

LULEÅ UNIVERSITY OF TECHNOLOGY

DEPT. OF COMPUTER SCIENCE, ELECTRICAL AND  
SPACE ENGINEERING

D7039E – PROJECT IN INDUSTRIAL COMPUTER SYSTEMS

---

## Project *SailorAid*

---

*Authors*

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

*Supervisor*  
van Deventer, Jan

January 12, 2018



**Abstract**

Throughout history sailing has been a key ingredient to our civilization, trading, fishing, exploring, transporting, i.e. today's civilization would not have been the same if sailing wasn't invented. Today sailing has been developed more into a hobby from large sailing boats all the way down to one-man dinghies. This system is developed to help the skipper on dinghies to make better decisions, for both experienced and novices sailors. This is achieved using inexpensive sensors and then give the skipper feedback using graphics, sound, and vibrations through an android application. Measurements are done on forces exerted on the centerboard, tilt of the dinghy, position, and velocity. The design has been focused on energy efficiency, waterproof and easy-to-handle.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
ERIKSSON, KENNY (SJÖLUND, JOHANNES)		
1.1	Goals . . . . .	2
<b>2</b>	<b>The Physics of Sailing</b>	<b>3</b>
THEOLIN, HENRIK (AXELSSON, OSKAR)		
2.1	Point of sail . . . . .	3
2.2	Velocity . . . . .	4
<b>3</b>	<b>Mechanical sensors</b>	<b>6</b>
LUNDBERG, JOSEF (GRAPE, ELIAS)		
3.1	Force sensors . . . . .	6
3.2	Amplifier . . . . .	10
3.3	Height of centerboard sensor . . . . .	11
<b>4</b>	<b>Main sensorboard design</b>	<b>14</b>
BROLIN, DANIEL (ERIKSSON, KENNY)		
4.1	Method . . . . .	14
4.2	Component choices . . . . .	18
ERIKSSON, KENNY (BROLIN, DANIEL)		
4.3	results (TiP) . . . . .	20
4.4	discussion (TiP) . . . . .	20
<b>5</b>	<b>Casing</b>	<b>21</b>
LUNDBERG, JOSEF ( , )		
5.1	First revision . . . . .	21
5.2	Second revision . . . . .	21
5.3	Third revision . . . . .	22
5.4	Fourth revision . . . . .	23
5.5	Results . . . . .	24
<b>6</b>	<b>Battery charging circuitry</b>	<b>25</b>
GRAPE, ELIAS (LUNDBERG, JOSEF)		
<b>7</b>	<b>Software Design</b>	<b>26</b>
SJÖLUND, JOHANNES (BROLIN, DANIEL)		
7.1	ARM firmware . . . . .	26
7.2	Android application . . . . .	33
THEOLIN, HENRIK (AXELSSON, OSKAR)		
7.3	Results . . . . .	45
<b>8</b>	<b>Inertial Navigation System and Kalman Filter</b>	<b>46</b>
AXELSSON, OSKAR (THEOLIN, HENRIK)		
8.1	Inertial Navigation System . . . . .	47
8.2	Sensor fusion . . . . .	47
8.3	Navigation Frames . . . . .	47

8.4	Transformation Between Frames . . . . .	49
8.5	Inertial Navigation Equation . . . . .	51
8.6	INS mechanization . . . . .	52
<b>9</b>	<b>Implementation of Sensor Fusion Using a Kalman Filter.</b>	<b>53</b>
9.1	Implementing the Fusion Kalman Filter . . . . .	55
9.2	Result . . . . .	59
9.3	Discussion/Future Work . . . . .	61
	<b>References</b>	<b>63</b>
	<b>10 APPENDIX</b>	<b>66</b>



# 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 sailor.

## 1.1 Goals

The primary functional goals are as follows:

- . Boat attitude
  - Implementing an appropriate Inertial Measurement Unit (IMU):
    - \* 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 sensor mount off the centerboard
  - 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 depending 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 achieve 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 around 135°

Running (drag) angle around 180°

and are represented in Fig. 1. A sailor wants to prevent the sail from luffing, luffing 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.

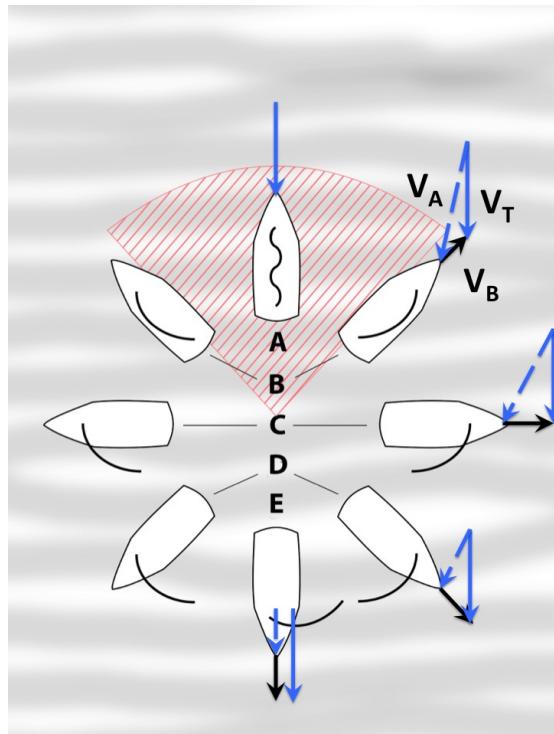


Figure 1: Points of sail

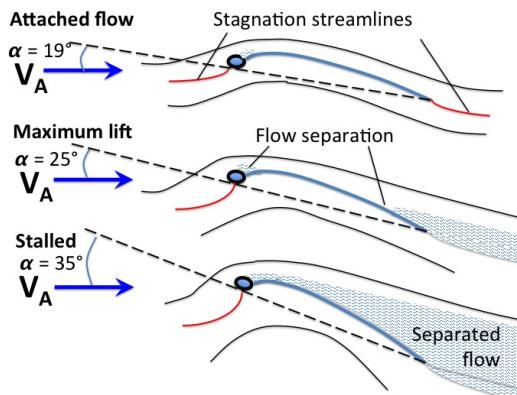


Figure 2: Sail angles of attack

## 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 that the fastest velocity is achieved when the dinghy is parallel to the true wind, however, this is 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<sup>29</sup>. 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 of 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 hiking, leaning outside of the hull to alter the center of mass for the dinghy. Lastly when all other measures were taken the sailor must perform reefing and reducing the area of the sail and lowering the center of effort from the sails.

### 3 Mechanical sensors

LUNDBERG, JOSEF (GRAPE, ELIAS)

Since all kinds of sailboats main feature is to move completely analogue without the need of fuel or electricity, the use and optimization of surrounding forces are of foremost importance. Measuring this will provide the sailor with all the information needed to optimize the way they move and control their dinghy.

To get any kind of measurements of the dinghy, sensors are needed. As of now there are sensors measuring a wide array of items. This section will talk about these sensors; 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 prototypes are made around the sensors to acquire the data. Designs are made with the Computer-aided design (CAD) software Fusion 360<sup>4</sup>. The models are manufactured using a 3D printer for fast and easy development.

#### 3.1 Force sensors

The function of the daggerboard is to compensate for the force that the wind is pushing on the sail which also helps to hold the dingy level in the water. The goal is to have a system that can measure the forces that pushes on the daggerboard by the water it goes through. By measuring the side forces on the board, a rough estimation of the exerted force on the sail can be made.

##### 3.1.1 The implementation

After some different solutions was suggested the final sensor and measurement method was chosen as the most prominent and clean approach. This 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 has as its home turf. With this implementation the daggerboard itself will not be disassembled or modified in any way. Solutions that required the sensors to be mounted on the outside or in parts that would be in danger if a crash might occur was scratched. Other solutions are, either way, more difficult to apply and mount or more complex.

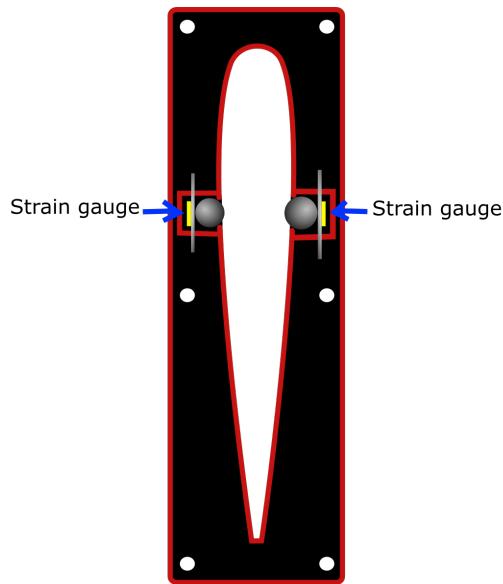


Figure 3: Function of first prototype

### 3.1.2 The prototype

To implement the gauges, a prototype is designed to show how the measurements will be made. The prototype is a bit bigger than the intended solution for this project but it's good to see how it would be constructed. The function is easy to understand. The board goes on the outside and can easily slide up and down past this steel bead. The bead itself is kept inside a small space where it can move in and out.

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

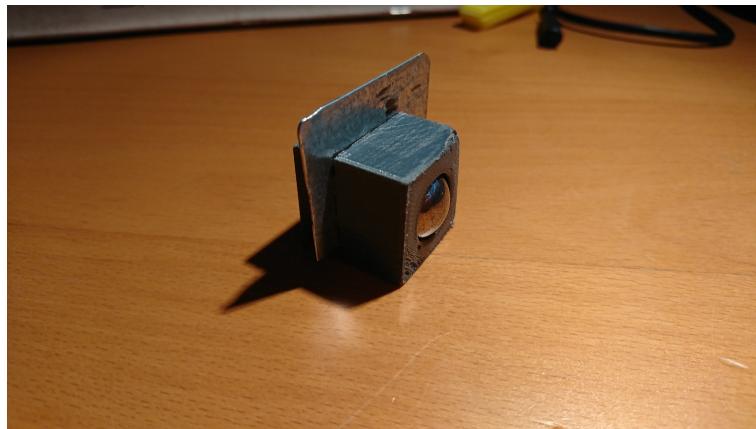


Figure 4: First prototype of a force sensor

The force is measured at the back where there will be a plate. The deflection of this plate which will be the origin to the strain will be measured through 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 a such small signal the system might have issues with noise. Another problem is the signal might drift, and therefore make different measurements as the circuit is running. And finally, with the measured values getting amplified with a big amount the resulting signal may be off by a large amount.

