Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Jacob Hemstad Gimle

Sensor integration and timing in sensor fusion systems

Applied to airborne direct georeferencing

Master's thesis in Cybernetics and robotics
Supervisor: Torleiv Håland Bryne
July 2022

NTNU
Norwegian University of
Science and Technology

Jacob Hemstad Gimle

Sensor integration and timing in sensor fusion systems

Applied to airborne direct georeferencing

Master's thesis in Cybernetics and robotics
Supervisor: Torleiv Håland Bryne
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MSc THESIS DESCRIPTION SHEET

Name:	Jacob Hemstad Gimle
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Sensorintegrasjon og timing i sensorfusjonsystemer
Thesis title (English):	Sensor integration and timing in sensor fusion systems
Thesis subtitle (English):	Applied to airborne direct georeferencing

Thesis Description: Accurate and precise sensor synchronization is assumed implicitly in sensor fusion systems where sensors must either be

- synchronized (sensors provide samples at the same time) or
- that differences in sampling times are exactly known such that relative timing offsets can be compensated for.

Nevertheless, accurate and precise timing in sensors is in many aspects underappreciated in the sensor fusion community and timing errors such as latency, jitter and clock drift are not necessarily considered or modeled. In other cases, mean values for latency are used while jitter and drift are assumed to average to zero. This master thesis considers sensor integration, timing accuracy and precision for airborne direct georeferencing using inertial sensors, global navigation satellite system (GNSS) receivers and thermal camera interfaced with different communication protocols.

The following tasks for the MSc thesis should be considered.

1. Present the problem of direct georeferencing in the context for sensor interfacing and time
2. Perform a literature review on
    a. The different sensor communication protocols/interfaces and synchronization primitives in the context of the sensors provided to you.
    b. How timing errors in sensor fusion systems affect the sensor fusion accuracy.
    c. Timing standards and references (TAI, UTC, GPS time etc).
3. Use the SentiBoard to time and collect sensor data.
    a. Describe the SentiBoard message envelope.
    b. Assign the sensors measurement a global time based on this envelope.
4. Study the effect on sensor timing accuracy using GPS time provided by the GNSS receivers.
5. Study the potential georeferencing accuracy using the sensors provided (Syncline) using different timing strategies (timestamp, TOV, TOV + GNSS, TOA, TOA+GNSS).
6. Derive and implement a sensor synchronization algorithm applicable for real-time use based on the SentiBoard's features.
7. Discuss the quality of the sensor synchronization and timing.
8. Conclude your results and suggest further work.

Start date:	2022-01-31
Due date:	2022-08-01
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU

Preface

This thesis is carried out at the Department of Engineering Cybernetics and is submitted as part of the degree of Master of Science in Cybernetics and Robotics at the Norwegian University of Science and Technology.

I would like to thank my supervisor, Torleiv Håland Bryne, for his support, feedback for writing reports and help with finding relevant background literature. He also provided me with the datasets I did not generate myself and helped me implement the Kalman filter. I would also like to thank Kristoffer Gryte for helping me collect data from the UAV.

Abstract

This thesis describes the development of algorithms to perform time synchronization in sensor fusion systems. These algorithms can be used in real-time and the increase in time synchronization performance can result in a significant increase in sensor fusion accuracy. In this thesis the use case of georeferencing from an unmanned aerial vehicle (UAV) is used as an example to measure the effect of the increased time synchronization performance. The datasets used in this thesis comes from different UAVs using georeferencing equipment that were either stationary or in flight. A global navigation satellite system (GNSS) receiver was used to get an accurate time source which was used to synchronize the other sensors. Using this, four different algorithms, where one was a Kalman filter was implemented to perform time synchronization. The algorithms were evaluated based on the root mean square (RMS) of the synchronization error for the different sensors during the entire dataset and the subsequent effect this had on the georeferencing accuracy. The best performance was achieved using the Kalman filter, but the third algorithm using a moving average had almost similar results. The synchronization error of the inertial measurement unit (IMU), decreased by a factor of up to 25 000 when using the time synchronization algorithms compared to no active synchronization. This resulted in a decrease in positional georeferencing errors by a factor of up to 4000 with the system dynamics and sensor precision chosen in this thesis. These results were found by correcting the data after a flight had been performed, so the real-time viability and actual performance in a real flight has not been verified. However, the results show that reducing synchronization errors can have a big impact on sensor fusion accuracy if the system dynamics are fast and the sensors are accurate.

Sammendrag

Denne masteroppgaven beskriver utviklingen av algoritmer som blir brukt til å utføre tidssynkronisering i sensorfusjonssystemer. Disse algoritmene kan bli brukt i sanntid og økningen i tidssynkronisering kan føre til en signifikant økning i nøyaktigheten til sensorfusjonen. Denne masteroppgaven bruker georeferering fra droner (UAV) som et eksempel for å kvantifisere effekten av den forbedrede tidssynkroniseringen. Datasettene som blir brukt i denne masteroppgaven kommer fra ulike droner som bruker utstyr for å utføre georeferering som enten har vært stasjonære eller har flydd. En satellittbasert navigasjonssystem (GNSS) mottaker ble brukt for å få en nøyaktig tidskilde som så ble brukt for å synkronisere de andre sensorene. Ved å bruke dette som utgangspunkt ble fire ulike algoritmer utviklet, hvor en av disse var et Kalmanfilter. Algoritmene ble evaluert ved å bruke det kvadratiske gjennomsnittet (RMS) av synkroniseringsfeilen på alle sensorene over hele datasettet. Deretter ble det sett på effekten dette hadde på nøyaktigheten til georefereringen. Det beste resultatet kom fra Kalmanfilteret, men den tredje algoritmen som brukte et glidene gjennomsnitt hadde nesten like bra resultat. Synkroniseringsfeilen til bevegelsessensoren (IMU), ble redusert med en faktor på opp til 25 000 når tidssynkroniseringsalgoritmene ble brukt, sammenlignet med ingen aktiv tidssynkronisering. Dette resulterte i en reduksjon i posisjonsfeilen til georefereringen med en faktor på opp til 4000, basert på den nøyaktigheten som var valgt for sensorene og hvordan dynamikken til systemet var bestemt i denne oppgaven. Disse resultatene ble funnet ved å bruke data som ble samlet inn etter en flyvning var fullført. Dermed så har ikke systemet blitt testet i sanntid og den faktiske ytelsen i sanntid mens dronen er i luften kan avvike fra resultatene som er presentert her. Resultatene viser fremdeles at det å redusere synkroniseringsfeilen kan ha en stor påvirkning på nøyaktigheten til sensorfusjonen, spesielt hvis dronen beveger seg fort og sensorene er nøyaktige.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abbreviations

DMA	=	Direct memory access
GNSS	=	Global navigation satellite system
GPS	=	Global positioning system
I2C	=	Inter-Integrated Circuit
IC	=	Input capture
IMU	=	Intertial measurement unit
JRCC	=	Joint rescue coordination centres
LAN	=	Local area network
NED	=	North east down
NTP	=	Network time protocol
OS	=	Operating system
PPS	=	Pulse per second
PTP	=	Precision time protocol
RMS	=	Root mean square
RS-232	=	Recommended Standard 232
RS-422	=	Recommended Standard 422
SAR	=	Search and rescue
SBC	=	Single board computer
SenTiBoard	=	Sensor timing board
SPI	=	Serial peripheral interface
SRU	=	Search and rescue units
TAI	=	International atomic time
TCG	=	Geocentric coordinate time
TOA	=	Time-of-arrival
TOT	=	Time-of-transport
TOV	=	Time-of-validity
TOW	=	Time of week
TT	=	Terrestial time
U(S)ART	=	Universal (synchronous) and asynchronous receiver-transmitter
UAV	=	Unmanned aerial vehicle
UT	=	Universal time
UTC	=	Universal coordinated time
WAN	=	Wide area network

1

Introduction

More than 2500 maritime incidents are handled by the Norwegian Joint Rescue Coordination Centres (JRCC) each year [1]. Many of these involve search and rescue (SAR) which is something that can be difficult on the open sea. With high winds and rough seas, detecting small vessels or people in the water, can be challenging as the object being searched for will continually move. Search and rescue units (SRU) must therefore use elaborate models for estimating the search area and it can be time consuming to exhaustively search a given area [2].

Unmanned aerial vehicles (UAVs) can be used for the search phase of the SAR mission to reduce the time required to search for the object. UAVs can cover a large area in a short amount of time and are therefore ideal for monitoring areas such as the sea surface. If an optical sensor is mounted to the UAV, it can be used to take images which can be used to detect objects. These objects can then be related from the sensor frame to a geographic coordinate system. This process is called georeferencing. Direct georeferencing is georeferencing that happens onboard the UAV in real-time, instead of post-mission.

If an SRU vessel is assisted by UAVs using direct georeferencing, it could get Global Positioning System (GPS) coordinates of objects detected in the water and as such focus on the rescue phase while the UAVs handle the search phase.

Performing direct georeferencing with a UAV is a complicated problem and the most accurate solutions require reference points on the ground, something which is not available in the open sea. The accuracy of the georeferencing can be more dependent on the calibration of misalignment errors and the time synchronization between sensors than the quality of the navigation sensors [3]. This thesis will focus on errors due to a lack of time synchronization.

The importance of time synchronization increases if the UAV is far from the surface, the UAV has a high velocity, the UAVs attitude changes quickly, the object it is tracking is moving quickly or the surface is changing rapidly. All these factors will be present if the UAV is supposed to search a large area quickly in rough seas. The reason for this is that the georeferencing relies on the position and attitude of the camera at the exact time that the image is captured. If the UAV has a significant attitude motion when the image is

captured a small time synchronization error can lead to a large error in the georeferenced position.

In order to have small time synchronization errors, high timestamping accuracy is also necessary. To achieve this, dedicated hardware for sensor timing is often used. The hardware used in this thesis is the Sensor Timing Board (SenTiBoard) [4]. In addition to giving coordinates GPS satellites also provide time information which is highly accurate. This timing information can be used in order to synchronize the other timing sources in the system to the global GPS clock.

1.1 Previous work

This thesis combines the accurate timestamping hardware of the SenTiBoard [4] to get very precise timestamps and the Syncline model [5] to evaluate the impact of time synchronization on sensor fusion accuracy.

1.2 Scope of work

This thesis aims to implement different time synchronization strategies and measure their timing accuracy. Doing this accomplishes multiple objectives. Firstly, it will quantify the effect of the different time synchronization methods. Secondly, the increase in timing accuracy can be used to see what impact increased time synchronization has on georeferencing accuracy. The main objectives of the thesis therefore consist of the following tasks:

- Literature review of sensor time synchronization methods, timing errors in sensor fusion systems and timing standards.
- Describing the current SentiBoard system and its limitations.
- Collecting data and measuring the timing accuracy of the different timestamping methods.
- Implementing time synchronization algorithms for real-time use.
- Evaluation of the time synchronization performance of the real-time algorithms and its effects on the georeferencing accuracy.

1.3 Outline of the thesis

Following the introduction, the theoretical background for this thesis which consists of relevant sensors, georeferencing, time standards and time synchronization will be presented in Chapter 2. Next, in Chapter 3 the UAV and the SenTiBoard is presented. After that the current timestamping performance of the system without using time synchronization is presented in Chapter 4. In Chapter 5 the different time synchronization algorithms that were implemented to reduce the time synchronization errors and their results are presented.

In Chapter 6 an example of the effect the synchronization errors has on georeferencing errors and suggestions for future work is discussed. Finally, Chapter 7 concludes the work presented.

2

Preliminaries

2.1 Sensors

2.1.1 Inertial sensors

Gyroscope

Gyroscopes can measure angular rate by utilizing the Coriolis force. Two masses that are vibrating linearly in opposite directions on a plane will be subjected to opposite forces of the same magnitude if a rotation is applied perpendicular to the vibration axis due to the Coriolis force. The implementation of the vibration and the measurement of the force varies based on the specific implementation chosen. Usually the difference in displacement of the two masses is converted into a difference in capacitance which is then used to find the angular rate for that axis [6, pp. 142-149].

Accelerometer

Accelerometers can be used to measure linear motion without a fixed reference and measure acceleration relative to free fall. Typically, a mechanical accelerometer is designed such that an electronic circuit will sense a small amount of motion of a mass. A force can then be applied to make the mass return to its neutral position. This can be due to electrostatic effects or for example an electromagnet. The displacement of the mass is then often measured as a change in capacitance or electrical charge if piezoelectric materials are used. This is then used to find the acceleration [6, pp. 139-142].

Inertial measurement unit

An inertial measurement unit (IMU) is an assembly of sensors that usually consists of a gyroscope and an accelerometer. An IMU usually has one or more gyroscopes and accelerometers that can measure the acceleration and angular velocity. A typical configuration has one gyroscope and one accelerometer per axis. Using these measurements,

the attitude, position and velocity can be calculated using an inertial navigation system. An IMU typically contains a hardware interface, measurement calibration software, error handling and the possibility to retrieve computed measurements. Due to errors in the measurements from the gyroscope and the accelerometer that accumulate over time a Kalman filter is often used to filter and correct the measurement. In many applications using an IMU by itself does not provide sufficient accuracy and therefore sensor fusion is needed in order to obtain better results [6, pp. 149-151].

2.1.2 Global navigation satellite system

A global navigation satellite system (GNSS) is a system that uses satellites in orbit around Earth to get the position of the receiver and the current time. A GNSS consists of three segments, the satellites in space, the control system and the GNSS receivers. The satellites in orbit broadcasts signals down to Earth. The satellites have atomic clocks to maintain a stable time reference. The satellites broadcast data that allows the GNSS receivers to determine the time that the received messages were transmitted and the position of the satellites, by using information about their orbits. The control system consists of ground stations that have precise locations and synchronized clocks. This allows them to calibrate the orbits and the satellite clocks. The GNSS receiver processes the data from the satellites and outputs position, velocity and time [6, pp. 299-311].

2.2 Airborne georeferencing

Georeferencing is the process of relating images from an optical sensors internal coordinate system to a geographic coordinate system [7]. North east down (NED) is a geographical coordinate system that represents position using three axis with its origin fixed at an aircraft or spacecraft. The north and east axis points along a longitude and along a latitude curve respectively. The down axis points towards the Earth's surface. An issue when performing georeferencing is that NED coordinates have three degrees of freedom, while pixel coordinates only have two degrees of freedom. One way of solving this problem is to assume that all the pixels in the image are in a single plane. This is called the flat-earth assumption and is necessary if no elevation map is available. This is not a problem when georeferencing objects in the sea since the sea is flat. Using this assumption, the down coordinate can be set to 0. Georeferencing of an object by UAV can then be performed by first detecting the object using image processing, then using the position and attitude of the UAV in addition to the angle of the camera to calculate the north and east coordinate of the object [3]. This can be done in real time on the UAV if direct georeferencing is needed.

2.3 Timing

2.3.1 Time standards

There are many different time standards that are used for different purposes. Some of them are theoretical ideals while others are physically realized. They are all related to each other and conversions between them are mostly exact.

Geocentric Coordinate Time (TCG) is a theoretical concept and is based on a clock at rest at the center of the Earth, following the Earths movement. It is used as the time component of spacetime coordinates centered at Earth. Terrestrial time (TT) is a linear scaling of TCG and gives the time on the surface of the Earth at mean sea level [8]. International Atomic Time (TAI) is a physical realization of TT. This is achieved by computing the weighted average of over 450 atomic clocks in meteorological conditions in laboratories around the world [9].

Universal Time (UT) has several different versions with UT1 being the principal form. UT1 is based on the Earth's rotation as it was in the mid-19th century such that one mean solar day is 86400 seconds. The rotation of the Earth is slowly decreasing, but also varies unpredictably and therefore has to be observed [10]. To avoid the mean solar day from becoming desynchronized with the mean sun, Universal Coordinated Time (UTC) was implemented. UTC is based on TAI, but is offset by an integer amount of seconds to always be within 0.9 seconds of UT1. To achieve this leap seconds are added or removed at the end of June or December to keep it synchronized. UTC is the time standard that is used for civil time and most countries have a time system that differs from UTC by an integer number of hours [8].

Global Positioning System (GPS) satellites use GPS time which was set to be equal to UTC in 1980. GPS time is continuous and therefore does not implement leap seconds, but each GPS satellite has an atomic clock that is synchronized by ground stations to be within 25 ns of UTC modulo 1 second [11]. GNSS receivers using GPS time compensates for the difference between GPS time and UTC and can be used as a reliable source for correct time in UTC [12, p. 18]. Time is stored within the GPS satellites as an integer week number and time of week (TOW) in seconds. This is easier for digital systems to handle, but it is converted to the conventional form using years, months, days, hours, minutes and seconds at external interfaces, but it is still possible to access the TOW data [13].

2.3.2 Timestamping

When a sensor acquires a measurement and when the measurement arrives at the processing unit is often at different points in time. The different points in time that are usually of interest are as follows. Time-of-validity (TOV) is the time at which the measurement was taken by the sensor and is the point in time that the measurement is considered valid. Time-of-transport (TOT) is the time that the first part of the message from the sensor is received by the receiver. Time-of-arrival (TOA) is when the full message from the sensor has been received, which indicates when the message can be passed on to the systems that will use the measurement [4].

The best of these timestamps is the TOV since that gives us the actual sensor measurement at the correct time. This is however not always possible, as getting the TOV requires both dedicated timestamping hardware and sensors that implement the TOV functionality. The timestamping hardware needs a separate timer and precise input capture with interrupts in order to get an accurate timestamp. It also needs a way to associate a TOV signal with the measurement proceeding it [5]. The sensors need to have a separate data output that is only used to communicate when the data is valid in the form of a short flank. If this is not available, then usually TOA is the only option. This does not need any special support from the sensors or the hardware receiving it, but the timestamp will vary from the

sensors side based on the processing time and the size of the message. The timestamp will also vary on the receiving side based on the hardware and the software running on it.

2.3.3 Clock synchronization

If a system uses multiple sensors which need to be synchronized with each other there needs to be some form of synchronization primitive that is used. It does not matter if the timestamps are accurate if the clocks of the different sensors drift at different speeds, therefore causing the clocks to not be synchronized anymore.

One way of solving this problem is to use a pulse per second (PPS) signal from a very accurate time source [12, p. 22]. The PPS is an electrical signal that accurately repeats a rising or falling edge each second. Sensors that accept PPS signals expect a signal each second and compares this with its internal clock to synchronize its clock with the one sending the PPS signal. A good source for a PPS signal is a GNSS receiver as they have a high level of accuracy as presented in Section 2.3.1.

