TTK4551 - Specialization Project

Data Synchronization in Maritime Target Tracking

Didrik Grove

June 14, 2021

Norwegian University of Science and Technology (NTNU)

Preface

This report presents the work done during my specialization project at the Norwegian University of Science and Technology (NTNU) fall of 2020. It is submitted as part of the requirements to achieve a masters degree in cybernetics and robotics at NTNU.

I would like to thank my supervisors Giorgio D. K. M. Kufoalor from Maritime Robotics and Edmund Førland Brekke from NTNU for their guidance and advice during the project period. I would also like to thank Giorgio for the work he did to set up the target tracking pipeline to support the data analyzed in this thesis and Maritime Robotics in general for the opportunity to help develop and use all their hardware for the project. Their employees have been of tremendous help when setting up and running their systems for the experiments.

Finally, i would also like to thank my partner for taking great care of our newborn son while i have been finishing up the project and writing this final report.

Trondheim, June 14, 2021
Didrik Grove

Abstract

Correct timing of sensor data serves a vital role in the development of autonomous systems. Without reliable sensor data a system would not be able to make qualified decisions when deciding how to act on complex environments, and most state of the art sensors support some method for time synchronization. Much research has been done into improving synchronization and correct time stamping of sensor data, however for a target tracking setting it is hard to find data about the consequences of poor time stamping of sensor data.

This project thesis gives an introduction to the most commonly used synchronization methods and investigates the consequences of poor time synchronization. An Otter USV from Maritime Robotics is equipped with a LIDAR and two electro-optical cameras. Several experiments are conducted where sensor data synchronized with a state of the art synchronization board is compared with a more crude method for synchronization. Sensor data collected by the LIDAR is fed into a target tracking pipeline where the quality of the tracks are compared with the targets actual GPS position.

The results from the experiments are be discussed and the thesis attempts to quantify the consequences and importance of good synchronization for autonomous vehicles travelling congested waters with little room for safe navigation. The experiments show that the sensor data needs to be delayed for well over 1500 milliseconds to have a significant impact on the quality of a tracked target when each boat is travelling at a speed of around 0.8 meter per second. The difference in time of arrival and time of capture synchronization on a local system is measured to be as low as 170 milliseconds.

Contents

List of Figures

List of Abbreviations

USV	Unmanned Surface Vehicle
UAV	Unmanned Aerial Vehicle
LIDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
ToA	Time of Arrival
ToC	Time of Capture
ToT	Time of Trigger
NTP	Network Time Protocol
EO	Electro-Optical
IR	Infrared
PPS	Pulse Per Second
PPM	Parts Per Million
IMU	Inertia Measurement Unit
INS	Inertial Navigation Systems
I/O	Input / Output
API	Application Programming Interface
IMM	Interactive Multiple Models
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
ROS	Robot Operating System
MB	Megabyte
NED	North East Down

1 Introduction

1.1 Motivation

Over the last years the funding and commercial interest for autonomy has skyrocketed. With the technology that has been developed during the last two decades, autonomy has gone from being a term only used in scientific context to being something we interact with and deal with in our daily life. Many of us now have some form of artificial intelligence in our living rooms, we have access to self-driving cars and companies are able to land rocket boosters returning from space, all by using some form of feed back control.

Because of the oil industry and a thriving fish industry, Norway is one of the leading countries when it comes to research and development of maritime solutions. With the increase in demand for autonomy in the maritime industry, Maritime Robotics seek to increase the situational awareness of their vessels and become one of the leading developers within the field. Autonomous systems demand the highest degree of performance, reliability and safety when in operation, and on a system that is exposed to waves and shifting weather, a small error in time delay might give a large error further down in the decision making pipeline. Having all timing requirements and corresponding delays quantified will increase the performance and ease the development of autonomous systems. Knowing the limitations on a system from a data synchronization perspective one can make qualified decisions during development that will not negatively impact the final product.

Maritime Robotics is making a sensor hub which will be used in their autonomy project. This hub is mountable on several vessels of different sizes and is built with flexibility in mind, with interchangeable sensors and the possibility to make adjustments and change data synchronization methods. The results from this project will give a base for further development and optimization of this sensor hub.

1.2 Problem Description

The goal of the specialization project is to measure the most commonly used synchronization methods up against each other towards a real-time target tracking problem. Quantifying the time-constraints on a system given by different synchronization methods will be a key part of this task, with the goal of setting a strong foundation for future development of real-time target-tracking systems.

This project is split into two main parts. Initially a literature study is going to be conducted with the main focus being on synchronization of different exteroceptive situational awareness sensors, mainly with a focus on RADAR, camera and LIDAR. With this as a base, the report will then discuss the role of data synchronization in sensor fusion.

The second part of the project is to measure and discuss the delays in the sensor hub. A target tracking method is going to be implemented on this to study the effects of the different data synchronization methods, including the use of dedicated synchronization boards.

1.3 Related Work

There has been a lot of research and there are several research articles that discuss methods for time synchronization within autonomy and sensor fusion. Edwin Olson suggests a passive solution to the synchronization problem in [16], with an algorithm that attempts to calculate the delay from the sensor data is captured to the time it is received at the host computer. The article by Sigurd Albrektsen and Tor Johansen about the Sentiboard [4], a dedicated synchronization board for UAVs, also discusses the impact and improvements in time synchronization by using dedicated hardware.
It is however hard to find data about the impact of these different synchronization methods in a target tracking setting. How much of a time delay is negligible and how much of an timing error is introduced to the system with the most conventional synchronization methods? The reason there is little research on this might be because of the different nature of such systems. The timing constraints in a real time target tracking system on a drone and a maritime vessel will be different, and it is hard to make a general statement that covers all fields of autonomy. This project will be limited to timing constraints in target tracking in a maritime setting.

1.4 Report Outline

Chapter 2 will give an introduction to some of the more commonly used methods for data- and time-synchronization. In chapter 3 the report will give an introduction to data collection with LIDAR and camera, and will discuss the consequences and importance of good synchronization of sensor data. Chapter 4 will introduce the setup of the most important hardware and software that is used in the experiment, as well as the synchronization setup. Chapter 5 gives a description of the experiments conducted and their results, while 6 concludes the report and suggests future work.

2 Methods for Data Synchronization

Time stamping of the incoming sensor data serves a vital role in obtaining correct situational awareness for a system. All of the sensor information received sets the foundation for the control algorithms who decide how to act on the environment, actuating the system and moving the vehicle. Most autonomous systems rely on high-quality sensors of different sorts, and poorly synchronized data from these will reduce the accuracy and reliability.
The synchronization methods used vary between the different types of sensors. Cameras and inertia measurement units (IMU's) are sensors that can sense their surroundings within an instance, while the most popular LIDAR and RADAR sensors often contain rotating parts and requires more time to initiate in order to make full use of their high field of view. Due to the different nature of these sensors, different synchronization methods are required. This chapter will describe the most commonly used synchronization methods used and list their advantages and shortcomings.

2.1 Time of Arrival

The time of capture (ToC) is the exact time that sensor data is captured, while time of arrival (ToA) corresponds to the time the sensor data is received at the host computer. The exposure time, or *capture time*, corresponds to the time it takes for the sensor to sense the environment. The readout time is the amount of time it takes for the sensor to put all the sensed data onto the network, while the transfer time is the duration it takes to transfer the data over a particular interface. Processing time is the amount of time a host computer uses to receive data packets from the network and for the driver to gather a complete frame of sensor data. This processing time is highly dependant on other tasks being run on the computer, as the delay before the computer can time stamp the sensor frame might become large if the computer is handling several other tasks.

```
while not rospy.is_shutdown():
    status, img = camera.read()

    if status == True:

        image_toa = bridge.cv2_to_imgmsg(img, encoding="passthrough")

        image_toa.header.timestamp = rospy.Time.now()
```

Figure 1: Timestamping with ToA in ROS

If the amount of data sent to the host computer is relatively low, we can expect the difference between ToC and ToA (see Figure 2) to be low. The readout time, transfer time and processing time might be negligible when dealing with little data, and time-stamping data by simply using ToA might be a relatively robust solution. This is often the case for an IMU, where the amount of data processed is relatively low and does not take a lot of time to transfer.

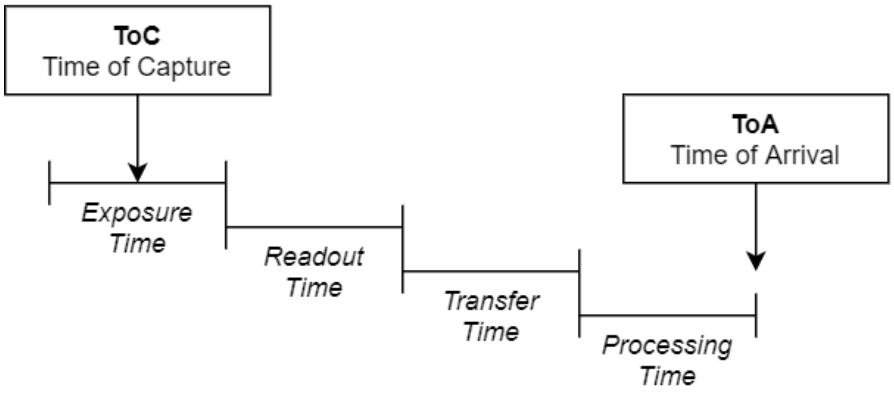

Figure 2: Extrapolating ToC from ToA

For high quality cameras outputting images on a gigabyte network however, this transfer time might not be negligible. Transfer of high resolution images can take everything from a few milliseconds to several seconds, depending on compression, transfer method and transfer distance. This will thus add extra delay to the system that give error in time stamping of the sensor data. Because of the amount of data transferred from state of the art cameras, vendors often give the ability to poll the cameras by the user to output the latest exposure-time, which can be used together with readout-times from data sheets to extrapolate a more precise time of capture. This reduces the error by a good margin, however network transfer time and delays in the host computer due to other processes are uncertainties that such a method can not precisely account for. ROS (Robot Operating System [2]) is a commonly used framework for software development on robots, and many drivers for different sensors in this framework use time of arrival as their time-stamping scheme. An example of how this is done is shown in Figure 1.

2.2 Sensor Triggering

Another way of obtaining an accurate time of capture is to use sensors that support triggering. This means that the sensor starts capturing data about its environment when receiving a signal, and most newer state of the art cameras support both software and hardware triggering. A software trigger is most often in the form of a command sent through the sensors API, while a hardware trigger is an electrical pulse from an external source onto an input-pin. This method of synchronization does require more from the sensor, and is often only possible with the more expensive alternatives, but it is also considered to be a lot more precise. While exposure- and transfer times for cameras can vary a lot, making it hard to extrapolate ToC from ToA, it is a lot more robust to extrapolate ToC from a hardware-trigger as shown in Figure 3.

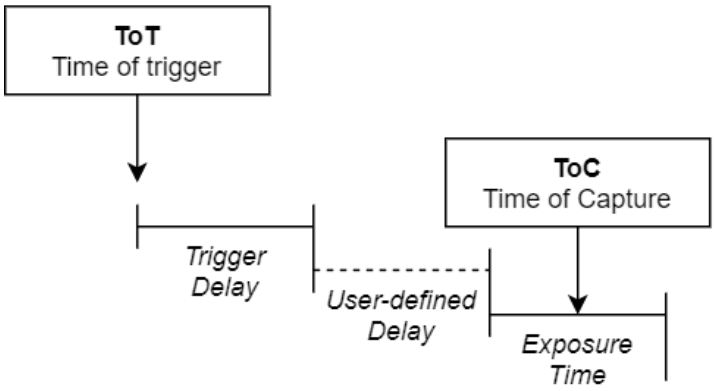

Figure 3: Extrapolating ToC from ToT

Most vendors provide accurate data in the data sheet about the internal delay between time of trigger (ToT) and start of exposure, often down to nanosecond precision. One problem that arrives with hardware-triggering of sensors is to tie the correct trigger signal to the correct sensor data. The trigger signal is often generated by an external source, and although the trigger signal itself is time-stamped, it is not tied to any sensor data. If capture is done at low frequencies this can be solved by making the assumption that the sensor data is tied to the most recent trigger signal, however at higher frequencies this might not be the case. Before the data has reached the host computer a new trigger signal might already be transmitted, and this could cause errors.
Many sensors also have the option to delay acquisition start using an internal counter. In combination with sensor data sheets, this programmable delay can be utilized to ensure different sensors capture at the exact same time using the same trigger source, should they have different internal delays. Using an internal counter can ensure that interocep-

tive sensors perform a reading at the same time as exteroceptive sensors.

2.3 Pulse Per Second (PPS)

Due to the nature of some sensors, triggering is not applicable. LIDAR and RADAR are examples of such sensors which are constantly rotated to achieve a high field of view, and they are mainly built to output data at a set frequency. Instead, most of these sensors have an internal clock that can be synchronized with external clocks for a common time frame. A popular way of synchronizing these sensors is through the use of a pulse per second (PPS) signal. This signal consists of two parts, a time stamp referencing the start of every second and an electrical pulse used to reference the time stamp.
Most GPS-receivers output such a pulse, and depending on the equipment this pulse can represent the start of a second down to 1 nanosecond accuracy. GPS satellites are setup with atomic clocks to ensure precise timing, and are commonly used as master clocks. Because GPS solutions use a constellation of satellites, the delays in the signals can be analyzed through trilateration [9] to find the actual delay between the timing sources and to compensate for travel time. The GPS time signal precisely references the start of every second, with the time stamp referencing the number of seconds since 00:00 the 6th of January 1980.

One might not need synchronization with GPS time if all the sensors are localized on a single system. The need for GPS synchronization to an external master clock adds complexity, and local clocks can also serve as the PPS master. Relative to each other the clocks will still be very well synchronized, but they lose a common time reference that can be used to synchronize with external systems.

2.4 Network Time Protocol (NTP)

If there is a need for synchronizing clocks on a distributed network a commonly used method is to setup an NTP-structure like the one shown in Figure 4 with the local system synchronizing to external time sources. While NTP is a protocol that can be used to synchronize with your own sources (e.g. GPS receivers), there are several publicly available timekeeping servers used for NTP-synchronization, split into different layers (also called stratums) which identifies their precision. Stratum 0 are the highest quality of timekeeping servers and include atomic, radio and high-precision clocks such as those in a GPS satellite. The next layer, stratum 1, are those directly connected to stratum 0

devices and devices on the same layer. The protocol supports up to 256 layers, however with every layer the synchronization to the top layer and the most precise clocks deteriorates [14].

