TDK-dolgozat

Péter Farkas Dominik Jámbor

LiDAR point cloud positioning using sensor fusion

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPT. OF SOFTWARE TECHNOLOGY AND METHODOLOGY

Authors:

Péter Farkas
Computer Science MSc
2. year

Dominik Jámbor
Computer Science MSc
2. year

Supervisor:

Máté Cserép
assistant lecturer

Budapest, 2022

Contents

Chapter 1

Introduction

In recent years the importance of accurately mapping the surrounding area of an object escalated notably primarily due to the rise of autonomous vehicle development. Other areas however, such as mapping hazardous or remote sites, detecting and analyzing changes in facilities or structures are actively researched as well, since advancements in these areas lead to safer working and living environments for countless people.

Multiple methods have been researched to achieve such mapping and fusion is a common approach in the majority of them. However, the kind of data sources fused together and the type of fusion leaves room for future research as there are a vast number of possibilities.

In this paper, we propose a pipeline based approach for processing LiDAR data and complementing it with other data sources, such as IMU and GNSS sensors to determine the pose of the LiDAR sensor and generate a merged point cloud, which is an accurate reproduction of the covered area. Our goal with the pipeline is to create a flexible mapping framework, which can easily be extended with new sensors whether as a sensor with varying accuracy and an accuracy metric or a sensor from which the transformation can always be executed as a pipeline step. We also set out to execute our research with a portable and relatively low-cost set of equipment which can be easily replicated.

We evaluated our design with SLAM-LiDAR, GPS-LiDAR, GPS-SLAM-LiDAR and IMU-SLAM-LiDAR combinations, some of which were done by pushing a cart by hand, others by mounting the equipment on a car. All of the combinations produced promising results, laying the foundation for further improvement and research with our pipeline model.

Chapter 2

Related Work

2.1 The point cloud positioning problem

To achieve accurate environment mapping, different techniques are being utilized for facility management, hazardous site discovery as well as positioning and navigating autonomous vehicles. One of the most frequently used sensors for site mapping are LiDAR (*Light Detection and Ranging*) sensors. A LiDAR sensor typically emits eye-safe pulsed lightwaves into the surrounding environment and calculates the distance they traveled based on the time of return. These devices come with different field of view characteristics, for example, in Figure 2.1 a 360 degree field of view sensor is presented. After collecting a considerable amount of measurements, the output of a LiDAR sensor is usually a point cloud. This cloud is essentially a 3 dimensional model of the environment, however only the part which is visible for the sensor at the moment of measurement. In order to get a complete model and thus map the surrounding site, multiple of these point clouds need to be fused together while moving the sensor around. Estimating the pose of the LiDAR sensor is a key part in fusing these point clouds together.

Figure 2.1: Teledyne CL-360 LiDAR sensor [1]

2.2 Hardware options

Depending on the use-case, numerous other sensors can be used to help determine the pose of an agent. Autonomous vehicles in most cases can make use of a rotary encoder attached to one of the wheels. Through this device, the angular motion of the wheel can be measured digitally which provides a reliable odometry. Such vehicles often utilize onboard cameras as well and process their data feed with computer vision algorithms. The shortcomings of such combinations are that rotary encoders can not provide heading data and conventional cameras are sensitive to light conditions.

Another very common type of device are GNSS (*Global Navigation Satellite System*) sensors. These devices as Figure 2.2 shows, use satellites to determine their position on Earth by calculating the amount of time it took for the signal from the satellites to reach the sensor. This method requires the satellites to have a very precise clock, which is solved by equipping them with atomic clocks. It is also required that the receiver device has a very accurate time reading, but this can be calculated by using 4 satellite signals. The drawbacks of a GNSS sensor is that it needs a clear line of sight to the satellites and in case of a cold start it can take several minutes for the sensor to map the required satellites.

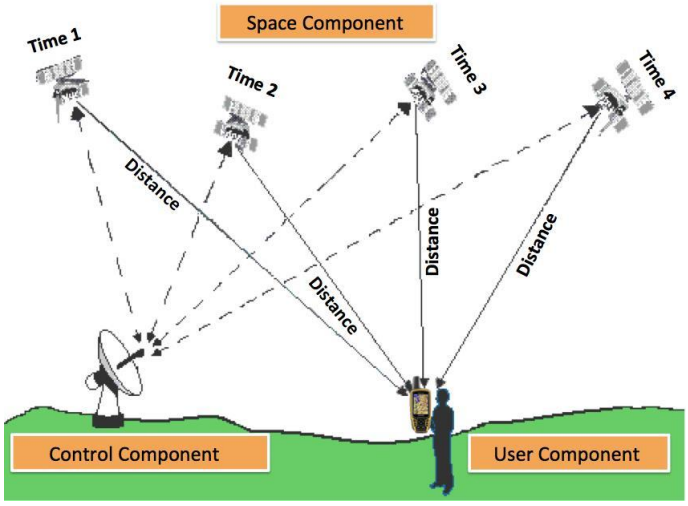

Figure 2.2: GNSS principles illustrated simply [2]

IMU (*Inertial Measurement Unit*) sensors are often used to estimate orientation and (short term) position as well. A typical 9 DOF (*degree of freedom*) IMU consists of 3 sensors. A 3 axis accelerometer to get a digital reading of the linear acceleration on each axis, a 3 axis gyroscope for angular rate reading and a 3 axis magnetometer to correct the heading calculation by serving as a compass in case of biases or errors in the other 2 sensors. 9 DOF IMUs today are often shipped with integrated GNSS sensors. 6 DOF IMUs without a magnetometer as seen in Figure 2.3 are common as well, because they are sufficient for a lot of applications where the heading is not needed or is calculated differently. Inertial measurement units, when used alone, can be affected by magnetic interference, and even the more advanced units have noise and measurement errors which add up over time.

Figure 2.3: SkyMEMS 6 Dof IMU Sensor [3]

Regarding the price categories of the previously mentioned devices, it can be quite costly to do such experiments. This paragraph provides a context for the price ranges and feature sets of these sensors.

- An outdoor LiDAR sensor can be purchased for under 1.4 million HUF (3 800 €) with 16 vertical channels, 100 m of range and 30° vertical field of view, however the price can be as high as 36 million HUF (95 000 €) for a sensor with 128 vertical channels, 300 m range and 40° vertical field of view.
- IMU sensors can vary from under 2 000 HUF (5 €) for a sensor array with 8.75 mdps angular rate sensitivity, 0.061 mg linear acceleration sensitivity and 0.08 mgauss sensitivity up to 450 000 HUF (1 100 €) with 1 mdps angular rate sensitivity, less than 0.04 mg in-run bias stability for the accelerometer and less than 7°/hr angular rate drift for the gyroscope with onboard soft and hard iron calibration for the magnetometer.
- GPS module prices start at 2 000 HUF as well for a sensor with 56 receiving channels and 2.5 m location location accuracy, whereas RTK GNSS sensors are available at 5 million HUF (13 000 €) with 448 channels a 6 cm location accuracy and a variety of complementary features.

The devices used during our measurements are on the lower end of the price and feature spectrum as we set a low overall equipment price as one of our goals.

Complementing the LiDAR reading with other sensory data can be very effective, however it is not the only way of fusing point clouds together. SLAM (*Simultaneous Localization and Mapping*) algorithms, introduced in the next section, calculate the position based on the readings themselves instead of relying on other measurements.

2.3 Software options

2.3.1 SLAM

Simultaneous Localization and Mapping is the problem of creating a map of the environment and positioning the mapping actor in it at the same time. The issue originally emerged in robotics, with the question being if it is possible for a mobile robot to build a consistent map of an unknown environment while simultaneously locating itself in it.

It was first described by Lu and Milios, which solved the task in a two-step approach [4]. The first step tries matching the subsequent scans to establish an initial estimation of the transformation between two poses. The second step aims to optimize the Mahalanobis distances [5] between all poses in the created pose node network. This two step approach gained quite a popularity and has become the base for other methods extending on it.

2.3.2 Iterative Closest Point

One of the most common and long-standing SLAM methods is the ICP [6] (Iterative Closest Point) algorithm. Given a reference and an input point cloud, it produces the transformation for the input to best match the reference. The first step of this process consists of first establishing point pairs from the input and reference sets, which are presumed to be the same points, then estimating the rotation and translation needed to move the input cloud to minimize the distance of these pairs. Finally, the transformation is applied and the process is repeated, hence the iterative nature, a single iteration of which can be seen in Figure 2.4. The algorithm is described in a more formal manner in chapter 3.

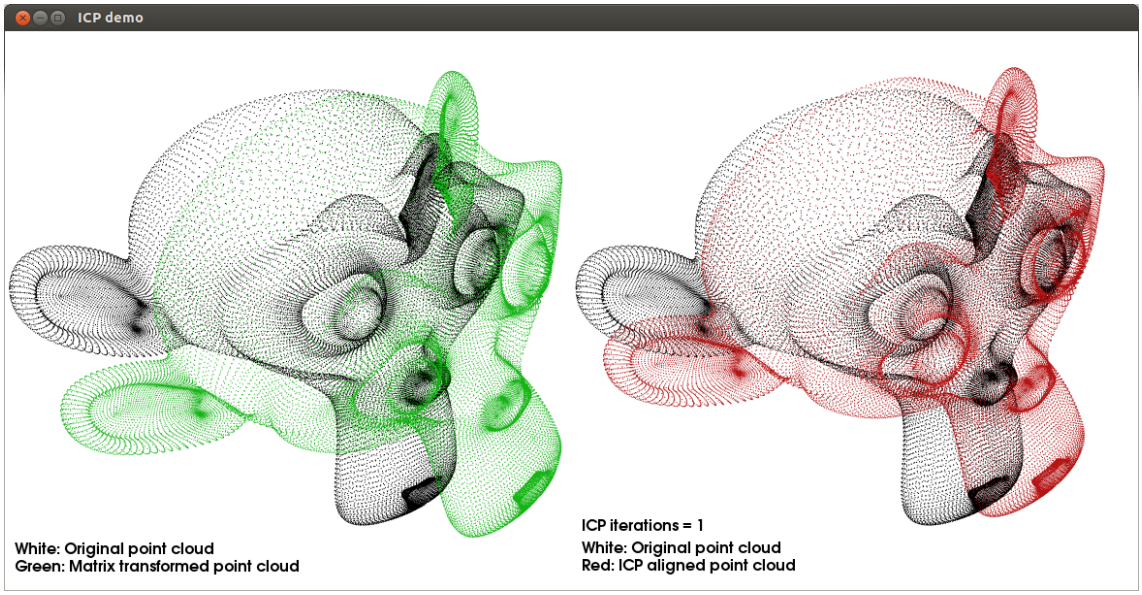

Figure 2.4: A single ICP iteration visualized [7]

The difficulty with this method is that like other gradient descent algorithms, it relies on having a decent enough input, so as to not get stuck on a local minimum. For this reason in use cases like 3D model scanning these pair points are often registered manually. Also

as plenty of points in the cloud are used in each iteration, it is considered computationally slow.

2.3.3 LiDAR Odometry and Mapping

A more recent widespread algorithm is LOAM [8] (*LiDAR Odometry and Mapping*), another SLAM approach proposed by Zhang and Singh, which as the name suggests, separates the odometry and mapping tasks and solves them individually. The first of the two algorithms running in parallel is high-frequency but coarse velocity estimation, roughly placing the frame, whereas the second, lower-frequency one corrects motion distortions in the point cloud, refining the end result. An overview of this process can be seen in Figure 2.5.

For the first step, this approach makes use of extracted plane and edge features via the minimization of point-to-plane and point-to-edge distances for the feature points. After this initial mapping, the refining step performs batch-optimization similar to ICP. This being lower frequency gives this step more time to reach convergence and increase accuracy, whereas the parallel nature makes real-time use feasible. Also because of its modularity, incorporation of other helper systems, like IMU for odometry is convenient.

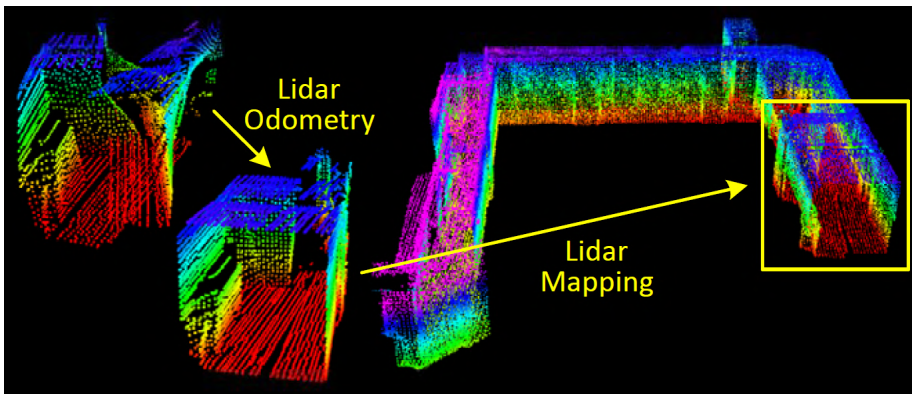

Figure 2.5: The two step localization and mapping [8]

2.3.4 State-of-the-art methods with LOAM foundations

Building upon these foundations, there have been a number of novel approaches in the recent years. F-LOAM [9] by Wang et al. aims at further improving real-time performance of the LOAM framework, by removing the iterative part and replacing it with a 2-stage lightweight distortion compensation. Their proposed model achieves this with features which tend to be extracted multiple times through overlapping scans. Using these and

their surrounding geometry the model manages to produce competitive results while being computationally less demanding.

BALM [10] is another model built upon a LOAM back-end, with the aim of reducing the accumulated registration error inherent to the frame-wise registration scheme. It does this by using bundle adjustment in a sliding window in which already registered scans can be readjusted, resulting in smaller drifts over longer mappings compared to regular LOAM.

Another approach to increase accuracy was demonstrated in M-LOAM [11]. This model uses multiple LiDAR sensors to help with data sparsity, along with a modified LOAM framework to achieve a robust and accurate system and provide a solution to the issue of calibrating multiple input sources.

2.3.5 Machine Learning approaches

It is also notable that with the popularity of machine learning in autonomous driving and road detection, ML approaches have been applied to the mapping problem as well. One of these recent models [12] combines conventional cameras and LiDAR in order to compensate for the shortcomings of each device and achieve a robust system less reliant on external conditions. With the added novelty of the two inputs being fused in a trainable fashion, the final performance was competitive with that of the state-of-the-art.

2.4 Fusion

As discussed earlier, several options exist to estimate the pose difference between two LiDAR frames, however each method has its shortcomings. The solution for these is to use a fusion of two or more of the previously mentioned sources of information.

As the research [13] of shows, using a GNSS / IMU / ODO / LiDAR-SLAM combination with high quality sensors resulted in robust tracking Chang, Niu, and Liu. Where the GNSS is not available, the IMU / ODO / LiDAR-SLAM provides accurate tracking even at places where LiDAR data is not usable e.g. tunnels with little to no identifiable features via ODO / IMU even at 30 and 60 km/h speeds. As seen in Figure 2.6, the navigation errors were under 7 meters on each axis even during the simulated 2 minute long GNSS outages. The heading errors were kept under 1° as well.

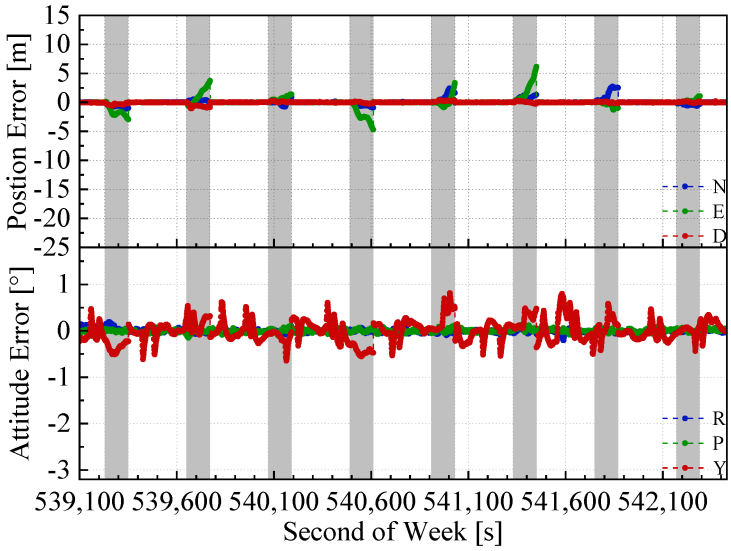

Figure 2.6: GNSS/IMU/ODO/LiDAR-SLAM navigation errors

Zuo et al. demonstrated, that IMU and LIDAR data can be complemented by higher frequency camera data, which can help determine movement between LiDAR scans by mapping LIDAR features to visual features [14]. This fusion resulted in very small average trajectory errors and was robust against aggressive movement. As Figure 2.7 shows, the LIC-Fusion proposed by the authors provided the most accurate tracking compared to the LOAM and MSCKF (*Multi-State Constraint Kalman Filter*) frameworks.