Another way to solve this problem is to use network synchronization. This is possible if the sensor has a microprocessor which can use the TCP/IP stack and that can support the IEEE 1588 Precision Time Protocol (PTP) or the Network Time Protocol (NTP). These protocols can be used to synchronize the sensor clock. NTP and PTP work based on the same principle. Multiple devices are connected together and form a hierarchy of time sources with the most accurate clock on top. In NTP a NTP client will receive timestamps from all the available sources in the network and decide how to combine these measurements to adjust the clock. In PTP each PTP client has a single master, and there is only one grandmaster clock which synchronizes all the other clocks. PTP is the newer than NTP and is more accurate, but PTP can only be used in local area networks (LANs) while NTP can be used in both wide area networks (WANs) and LANs [12, p. 23].

Another method of synchronization is needed if a sensor does not send timing information as an output, but can get timing information as an input. This can be the case in for example cameras where the data acquired is much larger than the other sensors and therefore needs to be stored separately. In this case the sensors are triggered by a signal from a synchronizing unit. The synchronizing unit generates a trigger flank and associates the next measurement from the sensor with the time the flank was generated.

2.4 Syncline

When using sensor fusion methods, they are usually developed and evaluated on data sets that have synchronous measurements or perfect time information. In reality the sensors are asynchronous and need to use some form of synchronization. The importance of the synchronization precision on the accuracy of the sensor fusion is dependent on the dynamics of the system. The Syncline model is a model that quantifies the effects of sensor accuracy compared to time synchronization precision in a visual way [5]. Synchronization errors during sensor measurement leads to inaccurate timestamping as a sensor measurement believed to be valid at time t is actually valid at time $t + \epsilon$. The consequence of this synchronization error is dependent on the dynamics of the system as that affects how much the measured states have changed between time t and $t + \epsilon$. Based on this the Syncline is

Table 2.1: Definitions for the variables that are used in the Syncline model.

τ	Mean value of the synchronization error
v_{max}	Norm of the expected maximum linear velocity
ω_{max}	Norm of the expected maximum angular velocity
d	Typical distance between the object and the sensor
σ_p	Standard deviation of the estimated position
σ_r	Standard deviation of the estimated ranging
σ_Θ	Standard deviation of the estimated attitude
σ_u	Standard deviation of the estimated bearing

defined as:

$$\delta^*_{sync}(\tau) := v_{max}\tau + d * \omega_{max}\tau \tag{2.1}$$

$$\delta^*_{sensor} := \sigma_p + \sigma_r + (\sigma_\Theta + \sigma_u) * d \tag{2.2}$$

$$\delta_{syncline}(\tau) := \delta^*_{sync}(\tau) + \delta^*_{sensor} \tag{2.3}$$

where Table 2.1 shows the definition of the variables used in the syncline model.

Figure 2.1 shows how to read the Syncline plot. The vertical axis shows the estimation accuracy, which is the inverse of the sensor fusion error while the horizontal axis shows the synchronization accuracy which is the inverse of the synchronization error. The roof is the maximum obtainable accuracy of the sensor fusion system and is limited by the sensor accuracy. The slope shows how the accuracy of the sensor fusion system increases as the timing synchronization improves. The critical synchronization precision is given as τ_{crit} and is defined by:

$$\delta^*_{sync}(\tau_{crit}) = \delta^*_{sensor} \tag{2.4}$$

If the mean value of the synchronization error is less than τ_{crit} then the overall system accuracy is sensor-bound which means that the sensor accuracy is the limiting factor. If the mean value of the synchronization error is more than τ_{crit} then the system is sync-bound, and the timing synchronization is the limiting factor for the systems accuracy.

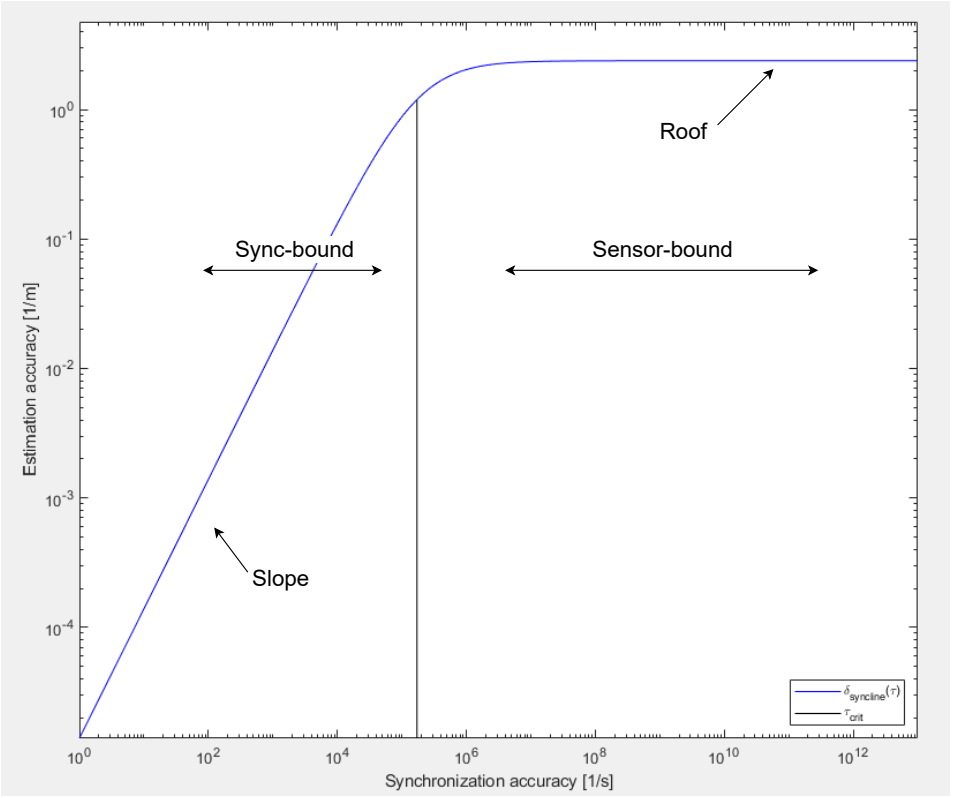


Figure 2.1: A plot showing the different regions and boundaries of the Syncline.

2.5 Kalman filter

The Kalman filter is an optimal linear estimator that can be used for real-time applications [14]. Given a general discrete linear state-space model:

$$x_n = \Phi x_{n-1} + G\omega_n \tag{2.5}$$
$$y_n = Hx_n + v_n \tag{2.6}$$

where x and y are the state vector and measurement vector respectively, Φ and H are the state transition and measurement matrix respectively. G is the noise matrix and ω and v are zero mean Gaussian noise. The Kalman filter consists of two steps, the prediction step, and the update step. The prediction step is as follows:

$$\hat{x}_n = \Phi \hat{x}_{n-1} \tag{2.7}$$
$$P_n = \Phi P_{n-1} \Phi^T + Q \tag{2.8}$$

where $\hat{x}$ is the state estimation, P is the error covariance matrix and Q is the process noise covariance matrix. The update step is as follows:

$$K_n = P_n H_n^T (H_n P_n H_n^T + R_n)^{-1} \tag{2.9}$$
$$\hat{x}_n = \hat{x}_n + K_n(y_n - H_n \hat{x}_n) \tag{2.10}$$
$$P_n = P_n - K_n H_n P_n \tag{2.11}$$

where K is the Kalman filter gain and R is the measurement noise covariance matrix. The Kalman filter works by predicting what the next state will be based on a model of the system. When a new measurement is available it is compared with the prediction and used to adjust the new predictions. The adjustments are done using a weighted average where measurements with a higher certainty is given a higher weight.

3

The current system

3.1 UAV

The UAV used for this thesis is a multirotor and can be seen in Figure 3.1. The relevant sensors for this thesis that are used in the UAV can be seen in Table 3.1. These sensors are connected to the SenTiBoard [4] which is used for accurate timestamping. The SenTiBoard is connected to the single board computer (SBC) Odroid XU4 [15]. The data from all the sensors except the camera flows through the SenTiBoard to get timestamped before going to the SBC. The camera is triggered by the SenTiBoard, but the data flows directly to the SBC. Figure 3.2 shows an overview of the system. The SBC then uses the data from the sensors in a sensor fusion algorithm. The SBC is also connected to the autopilot system.

3.2 SentiBoard

The SenTiBoard is a dedicated hardware timing solution that is designed to be configurable. The SenTiboard uses a PIC32MZ microprocessor which runs at 200 MHz and has a 32-bit timing counter that runs at 100 MHz which is used to reference the sensor readings. The SenTiBoard can communicate with desktop or embedded computers by using USB. The main functionality of the SenTiBoard is to record accurate timestamps. This is done by using 9 Input capture (IC) pins which can capture the TOV, TOT and TOA from

Table 3.1: The sensors used in the UAV.

Sensor type	Sensor name
IMU	ADIS16490 [16]
IMU	Kongsberg MiniMRU 30 [17]
GNSS receiver	u-blox ZED-F9P [18]
Thermal camera	Genie Nano C4040 [19]

Figure 3.1: UAV with the sensors and computing payload in the white box.

9 sensors simultaneously. The IC pins trigger an interrupt subroutine (ISR) when they detect an edge, and in addition to this the value of the clock timer is stored. The SenTiBoard supports multiple protocols including U(S)ART, RS-232, RS-422, SPI and I2C. The SentiBoard can also trigger sensors by generating pulses. This can be used to sample multiple sensors simultaneously or to activate sensors like a camera. This can be useful as for example cameras can send more data than the microprocessor is able to process. The camera will in this instance be synced either by controlling when it takes a picture by generating a pulse, or by registering a separate signal to get the TOV. Finally, the SenTiBoard can also control the power of the sensors that are connected to turn them on or off if necessary. In order to get a TOV timestamp from a sensor, the sensor needs to support this by having a dedicated output which toggles when the sensor data is valid.

The SenTiBoard uses a common data envelope for the data it receives from each sensor. The data is wrapped in a envelope format which is depicted in Figure 3.3. The format starts with a 8-byte header which includes 2 bytes for the header checksum (CS_H). The header consists of 2 bytes for the syncword which checks that the data is the start of a packet that comes from a configured sensor, 2 bytes for the length of the data, 1 byte for the id of the sensor port and 1 byte for the id of the revision of the envelope format. Following this is an optional timestamp that is only available if connected to an external computer. After that follows the main timestamps TOV, TOT and TOA. After the data is a package checksum (CS_PKG) and padding bytes. The padding bytes are needed to align the data for the direct memory access (DMA).

3.2.1 Sentiboard timing

Using the 100 MHz clock gives a resolution of 10 nanoseconds, but the clock will drift over time and the accuracy of the timestamps will reduce over time. In order to correct the drift a reference clock can be used. This reference clock needs to be more accurate than the

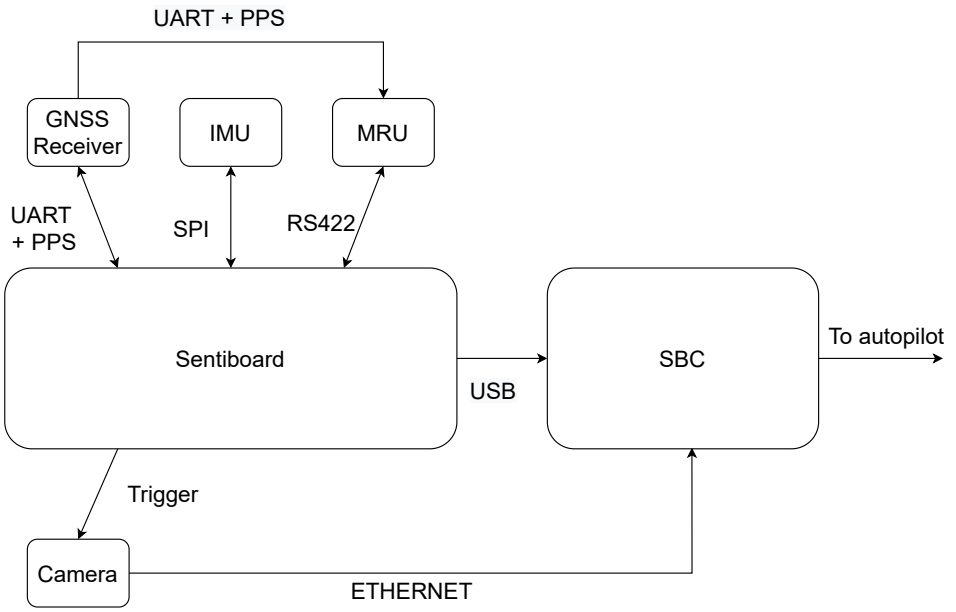


Figure 3.2: Overview of the system

clock it is correcting otherwise the drift will only get worse. Since a highly accurate clock is expensive and usually heavy a good alternative is to use a GNSS receiver which receives GPS time. As mentioned in Section 2.3.1 GPS time is continuously synchronized to be within 25 ns of UTC modulo 1 second and can therefore be used as a highly accurate reference clock to correct drift. If a sufficiently accurate GPS time signal can be retrieved from a GNSS receiver, this can be used to remove the clock drift of the SenTiBoard clock. The accuracy of the GNSS receiver is presented in Section 4.5 while the corrected timestamps can be seen in Chapter 5.

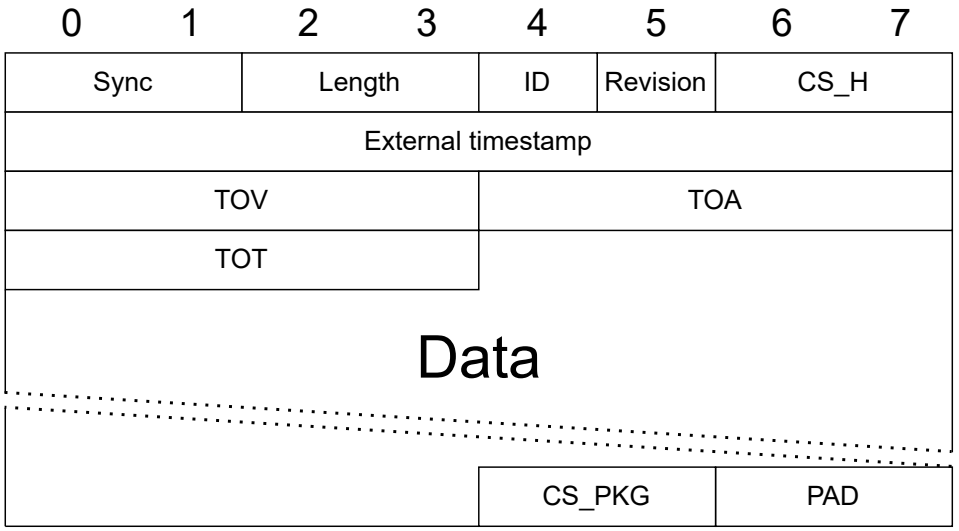


Figure 3.3: The contents of the data envelope for the SenTiBoard. The numbers at the top is the byte count.

4

Current performance

4.1 Collection of data

The data that is evaluated in this thesis is summarized in Table 4.1 and comes from three different sources:

- The first dataset is data collected by me where the payload of the system was connected to a battery and the data went from the SenTiBoard and to a computer running a Linux operating system (OS). This dataset uses the ADIS IMU. This test was done outside where the equipment was placed far away from buildings such that it had good access to multiple GNSS satellites. Due to the weather conditions a rain cover was placed over the electronics, but the GNSS antenna was in a raised position and was uncovered. The test lasted for one hour and the system was stationary during the test. Figure 4.1 shows what the test setup looks like at the test site. This data is what was used to make the algorithms presented in Chapter 5.
- The second dataset is from a few different flights using a UAV that also uses the SenTiBoard with the same GNSS receiver, but with a different IMU. The IMU used in this dataset is a Sensonor STIM300 [20]. This data is used to see general trends in the performance of the SenTiBoard and the IMUs. It is also used as a control to check that the algorithms developed using the first dataset are also applicable to a UAV in flight and that they work regardless of which sensor is used.
- The last dataset is from a flight using the same equipment as the test I performed myself, which means that it also uses the ADIS IMU. Due to an error this dataset has a very high amount of packet loss. This dataset will be used to check if the algorithms can handle packet loss. Packet loss from the GNSS receiver can for example happen if the UAV flies into a tunnel.

If data from the ADIS IMU or MRU is mentioned it is always from dataset 1 except if it is explicitly mentioned that it's from the dataset with packet loss, which is dataset 3. If data from the STIM IMU is mentioned, it is always from dataset 2.

Table 4.1: The sensors used for each dataset and the sample rate they were configured with.

Dataset	ADIS IMU sample rate	STIM IMU sample rate	MRU sample rate
Dataset 1	1062.5 Hz	no STIM IMU	200 Hz
Dataset 2	no ADIS IMU	200 Hz	NO MRU
Dataset 3	4250 Hz	no STIM IMU	not used

Figure 4.1: GNSS antennas, sensor suite and laptop with rain cover on trolley at test site.

4.2 Synchronization error calculation

The data collected will be evaluated based on the accumulated synchronization error over time. The accumulative effect of the synchronization error over time is calculated using [4, Eq. (1)] generalized to any sampling frequency:

$$e(n) = t(n) - \left(t(0) + \frac{n}{sensor\_freq} \right) \tag{4.1}$$

where $t(n)$ is the value of the timestamp at sample n for the current timestamping method, n is the sample number and $sensor\_freq$ is the nominal sampling frequency of the sensor. This equation measures how much each timestamp differs from the expected time at each sample given a constant sample rate. Therefore, this shows how different the timestamp is compared to a perfect clock showing the true time. As will be shown in Section 4.5, the accuracy of the timestamps from the GNSS receiver is very high and the other sensors timestamps will be synchronized to the GNSS receivers timestamps. Therefore, the synchronization error is calculated as how far off a given timestamp is to the actual time, which is approximately equal to the timestamps from the GNSS receiver. This approximation is sufficient given the level of accuracy achieved with the algorithms presented in Chapter 5. This synchronization error consists of two parts, the error due to the timestamping not

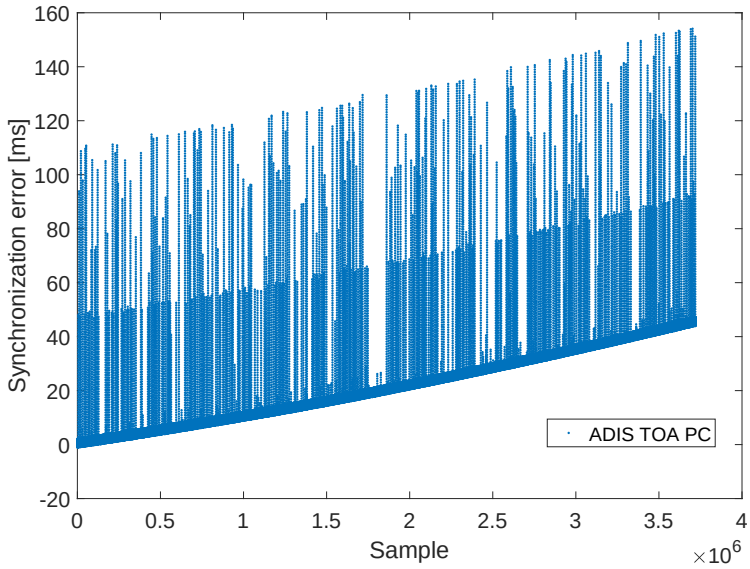


Figure 4.2: Synchronization error when using TOA for ADIS IMU connected to the PC running Linux.

being accurate and the error due to clock drift.

