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

December 7, 2017

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

1 Introduction **2**
WHO??? (NO ONE)

1.1 Goals . . . 2

2 The Physics of Sailing **3**
THEOLIN, HENRIK (AXELSSON, OSKAR)

2.1 Point of sail . . . 3
2.2 Velocity . . . 4

3 Product Application **5**

4 Hardware Design **5**
BROLIN, DANIEL (ONE, NO)

5 Sensors **6**
LUNDBERG, JOSEF (NO ONEL)

5.1 Force sensors . . . 6
5.2 The prototype . . . 6
5.3 Choice of component . . . 9
5.4 Amplifier . . . 10
5.5 Height of centerboard sensor . . . 11

6 Software Design **14**
SJÖLUND, JOHANNES (NO ONE)

6.1 ARM firmware . . . 14
6.2 Android application . . . 20
THEOLIN, HENRIK (NO ONE)

6.3 Results . . . 32

7 Inertial Navigation System and Kalman filter **33**
AXELSSON, OSKAR (THEOLIN, HENRIK)

7.1 Inertial Navigation System . . . 33
7.2 Sensor fusion . . . 33
7.3 Navigation Frames . . . 33
7.4 Transformation Between Frames . . . 35
7.5 Inertial Navigation Equation . . . 38
7.6 INS mechanization . . . 38

8 Implementation of Sensor Fusion Using a Kalman Filter. **39**
8.1 Implementing the Fusion Kalman Filter . . . 42
8.2 Result . . . 45
8.3 Discussion/Future Work . . . 47

References **49**

1 Introduction

WHO??? (NO ONE)

The art of sailing has been around for millenia. 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 sensor array:
  * Accelerometer
  * Gyroscope
  * Magnetometer
- Fusing the sensor output to get an accurate estimate of boat attitude

. Position tracking and velocity

- Implementing a 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 depeding on the system state
- Implement different ways for the user to retrieve information

2 The Physics of Sailing

Theolin, Henrik (Axelsson, Oskar)

2.1 Point of sail

A sailboat can achive velocity by catching wind in the sail at different angles. This is called point of sail and the velocity is dependent on the dinghys 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 alignemnt of the sail and the direction from where the wind is coming. There are five different states of point of sail that are divided into degrees away from the true wind origin. These are

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

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

Beam reach (lift) angle 90°

Broad reach (lift–drag) angle aound 135°

Running (drag) angle around 180°

and are represented in figure 1. A sailor wants to prevent the sail from luffing when the sail starts to flap in the wind and no propulsive force is achived. When the dinghy is in the close-hauled and beam reach state the sail produces lift force that is produced from the avarage 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 propelling the dinghy. When the dinghy is in the broad reach state both lift and also drag propells the dinghy. Drag acts like a parachute that that catches the wind and propelles 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.

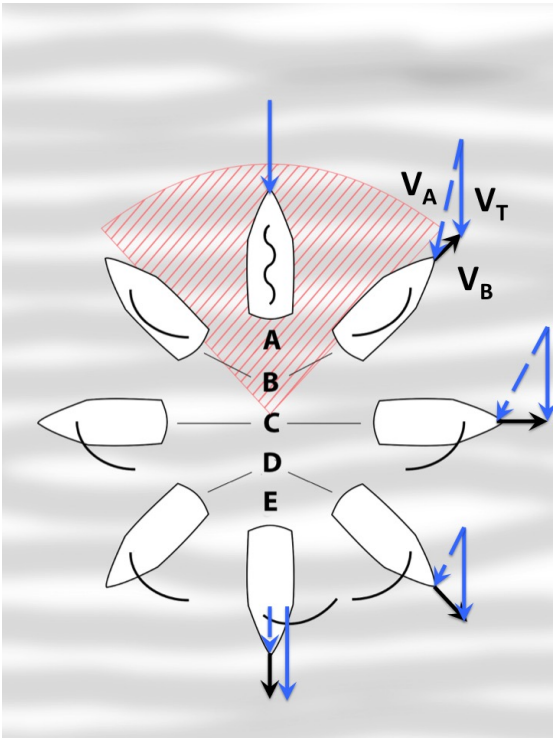

Figure 1: Points of sail

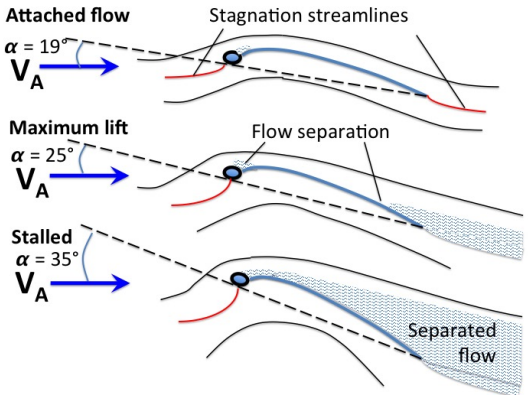

Figure 2: Sail angles of attack

2.2 Velocity

The main goal in sailing is to maximize the efficiency at which the forces on the sail translates into velocity of the dinghy. A layman might expect that the fastest velocity is achived when the dinghy is parallel to the true wind, however this is not accurate. The apperent wind which is the wind experienced from the dingys perspective is what propells the dingy. When sailing paralell to the

wind the dingys speed can never exceed the speed of the wind[18]. By sailing upwind close-halued the apperent wind is increased as the dingy accelerates until the drag from the water exceeds the forwad force created from the wind. To further increase the velocity of the dingy should not be heeling excessivly. This is to prevent the centerboard from acting as a rudder and changing the bearing, introducing more drag from the stern rudder when compansating 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 Product Application

4 Hardware Design

Brolin, Daniel (one, no)

Stuff about hardware.

5 Sensors

Lundberg, Josef (no onel)

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.

5.1 Force sensors

The goal is to have a system that can measure the forces from the water pushing on the centerboard.

By looking at some different solutions there are no other solutions that might be as clean looking and prominent as this approach. 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. The 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. The board itself will not be disassembled in any major part of way. Meaning that this approach doesn't need any modifications to the board itself. This has been the goal and the chosen approach. Modifications in the mounting plate is the way to go, the other solutions is either way more difficult to apply and mount or more complex.

5.2 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 ball. The ball itself is kept inside this small area where it can move in and out.