A better solution is to make some research into load cells, which is a sensor which also utilizes strain gauges to measuring forces. The difference is that the gauges are already implemented in the sensor. The difference in this prototype is instead of having a metal plate, it can be built with a piece of plastic or rubber which can deform so the force is distributed directly to the sensor. By implementing this sensor, a lot of time was saved in troubleshooting. And by having a sensor unit, the modified mounting plate will be easier to produce.

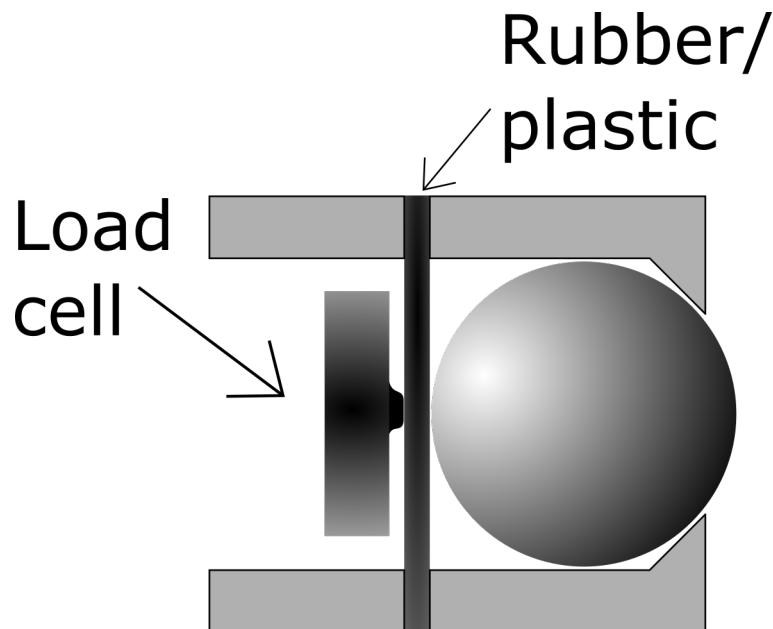


Figure 5: Function of second prototype

### 3.1.3 Sensor

The force from the board onto the mounting plate will be a considerable amount. The actual force is something that's 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 it's rated for a 150% overload without causing some damage to the sensor.

The chosen sensor for this application is this part, the compression load cell called FX1901<sup>7</sup>. From the datasheet, the voltage readings of this part could be calculated. The maximum voltage difference is calculated to be around 180mV.



Figure 6: Load cell, FX1901

### 3.2 Amplifier

With a such small signal, an amplifier is needed to get some desired measurements. A good measurement signal to the Microcontroller Unit (MCU) should be in the order of in between 0–3.3 volts. Since the maximum value from the load cell is 180mV a signal of 3.3 volts is achieved by an amplification gain of around 20. A suitable amplifier needs to be chosen from this implementation. Inspiration is taken from The University of Chicago?? in an experiment where they use this exact load cell together with an instrumental amplifier called INA125. This amplifier is somewhat more complicated and has some more features than other amplifiers.

In the same family of instrument amplifiers, a model called INA126<sup>?</sup> is selected as a less complicated and more power efficient solution. This amplifier has a smaller power draw due to some simpler functions and lesser precision. But for this application it's sufficient. The implementation is easy and the gain can easily be determined by connection a resistor  $R_g$  between two pins on the amplifier.

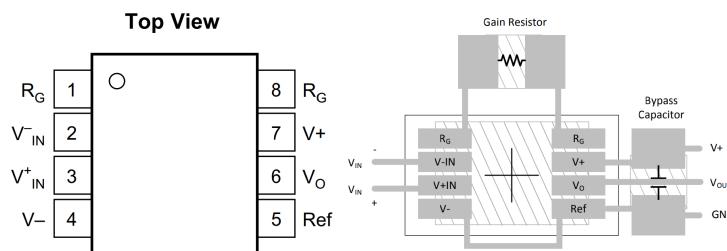


Figure 7: Amplifier for the load cell signal

The gain on this piece is easily calculated with following function.

$$\text{Gain} = 5 + \frac{80 \text{ k}\Omega}{R_G} \quad (1)$$

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 centerboard sensor

The height of the daggerboard is a crucial measurement in dinghy sailing. The daggerboard 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 totally watertight and concealed in the main centerplate. Other solutions can be to use some type of light sensor.

#### 3.3.1 Sensors

A suitable product has been found, which is very accurate and reliable, the Micro epsilon mk30<sup>7</sup>.

It's physical volume is small enough with a size of 3x5cm. As the product have some tight space constraints this can probably fit inside the case. This particular sensor is in the range of 2000kr. If no other solution works as intended this might be considered again. The main issues might be that the height of centerboard: One of the best implementations of a height sensor would be the use of a linear wire distance sensor. This particular 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 centerboard.



Figure 8: Linear draw wire sensor, Micro epsilon MK30

Light sensors have also been considered. This is 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. First, we looked into some Infrared radiation (IR)-sensors. They will send the signal in a widespread 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) 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. A new product called "micro-lidar" can be used, as it is both have a small form factor and is easy to implement. The chosen circuit is VL53L0X from Adafruit<sup>8</sup>.

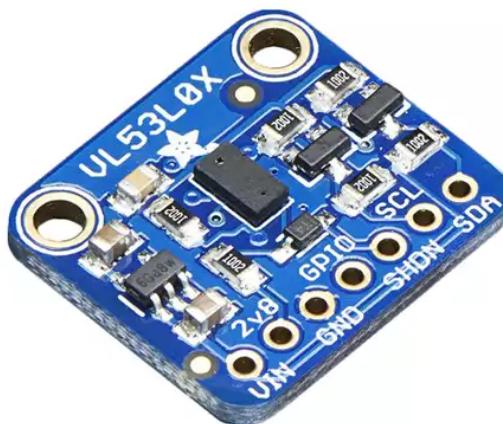


Figure 9: Time of flight,  $\mu$ LIDAR, distance sensor Adafruit VL53L0X

It can measure heights up one meters 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 plastic or even glass.

All the information about this sensor when used with a cover widow can be found in the specific datasheet: "VL53L0X ranging module cover window guidelines" ???. The maximum thickness of the cover window and the air gap between the sensor and the window is 2 mm. That's the parameters that need to be considered. The cover window can be some type of plastic or glass. To reflect the signal a detection plate is constructed and fastened to the top of the

daggerboard. This plate is level with the sensor and has a white underside for best performance of the sensor.



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 meters with great accuracy. It also works well when used behind a thin plastic cover. With a 1mm thin plastic cover the maximum height measurement is decreased to about one meter. In the case the sensor is mounted behind a thin epoxy window which was too thick at the beginning but was sanded down to a suitable thickness. The final window were then polished to a crystal clear finish for the best measurements. The measurement is heavily dependent on the finish of the window, if there are small imperfections on the window such as scratches the sensor can't take measurements.

As there will always be water around and on the center plate the light that is sent might get directed "wrong" if there are only some small water droplets in between the sensor and the panel that we want to reflect our light on.

This system can definitely 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 Printed Circuit Board (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<sup>4</sup>, see Figure 50.
- Circuit must be easy to test for both hardware and software errors. Even complete system needs this, since many of the sensors are very small and difficult to solder.
- 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

The main circuit board is built using the open source electronic design software KiCad<sup>3</sup>. 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<sup>3</sup> uses a four step process as follows:

1. Draw schematic:
  - (a) Place and connect all components.
  - (b) (Optional:) All non-standard components must be custom designed, this is referred to as “library”. Most sensors requires custom built libraries.
  - (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 51, 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 51, is designed to only allow for exactly +5V. This design is credited to, and derived from *Maxim Integrated*, Application note 760<sup>5</sup>. \*WRITE MORE HERE!\*

Non-standard library entries include following components:

Name	Type	Library	Used in last rev. <sup>a</sup>
A2235-H	GPS	a2235-h.lib	<b>YES</b>
FT232RL	USBtoSerial converter	ftdi.lib <sup>24</sup>	<b>YES</b>
HTS221	Humidity sensor	Custom_sensors.lib	<b>YES</b>
LM1117ADJ	Voltage Regulator	PowerSupply.lib <sup>23</sup>	<b>YES</b>
LPS22HB	Pressure sensor	Custom_sensors.lib	<b>YES</b>
LSM303AGR	Acc. <sup>b</sup> /Mag. <sup>c</sup> IMU	Custom_sensors.lib	<b>YES</b>
LSM303Cx	Acc./Mag. IMU	Custom_sensors.lib	NO
LSM6DSL	Acc./Gyro. <sup>d</sup> IMU	Custom_sensors.lib	<b>YES</b>
LSM9DS1	Acc./Gyro./Mag.	Custom_sensors.lib	NO
MCDTS2-4N	Tactile switch button	Custom_Switches.lib	<b>YES</b>

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

<sup>a</sup>revision

<sup>b</sup>Accelerometer

<sup>c</sup>Magnetometer

<sup>d</sup>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 USBtoSerial converter was broken, the I<sup>2</sup>C line was short circuited somewhere, the custom footprint for the GPS was mirrored, the BT worked, but poorly due to inferences 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 isolation's and trace widths had well thought out values. A 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.

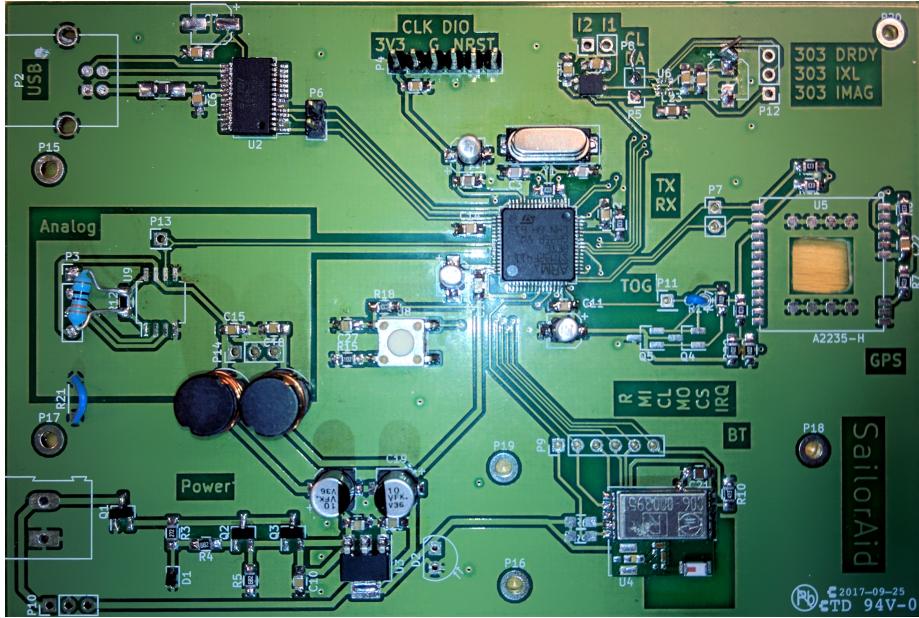


Figure 11: First (1) revision circuit board

#### 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 IMUs were currently powered were 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 was lost in the transition from revision one to two, as all net-labels in the schematic was 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 hot fixed 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.