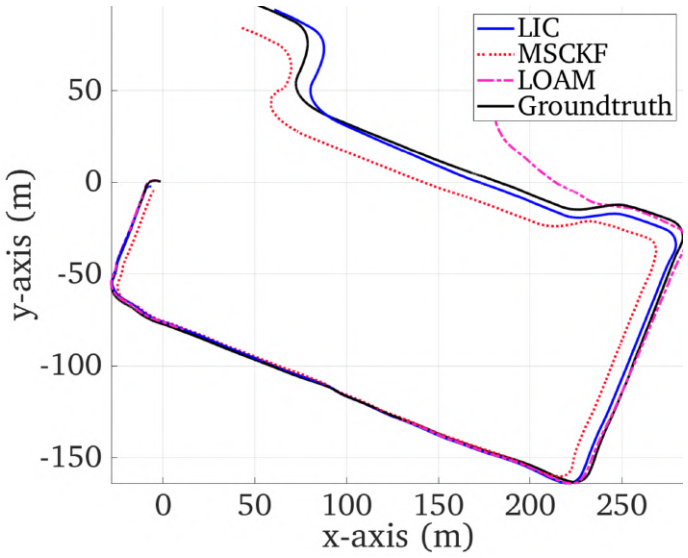

Figure 2.7: Top view of outdoor sequence trajectories

Discussed by the study of Xue, Fu, and Dai, an IMU / LiDAR / ODO combination can supply the LOAM algorithm with an initial guess for the pose [15], which the algorithm can use to calculate a significantly more precise result. This approach then

uses an extended Kalman filter to fuse the estimations. During an approximately 1.1 km long test trajectory with a velocity of 21 km/h a mean relative position error of 0.31% was reached.

Karam, Lehtola, and Vosselman examine multiple possibilities for fusing SLAM and IMU estimations [16]. The most accurate results compared to switching and exclusive approaches were produced by constructing a cubic spline from the two linear ones provided by the two information sources. This joint approach produced a root mean square error of 0.63° when reconstructing perpendicular surfaces of a data set which could not be processed by the IMU or the SLAM approaches separately.

As seen in the studies shown earlier, relying solely on SLAM algorithms or sensors alone is only effective in certain scenarios whereas fusing their outputs can produce a more robust and versatile tracking solution. According to this, our approach described in detail in the next chapter is based on fusion as well.

Chapter 3

Methodology

3.1 Hardware

Portability and accessibility were among our goals when choosing the hardware components of the project. We achieved this by only using devices that were available for the public to order online either on the manufacturer's site or a reseller. The hardware composition consists of a LiDAR sensor, an external GPS unit, an onboard computer, an IMU unit and a carrying vehicle. In the next sections, we are going to introduce the chosen components and their qualities which were important in terms of our research.

3.1.1 The carrying vehicle

The vehicle on which our devices are carried is not an integral part of the design, it can be an arbitrary vehicle with very few constraints. The 3 main criteria are to keep vibrations as low as possible, avoid magnetic interference and provide sufficient space for the other components and a battery. During our measurements we used multiple vehicles depending on the availability and the type of the measurement. Two examples of what a complete measurement setup looks like can be seen in Figure 3.1 for indoor configuration and Figure 3.2 for outdoor configuration.

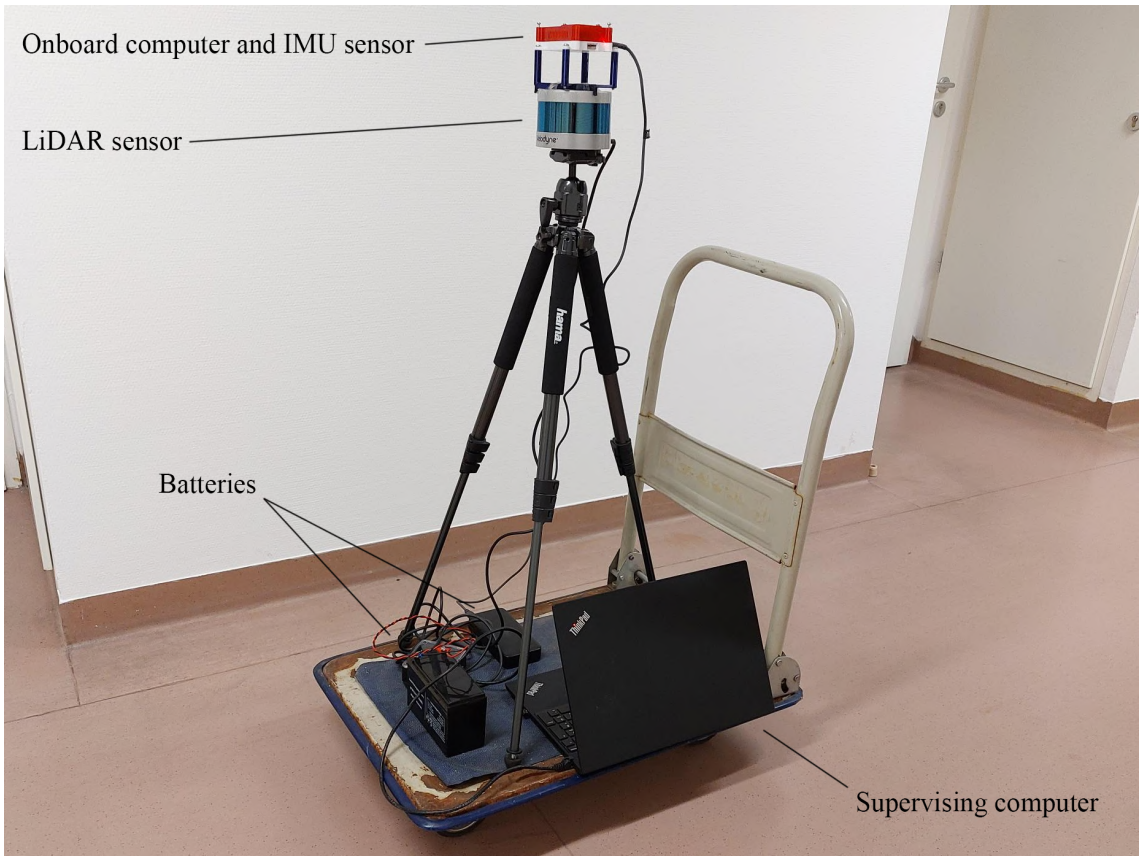

Figure 3.1: Indoor hardware composition

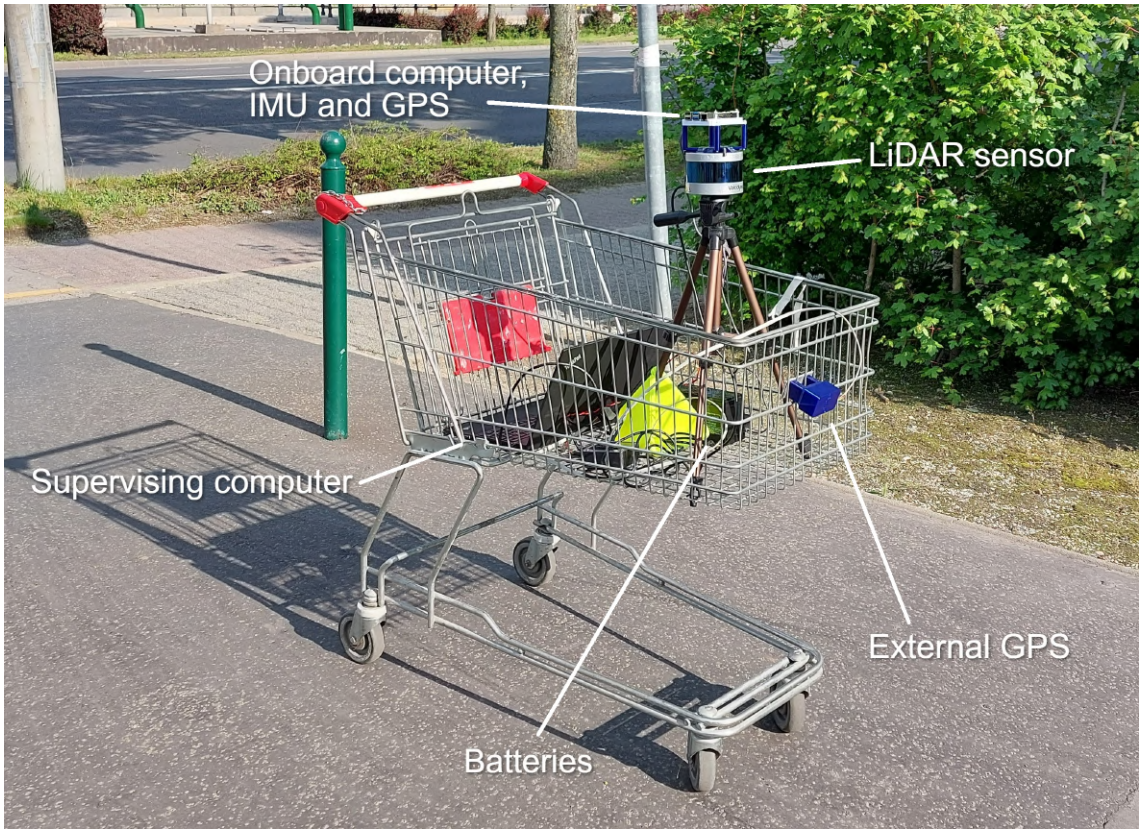

Figure 3.2: Outdoor hardware composition

3.1.2 The onboard computer

As portability was an important aspect during the hardware selection, we needed a versatile computer to read and process IMU and LiDAR data with a form factor small enough to be able to be attached to the LiDAR sensor. As seen in Figure 3.3, the Raspberry Pi 4 [17] fits these criteria as in spite of its smaller size, it is equipped with the necessary I/O ports to carry out the measurements.

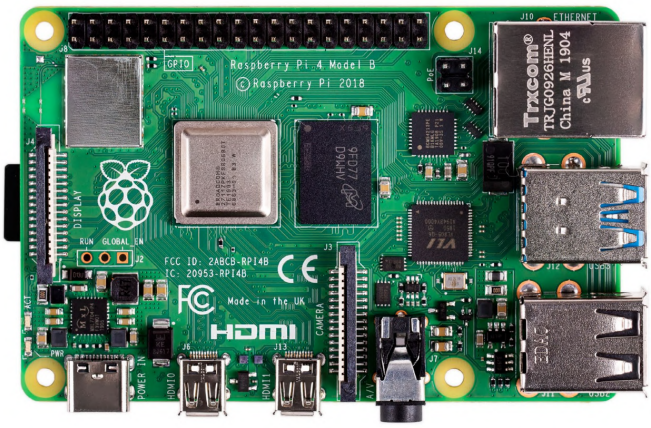

Figure 3.3: Raspberry Pi 4 model B

Our model is equipped with a quad core ARM v8 architecture SoC and 4GB of RAM. The operating system used during the tests was Raspberry Pi OS which is based on the Debian Linux distribution.

3.1.3 The LiDAR sensor

The LiDAR sensor we used during our research was the Velodyne VLP-16 [18]. This sensor was an adequate choice because of its versatile feature set and its size seen in Figure 3.4. Velodyne recently renamed this product to "Puck", due to its dimensions.

The VLP-16 scans its environment in 16 channels, each in a maximum range of approximately 100 meters. The 16 channels are arranged so that the vertical field of view is 30°. The sensor can create a 360° horizontal field of view by rotating its inner sensor with the help of an electric motor with a configurable frequency between 5 Hz and 20 Hz. Its horizontal angular resolution is between 0.1° and 0.4° depending on the rotation rate.

Figure 3.4: Velodyne VLP-16

A host computer can communicate with the sensor and record its output though an RJ45 ethernet port. The desktop software we used to record and replay LiDAR data is called VeloView. VeloView is specifically developed in order to visualize and record LiDAR data captured via Velodyne's sensors. The program is a flavor of ParaView developed by Kitware. VeloView alongside the various viewing assistance features seen in Figure 3.5, also provides a Python shell which enables the user to run script programs on the loaded data.

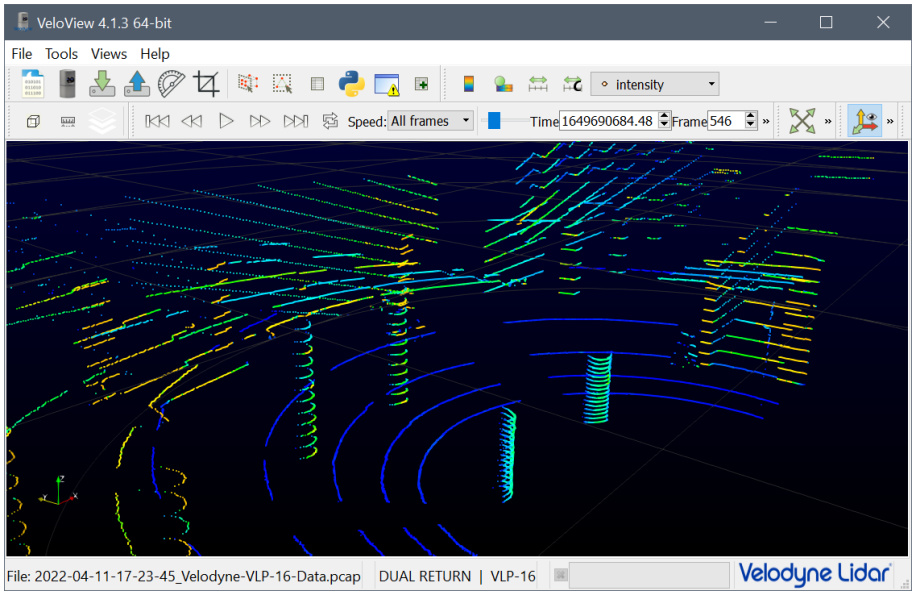

Figure 3.5: VeloView 4.1.3

The VLP-16 sensor also hosts a minimal webpage, which can be used to view and modify the sensor's settings and state without any dedicated programs.

3.1.4 The external GPS unit

Our external GPS unit utilizes a MediaTek MT3339 all-in-one GPS chipset [19]. It provides us with 66 acquisition-channels and 2.5 m location accuracy. The unit is connected to the LiDAR sensor's GPS port, which is specifically designed to be compatible with Garmin sensors, so a custom made adapter was required for us to be able to connect these devices.

3.1.5 The inertial measurement unit

Our choice for the IMU sensor was the BerryGPS-IMU V4 designed by OzzMaker [20]. The BerryGPS-IMU V4 as seen in Figure 3.6 is fitted with the following versatile array of sensors: GNSS (CAM-M8 module from uBlox), accelerometer, gyroscope, magnetometer, barometric and temperature sensor. In spite of the versatility, the BerryGPS is among the more affordable IMU solutions on the market today. Furthermore, the unit was specifically designed to fit Raspberry Pi computers, which caused the design procedure of the hardware composition to be notably more convenient.

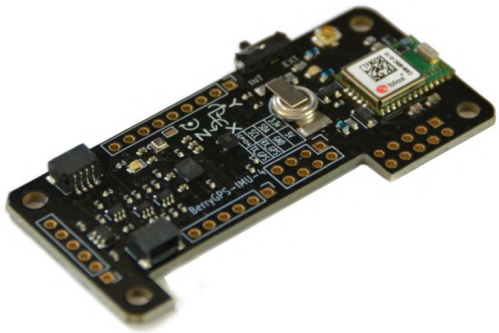

Figure 3.6: BerryGPS-IMU V4

This unit supports a maximum linear acceleration sensitivity of 0.061 mg/LSB, angular rate sensitivity of 4.375 mdps/LSB and magnetic induction sensitivity of $1.46156 \cdot 10^{-4}$ gauss/LSB. During our measurements, the sensitivity values were set to 0.244 mg/LSB, 70 mdps and $2.92312 \cdot 10^{-4}$, respectively.

The IMU board is connected to the Raspberry Pi through its GPIO (general-purpose input / output) pins and communicates via an I^2C communication bus.

The following section presents our experience with the previously detailed equipment.

3.1.6 Measurement experiences

We managed to reach a high level of portability with our hardware choices. Our measurement kit can be set up with a wide variety of carrying vehicles without significant modifications. The lack of violent shaking however turned out to be an important aspect as uneven ground surface or poor vehicle quality significantly impaired the IMU's positioning capability.

The LiDAR sensor's output was reliable throughout our experiments. The only problems we encountered were when *i*) moving around reflective glass surfaces, artifacts were detectable in the point clouds; and *ii*) uneven ground could produce distortions in the measurement due to the wobbling of the sensor.

The inertial measurement unit provided us with a few challenges. The gyroscope drift needed to be compensated for, because we measured a slight rotation even after passing the measurement data through our filter algorithm in 6 DOF mode. The gyroscope drift could also be corrected via the magnetometer, though it was prone to output data which could not be processed due to magnetic interference. After our first measurements, we concluded that the electric motor which provides the spin for the LiDAR sensor creates too much interference for the IMU's magnetometer for it to be placed right on top of the LiDAR housing. Our solution was to use a 3D printed structure seen in Figure 3.7 to hold the onboard computer with the IMU close to the LiDAR device, so their movement is as synchronised as possible, but far enough so that the motor's electromagnetic pulses are not detectable in the magnetometer data.

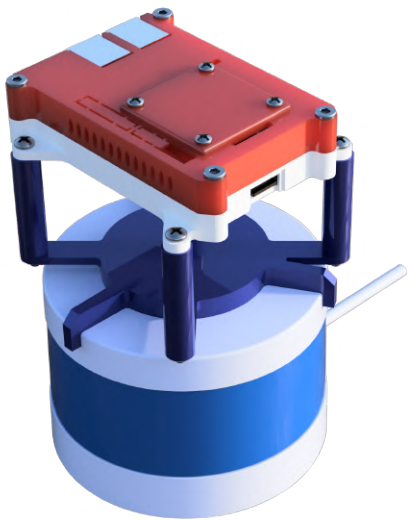

Figure 3.7: Mounting structure for the onboard computer and IMU

Although the 3D printed holder solved the interference with the electric motor, unprocessable data still occurred to an extend in our measurements even after hard and soft iron calibration (detailed in 3.2.3).