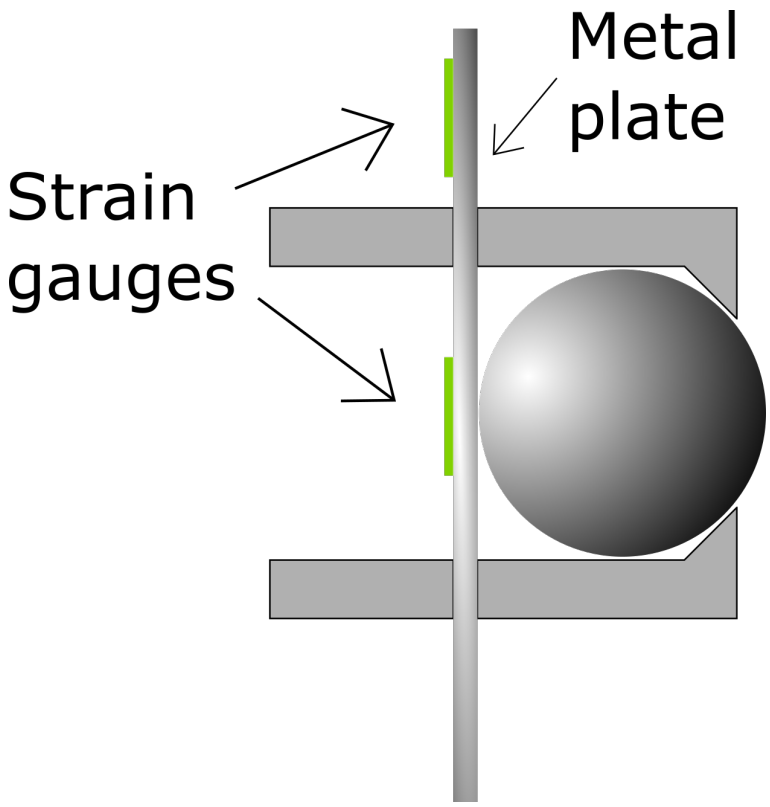

Figure 3: Function of first prototype

This way of implementing strain gauges was the first idea. The main case for this strategy was that in the start of this project these gauges were supplied to us, as a leftover from the last group. With this implementation, we could already start working on a prototype and get a small head start in to the project. But as some research shows, it is a more difficult way to solve this problem and it would take bit more work and some sensitive circuits to measure the force. The gauges also need to be stuck in place using some specific glue and can easily be done incorrectly and therefore prevent good measurements.

A model of the pressure sensor was constructed in the CAD program fusion 3D. This model 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.

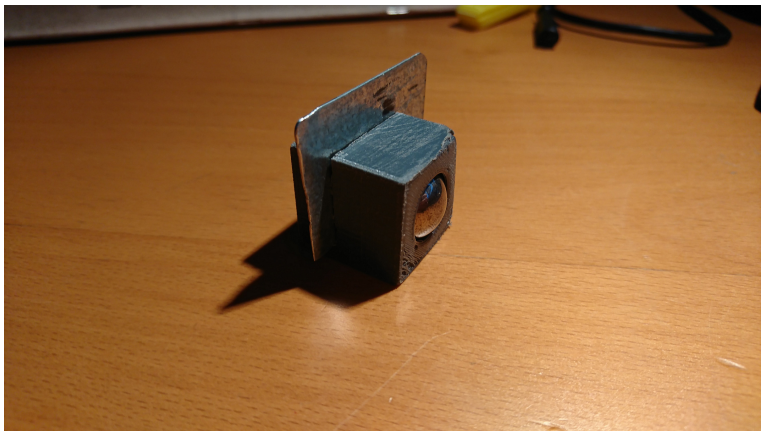

Figure 4: Function of first prototype

The force is then 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.

New idea: 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 the 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.

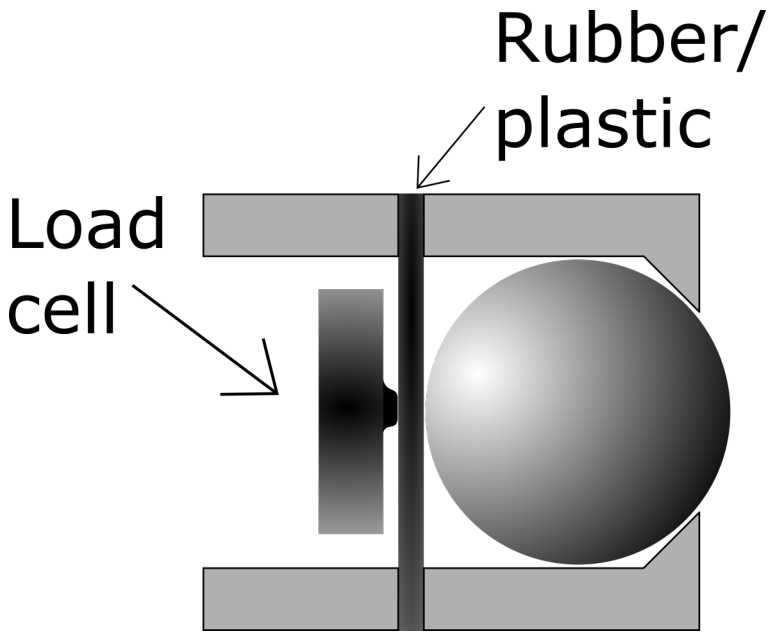

Figure 5: Function of second prototype

5.3 Choice of component

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 $90.75Kg$ force range should be enough. In the case 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 selected to this part, the compression load cell called FX1901. From the datasheet the voltage readings of this piece could be calculated. With a maximum voltage reading of around $36mV/V$

Figure 6: Load cell, FX1901

5.4 Amplifier

In order for the microcontroller to make some good measurements from the load cell an amplification for the That's a small signal and needs to be amplified to get some good measurements. A good measurement signal to the MCU should be in the order of in between $0 - 5$ volts. This is achieved by an amplification gain of around 20.

A suitable amplifier needs to be chosen from the vast ocean of different models. Inspiration is taken from The university of Chicago[33] in an experiment where they uses this exact load cell together with an instrumental amplifier called INA125. This amplifier is somewhat more complicated and have some more features than other amplifiers. In the same family of instrument amplifiers a model called INA126 is selected as a less complicated and more power efficient solution.

The choice fell on this little fellow, the INA126.

Figure 7: Amplifier for the load cell signal

Which not look so interesting but has the benefit of having a smaller power consumption then many others by being a bit simpler than many others. But sufficient for our purpose.

It seems like a smart choice because when the system is battery operated, like in this case, every watt counts.

The gain on this piece is easily calculated with this function. Gain is 5 plus 80k ohm divided by our chosen resistor Rg.

If our desired 20 gain might not work or the voltages calculated is off, the gain is easily redone with this expression.

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

Now that we know how to get the force measurements, we are going to talk about how to measure the frequency of waves at sea.

5.5 Height of centerboard sensor

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

We have found some sensors that might work for us, this is the smallest we found. It is $3cm$ wide and about $5cm$ high. As we have some tight space constraints this can probably fit inside or just stick out a little bit. This particular sensor is in the range of $2000sek$, which feel like a lot. But if no other solution works this might be considered again.