#### 4.1.3 Third revision

With almost all problems solved in the second revision, what remained was some hotfix for an unexpected wiring error in the GPS; interference between I<sup>2</sup>C 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<sup>2</sup>C was added, and since the power level of this depends slightly on the power left in the batteries a logic level converter was designed to allow for a variable voltage level, see Figure 52. For the third revision all decisions have been made and the schematic can be seen as Figure 51, 52 and 53.

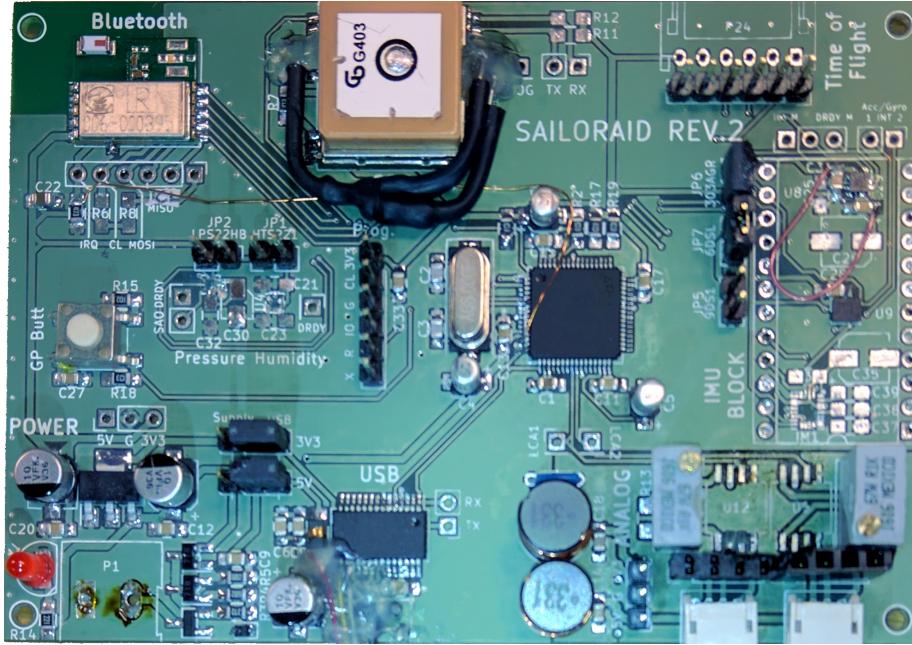


Figure 12: Second (2) revision circuit board

Following communicative devices will be used and act as “module areas”, i.e. these units will be the central units of the PCB layout. This was introduced in the second

- STM32F411RET - Microcontroller unit
- FT232RL - USBto serial converter (USART)
- A2235-H - GPS unit (UART)
- SPBTLE-RF - BT connection unit (SPI)
- Sensors ( $I^2C$ ):
  - LSM303AGR - Accelerometer & 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 are moved to one spot ensuring the shortest possible  $I^2C$  data line. Ground vias were added around communication traces to prevent unwanted interference, ground vias were also added to prevent antennas<sup>1</sup>. The power is placed in the

<sup>1</sup> 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.

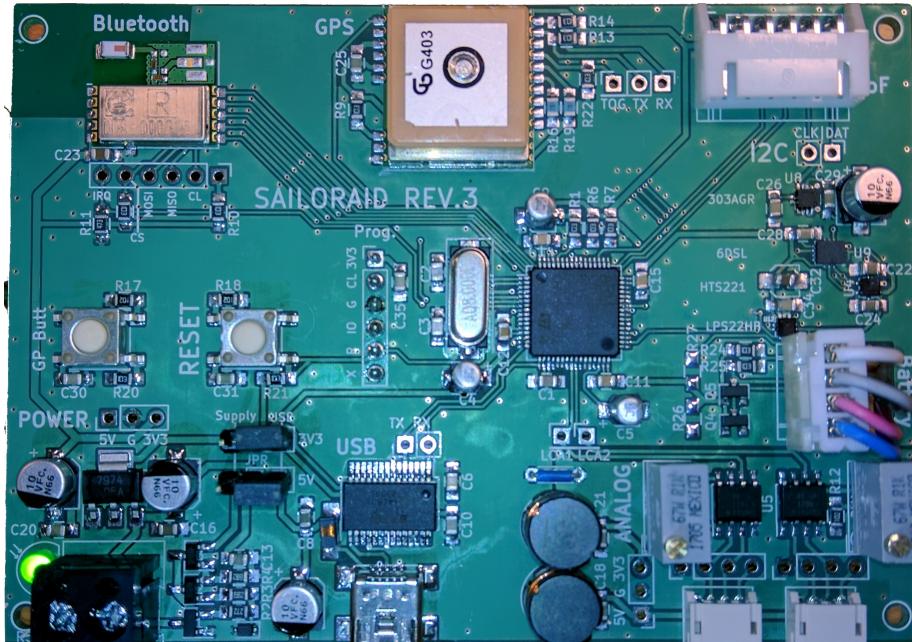


Figure 13: third (3) revision circuit board

bottom right, since this is as far as possible from everything else. USB is placed next to the power, as power can be opted to be drawn from USB rather than battery. Analog plane with the load cells are put right next to the power to ensure the star-ground<sup>2</sup> to be as close as possible to the power-source allowing for as little noise as possible. BT communication devices are rather fragile, it is 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 are 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 are placed in the middle-left, between the power source and the BT unit. 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.

## 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

<sup>2</sup>A common node ground connecting all grounds together to ensure the same set working level

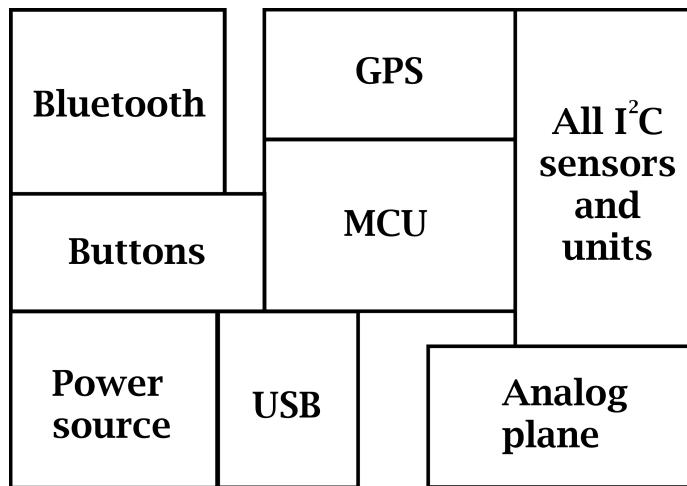


Figure 14: Simple model of third (3) revision circuit board areas

number of GPIO ports, useful peripherals ( $I^2C$ , SPI, timers, etc) and a high clock speed of up to  $100MHz$  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 in a compact form factor. It did however have some requirements on how it could be initialized, otherwise risking corruption of the GPS modules firmware.

#### **4.3 results (TiP)**

stuff4

#### **4.4 discussion (TiP)**

stuff5

## 5 Casing

LUNDBERG, JOSEF ( , )

This system has some different types of components and sensors, which all need to be housed in a watertight casing for security 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 will be revised and worked on over the whole length of the course, redefining and remodeling the construction over time. The cases are constructed with the use of the program Fusion 360, which is a free to use CAD software. They are then produced by using a 3D printer, which is a fast and easy way of try out different ideas and models.

### 5.1 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 daggerboard 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 is that the case is too big for any of the 3D-printers that we had access to. 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.

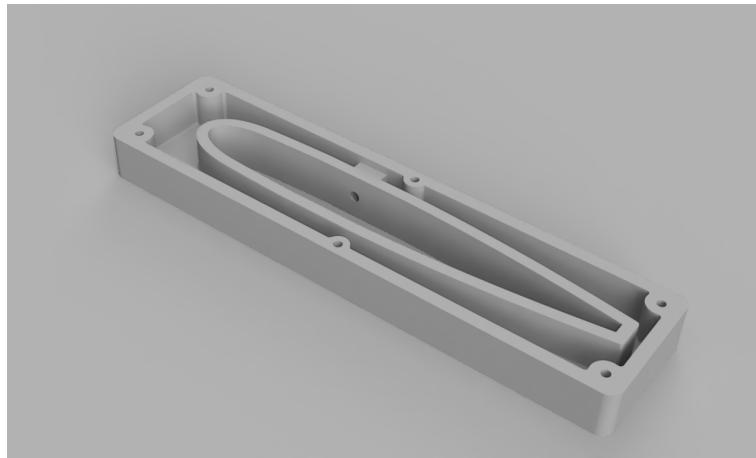


Figure 15: First revision of the case

### 5.2 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 can thus be made on a 3D-printer. It still has the problem of having very little space for all the components.

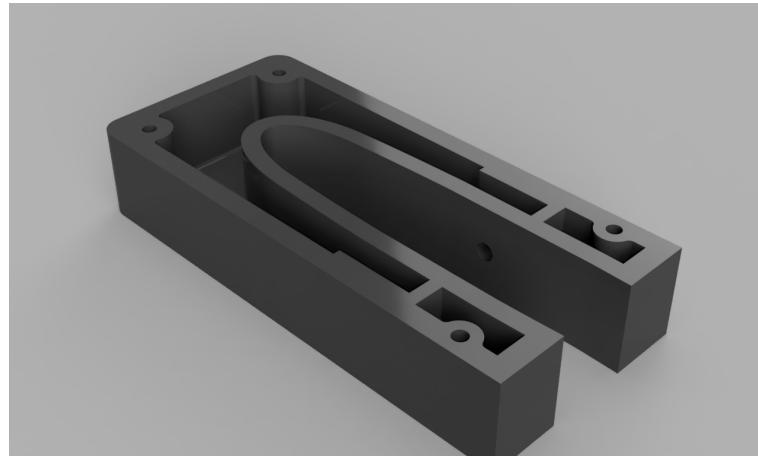


Figure 16: Second revision of the case

### 5.3 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.

This case still had some problems, it did neither include a holder for the distance sensor nor being watertight. But this version was also good in many aspects. The size of the part was enough to accommodate for the PCB, batteries and all the sensors. This part 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.

