Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology And Methodology

Spatial localization of LiDAR point clouds by sensor fusion

Supervisor:
Máté Cserép
Assistant Lecturer

Author:
Roxána Provender
Computer Science MSc

Budapest, 2020

Contents

Chapter 1

Introduction

Recently the need for creating accurate planar and three-dimensional maps of indoor and outdoor environment is increasing in fields of robotics and autonomous car industry. The advancement in the LiDAR technology made sensors capable of creating accurate dense point cloud representation of its environment regardless most weather conditions. Due to these good characteristics and decreasing cost nowadays this sensor becomes more and more popular in robotics and self-driving car industry.

The topic of this thesis is the evaluation of the fully automatized localization of a dynamically acquired point cloud in space, using the fusion of several sensors' output. The aim of the research is to present a framework that can calibrate the spatial position of a point cloud generated by a moving LiDAR sensor with the help of other sensor outputs (high precision geodetic GNSS, mobile GNSS). The framework is able to combine different inputs in order to position the generated point cloud in 3 dimensional space in a completely unattended way with the highest possible accuracy. The program heuristically weights the signal reliability of the localization sensors used and thus decides which sensor gives the best results.

The program can dynamically calibrate the position gained by this combination of sensors during its run to generate and localize the collected point cloud. The thesis details the difficulties and problems arising from the different characteristics of the different sensors and the techniques used to solve them.

The thesis specializes for outdoor measurements, as GNSS sensors mainly give a suitable signal there, but the flexibility of the presented architecture shows that it can be easily supplemented for indoor measurements by adding the appropriate

sensors or algorithms (e.g. IMU or SLAM).

The implementation is implemented in C++ using the Point Cloud Library (PCL) software library, taking advantage of its diverse collection of functions for the point cloud.

Chapter 2

Background and related work

The thesis is based on two main technologies: On Light Detection and Ranging (LiDAR) and on Global Navigation Satellite System (GNSS). In this chapter, their characteristics, operation and related concepts are described. Related work to the synchronization of the above-mentioned technologies will also be presented.

2.1 Light Detection and Ranging (LiDAR)

The *Light Detection and Ranging* (LiDAR) is a method to determine the distances between the emitting device and some kind of reflective surface, by illuminating the target with laser light and measuring the reflected light with a sensor, as shown in Fig. 2.1.

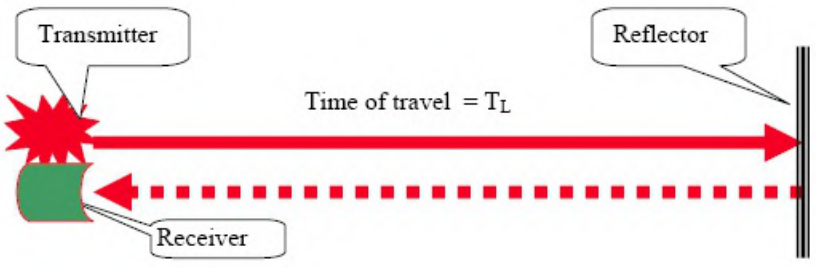


Figure 2.1: Operation of a LiDAR sensor

Source: IIT Kanpur, `http://home.iitk.ac.in/~blohani/`

The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it travelled. By repeating this process – with state of the art sensors – even more than a million of times per second, we produce an accurate, real-time 3D map of the environment. It is commonly used to make high-resolution maps, with applications in many areas, like geography, archaeology, security, robotic and also in control and navigation for self-driving cars. The method is provided by the LiDAR sensor, which is an active remote sensing system with its own source of signal. Most LiDAR systems have a single laser that fires onto a rotating mirror. In Fig. 2.2 we can see a typical example for an urban scene point cloud, created by LiDAR sensor.

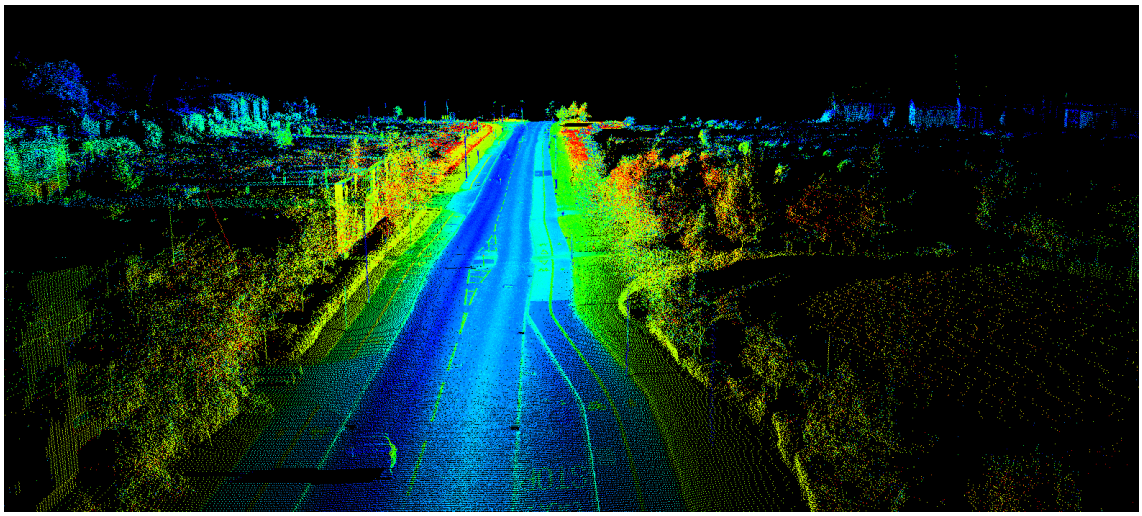


Figure 2.2: LiDAR point cloud for urban scene

Source: `www.geospatialworld.net`

2.1.1 Data acquisition

Looking at the platform with LiDAR sensor, a distinction can be made between terrestrial and airborne types. The area to be explored and the available budget plays an important role in the choice, which one to use.

Airborne LiDAR scanning (ALS)

In the case of airborne LiDAR the sensor is attached to a flight-capable device, and creates a 3D point cloud from the air. This variety is becoming more and more popular nowadays, because as we can see in Fig. 2.3, it makes it quick and easy to

map large areas. In the airborne LiDAR category, a distinction is sometimes made between high altitude and low altitude, but the main difference is a decrease in the accuracy and point density of data obtained at higher altitudes.

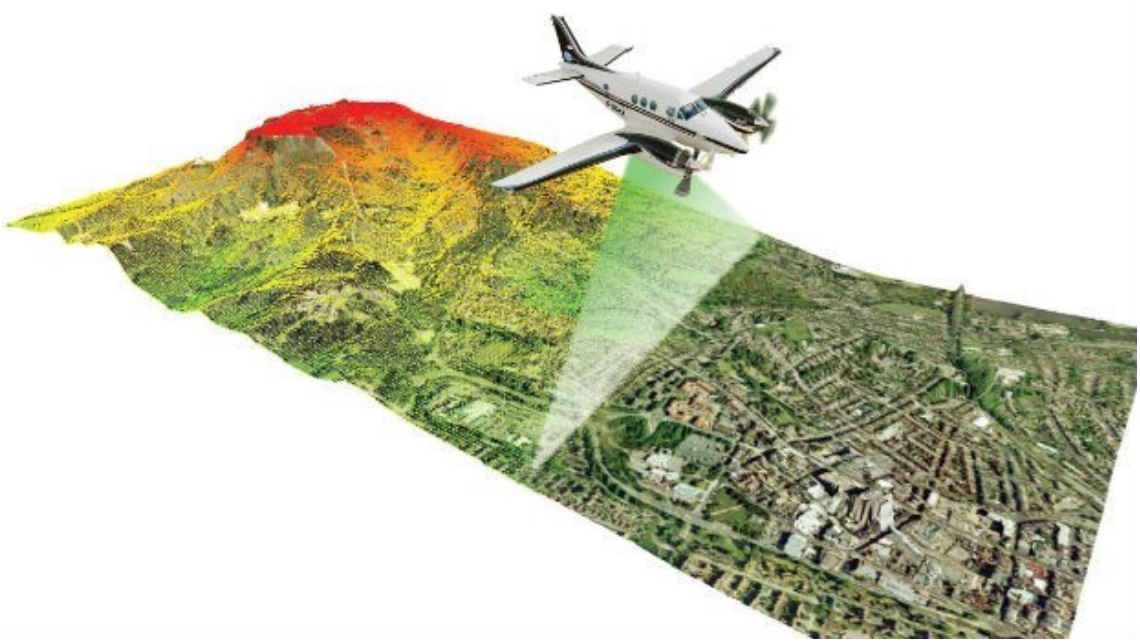


Figure 2.3: Airborne LiDAR scanning

Source: `medium.com`

Terrestrial scanning

Terrestrial applications of LiDAR (also terrestrial laser scanning) happen on the Earth's surface and can be either stationary or mobile. During stationary terrestrial scanning, as shown in Fig. 2.4, the sensor is in a fixed position. This is most commonly used as a survey method, for example in traditional topographic applications, geomorphology, and deformation analysis. In many cases, the point cloud is matched with some other digital image of the given area, in such a way that each point is colored from the pixel of the digital image that falls at the same angle. Due to this, we can get a realistic point cloud from the studied area relatively quickly and cheaply, compared to other methods.

In the case where the LiDAR sensor is mounted on a moving device, we speak about mobile scanning. In most cases, the LiDAR sensor is paired with another sensor, which is capable of detecting the change of the position, such as a GNSS or IMU sensor. This method can be used to create a large 3D point cloud from cities,

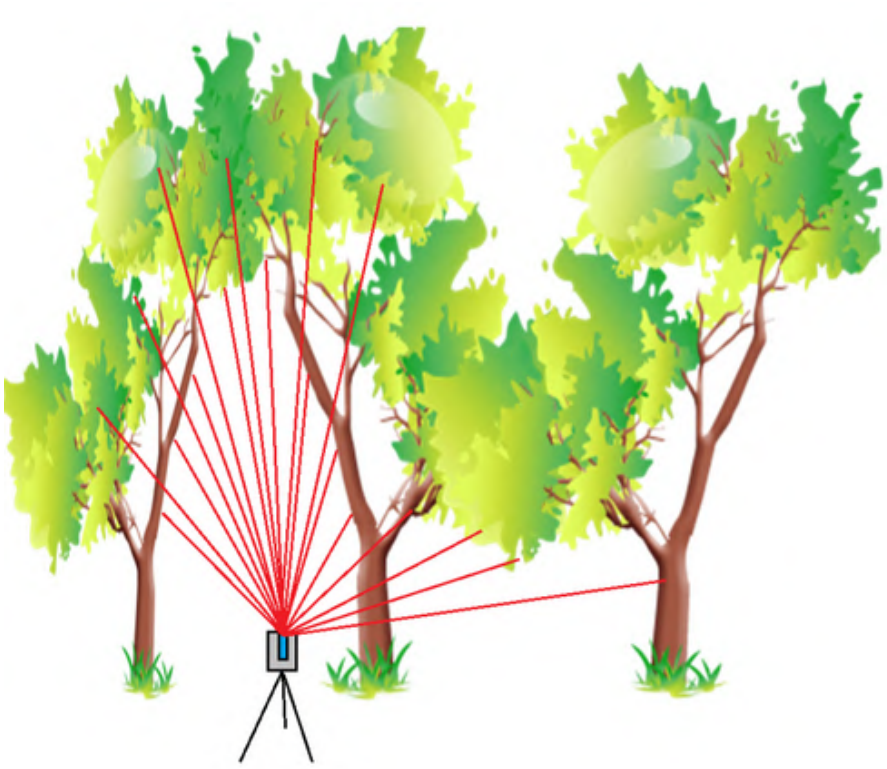


Figure 2.4: Terrestrial LiDAR scanning

Source: University of Göttingen, Forest Inventory Remote Sensing WIKI

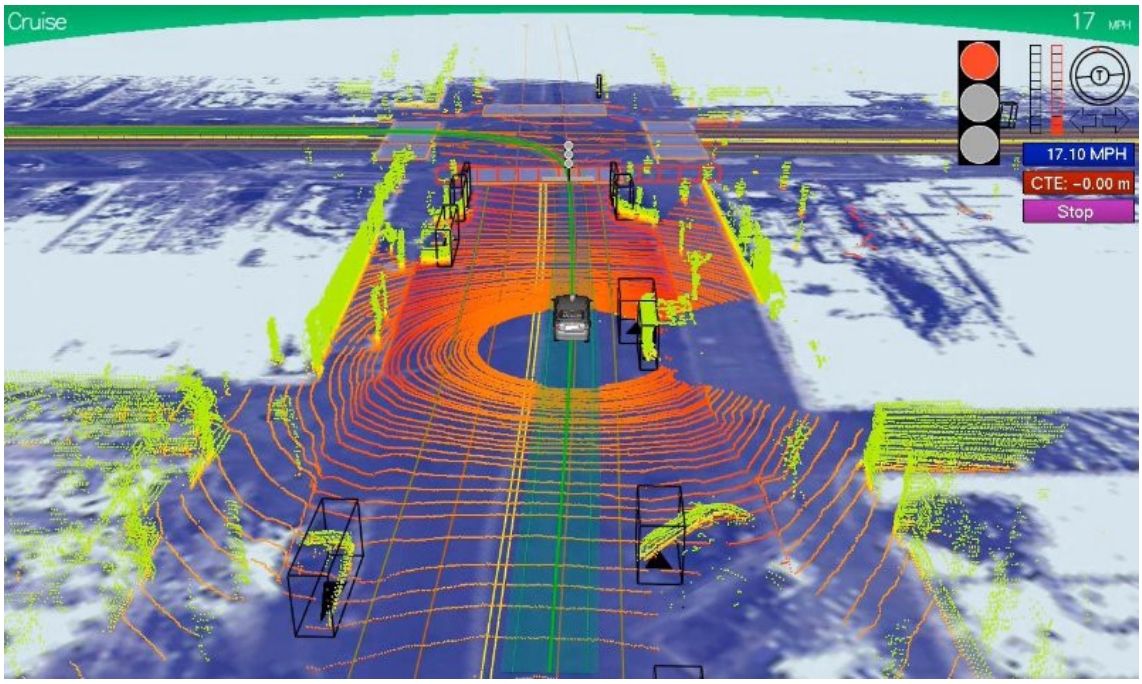


Figure 2.5: Visualization of the 3D point cloud and recognized objects by Google's autonomous self driving car. Source: Google

streets, railway lines, and so on. Today it is most commonly used in self-driving cars and robotics. The Fig. 2.5 shows a point cloud, created by a LiDAR sensor on top of a moving car. The information obtained in this way greatly helps to identify individual traffic situations. The current study also presents the application and difficulties of the moving LiDAR sensor.