We have also looked into some light sensors. This is implemented with the use of a plate placed ion 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 IR[1]-sensors; they will probably send the signal in a wide spread pattern which will make the distance measurement troublesome as this signal has just a small plate to bounce off of.

With the use of a LIDAR[2] 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 suitable small sensor for the Sailoraid system could be the "micro LIDAR" circuit from adafruit

[1] Infrared
[2] LIght Detection And Ranging

5.5.1 Component

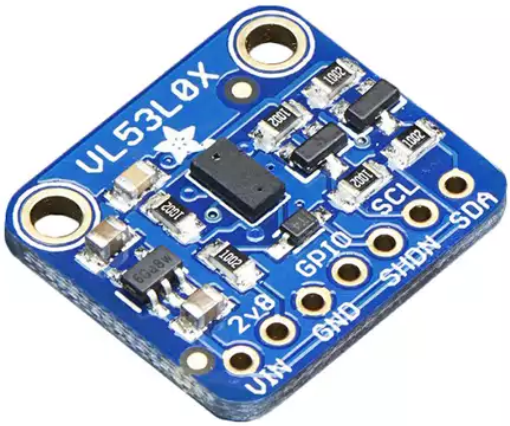

Figure 9: Time of flight, μLIDAR, distance sensor Adafruit VL53L0X

Problems:

As there will always be water around and on the centerplate there is risk of misdirection. The light that is sent might get misdirected if intercepted by water droplets between the sensor and the panel of which the light is to be reflected.

By the fact that the sensor has to be waterproof the signal has to go through a medium, which can be some type of plastic or even glass. If the signal can be read correctly through this medium or if the signal will get corrupted must be further tested.

6 Software Design

Sjölund, Johannes (no one)

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.

6.1 ARM firmware

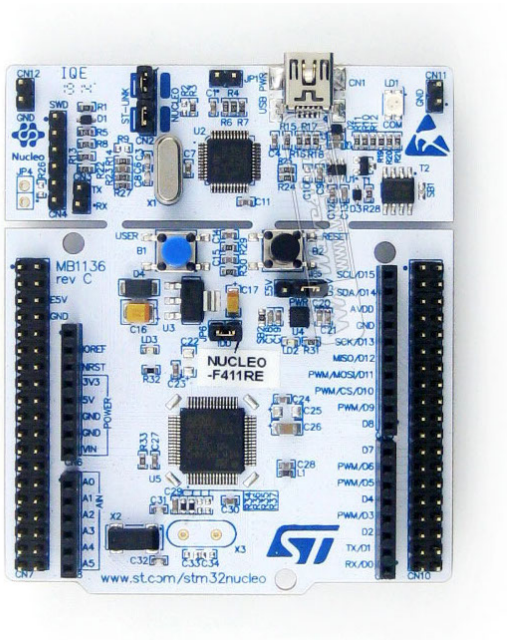

Figure 10: NUCLEO-F411RE development board for the ARM Cortex-M microcontroller STM32F411RE.

For rapid prototyping and firmware development purposes, the NUCLEO-F411RE development board seen in figure 10 was used. This board contains break-out pins for the ARM Cortex-M microcontroller STM32F411RE, a UART to USB bridging circuit and general purpose LEDs and buttons. It is compatible with various Arudino shields as well as expansion boards developed by ST.

In order to speed up firmware development, the STM32CubeMX[1] 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 therefor 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.

6.1.1 Bluetooth

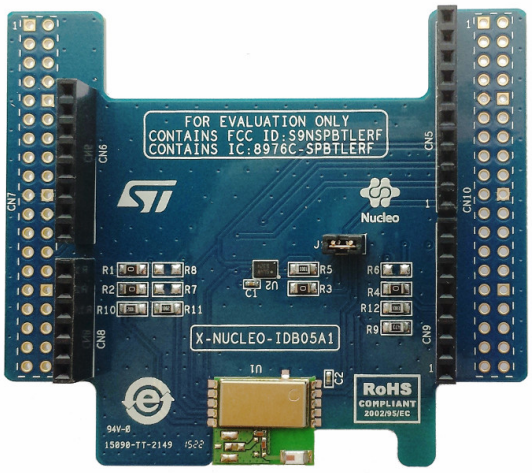

Figure 11: X-NUCLEO-IDB05A1 Bluetooth Low Energy evaluation board for the STM32 Nucleo

For prototyping, the Bluetooth evaluation board X-NUCLEO-IDB05A1[5] seen in figure 11 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[2] 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[4]. 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 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 miminal interference with other parts of the firmware code.

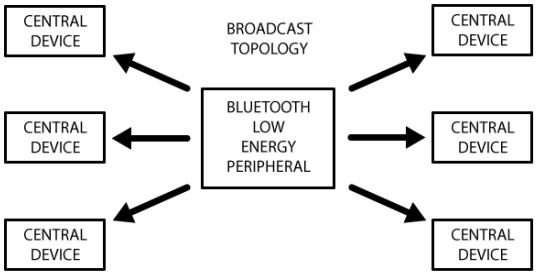

Figure 12: Bluetooth GAP topology.

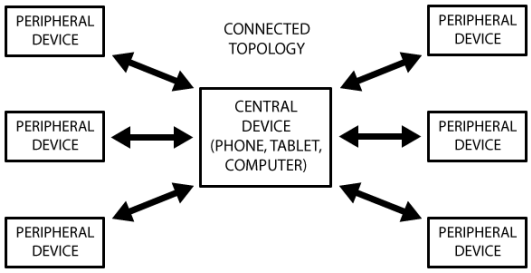

Figure 13: Bluetooth GATT topology.

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 13 and 12 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.

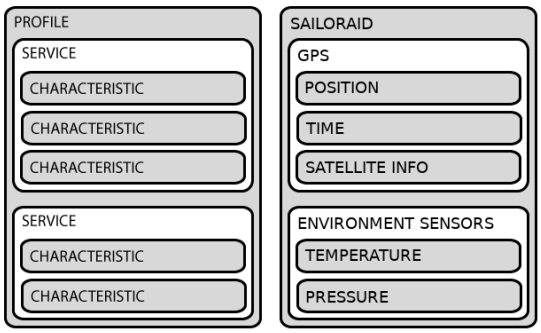

Figure 14: Bluetooth GATT transaction profile with usage example.

The GATT protocol performs transactions by nested structures called Profiles, Services and Characteristics. An example of this structure can be seen in figure 14. 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.