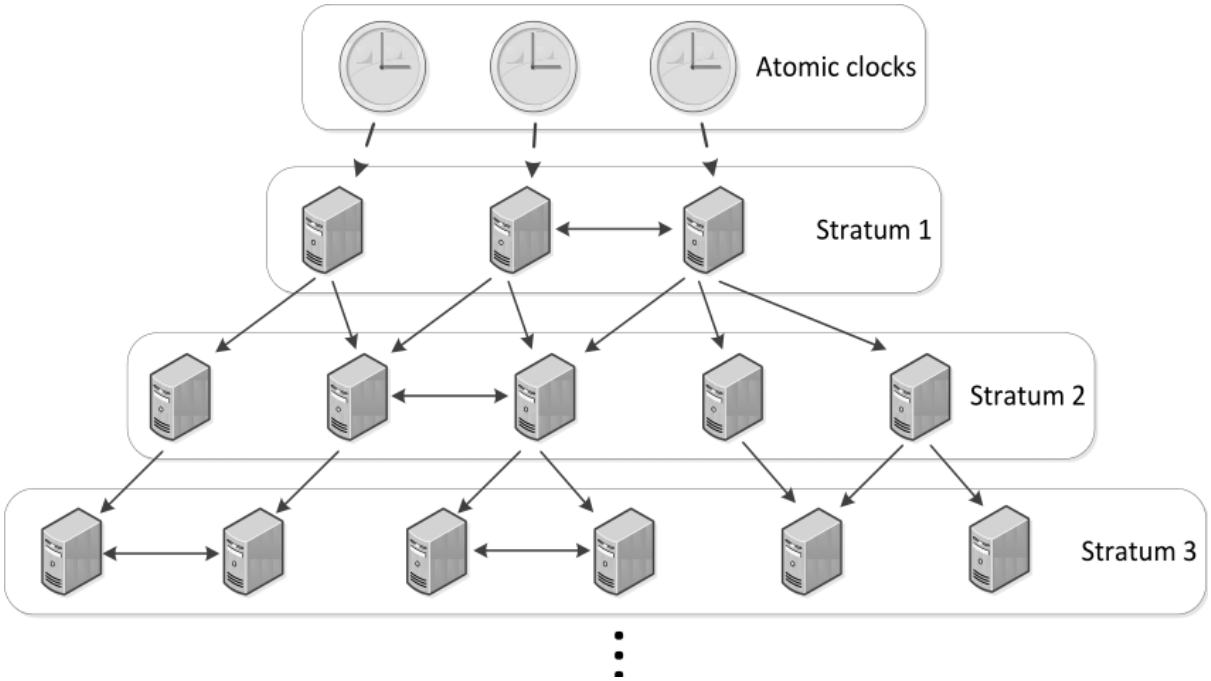

Figure 4: Structure of a NTP synchronized network [5]

Clients in this network of timekeeping servers are able to request accurate time from the servers. The system logs the time it takes to receive a response and uses this to extrapolate the response time from the server, increasing precision over long distances. To account for noisy measurements and differences in response time, clients do not instantly synchronize their clocks with the master. Instead this happens over a period of time in a feedback loop. Figure 5 shows how synchronization is done in a NTP client. The first and second columns identifies the server, while the third column identifies the stratum level. The fifth and sixth column represent the time since last polling and the polling interval. The delays are the response time from the timekeeping servers in milliseconds, and the offset is the offset on the local clock relative to the servers.

2.5 Precision Time Protocol (PTP)

While NTP servers provide accuracy on a millisecond level, certain systems require even more precise methods for time synchronization. PTP, following the IEEE1588 standard,

```
[root@fs5 etc]# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+vimo.dorui.net  209.51.161.238   2 u   10   64  377   65.341  -13.123   4.908
*time-b.timefreq .ACTS.           1 u   11   64  377   27.818  -12.278   5.287
+clock1.alb1.ino .CDMA.           1 u    7   64  377   68.468  -12.429   3.845
-mirror          173.230.149.23   3 u    8   64  377   84.308  -11.351   2.707
```

Figure 5: Responses to a client in a NTP synchronized network [13].

can reliably provide accuracy down to a nanosecond or even picosecond level depending on the hardware that is used.

The main difference between NTP and PTP is that the latter uses dedicated hardware to acquire more precise synchronization. As is the problem with time of arrival synchronization, an NTP server cannot properly calculate the processing time of the incoming data packets, which can vary a lot depending on the loads on the host computer. PTP solves this by having dedicated hardware that synchronizes towards high level stratum servers, and more accurately timestamps responses from time servers. Because this hardware is dedicated towards timestamping incoming data packets, it is also a lot easier to provide accurate statistics on the internal delays in such systems. This hardware is also often equipped with low drift oscillators, and should they loose connection to GNSS or timekeeping servers, they can stay within an accuracy of between a few seconds to a few milliseconds over a year depending on the hardware.

3 Data Synchronization in Target Tracking

3.1 Target Tracking

In a maritime setting, a target tracking algorithm needs to be versatile and dynamic enough to be able to detect and track everything from large ships to smaller debris and buoys. Targets of the same size have different dynamic properties. A fishing boat and a recreational boat of the same size are expected to act very differently, however looking at these though the eyes of a LIDAR or a RADAR, it is hard to securely apply either of these two models to the target. Figure 6 shows an example of this. While the LIDAR point clouds are relatively similar, we humans would know that the vessel on the left side of the picture would behave very different from the vessel to the right.

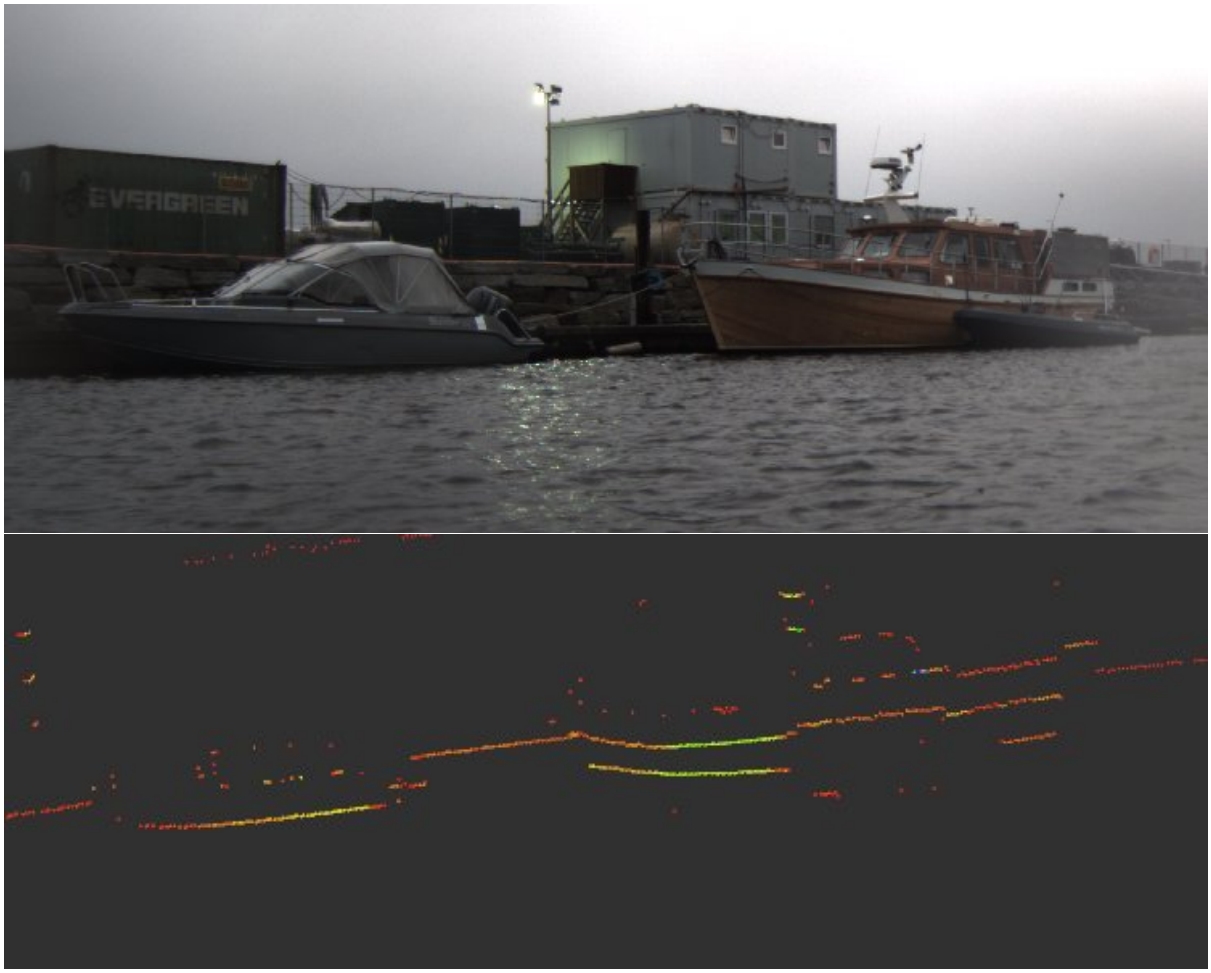

Figure 6: Two vessels that behave very differently seen through a LIDAR

Because of this limited knowledge, models in tracking algorithms needs to be flexible and general. Position and velocity estimated through a filter thus contain significant uncertainties, and reliable sensor data is important to obtain good tracks.

To be able to judge where the detections from the different sensors are relative to ones own vessel one needs to describe their placement relative to each other. This is done using the homogeneous transformation matrix described below.

$$T_b^a = \begin{pmatrix} R_b^a & r_{ab}^a \\ 0\,0\,0 & 1 \end{pmatrix} \tag{1}$$

Here R_b^a describes the translation and r_{ab}^a describes the rotation of point b relative to point a. In ROS, which is used in the experiments described in chapter 5, these transformations are defined as rotations in roll, pitch, yaw and translation in x, y and z-direction.

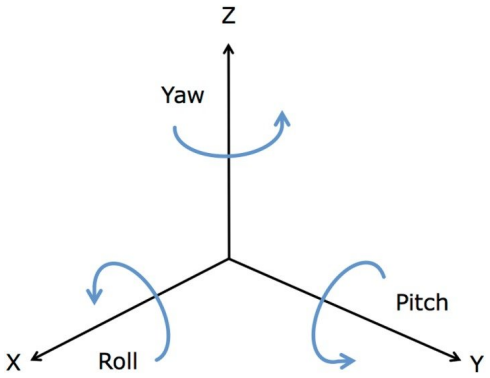

Figure 7: Transformation between two coordinate systems [8].

3.2 Camera Detections

Camera detections are nowadays often done through the use of convolutional neural networks. These networks are a class of deep neural networks, most commonly used in image analysis. Neural networks are built on the concept of neurons behaving somewhat similar to the neurons in the human brain, where different inputs trigger different reactions and outputs. This section won't go into detail on the workflow of convolutional neural networks, however it will scratch the surface of some of the most common object detection algorithms and explain briefly how these detections can be used in sensor fusion.

Figure 8: An object classifier would be able to tell that this image contains a boat but give no further spatial information.

The goal of a regular classification algorithm is to classify the content of the image, whether the image is of a car, boat or a plane. An object detection algorithm also attempts to draw a bounding box around the object to describe where in the image it is located. A regular classification algorithm also does not take into account more than one object within the image, while an object detection algorithm can detect several objects of interest within a single image without knowing the amount of objects beforehand. A naive approach to this problem would be to split the original image into a very large amount of smaller images and then run the object detection algorithm on each image. Because the objects can have different aspect ratios and be located in different sections of the image this would not be computationally tractable on a real-time system, as the

amount of possible locations would be extremely large. Doing this in an efficient manner is the problem that modern object detection algorithms attempts to solve.

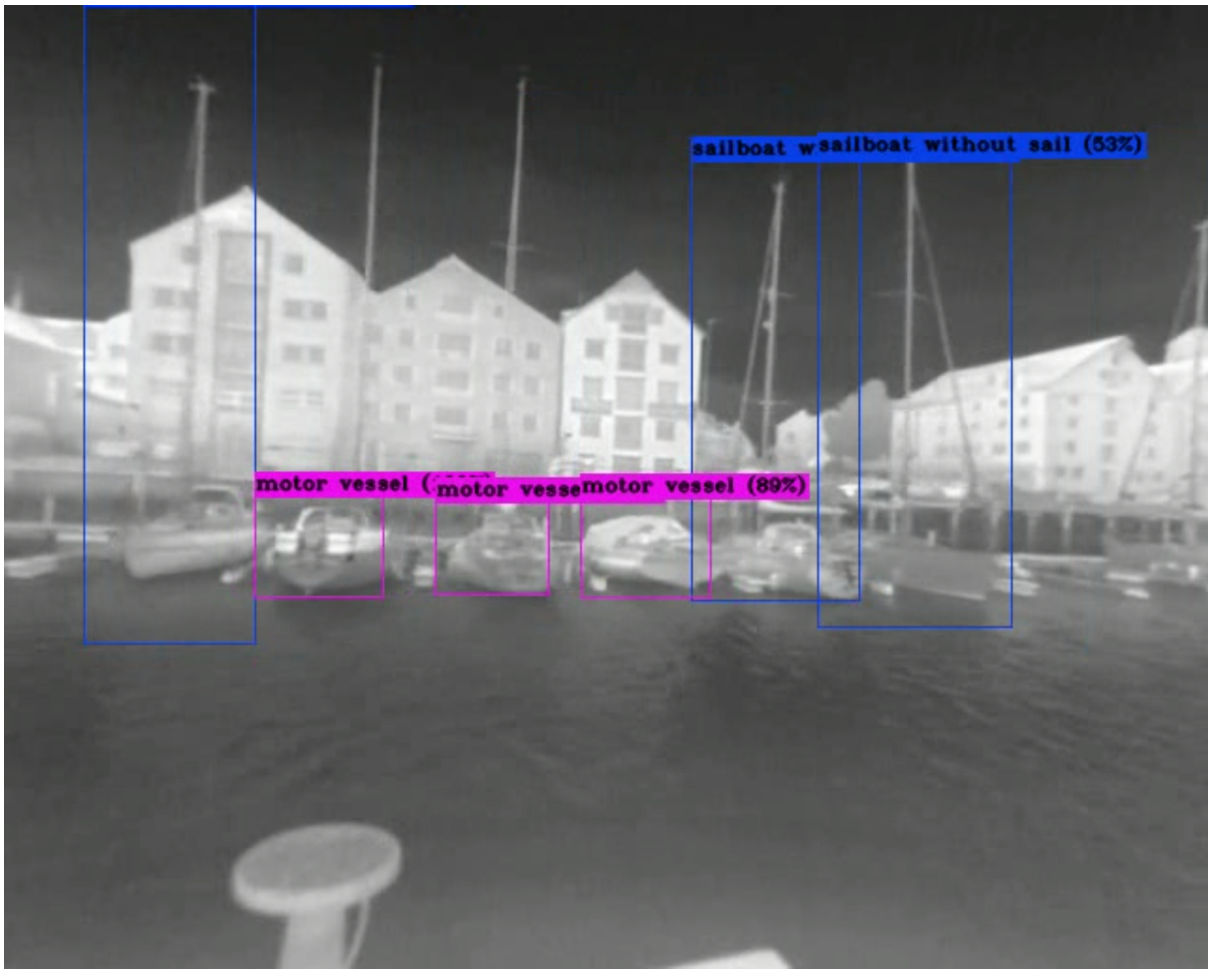

Figure 9: Object detections with YOLO V3 using an IR-camera

Two of the most popular object detection algorithms for real-time use are SSD and YOLO. SSD [12] uses extra convolutional layers of different resolutions that predicts where the bounding boxes are located and their size, then SSD runs a classification algorithm on each suggested bounding box. YOLO [17] instead applies a single neural network to the full image (hence the name "You Only Look Once"), which divides the image into regions and predicts bounding boxes and probabilities for each region. Because running a SSD is a two-step process it is a bit slower than YOLO, however image processing still happens within reasonable time for a real-time system. The trade-off for YOLO is limited accuracy on objects that are small and smaller objects located close to each other due to the limited size of the region proposals.