The reliability of the onboard GNSS sensor of the BerryIMU was proven to be questionable during our research. The sensor took multiple, in some cases more than 10 minutes to find GNSS satellites after a cold start. After acquiring a signal, the calculated positions were often not usable by our positioning pipeline. Figure 3.8 shows the path given by the onboard and external GPS outdoors next to a building compared to the ground truth.

Figure 3.8: Comparison of the external GPS and onboard GNSS sensor to the ground truth

Our external GPS provided better quality results, which we could feed into our pipeline. Occasional long startup times and long signal regain times did occur here as well, however the overall performance was sufficient for our needs.

In the following sections, we are going to elaborate on the software architecture applied in this research.

3.2 Software

3.2.1 GNSS based positioning

In our fusion based approach one major component is GNSS (*Global Navigation Satellite System*) [21] based positioning. These are based off of the fact that one's 3D position can be calculated using the coordinates and distances to any 3 points in space. While by now a number of systems exist, for our purposes we used GPS (*Navstart Global Positioning System*) for its easy availability and due to hardware requirements.

The original GPS file recording based positioning was established by Roxána Provender in her previous work on the project [22], which is briefly summarized in

this subsection, but also extended with a new and significant contribution of our work: real-time data reading from a sensor connected to the LiDAR.

For the base of the accuracy scores of the GPS measurements we used the so-called DOP (*Dilutation of Precision*) [23] info available. This is a scale of $n > 1, n \in \mathbb{N}$ values based on the number of satellites visible to the sensor, and their position relative to each other, with a lower score indicating higher confidence. As for our purposes measurements with 11+ DOP values[1] can be considered of little value, we map these values using the following formula:

$$Accuracy = \frac{100}{11}(11.5 - DOP)$$

which maps the 1..11 DOP values into a 0-100 scale in a linear fashion.

For offline use, the first step is pre-processing the GPS input. This is necessary to interpolate data for timestamps which do not have an associated measurement, as well as refining the measured path, and is done by running the data through a Kalman filter[24]. The accuracy scores of data synthesised by the filter are based off of the accuracy of the previous point, with decreased score for new points with changing directions from the previous segment. The effects of this on a path near a tall building with lack of clear signal can be seen on Figure 3.9.

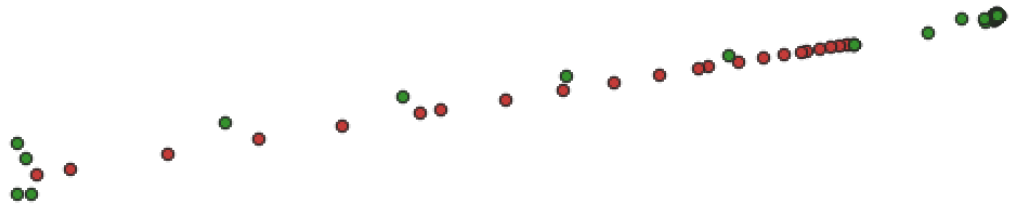

Figure 3.9: The original route (green) corrected by Kalman filter (red)

Real-time GPS data matching has its own set of challenges as well. While the pre-recorded data contained Unix timestamps, and as such were available on a 1 measure/second basis, the data provided by the VLP-16 has microseconds based stamping described in section 4.3. By experimenting with the data we found that rounding these to seconds worked without much loss of detail, as the multiple data points within the second provided almost the same measurement anyways.

[1] Interpretation of DOP values:
`https://en.wikipedia.org/wiki/Dilution_of_precision_(navigation)`

Another issue was the extrapolation of missing data, which could either be because of the lack of signal, or simply the data not arriving in time due to the asynchronous nature of the processing. The extrapolation itself is by the use of N-th grade polynomials using 2^N data points the following way:

- Let's denote the measurement points as $p_1, p_1, ...p_{2^N}$, with their matching timestamps being $t_1, t_2, ...t_{2^N}$.
- First we calculate the first order derivatives $d_{1,1}, d_{1,2}, ...d_{1,2^N-1}$

$$d_{1,j} = \frac{p_{j+1} - p_j}{t_{j+1} - t_j}$$

  Note that because of the uniform distance between the data points, the $t_{j+1} - t_j$ part will always be equal to 1, thus the denominator can be left out.

- In a similar fashion we calculate all the higher order derivatives $d_{i,1}, d_{i,2}, ...d_{i,2^N-i}$ for $i = 2..N$

$$d_{i,j} = d_{i-1,2j} - d_{i-1,2j-1}$$

- Finally, the new data point is extrapolated using the last point and the last derivatives of each order.

$$p_{result} = p_{2^N} + \sum_{i=1}^{N} d_{i,2^N-i}$$

The measurements are stored in a latitude longitude elevation system. To get the required transformation, we first calculate the distance between our current data point and the previous one[2]. This is done by the usage of the Haversine formula[25], which determines the great-circle distance between two points on a sphere. The formula itself is the following:

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\phi_2 - \phi_1}{2} + \cos\phi_1 \cos\phi_2 \sin^2 \frac{\lambda_2 - \lambda_1}{2}}$$

with R being the radius of the sphere, Earth in this case, ϕ being the latitude and λ the longitude coordinates.

[2]For the first data point we return the identity transformation.

In order to separate the horizontal and vertical components, the other part can be fixed, resulting in the following equations:

$$\Delta x = 2R \arcsin \sqrt{\sin^2 0 + 2\cos\phi_1 \sin^2 \frac{\lambda_2 - \lambda_1}{2}}$$

$$\Delta y = 2R \arcsin \sqrt{\sin^2 \frac{\phi_2 - \phi_1}{2} + \cos\phi_1 \cos\phi_2 \sin^2 0}$$

It's of note that this step also transforms the problem from the geographical coordinate system (lat-long) into the a Cartesian one (x-y-z). After the Δx and Δy values are calculated, the rotation angle is calculated using the formula below:

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

It should be kept in mind, that from the LiDAR's perspective, the forwards direction is the one its traveling towards, for example, in the first case $\frac{\pi}{2}$ if $\Delta x > 0 \wedge \Delta y = 0$ we can see from $\Delta x > 0$ that the sensor moved straight towards East, meaning a $\frac{\pi}{2}$ rotation from facing North.

It's also noteworthy that the elevation can also change during the measurements. It's not generally available throughout all sources, for example, it's not present in the currently parsed online GPS sources, but some file recordings were made with GPS having such data, which can be used to further increase the accuracy of the transformation. These are just the height of the measurement point in meters, and as such their difference can be quite easily calculated:

$$\Delta z = \tau_2 - \tau_1$$

where τ is the elevation.

Similar to the rotation, this by itself is not an accurate result, as the LiDAR itself tilts on a slope, thus perceiving it as straight ground, and so the elevation angle needs to be

calculated as well:

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

Finally, after having all the translation and rotation components the affine transformation can be assembled with ϕ being the rotation along the z axis and τ along the x axis:

$$R = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

For the translation, while we have the distances along each axes Δx Δy Δz, these are all based on a North facing observer, and to form the required transformation the inverse of the previous rotation must be applied to it:

$$t = R^{-1} * \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix}$$

3.2.2 ICP based positioning

As briefly already introduced in chapter 2, the iterative closest points algorithm matches two point clouds by attractively minimizing an error function.

The algorithm usually performs several prepossessing steps before the execution of the scan alignment to reduce computational complexity and extract further information [26].

Point down sampling For faster execution this step down sample the input point cloud. It uses random systematic sub-sampling and keeps one in every 5 points.

Normal calculation This step computes normal vectors for every points in scan. The normal computation is performed by Principal Component Analysis (PCA) with 10 nearest neighbours of each point. This step is important for the ICP algorithm described in next section.

Observation vector calculation This step calculates observation vector for each point in the cloud. The observation vector is the vector which points from the origin to the point.

Normal orientation This step orients the normal vectors so the angle between observation vector and normal vector is minimal. This is carried out for ensuring that every normal points to the same direction.

Given two independently acquired (and optionally preprocessed) sets of 3D points, M (model set,$|M| = N_m$) and D (data set,$|D| = N_d$) which correspond to a single shape, we want to find the transformation (R,t) consisting of a rotation matrix R and a translation vector t which minimizes the following cost function:

$$E(R,t) = \sum_{i=1}^{N_m} \sum_{j=1}^{N_d} w_{i,j} ||m_i - (R * d_j + t)||^2$$

where $w_{i,j} = 1$ if the i-th point of M and the j-th point of D are the same point in space, otherwise it's 0.

In each iteration first the point correspondences, then the transformation (R,t) which minimizes $E(R,t)$ with the correspondences is calculated. The pseudocode of the algorithm can be seen in Algorithm 1.

Algorithm 1 ICP algorithm

```
for i = 0 to maxiterations do
    for all d_j ∈ D do
        Find the closest point within a range d_max in the set M for point d_j.
    end for
    Calculate transformation (R,t) that minimizes the error function.
    Apply said transformation to point set D.
    Calculate the quadratic error ||E_{i-1}(R,t) − E_i(R,t)|| before and after the
      transformation. If their difference is smaller than threshold ε, terminate.
end for
```

It is noteworthy, that if the input point set is not in a decent enough position to begin with, the algorithm can get stuck on a local minimum, hitting the threshold without finding the actual proper match, or might not reach the threshold within the fixed iteration number.

The previously presented ICP algorithm is used to register subsequent point clouds with some refinement. For more robust working for each point in the model point set 3 points of the data set are matched. The distance between the points are calculated using the following equation:

$$d = (m - (R * d + t)) * n$$

where n is the oriented normal of the model point calculated in during the data prepossessing. The calculated distance showed in Figure 3.10.

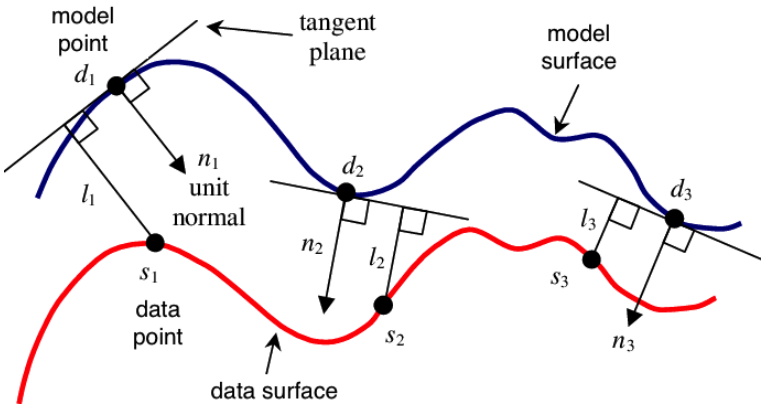

Figure 3.10: Point to plane distance [27]

Weights in the error function set to zero if the distance d between two points is greater than $1m$ or the angle between their normal's if grater than 60 degree. This makes ICP to consider only matches between points that belongs to similar planes. The number of maximum iteration set to 80 and the the difference threshold is $0.01m$ and $0.001rad$. The algorithm also terminates is the calculated t goes beyond $15m$ or R beyond $0.8rad$.

Increase resolution with local maps and keyframes

An issue with LiDAR sensors – especially with more affordable ones – is that the produced point clouds have a spatial resolution that drastically reduces with distance: objects have different spatial resolution on scans recorded from different distances. As a result, the precision of the computed normal between the data and model scans is different, and matches can be established on points that do not correspond to the same scene element.

To increase the resolution local maps are used as model point clouds which are basically concatenations of 3 keyframe scans. Keyframes are certain selected scans. As the mapping process starts the first perceived point cloud becomes the first keyframe and also the first local map. A point cloud can becomes a keyframe if the overlaping between it and the current local map drops below 75%. Not every scan is necessarily stored in the mapping process to avoid of insert further points to the map in previously visited areas.

3.2.3 IMU based orientation estimation

Theoretically, determining the orientation is a simple mathematical problem with the sensor readings and elapsed time as parameters. However, these sensor inputs can not be handled as perfect data sources as there are hardware limitations for the accuracy and interference tolerance as well.

AHRS filter algorithm

Working with raw sensor data would result in a jittery tracking pattern which not only would differ from the ground truth, but would not be syncable with the LiDAR data either. Because of this the raw sensor data is passed into an AHRS (Attitude And Heading Reference System) filter such as a Kalman filter, which continuously updates its inner state with new sensor readings and filters the unwanted spikes which have a high probability of being the result of temporary interference or inaccuracy. Further details about our choice of filter implementation can be found in 4.7.

Magnetometer calibration

As the magnetometer on our IMU can only provide uncalibrated measurements, we needed to introduce soft and hard iron calibration on our data.

Hard iron bias occurs when an object in the immediate environment of the sensor produces a constant additive magnetic field, distorting the magnetometer's measurements by an offset in X, Y and Z directions.

The Fusion library provides a parameter to feed our calibration matrices into, however the matrices need to be calculated and given to the algorithm. This bias can be measured by rotating the sensor around all of its axes, then solving the following simple equations:

$$1. \qquad h_x = \frac{x_{max} + x_{min}}{2}$$

$$2. \qquad h_y = \frac{y_{max} + y_{min}}{2}$$

$$3. \qquad h_z = \frac{z_{max} + z_{min}}{2}$$

After calculating the hard iron bias, the calibrated measurement values can be calculated by $c = u - h$ where u is the uncalibrated measurement vector, h is the hard iron offset vector and c is the calibrated measurement vector.

The cause of soft iron bias is similar to the previously detailed offset, however in case of soft iron distortion the magnetic field of the object is not additive to Earth's field, but distortive. In this case, after rotating the magnetometer around all of its axes, the measured points show an ellipsoid orientation instead of the sphere it is theoretically supposed to. The theoretical equation for applying soft iron correction is $c = S \cdot m$, where c is the calibrated measurement vector, S is the soft iron calibration matrix and m is the uncalibrated measurement vector.

Our soft iron correction calculation is based on Kris Winer's explanation[28] of a computationally less expensive method which is suitable for our application as we avoided strong static magnetic fields during our measurements. With this method our soft iron calibration matrix S looks like the following:

$$S = \begin{bmatrix} \frac{f}{g_1} & 0 & 0 \\ 0 & \frac{f}{g_2} & 0 \\ 0 & 0 & \frac{f}{g_3} \end{bmatrix}, \text{ where}$$

$$g = \begin{bmatrix} x_{max} - x_{min} \\ y_{max} - y_{min} \\ z_{max} - z_{min} \end{bmatrix} \text{ and } f = \frac{g_1 + g_2 + g_3}{3}$$

As S is a diagonal matrix, no actual matrix operations need to be performed in order to calculate soft iron bias.

After applying the calibrations, the calibrated values visualized should look like a sphere around the center of our coordinate system as the example shows in Figure 3.11.

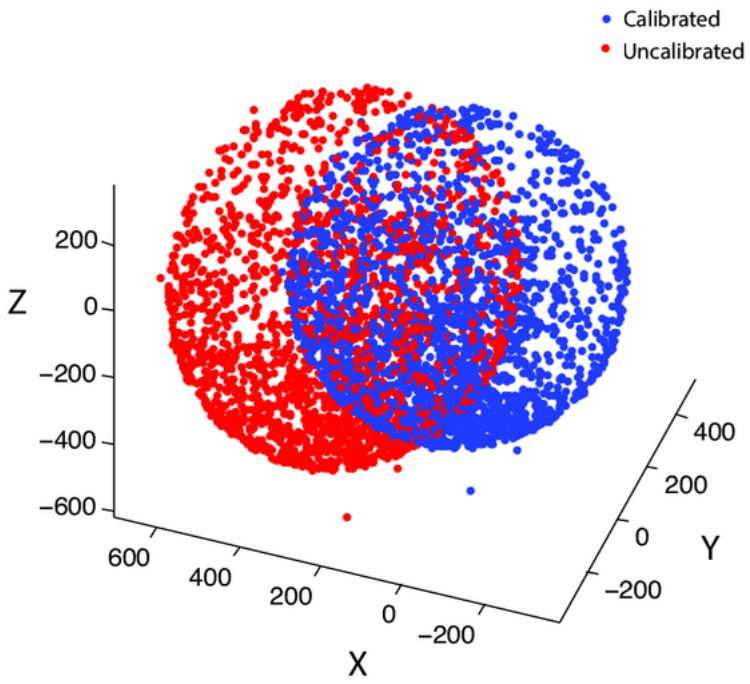

Figure 3.11: An example for magnetometer measurements before and after calibration. Axes represent magnetic field intensity in milligauss. [29]

Chapter 4

Implementation

While designing the software system, the main idea was to build a robust framework which enables easy integration of various mapping methods and supports further extensibility in the future. We implemented our prototype solution as the continuation of an already existing project available online on GitHub[1]. The implementation was carried out in standard C++14 (primarily due to the existing codebase), with a wide usage of the PCL[2] software library.

4.1 Architecture overview

From a higher abstract overview, the program implements the *producer-consumer design pattern*, with the product being the assembled map (the merged point cloud) up until that point in time. The producer begins with a *grabber*, which is responsible for providing the input LiDAR cloud, as described in section 4.3 in details.

