An autonomous marine environment surveying platform

Nick Østergaard Rasmus Lundgaard Christensen
Frederik Juul Attila Fodor Todor Muresan

December 18, 2012

Worksheet #42 by group 12gr730

Contents

Introduction 1

Measuring environmental parameters in and around the water in Greenland (or any coastal nation) is a time consuming task. Today measurements are carried out by manually navigating large vessels up and down along the coastline and into the fjords.

An example where this could be highly efficient would be at the Fukushima accident 2011. As seen on figure 1.1 there exists no measurements of the radiation in the water, due to the danger and resource demand of sending a manned expedition. Only estimates of the radiation level in the water exist, and therefore it is impossible to say what effects the radiation have had on the surrounding maritime life, as well as what proximity margin should exist. A fleet of small autonomous vessels would be able to quickly give an overview of the damage caused, as well as the spread and strength of the radiation. Another purpose of these measurements, which will be the primary focus for this project, could be to measure the water depth for bathymetric surveys Stewart [2008]. This would allow for safer voyages in and around the coastlines, and for general environmental studies.

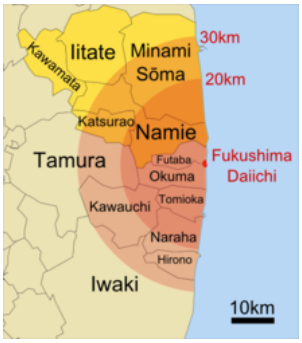


Figure 1.1: At the site of the 2011 Fukushima accident only landside radiation measurements exist

These measurements are today carried out by one large vessel, which could be exchanged for several smaller ones to reduce the surveying time and labor cost. This would make for a faster measurement of the coastline as well as giving an opportunity to measure previously un-surveyable waters due to costs or lack of access.

Reducing the size of a vessel poses some other challenges in regards to the weather conditions and other environmental parameters, that has a much bigger influence on small scale vessels, than on larger ones. Limitations are also imposed from the availability of power for the boats, the noise of measurements as well as limitations on the availability of computing power and communication links. This paper will deal with how such a surveying vessel can be developed, and still measure the water depth whilst under the influences of all the above.

A challenge using a single beam transducer to measure water depth is the roll/pitch of the craft. Figure 1.2 on the following page illustrates this problem, where an induced pitch/roll of the ship (eg. a wave) makes it impossible for the vessel to make a precise measurement.

Another challenge, using a fleet of measurement boats, would be how to enable them all to coordinate and effectivize their effort. This could be done by having a mothership which would be in charge of coordinating and controlling the smaller ships. This larger ship would be able to store more energy, and be able to charge the smaller ships as needed. Due to this easy access to power it would be able to turn up signal strength and computing power when necessary. This way it would be possible to reduce the requirements for the hardware aboard the small ships while still making sure there's enough to com-

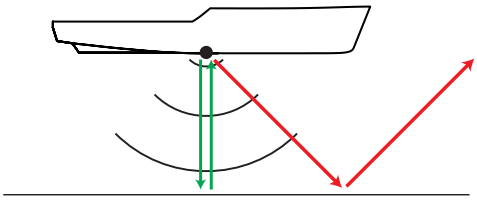


Figure 1.2: Overview of a depth mapping system. The green represents a measurement shot directly down and the red represents a measurement with induced pitch on the ship

pute complex algorithms necessary for control. Having this process centralized, however, offers some additional challenges. Packet loss and limited communication speed demand a robust system where these issues are addressed. In this project the effect of packet loss and limited available data is analyzed and a solution based on an implementation of a Kalman filter is suggested.

Hardware considerations 2

Because of the practical nature of the project, a wide range of components had to be chosen. There are several things to be taken into consideration when designing a scale ship from scratch

2.1 LLI electronics

The Low Level Interface (LLI) electronics is a plug and play device that controls several smaller ship instances. It provides outputs to control the actuators and motors as well as basic sensor readings to the High Level Interface (HLI).

The LLI has been designed to allow for the above functionality. As described in section 2.5 on page 5 the LLI is an embedded platform, for which an Atmel AVR microcontroller has been chosen. The main control board is an Arduino Mega 2560, and a shield interface to all peripherals has been designed 10 on page 89.

The final LLI supports the following features:

Serial
: interface to the HLI with a baud rate of 115200 bps

PWM
: outputs for actuators

I^{2}C
: option for aux communication

Analog
: inputs for various sensors

Relay driver
: output

5V
: regulated output

More detailed information about hardware layout 7 on page 88 and the hardware schematic 10 on page 89 can be found in the appendix.

2.2 Lithium Polymer batteries

Lithium Polymer batteries have been chosen for powering the boat motors as well as for all the electronics, because they offer a very high specific energy and energy density, and can be charged in a relatively short period of time.

Energy calculations

Each battery is composed of 4 series connected cells with a nominal voltage of 3.7 V, making a total of 14.8 V per battery. There will be 6 such units, each storing 3200 mAh worth of charge, which yields a total of $6 \cdot 14.8$ V $\cdot 3.2$ Ah = 284.16 Wh, around 1 MJ of energy.

There are two separate electrical circuits: a power circuit for the motors for which there are 4 dedicated batteries connected in parallel and 2 in the electronics circuit which is separated in order to avoid noise in the digital circuitry. These systems are alloted 198.44 Wh and 94.72 Wh respectively.

If the batteries run at full power (draining a maximum of 80 Amps) the ship will be able to sail for 5 minutes, however, a scenario where the engine runs at full power with a brake on the propellers is extremely unlikely.

Measurements show that the engines should be ran at a setting of 60 of the maximum range of 500 (discretized control steps) and that in this case they drain 2.5 A combined. This will give an operation time of 5.1 hours - thus giving quite a big range on the ships. Assuming a 1 m/s velocity kept constant for 2.5 hours, this gives a range of 9 kilometers forth and 9 kilometers back or being able to measure an area of 1 x 1 km in strips 100 meters apart and still have surplus power.

Battery care and charge meters

Due to the delicate nature of Li-Po batteries, it is absolutely required to never over-charge or over-discharge them because there is a high risk of permanent damage to the batteries. If the temperature continues to raise, there is even a risk of fire and/or explosion. In order to prevent this, we are using a dedicated Li-Po charger with an included balancer, which ensures that none of the cells in the batteries go above the absolute maximum of 4.2 V while charging.

The problem of over-discharge is solved in the power circuit due to the fact that the brushless motor controller has a safety switch which does not allow any cell to drop below 3 V, which would cause permanent damage to the batteries, as they cannot recover after being over-discharged.

The electronic circuit can drain the batteries more than the maximum limit though. In order to prevent this from happening, we implemented a couple of battery level monitors that are integrated in the LLI module and, which can compute the battery levels, thus allow the user to retrieve the boat in order to charge it. They should also include a master kill switch that can save the batteries as well.

2.3 Electronics design

Bow thruster controller

The bow thruster controller is a circuit whose function is to provide power to the bow thruster, under the control of the LLI. This uses a L298 H-bridge connected to a logic gate, so that it can be driven with just two signals: direction and Pulse Width Modulation (PWM).

Another use of this circuit is to provide a lossless voltage level conversion from the batteries' nominal 14.8V to the motor's 7.2V rated voltage. That means that the maximum theoretical duty cycle of the PWM bow thruster control signal is $14.8/7.2 = 48\%$. This, however, is not the actual value that will be used, due to the fall time of the L298 chip, which is pretty big at the 1kHz PWM frequency. By using an oscilloscope to measure the True RMS value of the electronic circuit's output, it was empirically determined that the maximum pulse width value is 33%. The motor used to drive the bow thruster also has a minimum starting voltage for which the PWM minimum duty cycle was empirically determined to be around 5% in this case.

Since the L298 chip has two H bridges inside it, the designed board which can be viewed in the appendix 10 on page 89 includes the circuitry and parts for the two controllers, since it would provide redundancy in case one of them is overloaded or otherwise stops functioning.

2.4 Motors and propellers

As the ship should be able to cope with rather powerful streams a powerful propulsion system had to be chosen. The main propulsion unit consists of 2 Grapuner Brushless 750 14.8 V 1200W engines - which together produce around 3 HP at maximum input. With these powerful engines we have a large dynamic range, that allows for fast acceleration and a fast maneuvering, as well as navigating in strong currents.

The propellers are 2 counterrotating Raboesch brass propellers with a diameter of 60 mm.

2.5 Platform

In this system overview, the platform along with the electronics and its software is described to aid in understanding the physical interconnections that exist between the modules.

The system consists of a wide range of modules which may be sensors or actuators. These modules are connected to a LLI and HLI computing device. The LLI is a microcontroller which takes care of the basic functions of the platform such as communicating with basic sensors and actuators. The software for this is embedded and written in C for the avr-gcc compiler.

The HLI is a x86 compatible computer of some sort that is able to run the higher abstraction layer code written in Python. This code implements the onboard generation of waypoints from map data to calculate the desired heading and speed for the vessel. It also implements the state space model and calculates propeller speeds and in turn sends these to the LLI.

Some modules that are connected to the LLI are not handled by the LLI itself, but instead it forwards the data to the HLI which uses it for the control algorithm. A illustration of this can be seen on figure 5.2 on page 22.

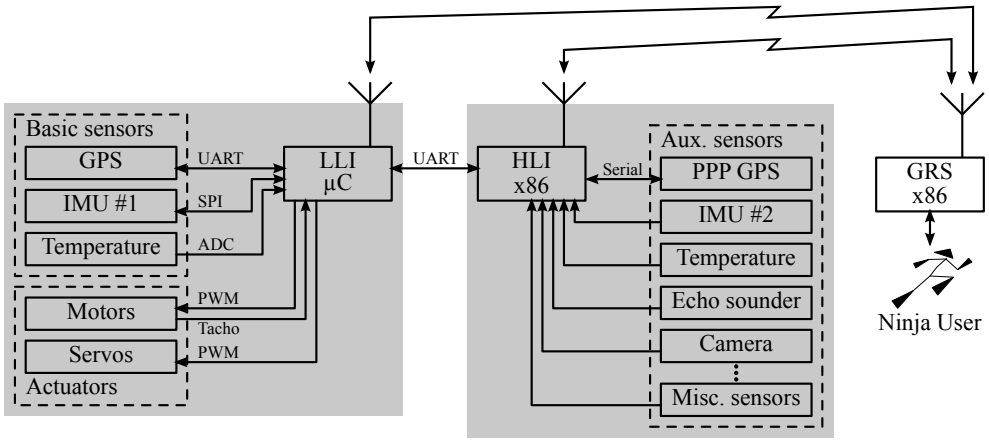


Figure 2.1: Overview of electrical interconnections between LLI and HLI together with periphal modules.

Ship Design 3

AAUSHIP1 has been designed by the ASV group during their 6th semester and is extensively designed in a 3D modeling program called RhinoCeros. The plan is to get the ship molded in plastic. As this process is expensive, it's quite important that we agree on the final design of the ship before we send it to the processing company.

This ship is designed using a function called Lofting, which takes some lines that runs along the ship, and then generates a surface from these. As none of us have any experience in creating ships - the design is loosely based on what we "think" is a good hull design.

However, to further strengthen the structure, inlays are made by slicing a box at the edges of the ship hull. These are then to be produced in the machine workshop at the department of electronics.

To get a better understanding of ship dynamics a rapid prototyping technique has been used, where 3D models of the ship have been printed - and if it didn't live up to our demands (visually deciding if the design was flawed), a revised model was then printed. Using an iterative process - the final design was reached.

3.1 Bow thruster

The ship design includes a bow thruster, which can be used to perform precision maneuvers in tight spaces. This component is not currently mounted on the ship nor is it included in our control algorithms, because it has little or no effect on the ship when used at a speed than exceeds the normal operating boundaries, because the dynamic buoyancy will lift the front of the ship above the water. If included, it would be controlled by the LLI with a direction and a PWM signal.

The electronic design of the bow thruster controller can be found in subsection 2.3: Bow thruster controller.

3.2 Ship hull

The hull of the ship follows a soft-chined displacement design, because in normal operating conditions the static buoyancy will dominate over dynamic. This shape is less affected by sea currents, and more by the wind. According to the data series of the British Oceanographic Data Centre[1] near the shores of Greenland the drift caused by the steady currents is generally higher than the wind with a more randomly distributed blowing direction.

[1] https://www.bodc.ac.uk/data/online_delivery/nodb/search/

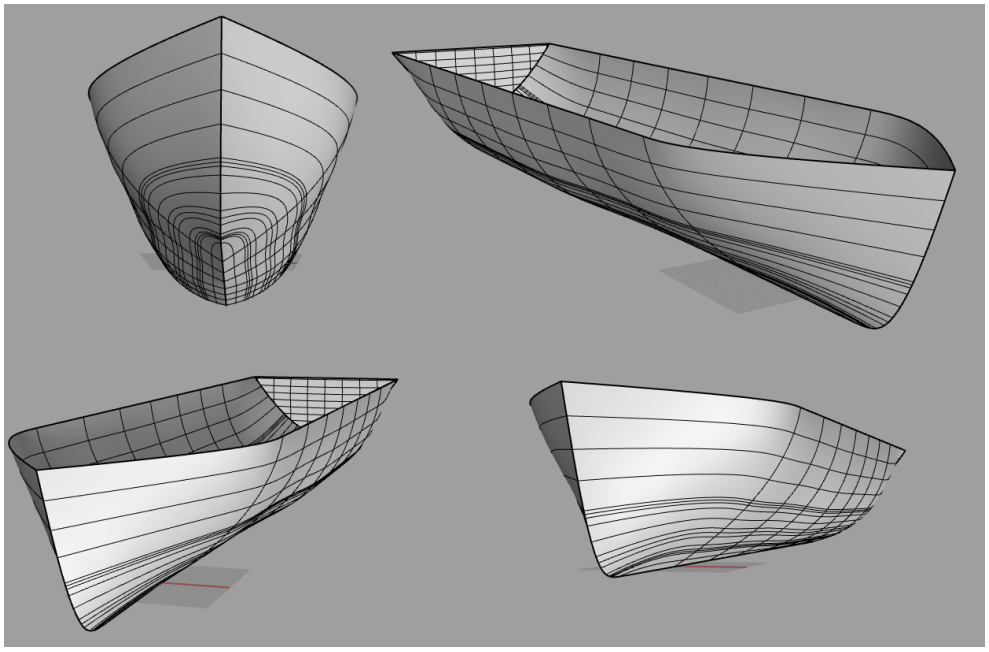


Figure 3.1: The shape of the hull

The ship is outfitted with powerful main engines, so the vessel can perform fast and precise maneuvering without the bow thruster. Though these engines provide a lot of dynamic range, the ship hull was not designed for high speed 3.2. The problem is caused by the size and power of the propellers. At slower speeds this effect ceases. Mirroring the setup and reversing the rotation movement would solve the pushing problem, but instead the back of the ship is pushed upwards, and air is sucked between the propellers. Instead, an additional vertical fin has been installed above the propellers, which keeps the water from being pushed upwards. As a positive side effect, the water after the hull is significantly less disturbed, thus the drag has been reduced as well. This has a positive effect on the range through the battery life 3.4.

Figure 3.2: An excessive and unregulated bouncing can be experienced at high speeds.

Figure 3.3: The power of the two engines force the water upwards between the propellers, therefore pushing the back of the ship down

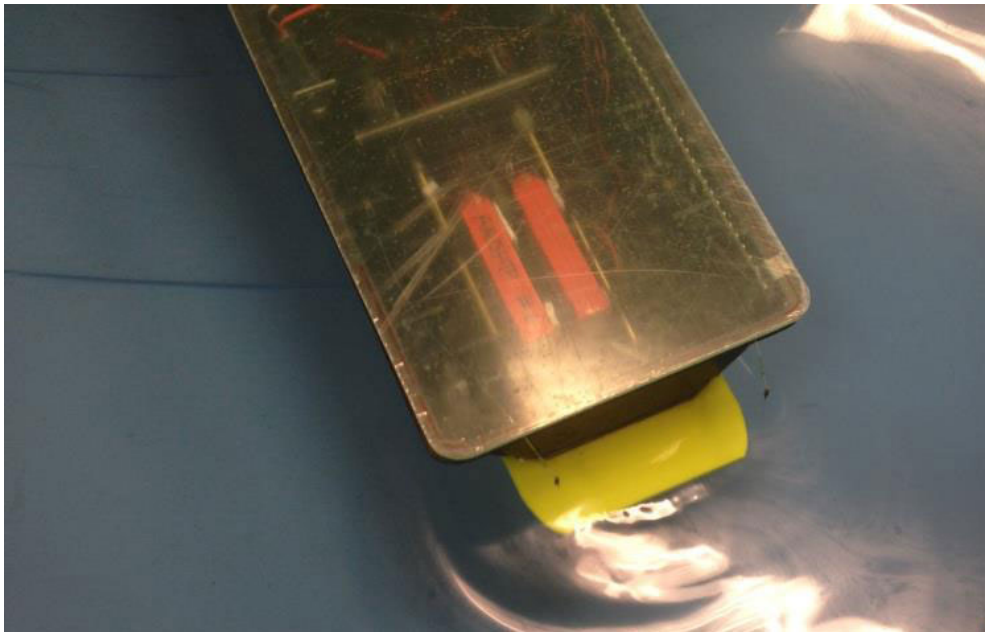


Figure 3.4: The rotating propellers with the fin mounted above them. The engine power is set in the normal operation range

Comunication Protocol 4

Documentation for the LLI part of the system. This describes the packet format and defines all commands.

4.1 Standard format of messages

To develop a suiting protocol for AAUSHIP1, the data to be sent via this is looked at in more detail in section 4.3 on page 14. For instance, the Inertial Measurement Unit (IMU) has several different outputs, which need to be processed independently of each other. Because of this, they are broken into separate packages, each with their own ID and checksum. This simplifies parsing and allows the protocol to easily be expanded to other sensors and actuators which were not included in the original design. A short, single state packet limits the effect of a flipped bit, since only a single state has been lost, compared to a packet containing all information.

As both the IMU and Global Positioning System (GPS) is sending packets of varying size, the data field in the protocol should be able to accommodate this variation. However, there are some fixed elements, on which it is possible to decide now. The number of sensors/actuators connected to the LLI was decided by design not to be more than 256, which makes way for a 1-byte device resolution. As each device might contain several outputs (as seen from the IMU) each device ID is then given an additional byte for message IDs. In addition to this header a length byte is contained to describe the length of the data field, to make it easier to parse. Lastly a Cyclic Redudancy Check (CRC) checksum is added on the end to verify the content. Figure 4.1 depicts the packet structure. Everything other than the data field is fixed length as described in table 4.1 on the next page. The data field can be either binary or ASCII, depending on the source. It is desired to limit the amount of calculations necessary at the LLI, seeing as the computational power of the HLI is many times greater, as well as the implementation being simpler due to the higher level scripting language being used (Python).

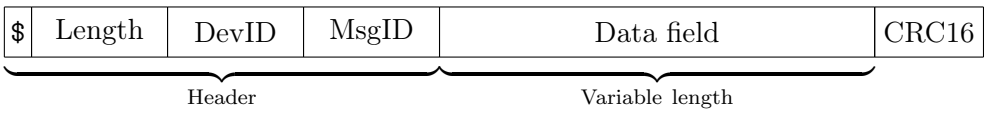


Figure 4.1: Generic message bytefield

The packet bytes are little endian (MSB first), which is consistent throughout the packet. The data field is arranged as always being a full number of bytes, to simplify transmission and receiving. This conforms to the RS232 standard.

,

Field name	Size [bytes]	Type	Description
Startchar	1	uint8	Start character ($)
Length	1	uint8	Length of data field in the range 1–255
DevID	1	uint8	Device identifier
MsgID	1	uint8	Message identifier
Data	1–255	uint8	Data portion (binary or ASCII)
Checksum	2	uint8	CRC-16 checksum on data part

Table 4.1: General description of the packet format

The device ID (DevID) also serves as the priority of the packets, enabling more important packages to be sent prior to less important ones. For example; auxiliary parameters as temperature measurements are less important in time than navigational informations from the IMU which has to be precisely known in time and preferably periodically.

4.2 Message definitions

The initial list of supported devices and message IDs is given in the tables at the end of this section : 4.3. The initial devices are distributed evenly with 10 IDs between each device to allow for other new devices to be implemented with lower or higher priority IDs. The general functions of the LLI are implemented with the highest available priority, since it includes the deadman switch implemented and a very low amount of regular activity. From the definition of the DevID and MsgID each packet can be identified as having to do with a given part of the system. This is the same throughout the system. For some parts of the system, such as the motors, it is not possible to see whether the packet is a set or a get command, from the initial data structure. To mediate this, a "Read thruster" command is implemented, and should be implemented for all future actuators or other devices which are not just passively feeding back data. This would be achieved by sending a "Read thruster" packet to the LLI from the HLI. The LLI will neglect the data part of this message, as the intention of the package is defined by the special combination of DevID and MsgID. It will then respond with the same MsgID and DevID, but with the data is has read from the MotorSpeed register. To facilitate the Plug and Play nature of the system, the first 3 available MsgIDs are defined to have a set meaning. These are given in table 4.2. If a packet with MsgID 0 is sent to a device, the device will respond with a list of all available MsgIDs, along with their definition. If a packet with MsgID 0 is sent to DevID 0, the General LLI, it will respond with a list of available devices. If these device definitions adhere to a strict protocol, it will be possible to automatically configure an LLI and HLI implementing this protocol. This protocol will not be defined in this project,

MsgID	Definition
0	List available MsgIDs along with their functionality
1	Set options for device
2	Read options from device

Table 4.2: Msgid

but the feature will useful in checking and validating the configuration.

An x in the first column indicates that functionality is implemented on LLI. The allocated functions is listed in tables 4.3 on page 16, 4.4 on page 17, 4.5 on page 18 and 4.6 on page 19.

The protocol is implemented as a class on the HLI.

4.3 Analysis of data and bandwidth

Some considerations when making remote control computations over a simplex Radio Frequency (RF) link is needed to be able to have control of the information flow between the LLI and HLI.

One of the features of the Kalman-filter is that it makes it possible to have a better reaction to a missing packet. This comes from the fact that a missing packet can merely be treated as a packet with very high noise, meaning a Kalman gain of 0. With a Kalman gain of 0, the measurement included in the packet will not be considered for the next prediction. This is a key feature of the filter as well as a key feature of the project.

If a GPS measurement is lost the ship is effectively sailing blindfolded. If however measurements of other parameters are available, it is possible to very effectively predict where the ship is positioned and what heading it has. With this information it is possible to still control the ship, remotely, until a new fix is received. This makes the system robust in regards to packetloss, as it is estimated that only a limited amount of the packets will be lost.

If we assume that 10% of the packets will be lost, regardless of the type, we can estimate the approximate average sampling rate. This can be calculated from equation (4.1). This is quickly observable by the fact that a packetloss of 0 results the actual sampling rate, a packetloss of 50% results in half the sampling rate, and a packetloss of 100% results in an 0 sampling rate.