As detections with cameras in a mono-setup are done in 2D-space they can only give accurate information about an objects height and angular position relative to the camera and give limited information about the distance to a target. With the recent development in convolutional neural networks, however, they are also a very convenient tool for object classification to find information about the targets dynamic properties. They are also often applied as a human-machine interface [11].

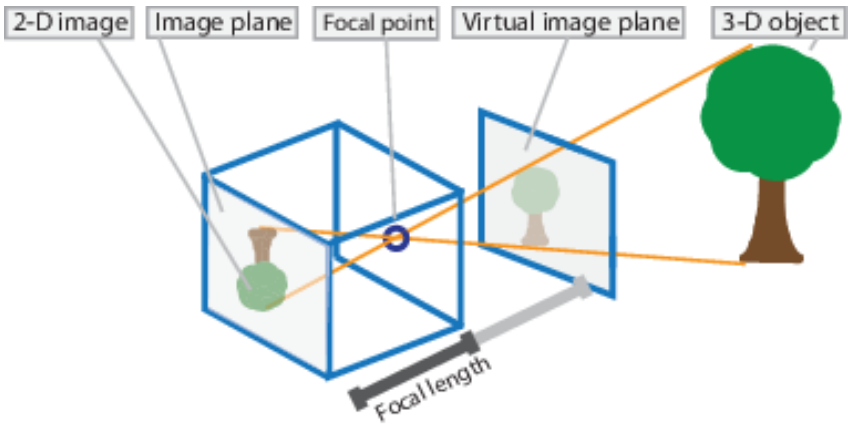

Figure 10: Pinhole camera model [15]

Accurate static transformations to the camera and its exintric, intrinsic and distortion parameters are needed to precisely describe the transformation of an object in the 2D image relative to the camera in 3D space. Newer cameras have specialized lenses that are built to minimize distortion caused by the curvature of the camera lens, but in general there is always a certain degree of distortion to account for. This is more thoroughly explained in [10].

The pinhole camera model describes transformations between three different coordinate frames. The world coordinate frame represents coordinates of the 3D object in the real world, and the virtual image plane represents 3D coordinates in the camera, with the origin at the center of projection and the Z-axis as the optical axis. The 2D image plane is pixel coordinates within the image. The transformation from the 3D object in the world frame to the 3D object in the camera frame is given as a standard 3D coordinate transform like the one described in equation 1, although often denoted as the extrinsic calibration matrix M_{ex}. The transformation between the virtual image plane and the 2D image plane is given with the intrinsic calibration matrix which is given on the following form.

$$M_{in} = \begin{pmatrix} fs_x & 0 & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

Where the parameters are as follows

f	Focal length
(o_x, o_y)	Piercing point coordinates
s_x	Pixel size in x-axis
s_y	Pixel size in y-axis

These can be used to achieve a mapping between a vector in the world frame $\overrightarrow{X}$ to pixel coordinates $\overrightarrow{p}$ as a set of linear transformations

$$\overrightarrow{p} = M_{in} M_{ex} \overrightarrow{X} \tag{3}$$

3.3 LIDAR Detections

While an EO-camera captures the reflected natural light in its lens, the LIDAR detects nearby objects by emitting light in a certain wavelength and measures the time of flight of the reflections. The LIDAR contains filters that excludes all light outside the desired wavelength, and in an automotive application the beam is often emitted and reflected through a rotating mirror to achieve a high field of view. The LIDAR outputs detections in a spherical coordinate frame, with range calculated by time of flight.

$$\mathrm{R} = \frac{\text{Speed of Light} \times \text{Time of Flight}}{2} \tag{4}$$

The rotated angle of the mirror gives the azimuth angle α, and a LIDAR can also have some vertical resolution with mirrors and reflectors mounted at different angles to add elevation angle ω. These can be transformed into a Cartesian coordinate frame using the equations

$$\begin{aligned} x &= R\cos\omega\sin\alpha \\ y &= R\sin\omega\sin\alpha \\ z &= R\cos\alpha \end{aligned} \tag{5}$$

The different data points that are reflected and registered by a LIDAR is often stored and visualized as a point cloud like the ones in Figure 6. Different clustering techniques are applied to organize the data points into clusters that share some similarities like the boxes in Figure 11. These clusters are usually used as a detection in a tracking pipeline.

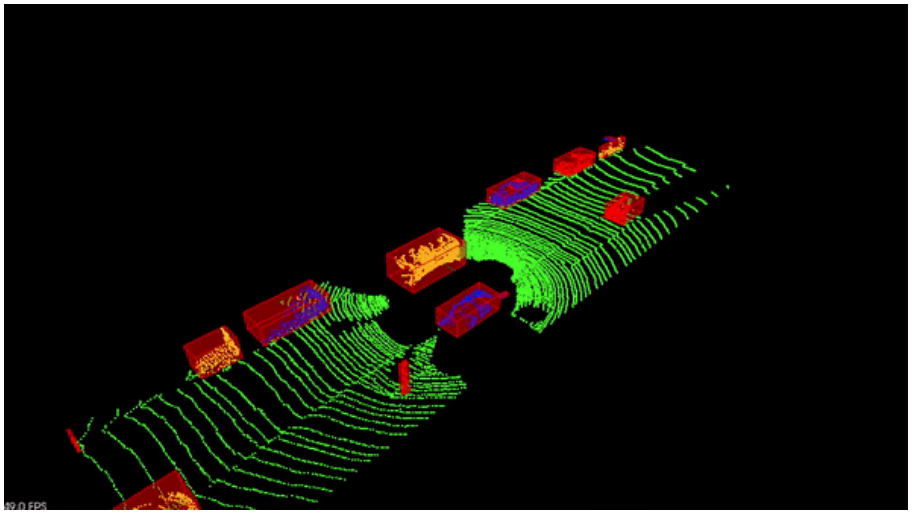

Figure 11: LIDAR Clusters [1]

3.4 Data Synchronization Problems

With an error in timing of the sensor data, the accuracy of the information will be uncertain. With little dynamic behaviour such an error is hardly noticeable, however with a lot of maneuvers they become more apparent. As an example, the Otter USV (described further in chapter 4.1) can rotate at a rate of around 90 degrees pr. second. With cameras mounted and perfectly calibrated transformations, an error in time stamping of just 50 ms would give an error in angular positions of around 1.6 meters on a detection that is just 20 meters away. A vessel of this size is also very susceptible to external forces such as waves, as can be seen in Figure 12.

Figure 12: Screen captures from "https://youtu.be/oIBdRzhpJpU?t=82"

Target tracking algorithms based on the Kalman filter can account for such inaccuracies

in the measurement model by increasing the uncertainty of the incoming measurements, however this degrades the performance of the tracking pipeline which already has to account for of other uncertainties.
As explained in chapter 3.1 target tracking filters need to be dynamic and general enough that they can track targets with different dynamical behaviour. The standard approach to this problem is to use different filters in parallel which applies different models and represent different dynamic behaviour. The "Interacting Multiple Model"-method (IMM) is an implementation of this which uses probability theory and Gaussian mixture reduction to merge all the filter probabilities into one state estimate. Attributing the wrong dynamical properties to an object because of an error in timing will degenerate the performance of such a filter.

In rough weather with high movement in pitch like what is seen in Figure 12 one might also get false detections of other targets because of poor timestamping, as waves might be attributed as other vessels.

4 Experiment Setup

Chapter 5 of this report will expand on the experiments conducted to quantify the constraints on a system with different synchronization methods. The pipeline used in data collection and the different time constraints in the system will be expanded upon in this chapter.

4.1 Otter USV

The Otter is an unmanned surface vehicle (USV) (Figure 13) from Maritime Robotics and will serve as the main development platform for their autonomy project. The idea is to have a high grade of scalability, as the end idea is to increase the autonomy on all of their products.

Figure 13: Otter USV

The Otter is the smallest USV delivered from Maritime Robotics and provides an easily deployable and modifiable testing platform for autonomy development, and is used for collecting data in the experiment. It is usually equipped with payload equipment performing tasks like seabed and environmental mapping. The Otter USV is 2 m long and 1.2 m wide, and because of its size and capabilities, it is often used in areas that cannot be reached by larger vessels. This increases the requirements for an autonomous system, being able to maneuver and make qualified decisions in smaller areas.
Because of the small size, it is also more susceptible to weather in open areas. Relatively small waves will cause big fluctuations to the Otter, and a small error in timing the sensor data might cause a large shift in the sensor frame like shown in Figure 12.

4.2 Sensor Hub

Because of the Otters modularity, a separate targa is constructed for the purpose of autonomy development. This targa interfaces with all the sensors and contains the processing power needed for the Otter to develop situational awareness. The targa and the different sensors can be seen mounted on the Otter as seen in Figure 14.

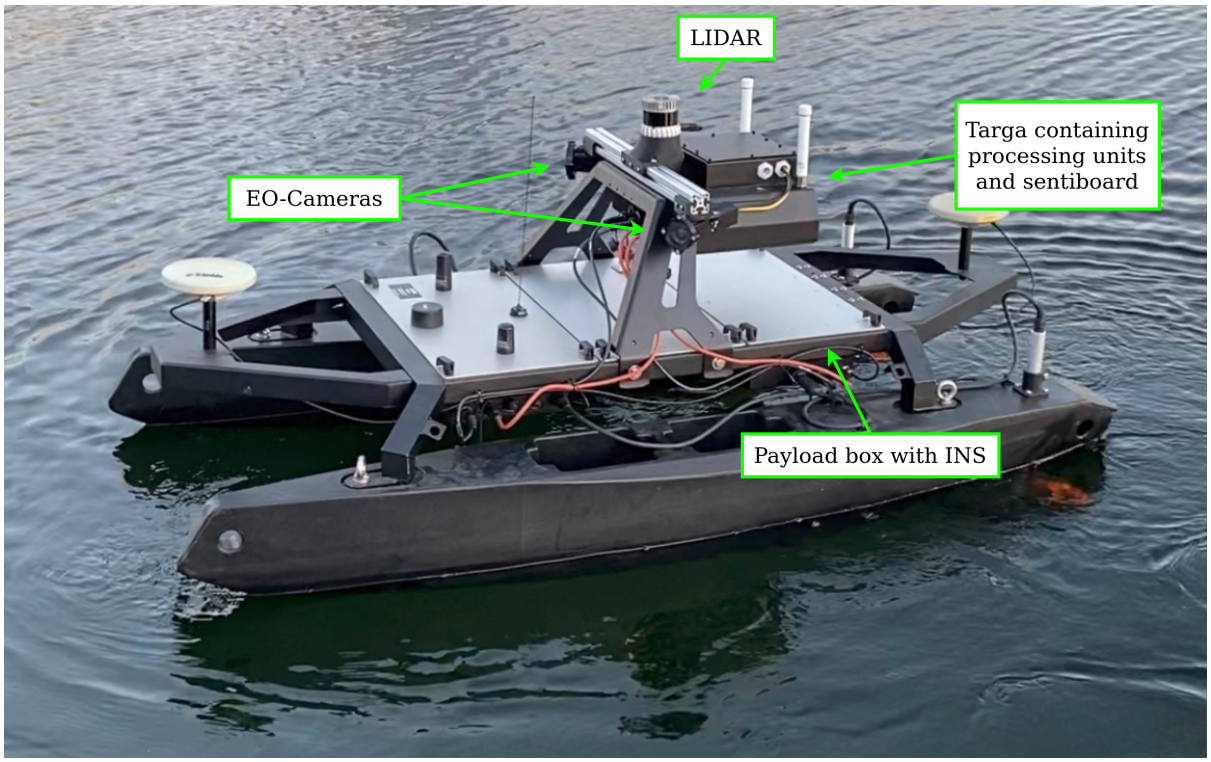

Figure 14: Otter with targa for autonomy development

The Otter is a versatile vessel, and is most often used to perform tasks like environmental monitoring and seabed mapping. The sonars that are used for mapping contains INS (Inertial Navigation System) with a very high accuracy, and the output of these are transferred to the targa and used for autonomy. For this project however, a Kongsberg mini-MRU [18] (Motion Reference Unit) is used to keep track of position, velocity, angular orientation and angular velocity. The MRU outputs time-stamped data and is synchronized with GPS time using PPS. It has an error in timing of less than 1 ms.