4.3 Timestamping accuracy

The most basic form of timestamping is timestamping on arrival which gives us a timestamp using TOA as presented in Section 2.3.2. The data collected is timestamped on arrival with three different methods. In all cases this data is passed through the SenTiBoard. The first and worst method is timestamping on arrival when the data arrives at the PC running the Linux OS. This OS runs many other tasks and as such the data can arrive at a time where the processor is busy with another task. This results in timestamps that vary wildly and are not very accurate. Figure 4.2 shows what this looks like for the ADIS IMU from the dataset without packet loss. This shows that the majority of the timestamps are close to each other and that they slowly drift from 0 to 40 ms of synchronization error. There are, however, quite a few outliers that have up to 100 ms worse synchronization error than the main cluster. This is likely due to the processor being busy with other tasks when the data arrives. The main drift from 0 to 40 ms is likely due to the drift of both the PC clock and the ADIS IMU clock which will be covered in more detail in Section 4.4.

To get improved accuracy, the timestamping on arrival can be done on an SBC running real-time software. The only dataset which has data collected in this way is the dataset with high packet loss. The synchronization error for this dataset using timestamping on arrival can be seen in Figure 4.3. The top plot shows the synchronization error during the entire runtime. Due to the high amount of packet loss, it is difficult to see how accurate it is. The synchronization error becomes very high due to the way the synchronization error

is calculated. As seen in (4.1) the synchronization error is calculated based on a constant sample rate, which is not the case when packets are lost. The result of this is that when a GNSS packet is lost, the synchronization error jumps by one second. A solution to plot this better is presented in Section 5.7, but that requires very accurate timestamping. When using timestamping on arrival it is difficult to differentiate a lost packet and the processor being busy. Therefore the acutal synchronization error is most likely similar to what is seen in Figure 4.2, but with smaller spikes in the synchronization error. To get a better impression of how well it actually performs the bottom plot is a zoomed in plot of the data where no GNSS packet loss occurs. Although this data is not very good due to the packet loss, it clearly shows that the worst synchronization errors has been reduced from around 100 ms to around 10 ms when using the SBC compared to the PC. This is likely due to the SBC only running what is needed for the UAV, and the fact that it is using real-time software.

To further increase the accuracy of the timestamping on arrival, dedicated hardware such as the SenTiBoard can be used. In the previous cases the data was sampled by the SenTiBoard and then sent to the Linux PC or the SBC for timestamping. If the data is timestamped directly in the SenTiBoard the accuracy greatly increases. Figure 4.4 shows the accuracy of timestamping on arrival using the SenTiBoard. Since the accuracy of the timestamps is so high compared to the synchronization error due to the drift, it becomes difficult to see any details when plotting the synchronization error. In order to see the accuracy of the timestamps, the difference between the expected time between two samples and the actual time between two samples is compared. From now on this is referenced as ΔTOV error or ΔTOA error depending on the timestamping method used. The ΔTOA for the ADIS IMU can be seen in the top plot in Figure 4.5. This shows that the error has now greatly reduced from around 10 ms to fluctuating 200 μs before and after the expected time.

To further improve the timestamping accuracy, the sensor hardware needs to be designed for high precision timestamping as well. The ADIS IMU has a pin that toggles when the current sample is valid. Combined with the IC capture functionality of the SenTiBoard this can be used to get very accurate TOV timestamps. In this case, it is necessary to look at the ΔTOV error, the outliers now fluctuate about 6 μs before or after the expected time. This functionality is also present in the STIM IMU which is even more accurate with fluctuation of about 0.1 μs before or after the expected time. These results can be seen as the middle and bottom plots in Figure 4.5.

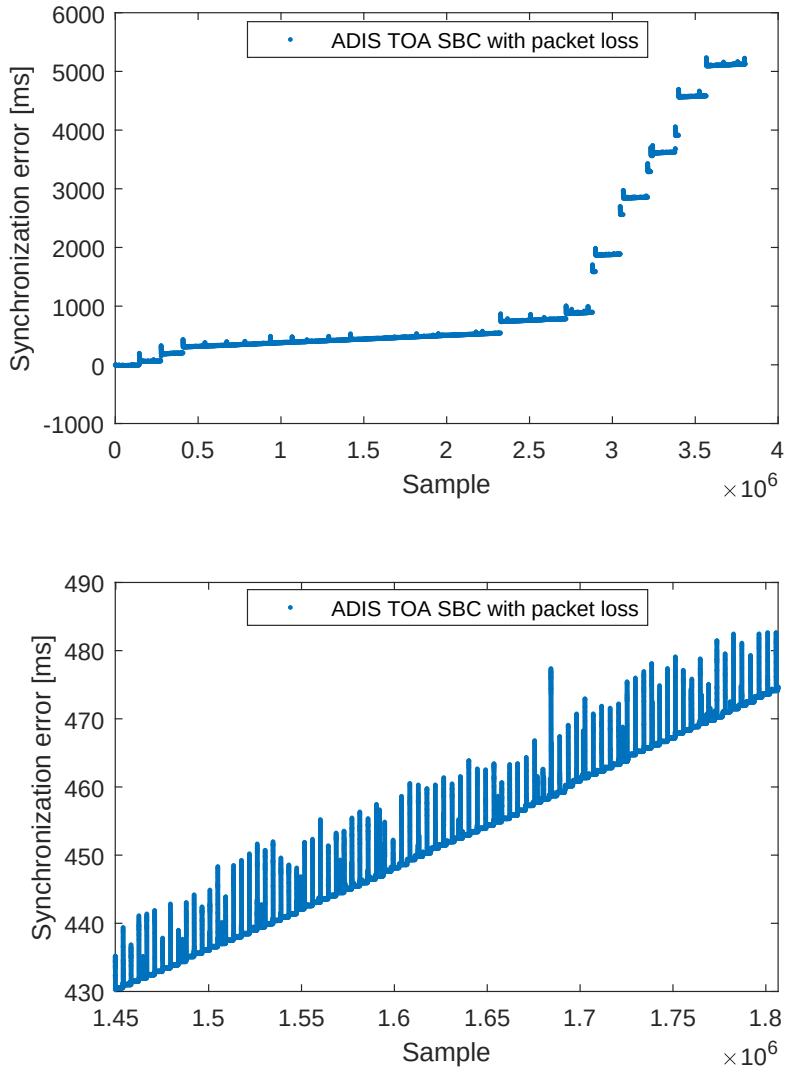


Figure 4.3: Synchronization error when using TOA for ADIS IMU connected to SBC with high amounts of packet loss. The top plot shows the entire runtime while the bottom plot shows a zoomed in view.

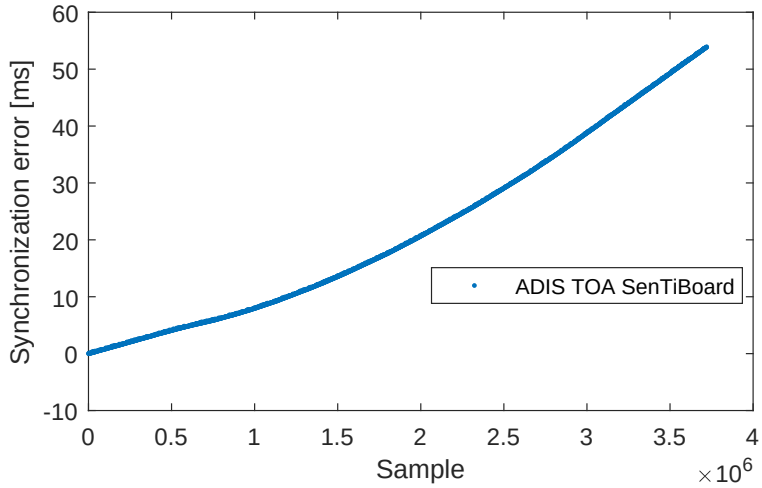


Figure 4.4: Synchronization error when using TOA for ADIS IMU where the SenTiBoard does the timestamping.

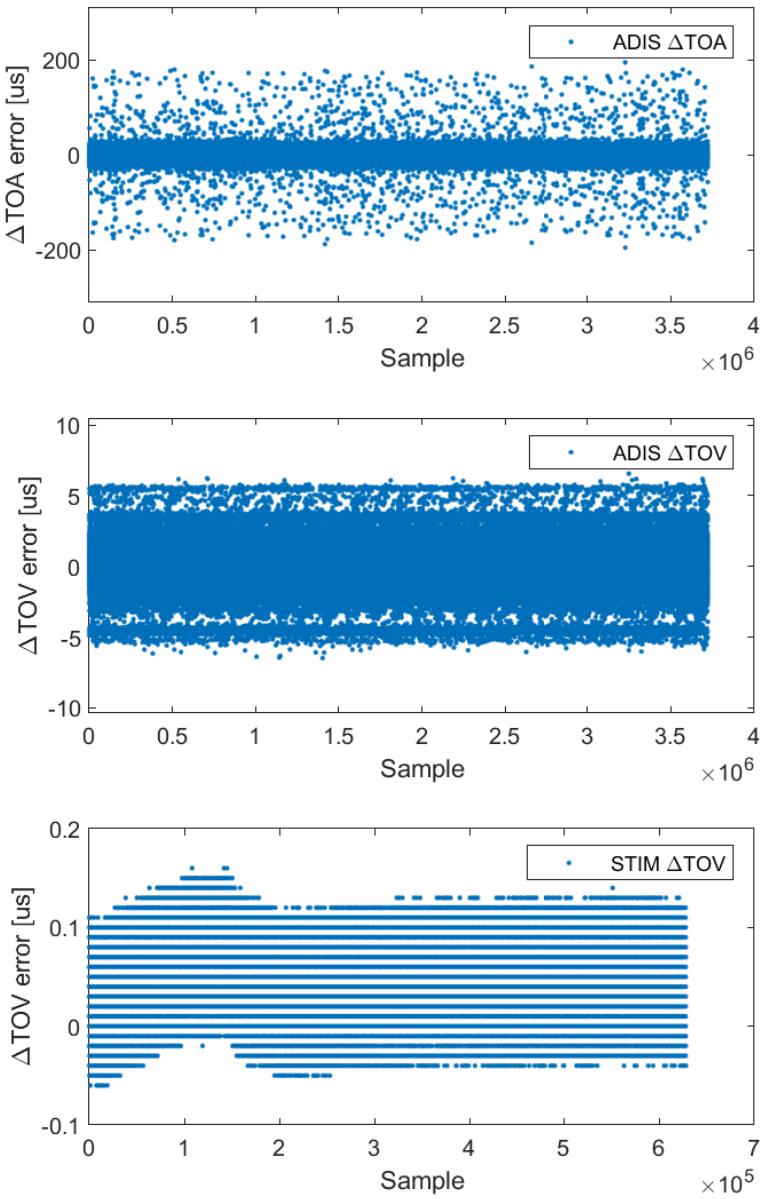


Figure 4.5: The ΔTOA error for the ADIS IMU and ΔTOV error for the ADIS And the STIM IMU. The top plot shows the ΔTOA error for the ADIS IMU. The middle plot shows the ΔTOV plot for the ADIS IMU. The bottom plot shows the ΔTOV error for the STIM IMU.

4.4 Drift

When using both sensor hardware and timestamping hardware that is designed for high presicion timestamping, the accuracy of each individual timestamp becomes very accurate. The problem is that even though the timestamp itself is very accurate, the clocks that are used in both the sensor hardware and the timestamping hardware can drift. When comparing ADIS ΔTOV and ADIS ΔTOA in Figure 4.5 we can see that the spread of the timestamps is much smaller when using TOV. However, when looking at the synchronization error for the TOV which can be seen in the top plot of Figure 4.6, the overall drift of the timestamps are the same as in Figure 4.4 which shows the ADIS TOA.

To see how this can be fixed, the drift of the sensors and the SenTiBoard needs to be evaluated. This can be done by looking at the TOV of the GNSS receiver. The GNSS receiver sends pulses once per second using a PPS signal. This pulse has an root mean square (RMS) accuracy of 30 ns [18]. Any synchronization error higher than this would therefore be the result of a drift in the SenTiBoard clock which is used for sampling this signal. The bottom plot of Figure 4.6 shows the TOV of the GNSS, it is clear that there is a drift and that it varies over time. This drift is much smaller than the drift seen in top plot, which shows the ADIS TOV, where both the SenTiBoard clock and the ADIS clock drifts. This indicates that the drift in the ADIS clock is higher than the drift in the SenTiBoard clock. The way the clocks drift is very closely related to the change in temperature of the sensors. The change in drift over time can be seen by using ΔTOV. In order to see a pattern, we need to average a given number of elements at a time to see the main trends and not just the outliers. For these plots all the samples were averaged such that only 100 points remained for each sensor. This results in the plots shown in Figure 4.7. The top plot shows the ΔTOV mean for the ADIS TOV, while the middle plot shows the ΔTOV mean for the GNSS TOV. These have the same general shape, but different values as the sample rate is different. The bottom plot shows the temperature at the ADIS IMU. They both follow the change in temperature quite closely. This connection between temperature and drift is even more clear when looking at the STIM data which can be seen in Figure 4.8. This figure follows the same structure as the previous figure. From these figures we can see that the drift of the TOV for the ADIS and the GNSS receiver actually change in the same way throughout the runtime even though the synchronization error looks quite different as seen in Figure 4.6.

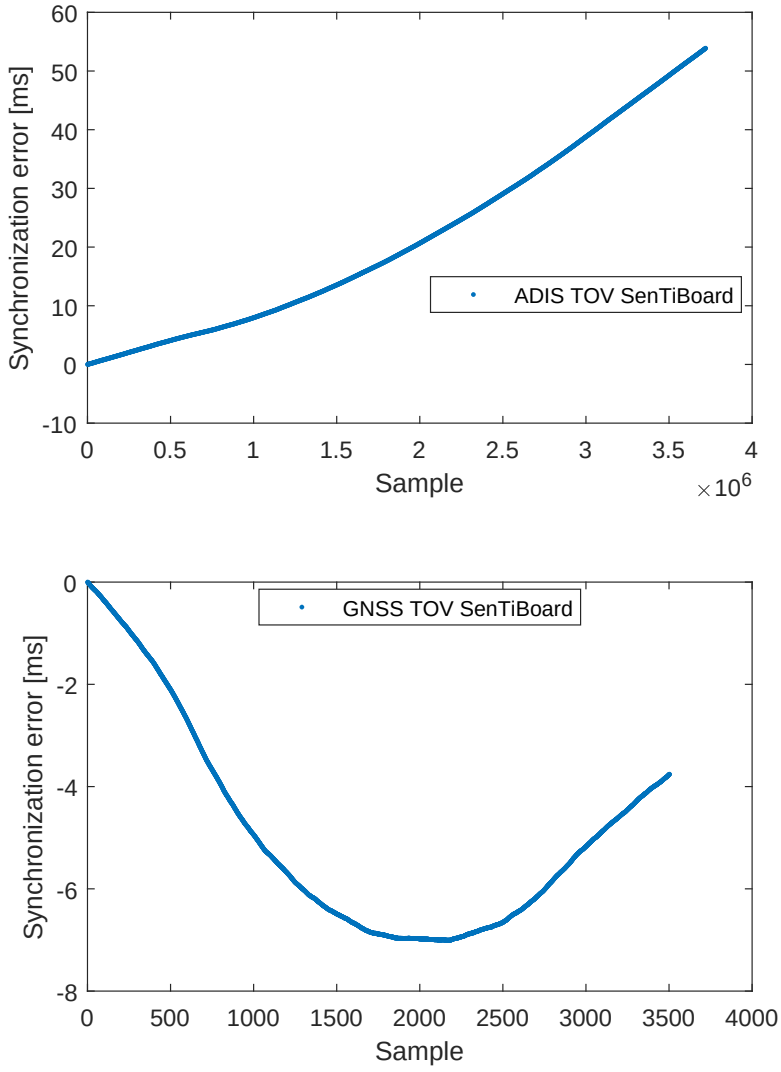


Figure 4.6: Synchronization error when using TOV for ADIS IMU and GNSS receiver where the SenTiBoard does the timestamping.