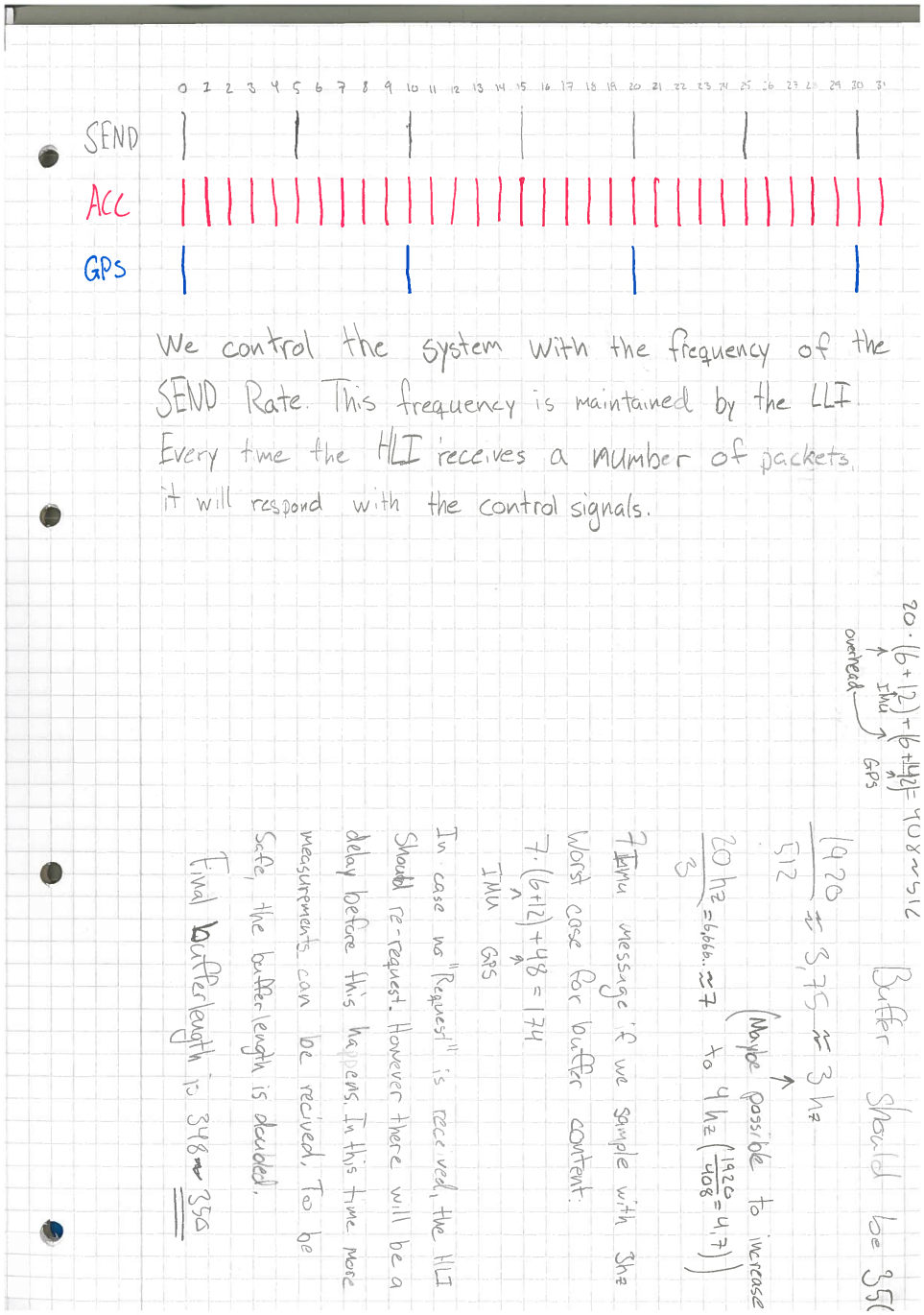
$$\hat{\mathbf{f}}_\mathbf{s} = f_s(1-p) \tag{4.1}$$

Where $\hat{\mathbf{f}}_\mathbf{s}$ is the effective average sampling speed
f_s is the actual sampling speed
p is the probability for packet loss

From this it can be seen that a system implementing this form of Kalman filtering as a way to deal with packetloss can effectively be estimated as a lossless system with a lower sampling rate.

It can also be seen that there are two ways to generate a better estimate – reduce the probability of packetloss or increase the sampling rate. If we assume a fixed samplingrate we can estimate p and, in many cases, decrease it by increasing the signal strength. If an estimate of p is achieved, the actual sampling rate can be calculated by giving a desired effective sampling rate and calculating the necessary requirement.

The packetloss can be handled by having a matrix with the index on the diagonal corresponding to a given type of sample being either 1 or 0, depending on whether a sample is available. Multiplying this with the Kalman gain matrix, B, will set the corresponding column to 0, which is what is desired.


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
SEND
ACC
GPS
We control the system with the frequency of the SEND Rate. This frequency is maintained by the LLI. Every time the HLI receives a number of packets, it will respond with the control signals.
20 · (6 + 12) + 6 + 42 = 410 ≈ 512
IMU
GPS
overhead
Buffer Should be 350
1920 / 512 ≈ 3,75 ≈ 3 hz
(Maybe possible to increase)
20 hz / 3 = 6,666... ≈ 7 to 4 hz (1920 / 1408 = 4,7)
7 IMU message if we sample with 3hz
Worst case for buffer content:
7 · (6+12) + 48 = 174
IMU
GPS
In case no "Request" is received, the HLI should re-request. However there will be a delay before this happens. In this time more measurements can be recived, To be safe, the buffer length is decided.
Final buffer length is 348 ≈ 350

	Message Name	MsgID	Data Size	Contains
0: General LLI				
	List available devices	0	Up to 255	Return a message for every available device
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
	Deadman Switch	3	1	On or off signal for actuators auto zero
	Status	4	Up to 255	Statuses of sensors and actuators
x	Ping	5	0	Empty
x	Pong	6	0	Empty
	ACK	7	0	Empty
	NACK	8	0	Empty
x	Build Info	9	Up to 255	Commit, target, date
	Surge	10	1	Speed
	Turn	11	1	Turning velocity

Table 4.3

	Message Name	MsgID	Data Size	Contains
10: Actuators				
	List available commands	0	Up to 255	Return a message for every available MsgID
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
x	Set PWM actuator 1	3	2	Engine 1, starboard side (default)
	Read PWM actuator 1	4	2	Engine 1, starboard side (default)
x	Set PWM actuator 2	5	2	Engine 2, port side
	Read PWM actuator 2	6	2	Engine 2, port side
x	Set PWM actuator 3	7	2	Engine 3 (auxiliary)
	Read PWM actuator 3	8	2	Engine 3 (auxiliary)
x	Set PWM actuator 4	9	2	Rudder
	Read PWM actuator 4	10	2	Rudder
x	Set PWM actuator 5	11	2	Auxiliary
	Read PWM actuator 5	12	2	Auxiliary
	Set PWM output 1	13	2	Bow thruster
	Read PWM output 1	14	2	Bow thruster
	Set PWM output 2	15	2	Auxiliary
	Read PWM output 2	16	2	Auxiliary
	Set PWM output 3	17	2	Auxiliary
	Read PWM output 3	18	2	Auxiliary
x	Set PWM output 1 and 2	19	4	Engine 1 and 2
	Read PWM output 1 and 2	20	4	Engine 1 and 2

Table 4.4

	Message Name	MsgID	Data Size	Contains
20: IMU				
	List available commands	0	Up to 255	Return a message for every available MsgID
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
	X-Acceleration	3	2	Acceleration in X-direction
	Y-Acceleration	4	2	Acceleration in Y-direction
	Z-Acceleration	5	2	Acceleration in Z-direction
	X-Gyro	6	2	Gyroscope in X-direction
	Y-Gyro	7	2	Gyroscope in Y-direction
	Z-Gyro	8	2	Gyroscope in Z-direction
	X-Mag	9	2	Magnetometer in X-direction
	Y-Mag	10	2	Magnetometer in Y-direction
	Z-Mag	11	2	Magnetometer in Z-direction
	Temp	12	2	Temperature in IMU
x	Burst read	13	24	Reading with all sensor data

Table 4.5

	Message Name	MsgID	Data Size	Contains
30: GPS				
	List available commands	0	Up to 255	Return a message for every available MsgID
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
	Velocity	3	2	Velocity
	Latitude	4	4	Latitude
	Longitude	5	4	Longitude
x	Reduced RMC	30	6	Cut of RMC sentence with the main part
40: Temperature				
	List available commands	0	Up to 255	Return a message for every available MsgID
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
	Temp 0	3	2	Temperature
	Temp 1	4	2	Temperature
	Temp 2	5	2	Temperature
50: Voltage				
	List available commands	0	Up to 255	Return a message for every available MsgID
	Set options	1	Up to 255	Option bytes
	Read Options	2	Up to 255	Option bytes
	Voltage	3	Up to 255	Voltage

Table 4.6

5 Control

5.1 Overview of the control system

This is an overview of the information flow for control in regard to the rest of the system containing, sensors, LLI and HLI.

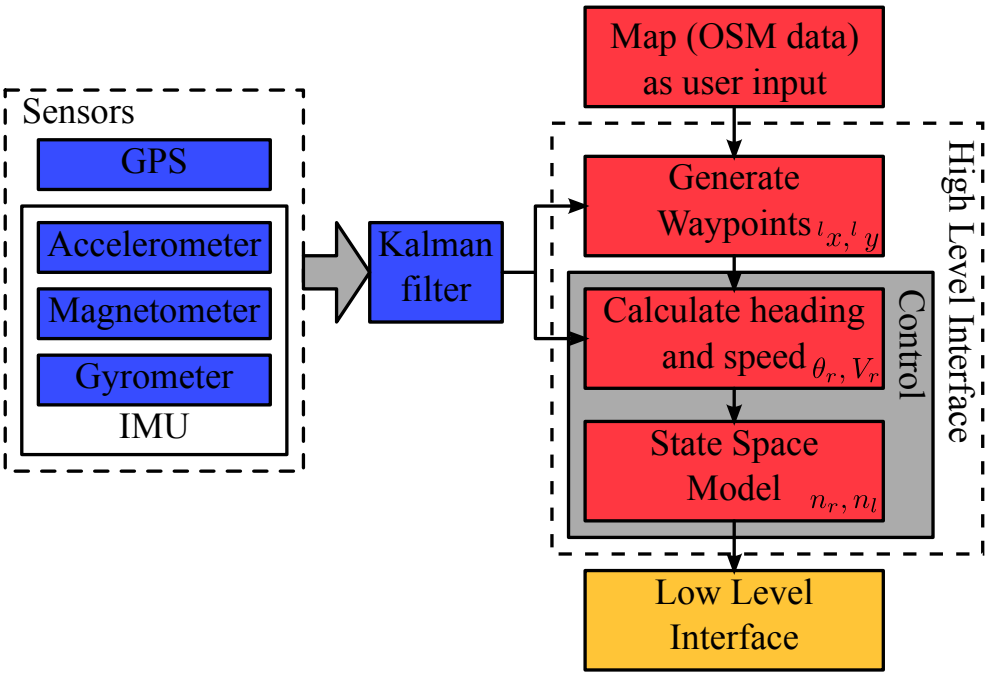
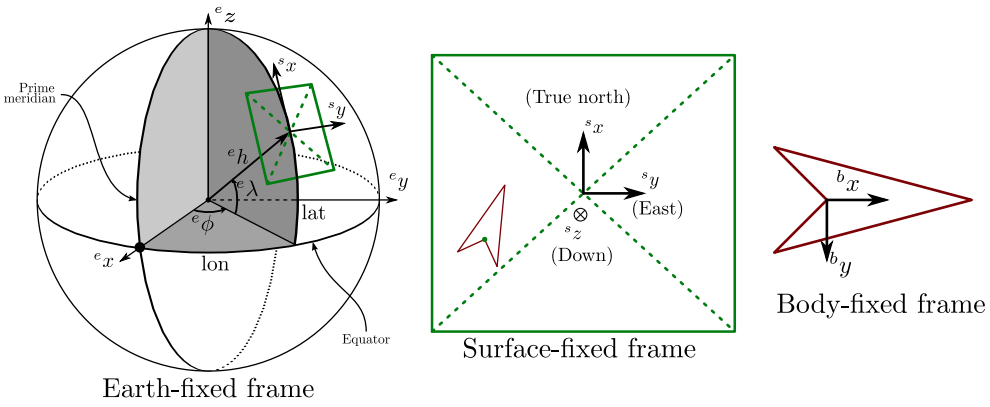


Figure 5.1: Overview of control information flow in regard to the subsystems. The blue part indicates information that is not fetched fed to the control part, it is electrically connected to the LLI whilst being forwarded to the HLI. The red parts is what happens on the HLI. The yellow part is the LLI itself.

5.2 Frames of reference

There are different frames of reference, which is defined here. There are the global frame which is the the earth, this frame operates in various coordinate systems such as the World Geodetic System 1984 (WGS84) and Earth-Centered, Earth-Fixed (ECEF). In addition to this there is a local frame/inertial frame, which is a tangential plane on the surface of the earth int he area the vessel has to operate. Lastly the vessel has a frame attached to this, which is the body frame. All x,y,z frames are righthanded.

Rotation matrices could be of interest here, but see Nielsen et al. [2012] for now.



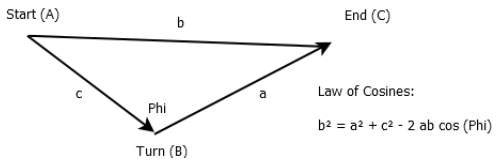
Note: Theese the x and y variables could be swapped in some of our software, because we started on that earlier than we defined theese frames.

Figure 5.2: Definitions of the reference frames, using the same system as Nielsen et al. [2012].

5.3 Control System

The stages on figure 5.2 is explained here in detail. In the First Stage the required heading of the ship is calculated that keeps the course closest to the planned waypoint.

In order to efficiently and accurately navigate along the path, a set of Sub-Waypoints is calculated for each route between two Waypoints. The main control strategy is to navigate through all of these SWPs in a predefined order, one by one. The heading of the ship is defined in NED coordinate system. The required heading is determined by the Law of Cosines, based on the Position of the Ship and the Position of the next Sub-Waypoint.



Problems rise and corrections are necessary, if the heading of the ship θ is $\theta < -\pi$ or $\theta < \pi$. The heading of the ship is calculated based on the Gyro sensor and the heading can have any value in the form of:

$$\theta = [-\pi; \pi] \pm 2 \cdot k \cdot \pi \tag{5.1}$$

Before invoking the control procedure, all of the heading angles must be transformed into the $[-\pi; \pi]$ interval. This procedure causes a possible error though.

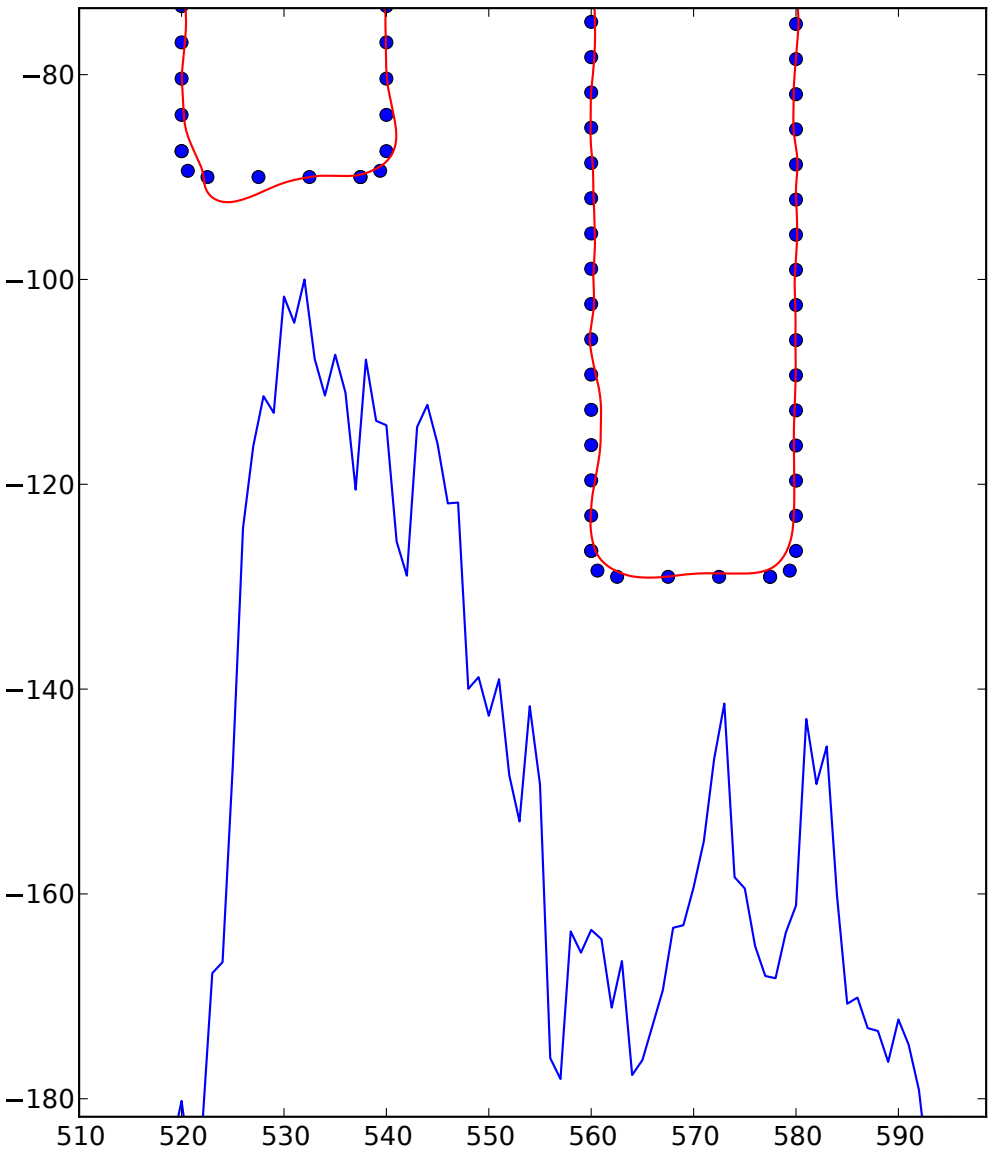
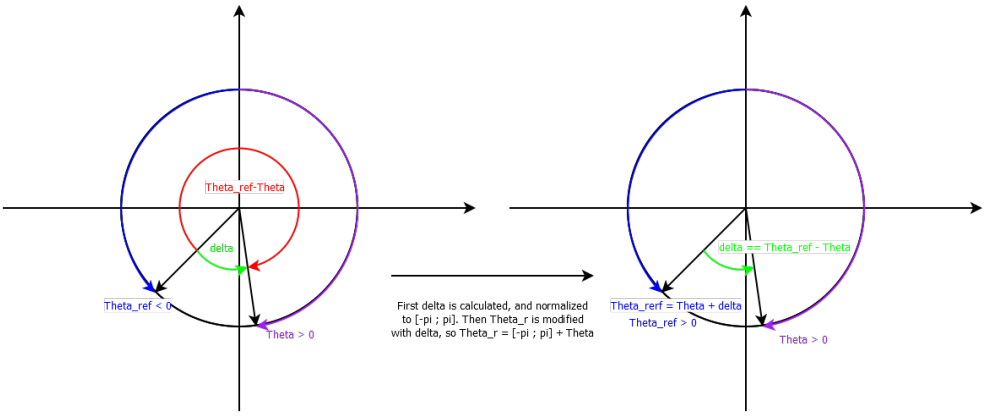


Figure 5.3: Part of the simulated path of the embedded system close to the shore along the sub-waypoints, during a regular oceanography task



The required heading or the heading of the ship must be transformed into a different representation, where

$$|\theta_r - \theta| < \pi$$

. To keep a consistent heading representation, first the deviation angle

$$\theta = \pi_r - \pi \tag{5.2}$$

is calculated, than transformed to the $[-\pi; \pi]$ interval and finally, with δ we can transform θ_r to

$$\theta_r(\theta) = \theta + \delta \tag{5.3}$$

If the conditions above are met, θ and $\theta_r(\theta)$ will always yield values that result in correct controller output.

Subwaypoint Validity

The overall navigation can be improved, if the Target SWPs are considered "reached" in a certain distance. Optimally this distance (Validity distance) should be somewhat longer than the minimum turning radius of the ship (R_{min}). If it is shorter, the ship might not be able to reach the SWP, if it is way longer, the ship will divert from its course in turns.

5.4 Control loop theory

The inner control is implemented by a State-Space controller, based on the difference of the estimated and the reference states, speed and heading:

$$Q = F(X - \hat{\mathbf{x}}) \quad , \quad x = [v, \theta]^T \tag{5.4}$$

Linearizing the outputs

Using Newton's second law of motion for forward and rotational movement ($F = m \cdot A$ and $T = I \cdot \dot{\omega}$), we can easily see that the inputs to our system will be F and T representing the forward force and the torque produced by the motors. It is worth mentioning that, since there are two propellers rotating in opposite directions, a torque that would induce a rotation around Z axis passing through the boat's center of mass only if the two propellers are rotating at different speeds.

These two rotational speeds N1 and N2 were initially included in the state space model as inputs to the system, as they could be controlled directly via a PWM pulse. This though proved to complicate matters because the relationship between the speeds of the propellers N1 and N2 and the force F and torque T generating accelerations is not a linear one. Because of this, our system would not be linear and would be beyond the scope of this project.

$$F_1 = C_1 \cdot \text{abs}\{n_1\} \cdot n_1 \tag{5.5}$$
$$F_2 = C_1 \cdot \text{abs}\{n_2\} \cdot n_2 \tag{5.6}$$

Where $C_1 = \rho \cdot K_T \cdot D^4$, which are constant for both the engines. The two propellers thus produce a total force as a sum of these functions. The torque produced by the propellers can be described as F multiplied with the distance to the propeller l and the angle of attack on the ship. This angle is the same for both engines (with opposite sign), as they are mounted symmetrically around the centre.

$$\tau_1 = F_1 \cdot l \cdot \sin(\theta_{\text{stbd.}}) \tag{5.7}$$
$$\tau_2 = F_2 \cdot l \cdot \sin(\theta_{\text{port}}) \tag{5.8}$$

The above can be put on matrix form, and gives the following equation which can be used to solve for n_1 and n_2.

$$\begin{bmatrix} n_1^2 \\ n_2^2 \end{bmatrix} = \begin{bmatrix} C_1 & C_1 \\ C_1 \cdot l \cdot \sin(\theta_{\text{stbd.}}) & C_1 \cdot l \cdot \sin(\theta_{\text{port}}) \end{bmatrix}^{-1} \begin{bmatrix} F_{\text{desired}} \\ \tau_{\text{desired}} \end{bmatrix} \tag{5.9}$$

Whats left to do is to solve for n_1 and n_2. These can be solved by:

$$n_1 = \frac{n_1^2}{\text{abs}\{n_1^2\}} \cdot \sqrt{n_1^2} \tag{5.10}$$
$$n_2 = \frac{n_2^2}{\text{abs}\{n_2^2\}} \cdot \sqrt{n_2^2} \tag{5.11}$$

In order to prevent this from happening, the state space model was updated to only calculate the force T and torque T necessary for driving and turning the ship, and this would keep our system linear. in order to transform from

[T, F] to [N1, N2], another module was introduced in the system which would implement this non-linear function.

This is a better solution anyway because if the state space model only outputs T and F, then it could be used for some other ship configurations, without modifications to the model, only to the module that translates the force and torque to N1 and N2. This, for example, can allow us to change the size of the propellers or their positions.

Linearizing the drag forces

The drag forces acting on the bow of the ship while it's moving through the water are influenced by the area of the hull that is below water depth in the direction of movement, the density of the water, a drag constant C_D , and the square of the ship's speed components v_x and v_y.

A worthy note is the fact that the drag coefficient C_D will have two values depending on the direction of the ship's movement. When moving forward in the x direction, the coefficient C_{Dx} will be approximated for a triangular shape, which is the closest simple shape that we have values for. For lateral movement in the y direction, C_{Dy} is given for a square box. These approximations should not present a problem in the operation of our control system as the errors ca easily be corrected and adjusted for by a good controller.

$$D_x(v_x) = \frac{C_{Dx} \cdot \rho_{water} \cdot A_{front} \cdot v_x^2}{2}$$

$$D_y(v_y) = \frac{C_{Dy} \cdot \rho_{water} \cdot A_{side} \cdot v_y^2}{2}$$

Where $D_x(v_x)$ and $D_y(v_y)$ are the drag forces dependent on the forward and lateral speeds respectively, ρ_{water} is the density of the water, A_{front} is the frontal area of the hull and A_{side} is the lateral area.

The drag torque that is produced by the drag force acting on the hull of the ship, as the ship rotates through the water is dependent on the square of the angular velocity ω:

$$\tau(\omega) = \frac{C_D \cdot \rho_{water} \cdot d \cdot (r_f^4 + r_b^4) \cdot \omega^2}{8}$$

Where $\tau(\omega)$ is the torque generated by the angular speed ω, d is the depth of the ship that is submerged in water, r_f and r_b are the radii of the ship from the center of mass towards the front and back end, respectively.

These forces are obviously non-linear and thus cannot be integrated into the linear Kalman filter or State Space Model. Because we will usually use a constant speed it is feasible to linearize these forces for that value and not

have big errors. The angular drag torque can also be linearized around a mean turning speed because the radii of the corners of the paths are also constant.