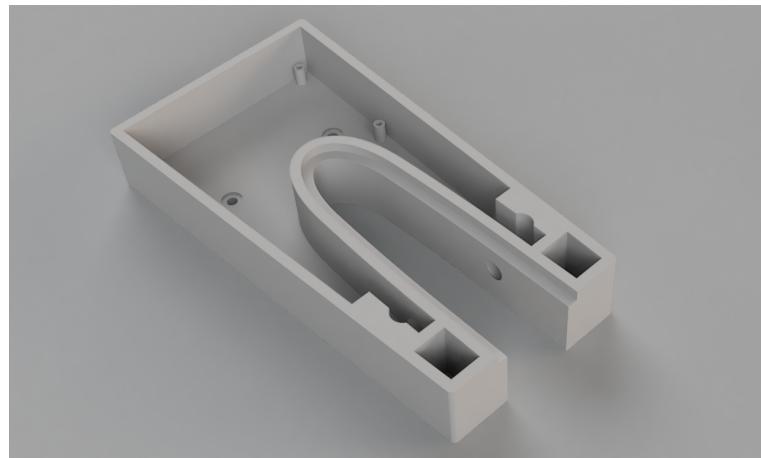


Figure 17: Third revision of the case

#### 5.4 Fourth revision

The fourth revision have all the features and can hold all the electronics. It's been 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 has been placed at the underside of the model to get a better water seal. The access panel is a big plate covering the most part of the underside of the model. To ensure it's watertight there a seal between the panel and the casing itself could be implemented. 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 Time of Flight sensor.

#### 5.4.1 The prototype

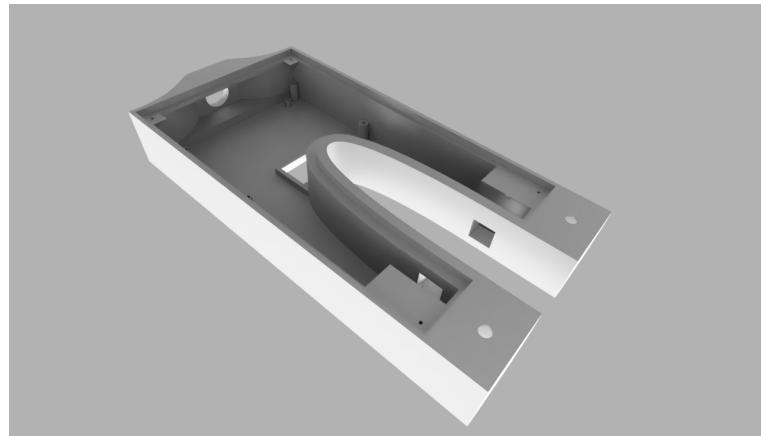


Figure 18: Final version of the case

#### 5.5 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 made to this is the rubber seal between the load cell and the steel beads used to measure the forces.

## 6 Battery charging circuitry

GRAPE, ELIAS (LUNDBERG, JOSEF)

## 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

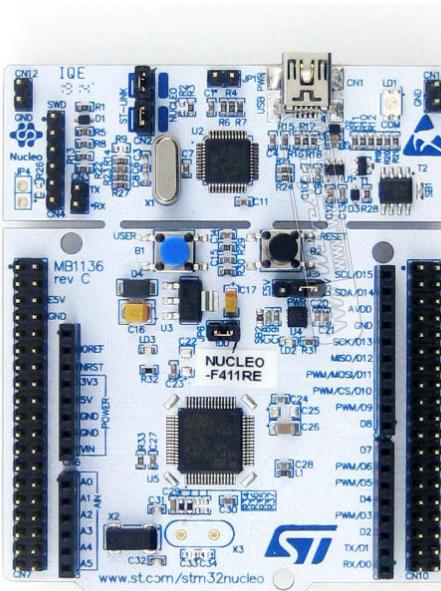


Figure 19: NUCLEO-F411RE development board for the ARM Cortex-M microcontroller STM32F411RE.

For rapid prototyping and firmware development purposes, the NUCLEO-F411RE development board seen in figure 19 was used. This board contains break-out pins for the ARM Cortex-M microcontroller STM32F411RE, a UART to USB bridging circuit and general purpose Light-emitting diode (LED)s and buttons. It is compatible with various Arduino shields as well as expansion boards developed by ST.

In order to speed up firmware development, the STM32CubeMX<sup>2</sup> 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 clocks, peripherals, interrupts and similar. It is controlled by a graphical interface 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 sensor.

### 7.1.1 Bluetooth



Figure 20: X-NUCLEO-IDB05A1 Bluetooth Low Energy evaluation board for the STM32 Nucleo

For prototyping, the Bluetooth evaluation board X-NUCLEO-IDB05A1<sup>14</sup> seen in figure 20 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<sup>6</sup> 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<sup>13</sup>. 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 quite difficult to get working. Instead it was decided that blocking SPI communication were to be used, since this was much 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.

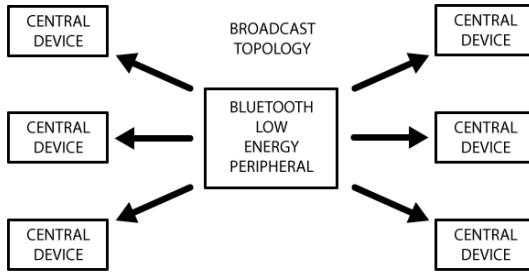


Figure 21: Bluetooth GAP topology.

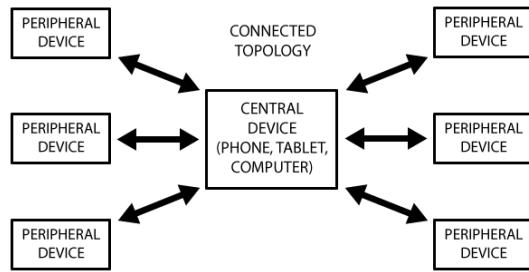


Figure 22: Bluetooth GATT topology.

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. Figures 22 and 21 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.

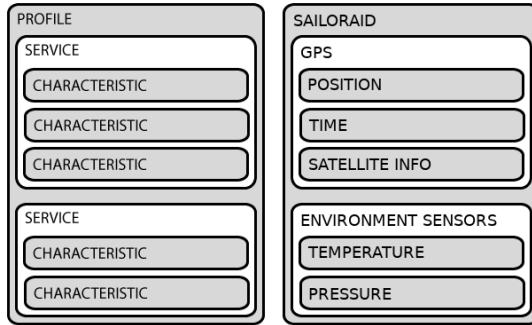


Figure 23: Bluetooth GATT transaction profile with usage example.

The GATT protocol performs transactions by nested structures called Profiles, Services and Characteristics. An example of this structure can be seen in figure 23. 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

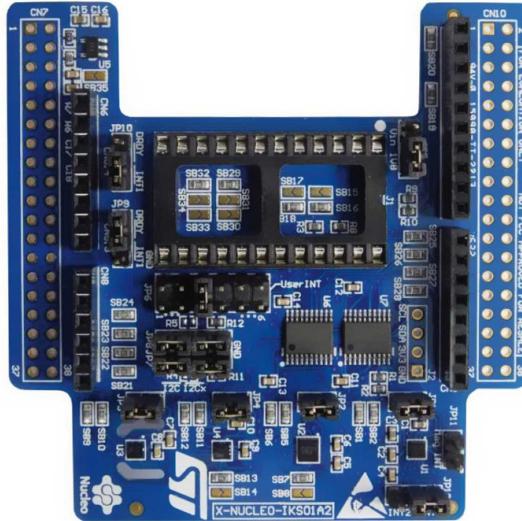


Figure 24: X-NUCLEO-IKS01A2 motion Microelectromechanical systems (MEMS) and environmental sensor expansion board for the STM32 Nucleo

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<sup>15</sup> evaluation board (figure 24) 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<sup>12</sup> library developed by ST was used. This library implemented the I<sup>2</sup>C 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<sup>2</sup>C 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) (section 7.1.3) was used.

### 7.1.3 Madgwick AHRS

Madgwick AHRS<sup>16</sup> 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 more performant alternative to the Kalman filter approach. 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 (rotation vectors with four elements), but can convert it to Euler angles, which can be more easily used.

The mathematical background of this algorithm is quite complicated and outside the scope of this report, see the official report<sup>17</sup> for more details.