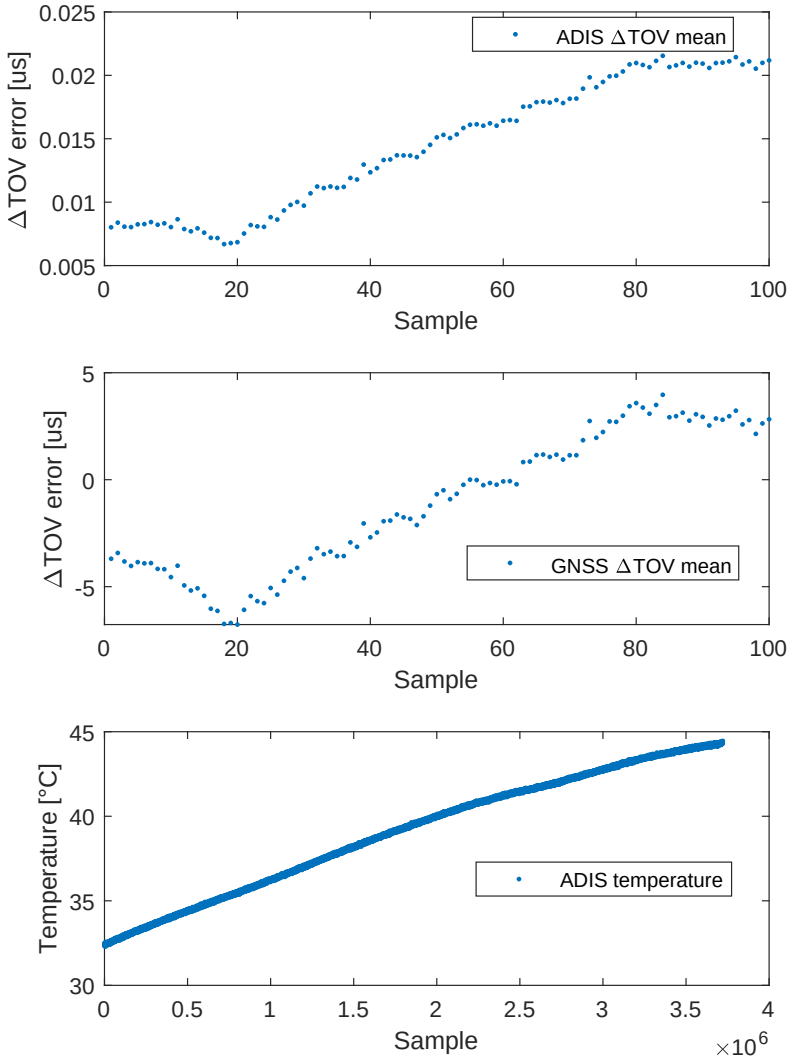


Figure 4.7: ΔTOV error mean for ADIS IMU and GNSS receiver and temperature at ADIS IMU.

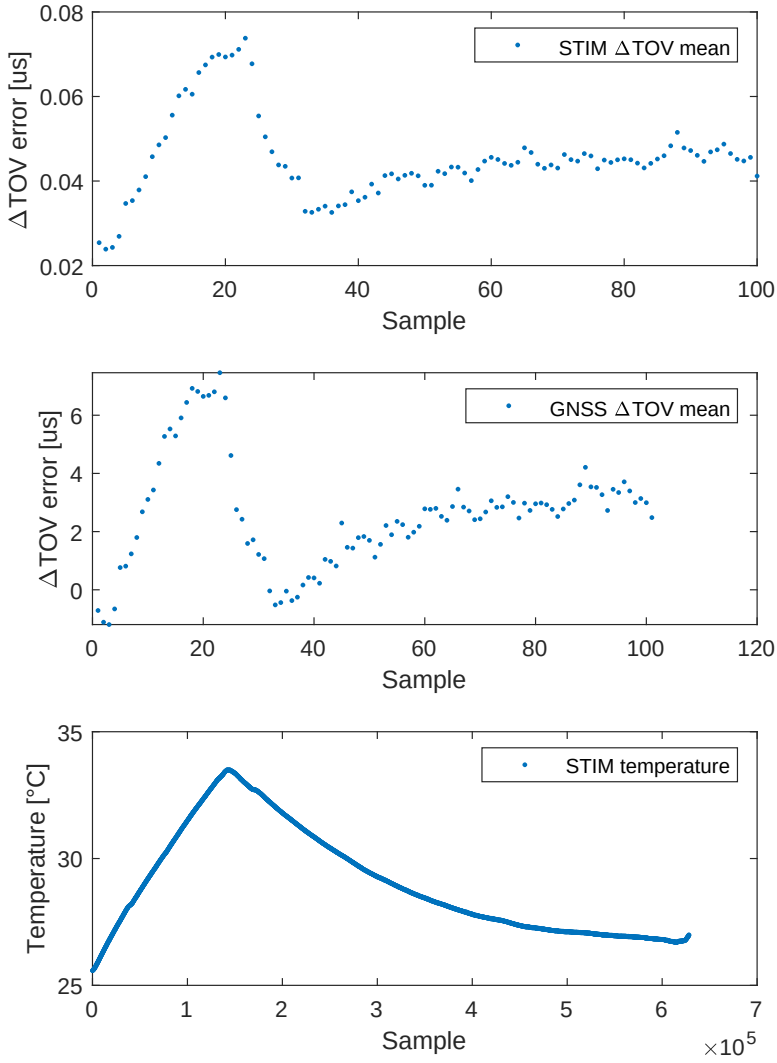


Figure 4.8: ΔTOV error mean for STIM IMU and GNSS receiver and temperature at STIM IMU.

This shows that the drift changes during runtime and is closely linked to the change in temperature. It is possible to design oscillators that minimize drift due to temperature changes by different methods such as utilizing polysilicon resistors with opposite temperature coefficients [21], or looking at the utilization of electron mobility in MOS transistors [22]. If it is not possible to tune the clock during runtime, the drift can be handled by software. This is what this thesis accomplishes in Chapter 5. This is possible if the software algorithms do not take too much time to run, and if the data being slightly delayed to its destination does not affect the result meaningfully. In this case it is important to know exactly when a measurement was taken, but it is not as important that it arrives at its destination immediately. This is because having a high synchronization error gives an incorrect result, while having a delay only causes the data to arrive later. If the delay is known and affects every measurement, it is even possible to compensate for the delay in the timestamp afterwards. As we saw from the bottom plot of Figure 4.3 the data sent to the SBC can be up to 10 ms delayed, which is many orders of magnitude more than the precision that the synchronization error ends up at after using the time synchronization algorithms presented in Chapter 5.

From Figure 3.2 in Chapter 3 we can see that the MRU is connected to the GNSS receiver and receives a PPS signal. This is because the MRU automatically synchronizes its internal clock by comparing it to the time pulse from the GNSS receiver [17]. This is clear from the MRU TOA seen in Figure 4.9 which looks identical to the GNSS TOV seen in the bottom plot of Figure 4.6. This is because the synchronization error in both is due to the SenTiBoard clock drift, as the MRUs internal clock drift has already been handled by the MRU itself. However, since the MRU does not have support for TOV and has to use TOA, the spread of the timestamps is higher than if TOV had been used as can be seen in Figure 4.10 which shows ΔTOA. The drift can be handled by software, but the spread in timestamps must be handled by hardware. Therefore, the end result of the MRU using the algorithms presented in Chapter 5 will be worse than for the IMUs. The MRU does however have a much lower synchronization error than the IMUs before the software timestamp correction due to its internal clock correction.

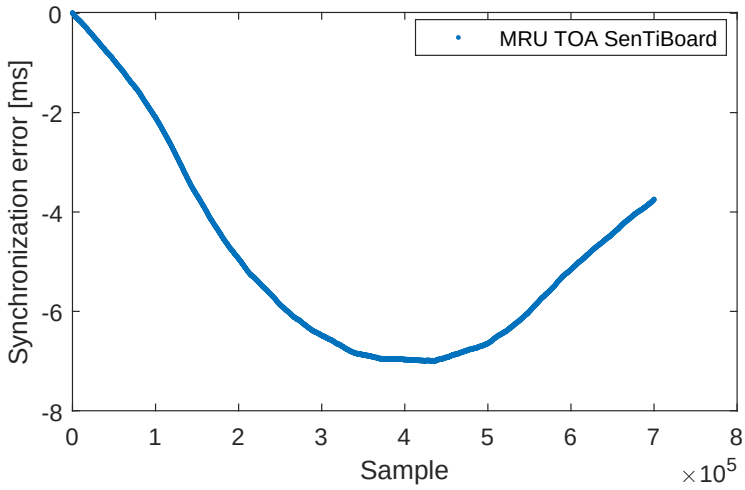


Figure 4.9: Synchronization error when using TOA for MRU where the SenTiBoard does the timestamping.

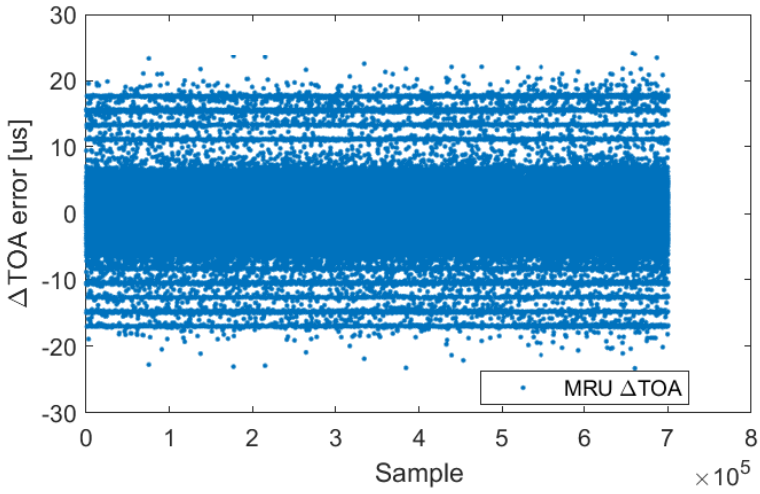


Figure 4.10: The ΔTOA error for the MRU.

4.5 GNSS receiver accuracy

In order to solve the problem of the SenTiBoard clock and the sensor clocks drifting the GNSS receiver is used. The GPS time is stored as TOW as mentioned in Section 2.3.1. The TOW is calculated using $iTow$ which gives TOW in milliseconds and $fTow$ which gives the fractional part of $iTow$ in a range of $\pm 5 \cdot 10^5$ ns. The equation for the precise GPS time can be found in the interface description of the GNSS receiver [23], and is as follows:

$$TOW(n) = (iTow(n) \cdot 10^{-3}) + (fTow(n) \cdot 10^{-9}) \tag{4.2}$$

The $iTow$ and $fTow$ values are retrieved from the NAV_TIMEGPS message sent from the GNSS receiver. The top plot in Figure 4.11 shows the accuracy of the TOW timestamps. The error is now in microseconds, but it is still present. This error, however, is due to the drift of the GNSS receiver clock and not the SenTiBoard clock. This can easily be fixed as the GNSS receiver calculates this internally by comparing its clock to the GPS time. If the TOW value is corrected with the clock bias measurement using the following equation:

$$TOW_{corrected}(n) = TOW(n) + (bias(n) - bias(0)) \tag{4.3}$$

the error becomes much smaller. The clock bias and time accuracy estimate values are retrieved from the NAV_CLOCK message sent from the GNSS receiver. The middle plot in Figure 4.11 shows the timestamp accuracy using TOW corrected with the clock bias. The outliers are $\pm$ 1 ns while most of the timestamps are within $\pm$ 10 picoseconds of the actual time. This gives a solid foundation for correcting the drift present in the other clocks. The last plot in Figure 4.11 shows the time accuracy estimated by the GNSS receiver. This seems to be conservative as using the worst-case values of $\pm$ 1 ns from the fixed TOW gives an accuracy of 2 ns. Since the NAV_TIMEGPS message was not activated for the flights from the second dataset using the STIM IMU, a simulated perfect clock with added normally distributed noise $\pm$ 1 ns was used to correct the STIM IMU.

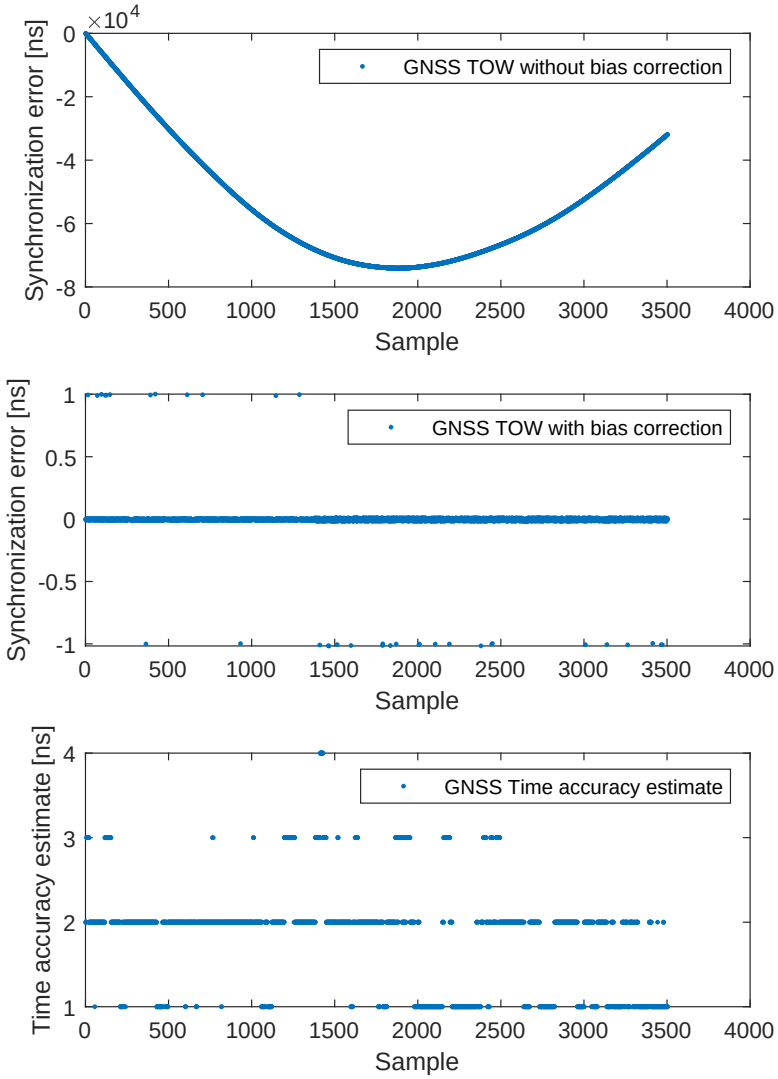


Figure 4.11: Synchronization error for the GNSS TOW with and without bias correction. The bottom plot shows the time accuracy estimated by the GNSS receiver.

5

Time synchronization algorithms

This chapter will present the general form of the algorithms used to correct the synchronization errors. Then three different methods of correcting this error will be presented along with the results from this by showing the synchronization error for each of the sensors. These results will be compared to a tuned Kalman filter. A method for handling packet loss will also be presented. The results from all of this will, in the end, be evaluated using the syncline model. Finally, a method for correcting the synchronization error of the camera without the SenTiBoard timestamping it will be presented.

5.1 Algorithm development

These algorithms were developed incrementally by initially comparing the sensor time with the GNSS receiver time and then using the results from each algorithm to gradually improve upon it. The results of the algorithms were checked visually by plotting and by looking at the RMS of the synchronization error. The algorithms were also designed to work in real-time, but were only evaluated with data from finished flights. The algorithms read the data as if it was continuously arriving and did not use information that a real-time system would not have. Due to this the code attached to this thesis might be a bit different than the algorithms and equations presented in this chapter.

5.2 General algorithm structure

The general algorithm is presented in Algorithm 5.1. There are some values that need to be initialized for all the methods when the algorithm starts. These are the sensor samples per GNSS receiver sample which is called f, the nearest integer of that value rounded down which is called f_int, and the error. The error is set to 0. Since we are using the TOW value of the GNSS receiver as presented in Section 4.5, we need to find out how many sensor samples arrive per GNSS receiver sample. Since the count of packets is an integer number and the sample frequency can be a real number, we need to use the nearest integer

of that value rounded down. This is the value that is then used as the amount of sensor packets to be counted before a new error is calculated. The amount of GNSS packets to be counted before a new error is calculated will then always be one. Since the TOW value is what is used to calculate the new error, this means that it can only be updated when a new GNSS receiver packet arrives. The error correction on the other hand can happen for every sensor packet that arrives. This general structure will be the same for all the methods, but the way in which the error calculation and error correction happens will change.

Algorithm 5.1: The general algorithm which is used as a base for the rest of the algorithms.

GENERAL ALGORITHM

```
1  Initialize values
2  repeat
3      Wait for packet
4      if Sensor packet
5          if Sensor packet count reached and GNSS packet count reached
6              Calculate new error
7          Apply error correction
8      if GNSS packet
9          if Sensor packet count reached and GNSS packet count reached
10             Calculate new error
11 until flight finished
```

5.3 Algorithm 1

The first algorithm is the simplest one and works by comparing the timestamp of the latest received GNSS packet with the timestamp from the sensor. Whenever a new GNSS packet is received this will reduce the synchronization error to close to zero if the drift has been constant since the last packet.

$$\epsilon = TOW_{corrected}(k) - \left(t(n) + TOW_{corrected}(k) \cdot \frac{f - f_{int}}{f} \right)$$
$$t_{corrected}(n) = t(n) + \epsilon \tag{5.1}$$

In the algorithm shown in (5.1), ϵ is the error, k is the count of the current GNSS receiver packet and $t(n)$ is the value of the timestamp at the sensor packet sample n for the current timestamping method. The last part of the error calculation handles the cases where f is not an integer by adding the expected value that the timestamp would have increased in the fractional part of f. The error caused by f not being an integer increases over time as there is always a fractional part of f that is not included. The algorithm fixes this by scaling that part of the error based on the time that has passed. This method calculates a new error when a GNSS receiver packet arrives by looking at the difference between the time that the TOW from the GNSS receiver reports and the time that the timestamp from the SenTiBoard reports. This error is then added to every sensor packet until a new GNSS receiver packet arrives.

Table 5.1: The RMS of the synchronization errors when using the algorithm in Section 5.3 for the ADIS IMU, STIM IMU and the MRU.

Sensor	RMS of synchronization error
ADIS	$9.4 \cdot 10^{-6}$s
STIM	$5.3 \cdot 10^{-6}$s
MRU	$3 \cdot 10^{-6}$s

From this first algorithm we can already see that the results are much better than what they were when just using TOV or TOA. From Figure 5.1 we can see that the synchronization errors have gone down from ms to us. However, if we looked at a zoomed in plot of for example the ADIS TOV as in Figure 5.2 we can see that the error repeatedly increases gradually before going back to being close to zero. The error becomes close to zero whenever a new GNSS receiver packet arrives, but between each packet the error increases with the speed of the clock drift. This can also be observed in the ADIS and STIM plots from Figure 5.1, as the error increases when the clock drift increases. The changes in clock drift was previously presented in Figure 4.7 and Figure 4.8. The results of the algorithm on the different sensors is summarized in Table 5.1 which presents the RMS of the synchronization errors.