6.1.2 IMU

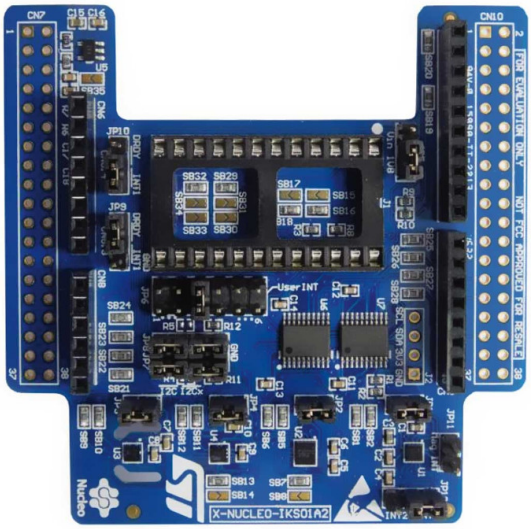

Figure 15: X-NUCLEO-IKS01A2 motion MEMS and environmental sensor expansion board for the STM32 Nucleo

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[6] evaluation board (figure 15) 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 [3] library developed by ST was used. This library implemented the I^2C communication protocol used by the previously mentioned IMU devices in the form of simple function calls, which saved a lot of development time. It was quite simple to integrate with the code generation from STM32CubeMX, only a few source definitions had to be modified. Like with the Bluetooth library (section 6.1.1) blocking communication was chosen to simplify the code, even though the MCU supported interrupt based DMA transfers. The I^2C operated in fast mode at 400 kHz which was thought to cause minimal interference with the rest of the system in blocking transfer mode.

An important use case for the IMU was to determine the current orientation of the dinghy. To accomplish this, a type of sensor fusion algorithm called Madgwick AHRS (section 6.1.3) was used.

6.1.3 Madgwick AHRS

Madgwick AHRS [7] 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 [8] for more details.

6.1.4 USB UART

Since this project involved analyzing sensor data for developing sensor fusion algorithms, for example combining GPS and accelerometer for accurate positioning (section 7) 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.

At a relatively low baud rate of 115200 bps, 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.

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

6.1.5 GPS

Figure 16: Maestro A2235-H GPS module with built-in antenna

The GPS module used in this project, A2235-H by Maestro[9] could communicate with the MCU through either I^2C or UART. Both protocols require only two pins to operate, but UART communication was determined to be easier to implement in code. The UART baud rate of the GPS module was set to 4800 Hz by default. While this could be changed by software there was no reason to do so. The low baud rate did however mean that blocking transmissions might cause problematic interruptions in the firmware code. To prevent this, interrupt based communication through DMA was implemented, using the same type of queuing system as the USB UART (section 6.1.4).

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

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

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

Several NMEA parsing libraries are freely available on the web. The one chosen for this project was called Libnmea[10] and allowed the sentences to be automatically recognized, parsed and stored into easy to use C structures.

6.1.6 Range sensor

In order to communicate with the VL53L0X range finder sensor, another software library called X-CUBE-53L0A1[11] 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.

6.2 Android application

THEOLIN, HENRIK (NO ONE)

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 sailing experience so that the sailor can focus on the joy of sailing. A crucial part was that the data was displayed in a manner that was easy to interpret and provide the help that was called for. To further increase the flexibility of the design it was decided to implement several different user layouts that the user could 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 representaion would be the best way to go since the sailor needed to be in constant motion to counteract the forces applied on the ship by the wind and current. This would make watching a screen to retreive information somewhat difficult. Other ways of representing data was implemented using text-to-speech where the sailor would get important information by sound aswell as text. Vibration was also used with different vibration sequences depending on different states of the ship.

6.2.1 Software design

A *UML*[22] model of the system (figure 17) was developed for an overview of the system implementation. The android system uses activities[21] to display content to the device screen. These activities have a life-cycle (figure **??**) that determined how data was accessed and displayed. What should be understood was that the application started in a main activity and switching to another activity was done by sending an intent that spawned as a child activity. While the application was in the child activity the main activity was paused but the state was stored and when switching back from the child all data from the previous state could be accessed. When the user returns from the child activity that one is destroyed and all data is erased. Sending data between activities is done using intents, to send intent to a child activity could be done by adding a bundle with a data object along with the intent use to spawn the child activity. Sending data back to the child activity was done by calling the method startActivityForResult, this allowed the child to send a data packet as a result back to the parent.