4.2.1 Sentiboard

The Sentiboard (Figure 15) deals with most of the low level time synchronization between the sensors on the rig. It has two SPI, two RS-232, three UART and one RS-422 I/O that all can be used to communicate with a range of different sensors [3]. Each I/O can be setup as an hardware interrupt when receiving data from a sensor. This data is re-packed and transmitted with an extra header containing a time stamp synchronized with GPS-time.
Each I/O can also be setup for hardware-triggering and is able to provide a steady PPS signal. It can also synchronize all of the I/O across each other, so you can ensure that all outputs trigger sensors at the same time. The trigger signals can also be output at a different frequency relative to each other, while still maintaining synchronization.

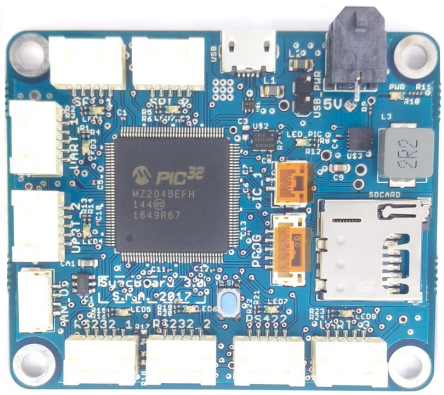

Figure 15: NTNU Sentiboard

The Sentiboard keeps track of time using a low drift oscillator with an accuracy of 10 PPM (parts per million.) This means that after a bit more than one and a half week (11.57 days) without any type of error correction, it can have a maximum inaccuracy of 10 seconds. Tests show that this synchronization board can relate sensor measurements to an absolute time reference with a clock drift of 1.9 microseconds per second RMS if it is connected to a GPS PPS-signal [4].

4.2.2 Cameras

The sensor hub is equipped with two Dalsa Genie Nano C4040 EO-cameras (Figure 16.) They provide 4112x3008 resolution images and are equipped with Fujinon CF8ZA-1S lenses providing 85.7 degrees horizontal field of view. The camera mounts are constructed such that the horizontal angle can be modified, giving support for both mono and stereo vision, and the cameras have support for hardware-triggering.

Figure 16: Genie Nano C4040

The API for the Genie-cameras can be used to continuously poll the latest exposure- and readout-time to help interpolate time of capture. The time delays within the camera can be viewed in Figure 17. The trigger to exposure delay in the figure is given as an absolute minimum and corresponds to the physical delays within the camera. This delay can be increased and set manually through the camera API.

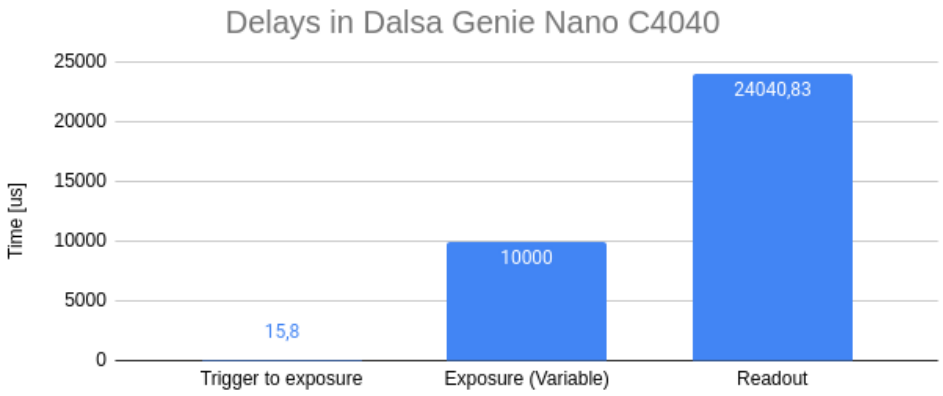

Figure 17: Time delays in Dalsa Genie Nano [7] with full image resolution. Note that the exposure time is variable and 10000 μs is an example value.

4.2.3 LIDAR

The sensor hub also has an Ouster OS-1 LIDAR (Figure 18.) It has a range between 0.25 m to 120 m, with a false positive rate of 1/10 000. Range resolution is around 0.3 cm, horizontal resolution is user-configurable between 512 to 2048 and it has 16 channels providing some vertical resolution. On max horizontal resolution it has a rotation rate of 10 Hz, but with lower resolution this can be increased to 20 Hz. The LIDAR is also equipped with an internal IMU with a 3-axis gyro and 3-axis accelerometer. It can use an internal 10 PPM drift clock as a timing source which counts the amount of nanoseconds since the LIDAR was last turned on. This internal time source can also be synchronized through a digital input with either GPS-PPS (explained in Chapter 2.3) or PTP (explained in Chapter 2.5.) The internal IMU and the LIDAR-data are both time-stamped with the internal clock. Ouster OS-1 can also be setup as a PPS-master, outputting a PPS-signal synchronized with the internal clock to synchronized with an external PPS-slave. This signal output can also be set at a different frequency or set to fire when the LIDAR captures data at a certain angle.

Figure 18: OUSTER OS-1

The Ousters internal clock has a timestamp resolution of 1 microsecond and a data latency of less than 10 milliseconds. Because the LIDAR timestamps packets with its internal clock it is convenient to synchronize this and use it as the reference time.

4.2.4 Jetson Xavier AGX

The rig contains two Jetson Xavier AGX computers from NVIDIA that does the data processing. For the data collection the computers interface with the cameras, Sentiboard and the LIDAR and synchronizes the camera's images with the Sentiboard. The development kits are shown in Figure 19 and are mounted inside the Otter targa.

Figure 19: Nvidia Xavier AGX Development Kit

In this project the computers are synchronized with GPS time through the Otter OBS as NTP-clients, but they also support synchronization through PPS. As the OBS is synchronized with GPS time these computers are considered a stratum 2 source.

4.2.5 Reference Frames

The sensors are connected with static transformations which can be viewed in Figure 20. The placement of the different sensors in the image correspond to the placement of the sensors in Figure 14, with the LIDAR mounted on the top and EO cameras in housings on the sides. The coordinate system in the bottom right is the output from the inertial navigation system (INS). All sensors on the targa are referenced to a common point on the targa body named "targa_base". This makes it mountable on different vessels without having to redo each individual transformation.

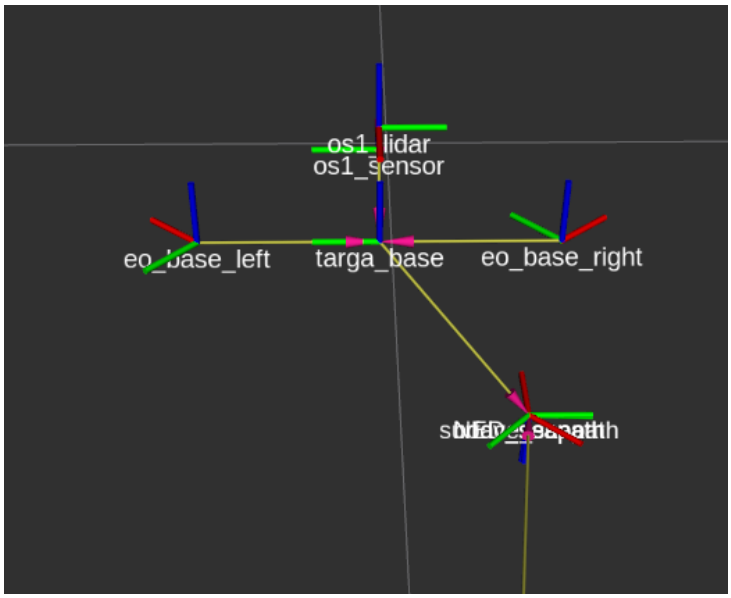

Figure 20: TF Tree in ROS

4.3 Software Setup

The software setup is done in ROS and the raw data analyzed in Chapter 5 is stored as rosbags. In this project the sensor data is stored in custom data formats and is stamped with both time of capture and time of arrival. The data from the rosbags is played back into a collision avoidance system developed in the Autosea-project [6] by NTNU in collaboration with DNV GL, Kongsberg and Maritime Robotics. This pipeline outputs the tracks that will be analyzed in Chapter 5.

4.4 Synchronization Setup

The current system has the NTNU Sentiboard synchronize with GPS time using PPS. This is then considered to be a stratum 1 source as described in Chapter 2.4. The position, pose and twist of the vehicle is output from a Kongsberg Mini-MRU and sent through the Otters On-Board System (OBS) to the targa where it is logged in ROS. The timestamping of this INS data is done on the MRU itself which is synchronized with GPS time.
The OBS is synchronized with GPS time via a u-blox receiver and setup as a NTP-server. The Xaviers are synchronized with this as NTP-clients. The cameras are hardware-trigged at a frequency of 5 Hz through the NTNU Sentiboard, and the time of the triggers are parsed in the sentiboard driver and broadcasted on a ROS-topic. This is merged with the camera images in the camera driver. The LIDAR data is timestamped locally on the sensor, with the internal clock of the OS-1 being synchronized with the GPS time messages from the OBS and a PPS-signal from the Sentiboard.

The timing setup in this system is visualized in Figure 21.

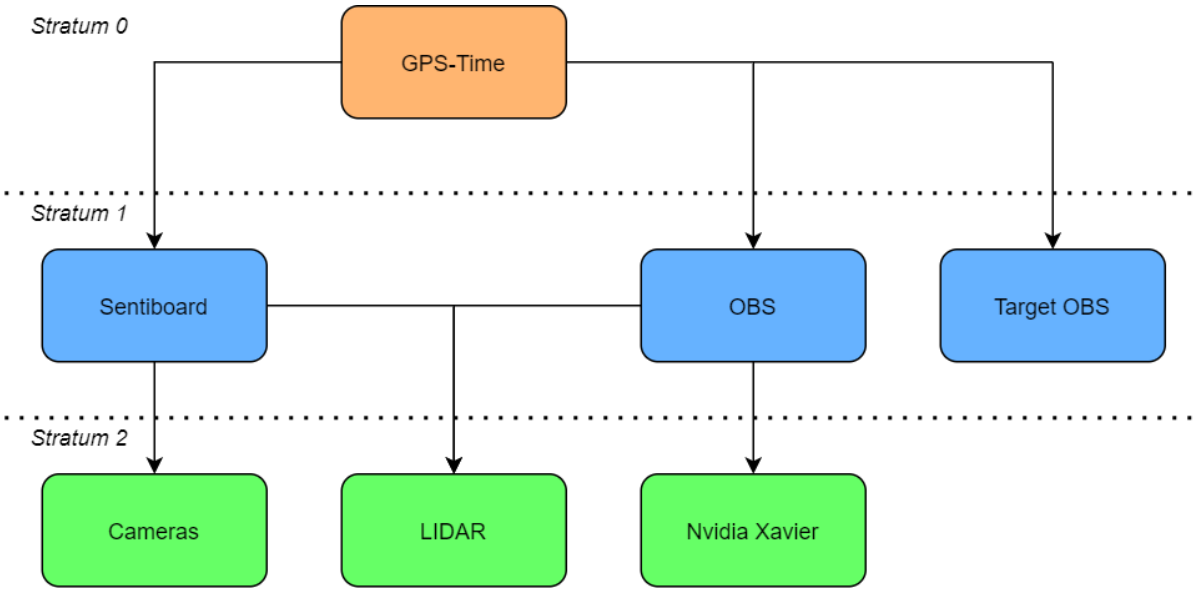

Figure 21: Time synchronization in the hardware setup

The delays between the stratum 1 sources should be minimal. With all three stratum 1 sources synchronized directly to GPS-PPS the clocks are synchronized with nanosecond precision. The Sentiboard adds a very small delay to the synchronization of the underlying sensors. As the LIDAR receives the PPS-pulse through the Sentiboard it adds a 1.9 microseconds delay to the signal. The cameras also have an internal delay of 15.8

microseconds from trigger to exposure like shown in Figure 17, and will have a total inaccuracy of around 17.7 microseconds.

4.5 Target Ship

To be able to judge the performance of the difference in synchronization in a target tracking setting we need to be able to attribute a ground truth to our target. A second Otter USV is equipped with a u-blox f9p development kit to record GPS-position and will serve as our target vessel, this position is logged locally with the time on the development kit as the reference time. This, like the time sources on own ship is synchronized with GPS-PPS. GPS position of the ground truth is given with a one meter accuracy as specified within the data sheet of the F9P GNSS module on the development kit [19].

5 Software vs. Hardware synchronization

5.1 Description

The experiment are limited to tracking targets at less than 100m to simulate activities in in-land waterways, narrow canals, harbors and congested water environments with limited sea room for safe navigation. The goal is to figure out if the delay in time stamping due to transmission of data from the sensor to the processing computer leads to significant errors in the tracking estimates. The tracking part of this experiment will only be based on sensor data from the LIDAR and the INS, and the difference in time stamping using time of trigger and time of arrival. Because of limited time the delay between the two time stamping schemes will be approximated by using the delays with the camera images.

The collected data is recorded at Trondheim harbour in three different scenarios shown in Figure 22, 23 and 24. The first scenario has little movement on both own ship and the target ship, the second scenario has little movement to own ship and some movement to the target ship, while the last scenario has some movement to own ship and little movement to the target ship. All three scenarios are conducted on the same day with a 20 minute interval and both Otters are travelling in a constant speed of around 0.8 meters per second.

Figure 22: Experiment Scenario 1 - Both ships passing each other driving in a straight line. Own ship making a turn and maneuvering back to the starting position.

Figure 23: Experiment Scenario 2 - Own ship driving towards the target while the target ship is making a turn. Own ship then makes a turn and maneuvers back to the starting position.

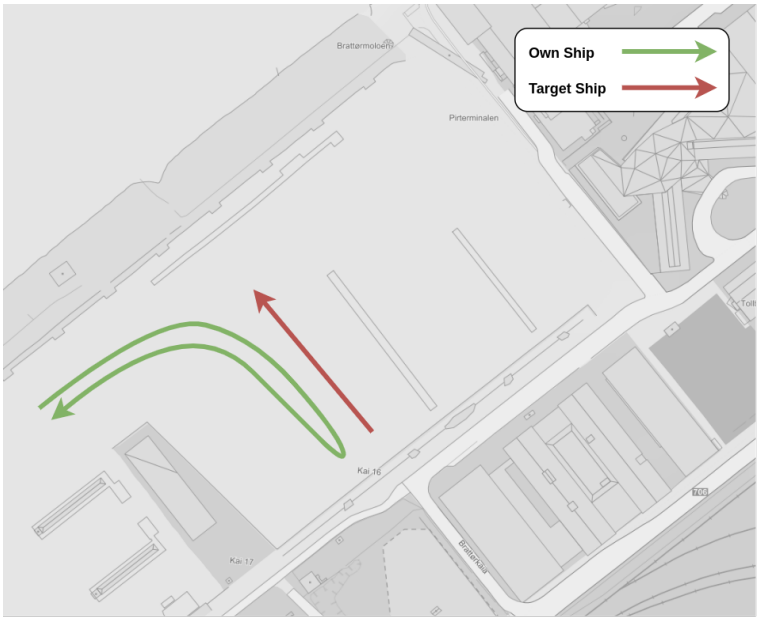

Figure 24: Experiment Scenario 3 - Own ship makes a turn while target ship drives in a straight line. Own ship turns at the end of the path and drives the same path back to the starting position.