After grabbing the input cloud, it is forwarded to the processing pipeline. This pipeline is assembled of so called *processors*, each representing a *Point cloud* $\rightarrow$ *Point cloud* custom method (modification) of a point cloud, which then becomes the input for the next processor in the pipeline. These processors can perform a variety of tasks, the most prominent of which are filtering the input, calculating and applying the transformation required for the mapping, or merging the new input cloud into our so-far built map, but the interface of taking a point cloud and returning one can be used for other tasks as well. Our system allows for easy customization of the pipeline, be it the selection of processors

[1] `https://github.com/mcserep/lidar-processor`
[2] `https://pointclouds.org/`

or their orders, both of which are arbitrary, although not all possible configurations are relevant for our purposes.

After the pipeline produces the final output cloud, it is given to various *consumers*. In our current system these perform desktop visualization in a window, or writing the point cloud into an output file. An example workflow in our model can be seen in Figure 4.1

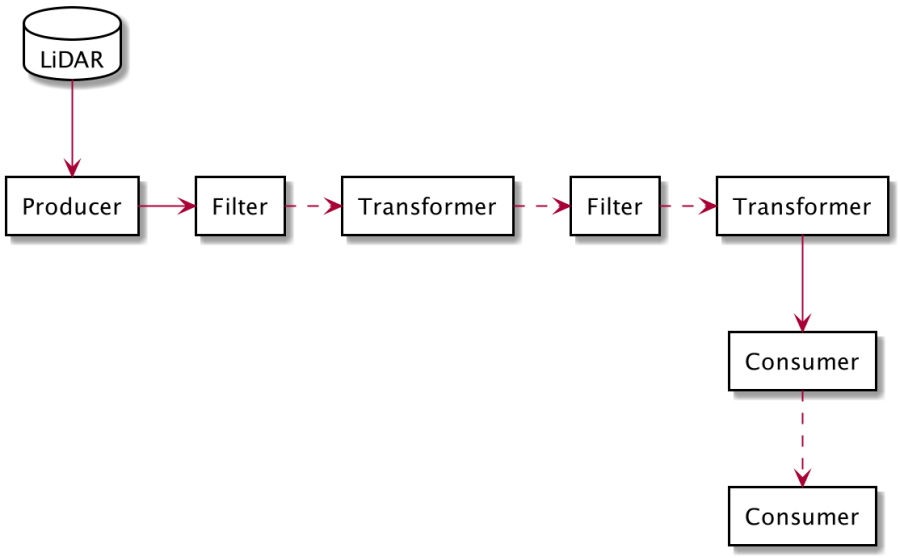

Figure 4.1: An example workflow in the model

4.2 Our contributions

Before our work, the application already had the general workflow laid out. As such, the producer, pipeline, filtering, and major transforming elements using offline GPS or ICP SLAM based positioning, along with the display and file writing capabilities of the program were already developed [22].

We extended the existing codebase with a multitude of features, the most prominent contributions being the following.

- In the previous version while using both ICP and GPS was possible, the two methods were not using the same transformation description system, which was refactored by making the GPS based positioning use *affine transformations* as well.
- We added support for reading the GPS data available from the LiDAR sensor, a process which required implementing our own grabber described in section 4.3, as well as solving the issues of synchronization and processing for this asynchronous source of data.

- For making use of the newly available IMU measurements, an ICP and IMU based transformation method was implemented.
- To improve user understandability of the visualization, trajectory display was added, with each segment having color based on the transformation method it was calculated with.

4.3 Grabber

On a high level the grabber is responsible for the task of producing the point cloud from some input source. For this PCL provides an out of the box solution, which takes either a PCAP file path or an IP address as an input, and after assembling the point cloud from the source, emits a signal for which a callback can be registered to.

One of our aims was reducing the number of input sources for all our data, as such we wanted to make use of the VLP-16's capability of having a GPS unit connected to it. These measurements are emitted just like the point cloud, but on a different port, 8308 by default. The structure of such a PCAP recording opened in WireShark can be seen in Figure 4.2. In the configuration seen there the packets of the point cloud are sent to the 2368 UDP port and have the payload length of 1206 bytes, whereas the GPS packet data arrive at the 8308 UDP port and have 512 bytes of payload.

As the grabbers in PCL only process the point cloud packets, for this purpose we implemented our own grabber extending on the VLPGrabber available in the library. For the structure of the data packet we consulted the official user manual [30]. Although according to the manual more unused bytes have been assigned meaning, for our purposes these were not relevant. The structure of the packets are described in Table 4.1, with an example shown in WireShark on Figure 4.3, with the non-unused parts being highlighted. While the NMEA sentences [31] themselves are of variable length, they are always terminated by the CR LF (0x0d 0x0a) bytes.

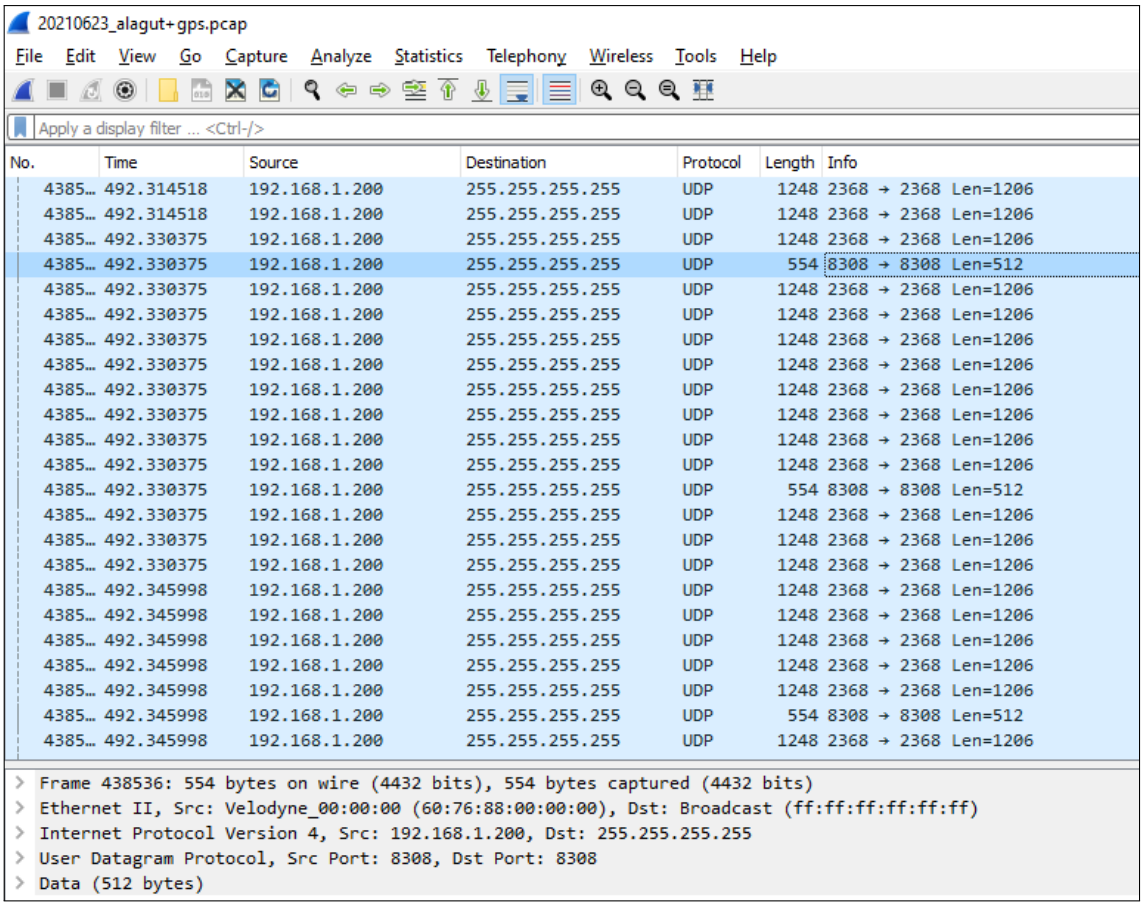

Figure 4.2: Packets recorded from VLP-16 with connected GPS

Number of Bytes	Total Byte Offset	Description
42	0x0000	UDP header
198	0x002A	unused
4	0x00F0	Timestamp (μs)
1	0x00F4	Pulse Per Second status, see Table 4.2
3	0x00F5	unused
variable length	0x00F8	NMEA sentence

Table 4.1: The structure of the GPS packets

Value	Description
0	No PPS detected
1	Synchronizing to PPS
2	PPS Locked
3	Error

Table 4.2: PPS status values

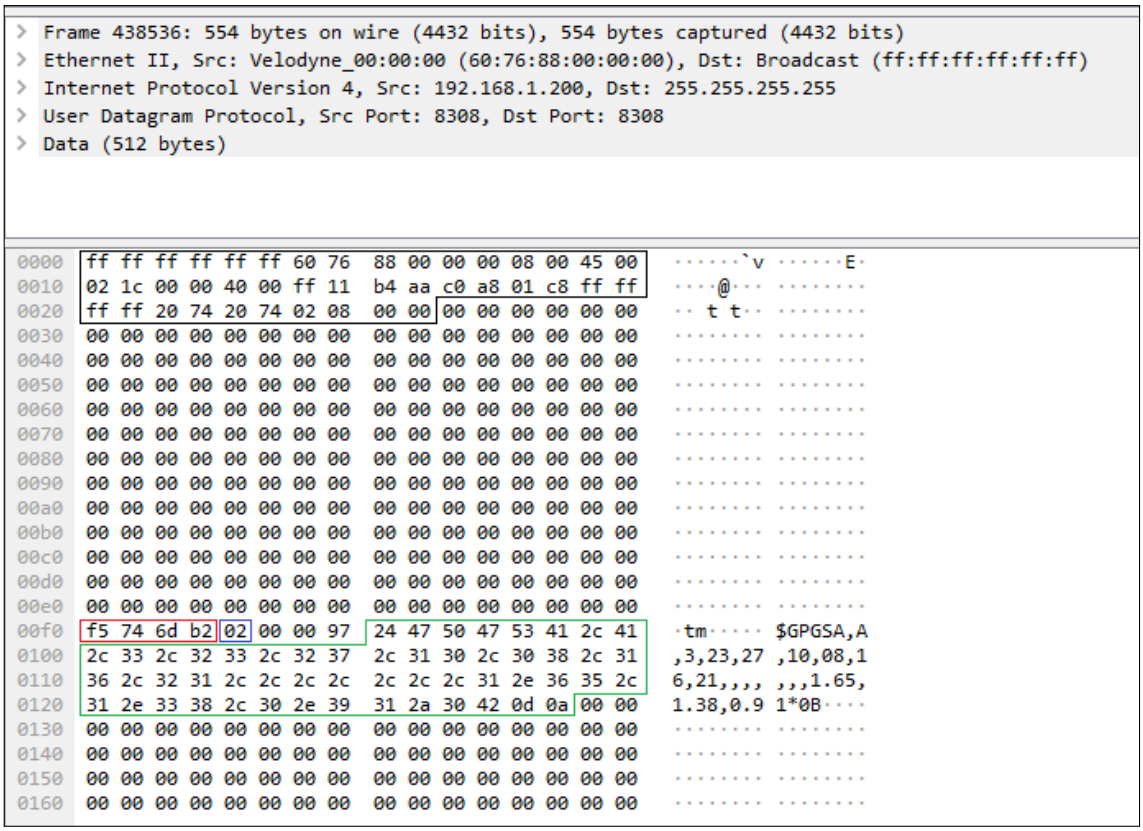

Figure 4.3: Structure of a GPS packet
UDP header (black)
Timestamp (red)
PPS status (blue)
NMEA sentence (green)

For the processing of GPS packets we followed a similar workflow to that of cloud packets, as can be seen in the source code of the PCL library[3]. As a first step the raw bytes are extracted and put into an asynchronous queue by a worker thread. After this another thread observing the queue pops the packet, parses the timestamp, the PPS, then the NMEA sentence, emitting a signal with a unified lightweight packet format only containing latitude, longitude, elevation, and accuracy information.

The timestamp itself is a 4 byte integer being the number of microseconds elapsed since the top of the hour, and as such is inconvenient to work with, so it is transformed by putting it on the lower 4 of 8 bytes and adding the lower 4 bytes of the current unix time as its top, as it can be seen in Code 4.1. The reasoning for this was simply consistency with the cloud packets, as they use the same timestamp generation.

The NMEA sentences in the packets, according to the user manual, are either in `GPRMC` or `GPGGA` format, but in practice looking at the packages we found plenty of

[3] `https://github.com/PointCloudLibrary/pcl`

other formats as well, such as GPVTG or GPGSV sentences. While these could provide useful additional information, for the time being in our prototype system only GPRMC and GPGGA messages[4] are parsed. This was done as they have sufficient information to produce lat-long measurements for each individual packet, and frequent enough to form a data set of usable size, with multiple data points for every second. It should be noted that GPRMC messages do not contain DOP information, and as such a fixed 60 value is used for these as a temporary accuracy score.

```
#include <time.h>

std::uint64_t timestamp;
time_t system_time;
time (&system_time);
timestamp =
    (system_time & 0x00000000ffffffffl) << 32 | packetTimestamp;
```

Code 4.1: C/C++ snippet for calculating the timestamp

4.4 The Producer

At design level this part of the workflow is the actual source of the input clouds. The public interface of the abstract Producer class can be seen in Figure 4.4. It defines the start and stop methods, to control the emission of data, and two handler registration methods, the registerHandler for point clouds, the registerGPSPacketHandler for GPS packets.

[4]For the structures of these, see:
https://docs.novatel.com/OEM7/Content/Logs/GPRMC.htm
https://docs.novatel.com/OEM7/Content/Logs/GPGGA.htm

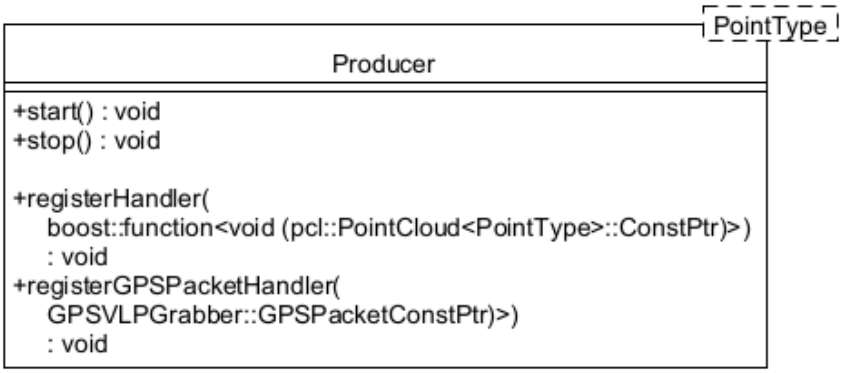

Figure 4.4: The `Producer` class

The class inherited from this `Producer` class is the `GrabberProducer`, which wraps a PCL grabber and exposes its supplied data by the two handler registration methods.

PCL provides many point types for extra information beyond X, Y and Z coordinates, such as intensity, but for our purposes the choice between these is arbitrary, with our final type being `PointXYZI`, although this is easily modifiable as a template parameter throughout our architecture.

4.5 Processor pipeline

The processor pipeline is a series of so called processors, each of which takes an input point cloud, does some operation on it, then returns the modified cloud. The configuration we used is visible on Figure 4.5.

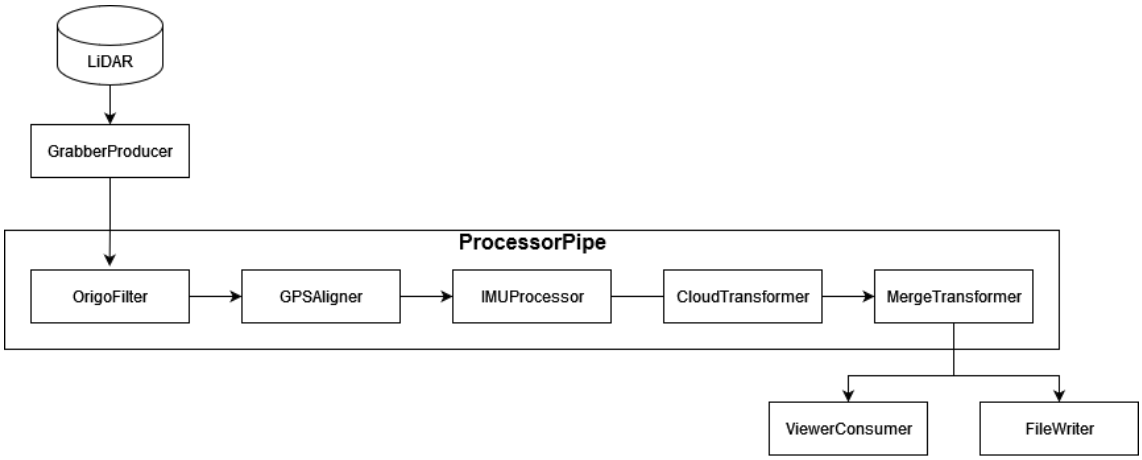

Figure 4.5: The used configuration

First, as already established, the `ProcessorPipe` receives its input data from the `GrabberProducer`. In the pipe itself first we apply some filtering to the point cloud, namely removing all points within a certain distance of the origin, in order to filter out elements like the carrying vehicle or the person pushing the cart. Empirically we found that filtering in a 2 meter wide, and arbitrarily large (32m in our case) high column was sufficient for this.

For the next part of the pipeline we have the `GPSAligner`, which simply makes the cloud processing thread wait until a GPS packet with a greater timestamp than that of the cloud arrives, or the 1 second timeout expires. This is necessary because of the parallel nature of the point cloud and GPS packet processing in the grabber. It is also noteworthy that the VLP-16 sensor prioritizes the emission of point cloud data above that of the GPS packets, meaning that a GPS packet arriving later in time could have a smaller timestamp than a cloud packet which arrived earlier, making this sort of synchronization even more necessary.