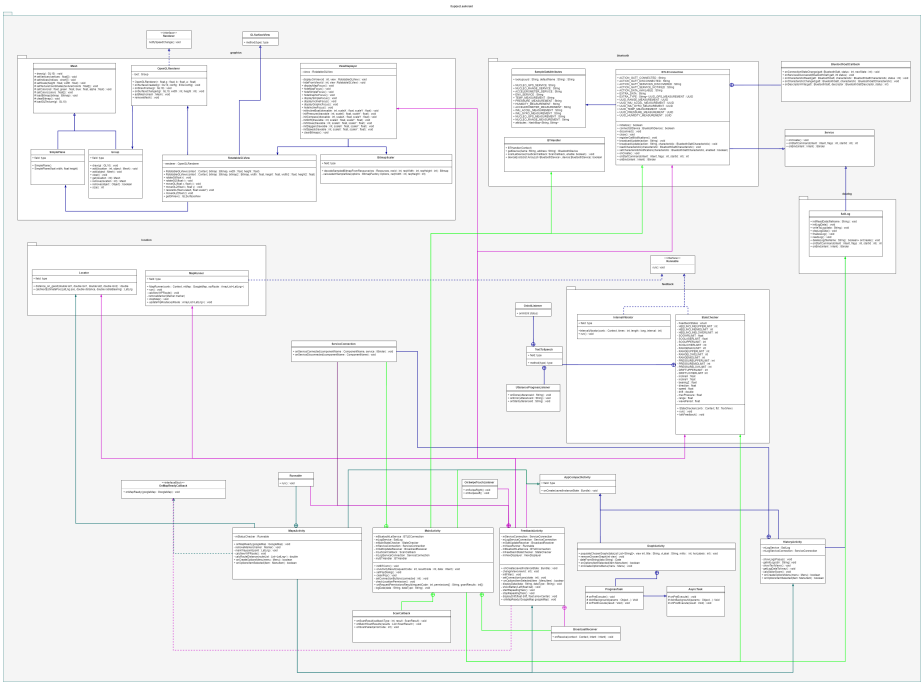

Figure 17: UML model

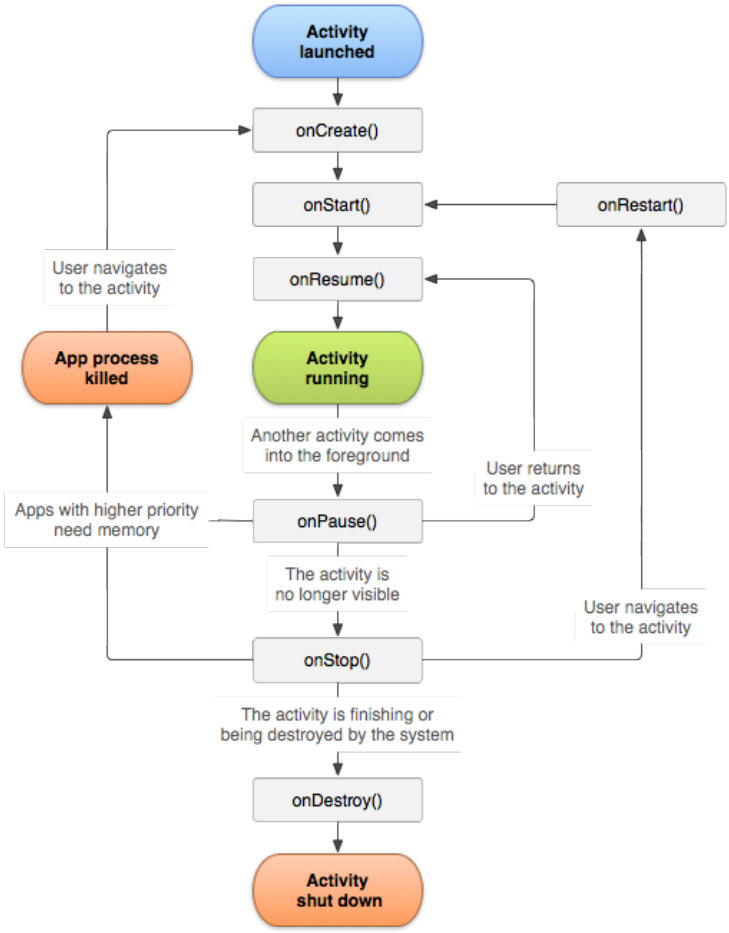

Figure 18: Activity lifecycle

The bluetooth connection and data logging class was implemented as services[12]. A service could be started by an activity and continue running until it was explicitly called to stop. This allowed for several activities to share resources and perform long-running operations in the background. The lifecycle of a service was seen in figure 19.

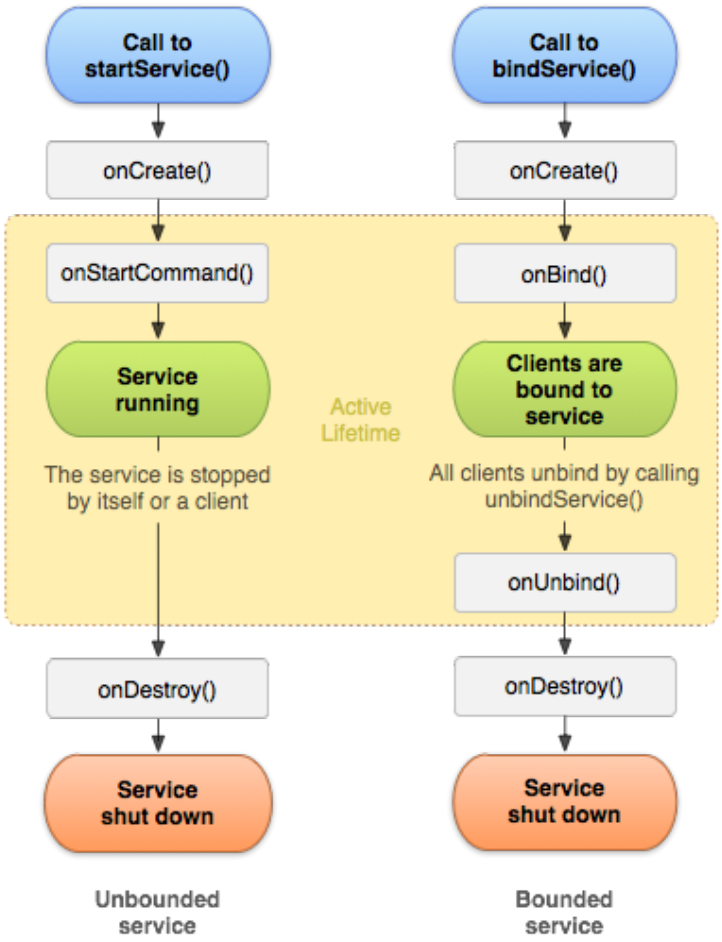

Figure 19: Activity lifecycle

6.2.2 Feedback

For the feedback states used in the application there were some approximations and simplifications made for easier implementation. As explained in the Physics of sailing chapter there are certain parameters that could be evaluated into different states of the dinghy. A big approximation was made that there is no water movement for all our implemention. The states implemented were

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

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 was given to the user. For handling states a class call StateChecker was implemented. This would store sensor values and check against limits defined at an interval that was also defined in the class. Depending on these states this class would also determine what feedback should be given.

6.2.3 Visual Feedback

It was determined that the visual feedback given to the sailor 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 ship. Because of different personal preferences the ability to switch between different layouts was implemented. This was done by a simple swipe on the screen to toggle the view to the next layout. All layouts consisted of a subset of views from the complete set including

20 **Incline** displayed a ships relative incline against and artificial horizont.

21 **Pressure** moved a pin along a colored bar to represent high of low pressure applied on the centerboard.

22 **Bearing** rotated a compass to show the ship relative bearing against true north.

23 **Map** displayed current location of the ship.

24 **Drift** was represented with a colored bar with two arrows that moved to the relative drift direction to show the sailor if the ship was holding it's set navigaitional reference.

25 **Speed** displayed a speedometer from a classical Swedish vehicle[19] with a movable bar representing speed over ground.

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

27 **Wave frequency** displayed a wave moving towards a ship at a speed representing different periods of the waves.

28 **Feedback** a text that changed values based on the current state of the boat.