5.2 Results

5.2.1 Observed Delays

The images delivered from the EO cameras are time stamped with two separate time stamps for this project. The header is stamped with ToA from the host computer and the ToC from the Sentiboard. For all three scenarios the delays between these two stamps are calculated and visualized in Figure 25, 26 and 27.

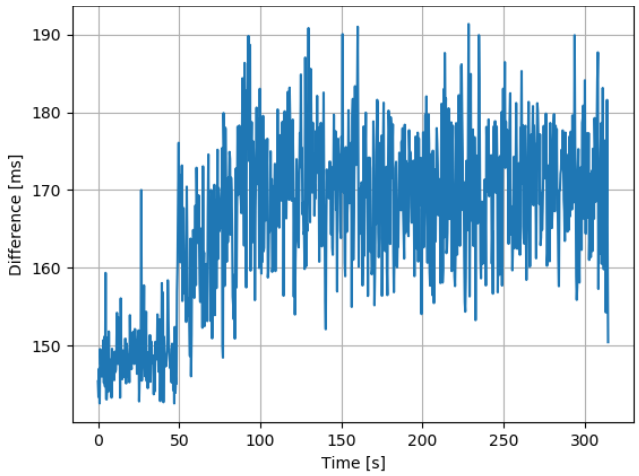

Figure 25: Time delays between ToA and ToC in Scenario 1.

Figure 25 shows a slight shift in the delay of around 20 ms after around 60 seconds. Looking at the position of the boat relative to the office (which is located on the top left in Figure 22 to Figure 24), it is highly likely that this is due to the Otter driving out of range of the office wireless network. The OBS has a dedicated antenna for wi-fi signals, and when it looses this connection it will instead transmit data over a radio on the targa. This data goes through the same network switches as much of the sensor data, and the increased loads on these is probably the cause of the increase in delay. The two figures on the next page shows the delay in the second and third scenario.

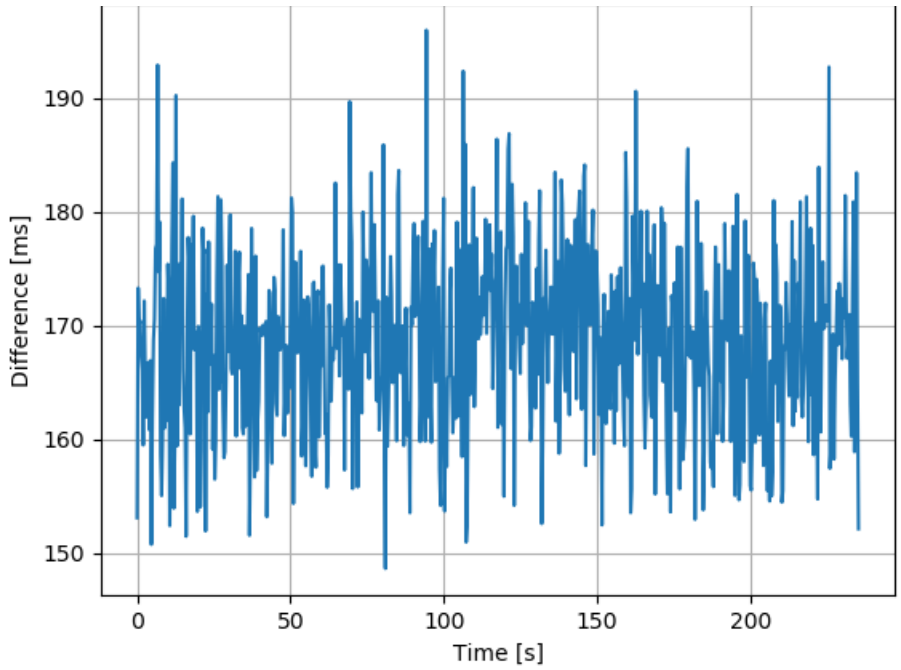

Figure 26: Time delays between ToA and ToC in Scenario 2.

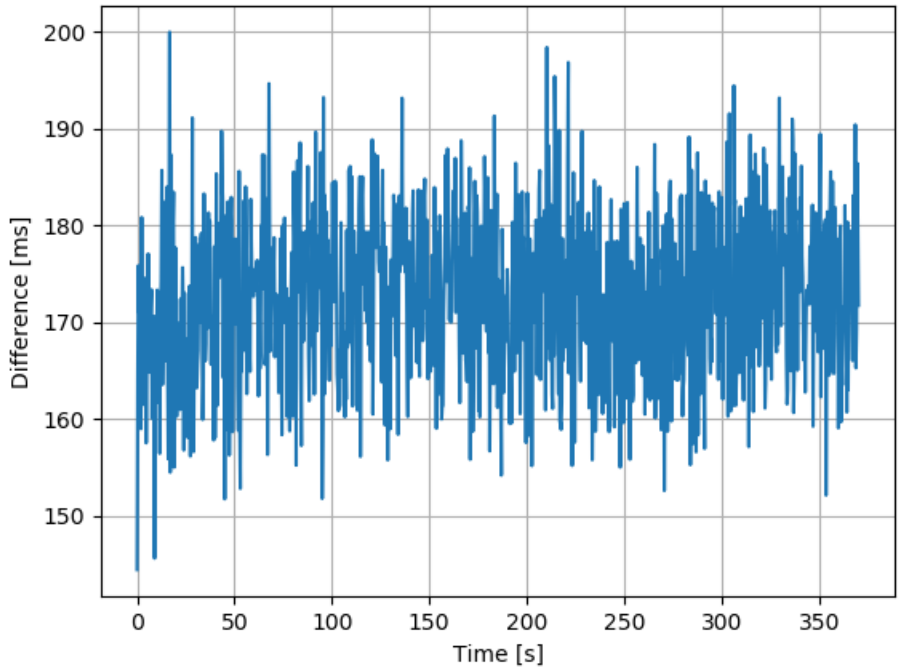

Figure 27: Time delays between ToA and ToC in Scenario 3.

The delays can be modelled as Gaussian distribution like whats shown in Figure 28, 30 and 31. This approximation could be used in the measurement model of a target tracking pipeline to help model the measurement uncertainty. Note however that the distribution in Figure 28 is shifted slightly to the left and has a larger variance because of the lower delay at the start of the scenario.

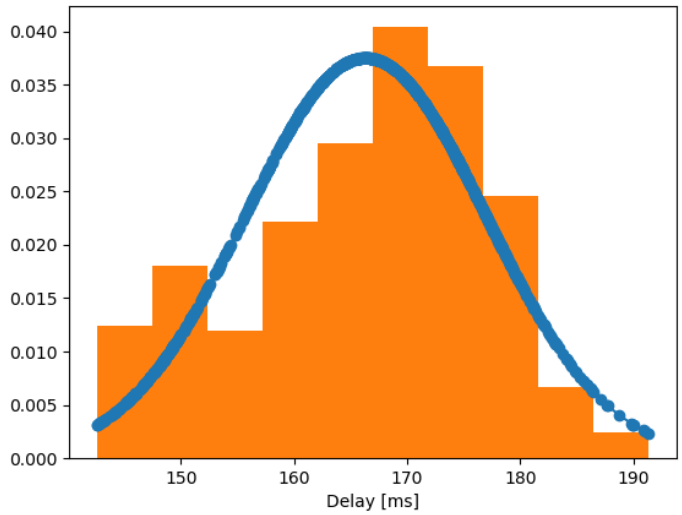

Figure 28: Distribution of delays during scenario 1

$\mu = 166.3$ $\sigma = 113.1$

As stated in Chapter3.4 the Otter can rotate at a rate of 90 degrees pr. second. For a camera detection with an expected delay of around 170 milliseconds we can calculate the expected error in target position as a function of distance using some basic trigonometry. This error is visualized in Figure 29.

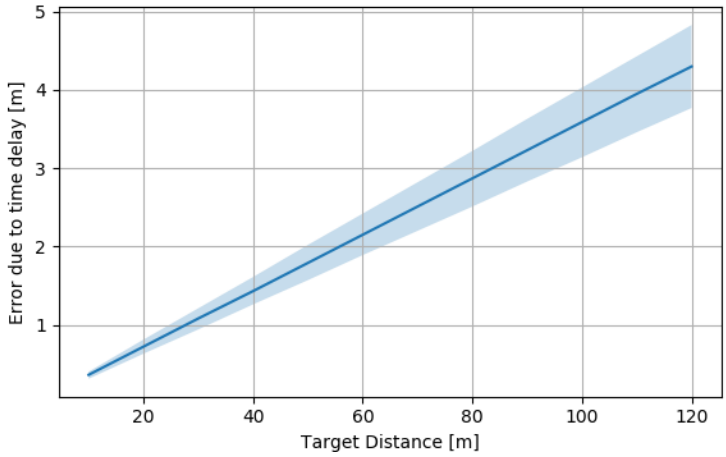

Figure 29: Detection error due to timing delay

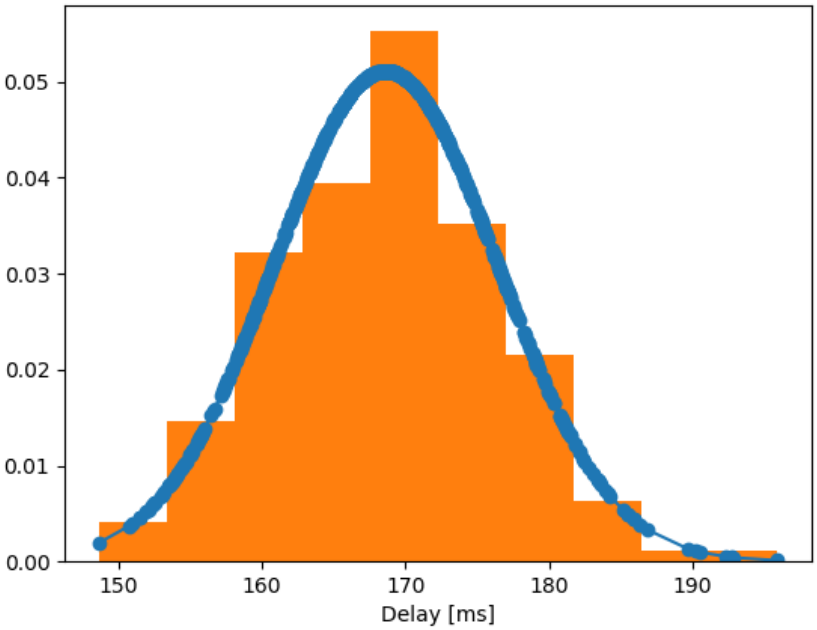

Figure 30: Distribution of delays during scenario 2

$\mu = 168.6$, $\sigma = 60.9$

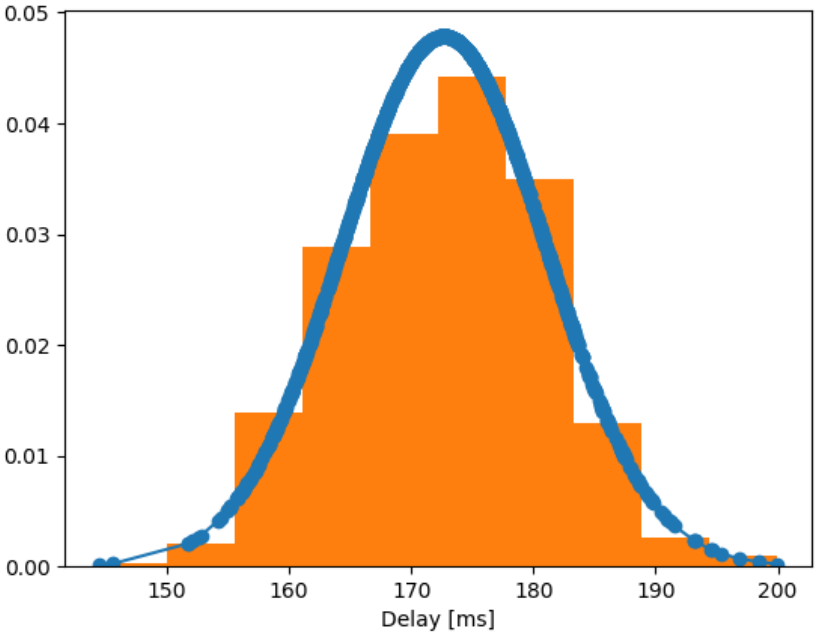

Figure 31: Distribution of delays during scenario 3

$\mu = 172.7$, $\sigma = 69.5$

When measuring the delays between messages in Figure 32 we observe that the ToA timestamping scheme is the biggest contributor of noise during these scenarios. While ToC has a steady update rate of 3 Hz the measurements from ToA come in a lot more irregularly. This is as expected since ToA time stamping will sometimes be blocked by other processes as explained in Chapter 2.