After the time synchronization, the `IMUProcessor` calculates the estimated orientation of the sensor array (detailed in section 4.7) and pre-rotates the point cloud, so as the SLAM based calculators can work with smaller differences when comparing the current reading with previous results.

The `CloudTransformer` is responsible for calculating and applying the transformation to properly map the current cloud frame. It contains a variety of calculators, each based on a different method, like GPS or ICP based positioning. These calculators take the current cloud frame as an input, and return the calculated affine matrix [32] along with an accuracy score. After receiving the transformations the `CloudTransformer` chooses the one with the highest accuracy score and applies it to the cloud frame. The transformation itself is done incrementally, keeping track of the total transformation up until that point, and only calculating the one step required to transform the frame from the previous position to the current.

The last part of the pipe is the `MergeTransformer`, which takes the transformed cloud frame and merges it into the final map using an octree data structure [33]. After the final map leaves the pipe, it is then displayed in a desktop window using PCL's built-in `PCLVisualizer`, which renders the cloud map along with the trajectory of the sensor. The transformed cloud is also exported as a LAS/LAZ file.

4.6 Calculators

In our infrastructure calculators are the objects responsible for calculating the transformation required to map the current cloud frame. For the uniform description of transformations we use affine matrices provided by the *Eigen library*[5]. The class diagram of the abstract calculator base class, as well as the `TransformData` class used for the coupling of the affine transformation and the accuracy score, are shown on Figure 4.6.

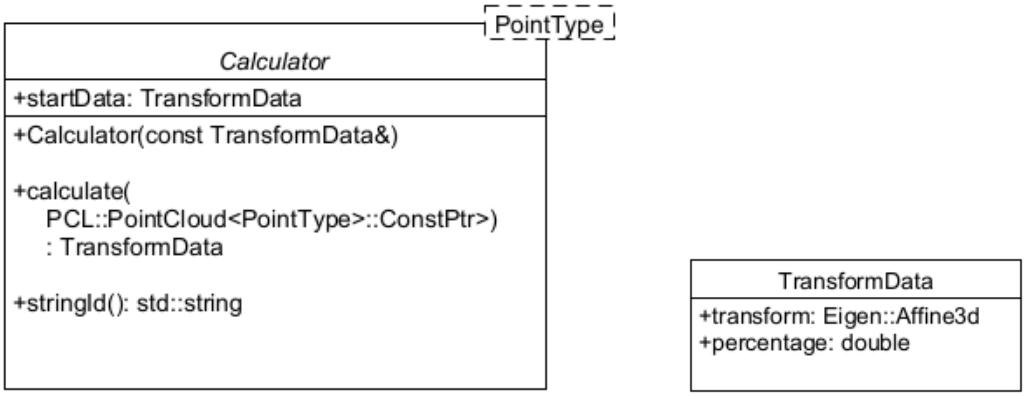

Figure 4.6: The `Calculator` and `TransformData` classes

While the `stringId` method is only used to differentiate between the positioning methods on display, the `startData` member serves a functional role. As during the run of the program we could be switching between different calculators, it is important to keep the individual calculators informed about what the latest actual transformation is, for which this member is used.

The offline GPS based positioning, as stated at the beginning of the chapter, was already implemented before we started our project [22]. For the online positioning another calculator was implemented, which receives GPS measurements and stores them in a thread-safe queue. While the original had support for the EOV projection system[34] available in Hungary, for the latter this was not taken into perspective in the current phase as it is not widely available and latitude-longitude descriptions are more general.

In our software, libpointmatcher's[6] implementation of the ICP algorithm was used. To configure it, the guides available on the official documentation's website at `https://libpointmatcher.readthedocs.io/en/latest/ICPWithoutYaml/` were followed.

[5] `https://eigen.tuxfamily.org`
[6] `https://libpointmatcher.readthedocs.io/en/latest/`

The accuracy metric is fixed 70 value at the moment, though this can be refined in the future.

4.7 Processing IMU data

This section elaborates on how data gets transported from the individual sensor registers to the processing pipeline and how it gets transformed on the way.

Reading and storing sensor data

When the measurement is carried out, a separate measuring program is launched and stopped at the end of the actual test. The program creates a text file with a header containing the following metadata:

- Time offset compared to LiDAR data (*ms*): measurements before this timestamp are going to be used to initialize the filter.
- Sample period length (*ms*): the time amound between each measurement.
- Gyroscope sensitivity ($mdps/LSB$)
- Accelerometer sensitivity (mg/LSB)
- Magnetometer sensitivity ($\mu T/LSB$)

Below the header is the actual measurement data. Each line contains the following readings:

- Elapsed time since last measurement (*ms*)
- Gyroscope X, Y and Z measurement ($mdps$)
- Accelerometer X, Y and Z measurement (mg)
- Magnetometer X, Y and Z measurement ($gauss$)

This file is later read into memory by our LiDAR processor program which performs the magnetometer calibrations and feeds the data into the pipeline through an AHRS filter.

The IMUProcessor

In this project, we decided to utilize the *Fusion filter library* [35] developed by x-io Technologies. The underlying AHRS algorithm is fusing the accelerometer, gyroscope and magnetometer measurements into Euler angles (pitch, roll, yaw) or a quaternion while updating its inner state and correcting for sensor errors and interferences. The filter algorithm's method of calculation is based on the PhD thesis [36] of Madgwick, which is an improved version of the original *Madgwick algorithm* [37].

To make it possible to experiment with multiple filter implementations in the future, we specified an interface through which our pipeline interacts with the algorithm. The UML class diagram extract of this can be seen in Figure 4.7.

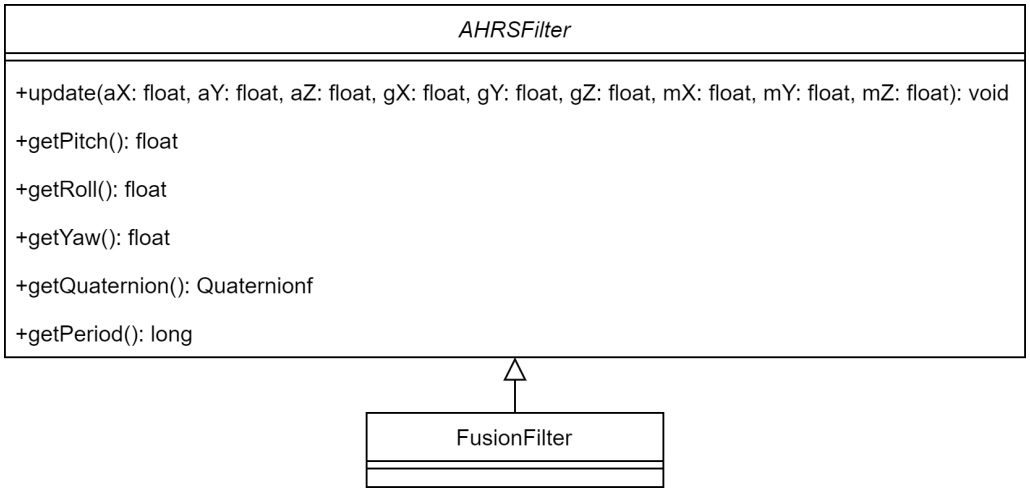

Figure 4.7: UML Class diagram extract of AHRSFilter

As the diagram shows, our requirements for an AHRS Filter is for it to have an update function, which updates the inner state by having sensor readings as parameters and to be able to present its current state as Euler angles and as a quaternion as well.

The `FusionFilter` realises this interface with the help of the Fusion AHRS library and is encapsulated by the `IMUProcessor`. The `IMUProcessor` either serves the estimated orientation as `Calculator` to the `CloudTransformer` or as a `Processor`, it calculates and executes the rotation as well to serve a pre-processed cloud to the other calculators. During our measurements we used the IMU data source as a processor, although future developments might include using its estimations as a calculator. A simplified UML class diagram extract of the mentioned area can be seen in Figure 4.8.

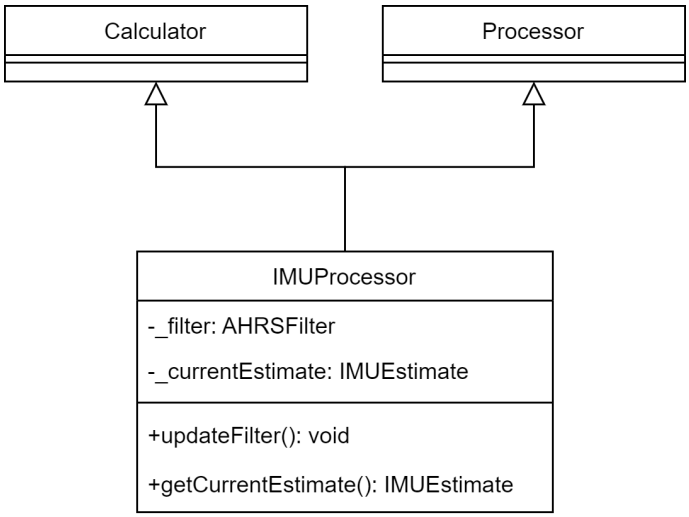

Figure 4.8: UML Class diagram extract of the `IMUProcessor`

Chapter 5

Results

We tested our hardware and software architecture with different sensor and environment combinations to see which sensor or algorithm compositions can effectively enhance each other.

5.1 Recording of measurements

The recording process did not come without its own set of difficulties. As for most of the recordings, the sensor configuration was carried around on a cart with hard wheels, there was no dampening, resulting in observable distortion in the recorded data. Moving objects, like other cars and pedestrians also made the task of building a static map harder by introducing information only present in a few frames. Having to open doors on a corridor posed a similar issue as well.

On the results shown in the upcoming sections, concentric circles can be seen in many places of the ground. These are points the LiDAR recorded around itself along its path. For the visual interpretation of trajectories on the images, it is useful to know that the estimated poses are shown as spheres, with lines connecting them, although the former might not be visible when zoomed out.

For the shown ground truth satellite imagery, Google Maps and Google Earth was used, available at `https://www.google.com/maps` and `https://earth.google.com/web/` respectively.

5.2 Mapping with only GPS

While our main focus was more about using more methods together, we did tests with LiDAR-GPS setups as well. This did produce fairly good results, although was much reliant on having good signal availability.

5.2.1 Offline GPS

The transformation itself is calculated the same way for both online and offline GPS, but other factors, like having all data available for pre-processing with a kalman filter make grounds comparison of the two.

Under almost ideal circumstances, being outdoors with no overshadowing high-rise buildings nearby, the mapping was quite accurate. Such a map can be seen on Figure 5.1 with ground truth being shown on Figure 5.2.

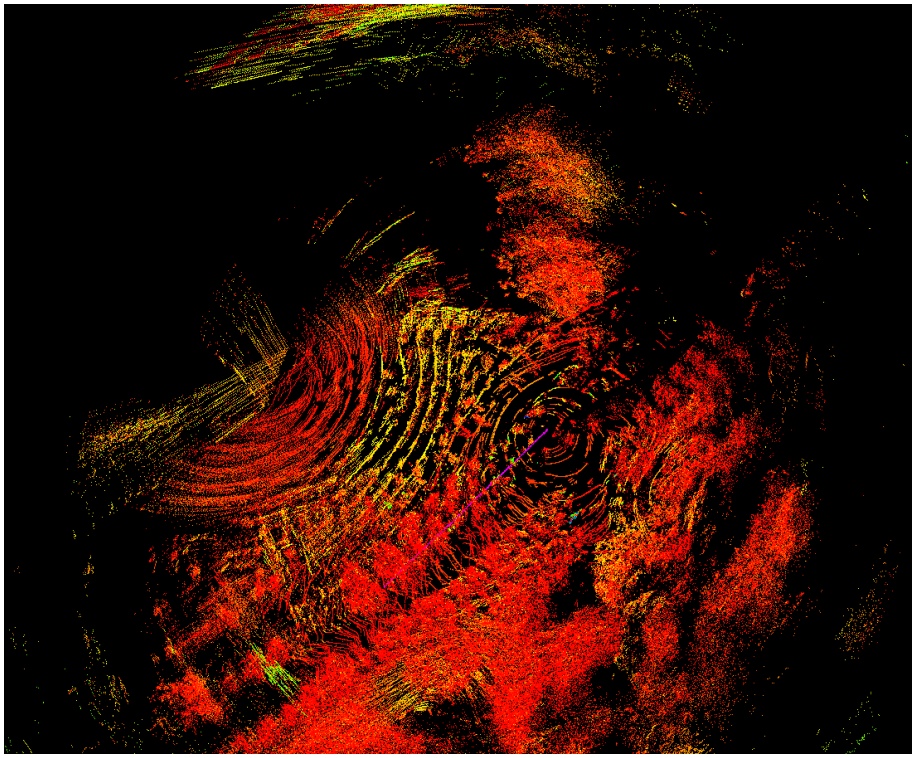

Figure 5.1: The final mapping under close to ideal circumstances

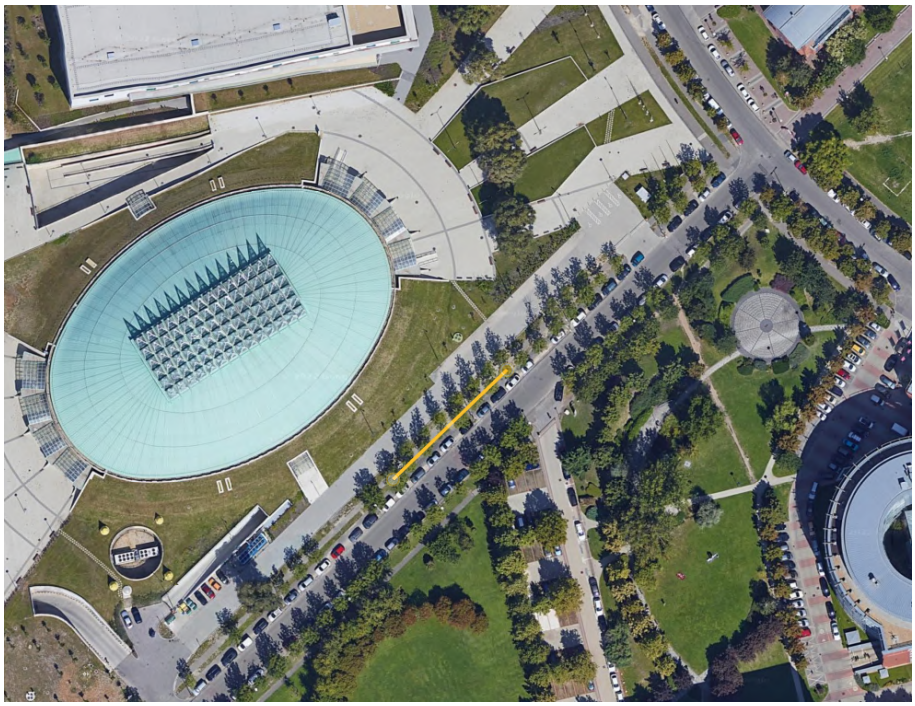

Figure 5.2: The ground truth

Although there is some distortion, and the large amount of greenery add noise to the image, the major shapes, like the bottom wall of the top building, or the concentric ellipses around the arena due to the slope of its sides are quite distinctive.

For the next example, the sensors were pushed through the inside of a mall for a longer section, during which no signal was available, up until exiting on the other side. Although this is more meant for a combined GPS - ICP approach, the performance of the Kalman filter can be interesting nevertheless. The attempted mapping can be seen on Figure 5.3, with the ground truth being shown on Figure 5.4.