Figure 20: Incline feedback view

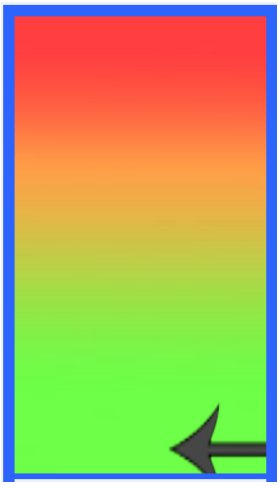

Figure 21: Pressure feedback view

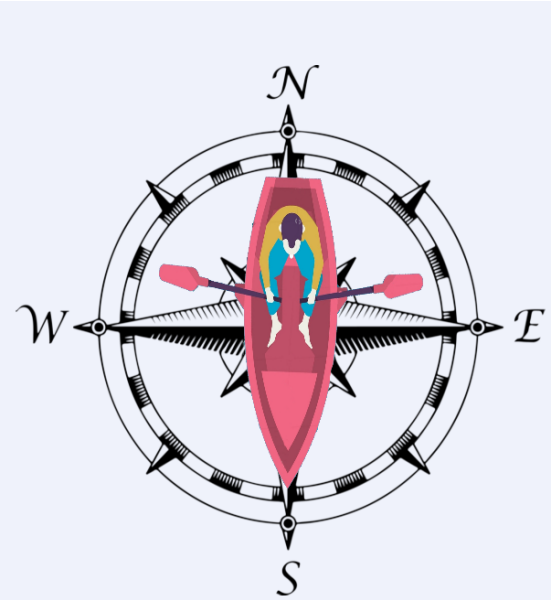

Figure 22: Bearing feedback view

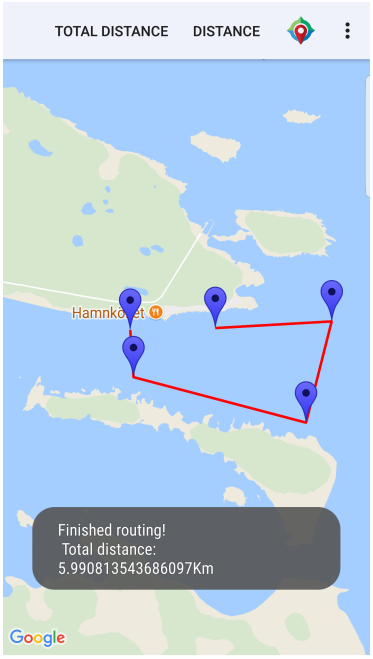

Figure 23: Map feedback view

Figure 24: Drift feedback view

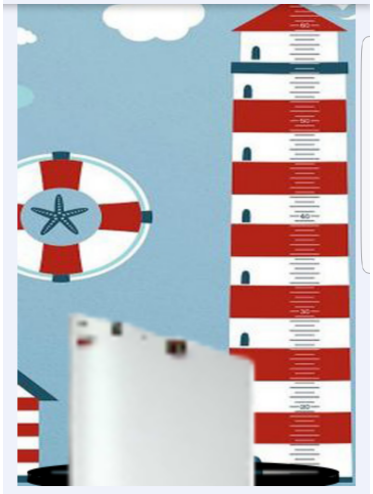

Figure 25: Centerboard height view

Figure 26: Speed over ground view

Figure 27: Wave period feedback view

Figure 28: Text feedback view

The figures were chosen at a development stage and improvements can easily be made by adding different *PNG*[20] files in android studio. They were implemented as GLSurfaceView[23] so that updating the figure could be done by calling a requestRender function, this improved the performance of the application when only updates to the figures were done when new sensor readings were received. To make the figure fit the application the GLSurfaceView was extented into a RotatableGLView that had the wanted features for rotation and positioning. Using theese figures four different layouts were developed to give multiple choices for the user and was seen in figure 29. Switching between theese layouts a swipeTouchListener was implemented which allowed the user to swipe across the screen to change layout. Changing view the figures needed to be redrawn onto a new view matching the current layout, this was handled in the ViewDisplayer class that contained all RotatableGLViews.

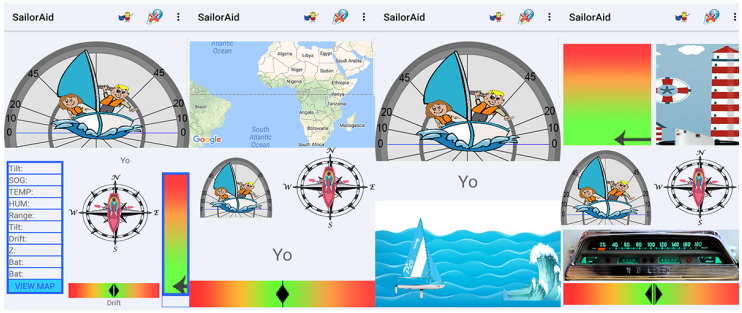

Figure 29: Different layouts

6.2.4 Audio feedback

Based on values from the sensors, different states were implemented and for each state a text was read out to the sailor. The feedback was implemented such that the audio feedback would continue even if the device was put in sleep mode. Handling the text to speach feature a TextToSpeech?? class was implemented as an inner class in StateChecker. To prevent the speech to be interrupted preemptively an UtteranceProgressListener[25] was used.

6.2.5 Haptic feedback

Based on the same states as the audio feedback vibration was also implemented. The frequency and length of the vibrations were implemented in such a way the each state had a unique signature that could be recognised by the sailor after some practise with the system. Handling the interval a IntervalVibrator was implemented and run by the StateChecker.

6.2.6 Log

The ability to analyse the sailtrip was determined an important feature for the user so a logging function was implemented and data was stored on the device internal storage. This was handled by the SailLog class and these files could be read or deleted at a later date. Storing a log was implemented such that the sailor would need to start a logging session after bluetooth connection had been established to the ship. After the log was started it would create a file on the device internal storage and write sensor data to the file at the same frequency as the data was transmitted by the system. While logging was active the device would continue to store information even if the screen was 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 sailtrip was finished the sailor could read the saved log file and receive a summary of the trip (figure 30). More information from the log could be analysed by reading graphs (figure 31) where the sensor data was shown with respect to time. These graphs were implemented such that the user was able to choose what graphs to be displayed on the device for improved comparability.

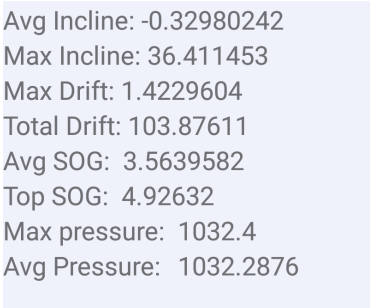

Figure 30: Log data summary

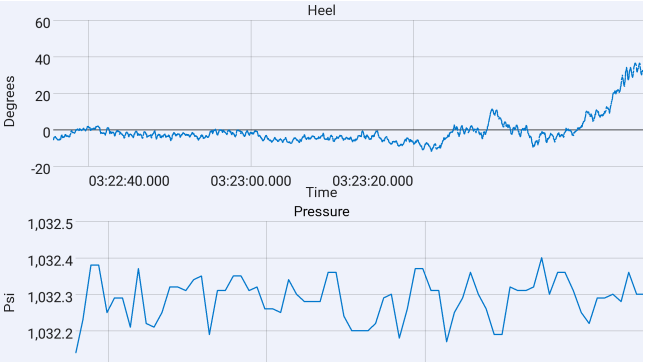

Figure 31: Log graphs

6.2.7 Bluethooth connection

A list of all devices in the nearby area was displayed when the user decided to connect to the system, this added flexibility to the user and allowed connection between multiple systems with different MAC-adressess. Scanning for nearby devices was done in the MainAcivity and handled by the BTHandler class. The application uses bluetooth low energy technology to match the systems implementation. This allowed for the device to recieve notifications when the data is altered on the system and updates would occur in different frequencies for different types of data. To ensure a stable connection between the system and the device the connection class was implemented as an android service[12]. This allowed the device to keep the connection alive between different views and even when the device was put to sleep mode. All functionallity regarding connecting and receivign data was implemented in the BTLEConnection class and know characteristics and the determined unique Gatt service identifiers was stored in the SampleGattAttributes class.