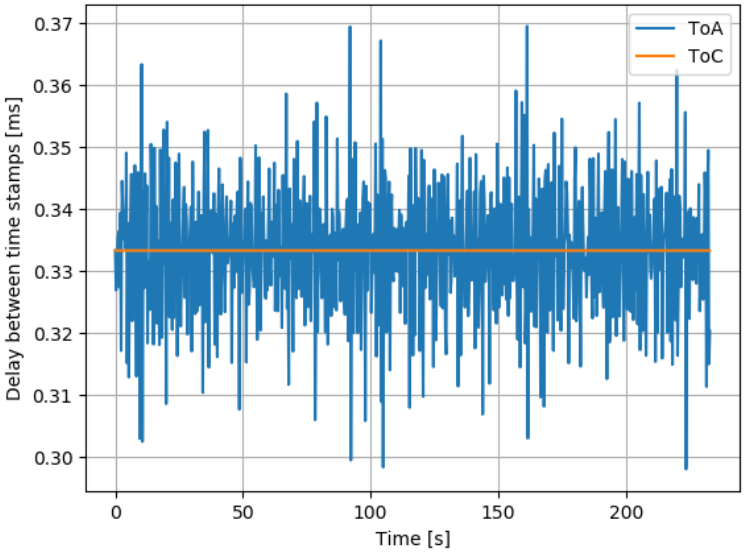

Figure 32: Time stamping noise

The transfer of camera images take up around 110 MB/s as shown in Figure 33. The LIDAR outputs data at 32MB/s, however the LIDAR this is a limited amount of data points and it is about to go out of production. We can assume newer state-of-the-art LIDARs to transmit data equal to that of the camera images. The OS-1 LIDAR used in the data collection has 16 vertical channels, however the same manufacturer now only delivers LIDARs with 32, 64 and 128 vertical channels with the same horizontal resolution. The amount of data transferred increases linearly with the amount of points, and the delays in the camera should be equal to that of the 64 channel LIDAR. The delays shown in Figure 28, 30 and 31 will hence serve as a good approximation of the delays introduced in a state of the art LIDAR.

```
average: 111.59MB/s
        mean: 37.11MB min: 37.11MB max: 37.11MB window: 100
average: 111.57MB/s
        mean: 37.11MB min: 37.11MB max: 37.11MB window: 100
average: 111.69MB/s
        mean: 37.11MB min: 37.11MB max: 37.11MB window: 100
```

Figure 33: Amount of data streamed from a single camera.

5.2.2 Track Quality

The Ouster OS-1 LIDAR sometimes output very few point around the target because of the low resolution and the black pontoons of the Otter absorbing the light. This varies from a single digit amount of points up to the hundreds when the target is close to the LIDAR, so it is hard to tune the segmentation so it does not filter out the few data points available to the tracker while simultaneously not accepting noise. In this project however we are more interested in the track of the target than the quality of the sensor data and the data points not corresponding to the target ship. Because we know the actual position of the target ship through GPS position we use the actual position to filter out all the points not corresponding to the target. This is done by the means of a box filter. This filter is set to filter out any data points that are not within a 5x5x2 meter cube with the target as the origin. An example of the cloud output is shown in Figure 34.

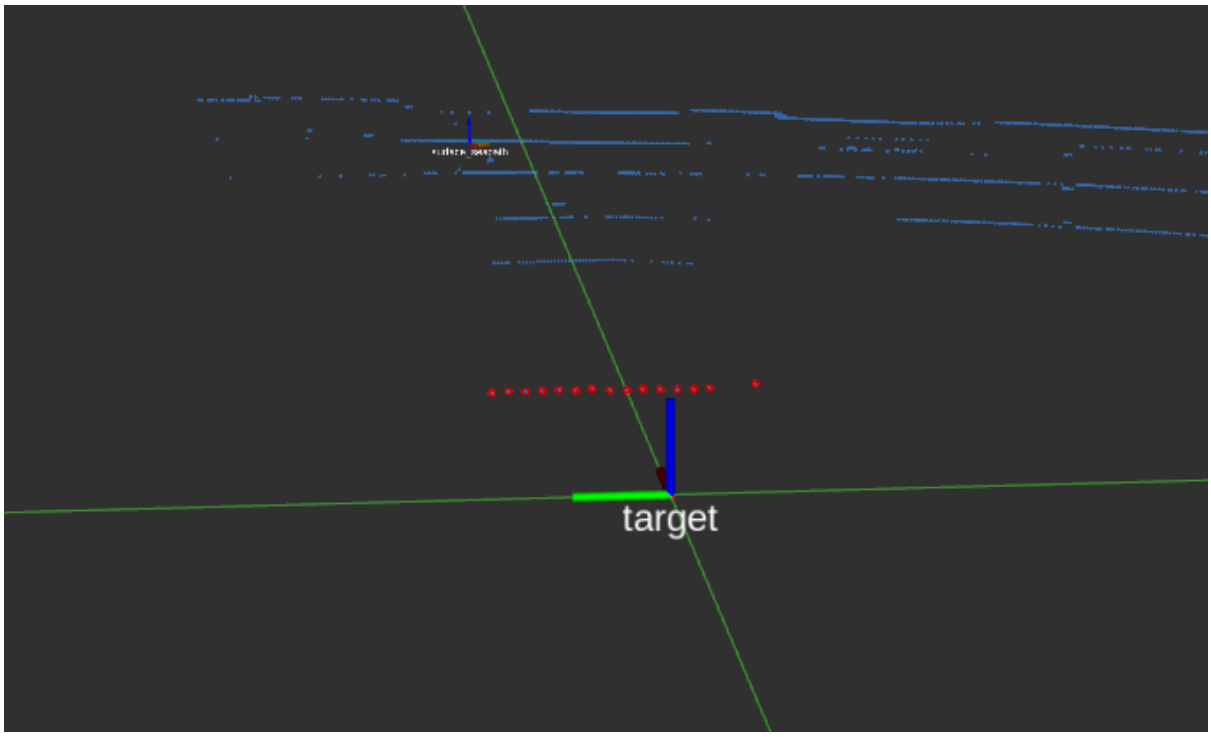

Figure 34: Box filtering of point cloud. Red points around target transform are filter outputs and the blue dots are parts of the original point cloud.

It should also be noted that due to the scope of the project and time limitations there has been limited time put into tuning of the tracking pipeline.

The first scenario shown in Figure 22 is created to observe the consequences of delays introduced in a constant velocity scenario. Both ships are travelling towards each other and pass half way into the scenario. Because of the range and limited data points we are unable to get a good track for the entire movement that is shown in the figure in Chapter5. We will instead focus on a smaller period of about 30 seconds where the target is within range of the LIDAR. This period in particular is also chosen because it has few sensor measurements in the start and more towards the end like shown in Figure 37. In this scenario we can also observe how the amount of measurements impacts the quality of the track with a time delay. The movement of the two boats during this period is illustrated in Figure 35, they start 41 m away from each other and end up side-by-side.

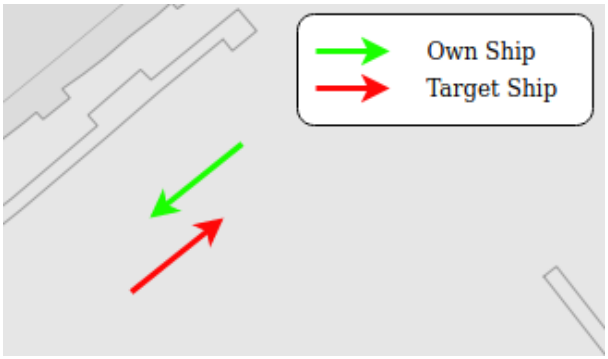

Figure 35: Track from scenario 1

Different delays to the point cloud is simulated ranging from 150 ms to 1500 ms with a 150 ms interval and the mean error is visualized in Figure 36.

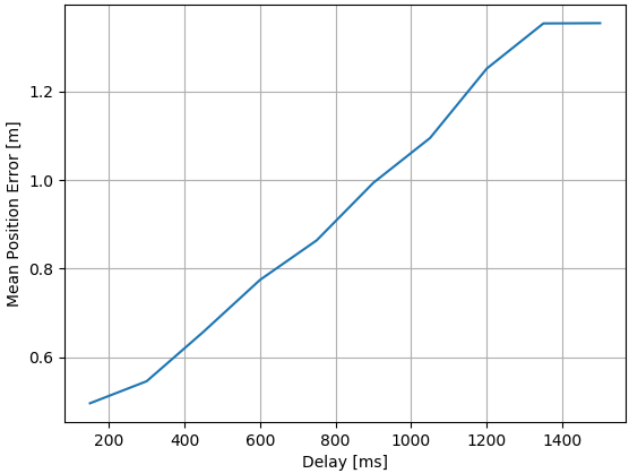

Figure 36: Mean positional error from time delay

The track estimate and GPS position in the NED-frame is plotted for the lowest and highest delays. In the lower left side of the plots we observe that the measurements (green dots in Figure 37 and 38) are more gathered around a straight line with a low delay, following the GPS-output. With a larger delay the measurements are more spread which degenerates the quality of the track. The delay in sensor data cause the clustering in the tracking pipeline to have increased uncertainty in the data.

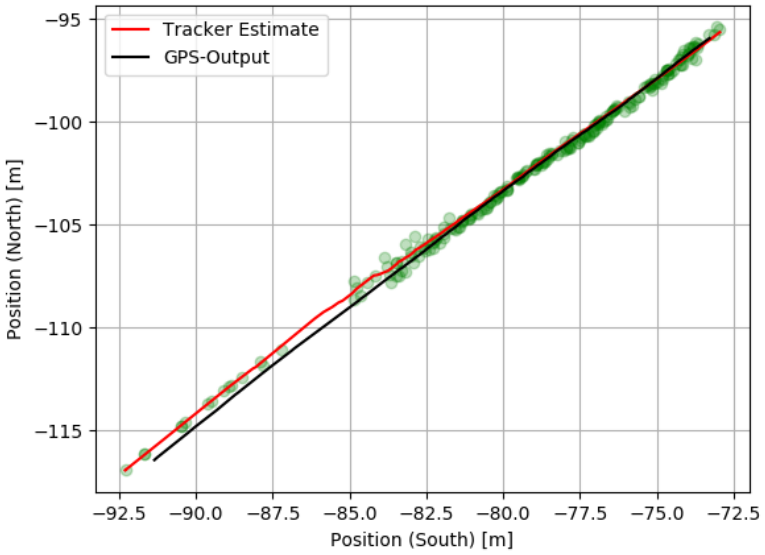

Figure 37: Track and GPS position in NED frame with 150ms delay

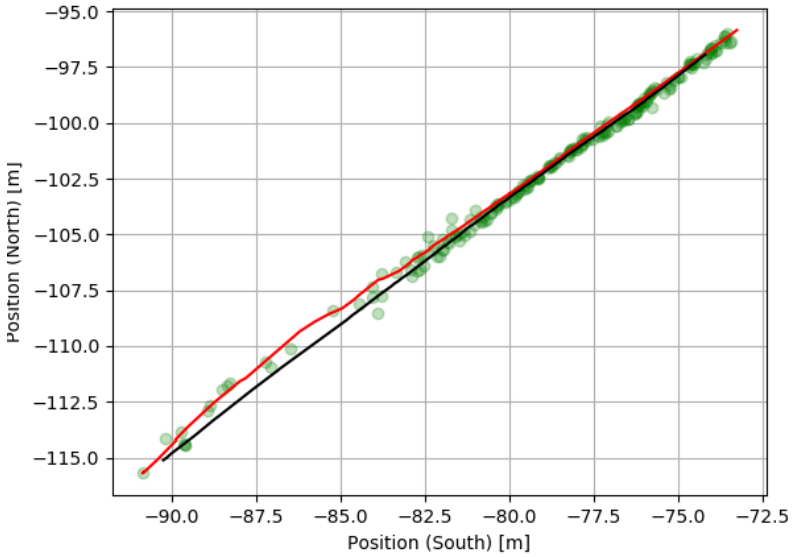

Figure 38: Track and GPS position in NED frame with 1500 ms delay

We also plot the tracking error with the same delays in Figure 39 and 40. As seen previously in Figure 36 it is clear that the average error increases as the track falls behind the GPS position. It can however also be noted that the increased uncertainty and spread of the sensor data at the start of the track when there are few measurements actually causes the error to decrease in this scenario.

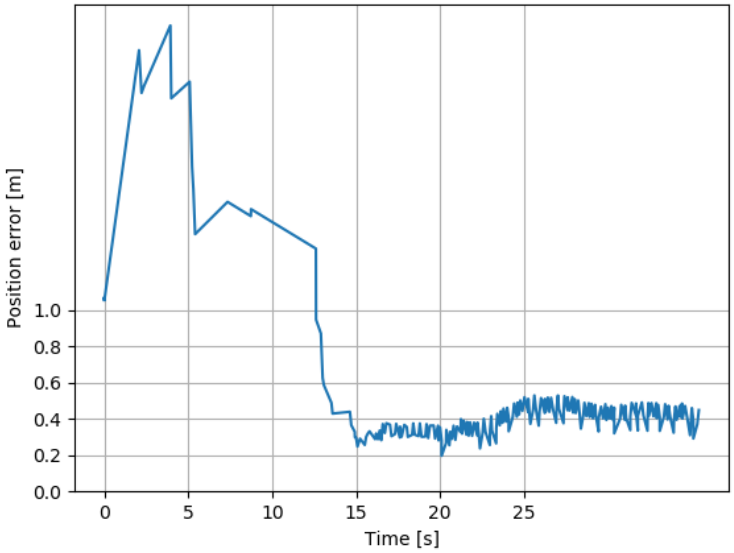

Figure 39: Track error with 150 ms delay

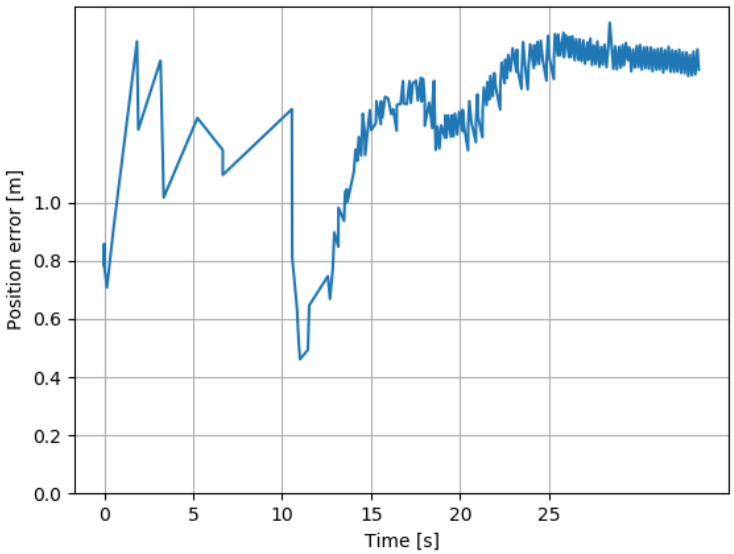

Figure 40: Track error with 1500 ms delay

With the second scenario (Figure 23) we wish to observe the consequences of delays in a constant turn rate scenario. Own ship is travelling in a straight line towards the target ship while the target ship makes a turn away from own ship. This scenario is also shortened down (see Figure 41) to avoid periods with no data or track on the target boat, or periods where the target boat does not follow the scenario we want to analyze. This scenario lasts for about 35 seconds and the turn starts about 13 seconds into the scenario when the target ship is about 32 meters away from own ship.