Approximation of the drag forces around the points of interest v_l and ω_l is done using the first order term of the Taylor expansion:

$$y_l(x) = f(a) + f'(a)(x - a)$$

Where $y = f(x)$ is the function we want to linearize around a value of interest a. Applying this formula to our situation yields:

$$Dx_l(v_x) = \frac{C_{Dx} \cdot \rho_{water} \cdot A_{front} \cdot v_l^2}{2} + (C_{Dx} \cdot \rho_{water} \cdot A_{front}) \cdot (v_x - v_l)$$

$$Dy_l(v_y) = \frac{C_{Dy} \cdot \rho_{water} \cdot A_{side} \cdot v_l^2}{2} + (C_{Dy} \cdot \rho_{water} \cdot A_{side}) \cdot (v_y - v_l)$$

$$\tau_l(\omega) = \frac{C_{Dy} \cdot \rho_{water} \cdot d \cdot (r_f^4 + r_b^4) \cdot \omega_l^2}{8} + \frac{C_{Dy} \cdot \rho_{water} \cdot d \cdot (r_f^4 + r_b^4) \cdot \omega_l}{4} \cdot (\omega - \omega_l)$$

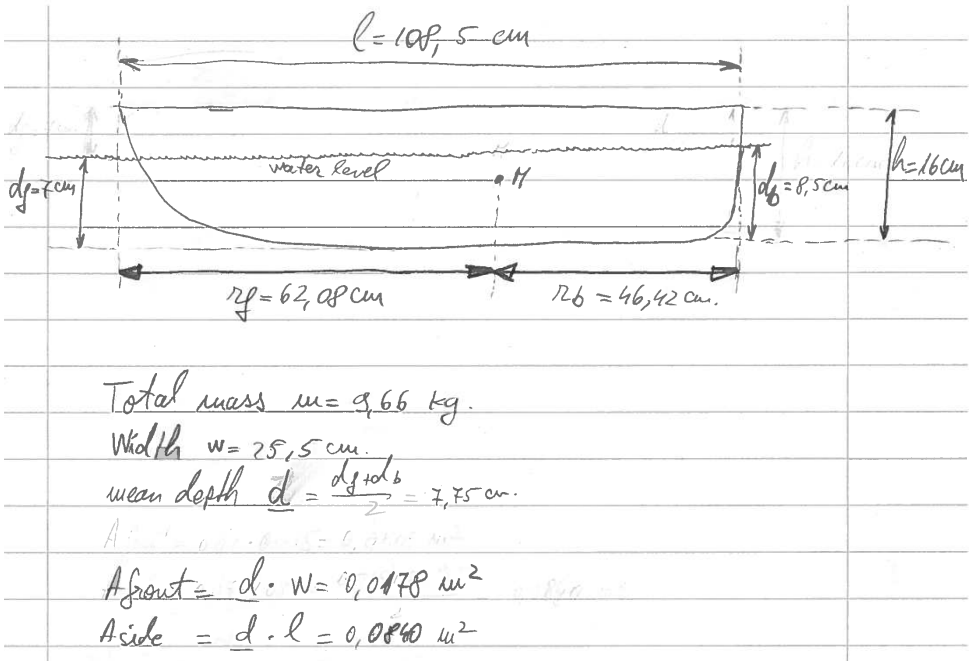


Figure 5.4: Sketch of ship sizes as measured after the maiden voyage

Given figure 5.4 on the previous page: $v_l = 1m/s$, $\omega_l = 1s^{-1}$, $C_{Dx} = 0.5$ (triangle) , $C_{Dy} = 1$ (box) , $\rho = 1000kg/m^2$, $A_{front} = 0.0178m^2$, $A_{side} = 0.0840m^2$, $d = 0.0775m$, $r_f = 0.6208m$, $r_b = 0.4642m$, the linearized formulas for the drag forces can be computed to be:

$$Dx_l(v) = -4.45 + 8.9 \cdot v_x \tag{5.12}$$
$$Dy_l(v) = -42 + 84 \cdot v_y \tag{5.13}$$
$$\tau_l(\omega) = -1.88 + 3.77 \cdot \omega \tag{5.14}$$

From which we can identify the coefficients that correspond tot the linear relation $y_l(x) = \alpha + \beta \cdot x$:

$$\alpha_x = -4.45 \qquad \alpha_y = -42 \qquad \alpha_\omega = -1.88$$
$$\beta_x = 8.9 \qquad \beta_y = 84 \qquad \beta_\omega = 3.77$$

Choosing the right states

The variables we chose to be the states in our model are the speed of the ship in the forward direction in the ship frame v, the angular position of the ship's x axis relative to true North θ and the angular speed ω.

$$\begin{cases} F - F_D = m \cdot \dot{v} \\ T - T_D = I \cdot \dot{\omega} \end{cases} \tag{5.15}$$

Where F and T are the force and torque produced by the propellers, F_D and T_D are the force and torque caused by water drag, m is the mass of the ship, I is the moment of inertia, $\dot{v}$ and $\dot{\omega}$ are forward and angular acceleration respectively.

The linearized drag force and torque from 5.4 can be substituted in these equations:

$$\dot{v} = \frac{1}{m}(-\beta_x \cdot v + \boldsymbol{F}) \tag{5.16}$$
$$\dot{\omega} = \frac{1}{I}(-\beta_y \cdot \omega + T) \tag{5.17}$$
$$\dot{\theta} = \omega \tag{5.18}$$

Note that we have also dropped the α term of the equations because we're currently only interested in the dynamic behavior of the system. The α offset will be compensated for when removing the steady state error. Consequently, these relations can be rewritten in a matrix notation as follows:

$$\begin{bmatrix} \dot{v} \\ \theta \\ \omega \end{bmatrix} = \begin{bmatrix} -\frac{\beta_x}{m} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{\beta_y}{I} \end{bmatrix} \cdot \begin{bmatrix} v \\ \theta \\ \omega \end{bmatrix} + \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & 0 \\ 0 & \frac{1}{I} \end{bmatrix} \cdot \begin{bmatrix} F \\ T \end{bmatrix} \tag{5.19}$$

Equation 5.23 corresponds to a 3 Degrees-of-Freedom (DOF) model for the ship - with the 3 degrees noted as movement in the X-direction (x), Y-direction (y) and the rotation about the Z-axis (θ). This model corresponds to the state equations, which are used as a base for developing the control algorithm.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{5.20}$$

Since we only need to output the velocity and the angle, the output matrix $\boldsymbol{C}$ is added to the system. No feedforward loop is implemented, so this part can be removed, resulting in the output equation for the system becomes:

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} \tag{5.21}$$

Discretizing for use with a particular timestep

The system described so far is continuous and needs to be discretized before being converted into an algorithm. Discretization is carried out in MATLAB using the `c2d` command. This command takes the continous time state space model and the discretization time step - and outputs the discrete equivalent. The `c2d` command uses a zero-order hold algorithm to discretize the system. The zero order hold algorithm is defined as:

$$\boldsymbol{x}(t) = \sum_{n=-\infty}^{\infty} \boldsymbol{x}[n] \cdot \text{rect}\left(\frac{t - T/2 - nT}{T}\right) \tag{5.22}$$

Where:

rect is a rectangular window function
T is the time interval

The matrices once discretized becomes:

$$\begin{bmatrix} \dot{v(k)} \\ \theta(k) \\ \omega(k) \end{bmatrix} = \begin{bmatrix} -\frac{\beta_x}{m} & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{\beta_y}{I} \end{bmatrix} \begin{bmatrix} v(k-1) \\ \theta(k-1) \\ \omega(k-1) \end{bmatrix} + \begin{bmatrix} \frac{1}{m} & 0 \\ 0 & 0 \\ 0 & \frac{1}{I} \end{bmatrix} \begin{bmatrix} F(k) \\ T(k) \end{bmatrix} \tag{5.23}$$

Optimal pole assignment

To achieve an optimal response of the system, an optimal pole placement strategy have been used. This is done by minimizing the cost function in equation 5.24. The optimal pole placement is then computed by solving the Algebraic Ricatti Equation.

$$\mathcal{J}(u) = \int_0^{\infty} (\boldsymbol{x}^T \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R}\boldsymbol{u}) dt \tag{5.24}$$

Again, this is done in MATLAB, and gives feedback gain:

$$\boldsymbol{F}_{opt} = \begin{bmatrix} 10.6956 & 0 & 0 \\ 0 & 2.2743 & 0.6681 \end{bmatrix} \tag{5.25}$$

This changes the input of the system to $\boldsymbol{u} = -\boldsymbol{F}_{opt}\boldsymbol{x}$, however, this does not take into account that we can end up with a steady state error. Therefore a reference gain is introduced.

Reference gain

The purpose of the reference gain is to zero out the steady state error. This is done by augmenting the state space system, and equating the error to zero. This can be written as:

$$\boldsymbol{0} = \boldsymbol{A}\boldsymbol{x}_{ss} + \boldsymbol{G}\boldsymbol{u}_{ss} \tag{5.26}$$

$$\boldsymbol{y}_{ss} = \boldsymbol{C}\boldsymbol{x}_{ss} + \boldsymbol{D}\boldsymbol{u}_{ss} \tag{5.27}$$

The gain N is now introduced as a scaling factor on the reference signal, to make this error zero. The input $\boldsymbol{u}$ of the steady state version is then changed to the reference signal multiplied with the steady state input of the system. Written on matrix form this becomes:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{I} \end{bmatrix} \tag{5.28}$$

Thus, solving for $\begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix}$ gives the following:

$$\begin{bmatrix} \boldsymbol{N}_x \\ \boldsymbol{N}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \boldsymbol{I} \end{bmatrix} \tag{5.29}$$

Thus the final control signal changing the input $\boldsymbol{u}$ to:

$$\boldsymbol{u} = -\boldsymbol{F}_{opt}\boldsymbol{x} + (\boldsymbol{N}_u + \boldsymbol{F}_{opt}\boldsymbol{N}_{\boldsymbol{x}})\boldsymbol{r} \tag{5.30}$$

The latter in equation 5.30 can be combined, and gives the following:

$$\boldsymbol{u} = -\boldsymbol{F}_{opt}\boldsymbol{x} + \bar{\boldsymbol{N}}\boldsymbol{r} \tag{5.31}$$

Thus giving the gain N

$$\boldsymbol{N} = \begin{bmatrix} 19.5956 & 0 \\ 0 & 2.2743 \end{bmatrix} \tag{5.32}$$

5.5 Implementation of control loop

The control loop is implemented in Python and is run on the HLI.

The control loop reads the data from the LLI communication and invokes the control function of the Ship Object at every timestep.

(Figure!)

The control function of the Ship Object processes the data sent by the LLI. The data is a 9-by-2 matrix, which consists of the measurement values and the validity of the measurements. A measurement is considered valid, if its validity value is 1.

$$M_{[9,2]} = \begin{pmatrix} M_1 & V_1 \\ M_2 & V_2 \\ M_3 & V_3 \\ M_4 & V_4 \\ M_5 & V_5 \\ M_6 & V_6 \\ M_7 & V_7 \\ M_8 & V_8 \\ M_9 & V_9 \end{pmatrix} \tag{5.33}$$

We can process this matrix to an array of measurements,

$$M_{1..9} = \left[x, \dot{x}, \ddot{x}, y, \dot{y}, \ddot{y}, \theta, \omega, \dot{\omega}\right] \tag{5.34}$$

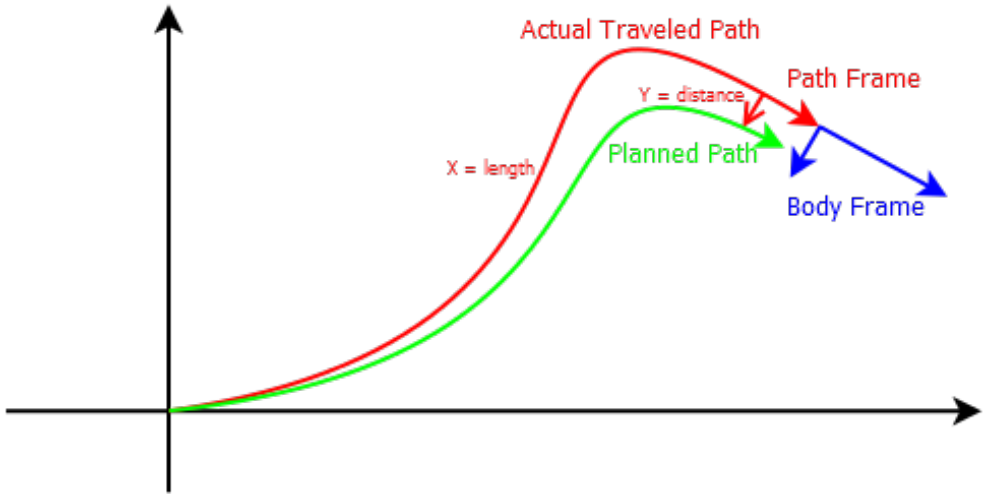
and a diagonal matrix of validities, which modifies the *Kalman Gain* matrix, in order to avoid the update of states with invalid or outdated measurement data.

$$V_{[9,9]} = \begin{pmatrix} V_1 & 0 & \cdots & 0 \\ 0 & V_8 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & V_9 \end{pmatrix} \tag{5.35}$$

The Kalman Filter works in the *Body Frame*, but the GPS Measurements are passed to the HLI in the *Local Frame*. Therefore some measurement values must be transformed.

The Position input of the Kalman Filter uses a hybrid frame, called *Path Frame*. This frame fixes the orientation of the ship to 0, but the x and y positions can vary. Therefore after transforming the measured GPS position from the *Local Frame* to the *Body Frame*, the *Path Frame* must be updated with the *Body Frame* positions, not completely replaced. The initial position

of the *Path Frame* is $P_{x,y} = (0,0)$, where x and y represents the distance traveled *forward* and *sideways* in the *Body Frame*.



5.6 Internal Measurement Unit

We are using an Internal Measurement Unit (IMU) to enhance the estimations and the control system by fusing the velocity data of the *GPS* and the *IMU* based on the accelerometer inputs. The IMU has the possibility to measure 6 degrees of freedom ($[\ddot{x}, \ddot{y}, \ddot{z}, \dot{Rot}_x, \dot{Rot}_y, \dot{Roz}_z]$). The IMU possesses significantly lower Signal-to-Noise Ratio (SNR) than the GPS velocity measurements. However, using the IMU only for the navigation is ill-considered, because of the small but substantial *bias* of the *Accelerometer* and *Gyro*.

$$\hat{\mathbf{x}} = x + C(t) = \int\int \ddot{x} dt dt \tag{5.36}$$

$$\hat{\mathbf{Rot}}_{\mathbf{x}} = Rot_x + C(t) = \int \dot{Rot}_x dt \tag{5.37}$$

Therefore we employ a Kalman filter, in order to implement the sensor fusion.

Challenges of the IMU signal-processing

The IMU is fixed to the body of the hull, so if the attitude of the Ship changes, so does the IMU. The data of the three accelerometers can be rotated to a fixed-attitude frame only if we possess the exact attitude data as well. But the attitude data over time will accumulate an unknown amount of offset error, because of (equation No. above). Since the GPS does not provide any information about the attitude of the ship, another sensor fusion is required, with a fixed outside point, like the Polaris star or the magnetic *North point*.

Pitch or roll

For the control system implementation we employed a simpler solution: assuming the ship will either have no roll or pitch, we could chose to improve either the position estimation or the velocity estimation, based on the natural accelerometer data only. The *Roll* and the *Pitch* are caused by *perpendicular* and *paralell* external forces attacking the hull of the ship, in respect to the motion. The measurements during various development phases suggested, that the variance of the fixed GPS position is very low, so enhancing the calculated position based on external forces is not required. In addition we can assume, that the sideways and vertical motion in still water is neglectable.

In the implementation, the sum of the acceleration vectors are calculated, and is compared to a reference gravity acceleration data:

$$\sum S = \ddot{\boldsymbol{x}} + \ddot{\boldsymbol{y}} + \ddot{\boldsymbol{z}} \tag{5.38}$$

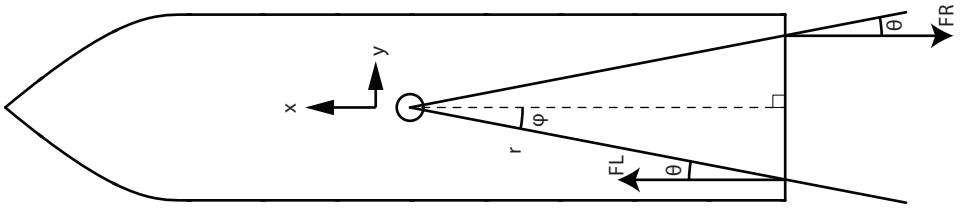
$$F = \arccos \frac{\sum S}{G} \tag{5.39}$$

In stormy weather the need for a more complex IMU processing is beyond doubt.

Modelling of Twin Screw Ship

6

When modelling the forces affecting a Twin Screw Ship (TSS) there few parameters which affect the movement of the ship. These are the forces applied by the motors, as well as the placement of the motors. For calculating these contributions simple trigonometric functions can be used. This gives the resulting forces in the boats X and Y directions, as well as the torque generated. First we see that the boat is mirrored along it's X axis. From this we can set



[h]

Figure 6.1: By controlling FL and FR, it is possible to control the translational and rotational forces of the ship. The division of these forces is dependant on the angle of attack, θ.

$\theta = \theta_L = \theta_R$ and $\phi = \phi_L = \phi_R$. Now we have the forces, F_L and F_R, as the controllable input to our system. These will be divided as into F_X, F_Y and τ, which are the variables that we wish to control. This division takes the form as seen in equation (6.1).

$$\begin{bmatrix} F_x \\ F_y \\ \tau \end{bmatrix} = \begin{bmatrix} \cos(\phi - \theta) \\ \sin(\phi - \theta) \\ \sin(\theta) \cdot r \end{bmatrix} \cdot F_R + \begin{bmatrix} \cos(\phi - \theta) \\ \sin(\phi - \theta) \\ -\sin(\theta) \cdot r \end{bmatrix} \cdot F_L \tag{6.1}$$

We observe that the motors will deliver force parallel to the x-axis, we can further set $\theta = \phi$. If this is the case $\sin(\theta - \phi) = 0$, which nullifies the F_Y term, and $\cos(\theta - \phi) = 1 \implies F_X = F_L + F_R$. The forces can then be modelled as equation (6.2).

$$\begin{bmatrix} F_X \\ \tau \end{bmatrix} = \begin{bmatrix} 1 \\ \sin(\theta) \cdot r \end{bmatrix} \cdot F_R + \begin{bmatrix} 1 \\ -\sin(\theta) \cdot r \end{bmatrix} \cdot F_L \tag{6.2}$$

6.1 State Space form

For better control of the system, it is desired to have the model of the ship in State Space form. This makes it possible to design an observer and a controller, which further makes it possible to design poles and zeros of the system.

The state space form for the system has the general form

$$\dot{x} = Ax + Bu \tag{6.3}$$

$$y = Cx + Du \tag{6.4}$$

Where x is the state vector.
$\dot{x}$ is the derivative of the state vector.
A is the state matrix
B is the input matrix
y is the output vector
C is the output matrix
D is the feedforward matrix

7 Noise of measurements

To facilitate a better estimate of the sensorvalues, the measurement noise is quantified and analysed, to allow for Kalman-filter to function as good as possible.

The noise can be generalized to the form

$$X(t) = Y(t) + W(t) \tag{7.1}$$

Where X is the measured signal
Y is the actual value
W is the added noise

When considering measurements there will typically be two kinds of noise: Gaussian and Uniform. The uniformly distributed noise comes from the rounding error from discretizing the measurements, also known as quantization noise, while the normally distributed noise is present in the measurements themselves. This usually arises from the effect of background noise, imperfections in hardware, thermal noise, magnetic fields, as generated by the motors, and other sources. While some of these noise sources may have other distributions than the normal distribution, they are calculated as normal in accordance with the central limit theorem. The quantization noise is included in this assumption. From this assumption it is possible to describe the noise using only two parameters: The mean, μ, and the variance, $sigma^2$.

7.1 Inertial Measurement Unit

The noise of the IMU is considering to be a zero mean process. From this it is easily seen that the variance of the noise can be estimated by letting the sensor be still and then logging the measurements. This process is documented in appendix 10. From this experiment the variance of the noise of the IMU measurements can be estimated as in table 7.1.

Device	$\boldsymbol{\sigma}^2$	
AccX	5.05	μG
AccY	4.98	μG
AccZ	5.96	μG
GyroX	2.28	$\frac{\mu^\circ}{s}$
GyroY	2.45	$\frac{\mu^\circ}{s}$
GyroZ	2.40	$\frac{\mu^\circ}{s}$
MagX		
MagY		
MagZ		

7.2 GPS

The GPS is an important part of the system, as the validity of the depth measurements are very dependent on the GPS estimate at that particular time. Therefore a very precise noise variance estimation is important in order to calculate the actual position. The GPS works by precisely measuring the time signals sent from a number of satellites. These satellites are not stationary which means that the signal can wander from time to time. This is obvious in the measurements which were done, where the signal is wandering until it reaches a steady state. This could be estimated by including a bias in the model and estimating this. It has been decided not to implement such a feature for the moment. Instead the variance of the GPS measurements is calculated with this bias included, resulting in a larger variance. According to the measurement journal, appendix 10, the variances are estimated to be 0.979 m and 1.12 m, for the x and y position, respectively.

7.3 Velocity

The estimate of the noise on the velocity measurements are assumed to be a zero-mean process with a given variance. The outputs of the measurements are however absolute values. This gives a different distribution of the measurements when the process is based around 0, than when it is distributed around another value. The distribution of the actual measurements is estimated to be shaped like the blue curve on fig 7.1, while the underlying distribution is actual normal, like the red curve. If a normal distribution is fitted to the sampled data, the result is the green curve. Luckily the unbiased sample variance is defined as equation (7.3), which means that the estimated sample variance is unconcerned with whether or not the samples are negative or absolute values.

$$s_{N-1}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2 \tag{7.2}$$

For a zero mean process the sample variance is easily computed to be , as documented in appendix 10, to be $0.0026\frac{\mathrm{m}}{\mathrm{s}}$.

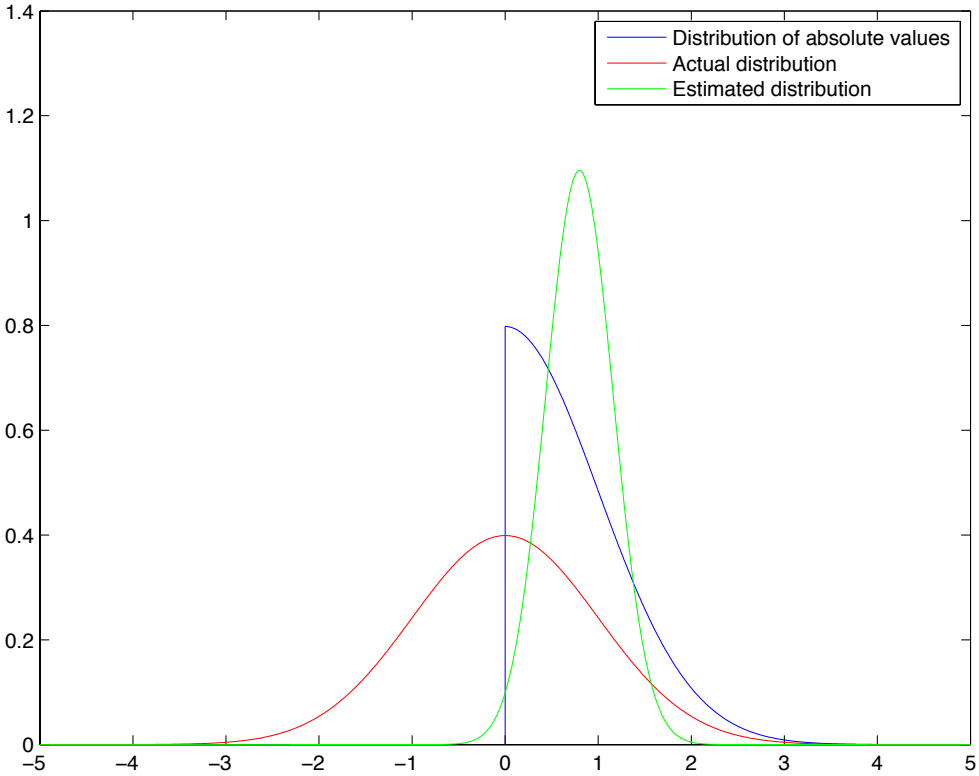


Figure 7.1: While the underlying distribution is normal (red), the absoluteness of the sampled data makes for a non-normal distribution (blue). If the variance and mean of the sampled data is taken, the result is the green curve.

8 State Estimation

To give a better position estimate which can be fed to the controller as well, the different data collected from the sensors mounted are put through a KF. This filter takes the different measurements as inputs and combines them even though they have different variances and sample rates in order to give a better estimate of the position, rather than the quite noisy measurements taken using just the raw GPS data directly.

To develop such a filter, the model of the ship is to be computed, as well as a mapping of the different inputs and outputs to the system. The model of the forward and sidewards case (surge and sway) are the same as for the discrete system.