2.1.2 Data formats

There are multiple open and proprietary file formats for storing point cloud generated by a LiDAR sensor with LAS/LAZ being the most widespreadly used in the field of remote sensing.

However, before the point cloud is localized and pre-processed, the raw data is often saved (or streamed) in a manufacturer dependent format like PCAP for Velodyne sensors.

PCAP

The PCAP format is designed to capture network packets. The PCAP file format is a binary format, with support for nanosecond-precision timestamps. A .pcap file is a digital data storage container, composed of captured data prefixed with a header (Global Header) and interlaced with chunks of metadata (Packet Header). The global header contains GMT offset, timestamp precision, the maximum length of captured packets (in octets), and the data link type. This information is followed by zero or more records of captured packet data (See: Fig. 2.6).

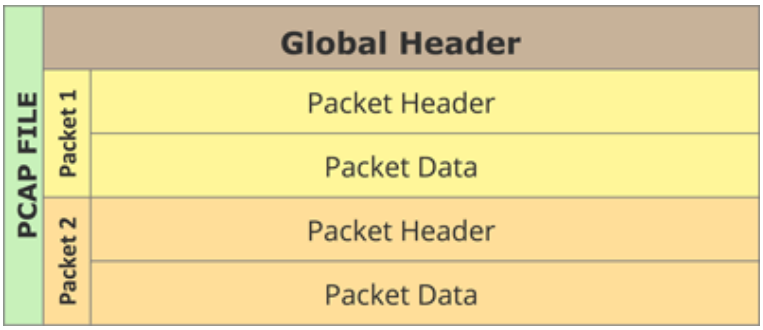


Figure 2.6: PCAP file structure

Source: `elvidence.com.au`

Each captured packet contains the UNIX epoch timestamp in seconds, the number of octets of packet saved in file, and the length of the packet. The capture packets are also known as frames [1].

In the case of the LiDAR sensor, each frame contains a point cloud for the given time. Using the PCAP file format, the entire measurement can be replayed, providing great repeatability of processing the data.

LAS/LAZ

The LAS (LASer) format [2] is a file format designed for exchanging and archiving LiDAR point cloud data. The format is widely used and it is considered as an industry standard for LiDAR data.

The format contains binary data consisting of a header block, variable length records, and point data. The points are not ordered by any attribute or spatially indexed. The role of each block is shown in Table 2.1.

Public Header Block	Describes format, number of points, extent of the point cloud and other generic data.
Variable Length Records	Any number of optional records to provide various data such as the spatial reference system used, metadata, waveform packet information and user application data.
Point Data Records	Data for each of the individual points in the point cloud, including X,Y,Z coordinates, intensity, classification (e.g. terrain or building), flight and scan data, etc.

Table 2.1: LAS Format Definition (Source: Wikipedia)

LAS projects can be very large (most often more GB) therefore compression is often used on it. LAZ [3] is a losslessly compressed variant of LAS.

2.2 Global Navigation Satellite System (GNSS)

The principle of operation of a *Global Navigation Satellite System* (GNSS) [4] is based on the fact that the current 3D position can be determined by knowing the coordinates and the distance of any three points in space. GNSS is a collective name for the navigation system, based on this principle. The best known and most used systems are:

- Navstart Global Positioning Sytem (GPS), which is a space-based radio navigation system owned by the United States Government (USG) and operated by the United States Air Force.
- GLONASS, which is developed by the Russian Federation, mostly used as a supplement of GPS to improve positioning in high latitudes in the civil sphere, but for military usage it is a true "rival" for GPS.
- Galileo, the Europe's Global Navigation Satellite System, made available by the European Union in 2016 after 5 years of launching and orbiting the satellites. Galileo is the only completely civilian and commercial system without military purposes among these four.
- BeiDou, which is a Chinese satellite navigation system. Originally only offering services to inland Chinese users, then to the Asia-Pacific region, lately it started providing global services in 2018.

Among the four mentioned systems, only GPS and GLONASS provide true global constellation and therefore coverage, while Galileo and BeiDou aim to reach it in 2020. The most used system in the western world is the GPS. The main source of GNSS receivers used in this thesis is also the GPS signal.

GPS signals include ranging signals, used to measure the distance to the satellite, and navigation messages [5]. The original GPS design contains two ranging codes: The Coarse/Acquisition (C/A) code, which is freely available to the public, and the Restricted Precision (P) code, usually reserved for military applications. Until 2000, signals received by civilians were intentionally disturbed – called as *Selective Availability (SA)* and caused $\sim 50m$ horizontal and $\sim 100m$ verodoical error – to prevent perfect accuracy. However, the C/A code was far from perfect in 2000,

even without SA. Without the new signals introduced since then, the receivers had accuracy around $\sim 5m$.

Nowadays, there are four signals available for civilian use. In order of date of introduction, these are: L1 C/A, L2C, L5 and L1C. L1 C/A, which is broadcasted by all satellites, is also called legacy signal. The other 2 (modernized) signals are called the second and third civil signals and are not broadcast by all satellites, but by the ones launched later in time. With the new signals a modern GPS receiver can achieve $\sim 0.5m$ precision.

The signals can be distorted by a variety of errors, such as *ephemeris errors* (satellites are affected by forces such as the Earth's gravitational field or different solar winds) or *propagation errors* (the GNSS signal must pass through the Earth's atmosphere, which usually delays and distorts the signal) [6].

Of course, as with any other sensor, accuracy is affected by the type and the price of the GNSS sensor. The cheaper sensors (such as those used in mobiles) typically have an accuracy of $\sim 2 - 4m$. In order to achieve centimeter accuracy, it is necessary to use a Differential GNSS sensor (DGNSS) [7]. DGNSS systems further develop the signal from satellites by using of a network of ground-based reference stations, which enable the broadcasting of differential information to the user to improve the accuracy of the position.

2.2.1 EOV

In Hungary, it is possible to describe GNSS coordinates using the EOV [8] projection system (EOV is the abbreviation of *Egységes Országos Vetület*, meaning *Uniform National Projection*). The EOV is the projection system of surveying maps in Hungary and was introduced in 1975. The whole territory of the country is covered by a cylindrical projection. The orientation of the coordinate axes is NE, that is, the direction of the positive X points to the north, and the direction of the positive Y points to the east. To simplify the calculations, the coordinate axes were shifted parallel to each other by 200,000 meters in the X direction and 650,000 meters in the Y direction so that the entire area of the country fell into the first coordinate quarter. The value of the X coordinate is always less than 400,000 m and the Y

coordinate is always greater than 400,000 m to provide protection against possible exchange of coordinates.

2.2.2 Kalman filter

One of the biggest problems with using a GNSS sensor is that the sensor might lose signal and we will miss multiple coordinates. A popular solution to minimize the error of various sensor signals is to use the Kalman filter.

The filter is named after Rudolf E. Kálmán, who was one of the primary developers of the method [9]. The essence of the algorithm is that it is capable of optimally estimating a series of measurements from a moving noisy system. This method provides much more accurate information about the object being investigated than if only one measurement was made. It has many applications and is commonly used in navigation and guidance.

The algorithm works in a two-step process:

Prediction : Calculation (estimation) of the current state variables, together with the uncertainties.

Update : Recursively using the previously calculated state and the current measurements with a weighted average, to achieve better results.

The Kalman filter has to decide which state to rely on, predicted state or measured state. This will be decided based on the process and measurement error. For better understanding, the two states can be represented as two Gaussian signal. As shown on Fig. 2.7 the predicted state has a wider curve, so it contains more uncertainty than the original measurement. The Kalman filter takes this uncertainty into account and shifts the resultant signal towards relatively more certain signal.

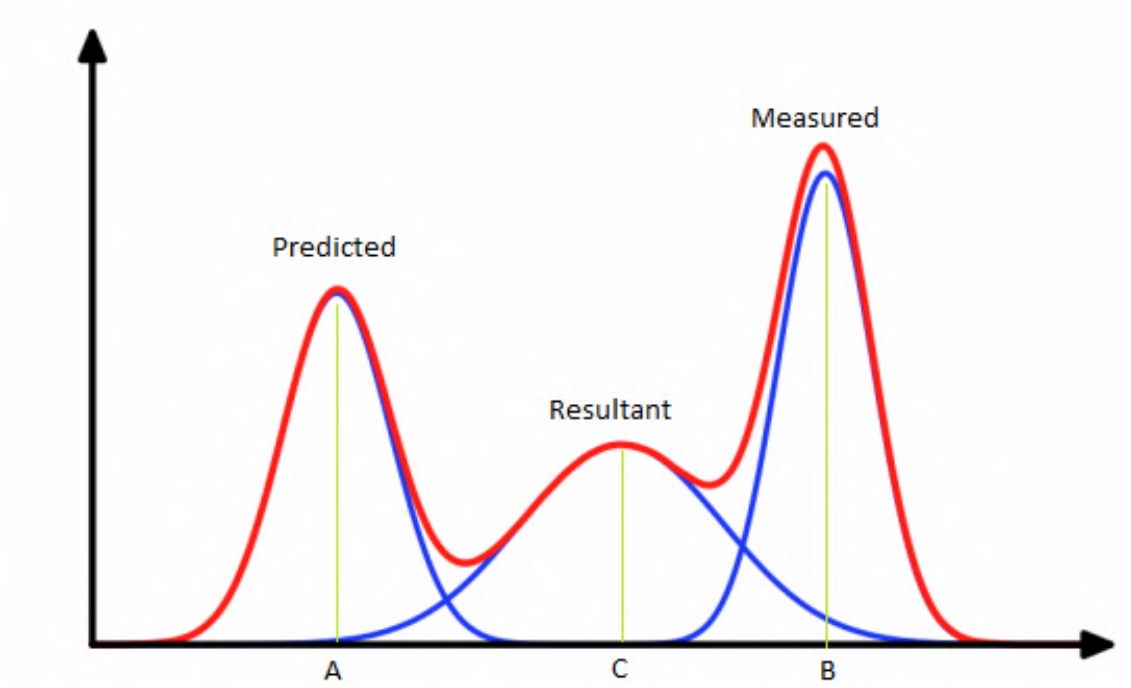


Figure 2.7: Optimal estimation

Source: `medium.com`

Theoretically, the basic assumption of the Kalman filter is that the system under investigation is a linear dynamic system and that all error functions and measurements have a normal distribution.

In order to use the Kalman filter to estimate the internal state of a process, it must be modeled within the Kalman filter framework. The model is represented formally in Eq. 2.1.

$$x_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{2.1}$$

where

- $\mathbf{F}_k$:The state transition model applied to the previous state $\mathbf{x}_{k-1}$.
- $\mathbf{B}_k$: The control-input model which is applied to the control vector $\mathbf{u}_k$.
- $\mathbf{w}_k$: Process noise which is assumed to be extracted from the zero mean multivariate normal distribution $\mathcal{N}$, with covariance, $\mathbf{Q}_k$: $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k)$.

As it seen, the state at time k is calculated from the state at $k-1$.

The other important element of the model is the value of real measurement at the time k, represented as:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{2.2}$$

where

- $\mathbf{H}_k$: The observation model that maps the real state space to the observed one.
- $\mathbf{v}_k$: The observation noise which is assumed to be zero mean Gaussian white noise with covariance $\mathbf{R}_k$: $\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k)$.

$\mathbf{u}_k$ and $\mathbf{v}_k$ are mutually independent.

The error equations are:

- $\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k$
- $\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathsf{T}} + \mathbf{Q}_k$

where $k \mid k-1$ means estimating k based on $k-1$. The first is the vector error, and the second one the error covariance matrix. With this two equations we can predict the accuracy the method and can use the difference to update. A summary of the process can be found on Fig. 2.8.

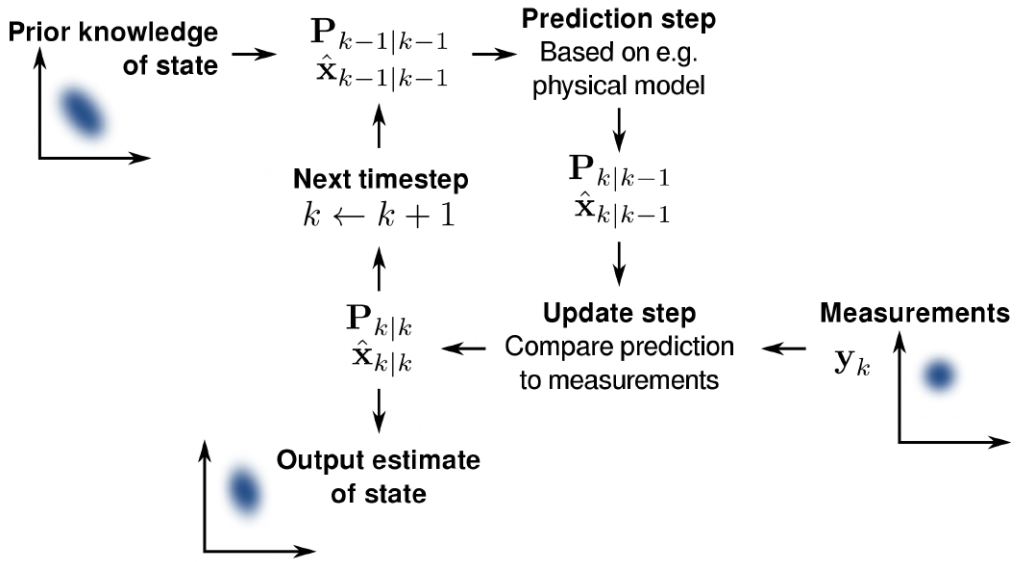


Figure 2.8: Basic concept of Kalman filter. Source: Wikipedia

There are many extensions and generalizations to the Kalman filter, such as the *Extended Kalman filter*, which is the non-linear version of the filter and it is widely used for the position estimation in GNSS receivers.

In the case of Extended Kalman filter state transition and observation models do not have to be linear functions of state, but rather can be differentiable functions:

$$\begin{aligned} \boldsymbol{x}_k &= f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \\ \boldsymbol{z}_k &= h(\boldsymbol{x}_k) + \boldsymbol{v}_k \end{aligned} \tag{2.3}$$

There have been also attempts to further improve the filter. Julier and Uhlmann introduced a new approach, called *Unscented Kalman Filter* [10]. In this paper, a new linear estimate is produced that provides performance equivalent to the Kalman filter for linear systems, yet is generalises into nonlinear systems without the linearization steps required by the EKF. The fundamental component of this filter is the unscented transformation which uses a set of appropriately chosen weighted points to parameterize the means and covariances of probability distributions.