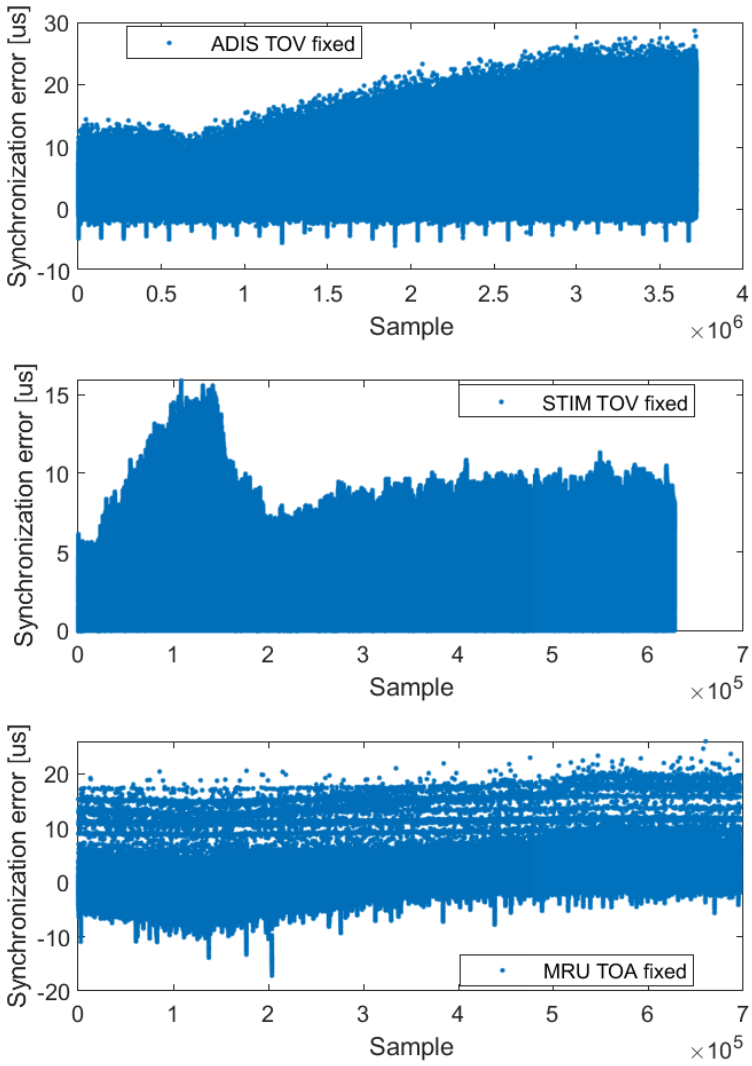


Figure 5.1: Synchronization error when using TOV or TOA that has been corrected using the algorithm from Section 5.3 for the ADIS IMU, STIM IMU and MRU.

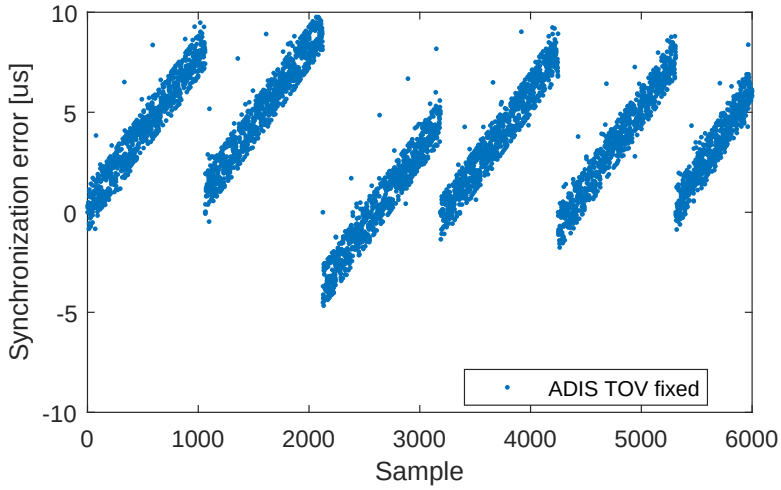


Figure 5.2: A zoomed in plot of the synchronization error when using TOV that has been corrected using the algorithm from Section 5.3 for the ADIS IMU.

5.4 Algorithm 2

The second algorithm seen in (5.2) improves on the first one by correcting not only when a GNSS packet is received, but by also changing the correction based on how many sensor packets that have been received since the previous GNSS packet.

$$\epsilon = TOW_{corrected}(k) - \left(t(n) + \epsilon_{sum} + TOW_{corrected}(k) \cdot \frac{f - f_{int}}{f} \right)$$
$$\epsilon_{sum} = \epsilon_{sum} + \epsilon$$
$$t_{corrected}(n) = t(n) + \epsilon_{sum} + \epsilon \cdot \frac{pkts\_since\_GNNS}{f} \tag{5.2}$$

This time the error is summed together after each error calculation, but also subtracted in the main error calculation. This makes it so that ϵ only contains the change in error while ϵ_{sum} contains the total error. ϵ is then added to the error correction and scaled based on the amount of sensor packets received so far since the previous GNSS receiver packet was received. This change causes the correction to change for each sensor packet that arrives.

By splitting the errors into ϵ and ϵ_{sum} it is possible to correct the synchronization error more often. The total synchronization error is corrected when a new GNSS packet is received as in Section 5.3, but the drift that occurs during the time between two GNSS packets being received can also be corrected. The value of ϵ is how much the synchronization error drifted since the previous GNSS packet was received, and can as such be used to correct the error until the next GNSS packet is received. Therefore ϵ is scaled with the amount of packets currently received divided by the amount of packets expected to be received before a new GNSS packet is received. The amount corrected whenever a new sensor packet is received is therefore how much it drifted on average per packet in the time since the previous GNSS receiver packet was received. This means that a high clock drift does not cause a high synchronization error as the value of the drift is updated whenever a new GNSS packet is received. A sudden change in the drift between two GNSS packets does affect the error as the drift of the previous packet will be used until the next packet arrives.

The results from the second algorithm is a clear improvement from the first algorithm as can be seen in Figure 5.4. The synchronization error has decreased and no longer varies based on the value of the clock drift. This can also be seen Figure 5.3 where the effect of the clock drift is mostly handled by changing the correction for each sensor packet. From the figure it is clear that there are still some packets that are not corrected the way they should. The results of the algorithm on the different sensors is summarized in Table 5.2 which presents the RMS of the synchronization errors.

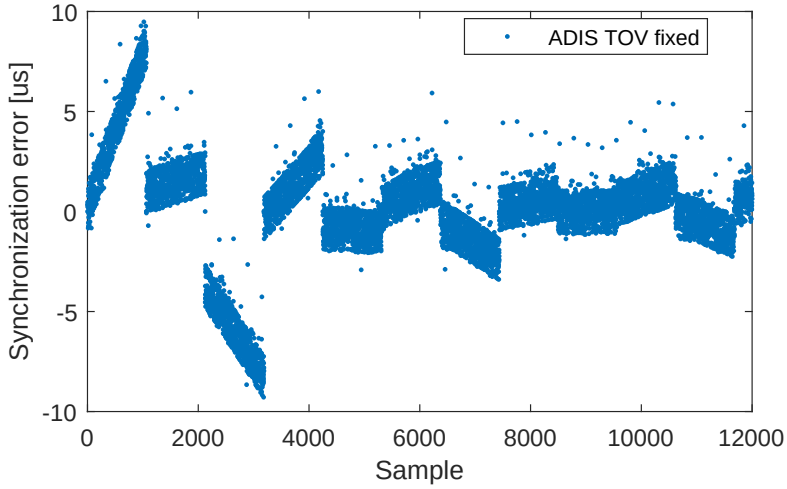


Figure 5.3: A zoomed in plot of the synchronization error when using TOV that has been corrected using the algorithm from Section 5.4 for the ADIS IMU.

Table 5.2: The RMS of the synchronization errors when using the algorithm in Section 5.4 for the ADIS IMU, STIM IMU and the MRU.

Sensor	RMS of synchronization error
ADIS	$1.3 \cdot 10^{-6}$s
STIM	$2.6 \cdot 10^{-7}$s
MRU	$2.7 \cdot 10^{-6}$s

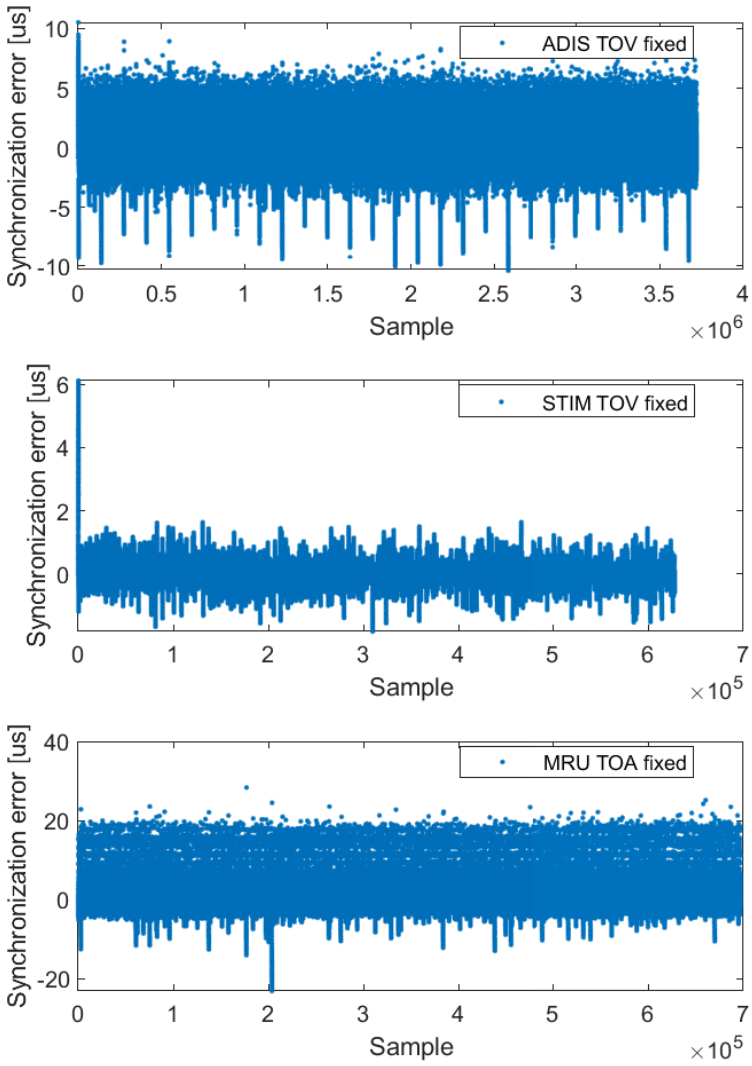


Figure 5.4: Synchronization error when using TOV or TOA that has been corrected using the algorithm from Section 5.4 for the ADIS IMU, STIM IMU and MRU.

5.5 Algorithm 3

The third algorithm seen in (5.3), attempts to solve the problem of the error increasing if the clock drift suddenly changes. This is done by taking a moving average of the error such that the correction is not solely based on the drift between two GNSS packets.

$$\begin{aligned}
&\epsilon(k) = TOW_{corrected}(k) - \left(t(n) + \epsilon_{sum} + TOW_{corrected}(k) \cdot \frac{f - f_{int}}{f}\right) \\
&\epsilon_{sum} = \epsilon_{sum} + \epsilon(k) \\
&mov_avg = mean(\epsilon(k - avg_count : k) \\
&t_{corrected}(n) = t(n) + \epsilon_{sum} + mov_avg \cdot \frac{pkts_since_GNNS}{f}
\end{aligned} \tag{5.3}$$

For this algorithm the moving average of the error of a given number of previous GNSS receiver packets is calculated. Up until the specified amount has been received the average is taken of the available error values. This change makes it so the error that is corrected for each sensor packet is an average of the previous errors, and not just the error that was present when the previous GNSS receiver packet arrived as was the case in the algorithm in Section 5.4. This works to some extent as in Figure 5.5 the errors are closer to zero and the drift seems to affect it less. The difference from the previous algorithm is, however, very minor and as shown in Table 5.8 the results are better for the ADIS and the MRU, but worse for the STIM. It is difficult to see any major differences in the total result that can be seen in Figure 5.6, but due to averaging, the outliers are a closer to the rest of the timestamps compared to using the algorithm from Section 5.4. In order to find out how many elements that should be included in the moving average, the algorithm was tested with every possible input. The result from this for the ADIS IMU can be seen in Figure 5.7. The top plot shows the RMS of the synchronization error for every possible amount of elements used in the moving average. The plot clearly shows that the accuracy decreases at higher values. A zoomed in plot of the same data is seen in the lower plot. This shows that the lowest RMS for the synchronization error is achieved by taking the moving average of 15 elements at a time. The results for the MRU and the three flights for the STIM IMU were very similar. Therefore, the algorithm should work on most datasets fairly well if the moving average of about 5 - 15 elements at a time is chosen. The results of the algorithm on the different sensors is summarized in Table 5.3 which presents the RMS of the synchronization errors.

Table 5.3: The RMS of the synchronization errors when using the algorithm in Section 5.5 for the ADIS IMU, STIM IMU and the MRU.

Sensor	RMS of synchronization error
ADIS	$9.6 \cdot 10^{-7}$s
STIM	$3.2 \cdot 10^{-7}$s
MRU	$2.5 \cdot 10^{-6}$s

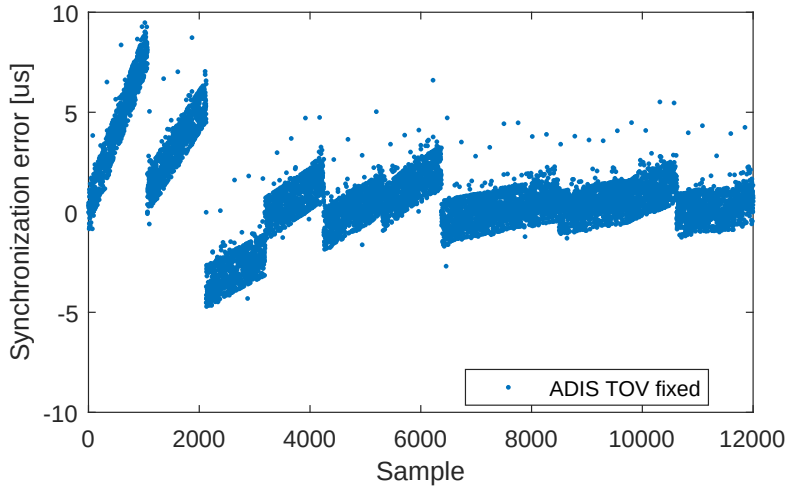


Figure 5.5: A zoomed in plot of the syncrhonization error when using TOV that has been corrected using the algorithm from Section 5.5 for the ADIS IMU.

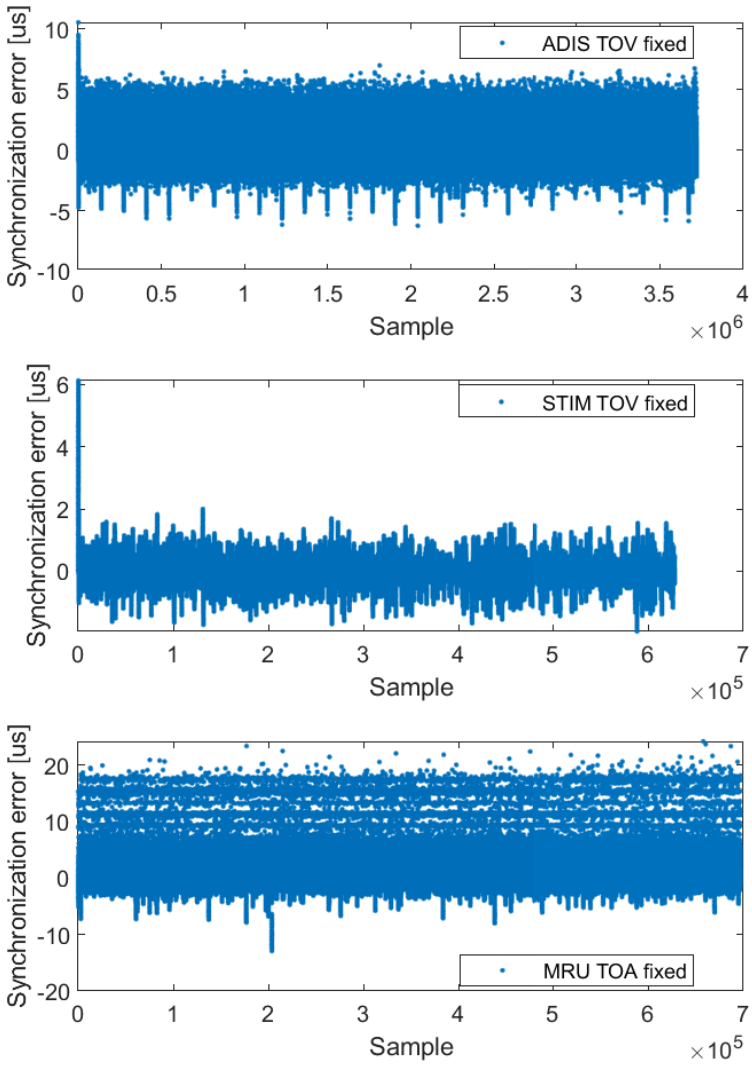


Figure 5.6: Synchronization error when using TOV or TOA that has been corrected using the algorithm from Section 5.5 for the ADIS IMU, STIM IMU and MRU.