6.2.8 Map

For the sailor to get more information about location two maps were implemented, one that was embedded into a layout in the FeedbackActivity and the functionallity of this was handled in the MapRunner class, another was implemented in MapsActivity. This could be improved by using the same class for

both maps to achive better readability and modifyability of the code. Both these implementations utilized the Google maps API[26], which could be used freely and the map service is highly accurate and suited our system well. Interaction with the maps could also be made quite easily which helped making development faster. The sailor had the ability to see a path over the trip and the total distance traveled. Waypoint could be placed if the sailor wanted to decide in advance a certain trip. The distance of all the waypoints was displayed and the distance currently traveled to give the sailor information on how long the trip was and how much of the trip was left to be sailed. A path to the nearest waypoint was shown to help navigation, when the sailor was close to the first waypoint the path to the next waypoint was shown. The waypoint route could also be viewed in the map of the FeedbackActivity layout.

6.2.9 Drift

Calculation of leeward drift was done by inspection 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*[14] was derived and the new estimated position was calculed with

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

where R is the earths radius, d is the distance traveled, θ is ships bearing, λ_1 and φ_1 is the point of origin in longitude and latitude. This new estimated position λ_2 and φ_2 was then compared to the next positional data from the *GPS* and the distance between these points was caluculated using *Equirectangular projection*[15]

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

where $\Delta\lambda$ and $\Delta\varphi$ are the differences in longitude and latitude for two location points. This function had high performace gain but was less accurate over large distances then for example the *Haversine formula*[16]. Since for this function only small changes in distance were calculated the accuracy was more then 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 and a small deviation of a couple of degrees resulted in a large drift. The solution was to use the directional data sent from the gps module which calculates direction based on last positional data. This solution would work good for movement forward and when fast sideways motion is detected the estimated position would be along the same vecor as the previous direction. However for constant drift sideways this approach would display that no drift were taking place. This calculation was handled by the Locator class.

6.3 Results

7 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 pray[34]. Sensor fusion theory is not only found in the living species it is found in cars, planes, computers and so on and this to enhance performance and accuracy[34]. In this project a sensor fusion will be designed to enhance the accuracy of the dinghy's position and velocity. The fusion will be between a global positioning system, GPS and an Inertial Navigation System, INS. The INS contains on a low priced, Inertial Measurement Unit, IMU.
The GPS's accuracy is not uniform since there might be building reflections, atmospherics delays or clock bias errors[28]. Using the only information provided by an 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.

7.1 Inertial Navigation System

In navigation there exist different techniques on how to navigate using an IMU, a typical technique is using a Strapdown Inertial Navigation System, 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 is 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. Integrate the resolved specific force measurements to obtain its velocity and position.[32]

7.2 Sensor fusion

A popular filter to use when applying sensor fusion is to use a Kalman filter, (KF). 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[28]. It's popular to use due to its efficiency when calculating predictions.[31]

7.3 Navigation Frames

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

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

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

The Navigation frame denoted n-frame for future notation is a local frame and has its origin located in the navigation system, in this case, point P, see Fig. 32, and its axes aligned with the directions of north, east and down, denoted NED. The turn rate of the navigation frame with respect to Earth's fixed frame, ω^n_{en}, is directed by the motion of point P with respect to the Earth and is referred to the transport rate.

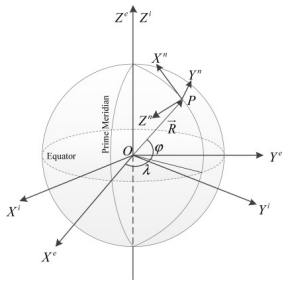

Figure 32: Inertial, Earth and Navigation frame.

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 Fig. 34.

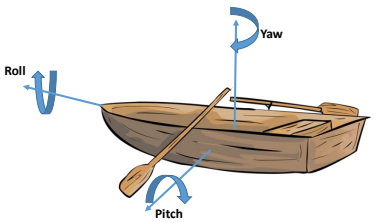

Figure 33: Body frame

- Yaw (ψ): Is an imaginary line running vertically through the ship and through its center of gravity. A yaw motion is a side-to side movement of the bow and stern of the dinghy[29].
- Pitch (θ): Is an imaginary line running horizontally across the ship and

through the centre of gravity. A pitch motion is an up-or-down movement of the bow and stern of the dinghy [29].

- Roll (ϕ): Is an imaginary line running horizontally through the length of the ship, through its centre 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 [29].

7.4 Transformation Between Frames

Referring to Fig. 32 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) [30]. Where λ and φ is the latitude and longitude, respectively. Then the transformation between the two frames can be done using the Direction Cosine Matrix, noted DCM for future notation, which is defined as [30].

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

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

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

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

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

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

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

Using 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 [30]

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

where

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

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

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

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

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

$$\tag{12}$$

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

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

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

The Euler angles is a transformation from one coordinate frame to another and are 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[30]. 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 s done in a single rotation, instead of three about a vector defined in the reference frame. The DCM using quaternion is defined such that[30].

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

Where q_1, q_2, q_3 and q_4 can be derived using the Euler angles from 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[30]

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

Where ω_e is the angular velocity of the Earth and has a value of 7.2921158×10^{-5} rad/s[30]. Using the projection matrix Eq. (2) its possible to project $\bar{\omega}_{ie}^{i}$ onto the n-frame

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

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

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

Where $\dot{\lambda} = v_e/(R_N+h)cos(\varphi)$ and $\dot{\varphi} = v_n/(R_E+h)$[30], here we assume the Earth to be an ellipsoid and that there is no variation in Earth's gravitation depending on where the user is on the ellipsoid. R_N is the meridian radius of curvature and defined as $R_N = R_0(1-e^2)/(1-e^2 sin^2\varphi)^{3/2}$[30]. R_E is the transverse radius of curvature and defined as $R_E = R_0/(1-e^2 sin^2\varphi)^{1/2}$. Where R_0 is the length of the semi-major axis and has a constant value of 6356752.3142 m, e is the major eccentricity of the ellipsoid and has a constant value of 0.081819, φ is the current latitude in radians, v_N and v_E is the velocity in north and east direction, respectively and h is the height above the surface of the Earth. Then Rewriting Eq. (18) with the new expressions.