Petovello et al. also tried to improve the filter with a new approach of the implementation [11]. The algorithm differs from previous implementations in that it does not suffer from numerical problems and does not contain inherent time latency or require reinterpretation of Kalman filter parameters. Their tests showed that the new method contains more realistic, thus more useful covariance and error information.

2.3 Sensor fusion

While a LiDAR sensor can provide a rich, real-time 3D map, but the origo of the map is always be the sensor, so it only provides static data collection. By moving the device, the collected data will be uninterpretable. The idea is to combine the output of different sensors to calculate the position of the current point cloud relative to existing cloud. The most common approach to solve this issue is to use a GNSS sensor.

During sensor fusion, the main problem is caused by the pairing of the outputs of different sensors. One aspect of this is the timestamp synchronization, so we know the precise time that each event occurred and can assign the different sensor signals to each others. Differences between coordinate systems is also an issue, as is the

already mentioned poor or missing signal of sensors in the case of GNSS. This issues has to be solved in order to achieve good results.

The fusion of the LiDAR sensor and the a GNSS sensor output has been the subject of several attempts in recent years. García-Moreno and Gonzalez-Barbosa combine an HDL-64 Velodyne LiDAR sensor with an GNSS ProMark3 with RTK technology to construct urban scenes [12]. The sensors are synchronized through their internal clocks using the Lamport [13] algorithm, and the position of the LiDAR acquisition is given by the GNSS data. The difference between coordinate system was resolved by creating an appropriate rotation matrix based on *Euler angles* [14]. To calculate missed coordinates, the article used a weighted interpolation based on existing GNSS coordinates and missed time, instead of Kalman filter. Their results are encouraging, but they plan to use the SLAM algorithm to further improve them.

Trafina used real-time point cloud reconstruction and encountered a problem with the delay on the hardware line and a variable delay with which the processor registers the information from different sensors [15]. The problem was solved by calculating the traffic delay and adding it to each timestamp. The author combine the Velodyne LiDAR VL16 sensor with an 3DM- GX4-45 from Lord Microstrain IMU sensor and an RTK (Real Time Kinematic) GNSS. The IMU sensor also included a built-in Kalmam filter. In his tests, the direction of travel was calculated using the IMU sensor and the translation was calculated using GNSS sensor. During the experiments, it was a problem that many times the IMU sensor did not return the real angle despite the built-in Kalman filter, due to the disturbing environmental influences. This problem has only been solved manually, but an automated method is planned for the future.

Yanbin Gao and Noureldin used GNSS and LiDAR as support systems as an alternative to provide regular corrections of INS in different environments [16]. A quaternary-based error model was used to merge information about the multi-sensor. Beside the sensors an innovative hybrid scan matching algorithm also was used. For resolve the issue of coordinate systems, an *quartation-based* [17] rotation matrix was used. The authors chose this solution because of its computational efficiency and lack of singularities. They started their experiment outdoors where the GNSS still provided a suitable signal. Moving indoors, the role of GNSS was taken over by an

INS sensor. Experimental results show that metric navigation accuracy and accurate position angle estimation can be performed over the entire orbit, including outdoor and indoor environments.

In their work, Qian et al. performed a sensor fusion in a wooded environment [18] between LiDAR based SLAM algorithms and GNSS receiver. The crown of trees greatly impairs GNSS accuracy by shading satellites. The use of a Kalman filter has greatly helped to reduce the resulting errors.

Chapter 3

Analysis and specification

3.1 Used devices

3.1.1 Velodyne LiDAR VLP-16

For my research I used the small and compact VLP-16 LiDAR sensor from Velodyne (see: Fig. 3.1). The sensor uses a rotating head featuring 16 semiconductor lasers, each firing approximately 300,0000 points per second in single return mode and 600,0000 points per second in dual return mode, providing in real-time a rich set of 3D point data.

Figure 3.1: Velodyne VLP-16 LiDAR sensor (Source: Velodyne)

Each of the lasers has its own dedicated detector, which closes a predetermined vertical angle to provide a 30° – from -15° to 15° – field of view [19].

The measurements range is close to 100m, and the range accuracy is up to $\pm 3cm$. With all this feature together, the sensor provide a 360° panorama view.

3.1.2 Stonex Survey S9III Plus

To record the displacement, a high precision geodetic DGNSS, Stonex Survey S9III Plus was used, which is a multiconstellation receiver [20]. The sensor can be seen on Fig. 3.2. It has been designed to receive and manage GNSS signals from Navstar GPS, Glonass, Galileo and other navigation systems. In addition to the coordinates in the WGS 84 format, the sensor can also return the coordinates interpreted in the EOV system.

Figure 3.2: Stonex S9III Plus sensor. Source: Stonex

The sensor achieves high accuracy (up to centimeter), but is quite sensitive to external influences such as tall buildings or overhanging tree branches. For this reason, in addition to the Stonex sensor, I also used the GNSS of an Android device,

which, although not capable of such accuracy, is less sensitive to the aforementioned effects.

3.1.3 Mobile GNSS

For mobile measurements, I used a Samsung A8 (model 2018) device with an Exynos 7885 chipset that includes the integrated localization chip [21]. It is capable of receiving GNSS, GLONASS, Gailelo and BeiDuo satellite signals. In terms of accuracy, 2-4 meters is typical, which is not even close to the accuracy of the Stonex sensor, but it is also able to receive signals in shaded areas. The coordinates were recorded in the raw format, the mobile did not apply Kalman filter to it.

3.2 Calibration of the sensors

During the measurements, it was important to properly calibrate and align the sensors. Since the Stonex sensor is highly sensitive for any objects covering the receiver unit, it had to be located at the top so that the other sensors would not block its signal. It was also important that the LiDAR sensor always looked in the direction of travel to make the direction of the Y-axis clear – which will be important when synchronizing coordinate systems, detailed in Sec. 3.4.1. The distance between the individual devices also had to be fixed. I also had to provide power to LiDAR and a computer (a laptop) to read information about it. Of course, it was also important that the devices placed in this way could be moved easily and evenly. I have tried several methods to achieve the following points. The sensors were located on a push cart dolly in different ways, as shown in Fig. 3.3.

3.3 Data acquisition

Important part of the measurements was the selection of the right location of data acquisition. As the GNSS sensors are not really suitable for indoor measurement, I mainly performed outdoor tests.

For a stable point cloud of the LiDAR sensor, it was important to choose a relatively even ground. The accuracy of the GNSS sensor signal is better, if the road

Figure 3.3: Placement of sensors for measurement

is not surrounded by many tall buildings or trees whose canopy reach over the road. These requirements were difficult to meet simultaneously. Most of the open spaces have cobblestones, which results an overly noisy point cloud. The roadway, where I could also have provided open space, would also have caused too much vibration due to the large asphalt grains. The most suitable sites were recently built sidewalks that provided relatively even ground. However, it was difficult not to be shaded by a tall building or tree. The Figure 3.4 shows one of the measurement locations where these the requirements met relatively well.

However, even in the case of relatively smooth ground, there were shifts in the line of LiDAR measurement. A more stable foundation would be needed to achieve more accurate results.

Figure 3.4: One of the location of measurements

3.4 Methodology

The essence of my algorithm is based on combining the outputs of different GNSS sensors and the output of the LiDAR sensor so that the point clouds provided by the LiDAR sensor can be properly transformed. The concepts of this approach and the synchronization of sensor results are presented below.

3.4.1 GNSS based localization

In the case of the sensor fusion, it is important that the individual dimensions of the devices are properly coordinated.

Synchronization of time

One of the important part is that the signal given by each sensor can be matched. For this I use the UNIX epoch time. However, obtaining the necessary information from each sensor was not straightforward.

Each frame of the LiDAR sensor at least with the implementation library used, does not show the exact UNIX time, but only the seconds that have elapsed since

the sensor was turned on. However the information was known using WireShark [1]. For this reason, it was necessary to specify a start time for each measurement.

In case of the Stonex GNSS sensor, a complete discharge could the sensor clock by several hours or even days off, so it had to be recalibrated before any measurement could take place.

Synchronization of coordinate systems

Another important aspect is that the translations and rotations in the coordinate system of each sensor can be matched to each other.

The coordinate system of the GNSS sensors is obviously determined by the applied geodetic datum. Basically all devices can provide the position according to the *World Geodetic System* (WGS84, also known as EPSG:4326[2]), and more advanced tools can produce the position in a custom coordinate reference system, like EOV introduced in Sec. 2.2.1. These are fixed for each measurement.

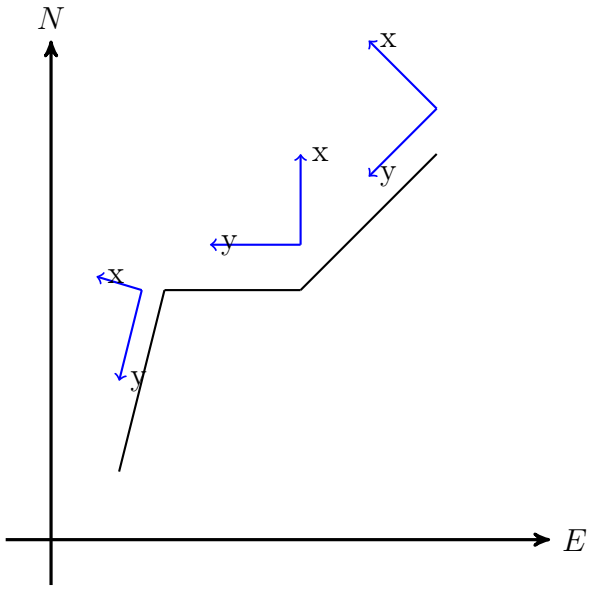


Figure 3.5: The GNSS (black) and LiDAR (blue) coordinate system relative to each other

In the case of LiDAR, the axes are always determined by the location of the sensor, so the Y axis always faces ahead. The relationship between the two coordinate systems is shown on Fig. 3.5. Since the translation and rotations are measured in the

[1] `https://www.wireshark.org`

[2] `https://spatialreference.org/ref/epsg/4326/`

GNSS coordinate system, it has become necessary to apply a rotation between the two coordinate systems, so that the calculated values could be transformed. There are several approaches to achieve this, for example *Euler angles* or *quaternion* based rotation matrix. In this work I used the Euler angles based approach.

To properly determine the direction during the measurements I assumed that the front of the LiDAR always looked in the direction of travel, and that the progress was straight for the first few meters.

The appropriate transformation of the point cloud based on the values calculated using GNSS coordinates is detailed in Sec. 3.4.3.

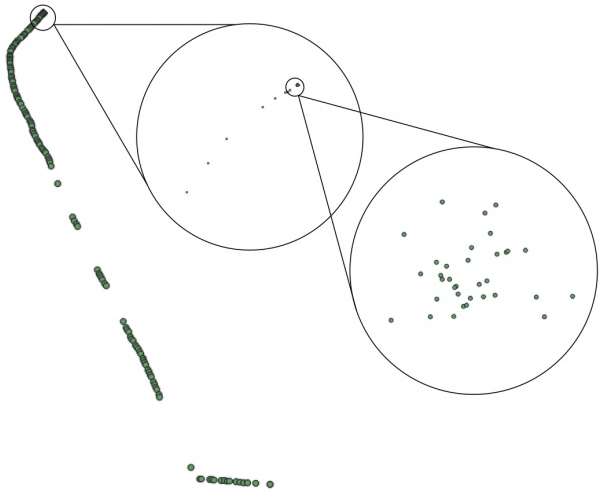
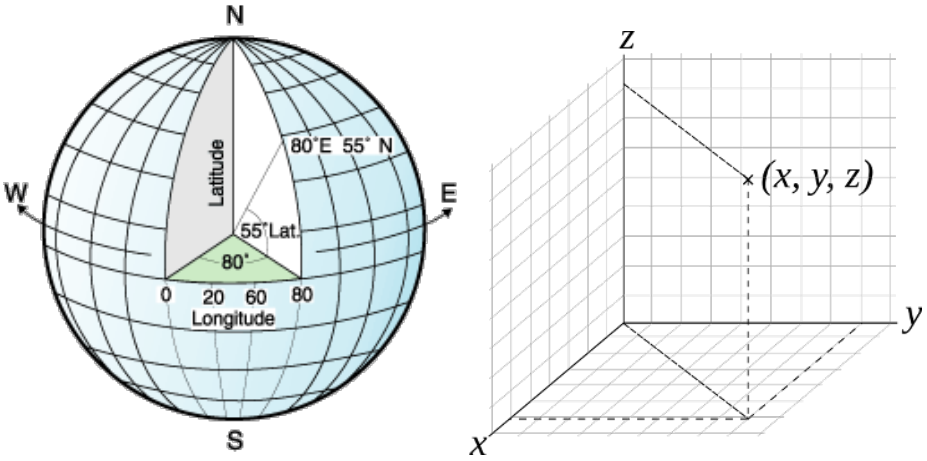


Figure 3.6: Noisy points at the beginning of the measurement

When defining transformations, it was a problem that at the beginning of the measurement, when the GNSS sensor is stationary or there is not much displacement, the distance between the recorded points is small (in the order of millimeters), but their standard deviation is relatively large. The problem is presented on Fig. 3.6. Because of this, these points had to be skipped (even if some of them might already belong to one of the initial LiDAR frames),and the first coordinate is selected by the fact, that the translation between this one and the following one be at least 0.09 meters.

Of course, not only the direction of the coordinate systems, but also their type differed for the GNSS and LiDAR sensor. The GNSS represents the latitude and longitude coordinates in a *Geographic coordinate system*, while the LiDAR sensor



(a) Geographical coordinate system. Source: IBM

(b) Cartesian coordinate system. Source: `study.com`

Figure 3.7: The different coordinate systems

system is based on the traditional *Cartesian coordinate system* (shown in Fig. 3.7). This problem was solved by calculating the translation and rotation between the coordinates with the appropriate formulas, as described in the Sec. 3.4.1.