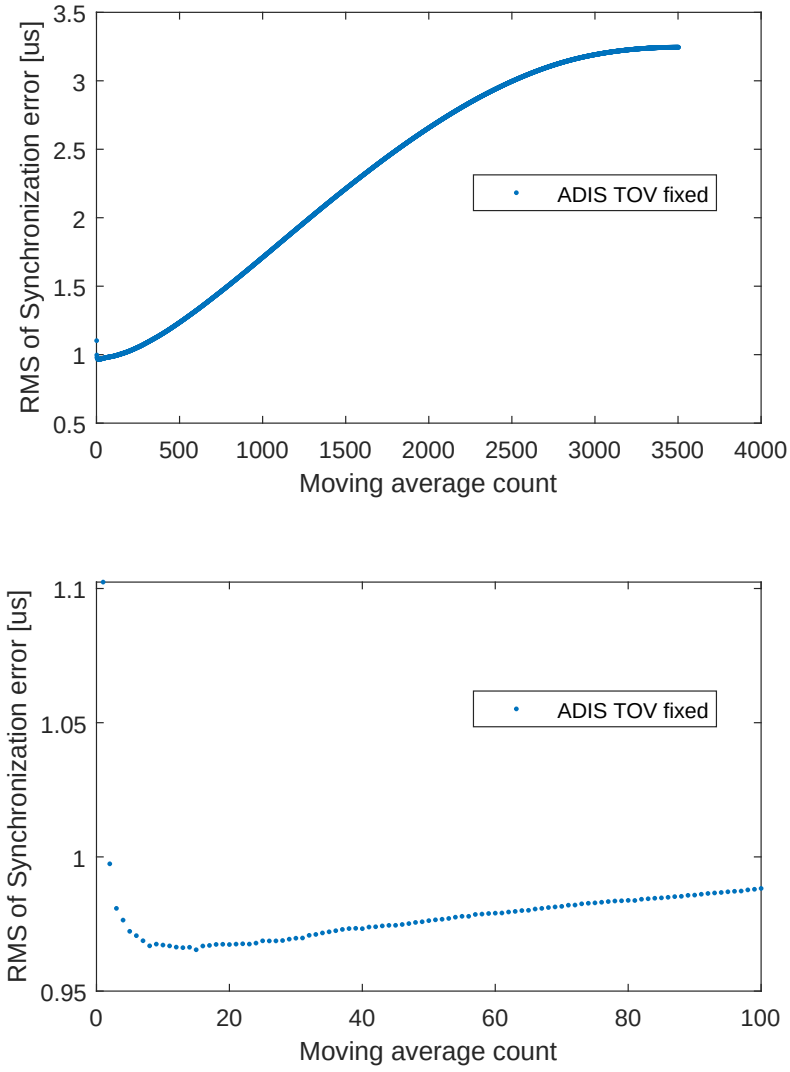


Figure 5.7: RMS of the synchronization error when using the algorithm in Section 5.5 with differing amounts of elements chosen for the moving average.

5.6 Kalman filter

Kalman filters have been used to estimate clock drift in order to perform clock synchronization. In [24] a wireless PTP network uses the grandmaster clock as a time reference similar to how the GPS time is used in this thesis. A Kalman filter is used to estimate the clock drift and clock offset to adjust the PTP synchronization, which improves the synchronization.

Since Kalman filters have been proved to work for estimating clock drift, this was used as the fourth method used to correct the synchronization error. The Kalman filter follows the general structure from Algorithm 5.1, but has to be a bit different due to how a Kalman filter works. As explained in Section 2.5, the Kalman filter has two main steps, the prediction step and the update step. In this case the prediction step functions as the error correction while the update step functions as the new error calculation. The initialization of a Kalman filter is also more complicated as it has to be tuned. The values selected for the tuning variables and the model variables can be seen in the initialization part of Table 5.5. The prediction part of the Kalman filter is performed whenever a new sensor packet arrives and the update part of the Kalman filter is performed when a new error should be calculated as per the structure in Algorithm 5.1. These formulas can be seen in the prediction and update part of Table 5.5. The results from using the Kalman filter can be seen in Figure 5.8. The results look similar to the previous two algorithms and, as shown in Table 5.6, the results are not that different from the other algorithms.

The clock drift was modelled as a first order Gauss-Markov Process, with states x where x_1 is clock time and x_2 is clock drift. This means that $\dot{x_1} = x_2$ and if discretized $\dot{x_2}$ becomes:

$$x_{k+1} = e^{-\Delta t/\tau} x_k + \sqrt{\sigma^2(1 - e^{-2\Delta t/\tau})}\, w_k \tag{5.4}$$

where k is the time index, Δt is the time sampling interval and w_k is white gaussian noise [25]. Using this we get F, G and σ_{drift_Q} from Table 5.5. F is then discretized to get Φ in Table 5.5. G is then used to get Q which is the process noise covariance matrix, which is based on the noise in the clock drift.

The value of the time constant τ was found by first checking how much the system had drifted in 60 seconds. That value was then used as $f(t)$ in the step response for a first order system: $f(t) = 1 - e^{\frac{-t}{\tau}}$ where t was replaced by 60. This was then used to solve for τ. This was repeated in increments of 60 second blocks for the entire dataset for each sensor and the result was approximately $1 \cdot 10^5$s for all the sensors. The initial value of the error covariance matrix P was found by looking at the steady state of P after many updates. The value was around $1 \cdot 10^{-15}$ for both states for all the sensors. To find the standard deviation of the clock drift the plots from Figure 5.1 were used. The synchronization error in these plots drift from 0 s to 10 μs in one second, so 10 μs was chosen as an initial value for tuning. For each sensor a binary search was then performed, and a test performed with each value until the RMS of the synchronization error was minimized. The result of this can be seen in Table 5.4. The value of the measurement noise covariance matrix R is based on the noise in the measurement signal which in this case is the timestamp of the GNSS receiver. This was chosen as the square of the drift of the GNSS clock at flight start since this Kalman filter does not use a time varying Q or R.

Table 5.4: The clock drift for the different sensors used when tuning the Kalman filter.

Sensor	standard deviation of clock drift (σ_{drift})
ADIS	$6 \cdot 10^{-6}$s
STIM	$50 \cdot 10^{-6}$s
MRU	$3 \cdot 10^{-6}$s

Table 5.5: The values selected for initializing the Kalman filter and the prediction and update formulas.

Initialization
$\tau = 10^5$
$P = \begin{bmatrix} 10^{-15} & 0 \\ 0 & 10^{-15} \end{bmatrix}$
$R = GNSS\_drift\_at\_start^2$
$\hat{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $x_1 =$ clock time, $x_2 =$ clock drift
$\sigma_{drift\_Q} = \sqrt{\frac{2}{\tau \cdot \sigma_{drift}{}^2}}$
$F = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-1}{\tau} \end{bmatrix}$
$sample\_time = \frac{1}{sensor\_freq}$
$G = \begin{bmatrix} 0 \\ \sigma_{drift\_Q} \end{bmatrix}$
$\Phi = discretize(F, G, sample\_time)$
$Q = G \cdot G^T \cdot sample\_time$
$H = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
Prediction
$\hat{x} = \Phi \cdot \hat{x} + \begin{bmatrix} t(n) - t(n-1) \\ 0 \end{bmatrix}$
$P = \Phi \cdot P \cdot \Phi^T + Q$
$P = \frac{P + P^T}{2}$
Update
$y = TOW_{corrected}(k) - TOW_{corrected}(k) \cdot \frac{f - f_{int}}{f}$
$\hat{y} = H \cdot \hat{x}$
$K = P \cdot H^T \cdot (H \cdot P \cdot H^T + R)^{-1}$
$\hat{x} = \hat{x} + K \cdot (y - \hat{y})$
$P = (I - K \cdot H) \cdot P \cdot (I - K \cdot H)^T + K \cdot R \cdot K^T$

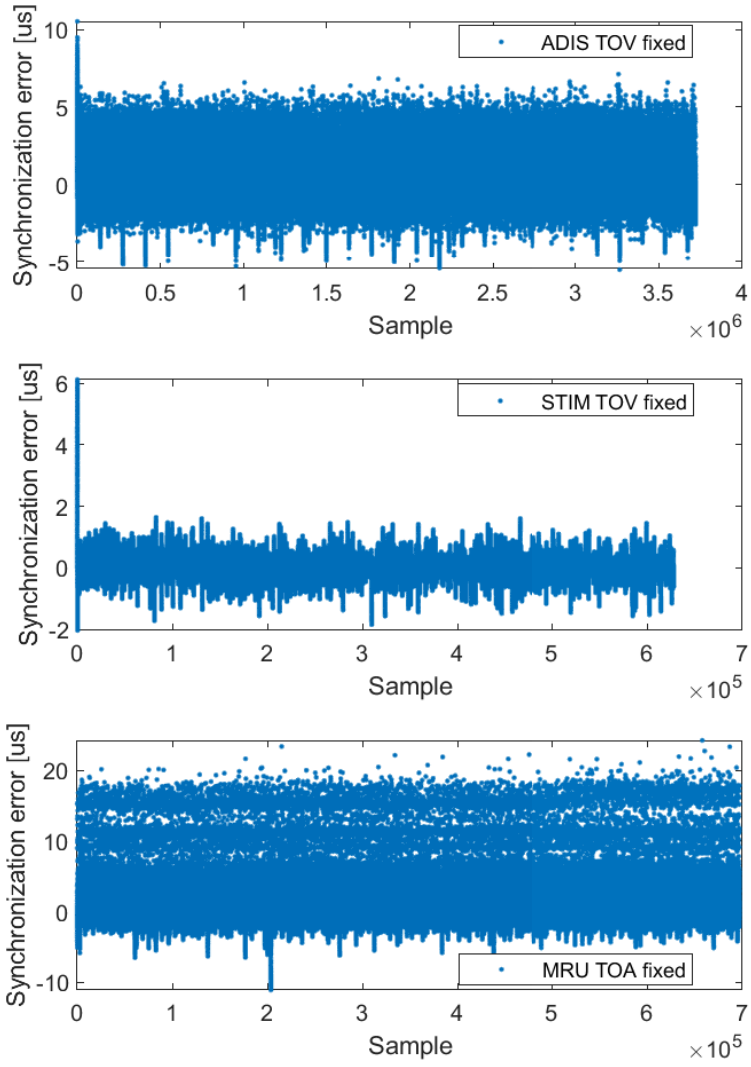


Figure 5.8: Synchronization error when using TOV or TOA that has been corrected using the kalman filter described in Section 5.6 for the ADIS IMU, STIM IMU and MRU.

Table 5.6: The RMS of the synchronization errors when using the algorithm in Section 5.6 for the ADIS IMU, STIM IMU and the MRU.

Sensor	RMS of synchronization error
ADIS	$9.4 \cdot 10^{-7} s$
STIM	$2.6 \cdot 10^{-7} s$
MRU	$2.5 \cdot 10^{-6} s$

5.7 Packet loss

All the algorithms presented above work well if there is no packet loss, however, as they expect the time between each packet to be fairly constant, they do not work that well if a packet is lost. If a single GNSS receiver packet is lost, it does not affect the result, but multiple packets lost could be an issue. This is because the algorithm will continue to correct the sensor packets with the last error calculation that was performed. If there is a large change in temperature, which in turn causes a large change in clock drift at the same time as multiple GNSS receiver packets are lost, then this would make the system less accurate until the next GNSS receiver packet arrives. However, if sensor packets are lost this causes a bigger issue as the error is calculated as the difference between the actual time and the expected time given the current packet count. As an example, consider the scenario where the time between GNSS receiver packets is one second and the amount of sensor packets per second is 1000 packets. If 100 sensor packets are lost within that time, the time reported at the GNSS receiver packet which will be compared with the 1000th sensor packet that is received will be off by 100 times the sample time of the sensor. The timestamp of the 1000th sensor packet received is what would have been the timestamp of the 1100th sensor packet if no packets were lost, and therefore, the amount of sensor packets that are lost needs to be counted and subtracted during the error calculation step. This error will add up over time and therefore the total count of sensor packets lost needs to be tracked. The synchronization error calculation presented in Equation (4.1) has to be modified to account for the packet loss. This is done by counting the amount of sensor packets lost and when they were lost. The error is then added to the synchronization error at the correct packet count to account for the sample rate of the received packets not being constant. The new synchronization error calculation can be seen in (5.5) and is used for the plots with packet loss in this section. Figure 5.9 shows the result of different kinds of packet loss, where all the plots show the synchronization error when using the algorithm from Section 5.5. The top plot is of the test dataset with simulated packet loss of GNSS packets. This causes no additional error, since only a few GNSS packets are lost. The middle plot shows the result of simulating having a lot of ADIS packets lost at the start of the runtime. This results in the synchronization error increasing a lot when the packets are lost and it stays that way for the rest of the runtime. This is because the GNSS receiver packet count and the sensor packet count is not synced. The bottom plot is the result from using the dataset with a high amount of packet loss and shows that the synchronization error increases whenever sensor packets are lost and keeps getting worse throughout the runtime since packets are lost frequently.

$$e(n) = t(n) - \left(t(0) + \frac{n + packets\_lost\_since\_start(n)}{sensor\_freq} \right) \tag{5.5}$$

To correct for this the general algorithm presented in Algorithm 5.1 needs to be modified slightly. Every time a sensor packet is received it checks if there was any sensor packets lost between this packet and the previous packet. The new structure of the algorithm can be seen in Algorithm 5.2. The method for checking if a sensor packet is lost or not is to check the difference between the previous and the current sensor packet and check if the time between them is more than expected. The way this is done can be seen in

Algorithm 5.3. The error calculation is then modified by subtracting the offset caused by each sensor packet lost which can be seen in (5.6). The results of using this algorithm on the three same scenarios as in Figure 5.9, can be seen in Figure 5.10. The result in the top plot is as before, since there were no ADIS packets lost. The middle plot is significantly improved, and only shows minor errors at the start. The bottom plot has a few outliers near the start, but generally stays close to 0 despite the high amount of packet loss. A zoomed in plot of the bottom plot from Figure 5.10 can be seen in Figure 5.11. This shows that the synchronization error suffers if the packet loss is very high, but that the algorithm works even in extreme conditions. The RMS of the synchronization error on the dataset with packet loss for the ADIS IMU is 2s without packet loss handling and $2.9 \cdot 10^{-3}$s with packet loss handling when using the algorithm from Section 5.5 and Section 5.7 respectively.

Algorithm 5.2: The general algorithm adjusted to compensate for packet loss.

GENERAL ALGORITHM FOR PACKET LOSS

```
1  Initialize values
2  repeat
3      Wait for packet
4      if Sensor packet
5          if Sensor packet lost
6              Increase lost sensor packet count
7          if Sensor packet count reached and GNSS packet count reached
8              Calculate new error
9          Apply error correction
10     if GNSS packet
11         if Sensor packet count reached and GNSS packet count reached
12             Calculate new error
13 until flight finished
```

Algorithm 5.3: Code to check for packet loss.

CHECK FOR PACKET LOSS

```
1  time between packets = current packet - prev packet
2  while time between packets > (expected time between packets + margin)
3      packets lost = packets lost + 1
4      time between packets = time between packets - expected time between packets
5  return packets lost
```

$$
\begin{aligned}
\epsilon(k) &= TOW_{corrected}(k) - \left(t(n) + \epsilon_{sum} + TOW_{corrected}(k) \cdot \frac{f - f_{int}}{f} \right) \\
\epsilon(k) &= \epsilon(k) - \frac{sensor\_packets\_lost}{sensor\_freq} \\
\epsilon_{sum} &= \epsilon_{sum} + \epsilon(k) \\
mov\_avg &= mean(\epsilon(k - avg\_count : k) \\
t_{corrected}(n) &= t(n) + \epsilon_{sum} + mov\_avg \cdot \frac{pkts\_since\_GNNS}{f}
\end{aligned} \tag{5.6}
$$

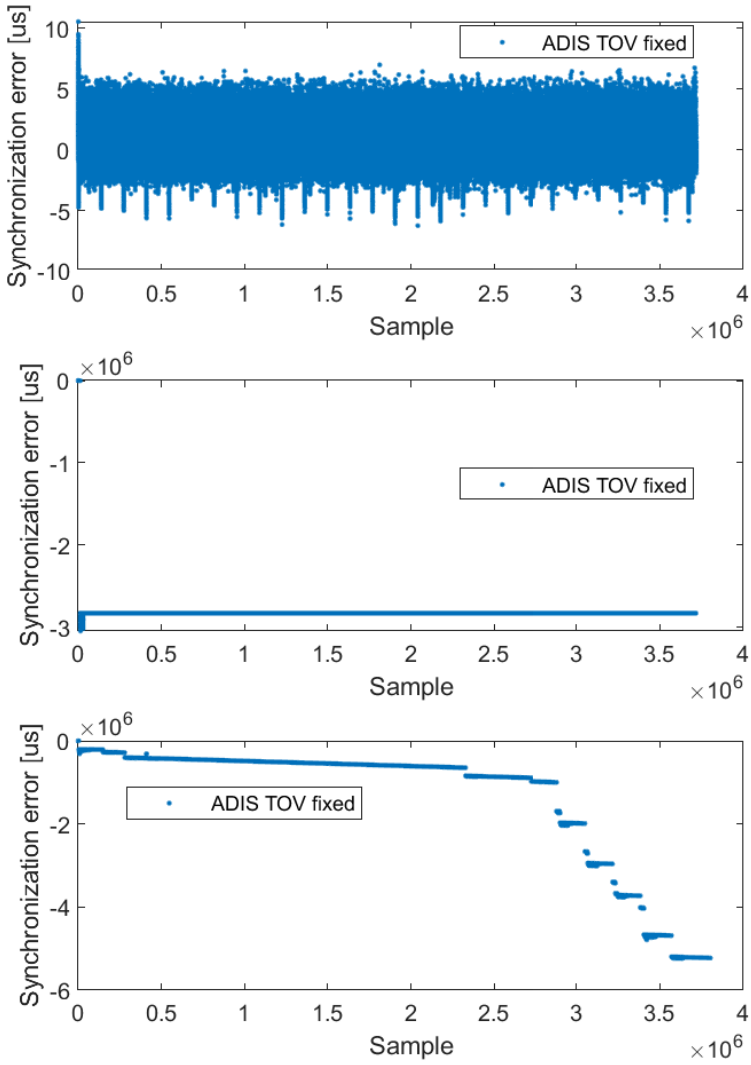


Figure 5.9: Synchronization error when using TOV that has been corrected using the algorithm in Section 5.5 for the ADIS IMU. The top and middle plot use the test dataset and has simulated packet loss while the bottom plot uses the dataset with a high amount of packet loss.