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

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

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

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

7.5 Inertial Navigation Equation

Describing the position of the dinghy in the n-frame is done by[30].

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

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

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

Where $\dot{-}$ symbolizes the first derivative with respect to time. Thus, $\dot{\varphi}$, $\dot{\lambda}$ and $\dot{h}$ can be derived by rewriting Eq. (23) as

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

Thus the dynamics describing the system can be expressed as[30].

$$\dot{\bar{r}}_n = D^{-1} v_n \tag{24}$$

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

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

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

7.6 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[32], Using Eq. (24),(25) and (26).

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

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

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

where

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

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

The Navigation frame Inertial Navigation System can now be seen in Fig. 34, Where its possible to examine all the steps which were derived earlier, from the IMU to the output.

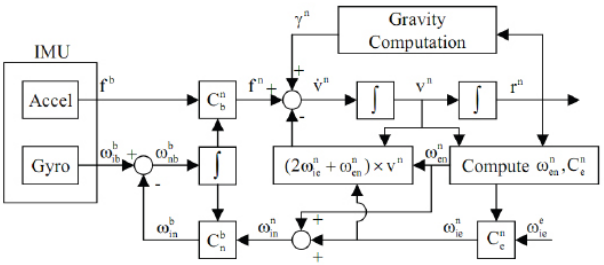

Figure 34: The Inertial Navigation Frame

8 Implementation of Sensor Fusion Using a Kalman Filter.

Perturbation Analysis

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

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

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

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

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

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

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

8.0.1 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

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

Where

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

and

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

8.0.2 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 [30].

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

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

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

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

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

where

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

and

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

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

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

Then expressing Eq. (38) as

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

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

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

and

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

8.0.3 Error Dynamics Attitude

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

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

Then relate the above equation with Eq. (33)

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

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

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

or in vector form

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

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

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

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

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

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

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

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

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

Where F_{er} and F_{ev} is

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

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

8.1 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 Fig. 35.

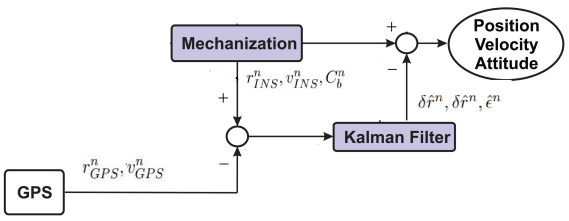

Figure 35: The feedback-method which is used in the system.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Since the sample interval, Δt is very small in this case Eq. (62) can be approximated

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

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[35]. Then the covariance matrix which is associated with $\bar{w}_k$ is[35]

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

and Q_k can then be approximated using first order of the discrete transition matrix[36]

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

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[36].

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

where z_k is the measurement vector and e_k is random measurement noise, with following characteristics[35].

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

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

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

then the state vector is updated

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

the last step is updating the covariance matrix

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

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

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

then updating its covariance

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

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

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

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

The measurement matrix H_k is defined such

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

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

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

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

The Kalman Filter is invoked every time the GPS is updated, i.e. $1Hz$, but since the IMU and the GPS updates at different frequencies a problem arises. The problem is such that at $t_{GPS}(k)$ there 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 Fig. 36.

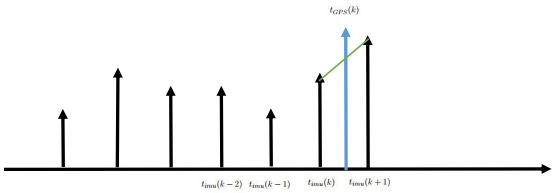

Figure 36: Different update sequences IMU and GPS

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

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

and the equations for the velocities

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

8.2 Result

The data provided by the Inertial Navigation System is working better than just using the raw GPS data, in 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 Fig. 37. 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 37: True VS. Filtered and raw GPS data.

Inspecting the Mean Squared Error, MSE for the position in XY pane in Fig 38 for both INS and raw GPS data we can see that the error is lower for the INS data.

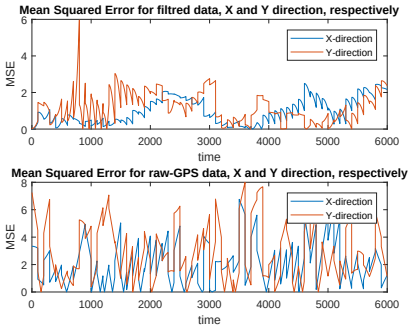

Figure 38: Mean Squared Error

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 Fig. 39. In this case, the GPS data is more accurate.

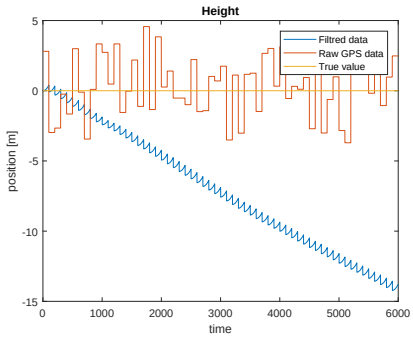

Figure 39: Height. True VS. Filtered and raw GPS data.

The velocities in the XY plane does not represent the true system in a good manner either this can be seen in Fig. 40. In this case, the GPS data is more accurate as well.

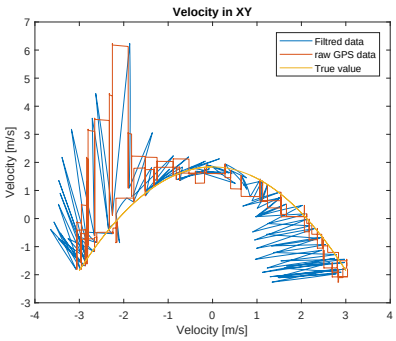

Figure 40: The velocities in XY plane.

8.3 Discussion/Future Work

Depending on what the user wants to measure, e.q. position in 2 dimensions, height or velocity, different sensors should be used. In the case of the first, 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 data, with the IMU and GPS wanted characteristics.
Also there is a static calibration program for the accelerometer. To see how the program works there exists a readme.txt file in the Matlab folder.

Implement a working real-time system for the sensor fusion between the GPS and INS, there exits C-code for this already, but there is some serious bug/bugs which makes the error stored in $\hat{X}$ converge 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 on 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 these more advanced calibration technique is to determine the following parameters[32]

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

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

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

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

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

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

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

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

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

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

[11] 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[28] BOKEN *BOKEN*.

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

[30] Dr. Oliver Nelles
nonlinear system identification.

[31] Duygun, M., Kutlu, L. & Sickles, R.C. J Prod Anal (2016) 46: 155. https://doi.org/10.1007/s11123-016-0477-z

[32] 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

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

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

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

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