Note. *This is not a problem for EOV coordinates, as it also uses the Cartesian coordinate system.*

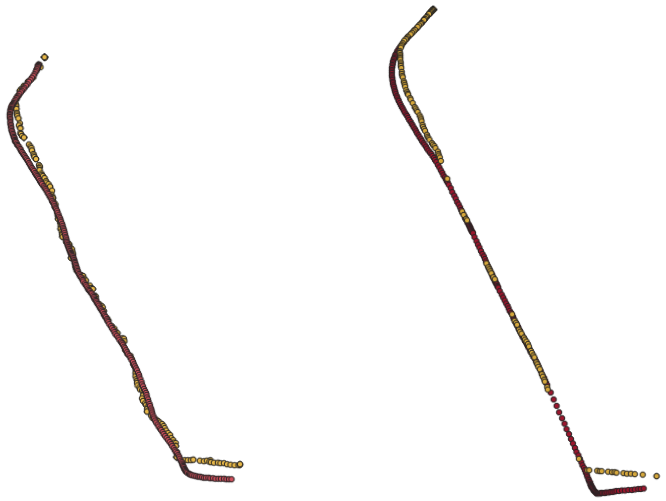
Preprocessing the coordinates

The coordinates provided by the mobile and GNSS sensors were processed separately, but using the same methods. In both cases, the coordinates were preprocessed with Kalman filter, based on thesis [22], where the author used Kalman filter for mobile GNSS sensor and described a good parametrization of the filter.

In my thesis, the Kalman filter has also been used mainly to correct the more inaccurate signal of the mobile GNSS sensor and to fill missing coordinates due to the sensitivity of the Stonex sensor.

With the help of this, it is ensured that even if a measurement is missed, a coordinate will fall every second during the measurement.

Fig. 3.8 (a) shows that the scattered coordinates of the mobile GNSS sensor have become more uniform due to the filter. Figure (b) shows that although the measurement was missing for several meters, the omitted sections could be bridged



(a) Coordinates corrected by Kalman filter

(b) Coordinates completed by Kalman filter

Figure 3.8: GNSS coordinates corrected and completed by Kalman filter

with the help of the filter. For both figure, the original path is shown in yellow and the path given by the filter is shown in red.

The Kalman filter was only run after leaving the previously mentioned initial noisy coordinates, thus achieving more accurate results. With the corrected coordinates and the properly synchronized dimensions, everything is given to calculate the changes between the individual coordinates.

Matching LiDAR frames and coordinates

As mentioned earlier, I used UNIX epoch time to match each frames and coordinates. The essence of the algorithm is to find for each frame the coordinate that is closest to it in time. To do this, it is necessary to define a function that assigns to each frame the right coordinate.

Let $F = (f_1, f_2, ..., f_n)$ be the sequence of LiDAR frames. Let $C = (c_1, c_2, ..., c_m)$ be the sequence of measured GNSS coordinates.

Let's define the

$$match : D_F \rightarrow D_C$$

partial function, which finds the appropriate GNSS measurement for all LiDAR frames, as described above. Formally:

$$match(i) = \begin{cases} c_j & \text{if } \exists j \in D_C : c_j.time \leq f_i.time \wedge c_{j+1}.time > f_i.time \wedge \\ & |c_j.time - f_i.time| \leq |c_{j+1}.time - f_i.time| \\ c_{j+1} & \text{if } \exists j \in D_C : c_j.time \leq f_i.time \wedge c_{j+1}.time > f_i.time \wedge \\ & |c_j.time - f_i.time| > |c_{j+1}.time - f_i.time| \\ \perp & \text{if } \nexists j \in D_C : c_j.time \leq f_i.time \wedge c_{j+1}.time > f_i.time \end{cases} \tag{3.1}$$

As we can see, if we do not find a coordinate whose time would be exactly the same as the time of the examined frame, then the function takes the two coordinates whose time is even smaller and already bigger as the examined moment. Of the two coordinates, the one with the smaller time difference is selected.

Calculation of the transformations

Once we have obtained which frame is paired with which coordinate, the next step is to determine what changes have taken place since the previous pair. In the case of mobile GNSS, this can only be determined from the coordinates in the WGS 84 format, while in the case of the Stonex sensor, we can choose to also the EOV coordinates instead.

In the case of *geographic coordinate system*, the translation in meters between two coordinates can be calculated using the help of the *Haversine formula* [23], which determines the great-circle distance between two points on a sphere. Formally:

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{3.2}$$

where R is the radius of the Earth, φ is the latitude, and λ is the longitude coordinates.

In order to be able to determine the displacement separately horizontally and vertically, I use the trick to fix the latitude coordinates in the former case and the longitude coordinates in the latter case to the coordinates of the first point.

$$\begin{aligned}\Delta x &= 2R \arcsin\left(\sqrt{\sin^2(0) + 2\cos(\varphi_1)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \\ \Delta y &= 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2(0)}\right)\end{aligned} \tag{3.3}$$

In the case of EOV coordinates, it is not necessary to use the *haversine* formula. In this case, the displacements can be calculated simply by subtracting the corresponding coordinate pairs from each other:

$$\begin{aligned}\Delta x &= x_2 - x_1 \\ \Delta y &= y_2 - y_1\end{aligned} \tag{3.4}$$

To apply the offsets correctly to the axes, we need to calculate the rotation angle between the two coordinate. This is done by calculate the angle between the starting coordinate y axis and the line connecting the start and the destination coordinate. The angle of rotation between two coordinates is determined in radians using the formula below:

$$\phi = \begin{cases} \frac{\pi}{2} & \text{if } \Delta x > 0 \wedge \Delta y = 0 \\ -\frac{\pi}{2} & \text{if } \Delta x < 0 \wedge \Delta y = 0 \\ \pi - \arctan(\frac{\Delta x}{\Delta y}) & \text{if } \Delta x <> 0 \wedge \Delta y <> 0 \\ 0 & \text{if } \Delta x = 0 \wedge \Delta y > 0 \\ \pi & \text{if } \Delta x = 0 \wedge \Delta y < 0 \\ \perp & otherwise \end{cases} \tag{3.5}$$

When calculating the angle, it is important to keep in mind the fact that the LiDAR frames always look ahead in the direction of travel. Therefore, by applying the above formula in each case, we can obtain how much the given frame needs to be rotated in order for it to face in the correct direction in the GNSS coordinate system. The required angle is shown in Fig. 3.9.

If the distance did not increase between the latitudes, then we went either straight to the East or West, therefore it is necessary to rotate the given frame by $\frac{\pi}{2}$ ($\pm 90°$) from its original position. If the distance has not increased between the longitudes, we have either moved towards the North, in which case no rotation is required, or we

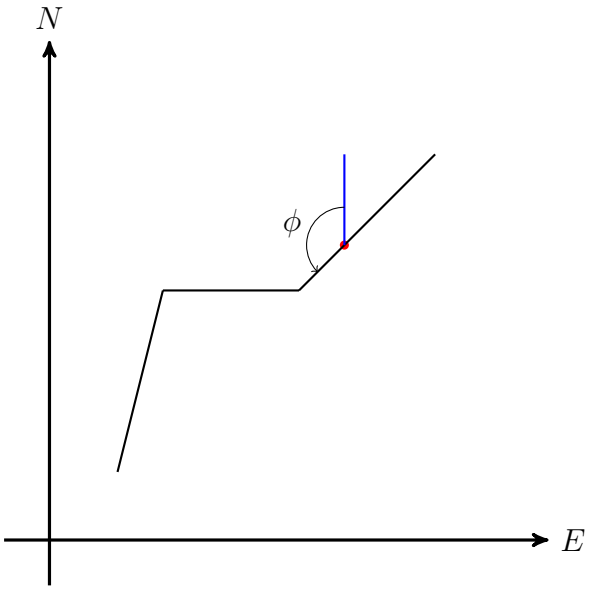


Figure 3.9: The angle of rotation

have taken the direction to the south, for which the current frame must be rotated by π (180°). If there is a change in both directions, we need the angle between the line connecting the two coordinates and the vertical axis of the first coordinate, which is provided by the formula $\pi - \arctan(\frac{\Delta x}{\Delta y})$.

So far only the transformations for the x, y axis were defined, however, the elevation can also change during the measurements. As a result, it is necessary to calculate the translation on the z axis and also the angle of elevation. Each elevation is transmitted to us in meters by GNSS sensors, so calculating the difference is a simple subtraction:

$$\Delta z = \tau_1 - \tau_2 \tag{3.6}$$

where τ means the elevation.

However, it is not enough to simply calculate the elevation differences, as the LiDAR sensor also tilts horizontally as a result of driving on a slope. The angle of

the elevation is calculated as shown in Eq. 3.7.

$$\beta = \begin{cases} \frac{\pi}{2} & \text{if } \Delta z > 0 \wedge d = 0 \\ -\frac{\pi}{2} & \text{if } \Delta z < 0 \wedge d = 0 \\ \arctan(\frac{\Delta z}{d}) & \text{if } \Delta z <> 0 \wedge d <> 0 \\ 0 & \text{if } \Delta z = 0 \wedge d <> 0 \\ \perp & otherwise \end{cases} \tag{3.7}$$

3.4.2 Fusion of results

For efficient and automated coordination of the two GNSS source, I needed a unified scale to decide which method has better accuracy to the given frame. For this, I used a simple percentage result, where 100% means that the method is likely to give a near-perfect result, while the lowest percentage means greater uncertainty.

Determining accuracy for GNSS

For Stonex and mobile GNSS, we have different information on how close the measured point can be to reality.

In the case of the Stonex sensor, each coordinate pair has a so-called dilution of precision (DOP) value [24]. This is a term used in satellite navigation to define error propagation as the mathematical effect of satellite navigation geometry on position measurement accuracy. Basically, the more satellites the GNSS receiver can see, the more accurate the result can be. However, the distance between the satellites matters. If they are too close to each other, the DOP value will be poor. The ideal arrangement (of the minimum four satellites) is one satellite directly overhead, three others equally spaced near the horizon. The signal provided in case of poor and good positioning of the satellites is shown in Fig. 3.10.

The DOP value scale is starting from 1. The lower the value, the greater the accuracy of the measurement. The approximate meaning of the different DOP values is given in Table 3.1.

Various type of DOP values can be used, depending on the location of the satellites:

- HDOP – horizontal dilution of precision

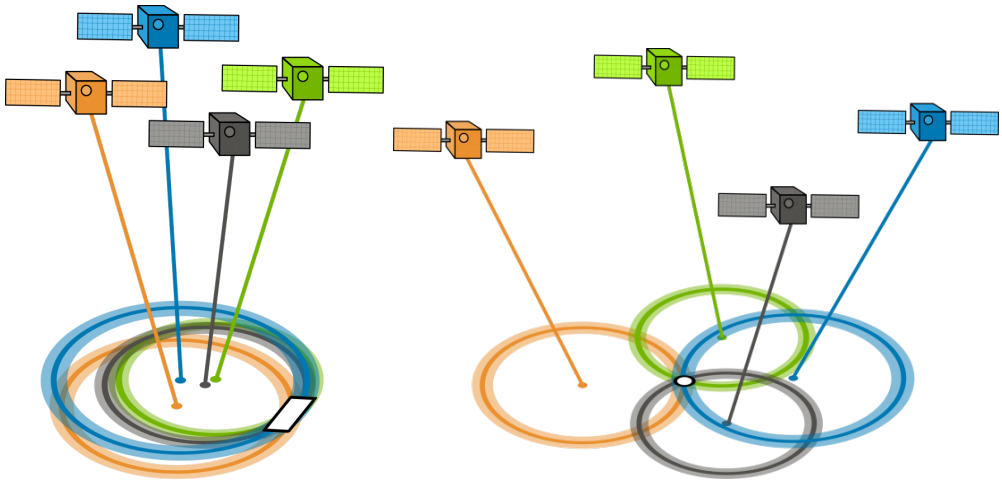


Figure 3.10: The alignment of the satellites, causing poor (a) and good (b) DOP
Source: `gisgeography.com`

- VDOP – vertical dilution of precision
- PDOP – position (3D) dilution of precision
- TDOP – time dilution of precision
- GDOP – geometric dilution of precision

However, the DOP value is only a multiplier. If we want to determine the accuracy in meters, we need to know how accurate our GNSS sensor is. If it's accuracy is e.g. 0.5 meters and the DOP value is 2.5, then the accuracy of the measurement is about 1.25 meters.

In the case of mobile GNSS the sensor does not return a DOP value, it only has an accuracy specified in meters. Therefore, I also had to determine the accuracy provided by Stonex in meters. Since the sensor can also achieve centimeter accuracy, I used 0.5 meters as an upper estimate. Multiplying the PDOP value for each coordinate by this, I was able to use the same scale for both sensors. The value of 0.5 meter means 100% accuracy and a value of 11 meter means 0%, since the measurement can no longer be used. Values between the two are based on a linear distribution.

In the event that coordinates are missed, and they are only calculated by the Kalman filter, the accuracy is determined by subtracting the amount of time since the last actual measurement.

DOP value	Rating	Description
1	Ideal	Highest possible confidence level to be used for applications demanding the highest possible precision at all times.
1-2	Excellent	At this confidence level, positional measurements are considered accurate enough to meet all but the most sensitive applications.
2-5	Good	Represents a level that marks the minimum appropriate for making accurate decisions. Positional measurements could be used to make reliable in-route navigation suggestions to the user.
5-10	Moderate	Positional measurements could be used for calculations, but the fix quality could still be improved. A more open view of the sky is recommended.
10-20	Fair	Represents a low confidence level. Positional measurements should be discarded or used only to indicate a very rough estimate of the current location.
>20	Poor	At this level, measurements are inaccurate by as much as 300 meters with a 6-meter accurate device (50 DOP x 6 meters) and should be discarded.

Table 3.1: Meaning of DOP values. Source: Wikipedia)

3.4.3 Proper transformation of the point cloud

Now that we know the translation and rotations calculated by each sensor and their accuracy, the next step is to apply the transformations correctly in the point cloud.

Since we always only know the delta from the previous point, we have to record the total translation and rotation, to which the newly calculated results are added.

The current total translation and rotation will be applied to the current point cloud, as they are obtained raw from the LiDAR sensor without the use of the previous transformations. This is accomplished with an *affine transformation matrix*

[25]. In linear algebra, linear transformations can be represented by matrices. If T is a linear transformation mapping $\mathbb{R}^n$ to $\mathbb{R}^m$ and $\vec{x}$ is a column vector with n entries, then $T(\vec{x}) = \mathbf{A}\vec{x}$ for some $m \times n$ matrix, $\mathbf{A}$ called the transformation matrix of T.

To represent *affine transformations* with matrices, we can use *homogeneous coordinates*. This means representing a 3-vector (x, y, z) as a 4-vector (x, y, z, 1), and similarly for higher dimensions. All conventional linear transformations (such as translation and rotation) can be described as an affine transformation and thus represented as general transformation matrix. Each transform can be combined by multiplying the transformation matrices that describe them.