State model

The state model is used as a base for computing the influence the different inputs have on the system. The below formula is a general description for the state space model of the system.

$$\boldsymbol{x}(k) = \boldsymbol{\Phi}(k)\boldsymbol{x}(k-1) + \boldsymbol{w}(k) \tag{8.1}$$

Where:

$\boldsymbol{\Phi}(k)$ is the state matrix
$\boldsymbol{x}(k-1)$ is the last input to the system
$\boldsymbol{w}(k)$ is the driving noise (the system input)

In this case, the driving noise $\boldsymbol{w}(n)$ will be considered to be the input to the system, which can then be used to estimate the different states. The states to be estimated are the velocity $\dot{x}$, the angular velocity ω and the angle of the vessel in the local frame θ. The driving noise (or input) can be defined as the $\boldsymbol{B}$ matrix in the state system, multiplied with the different inputs given to the system, namely n_1 and n_2. When inserted, the formula for the state model becomes:

$$\boldsymbol{x}(k) = \boldsymbol{\Phi}(k)\boldsymbol{x}(k-1) + \boldsymbol{w}(k) \tag{8.2}$$

The state model $\boldsymbol{\Phi}(k)$ can be seen as the same matrix 3-by-3 matrix inserted on the diagonal of a 9-by-9 matrix. This matrix expresses how an acceleration is turned into a position by defining the current position (or next) as a sum of the last position, the change due to the last velocity and the change due to the last acceleration, thus giving:

$$\mathrm{Pos}_x(k) = \mathrm{Pos}_x(k-1) + \mathrm{Vel}_x(k-1) \cdot ts + \mathrm{Acc}_x(k-1) \cdot \frac{ts^2}{2} \tag{8.3}$$

Using the same notation, it is possible to define the current velocity as:

$$\text{Vel}_x(k) = \text{Vel}_x(k-1) + \text{Acc}_x(k-1) \cdot ts \tag{8.4}$$

And finally define the acceleration as:

$$\text{Acc}_x(k) = \text{Vel}_x(k-1) \cdot \beta_x(k-1) + \text{Acc}_x(k-1) \tag{8.5}$$

As the input to the system is an acceleration, the $\text{Acc}_x(k-1)$ term in equation 8.5 is zeroed out, and is then input to the system via the input. These 3 formulas can be simplified and put on matrix form to make further computations easier, thus stating the state matrix as:

$$\boldsymbol{\Phi}_{X,Y,\omega}(k) = \begin{bmatrix} 1 & ts & \frac{ts^2}{2} \\ 0 & 1 & ts \\ 0 & -\beta_X & 0 \end{bmatrix} \tag{8.6}$$

Thus producing the state matrix as:

$$\boldsymbol{\Phi}(k) = diag\{\boldsymbol{\Phi}_X(k), \boldsymbol{\Phi}_Y(k), \boldsymbol{\Phi}_\omega(k)\} \tag{8.7}$$

$\boldsymbol{x}(k)$ is the state vector, and are given as:

$$\boldsymbol{x}(k) = [x(k), \dot{x}(k), \ddot{x}(k), y(k), \dot{y}(k), \ddot{y}(k), \theta(k), \omega(k), \alpha(k)]^T \tag{8.8}$$

It is not all the available states we want to output, but the more inputs the system have, the better the estimate becomes. As stated, the input to the system is given as an acceleration. This can be defined as an input given as a force and an input given as a torque. The input can be defined as $\boldsymbol{u}(k)$ (the controller output) this is multiplied with an augmented version of the $\boldsymbol{B}$ matrix from the state space model $\boldsymbol{B}_a$ to transform it to an acceleration thus can $\boldsymbol{w}(k)$ be described by:

$$\boldsymbol{w}(k) = \boldsymbol{B}_a \boldsymbol{u}(k) = \begin{bmatrix} 0 & 0 & \frac{1}{m} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I} \end{bmatrix}^T \begin{bmatrix} F(k) \\ \tau(k) \end{bmatrix} \tag{8.9}$$

Observation model

The observation model, is a model that models the different observations. In this case, the different observations are measured directly, as we can measure both the angular velocity, the angle and the velocity of the craft. The general formula for the observation model is given as:

$$\boldsymbol{z}(k) = \boldsymbol{H}(k)\boldsymbol{x}(k) + \boldsymbol{v}(k) \tag{8.10}$$

Where:

$\boldsymbol{H}(n)$ is the model linking the measurements to the observations
$\boldsymbol{v}(n)$ is the measurement noise on the sensors

The noise from the measurements is estimated using previous measurements which can be used to estimate the variance and the mean of the measurements. The noise can in general be seen as zero-mean Gaussian white noise processes, which makes for the assumption:

$$\boldsymbol{w}(n) \sim \mathcal{N}(0, \sigma_Z^2) \tag{8.11}$$

As $\boldsymbol{x}(n)$ is a row vector, $\boldsymbol{w}(n)$ is also a row vector with the same dimension. This calls for different variances on the different noise additions, for each of the measurements. As the variance of the noise on the IMU is a lot bigger than on the GPS. As all the measurements are available directly, the $\boldsymbol{H}(n)$ matrix is equal to identity. Giving the final observation model:

$$\boldsymbol{z}(n) = \boldsymbol{x}(n) + \boldsymbol{v}(n) \tag{8.12}$$

The Covariance matrices

As the KF is given as a vector KF, the covariance matrices is to be computed. The definition for a covariance matrix is given as:

$$Cov(\boldsymbol{X}, \boldsymbol{X}) = \mathrm{E}\langle[\boldsymbol{X} - \boldsymbol{\mu}_X][\boldsymbol{X} - \boldsymbol{\mu}_X]^{\mathrm{T}}\rangle \tag{8.13}$$

The covariance matrix is used to tune the KF, and weighs the different inputs according to the noise they experience. An assumption is to keep this constant. For the vector KF there are two noises added to the system, one depicts the measurement noise, and the other the system noise. The system noise is in this project considered the input to the system. This can therefore be seen as the covariance of the input signals to the ship. As seen in 7 on page 37 the distributions of the input signal $\boldsymbol{u}(k)$ can be seen as:

$$F \sim \mathcal{N}(5.3544, 55) \tag{8.14}$$
$$\tau \sim \mathcal{N}(0, 20) \tag{8.15}$$

The covariance of these inputs, are then multiplied with $\boldsymbol{B}$ to give the input to the system $\boldsymbol{w}(k)$. As the two inputs are independent, the covariance matrix collapses to a matrix with diagonal entries, which gives the following matrix:

$$Cov(\boldsymbol{X}, \boldsymbol{X})_{3,3} = \sigma_F^2 \tag{8.16}$$
$$Cov(\boldsymbol{X}, \boldsymbol{X})_{9,9} = \sigma_\tau^2 \tag{8.17}$$

When the system is simulated it gives the following response for estimating the position. Figure 8.2 on page 46 depicts how the system responds when running the measurements at the same sample rate. However - the GPS only samples

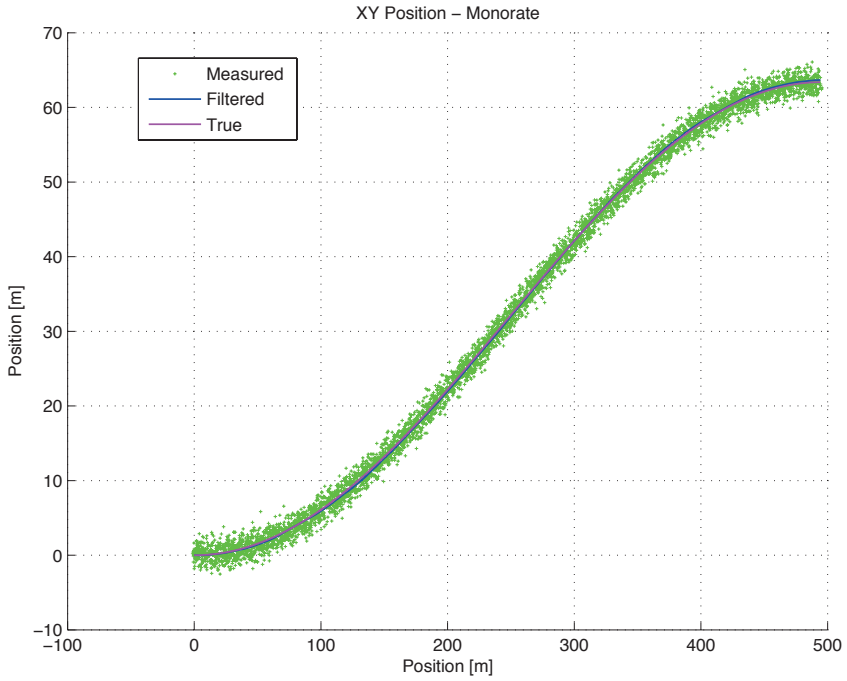


Figure 8.1: Test to see if the KF estimates the position

at 1Hz and the IMU samples at 10Hz, the filter needs to be changed. The KF must account for this change, which can be done by setting the noise of the measurement to a high value, which in turn will reduce the Kalman gain $\bar{\boldsymbol{K}}$ to zero, or just set the Kalman gain to zero. Both methods provide different results, as the estimate would then converge towards the last fixed value, whilst setting it to zero, would make the system use the other measurements available to the filter, and thus given an estimate, rather than convergin towards a fixed value.

8.1 KF description

The KF can be divided into two steps, an update step and a prediction step. The update step updates all the variables, while the prediction step does the actual prediction. The update step is given as:

$$\boldsymbol{x}(k) = \boldsymbol{\Phi}(k)\boldsymbol{x}(k-1) + \boldsymbol{w}(k) \tag{8.18}$$

$$\boldsymbol{z}(k) = \boldsymbol{H}(k)\boldsymbol{x}(k) + \boldsymbol{v}(k) \tag{8.19}$$

$$\hat{\boldsymbol{x}}(k)_{(-)} = \boldsymbol{\Phi}(k-1)\hat{\boldsymbol{x}}(k-1)_{(+)} \tag{8.20}$$

$$\tag{8.21}$$

The prediction step is then given as:

$$\boldsymbol{P}(k)_{(-)} = \boldsymbol{\Phi}(k-1)\boldsymbol{P}(k-1)_{+}\boldsymbol{\Phi}(k-1)^T + \boldsymbol{Q}(k-1) \tag{8.22}$$

$$\hat{\boldsymbol{x}}(k)_{(+)} = \hat{\boldsymbol{x}}(k)_{(-)} + \bar{\boldsymbol{K}}(k)[\boldsymbol{z}(k) - \boldsymbol{H}(k) \cdot \hat{\boldsymbol{x}}(k)_{(-)}] \tag{8.23}$$

$$\boldsymbol{P}(k)_{(+)} = [\boldsymbol{I} - \bar{\boldsymbol{K}}(k)\boldsymbol{H}(k)]\boldsymbol{P}(k)_{(-)} \tag{8.24}$$

$$\bar{\boldsymbol{K}}(k) = \boldsymbol{P}(k)_{(-)}\boldsymbol{H}(k)^T[\boldsymbol{H}(k)\boldsymbol{P}(k)_{(-)}\boldsymbol{H}(k)^T + \boldsymbol{R}(k)]^{-1} \tag{8.25}$$

Where

$\hat{\boldsymbol{x}}(k)_{(-)}$ = Predicted step of $\boldsymbol{x}(k)$
$\boldsymbol{P}(k)_{(-)}$ = Prediction of the covariance
$\hat{\boldsymbol{x}}(k)_{(+)}$= Estimate of the state
$\boldsymbol{P}(k)_{(+)}$ = Update of the covariance
$\bar{\boldsymbol{K}}(k)$ = Kalman gain with $\boldsymbol{\Lambda}$ multiplied onto it

8.2 Simulations of the KF

Through simulations, the covariance matrices have been determined, as a dynamic covariance matrix have produced some bad results. When the system is simulated with a $sin(\theta)$ (0.2 amplitude, 0.001 Hz) angle reference and a 1 m/s velocity reference using the MATLABscript found on the CD-rom for 5000 samples , the system produces the below figure 8.2 on the following page. All the figures below are zoomed in on the tip of the track, where the biggest error is expected as this is where the ship turns.

When the system is simulated it gives the following response for estimating the position.

Figure 8.2 on the next page depicts the output of the KF when the sampling rate of the GPS and the IMU are equal. However on the actual system - the GPS only samples at 1 Hz and the IMU samples at 10 Hz, the filter needs to be changed. The KF must account for this change, which can be done by setting the noise of the measurement to a high value, which in turn will reduce the Kalman gain $\bar{\boldsymbol{K}}(k)$ to zero. Another implementation would be to zero the gain manually. This produces the below figure 8.3 on page 47 as a function of the GPS being sampled only once every second, while the IMU is sampled at 10 Hz.

If the Kalman gain $\bar{\boldsymbol{K}}(k)$ is not zeroed out, the KF will use the same sample as input, this causes the filter to try and make the system converge towards the last known value. If this is simulated it produces the following plot shown on 8.4 on page 48:

Simulations of the filter

Below is 3 plots of the KF running with a square pulse input as the angle reference. Figure 8.5 on page 49 depicts the system running when the sensors

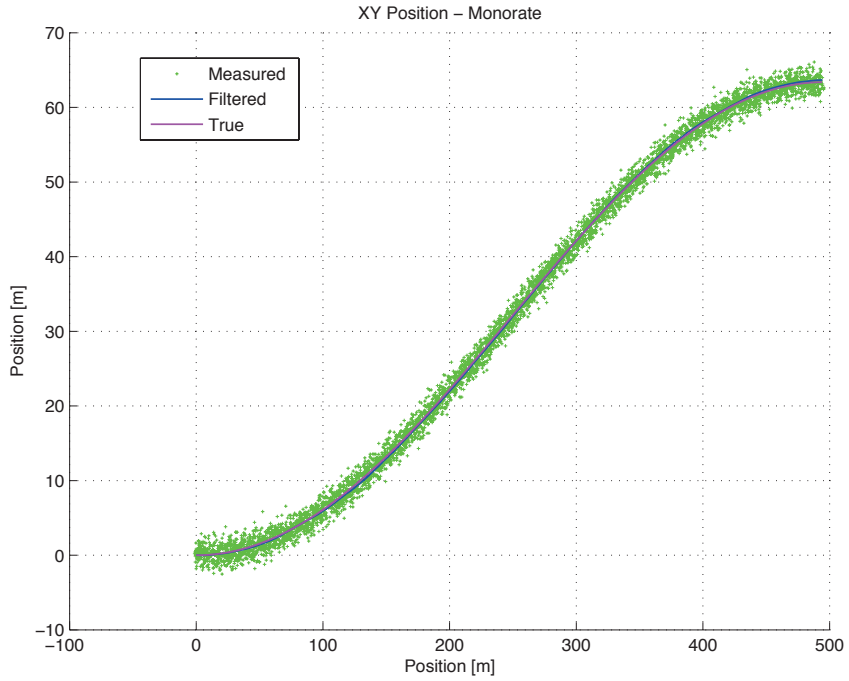


Figure 8.2: Simulation of the KF with reference signal given as: 1 m/s and angle as $sin(\theta)$, with 0.2 amplitude and 0.001 Hz frequency for 5000 samples. The two sensors samples at 10 Hz.

are running at the same sampling frequency. On the box on the plot the estimate clearly converges and follows the actual path nicely.

On 8.6 on page 49 the sampling rates are different and the Kalman gain $\boldsymbol{K}(k)$ is set to zero when no new GPS sample is present. This also tracks the true path very nicely, but varies a little more.

The last figure 8.7 on page 50 depicts the system where the input is only updated once every second. And this causes the system to diverge, but then pick up the lost sample again, thus making for a system that slowly will diverge.

8.3 Position estimation

During the development of the KF a lot or problems were discovered. One is to estimate the variances of the different measurement devies. The GPS delivers a position in latitude longitude format - which is converted into an x and y coordinate by rotating the entire system and shifting the local frame of the ship as a surface tangent to the earth. This will of course only be an estimate, but as the curvature of the earth is relatively small. The distance to the horizon can be estimated by $d \approx 3.57\sqrt{h}$ which on the ground, equals that

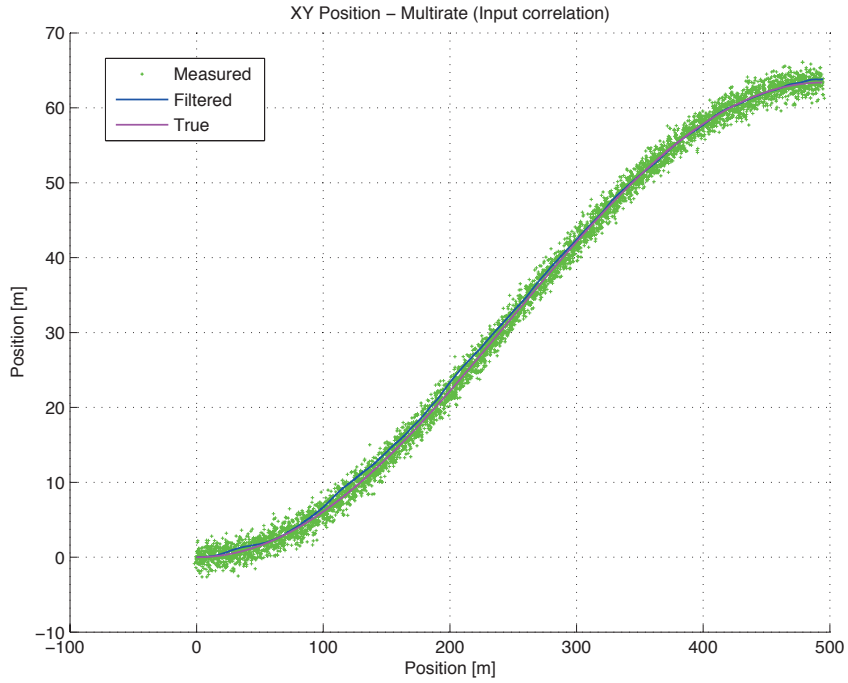


Figure 8.3: Simulation of the KF with reference signal given as: 1 m/s and angle as $sin(\theta)$, with 0.2 amplitude and 0.001 Hz frequency for 10000 samples. The system gains the measurements when no GPS amples are present with 0.

the distance to the horizon is approximately 3.57 kilometers. As the areas to be measured are defined by a local bounding box - this area will to be defined to be smaller than 3.5 kilometers.

This section will contain the different things we've considered during the miniproject in KFing, and will be used to give a better estimate of the actual position (from the measurements implemented on the ship).

Below is a description of the measurable inputs to the system. These are obtained from a GPS and a IMU mounted on the ship. Position and velocity from the GPS: rotated coordinates (from LatLon to Local frame) Linear acceleration from the IMU: accelerometer Angular acceleration from the IMU: gyrometer Angle from the IMU: magnetometer (compass)

8.4 Estimation of position on a lossy channel

As the controls are run on a remote platform, the KF should be able to work even though some sensor measurements are corrupted. To work around this, the Kalman gain should be set to zero of the readout is corrupted - so every

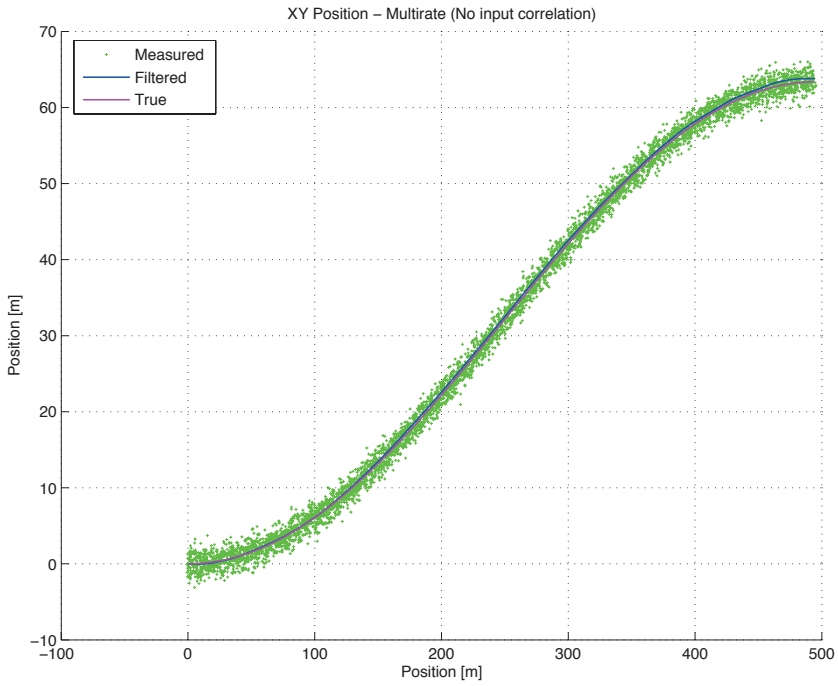


Figure 8.4: Simulation of the KF with reference signal given as: 1 m/s and angle as $sin(\theta)$, with 0.2 amplitude and 0.001 Hz frequency for 5000 samples. The system sets the current GPS measurement equal the last when no new sample is present.

time the receiver gets a packet where the checksom is invalid, it generates a vector mask that tells which rows of the Kalman gain should be zeroed.

$$\boldsymbol{\Lambda} = diag\{\lambda_x, \lambda_{\dot{x}}, \lambda_{\ddot{x}}, \lambda_{\lambda y}, \lambda_{\dot{y}}, \lambda_{\ddot{y}}, \lambda_\theta, \lambda_\omega, \lambda_\alpha\} \tag{8.26}$$

Where the individual λ's is given as:

$$\lambda = \begin{cases} 1 & \text{if measurement is valid} \\ 0 & \text{otherwise} \end{cases} \tag{8.27}$$

Thus augmenting the Kalman gain $\bar{\boldsymbol{K}}(k)$ equations to:

$$\bar{\boldsymbol{K}}(k) = \langle \boldsymbol{P}(k)_{(-)} \boldsymbol{H}(k)^T [\boldsymbol{H}(k) \boldsymbol{P}(k)_{(-)} \boldsymbol{H}(k)^T + \boldsymbol{R}(k)]^{-1} \rangle \boldsymbol{\Lambda} \tag{8.28}$$

The 4 figures represent the error with a 10 percent loss, a 20 percent loss, a 30 percent loss and finally a 40 percent loss in data.

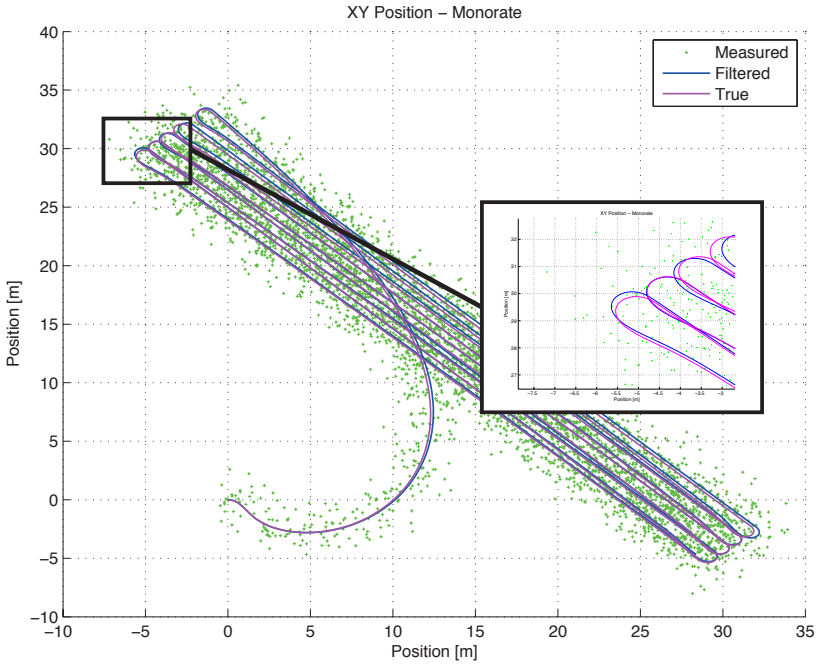


Figure 8.5: Simulation of the KF with the same sampling rate on the sensors

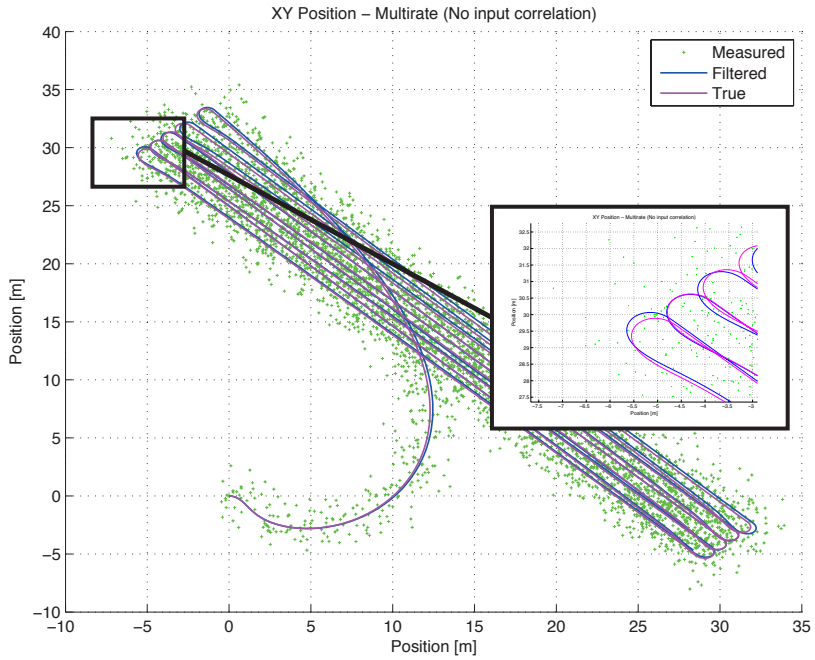


Figure 8.6: Simulation of the KF with the IMU running at 10 Hz and the GPS at 1 Hz

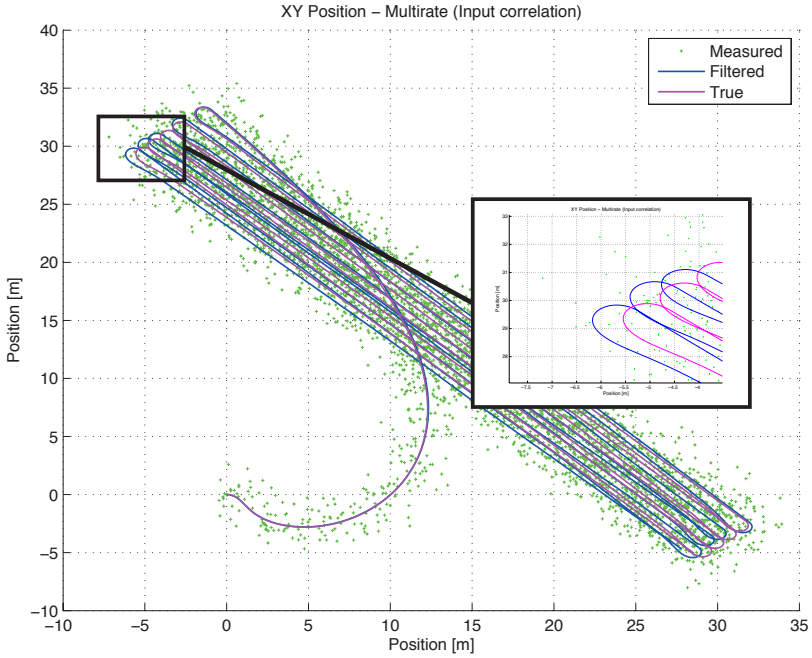


Figure 8.7: Simulation of the KF with the IMU running at 10 Hz and the GPS at 1 Hz - but converging towards the old sample

8.5 Implementation of the KF

The KF is implemented in Python and is run on the HLI. On the implementation it self, the KF is fed the control signals measured. A depiction of the actual implementation is seen on figure 8.8.