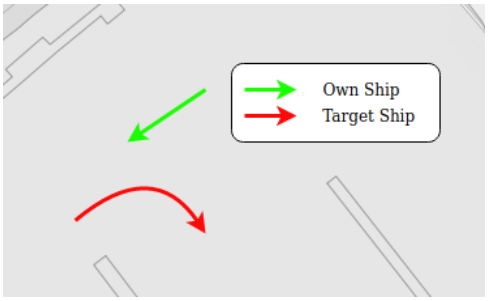

Figure 41: Track from scenario 2

The mean positional error due to a change in time delay is shown in Figure 42. We can see that unlike the error introduced by the time delay in the first scenario the error in this graph does not ramp up until there is around a 500 ms delay. Looking at Figure 43 and 44 we can see that the tracker overshoots slightly towards the end of the path with a low delay while it spends some time to converge with a high delay. When adding a slight delay in this scenario we get a middle-ground where the tracker converges faster to the path and does not overshoot.

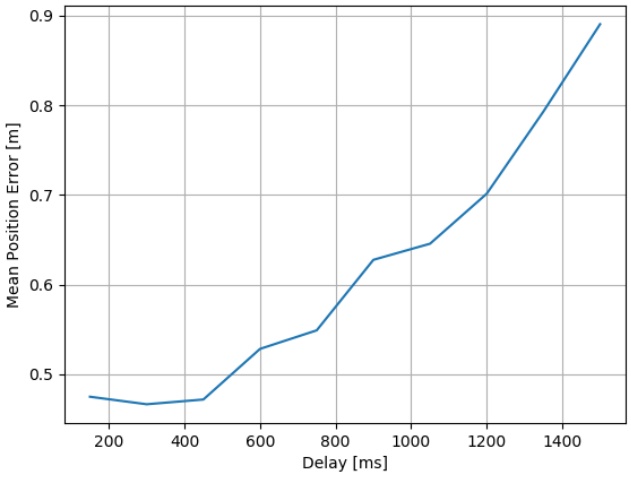

Figure 42: Mean positional error from time delay

Again we observe that the measurements scatter a bit more with a higher delay. There is an expected delay between the estimate and the true position of the target, and we see that the tracker overshoots the turn slightly more with an added delay.

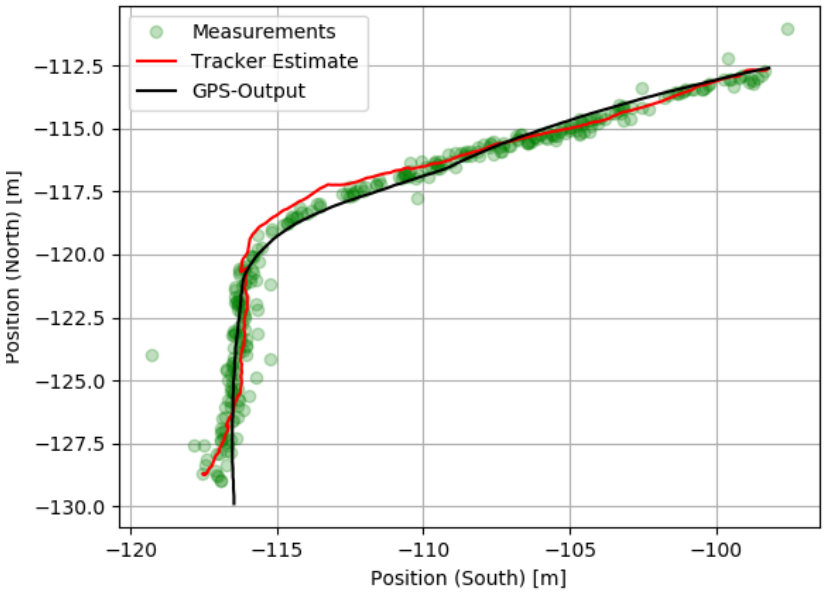

Figure 43: Track and GPS position in NED frame with 150 ms delay

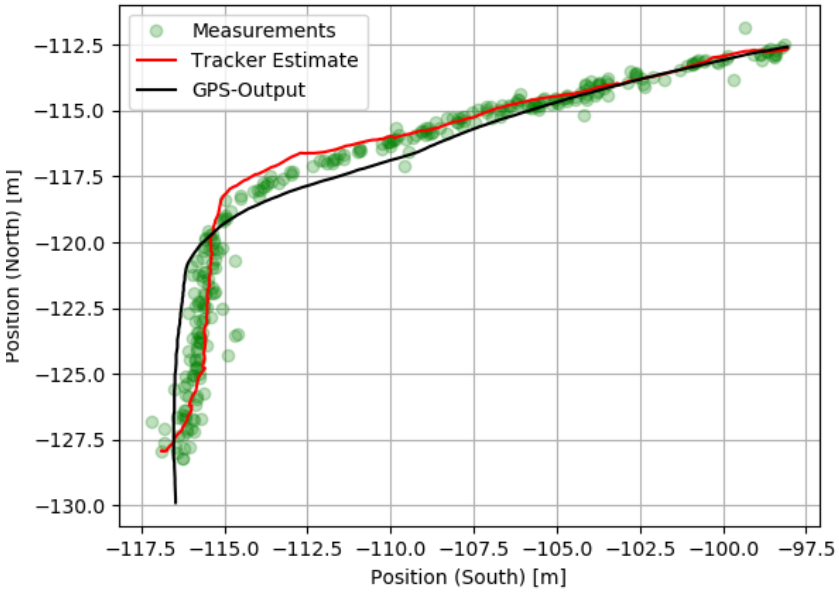

Figure 44: Track and GPS position in NED frame with 1500 ms delay

The track error in Figure 45 and 46 reflects what was said on the last page, that the error decreases towards the end of the path but increases at every other point when adding a delay to the measurements.

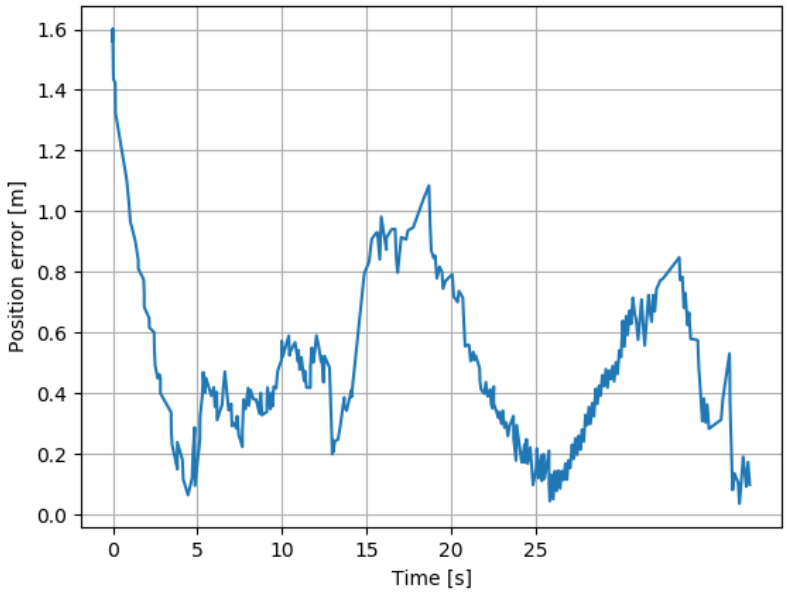

Figure 45: Track error with 150 ms delay

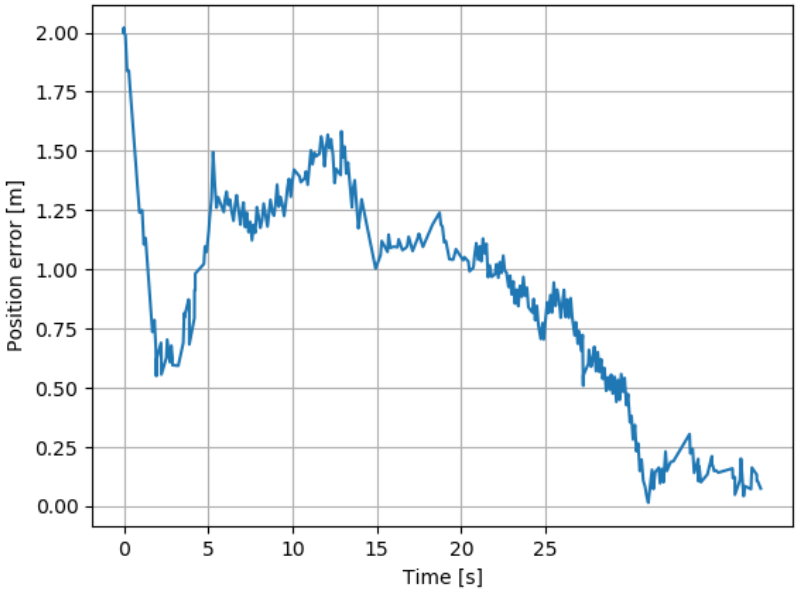

Figure 46: Track error with 1500 ms delay

The third scenario illustrated in Figure 24 is there to observe the consequences of delays when own ship is maneuvering while the target ship is travelling in a straight line. Also in this scenario we will only use a small portion of the collected data when the ship is within range of the LIDAR and we can get a good track. The movement of the boat during this period is illustrated in Figure 47, and the boats are around 25 meters apart.

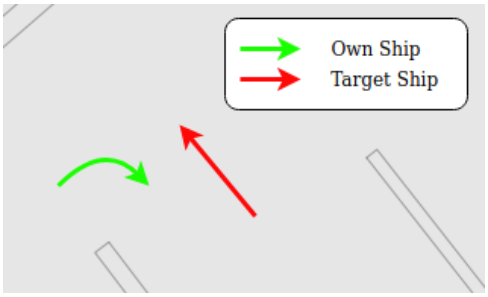

Figure 47: Track from scenario 3

The mean positional error due to a change in time delay is shown in Figure 48. Initially the error actually decreases when adding a time delay. Looking at the measurements in Figure 49 and 50 we can see that the measurements overshoots the GPS position with a low delay while it undershoots with a larger delay. As described in Chapter4.1 the Otter is around 2 m long and 1.2 m wide. In this scenario the LIDAR always senses the front of the target Otter while GPS measures the middle which causes us to get an offset.

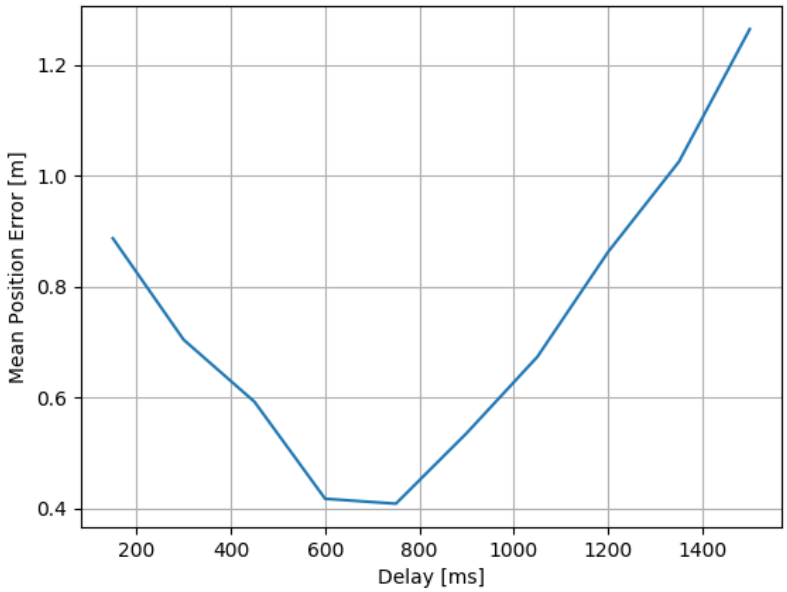

Figure 48: Mean positional error from time delay

Again we observe that the measurements are more scattered with a higher delay. This is especially visible on the lower right hand side of Figure 49 and 50.

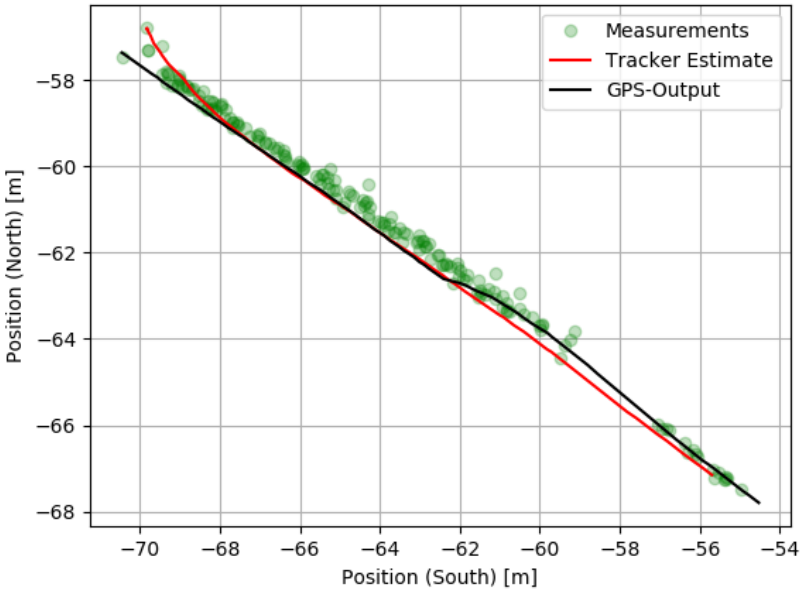

Figure 49: Track and GPS position in NED frame with 150 ms delay

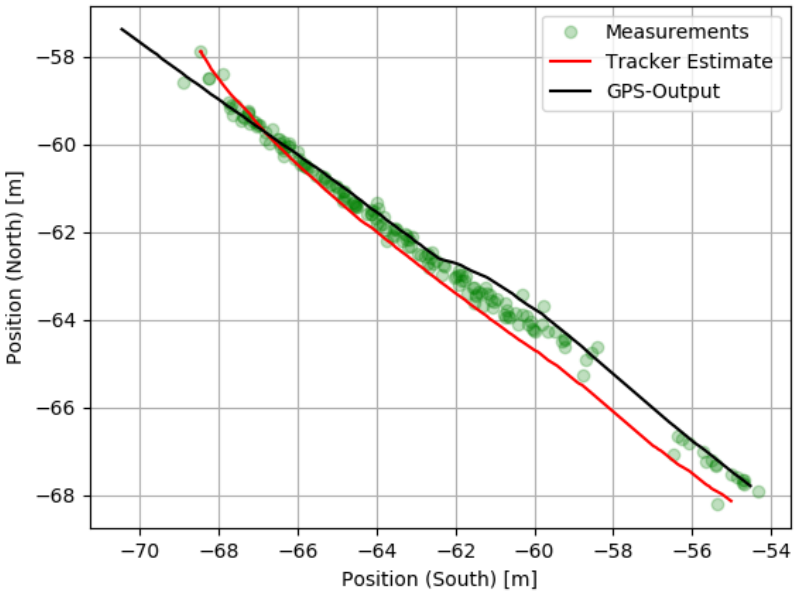

Figure 50: Track and GPS position in NED frame with 1500 ms delay

The track error with a lower delay shown in Figure 51 and 52 is consistently high when we have a lower delay on the LIDAR data until around two thirds into the scenario when it increases slightly. Looking at the two previous plots we can see that there is a slight spread towards the end because of a lack of measurements. With a higher delay we observe that the error is big when the LIDAR is looking at the front of the target but decreases towards the end when it is observing the back of the target.