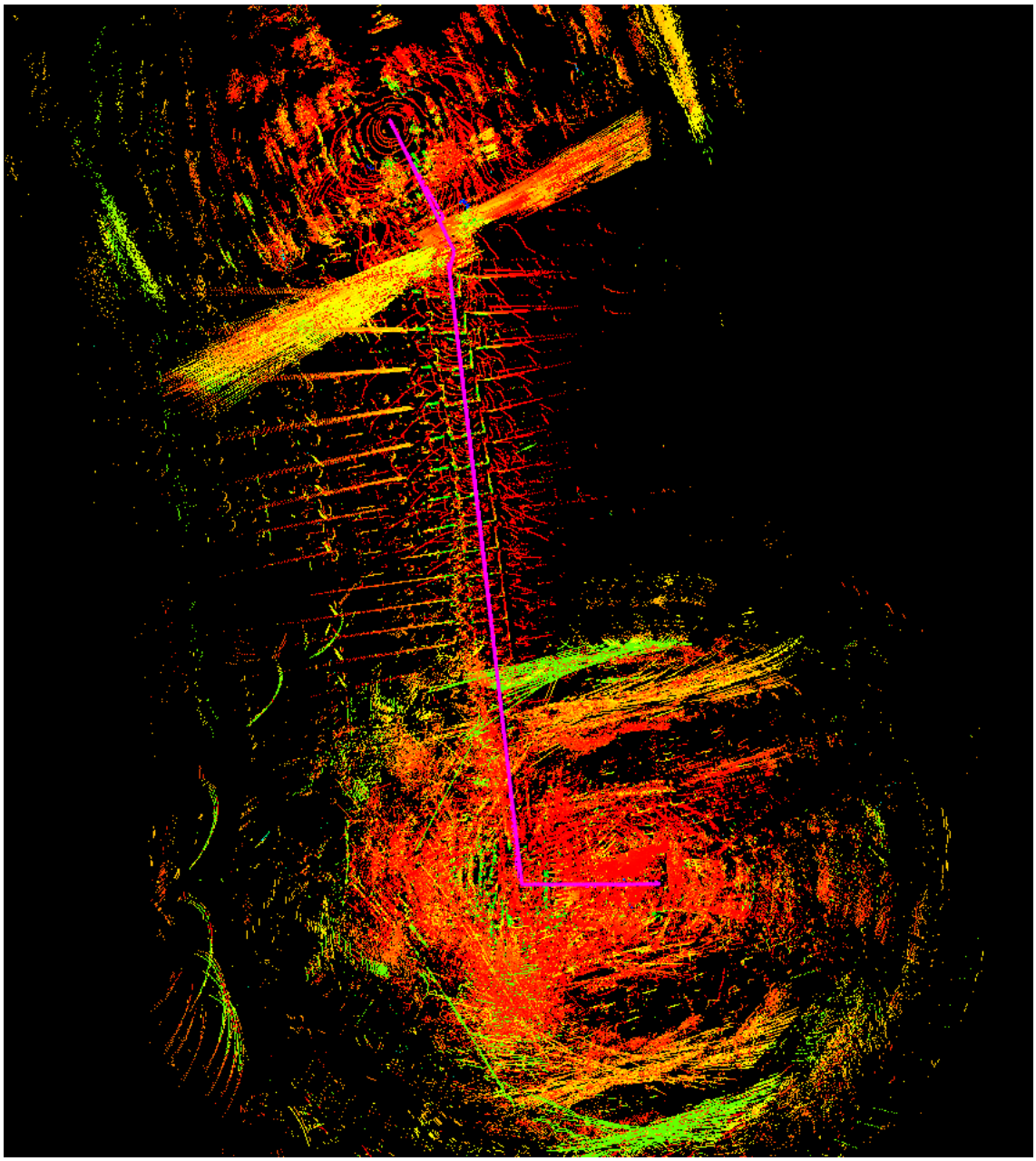

Figure 5.3: The GPS only mapping when going through a building

Figure 5.4: The GPS only mapping when going through a building

Here the algorithm performed about as well as it can be expected. The Kalman filter did a good job of connecting the last point before losing signal to the first one after regaining it, but the inside part contained a turn the filter had no chance of resolving.

5.2.2 Online GPS

For this configuration we mostly did measurements for fusion with other approaches, and as such the constantly changing signal quality lead to results similar to those in the previous example. Also as this approach does not encompass a Kalman filter, just polynomial extrapolation for missing data points, the results had more distortion when the measurements had inaccuracy to begin with.

A measurement which showcases many aspects of our general results was taken attached to the top of a car. First, the vehicle leaves the the parking lot it started in, then travels through a number of roads, waits for a traffic light, then does an uninterrupted section on a straight, long road.

The measurement had some inaccuracies, leading to distortion in the mapping and the trajectory. In the offline case, this might be corrected by the Kalman filter, but in this case, as it can be seen in Figure 5.5 it did show up.

Figure 5.5: Inaccuracy in the measurement seen in the trajectory

Travelling on roads comes with traffic lights and other cases when the vehicle must temporarily stop. As the GPS was not accurate enough to send the exact same position coordinates and the online processing does not incorporate data correction in the current stage, the in-place position was not recognized properly, and end up being multiple positions with slight distances between them as shown by the trajectory in Figure 5.6.

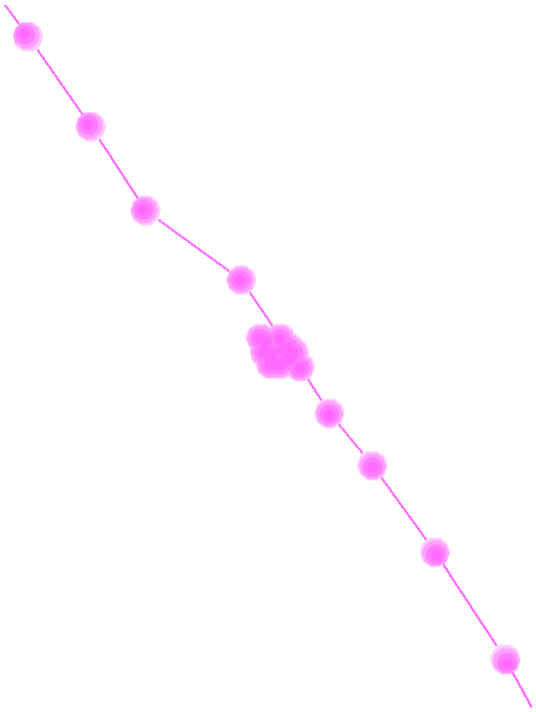

Figure 5.6: The predicted set of poses for the in-place ground truth

That being said, when the vehicle was traversing a long, straight road, with good signal availability, the method did produce good results, an example of which can be seen on Figure 5.7, with the ground truth show on Figure 5.8.

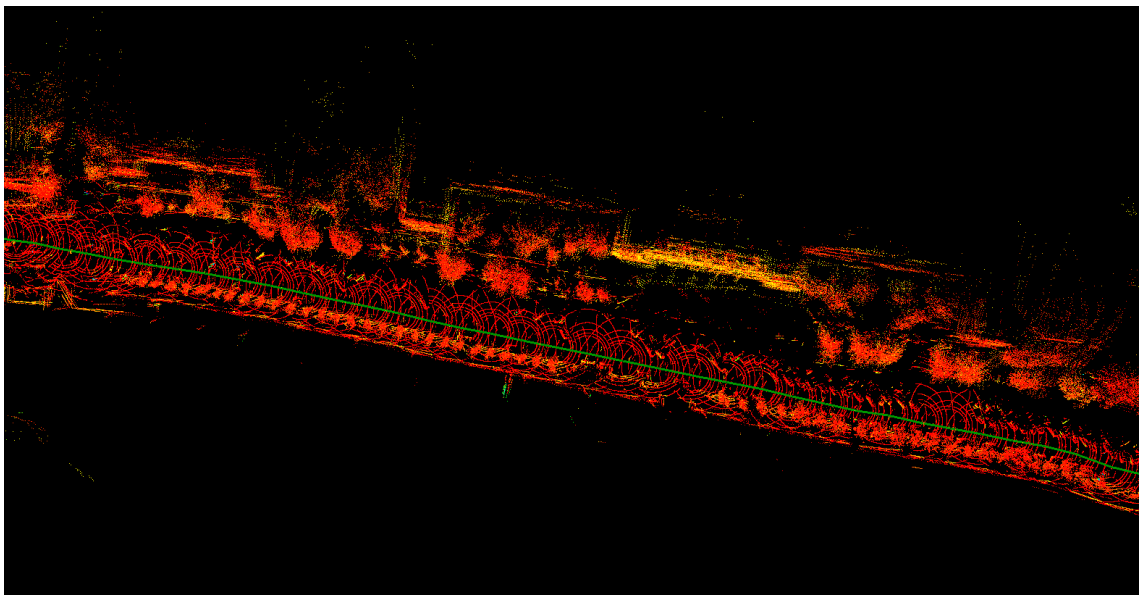

Figure 5.7: Mapping created by the car travelling on a long, straight road

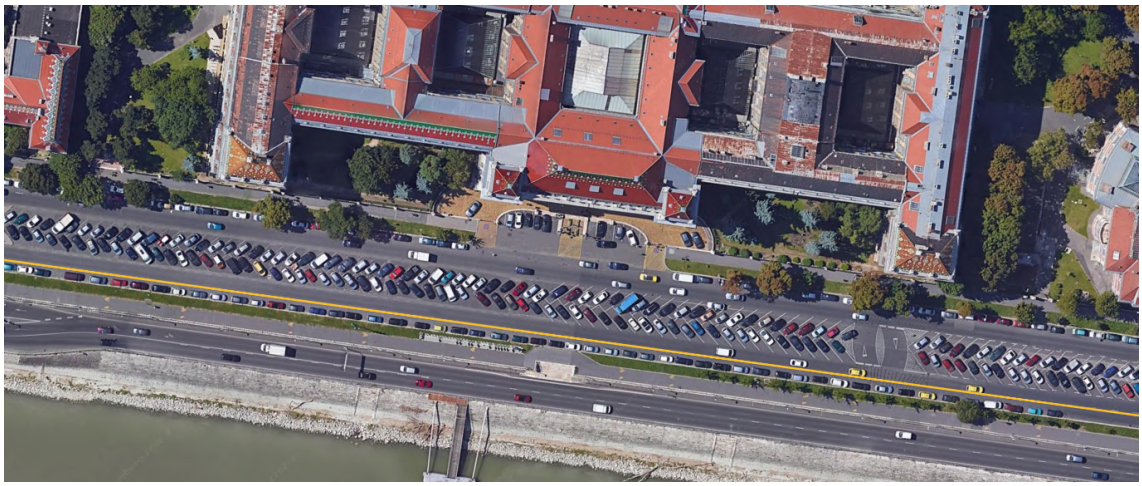

Figure 5.8: Ground truth along the road

5.3 Mapping with only ICP

Relying just on ICP for mapping had its issues as well. As mentioned before in chapter 2 and chapter 3, in order to produce good results the method largely relies on having a decent input. Sharp turns in the path pose another issue, due to the sparsity of frames during taking the turn.

One measurement we took, was taking a rectangular path with a handcar in the inside of a building. While the algorithm successfully mapped through the first turn, for the second and third turns the angle was not successfully estimated, resulting in a distorted path, as well as 3 rotated overlapping maps of the area. The successful rotation on the the first turn can be attributed to the turn being taken slower, which enabled the LiDAR to have more frames of the turn itself. The results of this mapping can be seen on Figure 5.9

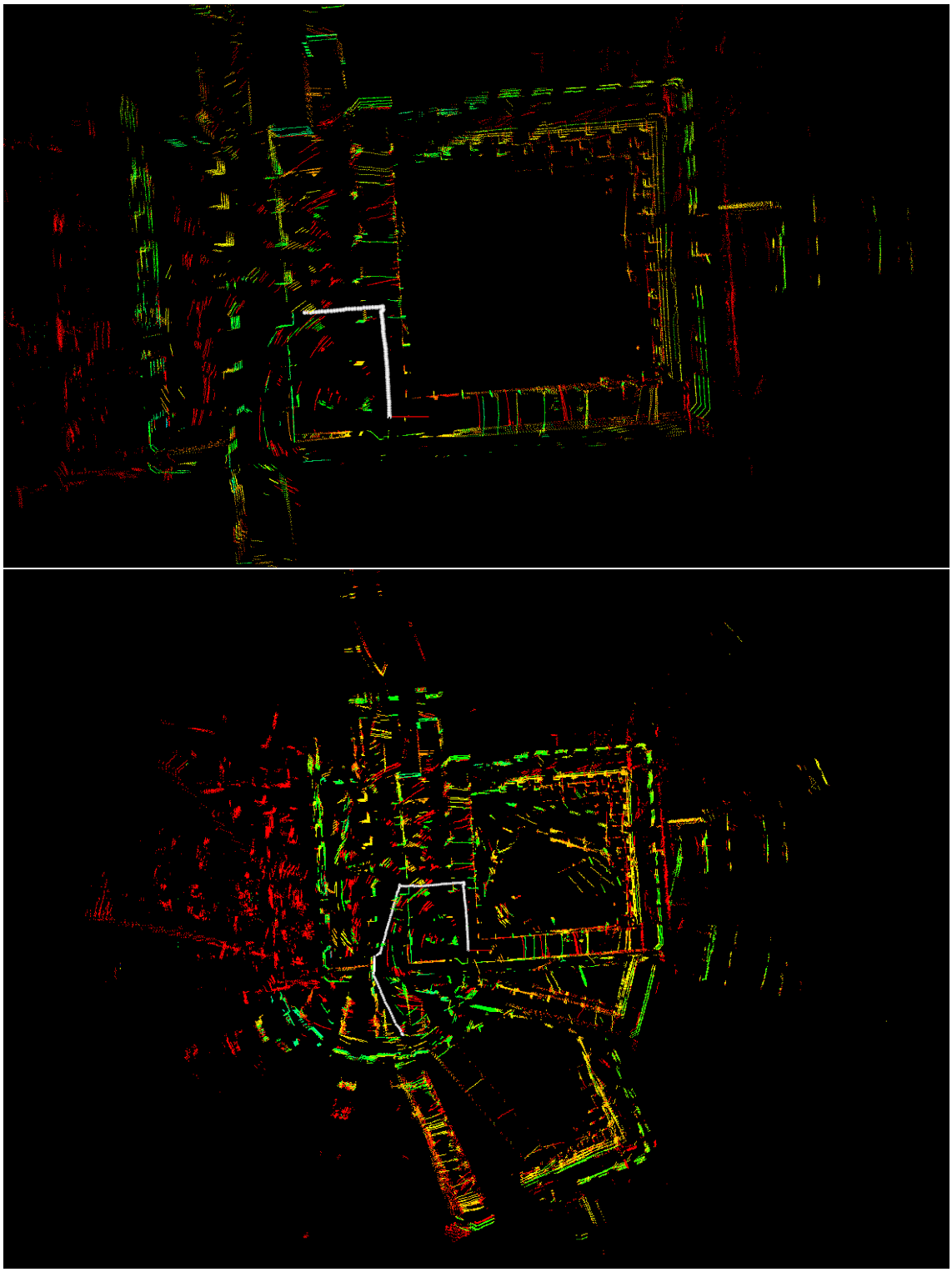

Figure 5.9: Mapping on a rectangular path before(top) and after(bottom) the second turn

Another measurement which showcases the first issue, namely not having good enough input was made by having the LiDAR on top of a car coming out of an underground garage. As the vehicle moves at a higher speed, the translation between the frames is greater too, making ICP not being able to find the proper match and settle for a

local minimum. As it can be seen on Figure 5.10, this results in the walls perpendicular to the direction of motion getting mismatched.

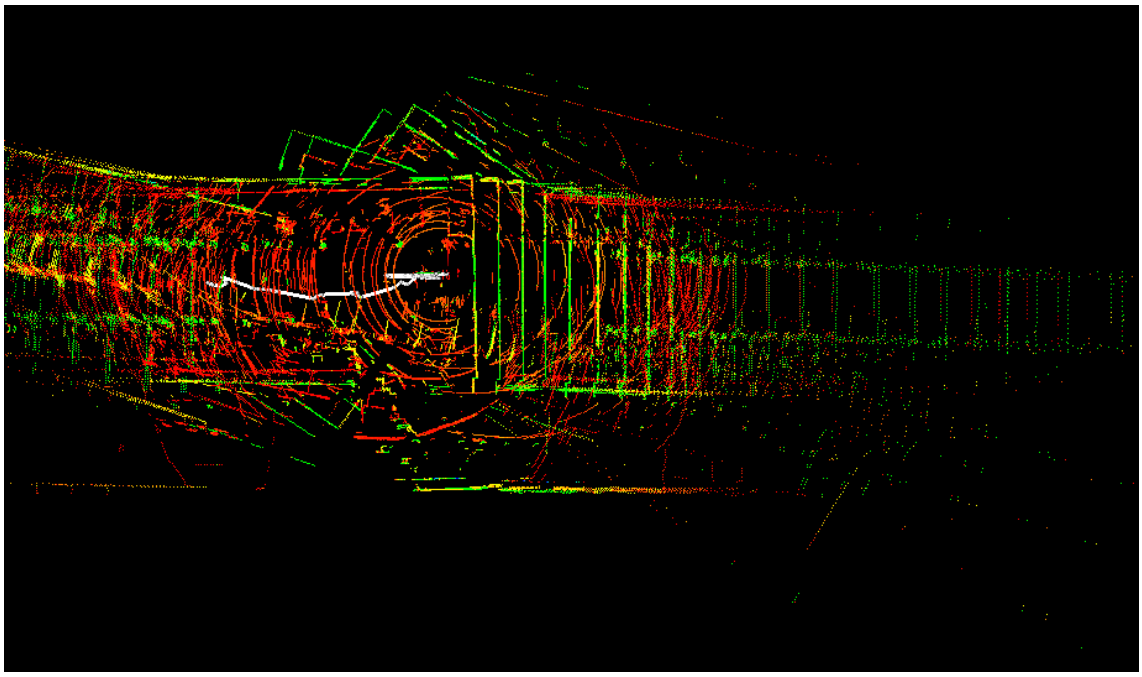

Figure 5.10: Missed matching on the walls on the right due to higher speeds

5.4 Evaluation of single method approach

As described in the previous sets of results, using a single approach for positioning – while capable of producing good results under the right circumstances –, is ultimately too dependent on the conditions. By pinpointing the difficulties of the individual methods, a complementing solution can be added to cover for the problematic cases.

- For the GPS the main issue is the poor quality or total lack of signal. While this can somewhat be remedied by using a Kalman filter, for indoors use the better solution is combining it with another positioning method which can function indoors too. For this, ICP based positioning was our choice.
- For the ICP method the issue is the data sparsity of the LiDAR leading to difficulties matching frames with larger rotations (taking a turn) or translations (higher speed) between them. This is a well known issue leading to solutions like LOAM. In our application we use the IMU data to help with the initial odometry of the frame.

5.5 Mapping with GPS and ICP SLAM combined

The combination of the GPS and ICP based positioning mainly provided a fallback for less accurate GPS measurements in the form of ICP based mapping, as the former is usually superior in accuracy.

A good demonstration of this is the example showed earlier in section 5.2. Inside the mall, due to the low quality signal the software switches to using ICP, which does a decent job of mapping the interior of the building, which can be seen on Figure 5.11.