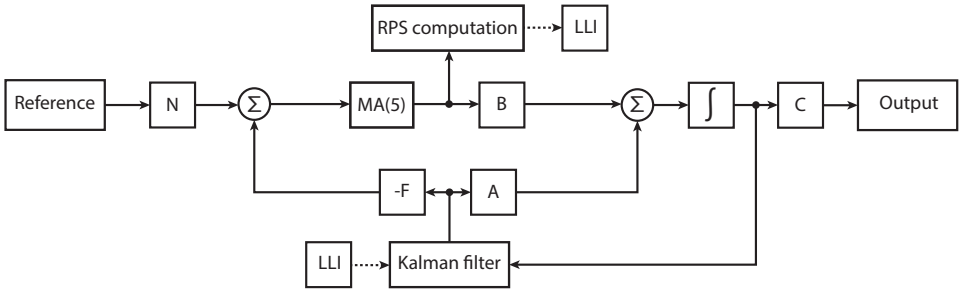


Figure 8.8: Depiction of the KF implementation. The KF is used to estimate the measurements more precisely.

8.6 Distributions of force and torque

The variance of the $\boldsymbol{F}$ and the $\boldsymbol{\tau}$ are estimated using normal distributions. The mean is estimated using a simulation of the ship motion, and then averaging out the amount of force applied to the ship at the given period.

$$\mu_{\boldsymbol{F}} = \frac{1}{n}\sum_{i=1}^{n} F_i \tag{8.29}$$

As the velocity of the ship is moving at the speed of 1 m/s the amount of force taken to acheive this speed is roughly 5.34 N. The variance of the force can however not be estimated that easily, as this can fluctuate from a negative value to a positive value. Therefore a trial-and-error approach have been used to estimate this, using the simulation in MATLABto to estimate this, when a number was reached it was reasoned wether that would be a good estimate. An estimate that proves good results have shown to be 55 N in either direction. As the ship doesn't need to move faster than 1 m/s it does not need to apply a big amount of force. Therefore, the distribution for the forward force can be given as:

$$\boldsymbol{F}_x \sim \mathcal{N}(5.34, 55) \tag{8.30}$$

Tests in the laboratory have shown that the ship produces around 20 N (using a manual-read out potentiometer) when both the input to the propellers are 150/500. And as the amount of force is related to the propeller revolutions squared, the ship is able to produce around this amount of thrust. The variance and mean of the torque are seen as a zero-mean process, as the ship mainly sails straight forward, but then the variance changes as the ship turns. The variance have proven good results with a variance of 20. This gives the following distribution for the torque of the vessel:

$$\boldsymbol{\tau}_\omega \sim \mathcal{N}(0, 20) \tag{8.31}$$

This concludes the state estimator worksheet. These can be used to estimate the covariance matrix of the ship.

9 Software

To implement the HLI functionality Python was chosen for its ease of writing as well as its extensive library of predefined functions. This enables the programmer to utilize highly complex functions through simple and On of these libraries is NumPy, which enables the user to define vectors and matrices and do calculations with these, in the same way Matlab does. This made it easy to port Matlab code from simulation to actual implementation. Another library is the MatPlotLib library. This eased the path planning implementation, as it was possible to plot the result of the proposed algorithms during testing. Further, another Python library allowed the script to easily access the OpenStreetMap API and retrieve the relevant data.

The LLI functionality was implemented in C as this is the primary embedded programming language with which the project group has experience.

9.1 System Description

The HLI consists of 3 main modules and an additional simulator module, which is not rqeuired for the embedded application.

The main module of the software is the Ship Class and its function definitions. The Ship Class itself represents an emphactor in the system. A Ship Object has individual dynamic model, controller and objective. Each Ship Object can plan its own waypoints, navigate trough them using their own sensor data and log informations.

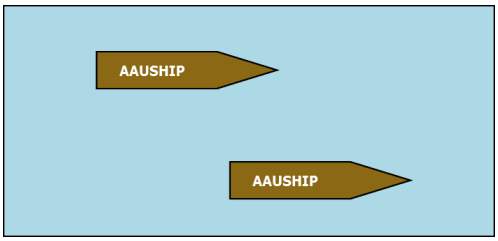


Figure 9.1: Actor Model

A Ship Object requires auxillary Classes and Functions. The two sidemodules of the system are the ObjectLibrary and FunctionLibrary. These Libraries store Classes and Functions, which are not directly connected to the physical behaviour of the ship, but provide an essential support for the Path Planning and Navigation control.

Each ship calculates its own *Waypoints*, depending on their individual capabilities and inputs. There is a possibility for the ships to communicate trough a wireless channel with each other directly or trough a nearby mothership/-

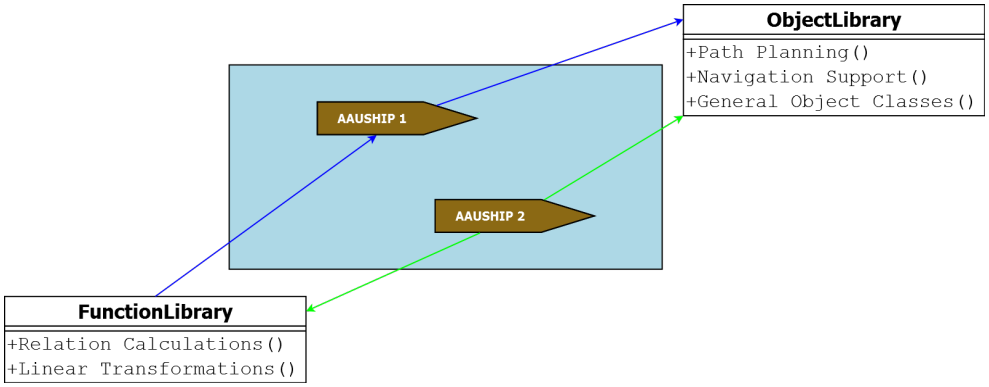


Figure 9.2: Actor Model

computer, in order to share position data (to avoid collisions), or distribute the Path Planning tasks for enhanced computation power.

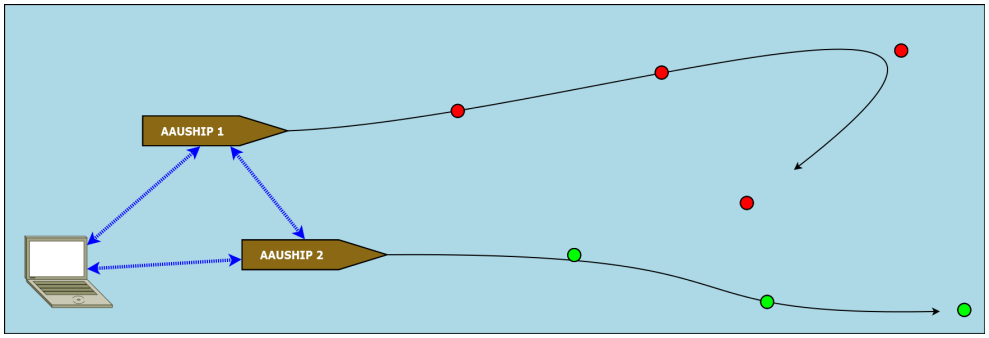


Figure 9.3: Actor Model

The additional simulator module is an extension of the *Ship Object*. In Simulation mode, to each *Ship* belongs a *Simulator*, simulating its interface communication and dynamic behaviour. The design of the simulator was led by two objectives:

- The *Simulation* and the *Embedded* application must run with the absolutely same *Ship Class Module*
- The simulator must predict the behaviour of the system well, but must be using a different model than the inner *State Estimator Kalman Filter* of the ship
- The simulator must be completely independent from the simulated system, except for the actuator values of the Ship.

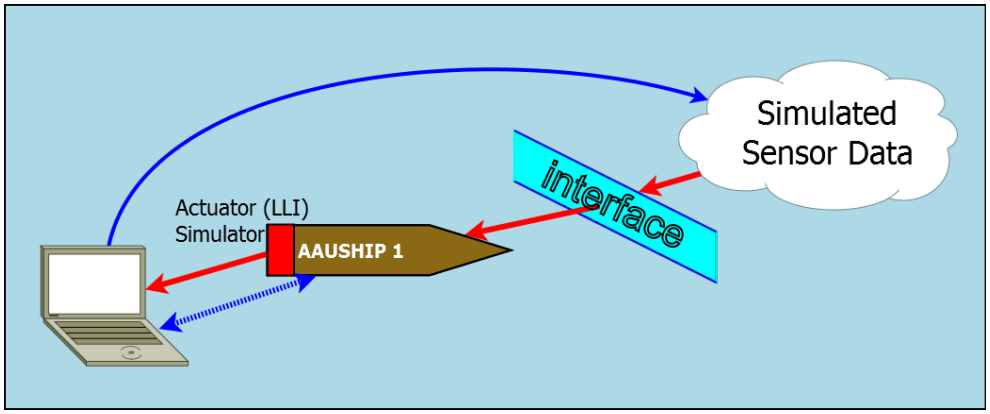


Figure 9.4: Simulation overview

Because of the predefined objectives above, the flowchart of the simulated and the actual task has only minor differences. See appendix 10 on page 89 on figure 9 on page 90 and 8 on page 89.

9.2 Course keeping control

The Course-keeping control is responsible for keeping the ship on-course during Autonomous navigation.

The general strategy is to have a state-space control algorithm in infinite loop as the main task. The control parameterization is based on the ship model, measurements and identification. The program controls the ship along the specified path. If there is no next sub-*Waypoint* or local path specified, the HLI calls the path planning methods for the next *Waypoint*, then the procedure starts again from the beginning. If there are no more *Waypoints*, the ship returns to the first *Waypoint*, or to the starting coordinates specified in a subfunction of the Ship class.

Using Sub-*Waypoints* instead of a full path line makes the navigation much easier. In open water there is significant sideways-motion caused by wind, ocean currents and waves, so staying perfectly true to a predefined path line is extremely difficult. Instead, by using a series of Sub-*Waypoints*, the navigation of the ship resembles to a series of Buoys guiding trough hazardous waters. The Ship approaches each Sub-*Waypoint* in a predefined order and turns to the next, when the ”buoy´´ is close enough. The characteristics of the navigation and course can be varied only by changing the definition of the Sub-*Waypoints*, setting optimal parameters for different locations and settings.

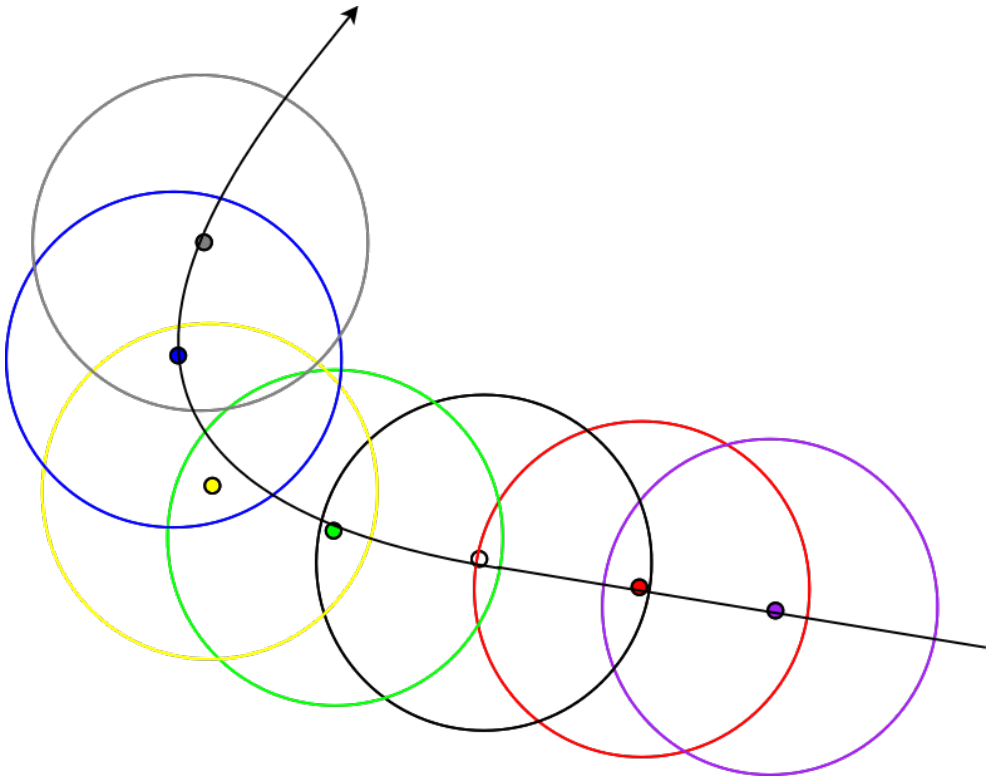


Figure 9.5: Concept of sub waypoint for navigation and the radis speceficaiton.

9.3 *Waypoint* Planning

The *Waypoint*-planning system provides the basic routing for the autonomous navigation system. The purpose of the *Waypoint*-planning is to set key coordinates that the ship must approach, either for strictly defined reasons or to keep the ship away from forbidden or dangerous areas (like the coast or an island)

There are two possibilities to set a collection of *Waypoints*: The operator can either manually set them, based on the GPS coordinates of the *Waypoints*, or by inputting a coastline data series is a specified data type. The coast input format is a series of perpendicular distances measured from a line parallel to the coast. The coast input can be generated based on freely available map data (OpenStreetMap). The automatic *Waypoint* planner divides the coast to smaller parts, based on the required oceanography definition. For each segment a minimum approach distance is calculated, and the *Waypoint* planner defines a set of *Waypoints* along the path in a snake way, or in any other predefined setting. The figure below shows a simulation of an oceanography task using automatic *Waypoint* planning. All values in meters. After the ship visited all of the *Waypoints*, it returns to the first *Waypoint*, or to a specified return

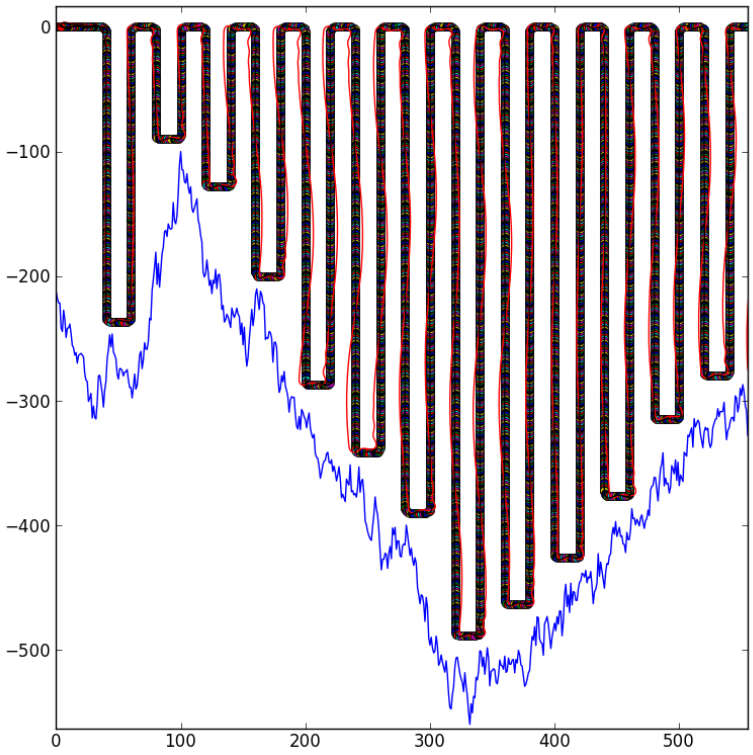


Figure 9.6: An example of the waypoint palnners output. The blue line simulates a random generated coastline whilst the border of the plot illustrates a bounding box for the area needed to be covered.

position.

Why are we using it? The need for a basic *Waypoint*-planner is essential, but optimizing it was not a high priority task. To test the prototype, a series of manually set *Waypoints* are adequate.

9.4 Local Planner

The Local planner is responsible for planning a segment of the path, in order to supply the controller with a series of Sub-*Waypoints*. The Local path should result in a set of points, which are lined up smoothly enough for the ship to sail through them with the reference speed. The path planning algorithm is based around the train-track transition problem originally solved in Talbot [1901], which divides the path into straight and turning parts and describes the transition between these using the normalized Fresnel integral, describing an Euler spiral, which allows the ship to maintain a linear acceleration through the turn, thus keeping the amount of jerk j as close to zero as possible. A series of key points are generated in the turn, which determine a path that meets

the above requirements Kiss [2011]. The two Fresnel integrals are given as in equation (9.1):

The path can be divided to a straight line and a turning sub-path. The combination of a straight path and a turn is a Path Segment. The local planner calculates a Path Segment from the current position to the end of the turn at the next Turning *Waypoint* (Footnote: If three *Waypoints* are in a line, the angle at the second is π, therefore it is not a turning *Waypoint*), based on the *Waypoint* after the Turning *Waypoint* as well.

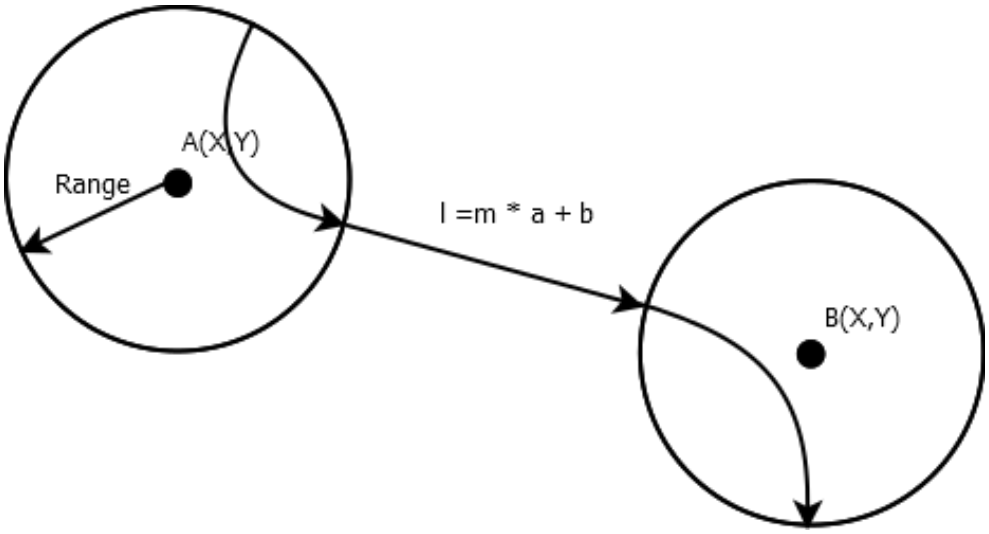


Figure 9.7: Illustration on how the Sub-*Waypoints* would approximate the *Waypoints* in corners, and become a straight line between *Waypoints* outside the range.

Calculating the smooth set of points for the turn: The initial idea was to use a pre-generated Euler-spiral or simple arc line, the program determines the required arc length and applies linear transformations to resize and rotate the path coordinates, thus creating a smooth path. This notion was dropped after the following considerations:

- Storing the full list of path coordinates is memory-consuming. Also, the path has either low resolution or the linear transformation would be CPU-heavy
- The path would be smooth but would not have optimal parameters ...

The new theory is originated from: Komlósi István: Mobilis robotok autonóm navigációja mozgó akadályok elkerülésével (English version: István Komlósi and Bálint Kiss: Motion planning for multiple mobile robots using time-scaling). The concept is to determine the maximum possible path curvature that the robot can handle. This is based on the Sigma and Kappa values of the Ship, where sigma is a function of the maximum speed of Torque

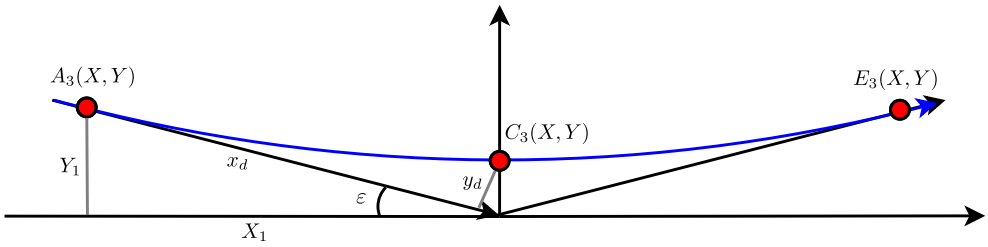


Figure 9.8: Path planning when $\varepsilon < \varepsilon_{max}$, the path is composed by two identical but mirrored Euler-spirals. 3 key points are generated, denoted A_3, C_3 and E_3

change, and kappa is the maximum curvature of the path at a given speed. The local path is generated in a specific way that the robot will always turn with the maximum possible curvature at the current speed, thus staying the closest to the *Waypoint* without losing speed. There is a threshold turn-angle, which determines if the robot requires only two identical Euler-spiral paths to turn or an arc path that has the maximum curvature, with two Euler spiral paths leading in or out.

$$C_F(x) = \int_0^x \cos(t^2)dt, \quad S_F(x) = \int_0^x \sin(t^2)dt \tag{9.1}$$

The functions two normalized Fresnel integrals produce the geometrical shape of the Euler spiral. However, the body on the track can endure only a limited amount of centripetal acceleration, which is the function of the Speed(v) and Path Curvature(κ). Vehicles have limited acceleration of heading $\dot{\kappa}_{max} = \alpha_{max}$ as well, which results in a necessary scaling (σ) of the Euler spiral:

$$A_{C_{max}} = \frac{v^2}{r} = v^2 \cdot \kappa \quad , \quad \sigma = \frac{\alpha_{max}}{v_{max}^2} \tag{9.2}$$

A threshold angle of turn (ε_{max})can be determined based on the vehicle parameters above:

$$\varepsilon_{max} = \frac{\kappa_{max}^2}{2\sigma} \tag{9.3}$$

In order to create the path the algorithm calculates 5 or 3 (depending on the threshold) key points and fits a Hermite-poly onto them. From this point on the local path in the given range is determined by these points only, thus saving memory and CPU time, while calculating a better path.

If the angle of the turn is lower than the threshold, the turn can be completed in two similar Euler spiral stages.