### 7.1.4 GPS



Figure 25: Maestro A2235-H GPS module with built-in antenna

The GPS module used in this project, A2235-H by Maestro<sup>18</sup> could communicate with the MCU through either I<sup>2</sup>C 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 (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,

```
PGPSA,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. PGPSA 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.

Several NMEA parsing libraries are freely available on the web. The one chosen for this project was called Libnmea<sup>19</sup> and allowed the sentences to be automatically recognized, parsed and stored into easy to use C structures.

### 7.1.5 Range sensor

In order to communicate with the VL53L0X range finder sensor, another software library called X-CUBE-53L0A1<sup>20</sup> 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 center 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 (section 8) and measuring water wave properties, it was important to be able to log data at a reasonably high frequency. 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 bps should be used, with eight data bits, no parity and one stop bit. Commands can be entered with 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.

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 a 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	$\approx 1$	$\approx 1$	$\approx 1$

Table 3: System input/output rates in Hz. BT stands 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<sup>2</sup>C and SPI data busses. Problems such as noise or framing errors 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)<sup>33</sup> model of the system (Fig. 26) was developed for an overview of the system implementation. The android system uses activities<sup>32</sup> to display content to the device screen. These activities have a life-cycle (Fig. 27) 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.

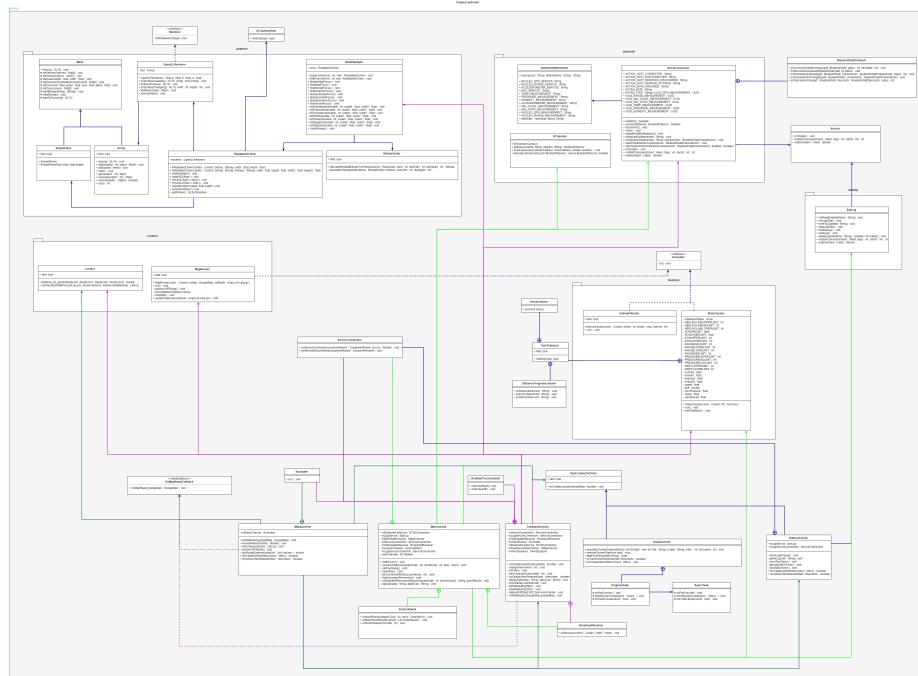


Figure 26: UML model

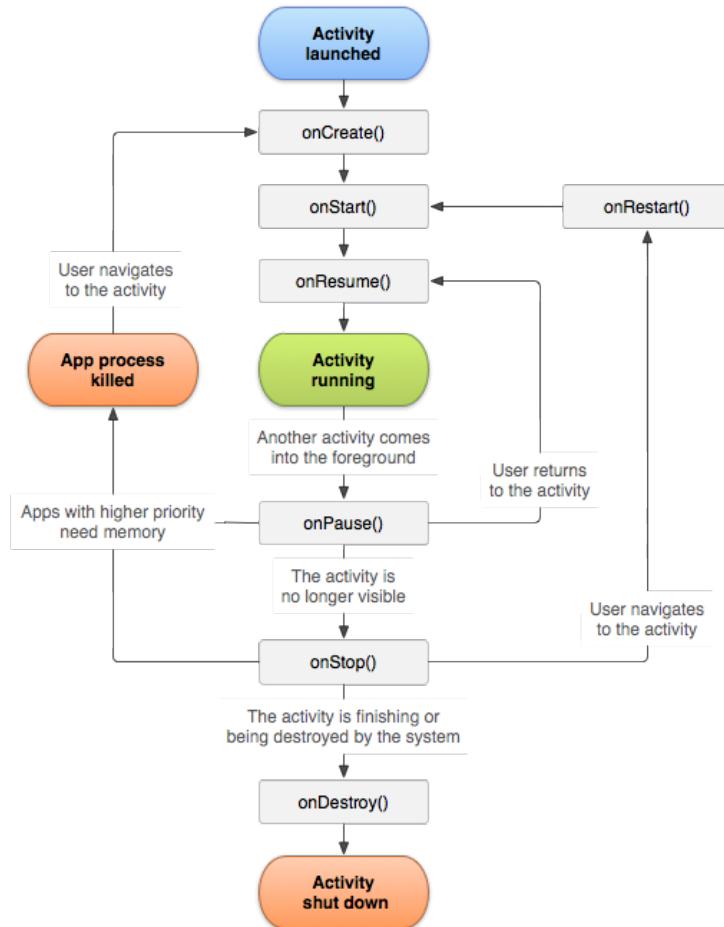


Figure 27: Activity lifecycle

The Bluetooth connection and data logging class is implemented as a service<sup>21</sup>. 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 Fig. 28.

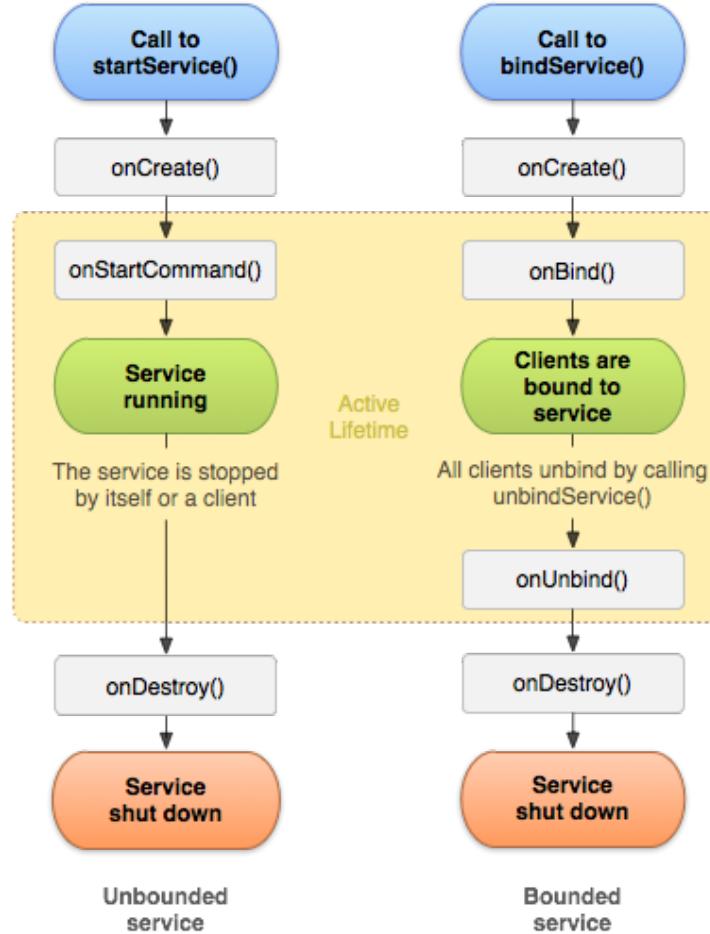


Figure 28: Activity lifecycle

### 7.2.2 Feedback

For the feedback states in the application, there are some approximations, and simplifications made for easier implementation. As explained in the Physics of sailing chapter 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 centerboard |
| Drift   | - The dinghy is drifting perpendicular to the bearing while the centerboard is in an upper position                |
| Heel    | - The dinghy has high heel angle   |
| Reefing | - The dinghy has high heel angle and the pressure on the centerboard is high                                       |

Wrspeed - The dinghys speed is above 65.45kn

Lrspeed - The dinghys speed is above 16.8kn

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

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

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

Runninglow - The dinghy is sailing directly windwards with centerboard 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

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

30 **Pressure** moves a pin along a colored bar to represent high or low pressure applied on the centerboard.

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

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

33 **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 it's set navigational reference.

34 **Speed** displays a speedometer from a classical Swedish vehicle<sup>30</sup> with a movable bar representing speed over ground.

35 **Height** of the centerboard is represented with a visual centerboard moving up and down along a graphical ruler.

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

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



Figure 29: Incline feedback view

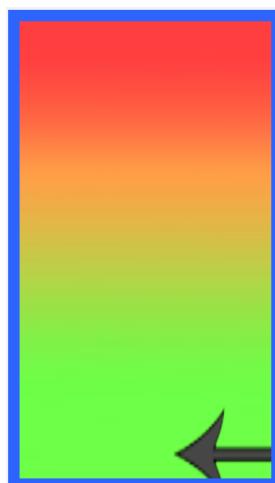


Figure 30: Pressure feedback view

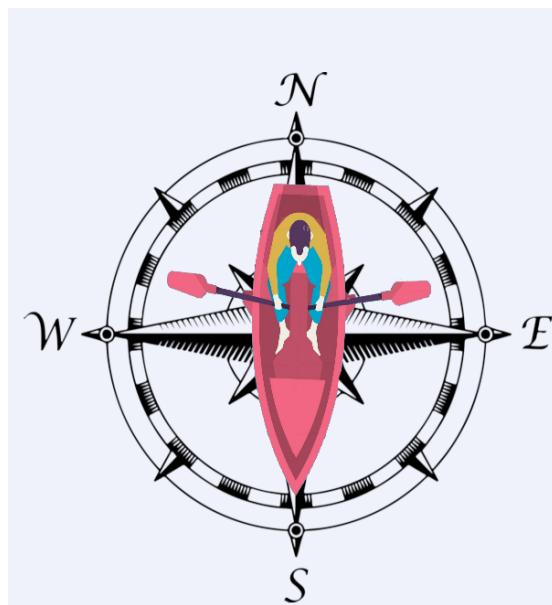


Figure 31: Bearing feedback view

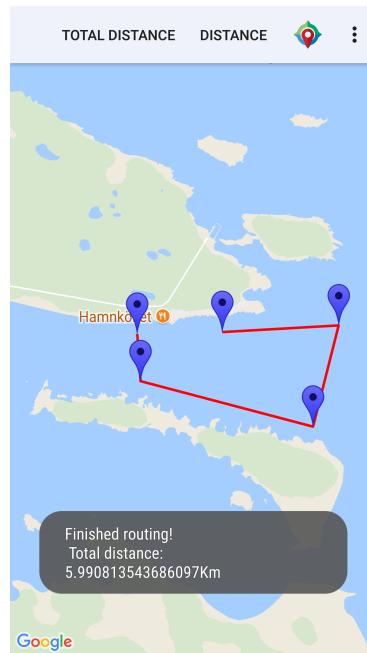


Figure 32: Map feedback view



Figure 33: Drift feedback view

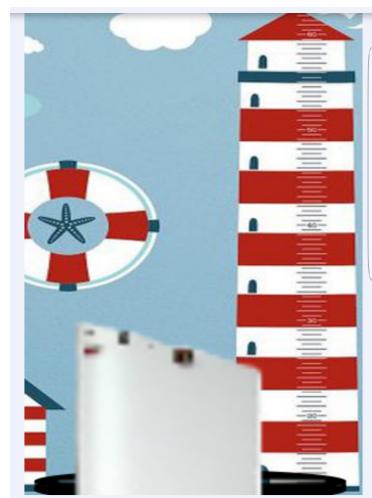


Figure 34: Centerboard height view



Figure 35: Speed over ground view



Figure 36: Wave period feedback view

**Ship Adrift!  
Lower Centerboard more!**

Figure 37: Text feedback view

The figures were chosen at a development stage and improvements can easily be made by adding different Portable Network Graphics (PNG)<sup>31</sup> files in android studio. They are implemented as GLSurfaceView<sup>34</sup> 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 38. 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.

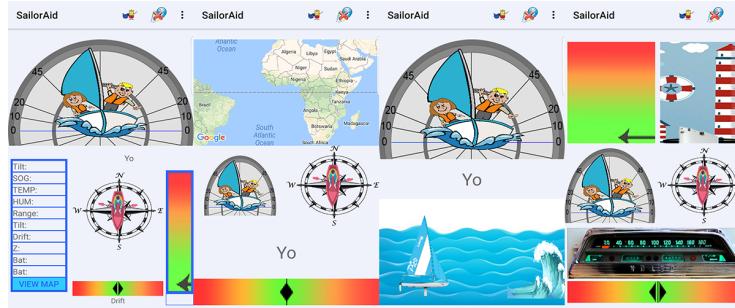


Figure 38: Different layouts

#### 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 TextToSpeech<sup>35</sup> class is implemented as an inner class in StateChecker. To prevent the speech to be interrupted preemptively an UtteranceProgressListener<sup>36</sup> 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 a 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 (Fig. 39). More information from the log can be analyzed by reading graphs (Fig. 40) 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 39: Log data summary

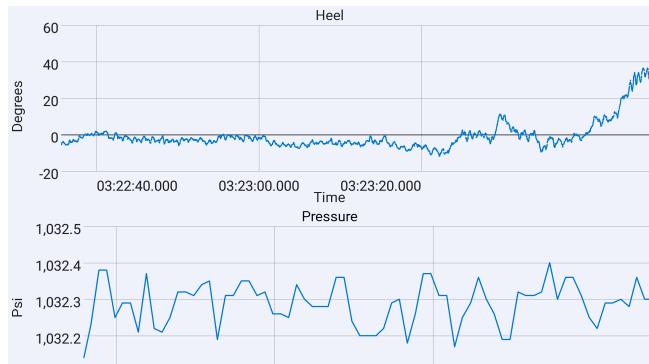


Figure 40: Log graphs

### 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 MainActivity 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<sup>21</sup>. 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)<sup>37</sup>, 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 sailor 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*<sup>25</sup> is derived and the new estimated position is calculated with

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

where  $R$  is the Earth's radius,  $d$  is the distance traveled,  $\theta$  is ships bearing,  $\lambda_1$  and  $\varphi_1$  is the point of origin in longitude and latitude. This new estimated position  $\lambda_2$  and  $\varphi_2$  is then compared to the next positional data from the GPS and the distance between these points is calculated using *Equirectangular projection*<sup>26</sup>

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

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*<sup>27</sup>. 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