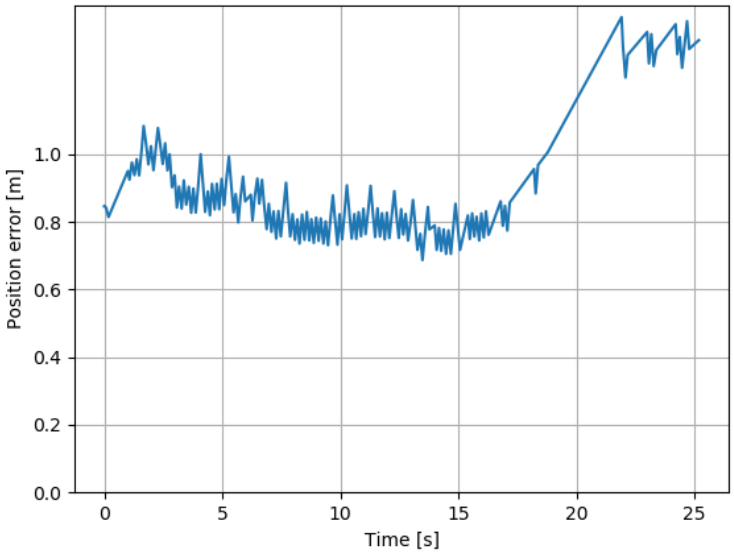

Figure 51: Track error with 150 ms delay

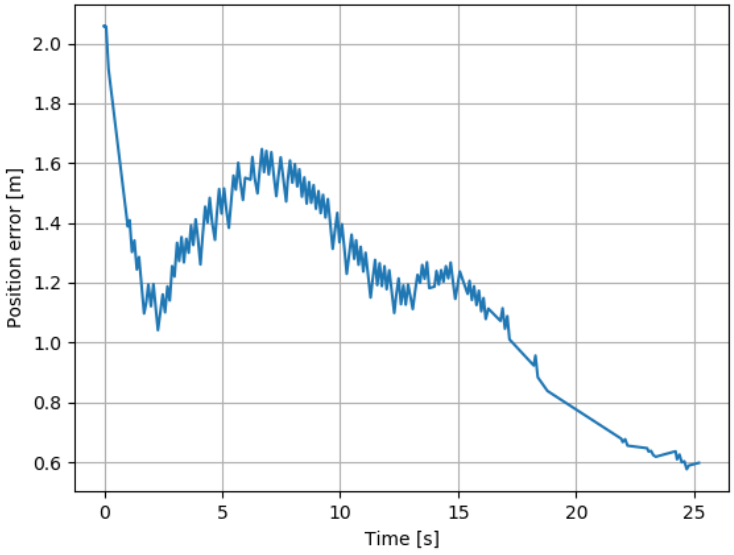

Figure 52: Track error with 1500 ms delay

6 Conclusion

When looking at the time differences in Chapter 5.2.1 we see that the difference between ToA using ROS and ToC using a dedicated synchronization board in a system like the Otter USV is around 170 ms. Especially ToA time stamping contain a lot of noise like what was shown in Figure 32, however this can be rather precisely approximated by a Gaussian distribution and added in a target tracking system as part of the uncertainty in a measurement model.

When attempting to obtain accurate sensor data the methods that were explained in chapter 2 seem to yield good results for an Otter USV. ToA and ToC timestamping aswell as PPS and NTP synchronization have all been implemented on the Otter, and although the less precise methods for time stamping degenerates the performance of a tracking pipeline, the results seem to be well within reason for the scope of this project. The scope of the experiments conducted were to simulate activities in in-land waterways and congested water environments with limited sea room for save navigation, and the highest error that was simulated was two meters at 30 meter range with a 1500 ms delay. This is nine times higher then the average delay that was measured on the sensor hub in Chapter5.2.1, and a significant increase in sensor data and a lot of post processing would be needed to even get close to it.

The size of the target Otter is also something that needs to be added into the calculation of the error. With its 2 meter length and 1.2 meter width an approximation of position of 2 meters is a very good result when the GPS position is relative to the center of the boat. The LIDAR also has a limited amount of data points, so some of the error will be attributed to the sensor not being able to capture the full size of the Otter.

With the results in this project it is no bold statement to conclude that the delays in ToA time stamping do not significantly decrease the performance of a target tracking pipeline. Due to the size of this project thesis the experiments were however conducted within some boundaries. The computers that gathered data for the scenarios in Chapter5 were only running drivers for the sensors and did no post-processing of the data, which would introduce further delays into the system. The LIDAR produces a limited point cloud and outputs little data, which is not necessarily identical to that used in a state of the art system.

6.1 Future Work

The current scenarios were all conducted during relatively calm conditions and should mirror the behavior of most boats in harbours and areas of safe navigation. Increasing the amount of movement on both own ship and target ship should cause a larger degeneration of the track. This is something that should be investigated before concluding that a ToA method of time stamping is good enough for most target tracking scenarios.

Analyzing the delays and noise in ToA timestamping caused by increased loads on the host computer and the computers network is also something that should be looked into. In this thesis the loads were relatively low as the host computer was only running the sensor drivers and logging the sensor data. The time stamps were however still noisy and even driving out of wi-fi range seemed to caused an increased delay of 20 ms.

Moving from target tracking to another field in sensor fusion where timing serves a crucial role is stereo vision setup of cameras. How does these different time stamping schemes impact the depth perception, how big of a quality increase does it give in the depth image going from a setup based on ToA to ToC.

References

[1] Lidar obstacle detection. https://github.com/enginBozkurt/LidarObstacleDetection.

[2] Robot operating system. https://www.ros.org/.

[3] Sentiboard documentation, 2020. https://gitlab.senti.no/senti/senti-doc/.

[4] Sigurd Albrektsen and Tor Johansen. User-configurable timing and navigation for uavs. Sensors, 18:2468, 07 2018.

[5] Marcin Bajer. Synchronization of current and voltage measumrents in modular motor diagnostic system. 2010.

[6] E Brekke, E Wilthil, Bjørn-Olav Eriksen, D Kufoalor, Øystein Helgesen, I Hagen, Morten Breivik, and Tor Johansen. The autosea project: Developing closed-loop target tracking and collision avoidance systems. Journal of Physics: Conference Series, 1357:012020, 10 2019.

[7] Teledyne Dalsa. Teledyne dalsa genie nano series manual, 2019.

[8] Katherine Ellis, Suneeta Godbole, Simon Marshall, Gert Lanckriet, John Staudenmayer, and Jacqueline Kerr. Identifying active travel behaviors in challenging environments using gps, accelerometers, and machine learning algorithms. 2014.

[9] GIS Geography. Trilateration vs triangluation - how gps receivers work, 2020. https://gisgeography.com/trilateration-triangulation-gps/.

[10] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision, 2003.

[11] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. Sensors and sensor fusion in autonomous vehicles, 2018.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. 2015.

[13] The Linux Man. Linux network time protocol (ntp). 2016. https://www.youtube.com/watch?v=EkQPkQb2D3g.

[14] Masterclock. Network timing technology - ntp vs. ptp. https://www.masterclock.com/support/library/network-timing-ntp-vs-ptp.

[15] MathWorks. What is camera calibration?, 2020. https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[16] E. Olson. A passive solution to the sensor synchronization problem. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1059–1064, 2010.

[17] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2015.

[18] Kongsberg Seatex. minimru - the compact reference unit, Dec 2020. https://www.kongsberg.com/globalassets/maritime/km-products/product-documents/datasheet_minimru.pdf.

[19] u blox. Zed-f9p datasheet, Dec 2020. https://www.u-blox.com/en/docs/UBX-17051259.

Attachments

Attachment 1: Target to NED transformation in ROS

```
#include "ros/ros.h"
#include <GeodeticPosition.h>
#include <sensor_msgs/NavSatFix.h>
#include <EcefPosition.h>
#include <vector>
#include <tf2_ros/transform_broadcaster.h>
#include <geometry_msgs/TransformStamped.h>
#include <tf2/LinearMath/Quaternion.h>
#include <math.h>

class TranslateTargetToNED
{
public:

        TranslateTargetToNED()
        {

                m_origin = GeodeticPosition(1.10723754069, 0.18152127588, 39.92);        //Kontoret
                m_origin_ecef = m_origin.toECEF();

                sub_ = n_.subscribe("/fix", 30, &TranslateTargetToNED::callback, this);

        }

        void callback(const sensor_msgs::NavSatFix& input)
        {

        GeodeticPosition targetPos(input.latitude / 180 * M_PI, input.longitude / 180 * M_PI, 0.0);
                EcefPosition targetPos_ecef = targetPos.toECEF();

                VectorNED posNED = targetPos_ecef.toNED(m_origin);

                static tf2_ros::TransformBroadcaster br;
                geometry_msgs::TransformStamped transformStamped;

                transformStamped.header.stamp = input.header.stamp;
                transformStamped.header.frame_id = m_tfParentFrame;
```

```
38                  transformStamped.child_frame_id = m_tfChildFrame;
39                  transformStamped.transform.translation.x = posNED.north();
40                  transformStamped.transform.translation.y = posNED.east();
41                  transformStamped.transform.translation.z = 0.0;
42
43                  ROS_WARN("GPS-Time: %8.8f --- ROSTime: %8.8f --- Delay (?):
                        %8.8f",
44                      transformStamped.header.stamp.toSec(),
45                      ros::Time::now().toSec(),
46                      ros::Time::now().toSec() - transformStamped.header.
                            stamp.toSec()
47                  );
48
49                  br.sendTransform(transformStamped);
50
51          }
52
53  private:
54
55          ros::NodeHandle n_;
56          ros::Subscriber sub_;
57
58          GeodeticPosition m_origin;
59          EcefPosition m_origin_ecef;
60
61          std::string m_tfParentFrame = "NED_munkholmen";
62          std::string m_tfChildFrame = "Target";
63
64  };
65
66  int main(int argc, char **argv)
67  {
68          ros::init(argc, argv, "TargetPosToNED");
69          TranslateTargetToNED NEDObject;
70          ros::spin();
71
72          return 0;
73  }
```

Attachment 2: Filtering of target point cloud

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <pcl_ros/point_cloud.h>
#include <pcl/point_types.h>
#include <pcl/filters/crop_box.h>
#include <boost/foreach.hpp>
#include <visualization_msgs/Marker.h>

class TargetFilter
{
public:

    TargetFilter()
    {
        sub   = nh.subscribe("/parser_pointcloud2", 1, &TargetFilter::
            callback, this);
        pub   = nh.advertise<sensor_msgs::PointCloud2>("/filtered_cloud",
            1);
        m_pub = nh.advertise<visualization_msgs::Marker> ("out",1);
    }

    void callback(const sensor_msgs::PointCloud2ConstPtr& cloud_msg)
    {
        tf::StampedTransform transform;
        try{
            listener.waitForTransform("/os1_lidar", "/target", ros::Time(0)
                , ros::Duration(6.0));
            listener.lookupTransform("/os1_lidar", "/target", ros::Time(0),
                transform);
        }
        catch (tf::TransformException ex){
            ROS_ERROR("%s",ex.what());
        }

        Eigen::Vector3f TargetTranslation(  transform.getOrigin().x(),
                                            transform.getOrigin().y(),
                                            transform.getOrigin().z());

        int input_size = cloud_msg->height * cloud_msg->width;

        pcl::PCLPointCloud2* cloud = new pcl::PCLPointCloud2;
        pcl::PCLPointCloud2ConstPtr cloudPtr(cloud);
```

```
39          pcl::PCLPointCloud2 cloud_filtered;
40
41          pcl_conversions::toPCL(*cloud_msg, *cloud);
42
43          pcl::CropBox<pcl::PCLPointCloud2> boxFilter;
44          boxFilter.setInputCloud(cloudPtr);
45          boxFilter.setMin(Eigen::Vector4f(-2.5, -2.5, -1.0, 0));
46          boxFilter.setMax(Eigen::Vector4f( 2.5,  2.5,  1.0, 0));
47          boxFilter.setTranslation(TargetTranslation);
48          boxFilter.filter(cloud_filtered);
49
50          sensor_msgs::PointCloud2 output;
51          pcl_conversions::fromPCL(cloud_filtered, output);
52
53          ROS_INFO("Incoming points: %8d    --> Outgoing points: %8d",
54                          input_size,
55                          output.height * output.width);
56
57          pub.publish (output);
58
59      }
60
61  private:
62
63      ros::NodeHandle nh;
64      ros::Subscriber sub;
65      ros::Publisher pub;
66      ros::Publisher m_pub;
67
68      tf::TransformListener listener;
69
70  };
71
72
73  int main (int argc, char** argv)
74  {
75
76      ros::init(argc, argv, "pcl_boxfilter");
77      TargetFilter TFilter;
78
79      ros::spin();
80
81  }
```