The 3 key points are given by the coordinate sets:

$$A_3 = (-X_1, Y_1),\ C_3 = (0, \frac{y_d}{cos(\varepsilon)}),\ E_3 = (X_1, Y_1) \tag{9.4}$$

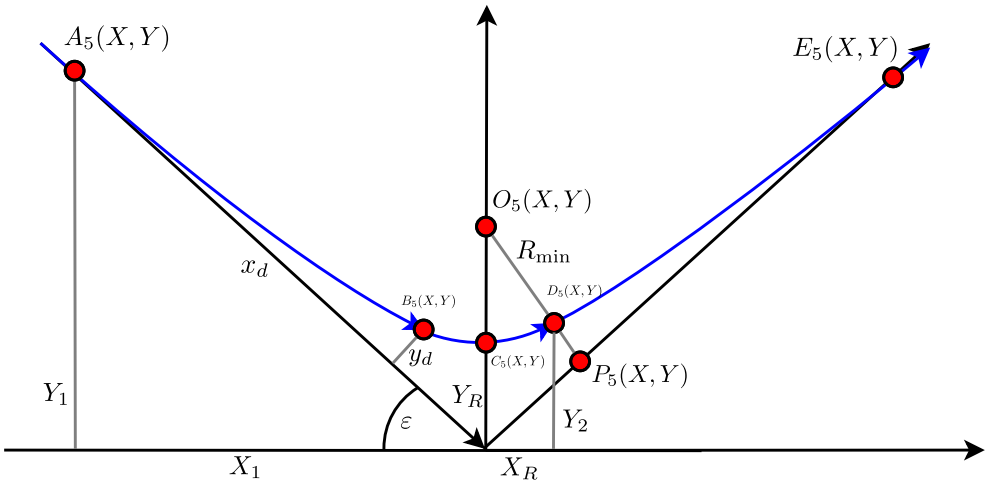


Figure 9.9: Path planning when $\varepsilon < \varepsilon_{max}$, the path is approximated by a curve and 5 key points are generated, denoted A_5, B_5, C_5, D_5 and E_5

From where the individual coordinates can be computed by simple trigonometric equations, if the angle of the turn ε is known. x_d and y_d are the length of the scaled Euler-spiral, when the turn of the spiral equals to ε, where x_d and y_d represents the (x, y) coordinate pair of the two normalized Fresnel integral functions with the same parameter t.

$$\varepsilon = \frac{\mathrm{dx}\ F(t)}{\mathrm{dy}} \quad \rightarrow \quad [x_d, y_d] = F^{-1}(\varepsilon)\sigma = \frac{\beta_m ax}{v^2} x_d = \frac{\pi}{\sigma} * C_F(\varepsilon) y_d = \frac{\pi}{\sigma} * S_F(\varepsilon) \tag{9.5}$$

Where C_F and S_F are the normalized Fresnel-integral functions.

$$X_1 = x_d * cos(\varepsilon) + y_d * sin(\varepsilon) Y_1 = X_1 * tan(\varepsilon) A = (-X_1, Y_1) C = (0, \frac{y_d}{cos(\varepsilon)}) E = (X_1, Y_1) \tag{9.6}$$

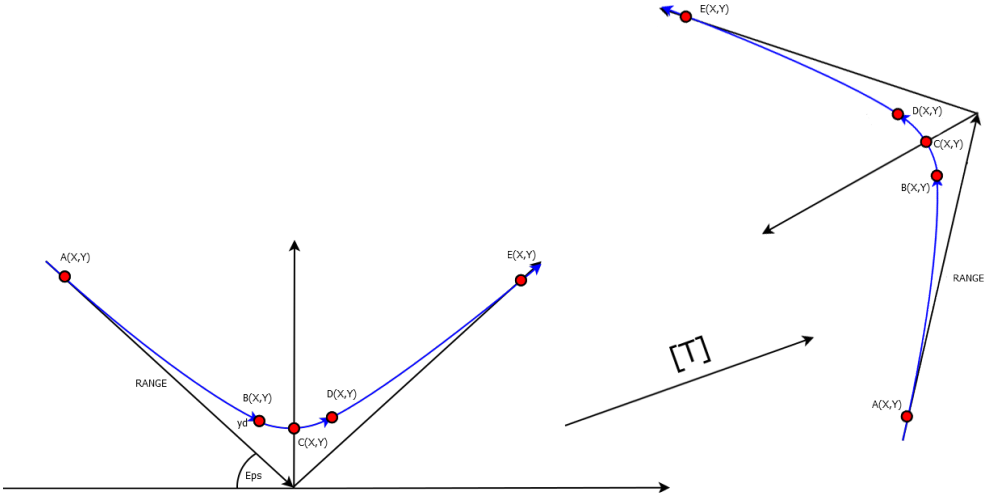
Once the angle grows larger than ε_{max} the system has to compute five key points, as the ship must transit onto a curve, and then back onto the Euler spiral ($\kappa_{Euler_{max}} = \kappa_{Arc}$) This adds the extra waypoints B_5 and D_5, the entry and exit points of the curve. The center of the circular path segment is O_5 and the radius is $R_{min} = \frac{1}{\kappa_{max}}$. The waypoints are depicted on figure (9.9), thus augmenting the waypoints to:

$$\sigma = \frac{\beta_m ax}{v^2} x_d = \frac{\pi}{\sigma} * C_F(\varepsilon) y_d = \frac{\pi}{\sigma} * S_F(\varepsilon) \tag{9.7}$$

Where C_F and S_F are the normalized Fresnel-integral functions.

$$X_R = R_{min} * sin(\varepsilon - \varepsilon_{max}) P_X = X_R + y_d * sin(\varepsilon) X_1 = P_X + x_d * cos(\varepsilon) Y_1 = X_1 * tan(\varepsilon) P_y = Y_1 - \quad (9.8)$$

The resulting path in both cases can not be described explicitly, therefore a Hermite-polynom is fit to the key points. In addition, a predefined number of uniformly distributed sub-waypoints are generated on the path, based on the describing polynom. The resulting sub-waypoints are rotated and moved to their correct position in the Local-Frame, thus concluding most of the work of the local-planner. The paths between turns are straight lines, populated with the same preset density of sub-waypoints.



The considerations behind this path planning method were based on the following conditions:

- The ship must be able to output the next *Waypoint* quickly, therefore calculating the whole path line in a single batch was to be avoided
- The *Waypoints* of the ship are subject to possible changes. Re-calculating the whole path every time a *Waypoint* is changed is very consuming
- The Ship is subject to unpredictable outside forces. Every path-segment is to be computed to be optimal, based on the actual, not the ideal prepared position
- The local planning should be as efficient as possible. Planning every Sub-*Waypoint* individually is a lot less effective than planning them in a batch ...

The Aurea mediocritas[1] lies in dividing the path to segments, and planning the Sub-*Waypoints* of each segment in a batch.

[1] Golden mean

Random notes about LLI concepts 10

φ := change of orientation /2

if $\varphi > \frac{\kappa^2}{2\sigma}$: Euler -> line -> Euler

if $\varphi \leq \frac{\kappa^2}{2\sigma}$: Euler -> Euler

If $\varphi_{max} \geq \frac{\kappa^2_{max}}{2\sigma}$

$$x_d = \sqrt{\frac{\pi}{\sigma}}\, C_F\left(\sqrt{\frac{2}{\pi}}\,\varphi_{max}\right)$$

$$y_d = \sqrt{\frac{\pi}{\sigma}}\, S_F\left(\sqrt{\frac{2}{\pi}}\,\varphi_{max}\right)$$

$$x_R = R_{min}\, \sin(\varphi - \varphi_{max})$$

$$y_R = 2R_{max}\, \sin^2\left(\frac{\varphi - \varphi_{max}}{2}\right)$$

$$x_1 = \cos(\varphi - \varphi_{max})\left(x_d + y_d \tan(\varphi - \varphi_{max})\right)$$

$$y_1 = x_1 \tan(\varphi)$$

$A = (-x_1, y_1)$

~~$B = (x_R, y_R)$~~

$B = (-x_2, y_2)$

$x_2 = x_R$

$y_2 = \sin(\varphi)\left(x_d + y_d \tan(\varphi)\right) - \frac{y_d}{\cos(\varphi)}$

$C = (0, y_2 - y_R)$

$D = (x_2, y_2)$

$E = (x_1, y_1)$

Once the calculation of the Point position is finished, it is possible to fit a Hermite-patinon on them.

~~If~~ $\varphi \leq \frac{\kappa^2_{max}}{2\sigma}$

$$x_d = \sqrt{\frac{\pi}{\sigma}}\, C_F\left(\sqrt{\frac{2}{\pi}}\,\varphi\right)$$

$$y_d = \sqrt{\frac{\pi}{\sigma}}\, S_F\left(\sqrt{\frac{2}{\pi}}\,\varphi\right)$$

$B = \left(0, \frac{y_d}{\cos\varphi}\right) = C = D$

$A = (-x_1, y_1)$

$x_1 = y_d \cdot \sin(\varphi) + x_d \cdot \cos(\varphi)$

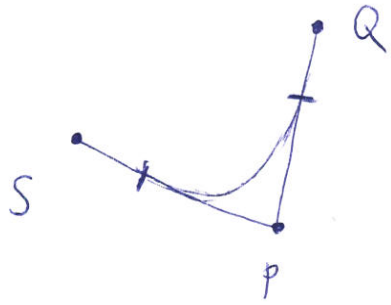
$y_1 = y_d \cdot \cos\varphi + x_d \sin(\varphi)$

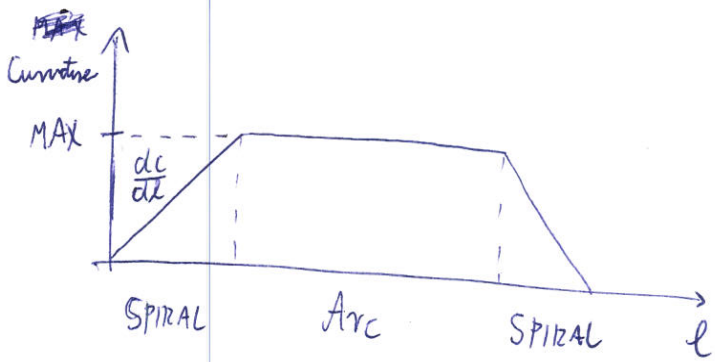
$E = (x_1, y_1)$

LOCAL PLANNER

(1) Generate Shape

Clotoide – Arc – Clotoide





$\frac{dc}{dl}$:= MAX Possible Curvature change

(e.g. Turning speed of steering wheel)

2D kinematic model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{\kappa} \end{bmatrix} = \begin{bmatrix} \cos\psi \\ \sin\psi \\ \kappa \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \beta$$

coordinates: x, y

orientation: ψ

Curvature: κ

$\frac{\text{d Curvature}}{dt}$: β

Max speed: v_{max}

Min radius: $R_{min} = \frac{1}{\kappa_{max}}$

$\frac{d\kappa}{dt} = \beta$

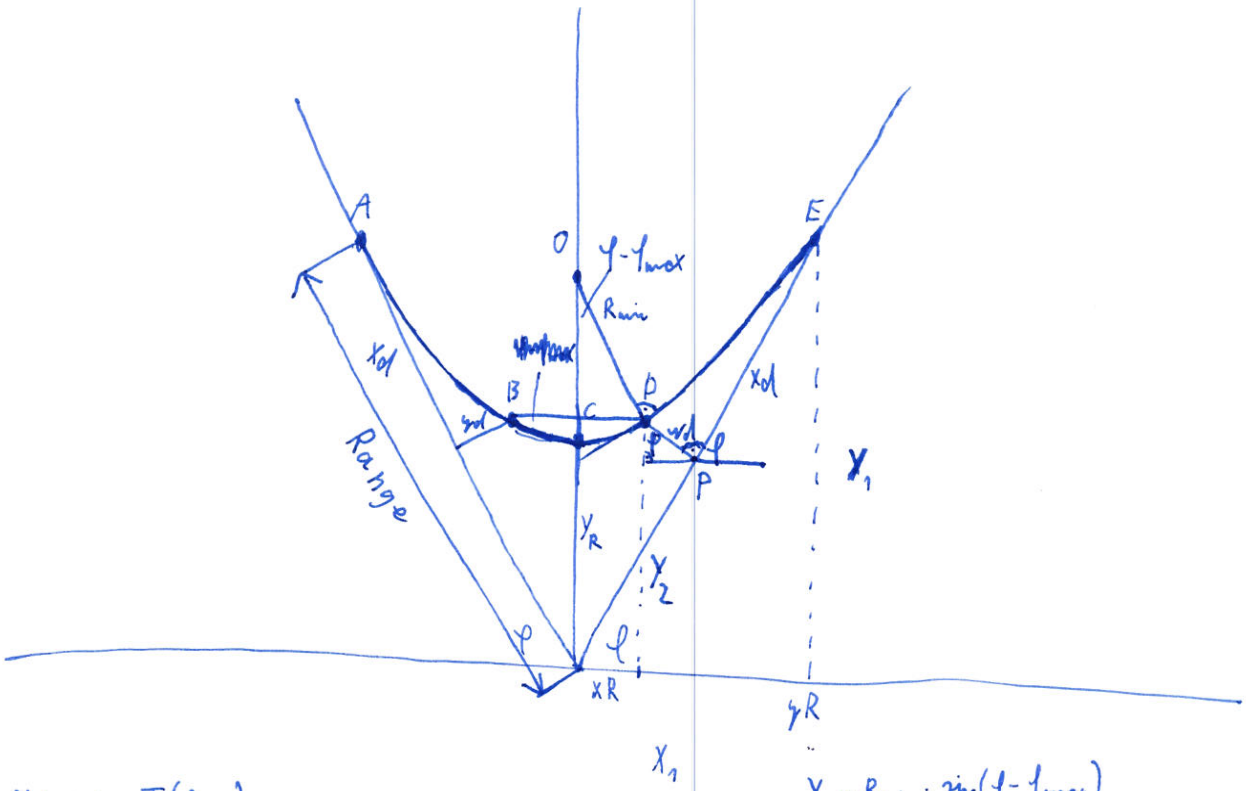
During a turn, the following conditions must be met:

$\kappa \leq \kappa_{max}$

$\beta \leq \beta_{max}$

$R_x(t)$ is the autocorrelation function of a WSS random process $X(t)$ if, and only if, $R_x(\tau)$ is a positive semi-definite function

LOCAL PLANNER



$x_d, y_d = F(\varphi_{max})$

$x_d = \sqrt{\frac{\pi}{\sigma}} C_F(\varphi_{max})$

$y_d = \sqrt{\frac{\pi}{\sigma}} S_F(\varphi_{max})$

$X_R = R_{min} \cdot \sin(\varphi - \varphi_{max})$

$Y_R = 2 R_{min} \cdot \sin^2\left(\frac{\varphi - \varphi_{max}}{2}\right)$

$X_1 = \cos(\varphi - \varphi_{max})(x_d + y_d \tan(\varphi - \varphi_{max}))$

$Y_1 = X_1 \tan(\varphi)$

$X_2 = X_R$

$Y_2 = \sin(\varphi) \cdot (x_d + y_d \tan(\varphi)) - \frac{y_d}{\cos\varphi}$

$X_R = R_{min} \cdot \sin(\varphi - \varphi_{max})$

$P_x = X_R + y_d \cdot \sin(\varphi)$

$X_1 = P_x + x_d \cdot \cos(\varphi)$

$Y_1 = X_1 \cdot \tan(\varphi)$

$P_y = Y_1 - x_d \sin(\varphi)$

$Y_2 = P_y + y_d \cdot \cos(\varphi)$

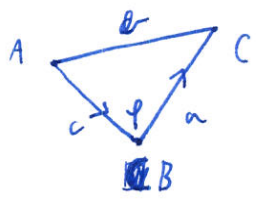
$O_y = R_{min} \cdot \cos(\varphi - \varphi_{max}) + Y_2$

$R_y = O_y - R_{min}$

$X_2 = X_R$

RANDOM REFERENCES

#1: Calculate angles:



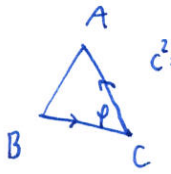
Law of cosines:

$$b^2 = a^2 + c^2 - 2ac\cos(\varphi)$$

$$\varphi = \arccos\frac{a^2 + c^2 - b^2}{2ac}$$

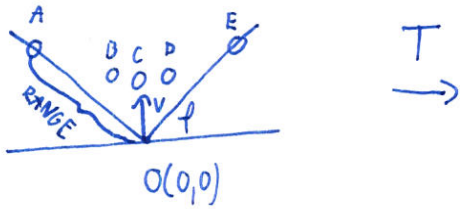
$$\varphi = \arccos\left[\frac{(B_x - C_x)^2 + (B_y - C_y)^2 + (A_x - B_x)^2 + (A_y - B_y)^2 - (A_x - C_x)^2 - (A_y - C_y)^2}{2\sqrt{(B_x - C_x)^2 + (B_y - C_y)^2}\cdot\sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}}\right]$$

Coded with the following rotation:

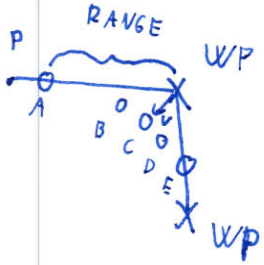


$$c^2 = a^2 + b^2 - 2ab\cos(\varphi)$$

#2: Linear transform of Poly-Point coordinates



T →



$$\underline{R_a} = \overrightarrow{OA} = (A_x, A_y)$$

$$\underline{R_e} = \overrightarrow{OE} = (E_x, E_y)$$

$$\underline{v} = \underline{R_a} + \underline{R_e} = (A_x + E_x, A_y + E_y)$$

$$\alpha = \operatorname{atan}\left(\frac{A_y + E_y}{A_x + E_x}\right)$$

$$\left(\alpha = \frac{\pi}{2}\right)$$

$$\underline{R_P} = \overrightarrow{WP\,P} = (P_x - WP_x, P_y - WP_y)$$

$$\underline{R_{WPP}} = \overrightarrow{WP\,WPP} = (WPP_x - WP_x, WPP_y - WP_y)$$

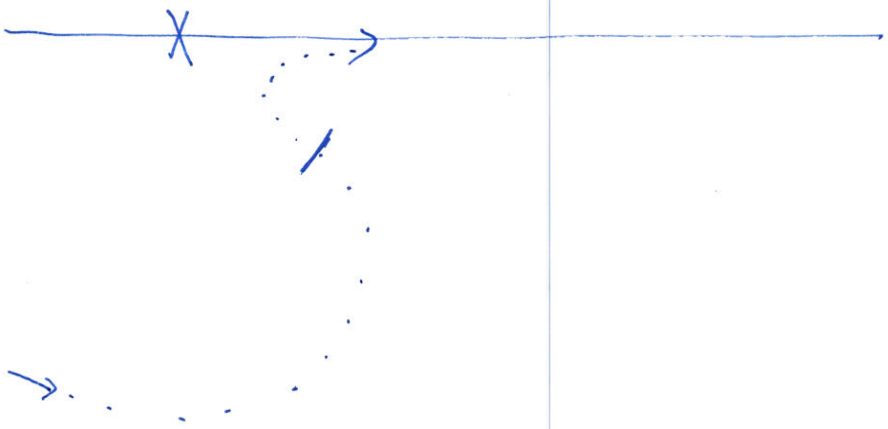
$$\underline{v} = \underline{R_P} + \underline{R_{WPP}} = (P_x + WPP_x - 2WP_x, P_y + WPP_y - 2WP_y)$$

$$\sigma = \operatorname{atan}\left(\frac{P_y + WPP_y - 2WP_y}{P_x + WPP_x - 2WP_x}\right)$$

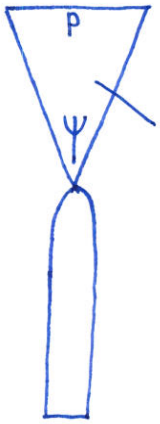
$$T = \left[rot(\sigma - \alpha) + offset(WP - 0)\right]$$

Deviation from Path:

No points in the scope ⟶ Need to generate some



NAVIGATION



Control - Scope

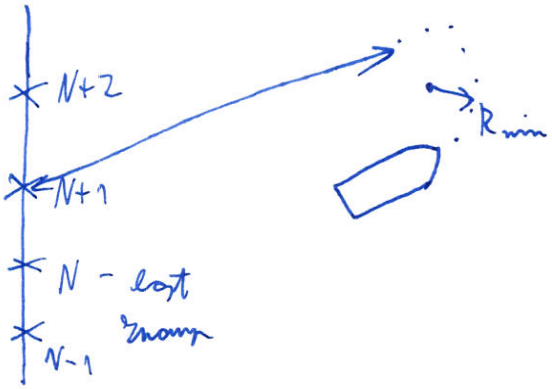
OUTDATED

- The ship always follows the SubW/P with the highest Order in the scope.

- If there is no SubW/P in the scope, the ship has left the Path and must be guided back, therefore we generate new SubW/Ps into the scope.

Derivation:

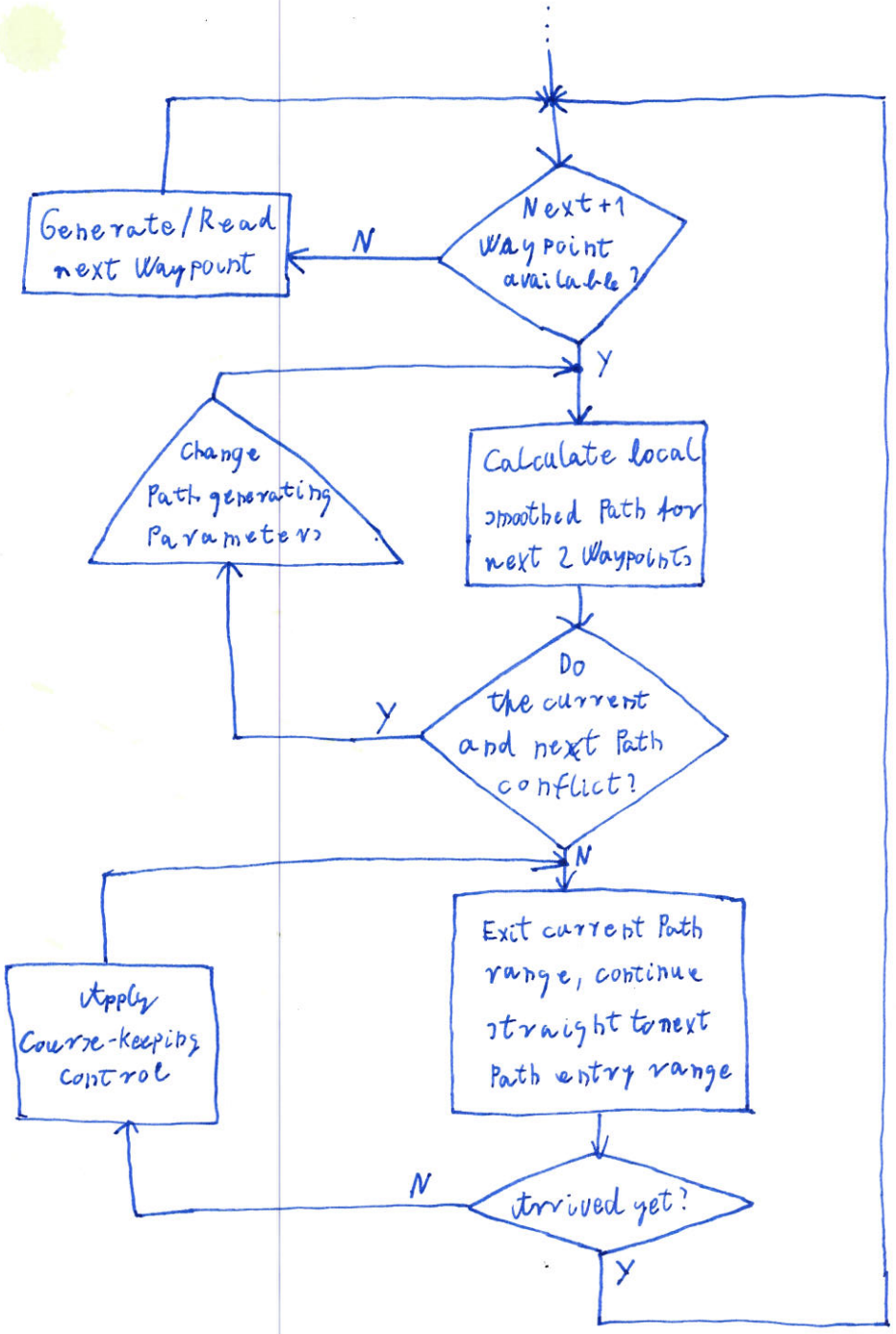
- The last seen highest SubW/P order is known.