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 it 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)

## 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<sup>45</sup>. Sensor fusion theory is not only found in the living species it is found 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<sup>39</sup>. 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.

### 8.1 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.<sup>43</sup>

### 8.2 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<sup>39</sup>. It's popular to use due to its efficiency when calculating predictions.<sup>42</sup>

### 8.3 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  $\omega_e$ .

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 41, 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,  $\omega_{en}^n$ , is directed by the motion of point P with respect to the Earth and is referred to the transport rate.

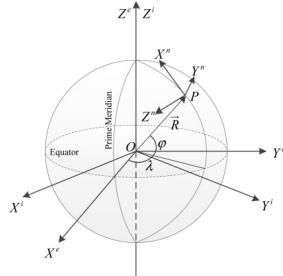


Figure 41: Inertial, Earth and Navigation frame.

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 IMU is mounted in. The body, in this case, is referred to the dinghy, see Figure 43.

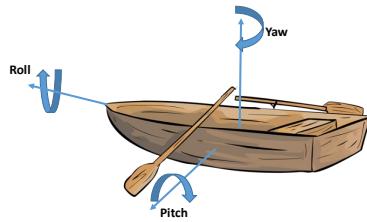


Figure 42: Body frame

- Yaw ( $\psi$ ): 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<sup>40</sup>.
- Pitch ( $\theta$ ): 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<sup>40</sup>.
- Roll ( $\phi$ ): 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<sup>40</sup>.

## 8.4 Transformation Between Frames

Referring to Figure 41 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)<sup>41</sup>. Where  $\lambda$  and  $\varphi$  is the latitude and longitude, respectively. Then the transformation between the two frames can be done using the Direction Cosine Matrix (DCM), which is defined as<sup>41</sup>.

$$C_n^e = R_z(-\lambda - 90)R_x(\varphi - 90) \quad (2)$$

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 Eq. (2)

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

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

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

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

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

Using Eq. (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<sup>41</sup>

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

where

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

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

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

Solving Eq. (7) with (8), (9) and (10)

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

(12)

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

Where  $\psi$ ,  $\theta$  and  $\phi$  are the Euler angles as described earlier. Expressing the matrix above in a more compact form.

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

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<sup>41</sup>. 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<sup>41</sup>.

$$C_b^n = \begin{bmatrix} (q_1^2 + q_2^2 - q_3^2 - q_4^2) & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 - q_1q_4) & (q_1^2 - q_2^2 + q_3^2 - q_4^2) & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & (q_1^2 - q_2^2 - q_3^2 + q_4^2) \end{bmatrix} \quad (15)$$

Where  $q_1$ ,  $q_2$ ,  $q_3$  and  $q_4$  can be derived using the Euler angles from Eq. (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<sup>41</sup>

$$\bar{\omega}_{ie}^e = [0 \ 0 \ \omega_e]^T. \quad (16)$$

Where  $\omega_e$  is the angular velocity of the Earth and has a value of  $7.2921158 \times 10^{-5}$  rad/s<sup>41</sup>. Using the projection matrix Eq. (2) its possible to project  $\bar{\omega}_{ie}^i$  onto the  $n$ -frame

$$\bar{\omega}_{ie}^n = C_e^n \bar{\omega}_{ie}^e = [\omega_e \cos(\varphi) \quad 0 \quad -\omega_e \sin(\varphi)]^T. \quad (17)$$

$\bar{\omega}_{en}^n$  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

$$\bar{\omega}_{en}^n = [\dot{\lambda} \cos(\varphi) \quad -\dot{\varphi} \quad -\dot{\lambda} \sin(\varphi)]^T. \quad (18)$$

Where  $\dot{\lambda} = v_e/(R_N + h)\cos(\varphi)$  and  $\dot{\varphi} = v_n/(R_E + h)$ <sup>41</sup>, 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  $R_N = R_0(1 - e^2)/(1 - e^2 \sin^2 \varphi)^{3/2}$ <sup>41</sup>.  $R_E$  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,  $\varphi$  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 Eq. (18) with the new expressions.

$$\bar{\omega}_{en}^n = [v_e/(R_E + h) \quad -v_n/(R_N + h) \quad -v_e \tan(\varphi)/(R_N + h)]^T \quad (19)$$

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

$$\bar{\omega}_{in}^n = [\omega_e \cos(\varphi) + v_e/(R_E + h) \quad v_n/(R_N + h) \quad -\omega_e \sin(\varphi) + v_e \tan(\varphi)/(R_N + h)]^T \quad (20)$$

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

## 8.5 Inertial Navigation Equation

Describing the position of the dinghy in the  $n$ -frame is done by<sup>41</sup>.

$$\bar{r}^n = [\varphi \quad \lambda \quad h]^T. \quad (21)$$

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.

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

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

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

Thus the dynamics describing the system can be expressed as<sup>41</sup>.

$$\dot{r}_n = D^{-1}v_n \quad (24)$$

$$\dot{v}_n = C_b^n \bar{f}^b - (2\omega_{ie}^n + \omega_{en}^n) \times \bar{v}_n + \bar{g}^n \quad (25)$$

$$\dot{C}_b^n = C_b^n (\Omega_{ib}^b - \Omega_{in}^b) \quad (26)$$

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  $\Omega$  is the skew matrix of  $\omega$ , more precise  $\Omega_{ib}^b$  the skew-matrix of the outputs of the strapdown gyroscopes and  $\Omega_{in}^b$  is the skew-matrix of Eq. (20). Skew-matrix is defined as the cross product between the vectors.

## 8.6 INS mechanization

Since the INS has to work in discrete time domain, Eq. (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 100Hz, this means that  $\Delta t_k = \Delta k_{k-1} - t_k$ , this to improve the accuracy. The discrete dynamics are<sup>43</sup>, Using Eq. (24),(25) and (26).

$$\bar{r}_{k+1}^n = \bar{r}_k + 0.5D^{-1}(\bar{v}_k^n + \bar{v}_{k+1}^n)\Delta t \quad (27)$$

$$v_{k+1}^n = \bar{v}_k + \Delta \bar{v}_{k+1}^n \quad (28)$$

$$\dot{C}_b^n = C_b^n (\Omega_{ib}^b - \Omega_{in}^b)\Delta t \quad (29)$$

where

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

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

The Navigation frame Inertial Navigation System can now be seen in Figure 43, Where its possible to examine all the steps which were derived earlier, from the IMU to the output.