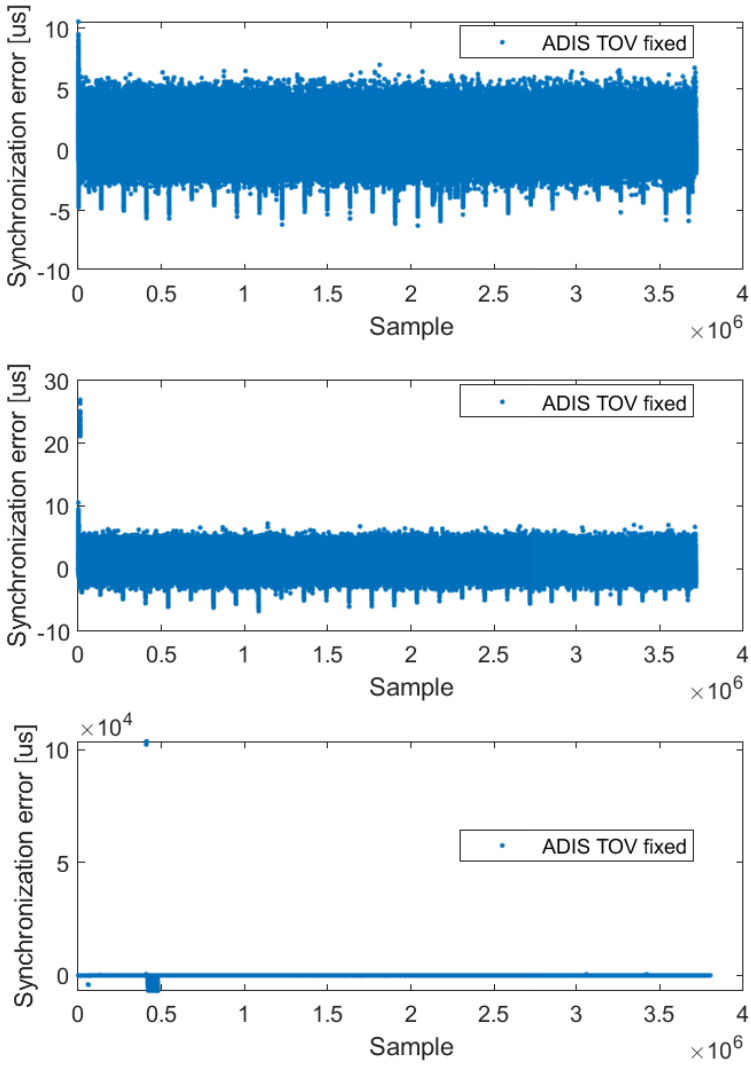


Figure 5.10: Synchronization error when using TOV that has been corrected using the algorithm in Section 5.5 with the packet loss correction from Section 5.7 for the ADIS IMU. The top and middle plot use the test dataset and has simulated packet loss while the bottom plot uses the dataset with a high amount of packet loss.

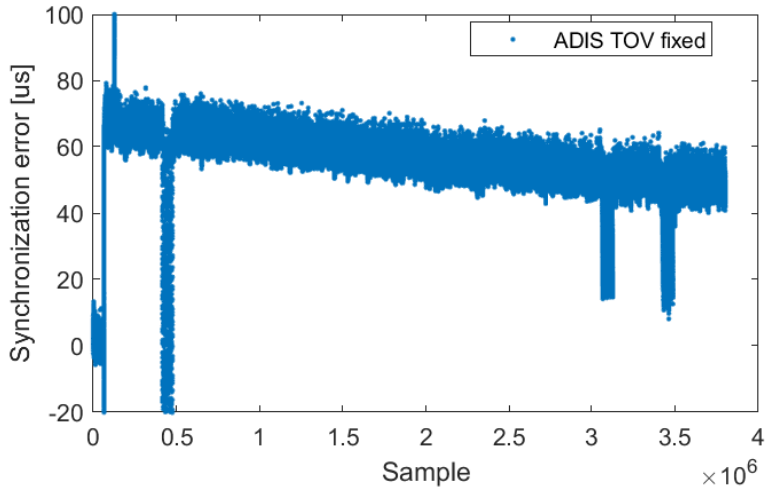


Figure 5.11: A zoomed in plot of the Synchronization error when using TOV that has been corrected using the algorithm in Section 5.5 with the packet loss correction from Section 5.7 for the ADIS IMU. The plot uses the dataset with a high amount of packet loss.

5.8 Syncline

In order to compare the accuracy of each of the algorithms presented above, the syncline model as presented in Section 2.4 is used. The packet loss algorithm is not included as it is simply meant to show how the algorithms can be adapted to account for packet loss. The RMS of the synchronization error from each algorithm is used in the syncline model to compare the accuracy of the different algorithms and to check if the timing precision is a limiting factor or not given the systems dynamics and the sensor accuracy. In order to generate a syncline model for the IMUs and the MRU we need values for v_{max}, ω_{max}, d, σ_p, σ_r, σ_Θ and σ_u. The values chosen for the syncline can be seen in Table 5.7. The exact values chosen here are not important as this is mainly supposed to be used to compare the different algorithms. The precision of the sensors and the dynamics of the system can be changed in order to test a system with different conditions. As a baseline, the Syncline when using timestamping on arrival and TOV with the ADIS IMU, is shown in Figure 5.12. The RMS of the synchronization error is used here as well, but it is not as representative of the actual synchronization error since the error increases a lot over time. The value shown is therefore roughly equivalent to the synchronization error after half the runtime has passed. The RMS of the synchronization error when using timestamping on arrival and TOV is roughly $2.5 \cdot 10^{-2}$s. The synclines for the different algorithms for the ADIS IMU without packet loss can be seen in Figure 5.13. This shows that the first algorithm performs the worst, with each new algorithm performing better. The third algorithm and the Kalman filter has the same accuracy. With the system values chosen here, the first algorithm is sync-bound while the other algorithms and the Kalman filter is sensor-bound. The syncline for the STIM IMU shown in Figure 5.14 shows mostly the same results, except this time all the algorithms are sensor-bound. The syncline for the MRU also follows the same pattern as seen in Figure 5.15. Table 5.8 shows the results from all of the algorithms. The results show that the Kalman filter produces the best result, closely followed by the third algorithm. The second algorithm performs very well compared to the third and the Kalman filter considering that it does not need to be tuned at all and is less complex.

Table 5.7: The values used for the syncline model

d	400
v_{max}	5
ω_{max}	180
σ_p	0.01
σ_r	0.01
σ_Θ	0.001
σ_u	0.01

Table 5.8: The RMS of the synchronization errors for every algorithm for the ADIS IMU, STIM IMU and the MRU.

Sensor	RMS Algorithm 1	RMS algorithm 2	RMS algorithm 3	RMS Kalman filter
ADIS	$9.4 \cdot 10^{-6}$s	$1.3 \cdot 10^{-6}$s	$9.6 \cdot 10^{-7}$s	$9.4 \cdot 10^{-7}$s
STIM	$5.3 \cdot 10^{-6}$s	$2.6 \cdot 10^{-7}$s	$3.2 \cdot 10^{-7}$s	$2.6 \cdot 10^{-7}$s
MRU	$3 \cdot 10^{-6}$s	$2.7 \cdot 10^{-6}$s	$2.5 \cdot 10^{-6}$s	$2.5 \cdot 10^{-6}$s

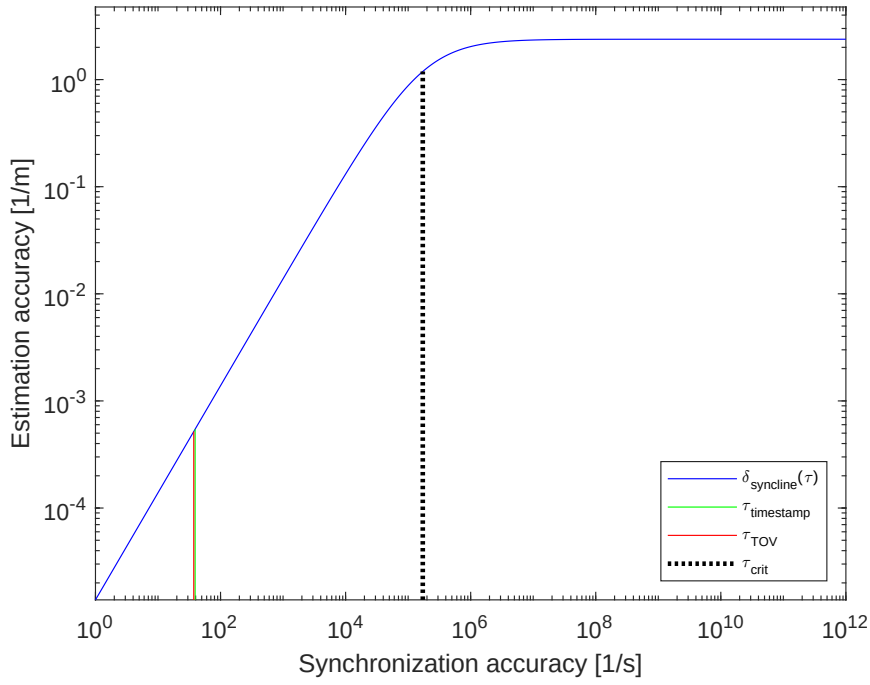


Figure 5.12: Syncline model showing the estimation accuracy and synchronization accuracy when using timestamping on arrival and TOV for the ADIS IMU.

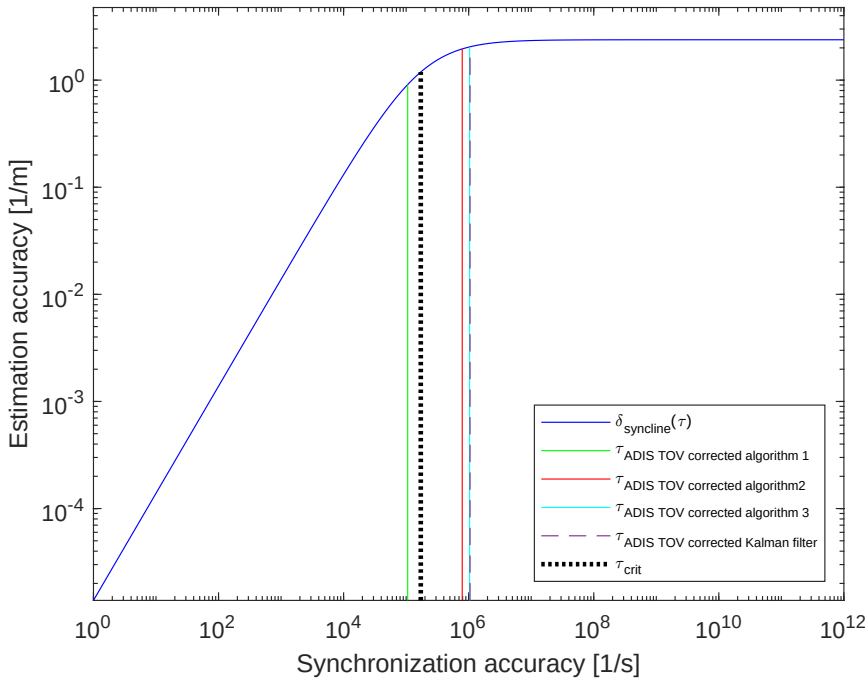


Figure 5.13: Syncline model showing the estimation accuracy and synchronization accuracy of the different error correction algorithms for the ADIS IMU.

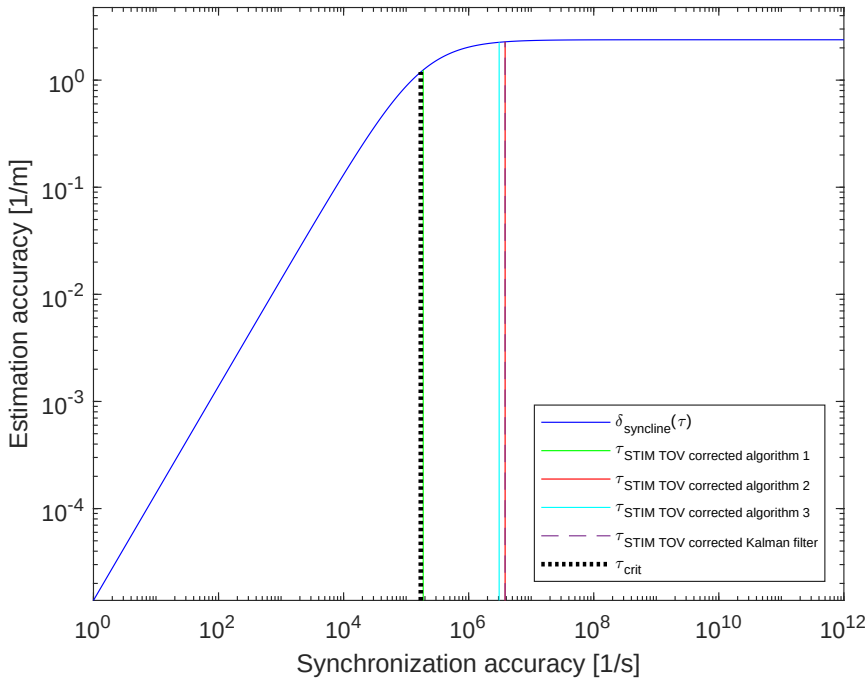


Figure 5.14: Syncline model showing the estimation accuracy and synchronization accuracy of the different error correction algorithms for the STIM IMU.

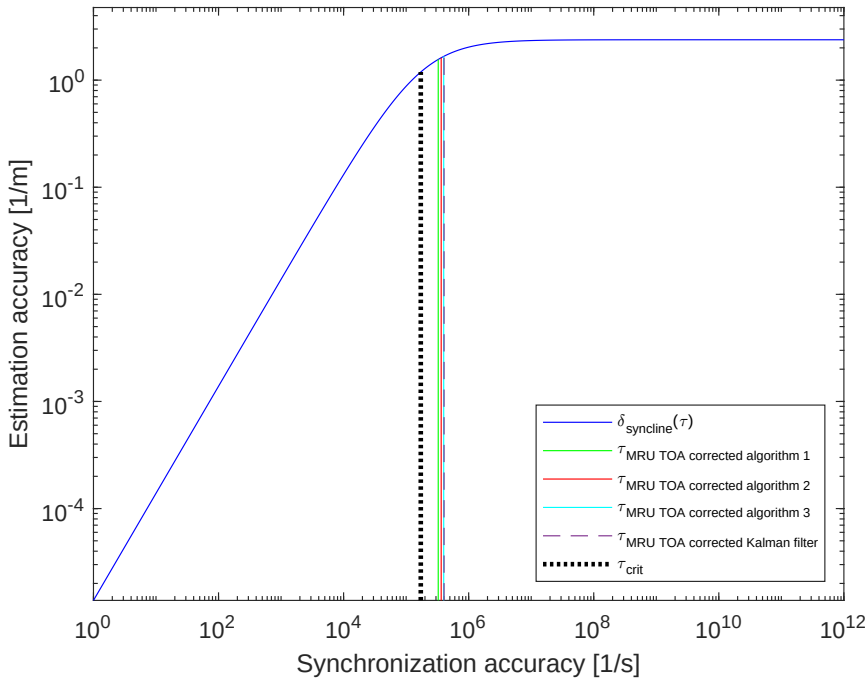


Figure 5.15: Syncline model showing the estimation accuracy and synchronization accuracy of the different error correction algorithms for the MRU.

5.9 Correcting the camera

As shown in Figure 3.2 the camera is triggered by the SenTiBoard, but the data flows to the SBC and does not get timestamped by the SenTiBoard. The error that we want to correct is the SenTiBoard drift, since this is what affects when the camera will be triggered. We do not need to correct the drift of the camera clock since the timestamp counter of the camera can be reset to 0 when taking a picture [26]. Therefore, the timestamp of the camera should count up to the sample time used for the SenTiBoard triggering and then reset to 0. No data from the camera was collected, so to check how the camera would be corrected the MRU data can be used as an example. This works since the MRU autocorrects, which results in only the SenTiBoard drift affecting the timestamps. To correct for the SenTiBoard drift we simply find the difference between the TOV and the TOW of the GNSS receiver as shown in (5.7). The result of this on the MRU TOA can be seen in Figure 5.16. The RMS of this synchronization error is $3 \cdot 10^{-6}$. In this case it is not important to count the GNSS packets, since the newest error available gives the current bias and can be added whenever a new MRU packet is received. The correction calculation has to change when actually using the camera as the camera only counts up to the sample time of the SenTiBoard trigger. The new correction is shown in (5.8). The difference from the last correction is that only the change in error that has occurred since the last camera packet is added. This is because the MRU adds the sum of the drift over time, while the camera only adds the current drift due to it only counting up to the sample time of the SenTiBoard trigger. The corrected timestamp of the camera is then found by initializing a counter to 0 in the SBC when the other timers start and then incrementing it with the corrected TOV.

$$\begin{aligned} \epsilon &= TOW_{corrected}(k) - t_{GNSS}(k) \\ t_{camera\_corrected}(n) &= t_{camera}(n) + \epsilon \end{aligned} \tag{5.7}$$

$$\begin{aligned} \epsilon(k) &= TOW_{corrected}(k) - t_{GNSS}(k) \\ t_{camera\_corrected}(n) &= t_{camera}(n) + \frac{\epsilon(k) - \epsilon(k-1)}{f} \end{aligned} \tag{5.8}$$

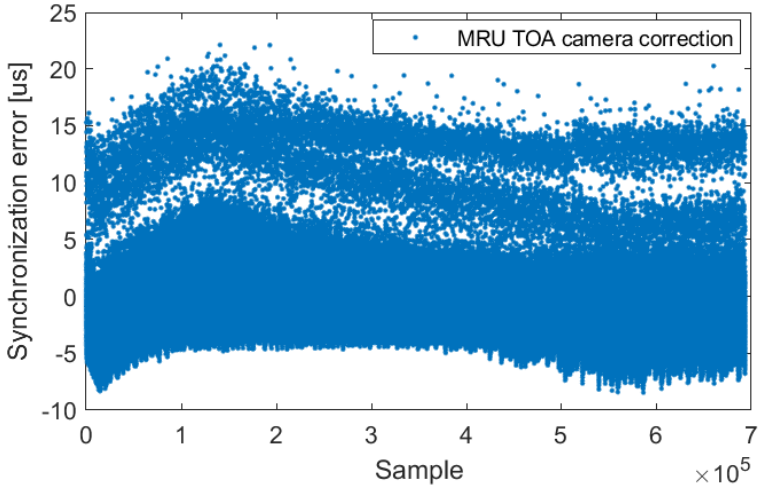


Figure 5.16: Synchronization error when using the camera correction algorithm from Equation (5.7)

6

Discussion

To see how the results from Chapter 5 could affect the georeferencing accuracy an example of a potential flight is presented. Assume that the UAV dynamics is as in Table 5.7, flying at an altitude of 400 m, with a linear velocity of 5m/s and an angular velocity of $180^\circ/\text{s}$. Assume that the UAV has been stationary for a while and as such has very accurate position and attitude measurements. We also assume that the UAV reaches its linear and angular velocity instantaneously. The camera takes a picture every 0.25 seconds and the GNSS receiver receives a signal every second. Then consider this scenario 2000 seconds into the flight. The GNSS receiver has just received a signal and then flies for 0.75 seconds horizontally at 5m/s and then rotates for 0.25 seconds at $180^\circ/\text{s}$. This scenario can be seen in Figure 6.1.