- The ship is set to a circular Path until it faces N+1 OR it SPOTS a SubW/P ≥ N+1

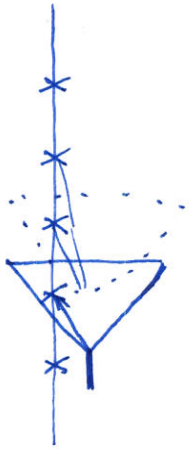
- Then a dummy W/P is generated towards N+i, and a local Path is generated with N+1 as a turn point

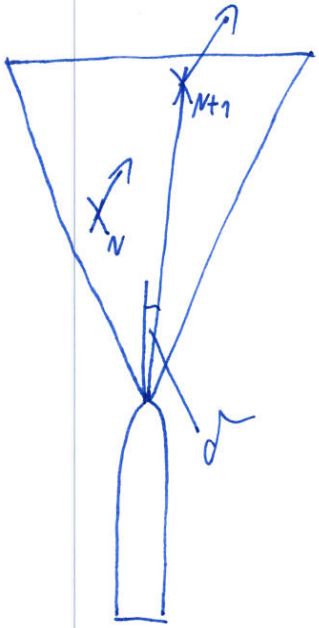
NAVIGATION

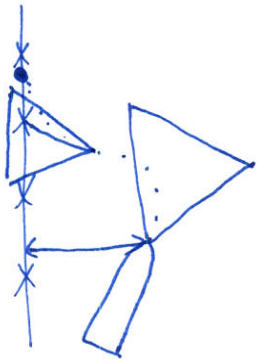


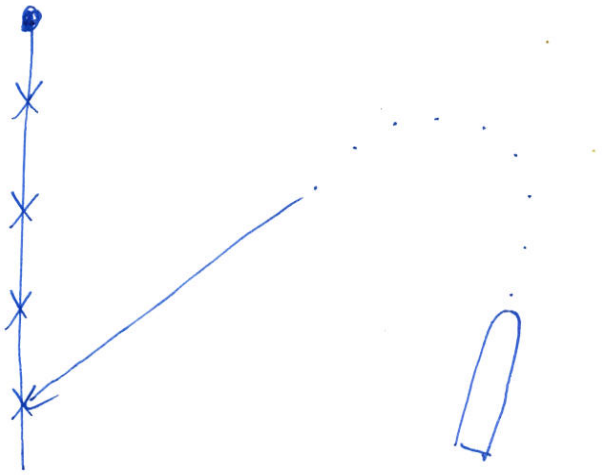
OUTDATED

Navigating along Path:



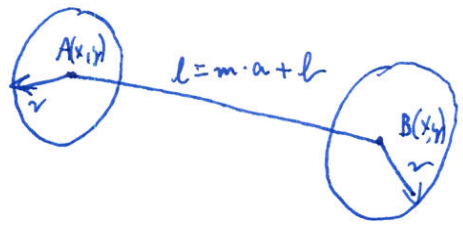




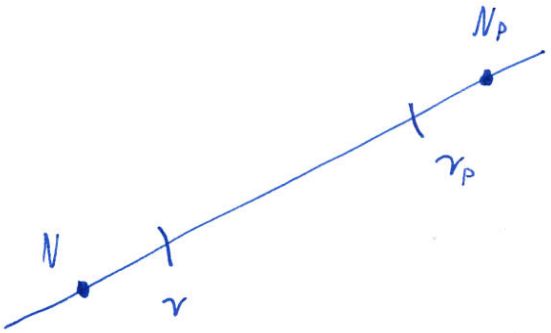


OUTDATED

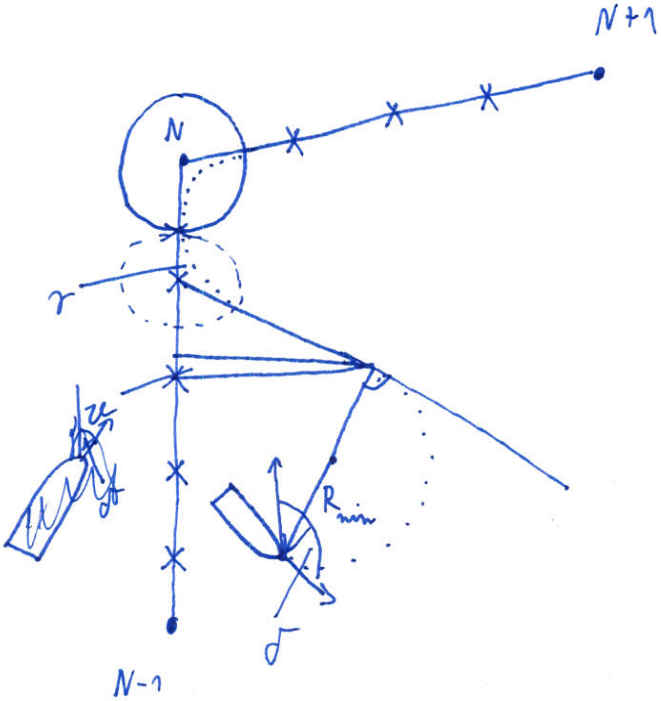
#3: Route between 2 Waypoints:

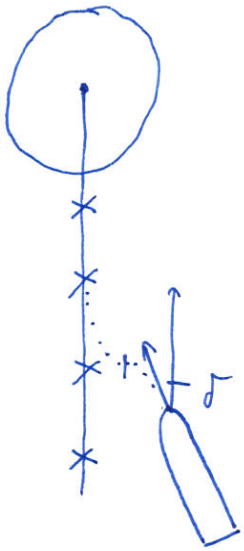


Sub - WayPoints:



#4: Deviation from Path ($\sigma < \frac{\pi}{2}$)





-1: Circular Path towards Next WP(N)

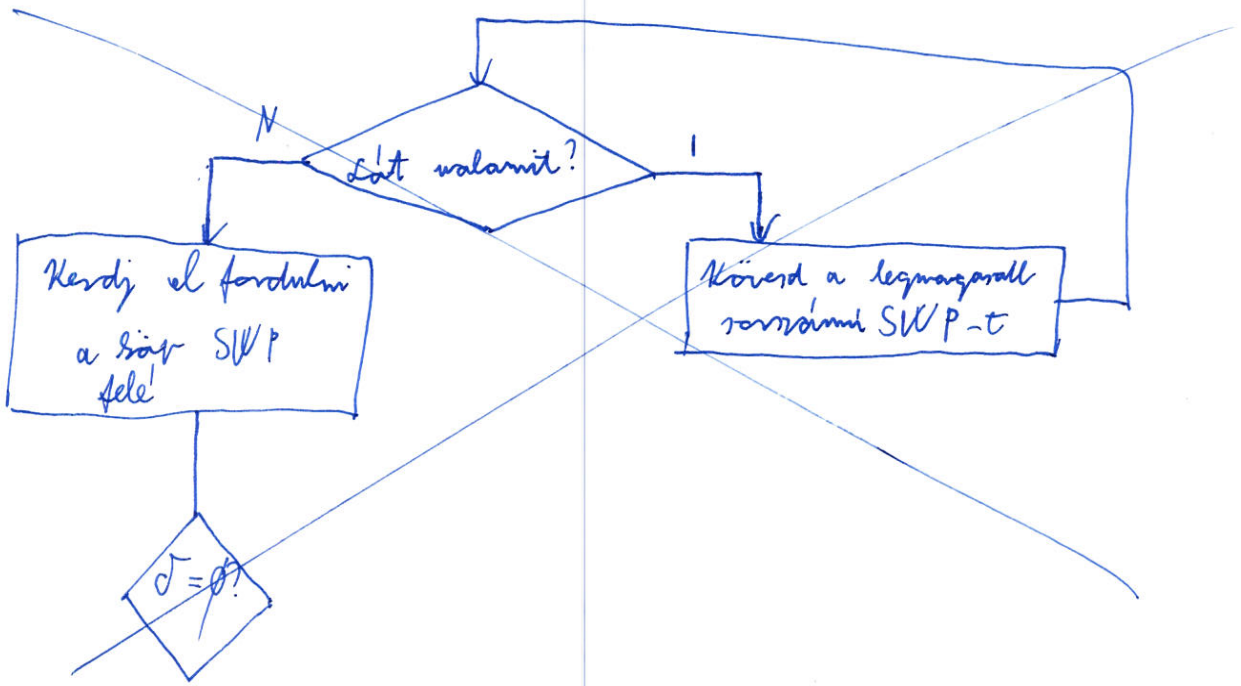
-2: When the ship faces the last sub-coordinate that is toward the next WP ($\sigma \geq \frac{\pi}{2}$ and $\alpha < \frac{\pi}{2}$) the circular path continues straight

-3: The selected sub-WP is treated as a regular WP. The WP+1 that belongs to the Path is one of the following:

- Sub-WP(n+1)
- N (if Sub-WP(n) was the last Sub-WP)
- Sub-WP+1(1) if the range of the return-Path and the range of the normal Path conflicts

-4: Continue on toward either N or N+1 depending on the solution of (-3.)

Navigáció


Lát valamit?
N
I
Kezdj el fordulni a sárga SVP felé
Követed a legnagyobb területű SVP-t
T = ∅?

Proportionality of the Euler - spiral:

$$\sigma = \frac{p_{max}}{v_{max}^2}$$

Describing equations:

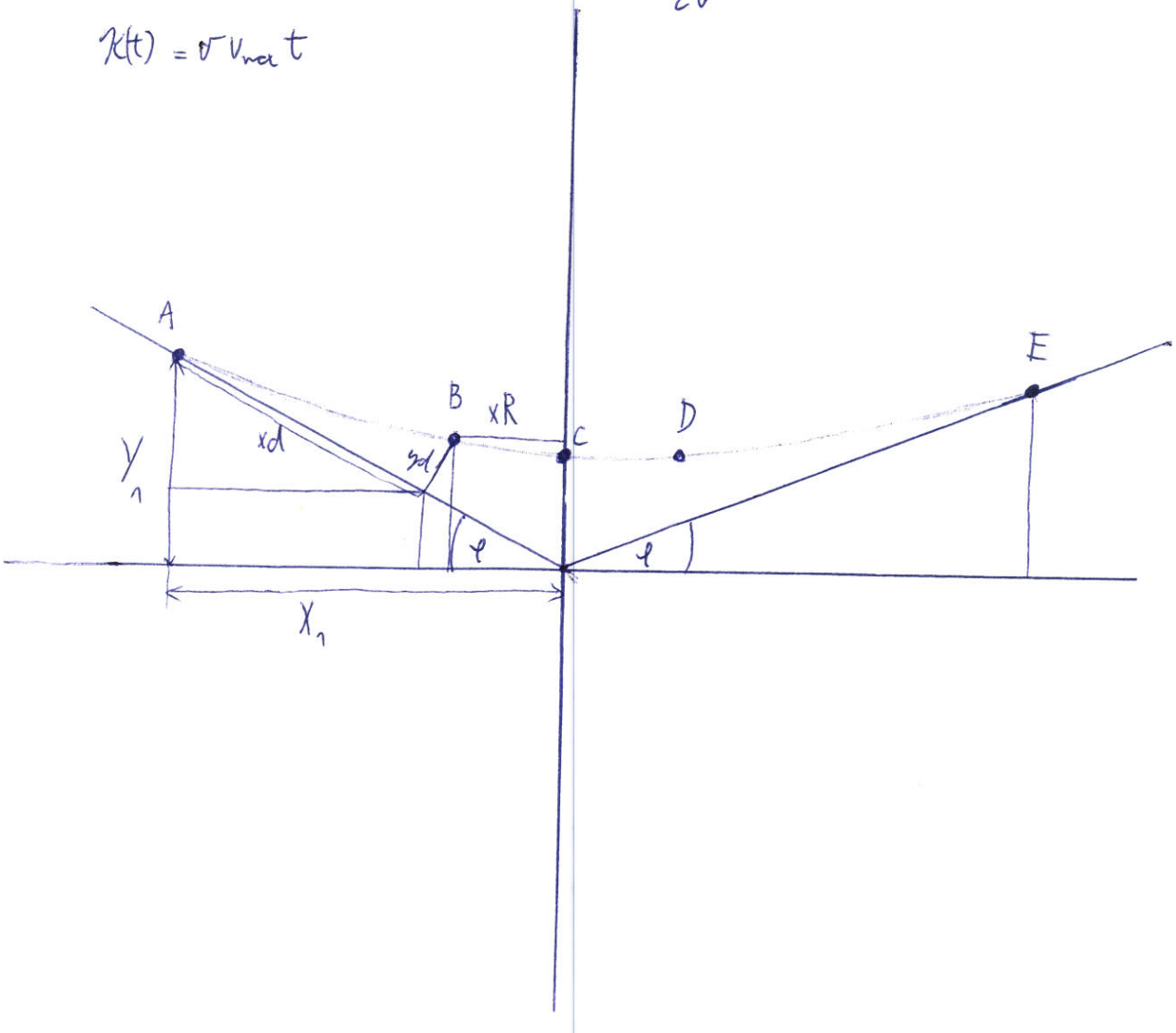
if $v_{max}\, t = \frac{\kappa}{\sigma}$ ↓

$$x(t) = \sqrt{\frac{\pi}{\sigma}}\, C_F\left(\sqrt{\frac{\sigma}{\pi}}\, v_{max}\, t\right) = \sqrt{\frac{\pi}{\sigma}}\, C_F\left(\frac{\kappa}{\sqrt{\pi\sigma}}\right)$$

$$y(t) = \sqrt{\frac{\pi}{\sigma}}\, S_F\left(\sqrt{\frac{\sigma}{\pi}}\, v_{max}\, t\right) = \sqrt{\frac{\pi}{\sigma}}\, S_F\left(\frac{\kappa}{\sqrt{\pi\sigma}}\right)$$

$$\psi(t) = \frac{1}{2} v_{max}^2\, t^2 \qquad \psi(t) = \frac{\kappa^2}{2\sigma}$$

$$\kappa(t) = \sigma\, v_{max}\, t$$



Appendices

Measurement Journal: IMU Variance

The intention of this journal is to measure the variance of the output of the ADIS16405 IMU.

Equipment

- ADIS16405
- Macbook Pro 2010 15 inch 2.53 GHz Intel Core i5, 4GB Ram
- APC220 Wireless communication Module

Method

A python script was implemented which logged the data which was accepted according to the protcol specifications chapter 4. The board on which the IMU is mounted is reading the data with a set delay between reads, and then bursting the data as soon as it has read. The data is timestamped when it is received by the python script. The script will keep running, receiving data, until it is cancelled. As it is very easy to gather samples, a large amount of samples has been gathered, by allowing the system to run overnight.

From an early trial run of the system it was discovered that the IMU would sporadically deliver measurements which were clearly wrong. Luckily all of the measurements were affected by this. Therefore it was possible to filter this by attaching the ADC of the IMU to ground, and then discarding the measurement when the ADC reads a high value.

The measurements are then input in Matlab and the variances of the measurements are estimated, using the 'var()' function.

Measurements

The measurements can be found in the file accdata.csv. The different values from the IMU are comma separated with each new measurement on a new line. The output follows the format: Voltage, X-,Y-,Z-Gyro, X-,Y-Z-Accelerometer,X-,Y-,Z-Magnetometer, Temperature, ADC.

Results

The resulting variance of the accelerometer measurements were found to be 5.05, 4.98, 5.96 μG for the X, Y and Z direction. For the Gyrometer the variances were found to be 2.28, 2.45 and 2.40 μdegrees per second.

Measurement Journal: GPS Variance

The intention of this journal is to measure the variance of the output of the UP501 GPS.

Equipment

- UP501 Fastrax GPS receiver
- HP laptop
- APC220 Wireless communication Module

Method

A python script was implemented which logged the data which was accepted according to the protcol specifications chapter 4. The board on which the IMU is mounted is reading the data with a set delay between reads, and then bursting the data as soon as it has read. The data is timestamped when it is received by the python script. The script will keep running, receiving data, until it is cancelled. The system was then set to run overnight.

It was expected to see the GPS wander for some time, but then to reach some form of steadystate where the measurements would primarily be affected by the measurement noise. This was clearly observed, as seen on figure 10

From an early trial run of the system it was discovered that the IMU would sporadically deliver measurements which were clearly wrong. Luckily all of the measurements were affected by this. Therefore it was possible to filter this by attaching the ADC of the IMU to ground, and then discarding the measurement when the ADC reads a high value.

The measurements are then input in Matlab, transformed into a local reference frame and the variances of the measurements are estimated, using the 'var()' function for the coordinate measurements. For the speed measurements, the unbiased sample variance formula as seen in equation (10). Since the sample base is large, the effect of $\frac{1}{n}$ compared to $\frac{1}{n-1}$ is negligible, but the later should still be used, as the unbiased estimation is desired.

$$s^2_{N-1} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2 \tag{1}$$

Measurements

The measurements of the position can be found in `gpsSDlog291112.log`, while the speed data can be found in `speedlog.log`. As the speed measurements

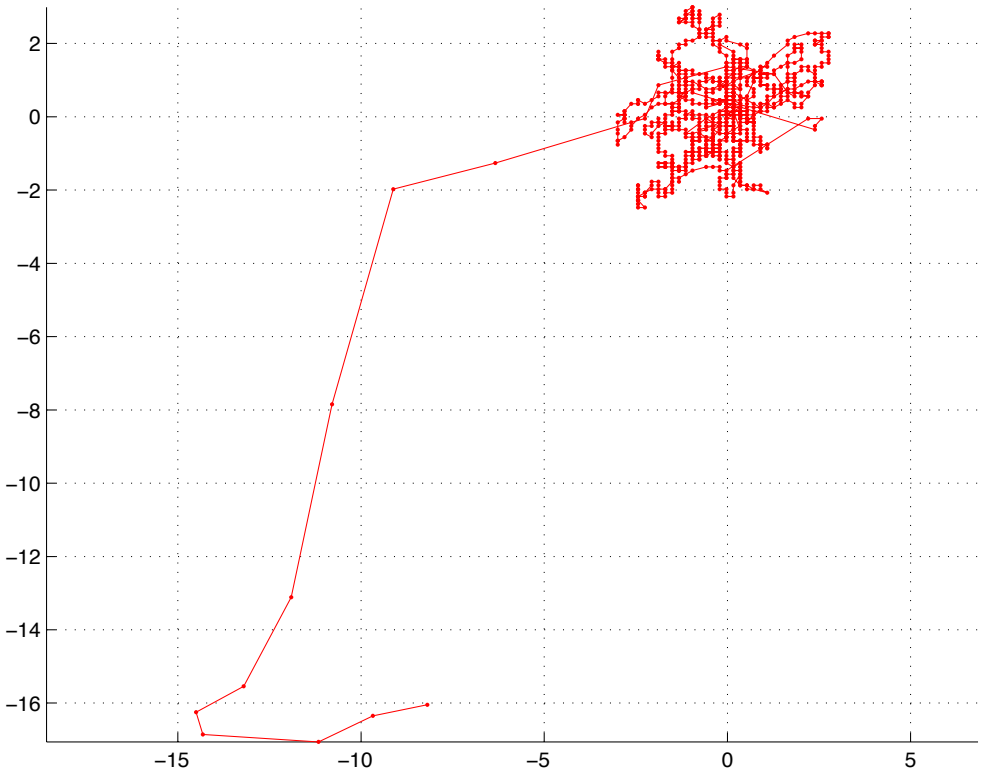


Figure 1: The GPS measurements, after being transformed into a local reference frame

are not present in every sample from the GPS, the two files contain a different amount of samples. While position log contain 25220 measurements, the speedlog only contain 1690. This is however deemed sufficient for the variance estimate.

Results

The resulting variance of the GPS measurements were found to be $0.0026\frac{\mathrm{m}}{\mathrm{s}}$ for the speed measurements, while the variances for the position measurements were found to be 0.979 m and 1.12 m, for the x and y position, respectively.

Maiden Voyage

After the boat had arrived and was properly equipped, everything was ready for a basic test with simple, manual control in order to observe its stability, its ability to take turns, as well as the speed the ship could navigate at. Because all pools or other forms of standing water were frozen at the time, this test had to be performed at the marina located here, on the corner of Skudehavnsvej and Vestre Fjordvej, in West Aalborg.

Figure 2: Manual control: GPS measurements with map

The boat was sealed with a lid and weighted with around 1.5 kg of lead in order to keep it as low in the water as possible, so that the propellers would not suck air in. Then a Python script was used which, when pressing "forward" or "back" would tell both motors to go faster of slower. The left arrow would increase the left propeller speed and would turn the right one down and vice versa. In this way, we were able to have basic control of the ship and test it. Space was used as an "emergency stop".

In pictures 2 and 3 on the following page the data obtained from the GPS sensor during this trip is plotted on top of an Open Street map of the marina in which the tests took place. They begin in the top left of the picture, where

Figure 3: Manual control: GPS measurements with map, zoom in

we took the boat out of the car and walked to the water, then all the dots are the GPS messages received while performing sailing tests.

Both Figure 4 on the next page and 5 on page 84 represent the ship in a continuous spiral. The turning radius will depend on the difference between the two motor speeds. It can be seen from the figures that this radius can be as small as 1 meter, which is a very good response. The red dot at (0, 0) represents the starting point.

The first wet test was also helpful in calibrating the control algorithm by noting that the motors are very powerful and that they should never be used at a very high throttle. Also, the motor controllers have some tolerance, meaning that the motors actually started turning forward at +10% and reversing at -10% of the available control signal. This hysteresis may induce jerks in the boat if not properly compensated because of overcompensating.

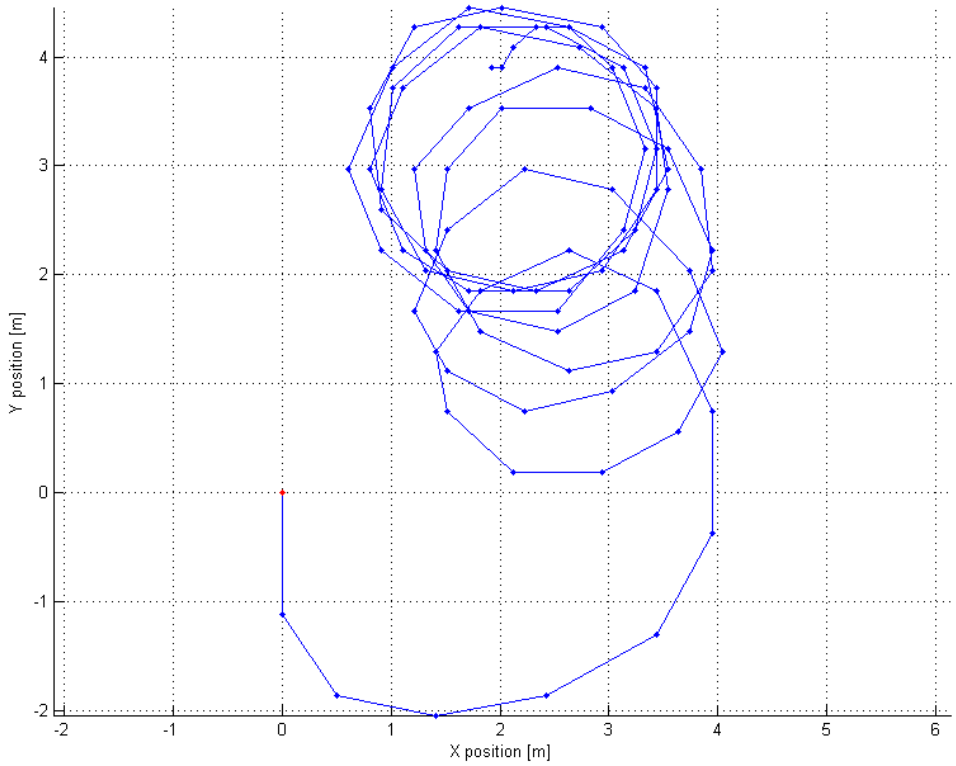


Figure 4: GPS measurements of ship turning as logged by the LLI during maiden voyage

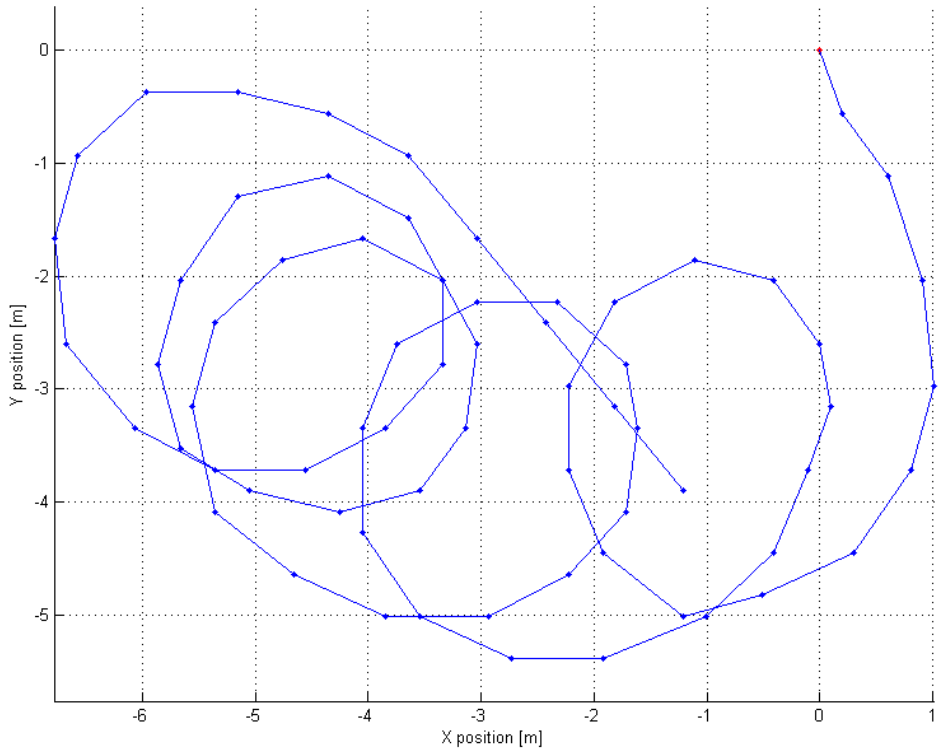


Figure 5: GPS measurements of ship turning as logged by the LLI during maiden voyage

Electronics schematics

This and the following pages is the schematics for the hardware developed.