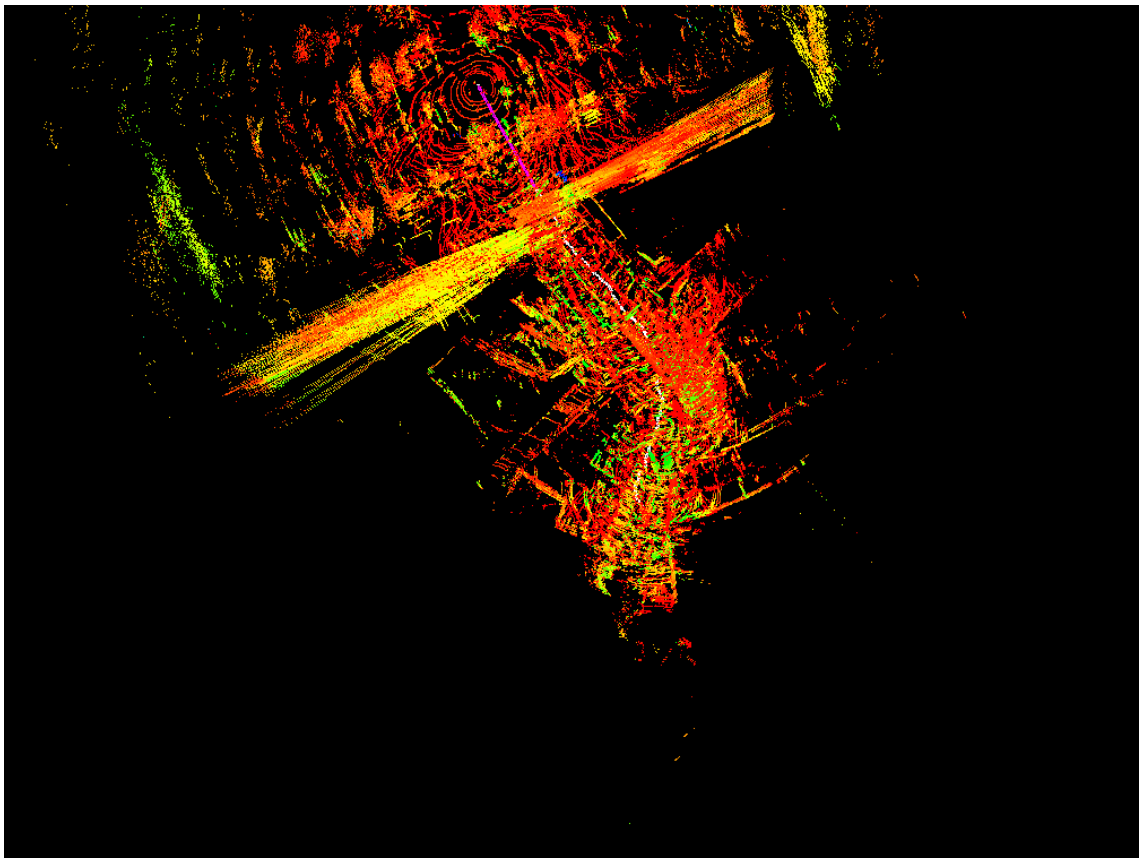

Figure 5.11: The interior of the mall as mapped by the switching to ICP

That being said, the image itself is quite noisy, due to the complex interior of the building with features like crossing corridors, multi-level structure and overhead walkways. The predicted trajectory is visible on Figure 5.12, with the ground truth already shown on Figure 5.4.

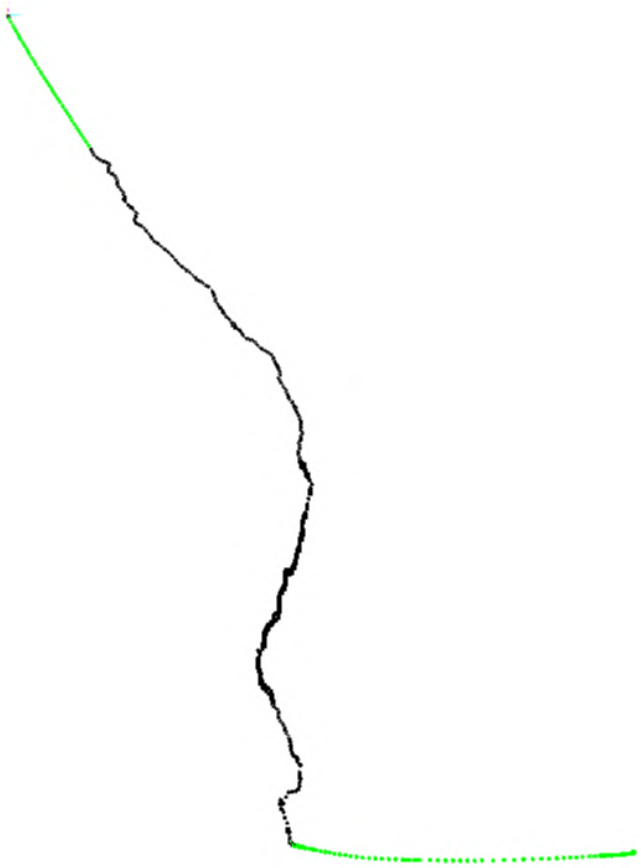

Figure 5.12: The trajectory of the sensor as predicted by GPS (green) and ICP (black)

Other issues of GPS based positioning explained in subsection 5.2.2 could suggest the use of a LOAM framework with GPS odometry and ICP mapping, but in our research this has not been explored yet.

5.6 Mapping with IMU and ICP SLAM combined

Complementing the calculations with IMU data has proven to be quite effective, especially around corners, where it is difficult for the ICP algorithm to determine the amount of the rotation due to the low number of frames.

5.6.1 Indoor corridor

An area where the assistance of the IMU sensor can be examined well is an indoor corridor, where due to the lack of characteristics and sharp turns, the ICP algorithm's

ability to accurately track movement is significantly impaired. During this test, our cloud merging algorithm was seriously affected as well, therefore the resulting point does not clearly show the architecture of the building. The odometry seen in the following figures however, is rather promising.

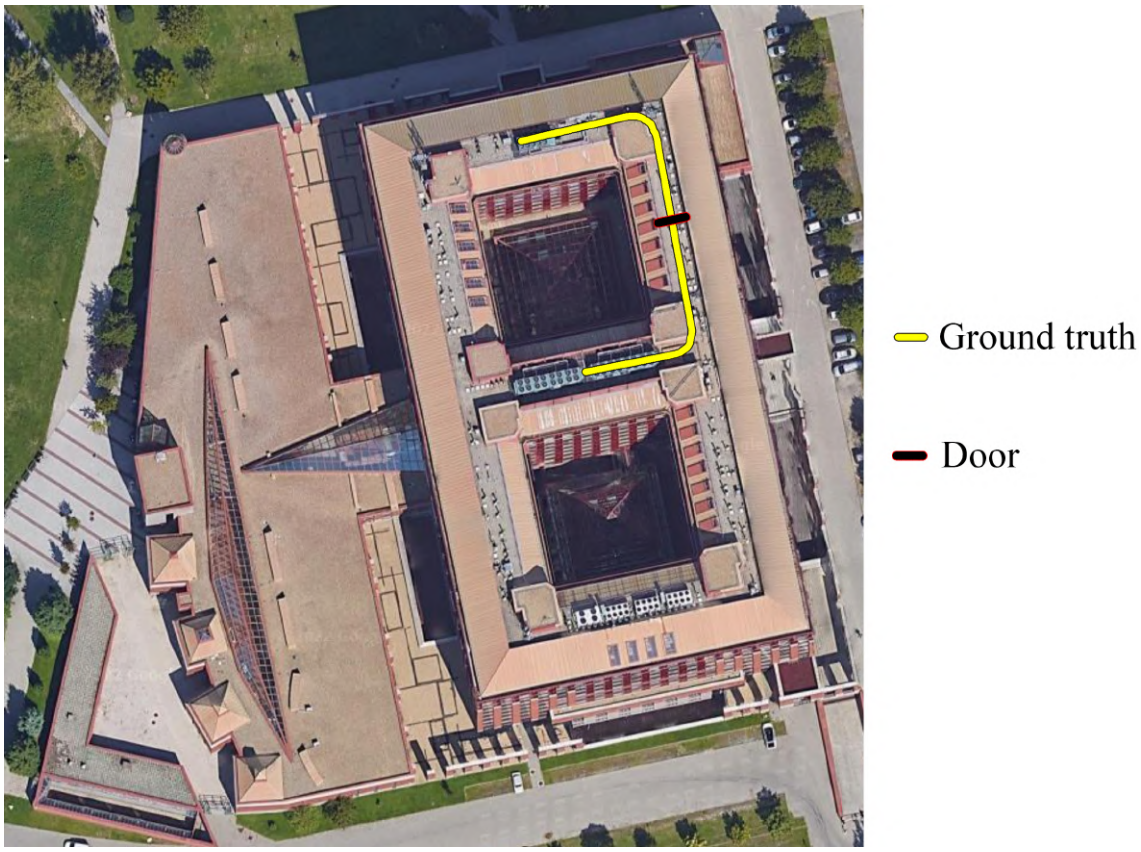

Figure 5.13: Path of the indoor corridor measurement

Figure 5.13 shows the path we scanned during the experiment. We marked the location of the glass door on the path because glass surfaces can introduce artifacts in LiDAR measurements which can affect the SLAM algorithm, and the carrying vehicle had to be stopped for a brief time as well. A slight path error can be seen because of this in both measurements in Figure 5.14.

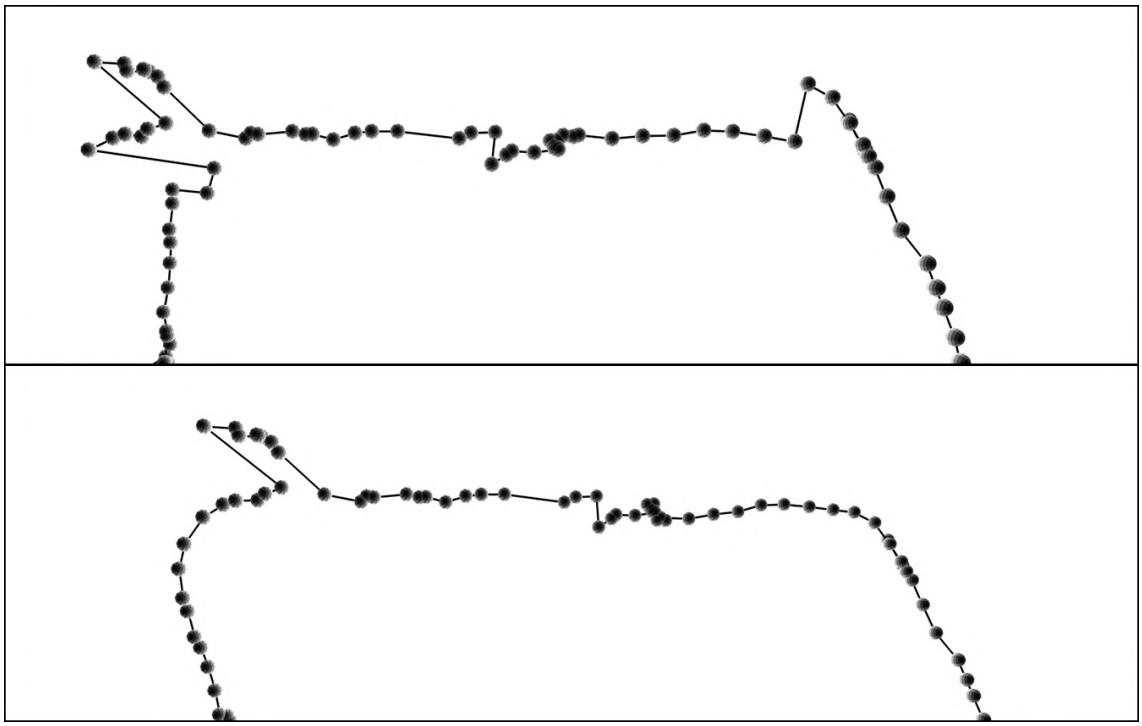

Figure 5.14: Comparison of the paths calculated via ICP (top) and IMU-assisted ICP (bottom)

As Figure 5.14 shows, improvements have been achieved around the corners, where the ICP algorithm introduced errors. One of the errors at the left corner was not corrected even with IMU assistance, but such anomaly can be later filtered by an algorithm such as a Kalman filter.

5.6.2 Indoor and outdoor combined

To demonstrate the effects of IMU assistance on a larger scale, we arranged a combined measurement in which the vehicle departs in an outside environment containing vegetation, a building wall and occasionally walking people. The vehicle then enters the building on the left side and travels the shortest path to the exit on top. This test includes people moving inside the building as well as corridors similar to the indoor measurement. The path traveled can be seen in Figure 5.15.

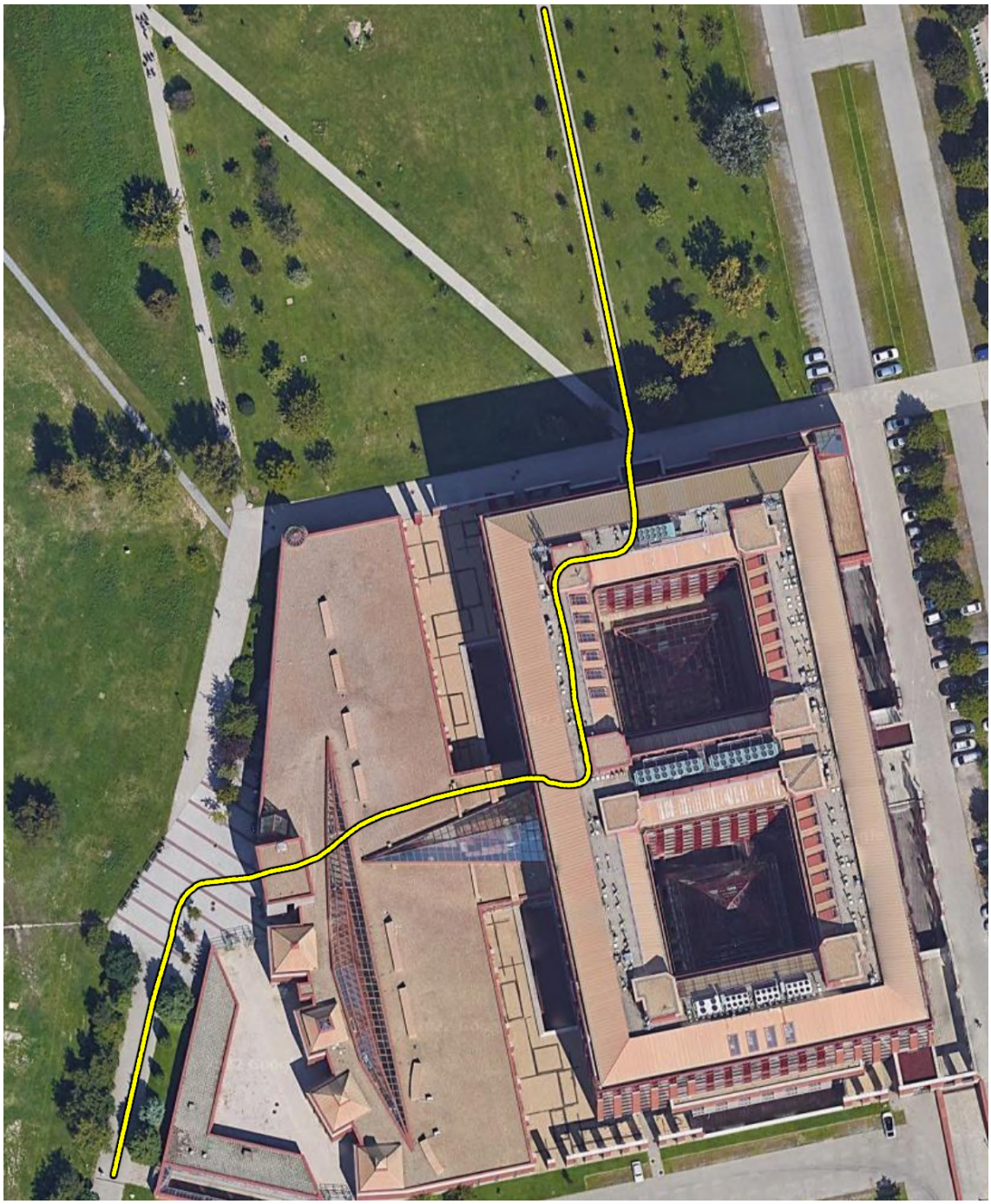

Figure 5.15: Path of the combined measurement

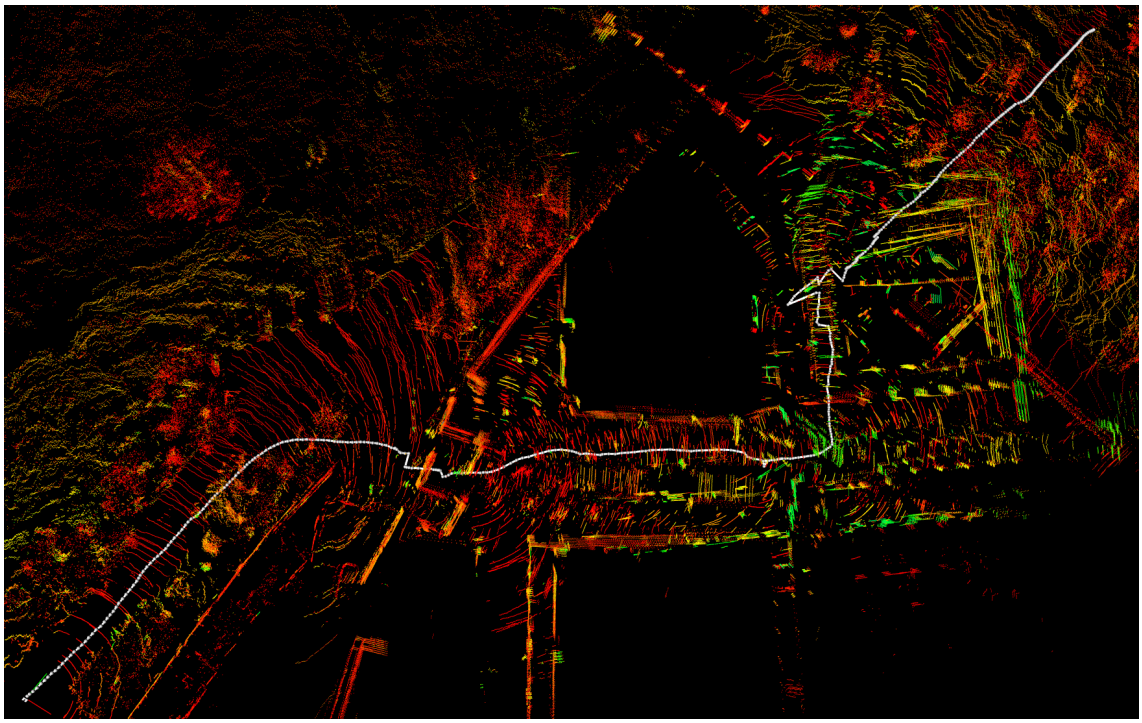

Figure 5.16: Calculated path (white) by ICP algorithm without IMU assistance

As seen in Figure 5.16 the ICP SLAM algorithm did fairly well during the first two thirds of the path. Around the last two corners however there are similar corridors to the first experiment and the algorithm could not correctly map the exit section from the building.