In our case, the transformation matrix applied to the point cloud can be described in the following form:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The first matrix is describing translation: t_x is the total translation along the x axis, t_y is the total translation along the y axis and t_z is the total translation along the z axis. The second matrix is the rotation matrix around x axis, described by *Euler angles* based on β, the elevation angle calculated in the previous section. Similarly, the third matrix is the rotation matrix that describes the rotation about the z axis based on ϕ.

3.4.4 Generation of output

The final part of the algorithm after the proper application of the computed transformations to concatenate the modified frames into a large common point cloud.

The problem with concatenating frames is that if each point cloud is concatenated one at a time, the resulting point cloud will contain too many points, making the actual image indelible and the run time of the algorithm too long. I used two methods to solve the problem:

On the one hand, since the LiDAR sensor produces several frames per second – which in the case of GNSS, will fall on the same coordinate, thus using the same

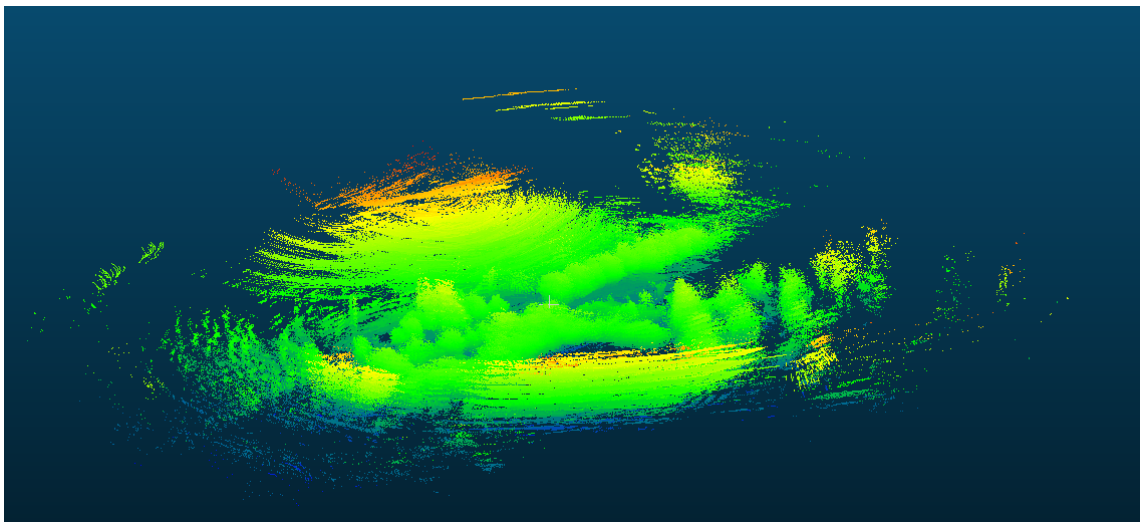
transformation – and they do not contain very different information, only one frame will be used for the given second.

On the other hand, it was necessary to decide which of the newly arriving points are those that were not yet included in the concatenated point cloud at all or were not yet in sufficient numbers. Only these points need to be used, the rest are disposable. To decide this, it was necessary to build a structure in which the individual points and their neighbors could be stored in order to be able to decide which points have a high density and which have low. Important aspect was that the structure could be indexed and searched easily and quickly.

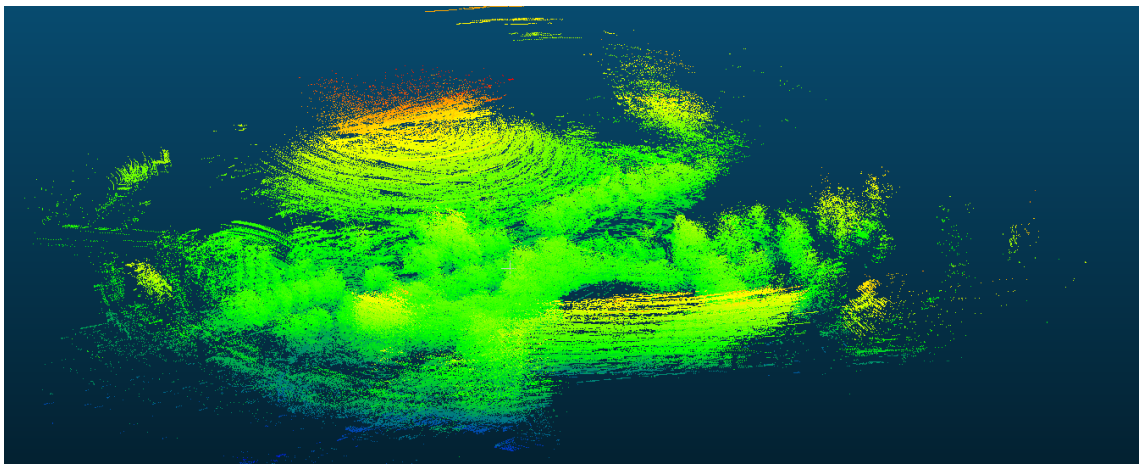
My first approach was to use the *k-d tree* structure [26]. It is a binary search tree where data in each node is a k-dimensional point in space. It allows to easily perform searches such as *nearest neighbor search*[27] or *radius neighbor search* [28], which can be used to solve the above-mentioned problem. The problem with the structure was that updating the tree due to the balancing problem is extremely costly, thereby drastically increasing the runtime of the algorithm.

For this reason, I decided to use the *octree* [26] data structure as second approach, which is also a tree structure, where each internal node has exactly eight children. In this – although less efficient way – the above-mentioned searches can be performed as well, and updating the tree is less expensive, giving significantly better runtime overall.

The tree will be built as soon as the first frame arrives and the necessary points will be inserted into the tree when the next frames arrive. Currently, a point is inserted only if there are less than 500 neighbors in the constructed tree within a specified radius. This is determined using the *radius neighbor search* algorithm. The difference between point clouds containing all points and only significant points is clearly shown in Fig. 3.11.



(a) The point cloud without octree



(b) The point cloud with octree

Figure 3.11: The difference resulting from the use of octree

Chapter 4

Implementation

4.1 Architecture

The essence of the method is that the information provided by the various sensors can be used properly to achieve the best possible result, in a fully automated way. To achieve this, it was necessary to create a well customizable architecture.

The framework of the program follows a standard producer-consumer architecture, in the form of a piping structure. The process begins with a producer, that reads each frame of LiDAR and transmits it to a consumer, trough a processing section. The processing part consists of independent processors. Each of them receives a point cloud and returns a point cloud, modified in some way. Each part of the pipe receives the point cloud modified by the processor before it, making its own modifications and then passes it on to the one following one. The relative order of the processors is freely changeable in the light of the desired result. We can also easily add new processors or take away existing ones, or make their participation conditional. The units can perform filtering tasks (to filter out certain points), transformation tasks (to determine the relative position of each frame) and also can be responsible for concatenation. At the end of the pipe are located (even several) consumer, that display the point cloud modified by the processing part, either in the form of visual or a file. Fig. 4.1 shows the model of the imagined architecture.

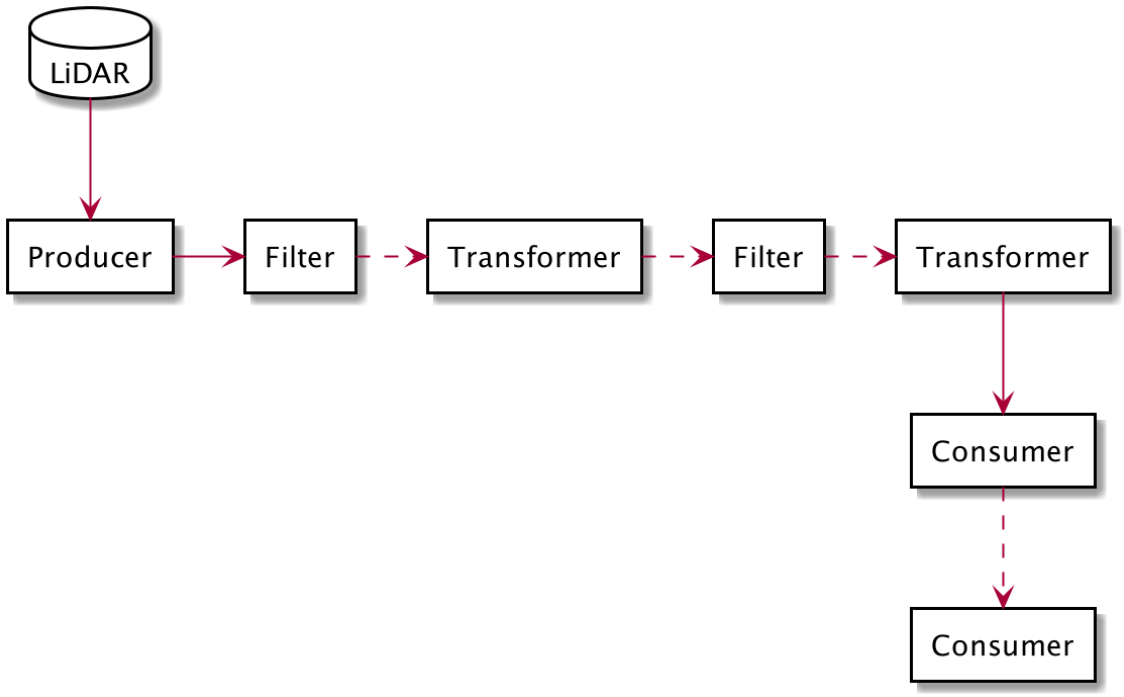


Figure 4.1: The workflow of the architecture

4.2 Implementation in C++

The architecture was implemented in C++ and the PCL library was used to process the point clouds.[3].

PCL supports several types of point cloud, depending on the properties of the point we want to use beside the X, Y, Z coordinates (such as intensity). Although my testing was performed using the `PointXYZI` type, I tried to have as much flexibility and reusability as possible during the design, so the classes performing point cloud operations were designed with the help of templates, where the template parameter is the type of the point cloud.

The program starts with a preprocessing phase where the GNSS coordinates are read from CSV file and the Kalman filter is applied to them. The starting rotation angle - which will form the basis for later calculations - will be also calculated.

For the producers, I have defined a `Producer` base class, which contains a `start` and `stop` function, with the help of which the reading of point clouds can be started or stopped. The class also includes the `onNewCloud()` function that notifies the arrival of a new point cloud and the `registerHandler()` function that assigns an

[3] `https://pointclouds.org/documentation/`

external function to handle the arrived frame.

From the `Producer` class inherits the `GrabberProducer` class, which is currently responsible for reading frames from .pcap files, or directly from the LiDAR sensor. This is done with the help of an instance of the PCL built-in `Grabber` class.

The processing part is based on the `Processor` class. The class has a virtual `process()` function, which is expect a constant reference to a point cloud as input parameter, and returns an reference to the modified point cloud. Both the filters and the various transformers are derived from the `Processor` class, each implementing the `process()` function according to its own purpose.

The algorithm currently includes an `OrigoFilter`, a `CloudTransformer` and a `MergeTransformer` class.

The job of the `OrigoFilter` is to filter out the very close points, thus the people pushing the structure, so that their silhouette does not disturb the final result. This is done using the `CropBox` class in the pcl library, which removes points within a specified area.

The filtered point cloud is received by the `MergeTransformer` that currently includes 2 calculators: one instance from `GPSCalculator` for mobil GNSS and one for Stonex sensor. Each calculator implements the `Calculator` base class and its virtual `calculate()` function. They use the same methods, but difference source to determine the direction and amount of translation on each axis since the last Frame, together with the rotation and elevation angle. Each calculator also calculates a certainty, how good is the given result. As mentioned above, this certainty is expressed as percentages. The data calculated in this way is returned using the `TransformData` structure.

As mentioned earlier, the `GPSCalculator` class use GNSS coordinates to calculate the required values - one based on signals from the Stonex sensor and the other from mobile GNSS. In the case of the Stonex senor, it can also be decided to use the WSG 84 or EOV coordinates for the calculation. In the case of one of the GNSS sensors does not have information, the associated calculator does not will be applied.

The `CloudTransformer` examines the certainty of the calculators and applies the transformation calculated by the best to the point cloud.

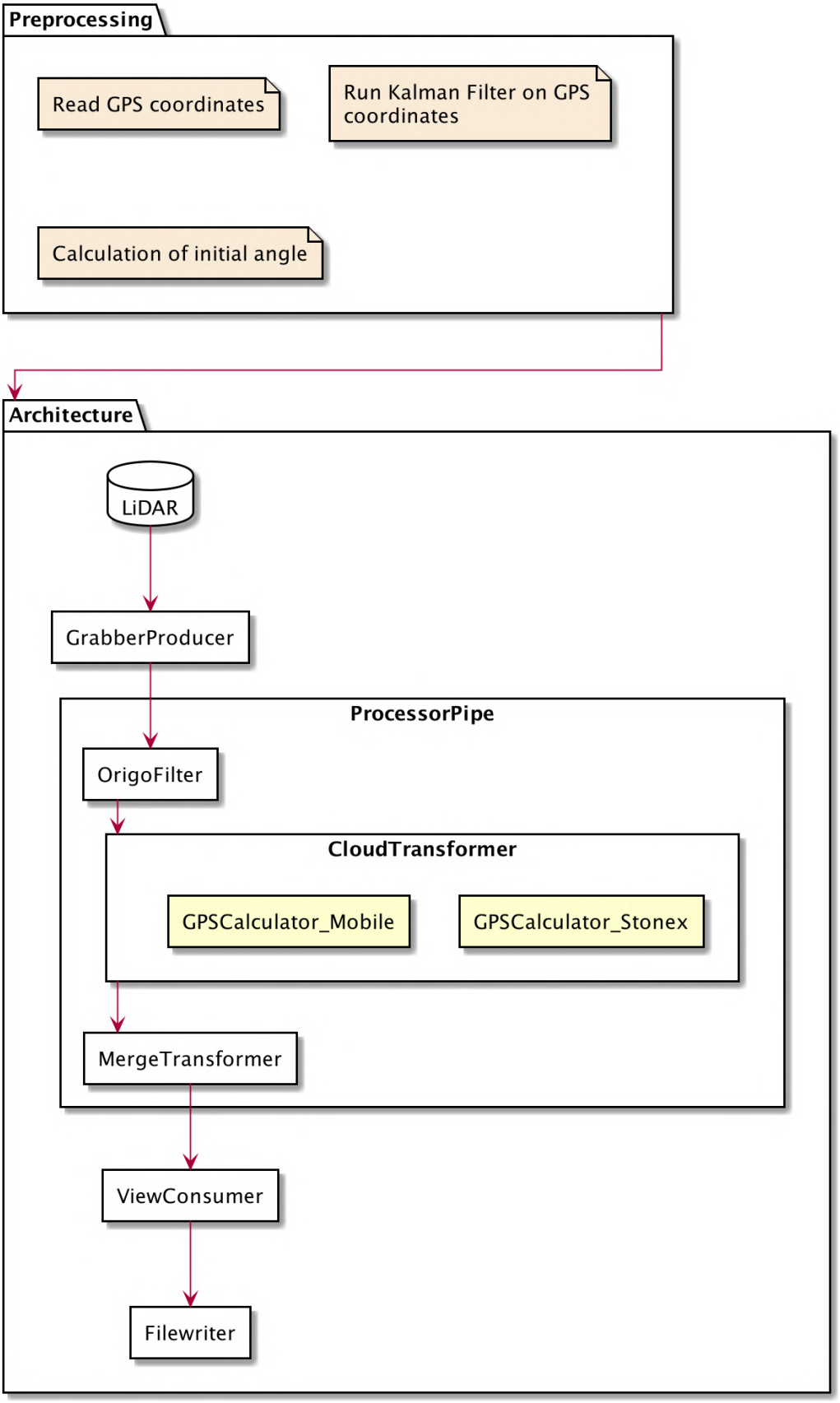


Figure 4.2: The architecture of the system

The point cloud, which now has the appropriate offsets and rotations, is merged with the existing point cloud by `MergeTransformer`, with the help of an *octree* data structure.