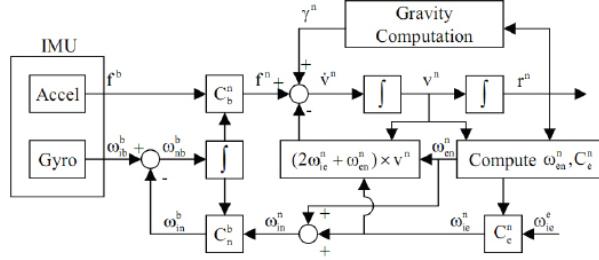


Figure 43: The Inertial Navigation Frame

## 9 Implementation of Sensor Fusion Using a Kalman Filter.

### Perturbation Analysis

Since the differential equations describing velocity, position, and the attitude, DCM are non-linear, utilize a perturbation series to minimize the errors<sup>48</sup>.

$$\hat{r}^n = r^n + \delta r^n \quad (31)$$

$$\hat{v}^n = v^n + \delta v^n \quad (32)$$

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

Where  $E^n$  is the skew matrix and given as.

$$\begin{bmatrix} 0 & -e_D & e_E \\ e_D & 0 & -e_N \\ -e_E & e_N & 0 \end{bmatrix} \quad (34)$$

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

#### 9.0.1 Error Dynamics Position

Perturbing (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

$$\delta \dot{r}^n = F_{rr} \delta r^n + F_{rv} \delta v^n. \quad (35)$$

Where

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

and

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

### 9.0.2 Error Dynamics Velocity

The same is used when deriving the perturbation for the velocity, Eq. (32), here, velocity is described by forces and gravity acting on the dinghy. Using Eq. (25), and perturbation Eq. (32), the following is obtained<sup>41</sup>.

$$\delta \dot{\bar{v}}^n = \bar{v}_n \times \underbrace{(2\delta \bar{\omega}_{ie}^n + \delta \bar{\omega}_{en}^n)}_I + \delta \bar{g}^n - \underbrace{(2\bar{\omega}_{ie}^n + \bar{\omega}_{en}^n)}_{II} \times \delta \bar{v}^n + (\bar{f}^n \times) e^n + C_b^n \delta \bar{f}^n \quad (38)$$

where  $II$  is expressed as follow, using Eq. (20).

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

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

$$2\delta \bar{\omega}_{ie}^n + \delta \bar{\omega}_{en}^n = \delta \Omega_r \delta \bar{r}^n + \delta \Omega_r \delta \bar{v}^n \quad (40)$$

where

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

and

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

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

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

Then expressing Eq. (38) as

$$\delta \bar{v}^n = F_{vr} \delta \bar{r}^n + F_{vv} \delta \bar{v}^n + (\bar{f}^n \times) e^n + C_b^n \delta \bar{f}^n \quad (44)$$

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

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

and

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

### 9.0.3 Error Dynamics Attitude

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

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

Then relate the above equation with Eq. (33)

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

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

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

or in vector form

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

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

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

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

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

Substituting Eq. (52) into Eq. (50), the following is obtained

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

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

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

Where  $F_{er}$  and  $F_{ev}$  is

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

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

## 9.1 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 44.

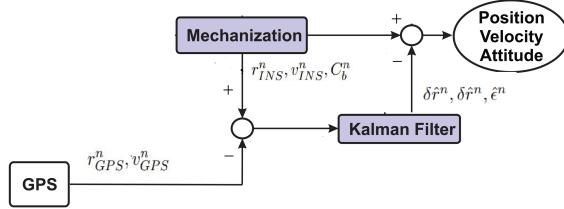


Figure 44: The feedback-method which is used in the system.

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

$$\hat{x} = F\bar{x} + G\bar{u} \quad (57)$$

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  $\hat{x}$  is the estimated state vector.

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

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

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

$$G = \begin{bmatrix} -C_b^n & 0 \\ 0 & C_b^n \end{bmatrix}, \quad \bar{u} = \begin{bmatrix} \delta\bar{\omega}_{ib}^b \\ \delta\bar{f}^b \end{bmatrix} \quad (59)$$

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

$$Q = \text{diag} [\sigma_{ax}^2 \quad \sigma_{ay}^2 \quad \sigma_{az}^2 \quad \sigma_{gx}^2 \quad \sigma_{gy}^2 \quad \sigma_{gz}^2]^T \quad (60)$$

where  $\sigma_{a(x,y,z)}^2$  and  $\sigma_{g(x,y,z)}^2$  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

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

Here  $\Phi_k$  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.

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

Since the sample interval,  $\Delta t$  is very small in this case Eq. (62) can be approximated using Taylor approximation

$$\Phi_k = e^{F\Delta t} \approx I + F\Delta t. \quad (63)$$

From Eq. (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<sup>46</sup>. Then the covariance matrix which is associated with  $\bar{w}_k$  is<sup>46</sup>

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

and  $Q_k$  can then be approximated using a first order of the discrete transition matrix<sup>47</sup>

$$Q_k \approx \Phi_k G Q G^T \Phi_k^T. \quad (65)$$

If Eq. (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 don't 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. Eq. (61) and the following observation matrices<sup>47</sup>.

$$z_k = H_k \bar{x}_k + \bar{e}_k \quad (66)$$

where  $z_k$  is the measurement vector and  $e_k$  is random measurement noise, with following characteristics<sup>46</sup>.

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

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

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

then the state vector is updated

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

the last step is updating the covariance matrix

$$P_k = (I - K_k H) P_k^- \quad (70)$$

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

$$\hat{x}_k^- = \hat{x}_k \Phi_k \quad (71)$$

then updating its covariance

$$P_k^- = \Phi_k P_k \Phi_k^T + Q_k \quad (72)$$

Where  $(\bar{\cdot})$  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

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

The measurement matrix  $H_k$  is defined such

$$H_k = \begin{bmatrix} I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & I_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \quad (74)$$

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

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

