
232k	0603	1
1M	0603	1
10M	0603	2
103AT	Axial Standing 100mil Pitch	2
Inductors:		
3.3u	7.3x7.3mm SMD	1
33u	12x12mm SMD	1
Integrated Circuits:		
BQ24133	VFQN-24	1
BQ28Z610	S-PDSO-N12	1
LM2529	TO-263-5	1
Semiconductors:		
Schottky Diode	SOD-323	4
1N5824	Axial 10.16mm Pitch	1
N-MOS CSD16412Q5A	TDSON-8-1	4
Red LED	ø3mm Radial 100mil Pitch	1
2N7002	SOT-23	1
Power P-MOS	SOT-23	1

The project; complete with all software code, the application, hardware files and more, can be found online in our github-repo:
https://github.com/jsjolund/sailoraid