The filter and transformer classes are grouped together by the `ProcessorPipe` class, which implements a pipe structure. The class contains `std::vector<olp::Processor<PointType>*> processors` data member. A new member can be added to the vector using the `add()` function of the class. By calling the `execute()` function, the program iterate through the list of processors. The loop calls the `process()` function to the next processor, with the output of the `process()` function of the previous processor. The first processor receives the raw point cloud received from the `GrabberProducer` class, which the `execute()` function expects as an input parameter.

The classes that implement the `Consumer` class and its `virtual show` function are responsible for displaying the concatenated point cloud. The `ViewConsumer` class draw the point cloud on the terminal using the PCL library `Viewer` class. After closing the viewer, the `FileWriter` class - whose base class is also the `Consumer` class - can be written to the created point cloud file in LAS format. The described operation of the system is illustrated on Fig. 4.2.

4.3 Compilation and execution

Proper installation of the program requires the installation of the PCL library and its dependencies. It is important to note that in order for the PCL library to be able to handle PCAP files, its source code must be recompiled with the `-DWITH_PCAP = YES` flag. External dependencies belonging to the program are located in the vendor folder and are added to the project as *git submodules* [4] . The program can be compiled using *cmake* [5] after installing the appropriate dependencies. See the repository README file for more information and platform-specific installation instructions.

Information required for the program to run properly, such as the path to a .pcap file or CSV files with GNSS coordinates, can be specified as arguments. It is

[4] `https://git-scm.com/book/en/v2/Git-Tools-Submodules`

[5] `https://https://cmake.org`

also necessary to specify the start time of the measurement in UNIX epoch time as argument, and the filtering of nearby points can be optionally enabled in this area too. By using the `--help` command, the program lists the possible arguments and their expected format.

4.4 Code Availability

The source code of the program is an attachment of the thesis. It can also by viewed online at the `https://github.com/mcserep/olp` GitHub repository.

Chapter 5

Results

The measurements were performed with the help of the university tools and my supervisor, in areas close to the university. Fortunately, we were able to find a couple of locations, (such as the Tüske Hall or the BEAC sidewalk) where the measurements made could be used for testing. The different aspects to achieve a good measurement are explained in Sec. 3.3. The samples below examine the efficiency of the algorithm with or without the presence of these aspects.

5.1 Measurement experiences and results

The first measurements

In our very first measurements, neither the setting of the GPS instruments nor the ground was smooth enough to achieve proper results. In the case of the Stonex GNSS sensor, the sensor clock was set for several days and hours, and the setting to communicate with ground stations was missing, so the results were not accurate enough. In the case of the LiDAR sensor, not only was the ground uneven enough, also the sensor was located in the hand, which caused additional resonance. Because of these, running the algorithm did not yield evaluable results.

Learning from these problems, the GNSS sensor has been properly configured and a stable placement of the sensors was designed, which was presented in Sec. 3.2.

Measurement near tall buildings

The next measurement - with sensors that are now properly calibrated and positioned - took place on a much more even ground, but in a location surrounded by tall buildings, as the Ericsson House.

Since the measurement started near a tall building, which we were getting closer and closer to during the measurement, there was no completely good signal for any of the GNSS sensors. This was helped a lot by the use of the Kalman filter. Fig. 5.1 shows a detail of the route taken. Green shows the original points located differently from the straight line, and red shows the improved path created by the Kalman filter, which is almost straight.

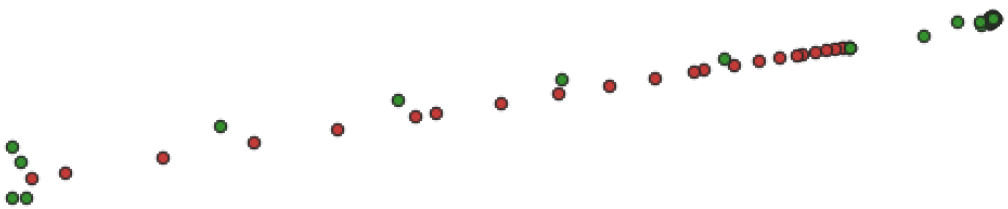
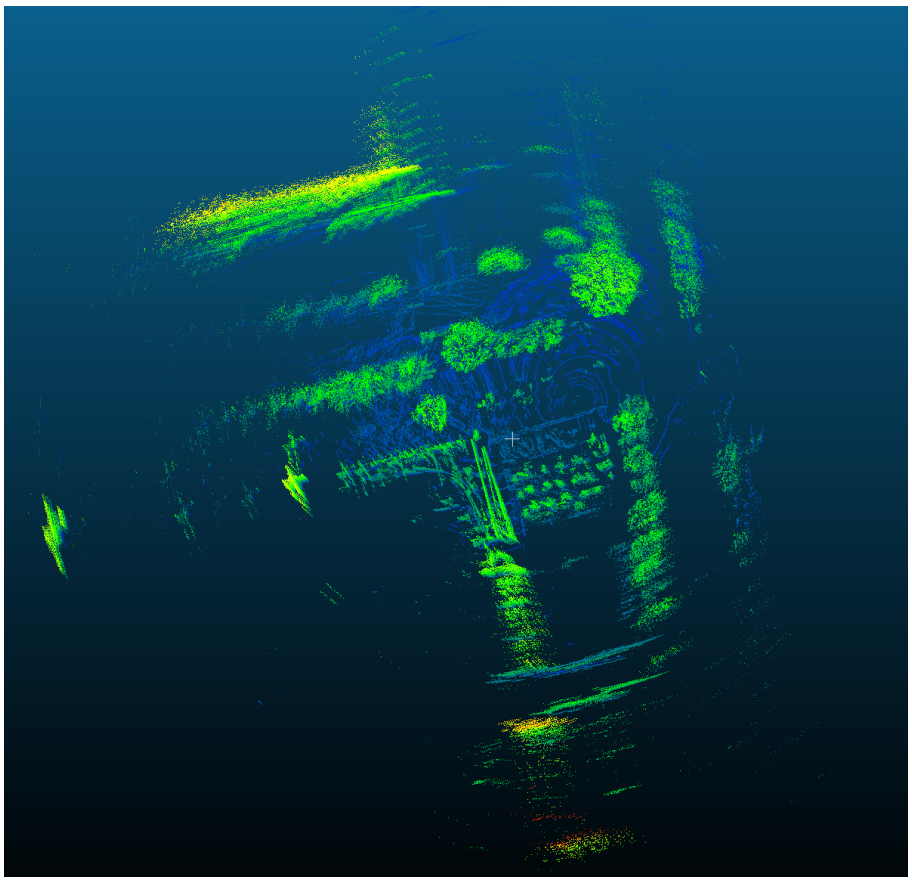


Figure 5.1: The original route (green) corrected by Kalman filter (red)

Thanks to this I managed to get a relatively smooth route but the shifts in the created point cloud can still be seen, as it shown on Fig. 5.2.

As both sensors completely lost their signal when approaching the building, the route itself became relatively short, about 25 meters.

(a) The studied area represented on the map



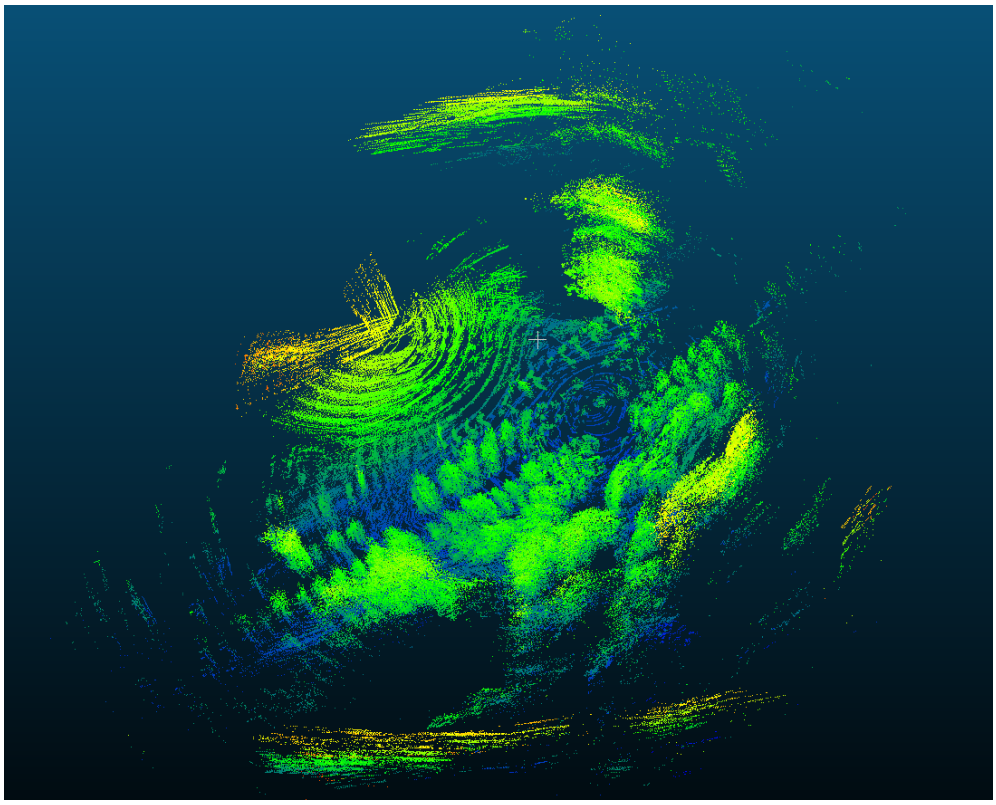
(b) The created point cloud

Figure 5.2: Measurement surrounded by tall buildings

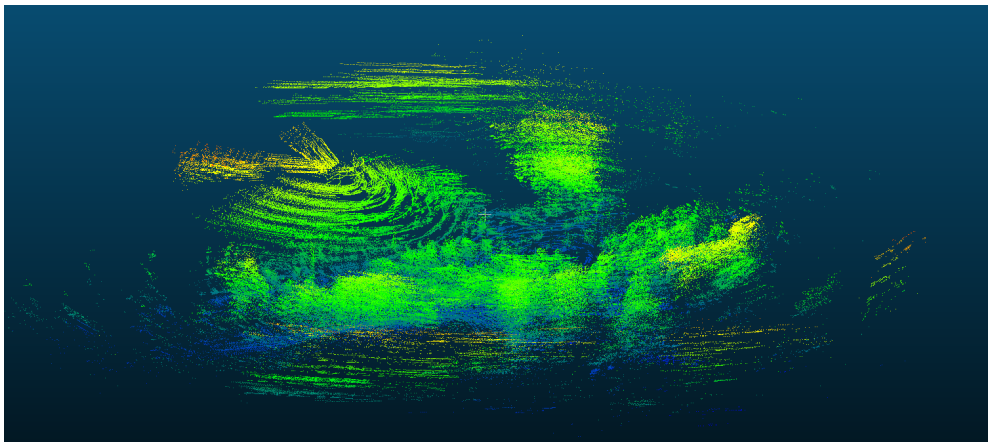
Measurement with straight path and open area

For the next measurement, we were already looking for a location with a much more open area. Our choice fell on the area around Tüske Hall. Here, the GNSS signal was only disturbed by trees. As a result, the path was smooth even without the use of the Kalman filter in the case of GNSS coordinates, but by applying it, small oscillations were also corrected. However, in the resonance due to the uneven ground, there are still inaccuracies in the point cloud given by the LiDAR sensor. The Fig. 5.3 shows the map of the studied area and Fig. 5.4 the point cloud created based on it. The traveled route – which was 34 meter in X axis and 33 meter in Y axis – is marked in red. The created point cloud in 3D can be seen better from side view, as shown on Fig. 5.4 (b).

Figure 5.3: The studied area represented on the map



(a) The created point from side view cloud



(b) The created point from top view

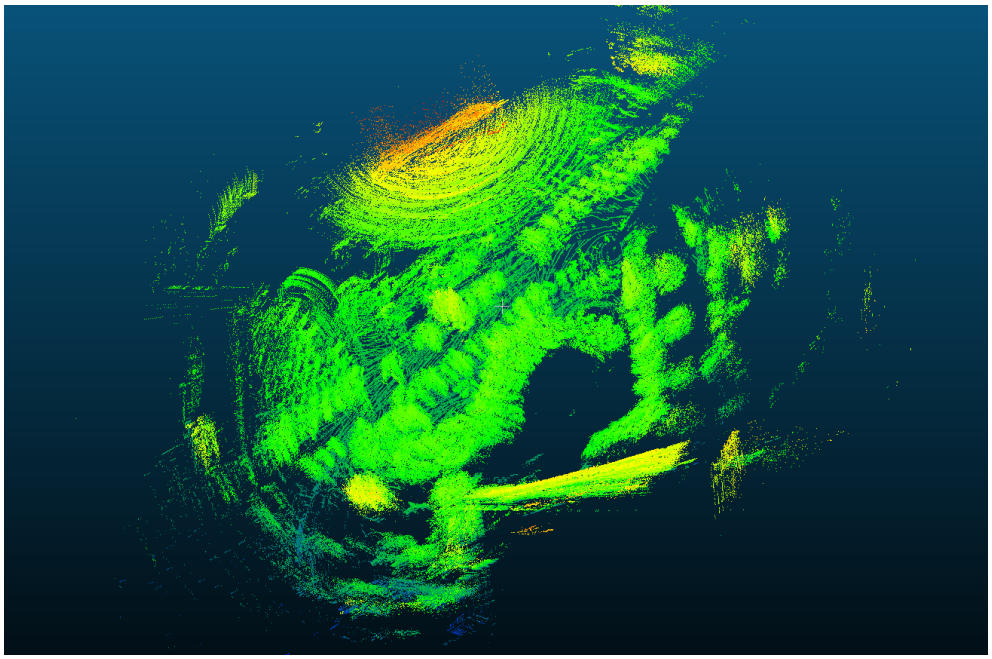
Figure 5.4: Measurement with a straight path and open area

Measurement with straight path and a slightly shaded area

At the same location, but on the other side of the road, where there were several trees with large crowned covering the route, we also took a measurement. As a result, the Stonex sensor completely lost the signal on a stretch and the mobile GNSS deviated from the original route. Fortunately, the Kalman filter was used to

improve so much that the measurement still gave good results. Fig. 5.5 shows the route given by Stonex sensor and corrected by the Kalman filter and the resulting point cloud. In this case, our route was also longer, a total of 70 meters.