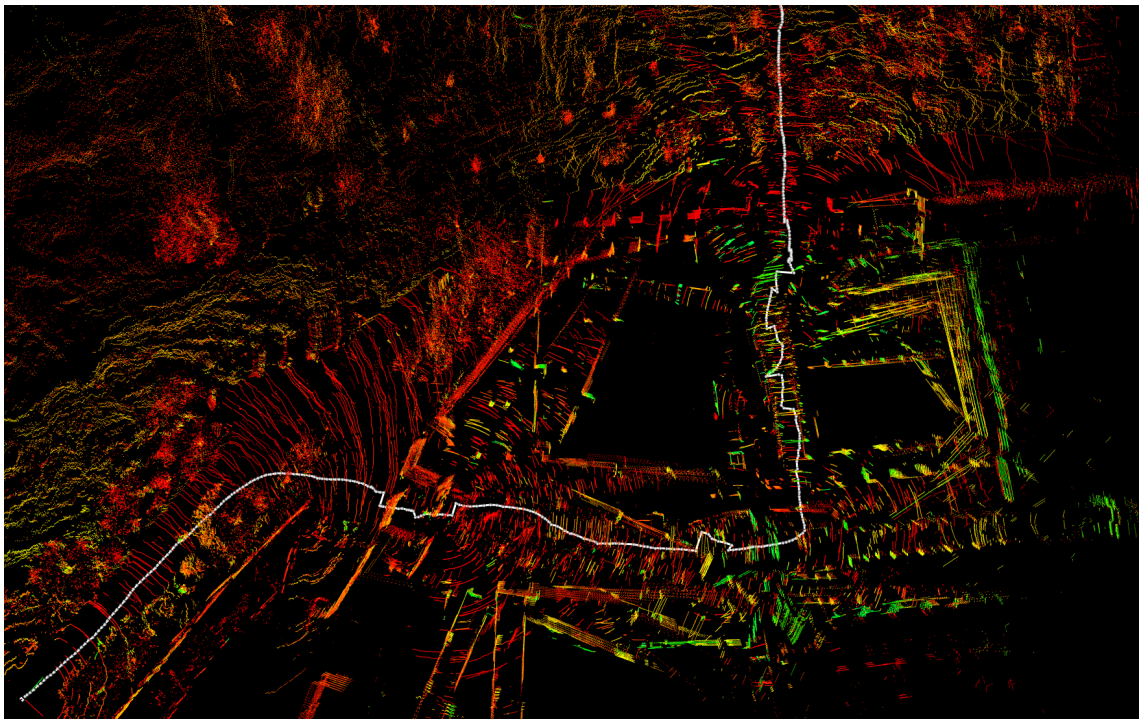

Figure 5.17: Calculated path (white) by ICP algorithm with IMU assistance

Figure 5.17 shows how the cloud pre-rotation of the IMU processor prevented the ICP algorithm from calculating a faulty final direction. Corrections can still be applied to the shown result, the overall directions however are preserved, which is a good starting point for further improvements.

The final mapping after processing this data set is rather promising as well considering our LiDAR sensor has 16 channels. Figure 5.18 shows the 3D model of the captured are modelled by the Google Earth application. Figure 5.19 shows the final point cloud from a similar angle.

Figure 5.18: ELTE Southern block modeled by Google Earth

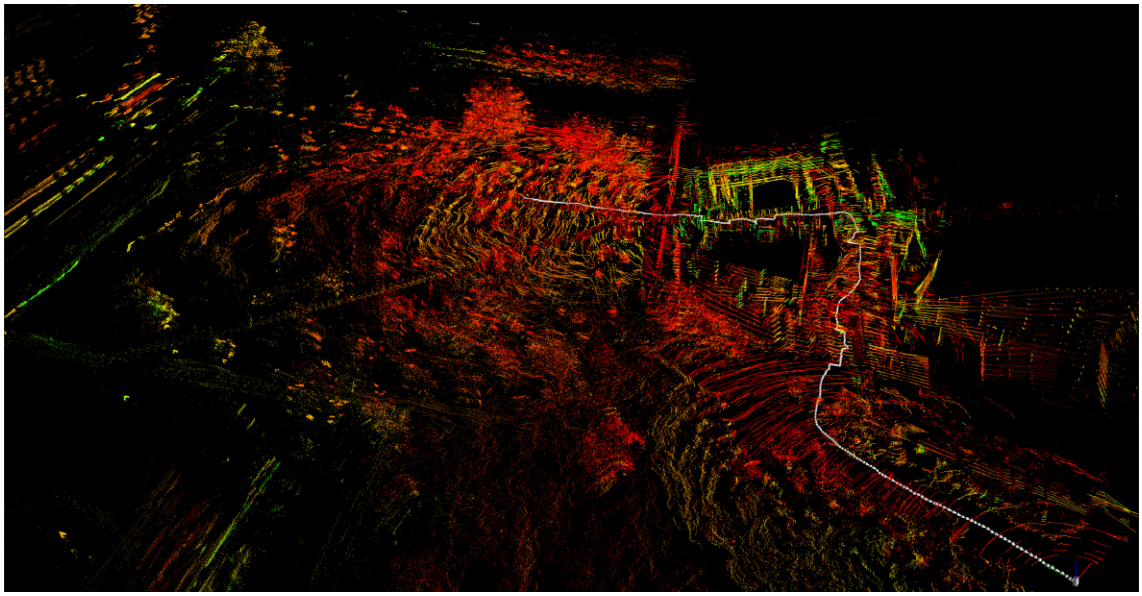

Figure 5.19: ELTE Southern block modeled by our LiDAR pipeline

As the previous results show, rotating the point cloud according to IMU data as a pipeline step before determining translation via a SLAM algorithm is a promising method of fusing these two data sources together.

5.7 Evaluation of combined approach

As shown previously, fusing the output of single methods indeed improves the versatility of the mapping process.

- Combining GPS with ICP SLAM helps with solving the problem of the poor or non-existent GPS signal. As soon as the GPS accuracy metric signifies a signal loss, the ICP SLAM can take over and continue mapping the area.
- The fusion of IMU data and ICP SLAM produces significantly better results in narrow corridors and sharp turns, where the SLAM algorithm loses track of the trajectory either because the lack of features in the environment or drastic changes in the orientation.

These improvements turned previously unusable results into recognizable reproductions of the scanned areas, leading us to the conclusion, that these sources of information can, and should be utilized together.

Chapter 6

Conclusion

The main goal of our work was to evaluate and compare different LiDAR mapping methods and their combinations, ranging from external odometry devices, like GNSS sensors or an IMU, to purely mathematical SLAM approaches. For the positioning methods, online and offline GPS with slightly different workflow, the ICP algorithm, and ICP with IMU assisted odometry were implemented. The issue of unifying them in a common transformation description system was solved. For online GPS processing the difficulties of data synchronization and timing, which come naturally with the asynchronous workflow were solved.

While under the proper circumstances just GPS or ICP could produce good results, combinations of them or enhancement via other odometry like IMU for ICP did improve results. Upon inspection of the taken measurements often further problems were discovered, like the need of proper calibration for the IMU sensor, or the mechanical spinning component of the LiDAR causing interference in the IMU's magnetometer.

We've also made efforts towards real-time mapping with the introduction of online GPS parsing, as well as using a Rasberry Pi for data recording, however, turning our model into a real-time solution will be the objective of a future research.

Our research was implemented as the extension of the foundations established by Máté Cserép and Roxána Provender, who built a highly customizable pipeline infrastructure for future development [22].

6.1 Future work

While we did get promising results from our current integration of the used methods, they were still somewhat rudimentary in their implementation at some parts. An example of this is the use of a quite basic SLAM approach in ICP, instead of something more state-of-the-art, like a LOAM framework.

We successfully introduced additional sensor data into the system in the form of an IMU, but further usage of the information from this could be explored.

The vertical combination of the used methods also has further ways to go, experimenting with systems like GPS odometry – ICP mapping, with even possibly having IMU aid could yield promising results.

From a more software-oriented viewpoint, we have accumulated a number of data sources by now, each requiring different capture software, as such developments towards separated capture and processing binaries could improve ease of use. For better real-time performance, assigning the independent transformation tasks to separate processing cores could result in better runtime. Another approach for this could also be making use of the separated capture-processing model and making the latter into a cloud application to remotely handle the computationally expensive parts.

Acknowledgements

The thesis project and research was completed in the GIS Laboratory[1] of the Eötvös Loránd University. The project was supported through a student research scholarship program of the Faculty of Informatics.

We would like to express our gratitude to our supervisor, Máté Cserép, for his guidance through the stages of the research work. We are grateful to Bandó Kovács, institute engineer at the Faculty, for the designing and 3D printing of custom parts, as well as for technical advice.

We would also like to express our appreciation to former master students in the GIS Laboratory for their prior work providing preliminary results and an established software framework for our thesis to build on. Thanks to Roxána Provender for the original, file-based GPS positioning, and to Levente Kiss for implementing the ICP based positioning, as well as his theoretical overview of the method.

[1] `https://gis.inf.elte.hu/`

Bibliograhpy

[1] Teledyne Digital Imaging, Inc. *Teledyne CL-360*. [Online; accessed April 24, 2022]. URL: https://www.teledyneoptech.com/en/products/compact-lidar/cl-360/.

[2] Colorado State University. *GPS principles*. [Online; accessed April 24, 2022]. 2020. URL: https://trakkitgps.com/how-gps-works/.

[3] Nanjing Sky MEMS Technology Co.,Ltd. *SkyMEMS 6 Dof IMU Sensor*. [Online; accessed April 24, 2022]. URL: http://www.inssensor.com/imu/6-dof-imu-sensor.html.

[4] Feng Lu and Evangelos Milios. "Globally consistent range scan alignment for environment mapping". In: *Autonomous robots* 4.4 (1997), pp. 333–349.

[5] Mahalanobis Prasanta Chandra et al. "On the generalised distance in statistics". In: *Proceedings of the National Institute of Sciences of India*. Vol. 2. 1. 1936, pp. 49–55.

[6] P.J. Besl and Neil D. McKay. "A method for registration of 3-D shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (1992), pp. 239–256. DOI: 10.1109/34.121791.

[7] Victor Lamoine. *Interactive ICP*. [Online; accessed May 2, 2022]. 2018. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/interactive_icp.html.

[8] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time." In: *Robotics: Science and Systems*. Vol. 2. 9. Berkeley, CA. 2014, pp. 1–9.

[9] Han Wang et al. "F-loam: Fast lidar odometry and mapping". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 4390–4396.

[10] Zheng Liu and Fu Zhang. “Balm: Bundle adjustment for lidar mapping”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3184–3191.

[11] Jianhao Jiao et al. “Robust odometry and mapping for multi-lidar systems with online extrinsic calibration”. In: *IEEE Transactions on Robotics* (2021).

[12] Luca Caltagirone et al. “LIDAR–camera fusion for road detection using fully convolutional neural networks”. In: *Robotics and Autonomous Systems* 111 (2019), pp. 125–131.

[13] Le Chang, Xiaoji Niu, and Tianyi Liu. “GNSS/IMU/ODO/LiDAR-SLAM integrated navigation system using IMU/ODO pre-integration”. In: *Sensors* 20.17 (2020), p. 4702.

[14] Xingxing Zuo et al. “Lic-fusion: Lidar-inertial-camera odometry”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 5848–5854.

[15] Hanzhang Xue, Hao Fu, and Bin Dai. “IMU-aided high-frequency lidar odometry for autonomous driving”. In: *Applied Sciences* 9.7 (2019), p. 1506.

[16] S Karam, V Lehtola, and G Vosselman. “Strategies to integrate IMU and LiDAR SLAM for indoor mapping”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 1 (2020), pp. 223–230.

[17] Raspberry Pi Ltd. *Raspberry Pi 4 model B*. [Online; accessed April 28, 2022]. 2018. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[18] Velodyne Lidar, Inc. *Velodyne VLP-16 LiDAR sensor*. [Online; accessed April 30, 2022]. 2019. URL: https://velodynelidar.com/products/puck/.

[19] MediaTek Inc. *MediaTek MT3339 all-in-one GPS datasheet*. Tech. rep. version 1.0. [Online; accessed May 9, 2022]. Jan. 2017. URL: https://cdn.compacttool.ru/downloads/MT3339_Mediatek_Datasheet.pdf.

[20] OzzMaker. *BerryGPS-IMU V4*. [Online; accessed April 27, 2022]. URL: https://ozzmaker.com/product/berrygps-imu/.

[21] Subirana J. Sanz, Zornoza JM. Juan, and Hernandez-Pajares M. *GNSS signal*. [Online; accessed May 8, 2022]. 2011. URL: https://gssc.esa.int/navipedia/index.php/GNSS_signal.

[22] Roxána Provender. “Spatial localization of LiDAR point clouds by sensor fusion”. https://edit.elte.hu/xmlui/handle/10831/56231. MSc thesis. ELTE, Faculty of Informatics, 2020.

[23] Richard B Langley et al. “Dilution of precision”. In: *GPS world* 10.5 (1999), pp. 52–59.

[24] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960).

[25] C Carl Robusto. “The cosine-haversine formula”. In: *The American Mathematical Monthly* 64.1 (1957), pp. 38–40.

[26] Ellon Mendes, Pierrick Koch, and Simon Lacroix. “ICP-based pose-graph SLAM”. In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2016, pp. 195–200.

[27] Kok-Lim Low. “Linear least-squares optimization for point-to-plane icp surface registration”. In: *Chapel Hill, University of North Carolina* 4.10 (2004), pp. 1–3.

[28] Kris Winer. *Simple and Effective Magnetometer Calibration*. [Online; accessed May 4, 2022]. 2017. URL: https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration.

[29] Andre Amador and Miguel Canals. “Probing the Hydrodynamics of Plunging Gravity Waves Through Lagrangian Observations of Inertial Particle Dynamics”. PhD thesis. July 2013, p. 73. DOI: 10.13140/RG.2.2.14948.30089.

[30] Velodyne Lidar, Inc. *VLP-16 User Manual*. Tech. rep. [Online; accessed May 4, 2022]. 2019. URL: https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf.

[31] National Marine Electronics Association. *NMEA 0183 - Standard For Interfacing Marine Electronic Devices*. Tech. rep. [Online; accessed May 10, 2022]. 2002. URL: https://www.plaisance-pratique.com/IMG/pdf/NMEA0183-2.pdf.

[32] A.H. Watt. *3D Computer Graphics*. 3D COMPUTER GRAPHICS. Addison-Wesley, 2000. ISBN: 9780201398557. URL: https://books.google.hu/books?id=xOYBkgAACAAJ.

[33] Hanan Samet. *The design and analysis of spatial data structures*. Vol. 85. Addison-Wesley Reading, MA, 1990.

[34] *EOV coordinate system*. [Online; accessed May 8, 2022]. URL: http://lazarus.elte.hu/gb/geodez/geod2.htm.

[35] x-io Technologies. *Fusion AHRS library*. [Online; accessed May 4, 2022]. 2021. URL: https://github.com/xioTechnologies/Fusion.

[36] Sebastian O. H. Madgwick. "AHRS algorithms and calibration solutions to facilitate new applications using low-cost MEMS". In: University of Bristol, EThOS. 2014. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.681552.

[37] Sebastian O. H. Madgwick. "An efficient orientation filter for inertial and inertial/magnetic sensor arrays". In: [Online; accessed May 4, 2022]. 2010. URL: https://courses.cs.washington.edu/courses/cse474/17wi/labs/l4/madgwick_internal_report.pdf.

List of Figures