On the following pages the schematics for the hardware that was made is posted, together with the layout and an image of the LLI

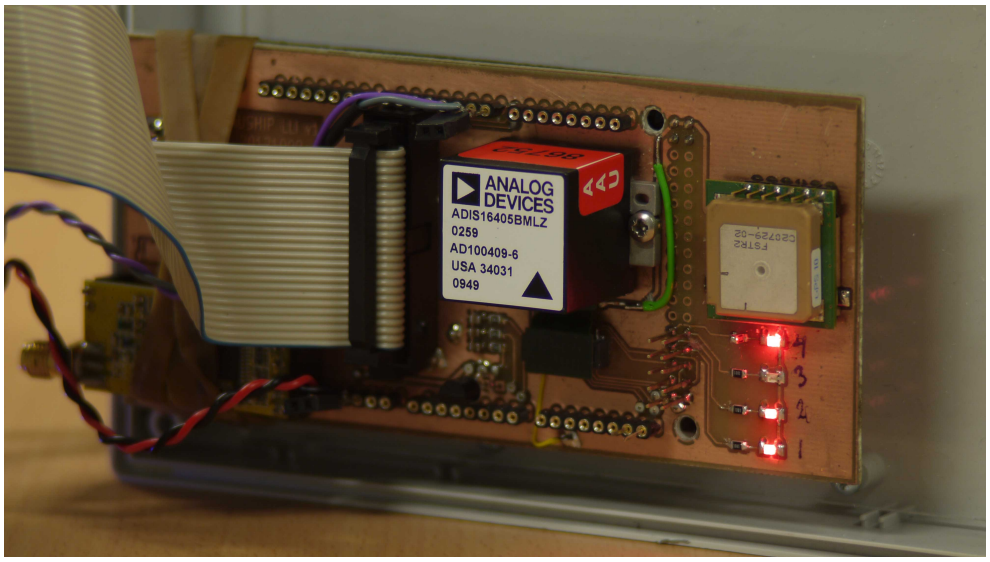
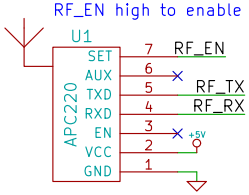


Figure 6: Assembled LLI

DCPWMx PWM pulse for 0–100% duty cycle for custom H–bridge like driver
DCDIRx direction control for the H–bridge like dirver
RCPWM RC standard 50 Hz 1–2ms signal

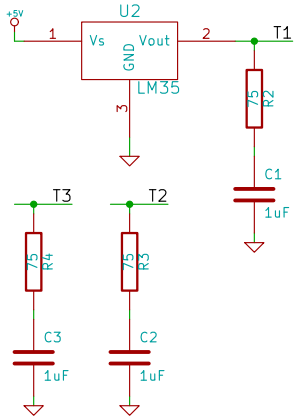
UHF Radio Module

RF_EN high to enable



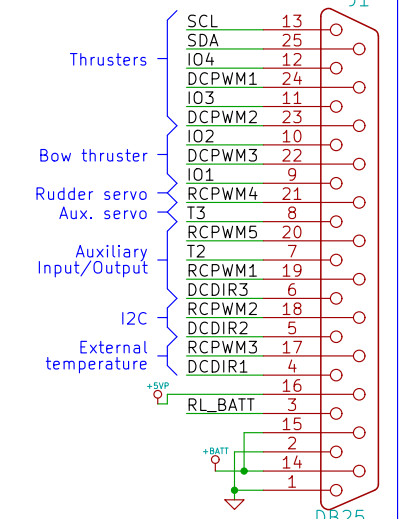
Auxiliary temperature sensors

10.0 mV/°C up to +150°C

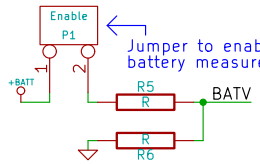


Interface connector

Note	Use	Pin mapping
RCPWMx is common RC pulse standard while the DCPWMx is common PWM with dutycycle 0–100%	Thrusters	SCL 13, SDA 25, IO4 12, DCPWM1 24, IO3 11, DCPWM2 23
	Bow thruster	IO2 10, DCPWM3 22
	Rudder servo	IO1 9, RCPWM4 21
	Aux. servo	T3 8
Auxiliary Input/Output pins, can also be used with the ADC. Comms with advanced survailence of power splitter.	Auxiliary Input/Output	T2 7, RCPWM1 19, DCDIR3 6
I2C not planned for any use, maybe for OpenServo devices?	I2C	RCPWM2 18, DCDIR2 5
External sensors to be mounted i.e. on the motors.	External temperature	RCPWM3 17, DCDIR1 4
RL_BATT and +5VP has to be activated by the microcontroller		+5VP 16, RL_BATT 3, 15, 2, 14, 1



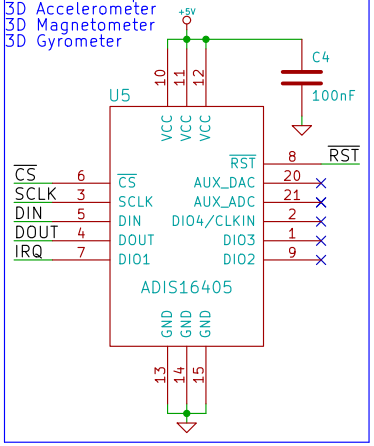
Battery Sensor



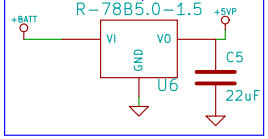
Fastrax UP501 default firmware configuration:
1. Port 0: NMEA 9600 baud
2. NMEA output: GGA, RMC, GSV, GSA (all 1 sec interval)
3. DGPS/SBAS: Disabled (Module supports WAAS/EGNOS)
4. Datum: WGS84

IMU

1 internal temperature sensor
3D Accelerometer
3D Magnetometer
3D Gyrometer

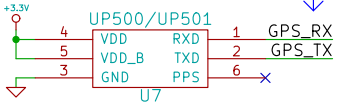


DC–DC converter 5V

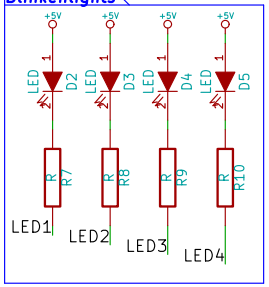


GPS receiver

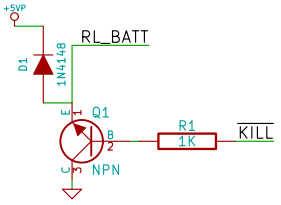
If pull up resistors is needed, enable them in the AVR



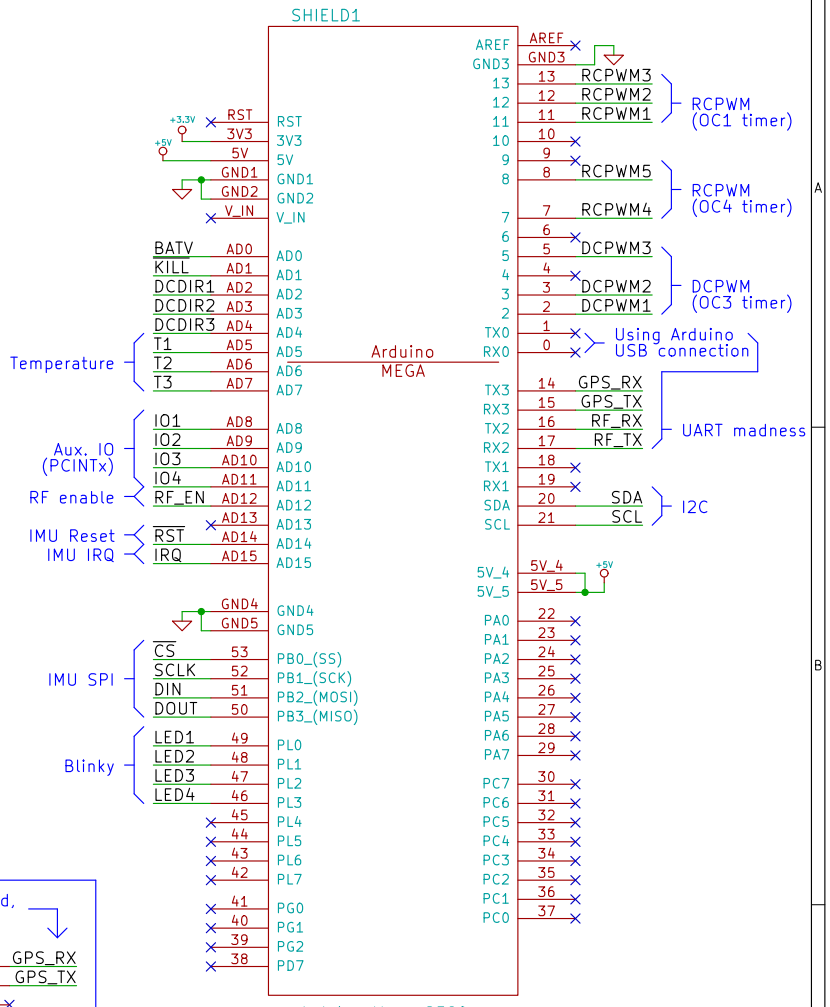
Blinkenlights



Kill Switch

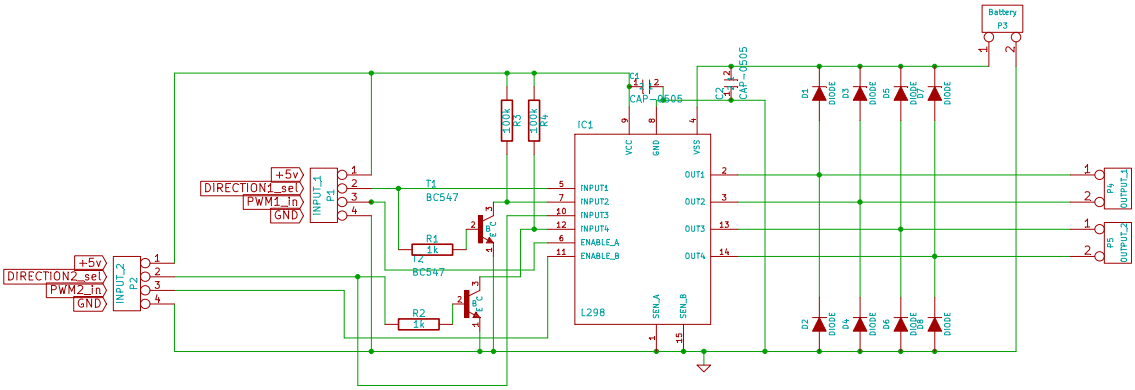


Disconnects supply to all actuators



3.3 V from Arduino
5.0 V from Arduino
5.0 V seperatly directly from battery packs
14.8 V battery supply

AAUSHIP		
Nick Østergaard		
Aalborg University		
File: lli–mega–shield.sch		
Sheet: /		
Title: Low Level Interface		
Size: A4	**Date: 28 nov 2012**	**Rev: 1**
KiCad E.D.A.		Id: 1/1



AAUSHIP		
File: bowThrusterControl.sch		
Sheet: /		
Title: Bow thruster		
Size: A4	Date: 12 nov 2012	**Rev: 1.2**
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable		Id: 1/1

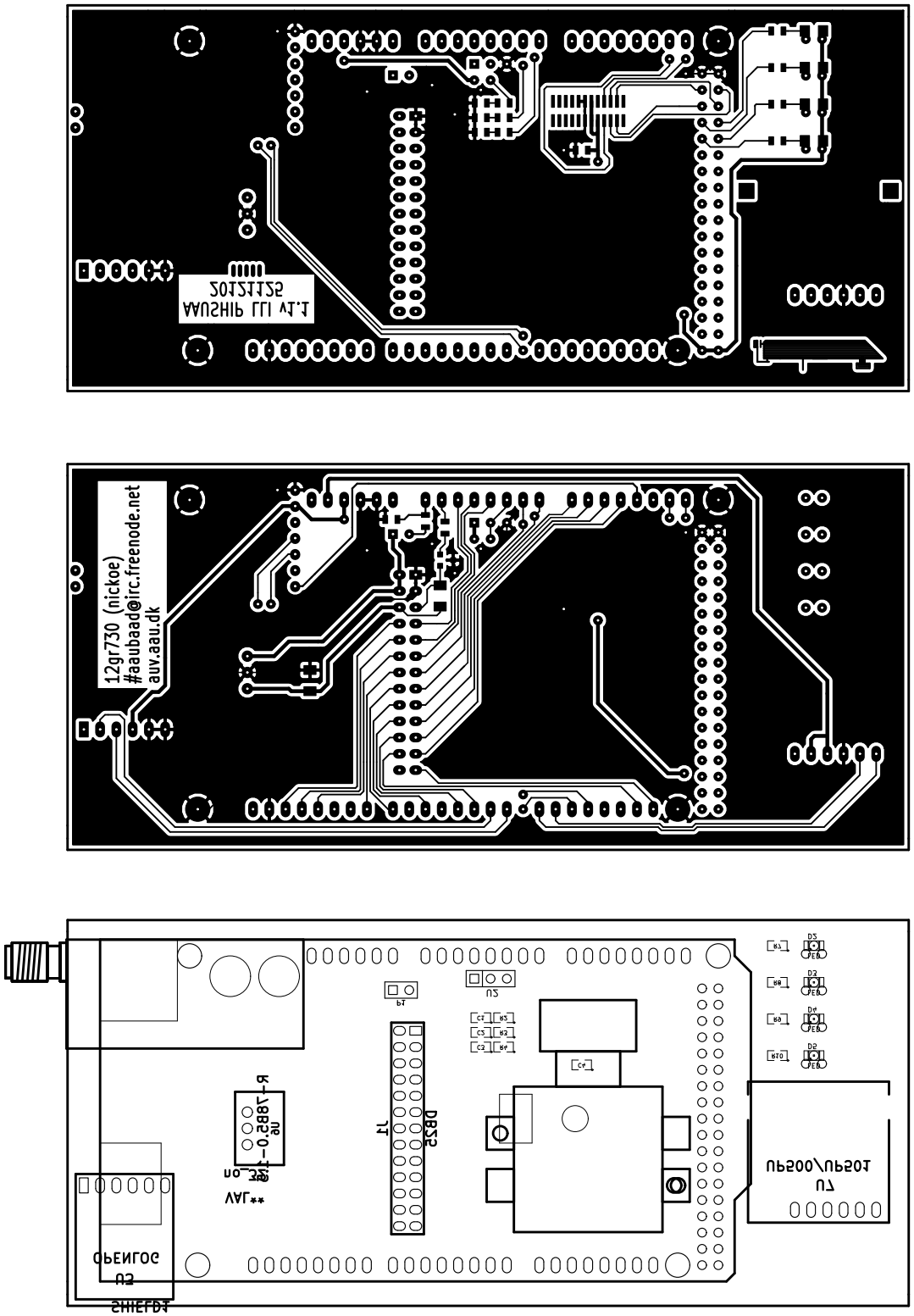


Figure 7: Hardware layout for the LLI

System Flowcharts

The following Flowcharts demonstrate the slight difference between the simulated and embedded application.

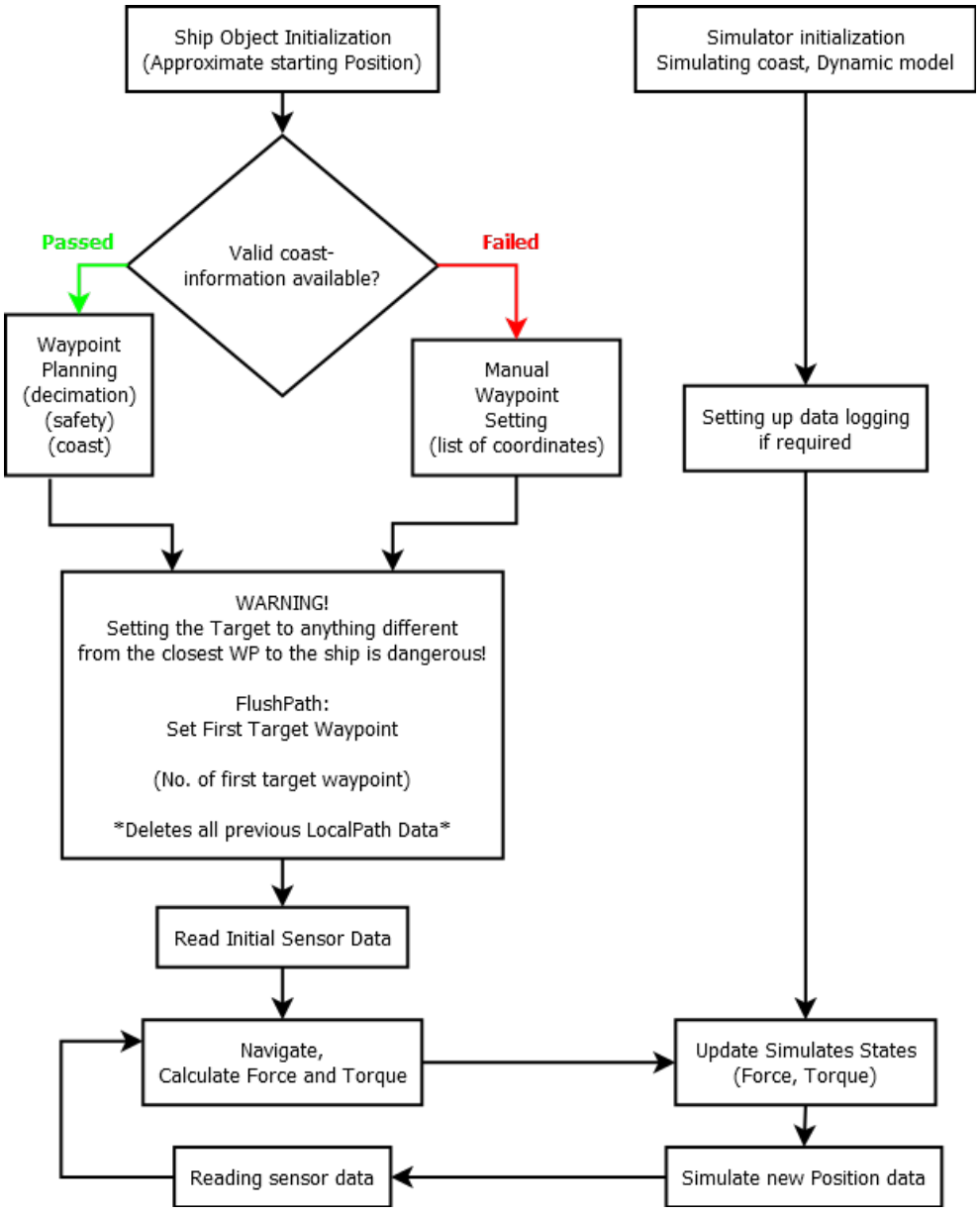


Figure 8: Flowchart of the Simulated Oceanography Task

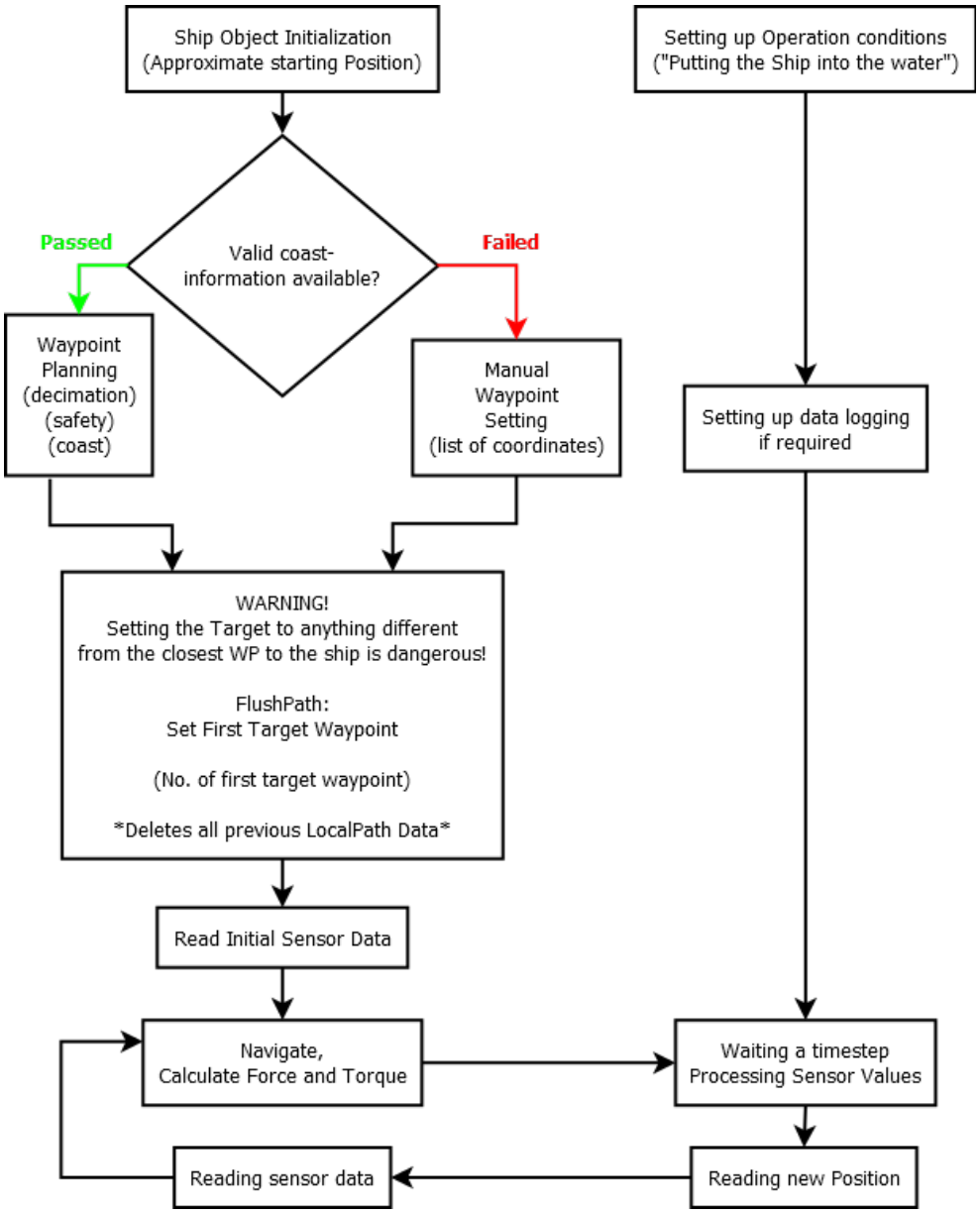


Figure 9: Flowchart of the Embedded Oceanography Task

Bibliography

István Komlósi & Bálint Kiss. Mobilis robotok autonóm navigációja mozgó akadályok elkerülésével. Master's thesis, Budapest University of Technology and Economics, 2011.

Brian Bach Nielsen, Nick Østergaard, Simon Als Nielsen, Rasmus Lundgaard Christensen, and Christoffer Stagsted Andreassen. *Initial Development of a Course Keeping Vessel*. 2012.

Robert H. Stewart. *Introduction to Physical Oceanography*. 2008.

Arthur N. Talbot. *The Railway Transition Spiral*. 1901.

List of acronyms

CRC	Cyclic Redudancy Check
DOF	Degrees-of-Freedom
ECEF	Earth-Centered, Earth-Fixed
ESC	Electronic Speed Controller
FAPS	Floating Autonomous Positioning System
GPS	Global Positioning System
GRS	Ground Segment
HLI	High Level Interface
IMU	Inertial Measurement Unit
KF	Kalman filter
LLI	Low Level Interface
OSM	OpenStreetMap
PWM	Pulse Width Modulation
SSS	Single Screw Ship
SSM	State Space Model
RF	Radio Frequency
TSS	Twin Screw Ship
WGS84	World Geodetic System 1984