(a) The studied area represented on the map



(b) The created point cloud

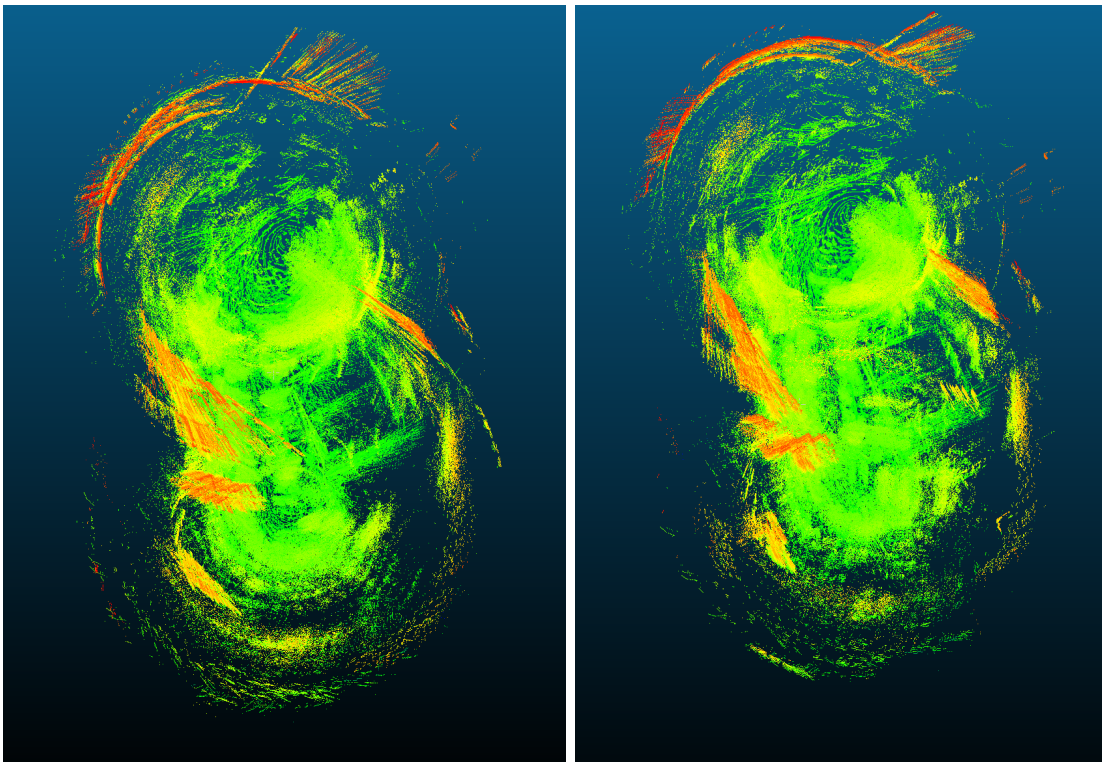
Figure 5.5: Measurement with straight path and a slightly shaded area

Measurement with shaded area and winding road

In the next measurement, we tested the algorithm in near the BEAC sport field, where we had to turn several times during the measurement.

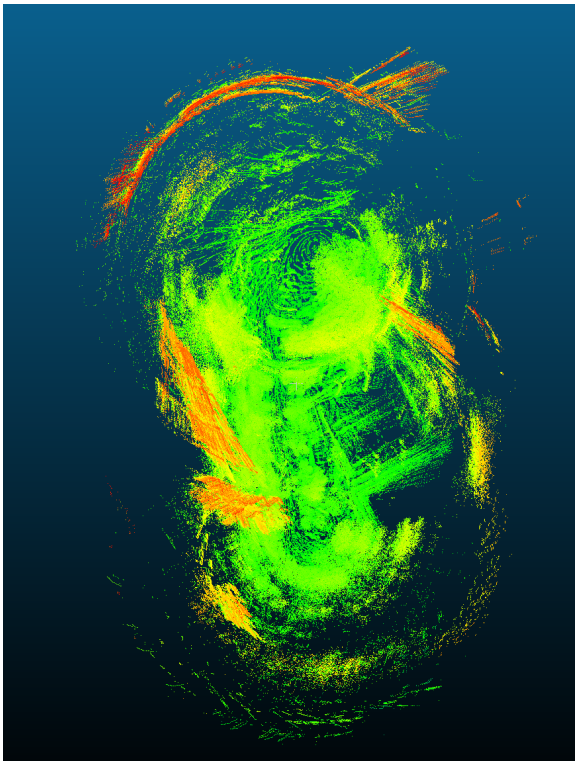
Figure 5.6: Measurement with shaded area and winding road

In this measurement, it has also been well tested that the Stonex sensor signal, which is sensitive to interference factors, can be effectively supplemented with a mobile GNSS signal, which is less accurate but less sensitive and can give good results, especially if a Kalman filter is applied to it. The examined area is shown on Fig. 5.6. The yellow route is the original path provided by the Stonex sensor, the red route is the path corrected by the Kalman filter, while the pink one is the corrected path of the mobile GNSS.



(a) Only mobile GNSS

(b) Only Stonex GNSS



(c) Stonex + mobile GNSS

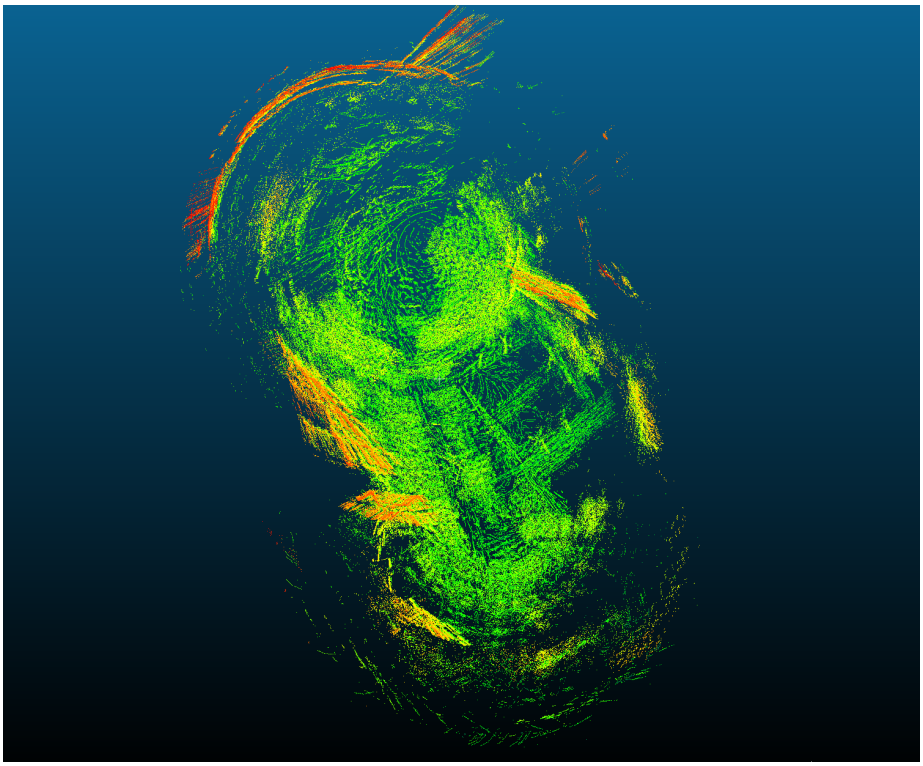
Figure 5.7: Point cloud resulting by the use of different GNSS sensors

As shown in Fig. 5.7, when using only the Stonex sensor (a) the beginning of the measurement, while using only the mobile, the end of the measurement is more accurate. This can be explained by the fact that, in the case of the Stonex sensor, due to an unexpected factor such a wider tree or a larger billboard the measurement was missed by up to meters, mainly half of the end of the measurement. The Kalman filter was able to fill it, but with less and less accuracy. In the case of the mobile, although the route is not so straight, applying the Kalman filter to it gives a result that can be used well. The original path given by the different sensors and corrected path by the Kalman filter can be seen in Fig. 3.8 which was already shown in previous section. Combining the two gave a better result for the whole measurement than if the sensors had been used separately.

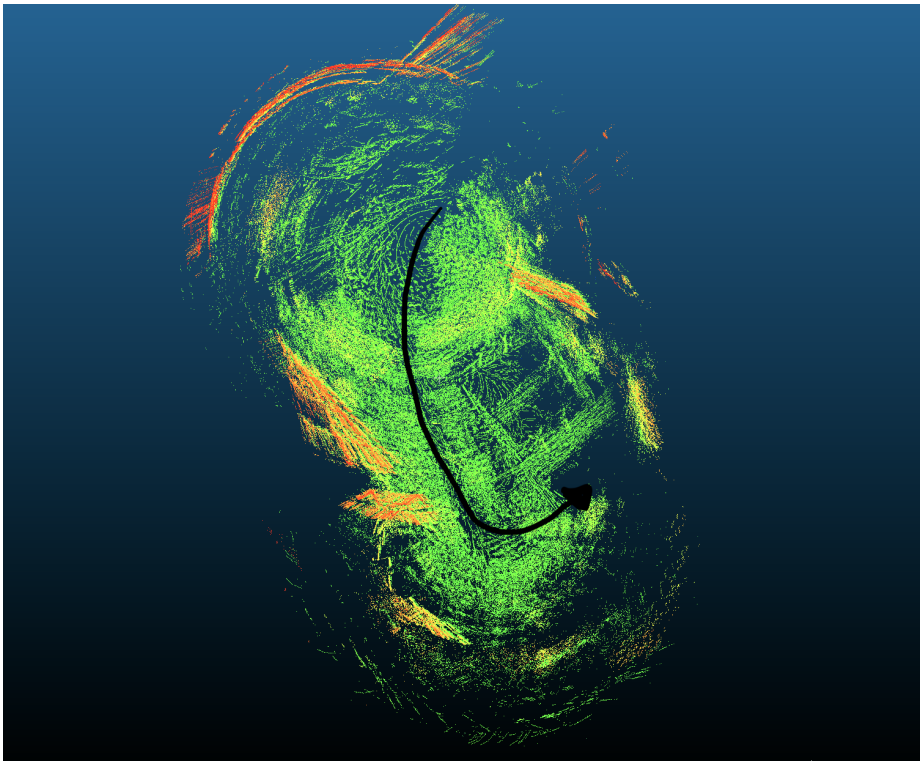
In this case, lowering the density of the points results in a more visible point cloud, as shown in Fig. 5.8.

The final result shows that we moved left at the beginning of the measurement, then turned right, followed by a long straight section followed by a left bend. Although the rotations caused a minor error in the point cloud, the overall direction of progress can be clearly seen on Fig. 5.8. The measurement from side view can be seen on Fig. 5.9. Not only in complexity, but also in length, it was my biggest measurement, roughly 150 meter.

As with the other measurements, the resonance due to the uneven ground and the uncertainty of the car with sensors caused a problem in the point cloud. Unfortunately, this problem could not be eliminated during the measurements.