If we use the synchronization errors from the ADIS IMU TOV (top plot in Figure 4.6) for the ADIS and the synchronization error from the SenTiBoard (bottom plot in Figure 4.6) for the camera at 2000 seconds we get the following values. The SenTiBoard synchronization error is at -7 ms while the ADIS synchronization error is at 23 ms which means that the SenTiBoard clock is 7 ms behind the true time and the ADIS clock is 23 ms ahead of the actual time. Adding this together a timestamp with the same time logged by the camera happens 30 ms later than a timestamp with the same time logged by the ADIS IMU.

Using this information with the scenario presented gives the following errors. The reported position that picture 1,2 and 3 was taken at will be off by 15 cm which is given by:

$$5\text{m/s} \cdot 30 \cdot 10^{-3}s = 0.15\text{m}$$

Picture 4 will be much worse as the error from the rotation is proportional to the distance from the ground. In this case the angle the picture is taken will be off by 5.4° which is given by:

$$180^\circ/\text{s} \cdot 30 \cdot 10^{-3}s = 5.4^\circ$$

To figure out how much this will affect the position we can use basic trigonometry. The UAV forms a right triangle with the ground where one side is 400m and the angle between

that side and the hypotenuse is what was just calculated. Using this we can calculate the length of the other side by using the equation

$$b = a * \tan(\beta) \tag{6.1}$$

where a is the known side and β is the known angle. To get the error we then compute the difference between the length of what the side would be with the correct angle of 45 degrees and subtract the length of the side with the actual calculated angle. If we put in numbers from earlier this results in an error of 69.1 m. This is found from:

$$400\text{m} \cdot \tan(45) - 400\text{m} \cdot \tan(45 - 5.4) = 69.1\text{m}$$

To see how much this error has improved by using the algorithms in Chapter 5 we can use the RMS of the synchronization errors from algorithm 3 as seen in Table 5.3. For the ADIS this is $9.6 \cdot 10^{-7}$s. We assume that the camera is corrected as in (5.8) and has about the same error as the MRU had in Figure 5.16 which gives a synchronization error of $3 \cdot 10^{-6}$s. Adding this together gives an error of about $2 \cdot 10^{-6}$s which means that the same time logged by the camera happens 2μs later than a timestamp with the same time logged by the ADIS IMU.

Performing the same calculations as before gives us a positional error of 10μm for picture 1,2 and 3 given by:

$$5\text{m/s} \cdot 2 \cdot 10^{-6} = 1 \cdot 10^{-5}\text{m}$$

For picture 4 the angle will be off by $3.6^\circ \cdot 10^{-4}$ given by:

$$180^\circ/\text{s} \cdot 2 \cdot 10^{-6}\text{s} = 3.6^\circ \cdot 10^{-4}$$

This results in a georeferencing error of 52 cm. This is found from:

$$400\text{m} \cdot \tan(45) - 400\text{m} \cdot \tan(45 - 3.6 \cdot 10^{-4}) = 0.52\text{m}$$

This shows that for this simple example the georeferencing errors due to synchronization errors when flying straight is improved from 15 cm to 10μm which is an improvement of a factor of 15000 while the errors when rotating is improved from 69.1 m to 52 cm which is an improvement of a factor of 100. This only accounts for the errors introduced due to synchronization errors and does not account for the errors introduced by the sensor measurements.

Since the algorithms were not tested in real-time and were not used in a flight it is difficult to know whether or not the system would work and how much of a difference the increased synchronization would have on the georeferencing accuracy. The Syncline model is shown to be a good model for direct georeferencing and therefore an increase in the synchronization accuracy should give increases in georeferencing accuracy [5]. Comparing the results from the Syncline with and without error correction gives an improvement from $2.5 \cdot 10^{-2}$s to $9.6 \cdot 10^{-7}$s when using no error correction and the third algorithm. This is an improvement of a factor of 25000 in regards to synchronization accuracy. The Syncline model also reports an increase in estimation accuracy from $5 \cdot 10^{-4}$ to 2 which is an improvement of a factor of 4000. This is equivalent to the errors in the measurement

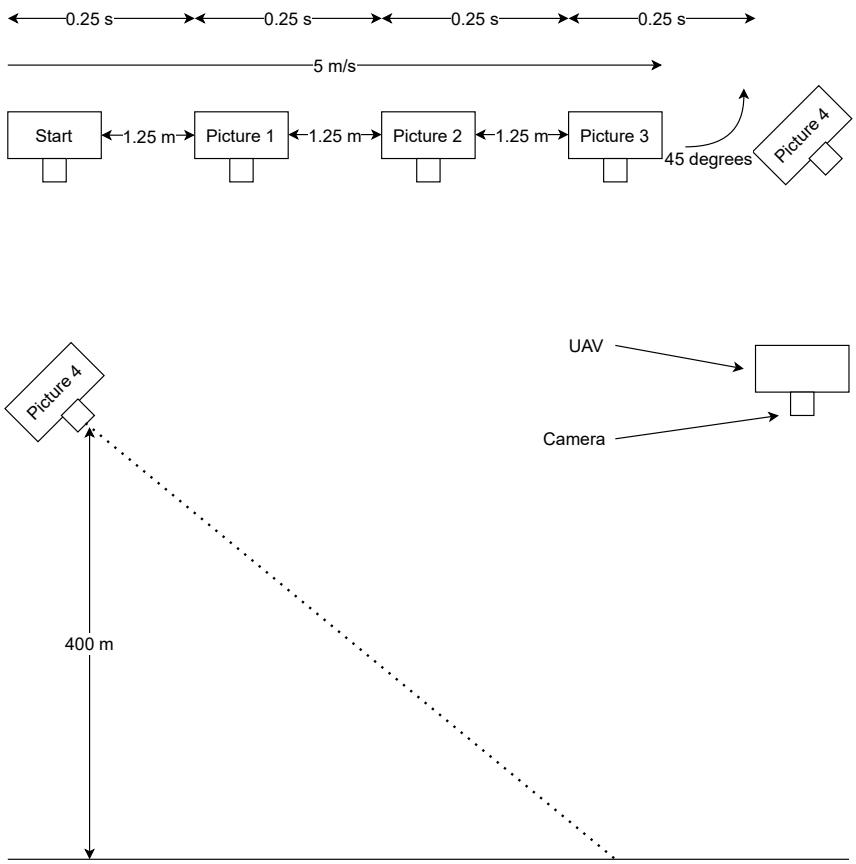


Figure 6.1: Example flight where the UAV flies at an altitude of 400m for 0.75 seconds at 5 m/s before rotating for 0.25 seconds at $180^\circ/\text{s}$. It also takes a picture every 0.25 seconds.

first being 2000 m and then going down to 50 cm as the estimation accuracy is the inverse of the error.

The results from the Syncline model combined with the simple example presented here proves that theoretically these error correction algorithms should improve the georeferencing accuracy by a significant amount. In practice there are always things that work differently than expected or things that are not accounted for. Therefore, the real-time benefit of reducing the error synchronization in a complete georeferencing system by using these error correction algorithms is likely lower than what is presented here.

6.1 Limitations of the thesis

The hardware that was used for the thesis was also in use by others and due to time constraints, it was not possible to have a test flight using the algorithms from Chapter 5. The algorithms were also only evaluated after a flight had finished and not tested with the SBC

used during the flights. Therefore, the real-time viability of the algorithms have not been verified.

6.2 Further work

Implementing the algorithms on a SBC connected to the UAV and running them in real-time would be the first step moving forward. Doing this would check if the assumptions made in this thesis is correct and can also be used to see whether or not the synchronization accuracy is as good as this thesis claims.

Following that a test flight using the camera to test the georeferencing accuracy should be performed. If this is done it is important to account for all the errors that occur due to sensor measurement errors and misalignment error caused by the sensors not being placed correctly.

Fixing the synchronization errors from the camera is difficult as it has to be triggered and the data received from it can not be accurately timestamped. Using the SenTiBoard works to a certain extent, but is lacking due to the SenTiBoard clock drift as presented in Section 5.9. Another way of solving this problem could be to use another GNSS receiver which sends a signal directly to the camera to trigger it. This signal is very accurate and can be triggered with a frequency up to 10 MHz [18]. Multiple GNSS receivers are all synced to each other since they are synced to the same satellites (Section 2.1.2) and therefore the camera triggering will be synced to the other sensors.

The accuracy of the Kalman filter could also probably be improved by adding in the effect of temperature on the modelling of the clock drift. Then the temperature measurement could be used to correct the predictions. The Kalman filter can also be updated to use time varying process noise and measurement noise covariance matrices.

7

Conclusion

This thesis aimed to analyze the current timing errors present in a UAV georeferencing system and to implement algorithms to perform time synchronization that would in turn improve the georeferencing accuracy. This was achieved by looking at data from flights and correcting the sensor timestamps using timestamps from a GNSS receiver. It was showed that when using algorithms that are applicable for real-time use the timing errors due to lack of time synchronization decreased by a factor of 25000 and the positional georeferencing accuracy was improved by a factor of 4000 given the system dynamics and sensor accuracy chosen for the Syncline model. This is only a theoretical increase using arbitrary values and have not been tested during an actual flight with the algorithms implemented. The results from this thesis are, however, very promising and further testing and implementation should be performed. The code used and the algorithms presented are general and should be applicable regardless of the sensor used as long as a GNSS receiver and hardware capable of accurate timestamping is used.

Bibliography

[1] Hovedredningssentralen, "Hovedredningssentralen hrs statistikk," Accessed: 24th of April 2022. https://www.hovedredningssentralen.no/wp-content/uploads/2022/01/HRS-S-N-og-Samlet-2021-ver4.pdf.

[2] Øyvind Breivik and A. A. Allen, "An operational search and rescue model for the norwegian sea and the north sea," *Journal of Marine Systems*, vol. 69, no. 1, pp. 99–113, 2008. Maritime Rapid Environmental Assessment.

[3] H. H. Helgesen, F. S. Leira, T. H. Bryne, S. M. Albrektsen, and T. A. Johansen, "Real-time georeferencing of thermal images using small fixed-wing uavs in maritime environments," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 154, pp. 84–97, 2019.

[4] S. M. Albrektsen and T. A. Johansen, "User-configurable timing and navigation for uavs," *Sensors*, vol. 18, no. 8, p. 2468, 2018.

[5] E. R. Jellum, T. H. Bryne, T. A. Johansen, and M. Orlandić, "The syncline model - analyzing the impact of time synchronization in sensor fusion," in *2022 IEEE Conference on Control Technology and Applications (CCTA)*, IEEE, 2022. Accepted.

[6] G. Paul D., *Principles of GNSS, Inertial, and Multi-sensor Integrated Navigation Systems, Second Edition.*, vol. Second edition of *GNSS Technology and Application Series*. Artech House, 2013.

[7] X. A. Yao, "Georeferencing and geocoding," in *International Encyclopedia of Human Geography (Second Edition)* (A. Kobayashi, ed.), pp. 111–117, Oxford: Elsevier, second edition ed., 2020.

[8] P. Seidelmann and T. Fukushima, "Why new time scales?," *Astronomy and Astrophysics*, vol. 265, pp. 833–838, 1992.

[9] Consultative Committee for Time and Frequency, "Mise en pratique for the definition of the second in the si," Accessed: 5th of May 2022. https://www.bipm.org/en/publications/mises-en-pratique.

[10] D. D. McCarthy and P. K. Seidelmann, *Time: from Earth rotation to atomic physics*. Cambridge University Press, 2018. pp. 41-94.

[11] Navipedia, "Time references in gnss," Accessed: 5th of May 2022. https://gssc.esa.int/navipedia/index.php/Time_References_in_GNSS.

[12] B. Jalving and E. Berglund, "Time referencing in offshore survey systems," 2006.

[13] U-blox, "Zed-f9p - integration manual," p. 61, Accessed: 5th of May 2022. https://www.u-blox.com/en/product-resources.

[14] S. Zhao and B. Huang, "On initialization of the kalman filter," in *2017 6th international symposium on advanced control of industrial processes (AdCONIP)*, pp. 565–570, IEEE, 2017.

[15] Hardkernel, "Odroid xu4," Accessed: 4th of July 2022. https://magazine.odroid.com/odroid-xu4/.

[16] A. Devices, "Adis16490 imu," Accessed: 4th of July 2022. https://www.analog.com/en/products/adis16490.html#product-overview.

[17] K. Maritime, "minimru - data sheet," Accessed: 1st of July 2022. https://www.kongsberg.com/no/maritime/products/vessel-reference-systems/motion-and-heading-sensors/mru/minimru/.

[18] U-blox, "Zed-f9p - data sheet," p. 4, Accessed: 1st of July 2022. https://www.u-blox.com/en/product-resources.

[19] T. D. Imaging, "Genie nano c4040," Accessed: 4th of July 2022. https://www.teledynedalsa.com/en/products/imaging/cameras/genie-nano-1gige/.

[20] S. S. Technologies, "Stim300 imu," Accessed: 4th of July 2022. https://www.sensonor.com/products/inertial-measurement-units/stim300/.

[21] V. De Smedt, P. De Wit, W. Vereecken, and M. S. Steyaert, "A 66 μ w 86 ppm /° c fully-integrated 6 mhz wienbridge oscillator with a 172 db phase noise fom," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 7, pp. 1990–2001, 2009.

[22] U. Denier, "Analysis and design of an ultralow-power cmos relaxation oscillator," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 8, pp. 1973–1982, 2010.

[23] U-blox, "Zed-f9p hpg 1.32 - interface description," p. 159, Accessed: 13th of June 2022. https://www.u-blox.com/en/product-resources.

[24] X. Xuan, J. He, P. Zhai, A. Ebrahimi Basabi, and G. Liu, "Kalman filter algorithm for security of network clock synchronization in wireless sensor networks," *Mobile Information Systems*, vol. 2022, 2022.

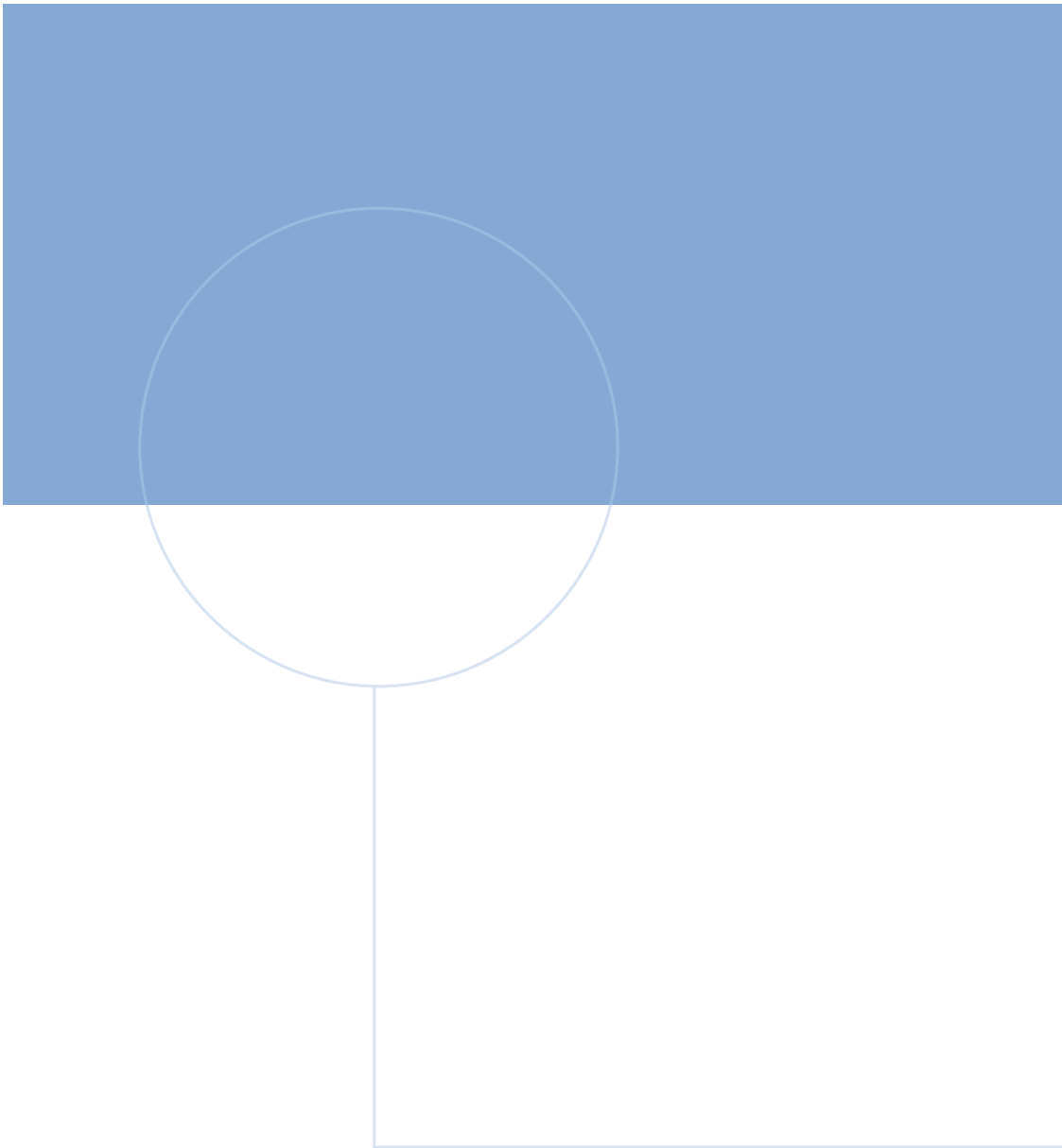
[25] O. G. Crespillo, M. Joerger, and S. Langel, "Overbounding gnss/ins integration with uncertain gnss gauss-markov error parameters," in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 481–489, 2020.

[26] T. D. Imaging, "Camera user's manual," pp. 146–147, Accessed: 4th of July 2022. https://www.stemmer-imaging.com/media/uploads/cameras/dalsa/12/122239-Teledyne-DALSA-Genie-Nano-Series-Manual.pdf.

Appendix

The zip file handed in with this project contains the following:

- The code used for extracting the variables, correcting the timestamps and plotting the data:
  - (Master_code.m) The main codefile with everything except the combined packet loss plots.
  - (Packet_loss_simulation.m) The code used for the combined plots for the packet loss. simulation
- The datasets used in the thesis.



NTNU

Norwegian University of
Science and Technology