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 won't 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 45.

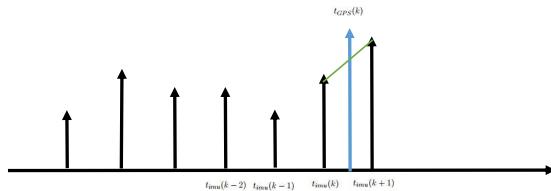


Figure 45: Different update sequences IMU and GPS

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

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

and the equations for the velocities

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

## 9.2 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's better to use the INS connected to a Kalman Filter, this can be seen in Figure 46. 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.

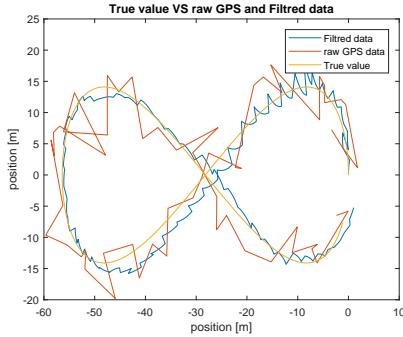
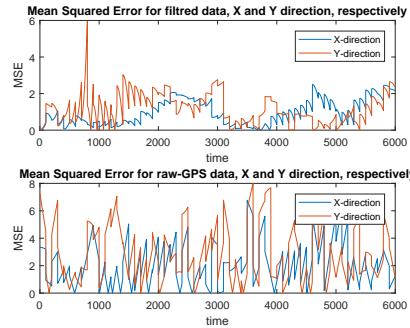


Figure 46: True VS. Filtered and raw GPS data.

The Mean Squared Error, MSE for the position in XY plane, can be seen in Figure 47 for both INS and raw GPS data we can see that the error is lower for the INS data, in both cases.

Figure 47: 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 48. In this case, the GPS data is more accurate.

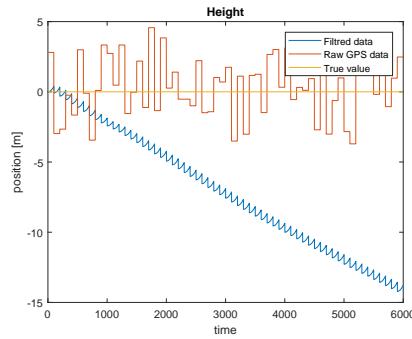
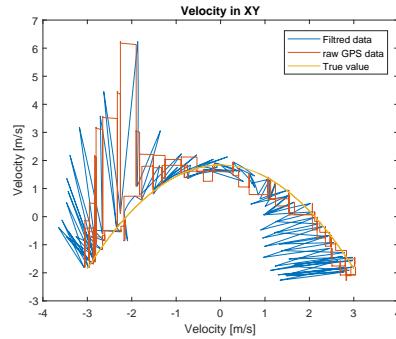


Figure 48: Height. True VS. Filtered and raw GPS data.

The velocities in the  $XY$  plane does not represent the true system in a good manner either this can be seen in Figure 49. In this case, the GPS data is more accurate as well.

Figure 49: The velocities in  $XY$  plane.

### 9.3 Discussion/Future Work

Depending on what the user wants to measure, e.g. position in two dimensions, height or velocity, different sensors should be used. In the first 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 exists 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<sup>43</sup>

- 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]
- [2] <http://www.st.com/en/development-tools/stm32cubemx.html>
- [3] KiCad open source electronics design, [kicad-pcb.org/](http://kicad-pcb.org/)
- [4] <https://www.autodesk.com/products/fusion-360/overview>
- [5] Overvoltage and Reverse-voltage Protection in Automotive Systems, Application note 760, *Maxim integrated*, Apr 02, 2002, <https://www.maximintegrated.com/en/app-notes/index.mvp/id/760>
- [6] <http://www.st.com/en/embedded-software/x-cube-ble1.html>
- [7] [https://www.micro-epsilon.se/displacement-position-sensors/draw-wire-sensor/MK\\_Serie/](https://www.micro-epsilon.se/displacement-position-sensors/draw-wire-sensor/MK_Serie/)
- [8] <https://www.adafruit.com/product/3317>
- [9] <http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=FX19&DocType=DS&DocLang=English>
- [10] <http://www.ti.com/lit/ds/symlink/ina126.pdf>
- [11] [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](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)
- [12] <http://www.st.com/en/embedded-software/x-cube-mems1.html>
- [13] <http://play.google.com/store/apps/details?id=com.st.blunrg>
- [14] <http://www.st.com/en/ecosystems/x-nucleo-idb05a1.html>
- [15] <http://www.st.com/en/ecosystems/x-nucleo-iks01a2.html>
- [16] <http://x-io.co.uk/open-source-imu-and-ahrs-algorithms>
- [17] [http://x-io.co.uk/res/doc/madgwick\\_internal\\_report.pdf](http://x-io.co.uk/res/doc/madgwick_internal_report.pdf)
- [18] [http://update.maestro-wireless.com/GNSS/A2235-H/Maestro\\_GPS\\_Evaluation\\_Kit\\_EVA2235\\_H\\_User\\_Manual\\_V01.pdf](http://update.maestro-wireless.com/GNSS/A2235-H/Maestro_GPS_Evaluation_Kit_EVA2235_H_User_Manual_V01.pdf)
- [19] <http://github.com/jacketizer/libnmea>
- [20] [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](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)
- [21] <https://developer.android.com/reference/android/app/Service.html>
- [22] <https://www.movable-type.co.uk/scripts/latlong.html>

- [23] KiCad library entry for voltage regulator component lm1117 by *ObKo*, 2012 <https://github.com/ObKo/kicad-libraries/blob/master/libraries/lm1117.lib>
- [24] KiCad library entry for USBtoSerial component FT232RL by *jbaker0428*, Oct 3, 2010 <https://github.com/jbaker0428/Kicad-Libraries/blob/master/library/ftdi.lib>
- [25] [https://en.wikipedia.org/wiki/Rhumb\\_line](https://en.wikipedia.org/wiki/Rhumb_line)
- [26] [https://en.wikipedia.org/wiki/Equirectangular\\_projection](https://en.wikipedia.org/wiki/Equirectangular_projection)
- [27] [https://en.wikipedia.org/wiki/Haversine\\_formula](https://en.wikipedia.org/wiki/Haversine_formula)
- [28] <https://en.wikipedia.org/wiki/Sailing>
- [29] <http://newt.phys.unsw.edu.au/~jw/sailing.html>
- [30] [https://sv.wikipedia.org/wiki/Volvo\\_Amazon](https://sv.wikipedia.org/wiki/Volvo_Amazon)
- [31] <https://sv.wikipedia.org/wiki/PNG>
- [32] <https://developer.android.com/reference/android/app/Activity.html>
- [33] [https://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language)
- [34] <https://developer.android.com/reference/android/opengl/GLSurfaceView.html>
- [35] <https://developer.android.com/reference/android/speech/tts/TextToSpeech.html>
- [36] <https://developer.android.com/reference/android/speech/tts/UtteranceProgressListener.html>
- [37] <https://developers.google.com/maps/>
- [38] D.L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the*. IEEE, 85(1):6 –23, jan 1997
- [39] B.R. Grover H. Patrick. Introduction to random signals and applied Kalman filtering. 1997 *BOKEN*.
- [40] Society of Naval Architects and Marine Engineers (SNAME), "Principles of Naval Architecture", 1989, Vol. III, *SNAME*.
- [41] Dr. Oliver Nelles  
nonlinear system identification.
- [42] Duygun, M., Kutlu, L. & Sickles, R.C. J Prod Anal (2016) 46: 155. <https://doi.org/10.1007/s11123-016-0477-z>
- [43] 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

- [44] [https://edg.uchicago.edu/tutorials/load\\_cell/](https://edg.uchicago.edu/tutorials/load_cell/)
- [45] D.L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, jan 1997
- [46] Probability and Random Processes with Applications to Signal Processing, 4/E (2012). Henry Stark, John W Woods. Pearson Higher Education.
- [47] Estimation, Control, and the Discrete Kalman Filter, 1989. Donald E. Catlin
- [48] [https://en.wikipedia.org/wiki/Perturbation\\_theory](https://en.wikipedia.org/wiki/Perturbation_theory)

## 10 APPENDIX

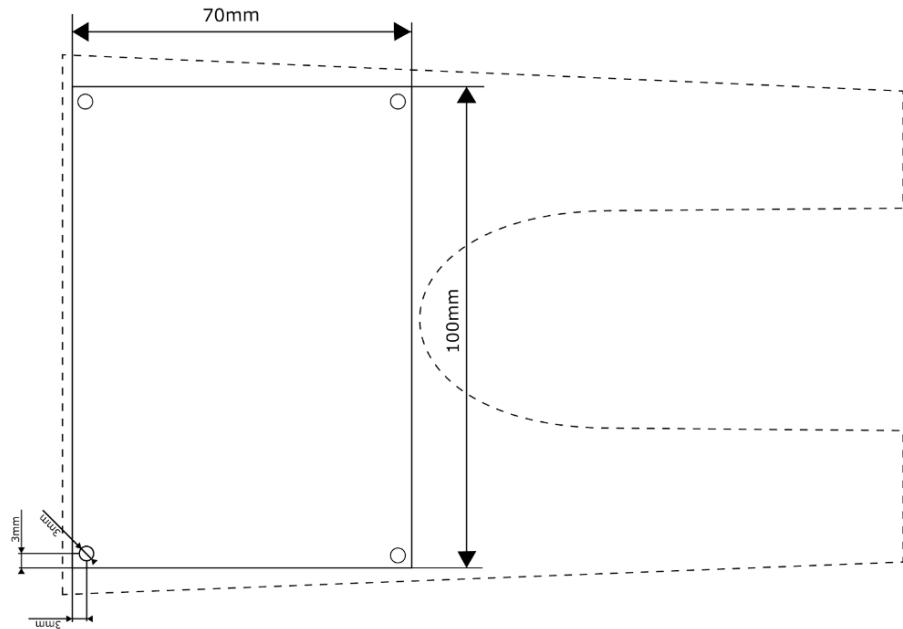


Figure 50: System enclosure shape and dimensions

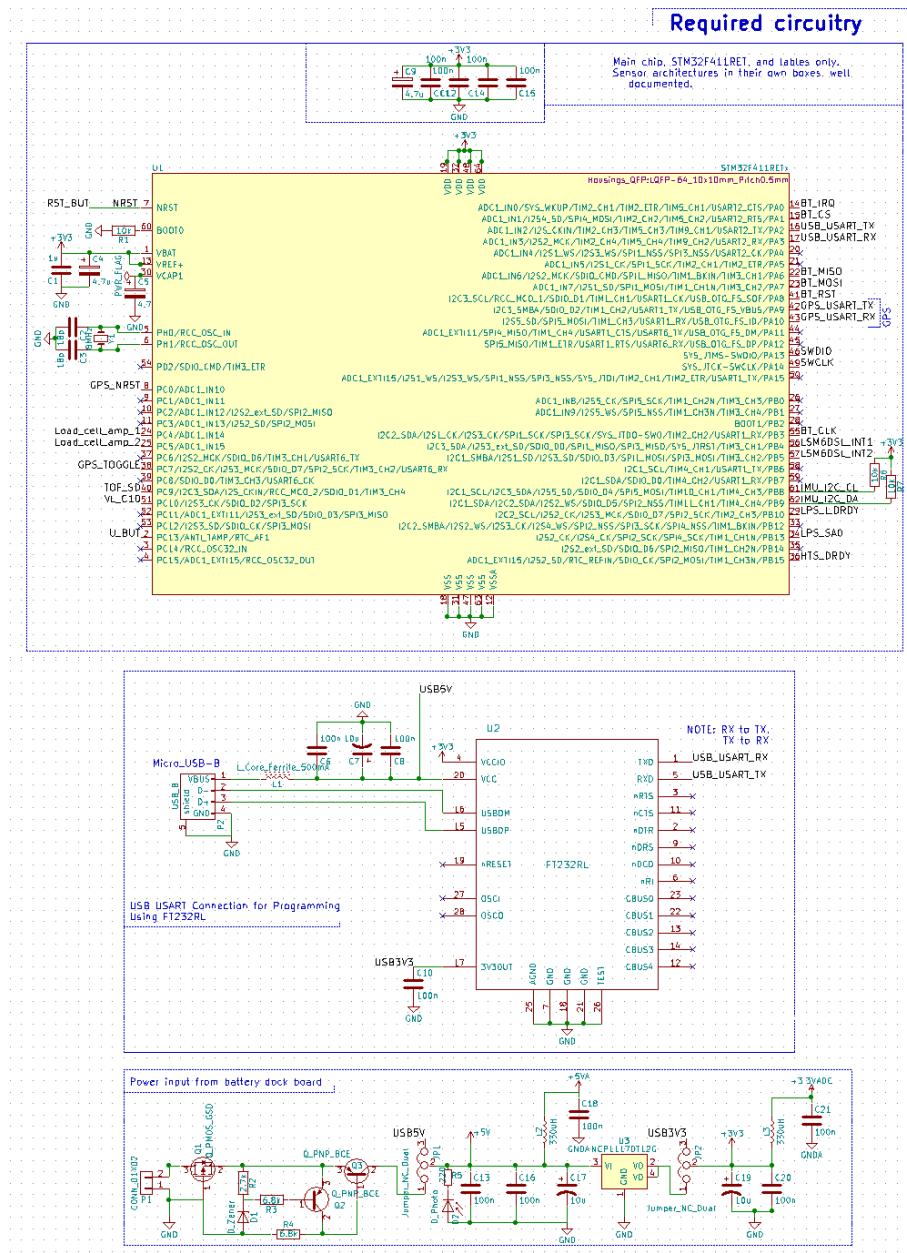


Figure 51: Schematic of revision 3, part a: Required circuitry

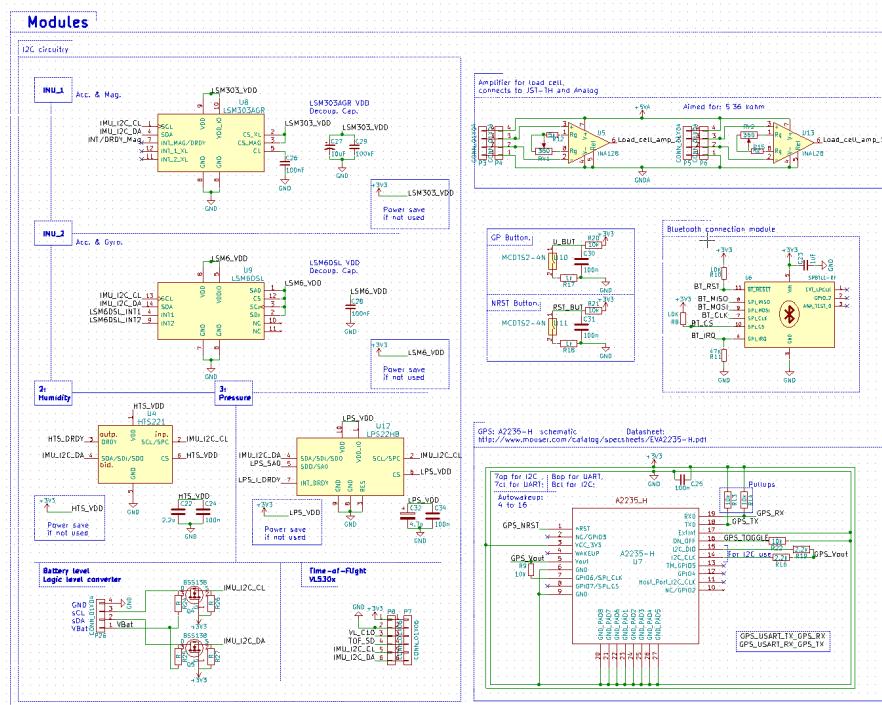


Figure 52: Schematic of revision 3, part b: Modules

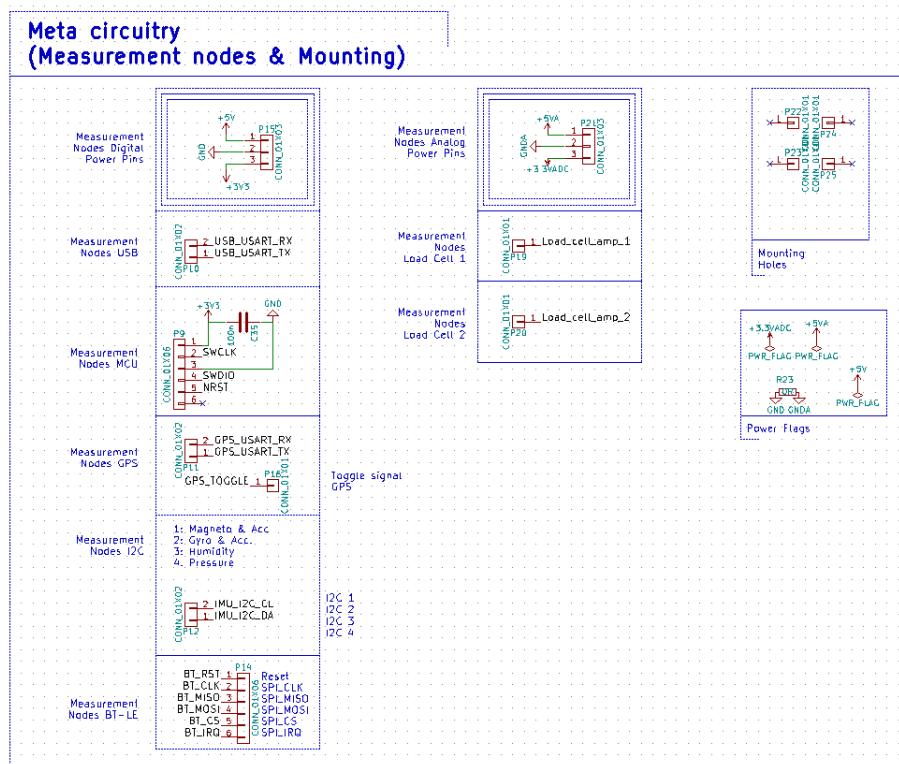


Figure 53: Schematic of revision 3, part c: Meta circuitry