(a) The created point cloud



(b) The traveled road in the point cloud

Figure 5.8: Point cloud with lower density

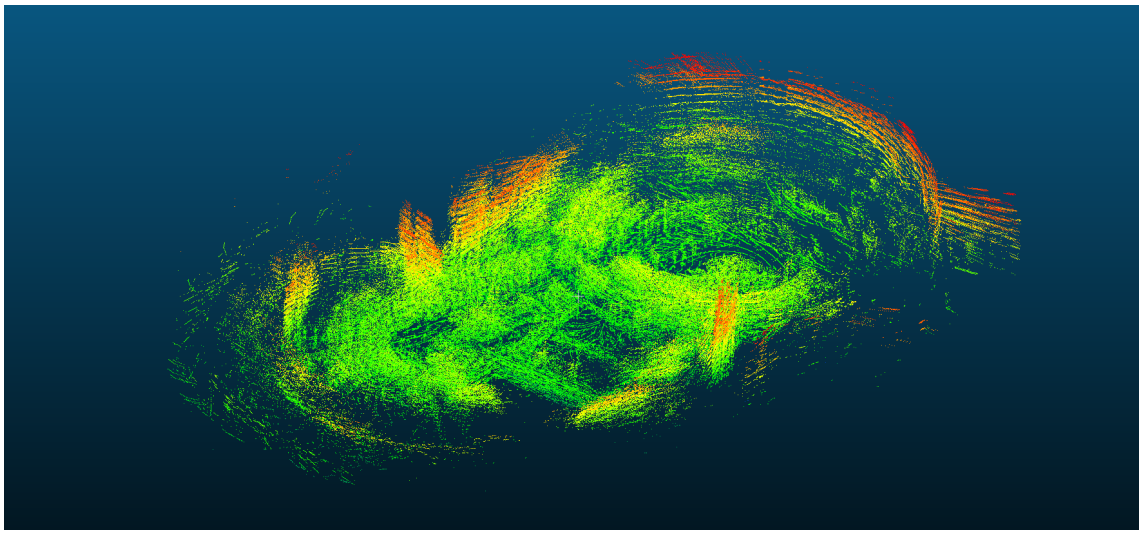


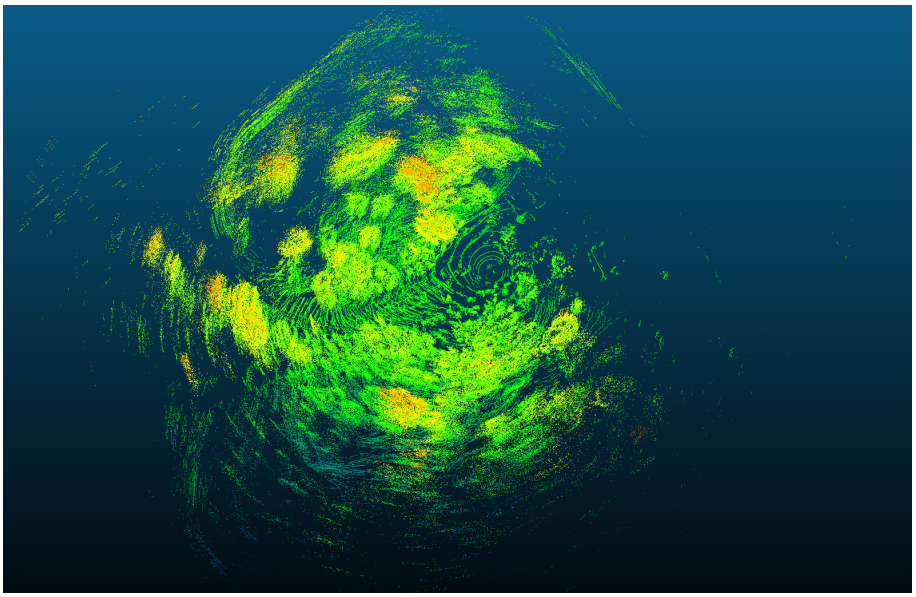
Figure 5.9: Studied area from side view

Measurement with elevation difference

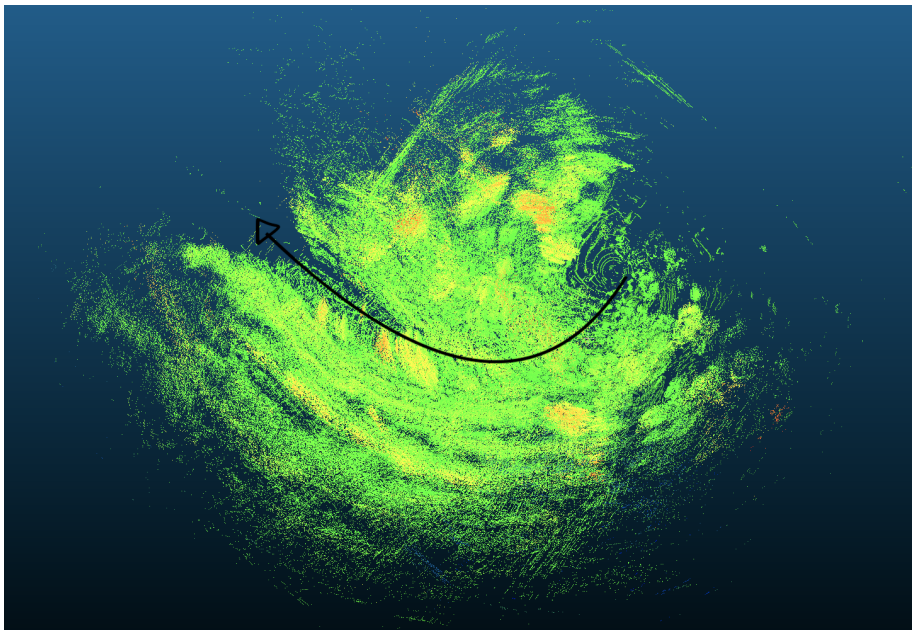
The measurements so far have been made mainly in locations where the change in elevation difference was negligible. For the next measurement, we selected a location where, in addition to the continuous bending of the road, the height also changes. The winding road near the Petőfi Bridge was perfect for this. The studied area with the traveled route is shown on Fig. 5.10.

At the beginning of the measurement, the route was nicely visible, by the end of the it the point cloud was a bit blurred, but the direction of progress can still be perceived, as it shown on Fig. 5.11.

Figure 5.10: The studied area



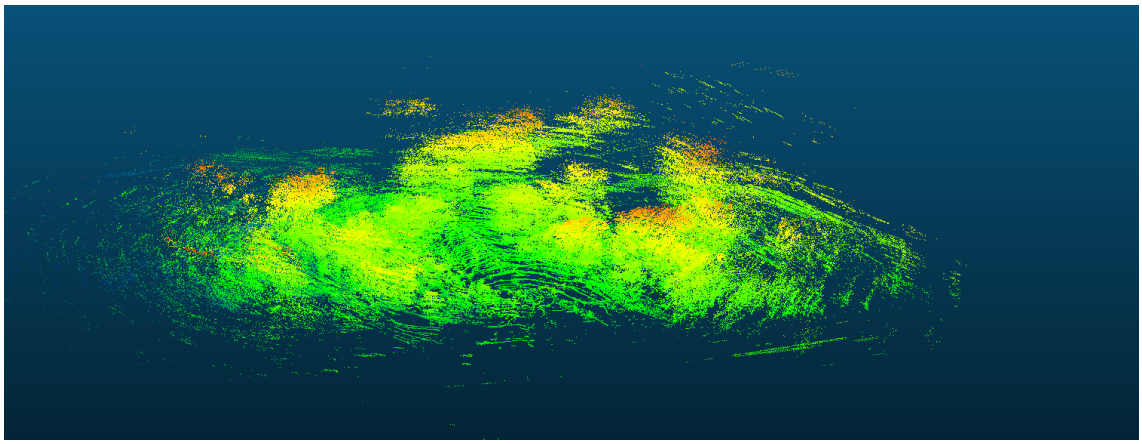
(a) The beginning of the measurement



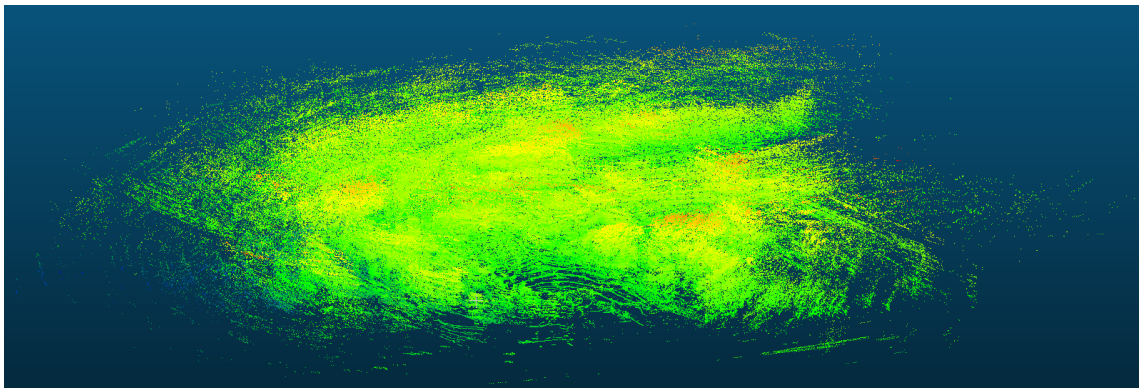
(b) The end of the measurement with the traveled path

Figure 5.11: Different stages of measurement

Although it is difficult to recognize objects, the difference in elevation - which was 3 meter - between the beginning and the end of the measurement is clearly seen on Fig. 5.12.



(a) The beginning of the measurement



(b) The end of the measurement

Figure 5.12: Elevation difference during the measurement

5.2 Processing parameters

The algorithm was tested with 16 GB RAM and a 256 GB SSD under macOs Catalina operating system, on a Macbook from the year 2016.

The point cloud was read from pcap files, while the GNSS coordinates were read from csv files. The created point cloud was saved in LAS format.

Although obviously the number of frames affects the runtime, the number of points in each frame and to be inserted is also important, as the runtime is mainly increased by operations with *octree*.

The running time for each measurement is shown in Table 5.1.

Measurement	Number of frames	Run time
Measurement near tall building	1070	5 min
Measurement with a straight path and open area	1410	6 min
Measurement with straight path and a slightly shaded area	2800	8 min
Measurement with shaded area and winding road	3027	13 min
Measurement with elevation difference	2737	11 min

Table 5.1: Processing time of measurements

Chapter 6

Conclusion

The aim of my thesis was to develop a framework that can efficiently and fully automatically combine the signals of several sensors and the help of appropriate algorithms can properly position the point cloud created by LiDAR.

A producer-consumer architecture has been developed, that can efficiently and dynamically combine individual producers, consumers and processing units, whether it is a filter or a transformation class.

To test the algorithm, a Stonex GNSS and a mobile GNSS sensor were combined with a Velodyne LiDAR sensor. The advantages and disadvantages of GNSS sensors were analyzed, and using them, an accuracy system was identified that the algorithm can use to decide between the sensors. An extended Kalman filter was used to improve the signal of the sensors.

Different measurements were made, which tested the created process along different parameters. During the measurements, I always encountered new difficulties, such as improper calibration of the sensors or factors interfering with the GNSS signal.

The thesis managed overcome most of the difficulties, and achieve good results. However, the resonance resulting from the uncertainty of the push cart dolly and the uneven ground was a problem in all measurements, causing errors in each point cloud.

6.1 Future work

To improve the accuracy of the algorithm, it would definitely be necessary to develop a more stable base for the sensors. It is also planned to use a smaller, easier-to-place device, such as a Rasberry Pi, instead of the laptop during the measurements.

Involving additional sensors, such as an IMU sensor, could also further improve the accuracy of the fusion and make it usable also indoors.

Another popular solution to the problem introduced in this thesis is the use of SLAM algorithms, which give good results mainly indoors. Due to this, the two methods are well combined. The combination of the sensor fusion algorithm with a solution based on SLAM algorithms is currently in progress. Thanks to the easy-to-expand architecture and the percentage-based accuracy estimation, integration is easy to implement. The only problematic point at present is the coordination of the created transformation matrices.

Acknowledgement EFOP-3.6.3-VEKOP-16-2017-00001: The research behind this master thesis is supported by the Hungarian Government and co-financed by the European Social Fund.

Bibliography

[1] *PCAP file format.* https://wiki.wireshark.org/Development/LibpcapFileFormat. Online; accessed 20 April 2020.

[2] ASPRS Board of Directors. *LAS Specification.* Tech. rep. 1.4 – R13. American Society for Photogrammetry and Remote Sensing, July 2013. URL: https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf.

[3] Martin Isenburg. "LASzip: lossless compression of LiDAR data". In: *Photogrammetric engineering and remote sensing* 79.2 (2013), pp. 209–217.

[4] *GNSS systems.* https://gssc.esa.int/navipedia/index.php/GNSS_signal. Online; accessed 20 April 2020.

[5] *Global Positioning System Wide Area Augmentation System (WAAS) performance standard.* Tech. rep. Federal Aviation Administration, Oct. 2008.

[6] Simon Cooper and Hugh Durrant-Whyte. "A Kalman Filter Model for GPS Navigation of Land Vehicles". In: *1994 Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE. 1994, pp. 157–163.

[7] *DGNSS systems.* https://gssc.esa.int/navipedia/index.php/Differential_GNSS. Online; accessed 21 April 2020.

[8] *EOV coordinate system.* http://lazarus.elte.hu/gb/geodez/geod2.htm. Online; accessed 17 April 2020.

[9] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering.* 1960, pp. 35–45.

[10] Simon J Julier and Jeffrey K Uhlmann. "New extension of the Kalman filter to nonlinear systems". In: *Signal processing, sensor fusion, and target recognition VI*. Vol. 3068. International Society for Optics and Photonics. 1997, pp. 182–193.

[11] Mark G Petovello et al. "Consideration of time-correlated errors in a Kalman filter applicable to GNSS". In: vol. 83. 1. Springer, 2009, pp. 51–56.

[12] Angel-Iván García-Moreno and José-Joel Gonzalez-Barbosa. "GPS precision time stamping for the HDL-64E Lidar sensor and data fusion". In: *2012 IEEE Ninth Electronics, Robotics and Automotive Mechanics Conference*. IEEE. 2012, pp. 48–53.

[13] L. Lamport. "Time, clocks, and the ordering of events in a distrubuted system". In: mdpi. 1978.

[14] Richar L. Pio. "Euler angle transformations". In: mdpi. 1966, pp. 707–715–163.

[15] Tomáš Trafina. "Construction of 3D Point Clouds Using LiDAR Technology". Bachelor's Thesis. Czech Technical University in Prague, 2016.

[16] Mohamed M. Atia Yanbin Gao Shifei Liu and Aboelmagd Noureldin. "INS/GPS/LiDAR Integrated Navigation System for Urban and Indoor Environments Using Hybrid Scan Matching Algorithm". In: mdpi. 2015.

[17] Ken Shoemake. "Quaternions". In: Department of Computer and Information Science University of Pennsylvania.

[18] Chuang Qian et al. "An integrated GNSS/INS/LiDAR-SLAM positioning method for highly accurate forest stem mapping". In: vol. 9. 1. Multidisciplinary Digital Publishing Institute, 2017, p. 3.

[19] *VLP-16 UserManual*. VelodyneLiDAR,Inc. SanJose,CA95138, 2019.

[20] AM-Ver.1-Rev.4 CS. *User Manual STONEX S9III Plus GNSS Receiver*. Stonex. 2014.

[21] *Exynos 7885*. `https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-7-series-7885/`. Online; accessed 15 April 2020.

[22] Mattias Eliasson. "A Kalman filter approach to reduce position error for pedestrian applications in areas of bad GPS reception". Bachelor's Thesis. UMEA Universitet, 2014.

[23] C. C. Robusto. "The Cosine-Haversine Formula". In: *The American Mathematical Monthly Vol. 64, No. 1 (Jan., 1957)*. 1957, pp. 38–40.

[24] Richard B. Langley. "Dilution of Precision". In: GPS World. 1999.

[25] *Affine Transformations*. `https://people.cs.clemson.edu/~dhouse/courses/401/notes/affines-matrices.pdf`. Online; accessed 20 May 2020.

[26] Leonard Susskind and George Hrabovsky. *Mark de Berg, Otfried Cheong Marc van Kreveld, Mark Overmars*. Springer, 2008.

[27] Xiaoyao Xie Qing Xie. "Point cloud data reduction methods of octree-based coding and neighborhood search". In: *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*. IEEE. 2011, pp. 3800–38003.

[28] Volker Steinhage Jens Behley and Armin B. Cremers. "Efficient Radius Neighbor Search in Three-dimensional Point Clouds". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015.

List of Figures