Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institut für Informatik
Abteilung VI
Autonome Intelligente Systeme

Rheinische
Friedrich-Wilhelms-Universität Bonn

Bachelorarbeit

Improving Pose Graph Consistency using LiDAR Map Registration against Georeferenced Models

Autor:
Linus T. Mallwitz

Erstgutachter:
Prof. Dr. Sven Behnke

Zweitgutachter:
Dr. Jens Behley

Betreuer:
Jan Quenzel
Benedikt T. Imbusch

Datum: May 12, 2023

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen verwendet habe. Die Stellen der Arbeit sowie evtl. beigefügte Abbildungen, Zeichnungen oder Grafiken, die anderen Werken dem Wortlaut oder Sinn nach entnommen wurden, habe ich unter Angabe der Quelle kenntlich gemacht.

Bonn, den 12.05.2023

Ort, Datum

Unterschrift

Abstract

The increasing availability and affordability of unmanned aerial vehicles (UAVs) have facilitated their use in various applications, including construction and disaster response. In many of these applications, it is necessary to determine the global position of the UAV.

Global navigation satellite systems (GNSS) can provide this information but are susceptible to errors caused by terrain obstruction. GNSS results can be highly coarse, especially in urban environments where high buildings can block or reflect signals. This thesis presents a method for improving pose graph consistency through global pose refinement. To generate these improved robust global results in urban environments, we utilize georeferenced models.

For our method, we first assemble a local map by combining multiple LiDAR scans and applying semantic and constraint filters. After placing the local map at a GNSS pose estimate, we repeatedly register it against a georeferenced model at different positions inside the margin of error of the GNSS pose. We use a novel plausibility score to determine the most plausible local map placement. We base the score on the principle that LiDAR rays may not cross through objects while the ray endpoints should not be in free space. The refined coordinates are then passed along with relative data to a graph optimizer to obtain globally consistent georeferenced results.

We evaluated our GNSS refinement strategy on numerous flight datasets and utilized a georeferenced model of North Rhine-Westphalia. The results show significant improvements near buildings. Finally, we discuss possible improvements and modifications to our method.

Contents

1. Introduction

The demand for unmanned aerial vehicles (UAV) has rapidly increased due to lower prices, maturing software and many applications in the industry becoming more feasible. They are already being used in industries, such as agriculture [10, 35, 58, 62], constructions [24, 25], or disaster response [1, 2, 28, 44, 46, 50, 57, 59].

The German rescue robotics center (DRZ) aims to support first responders in high-risk situations by employing multiple robots in a team. One of the robots is a UAV tasked to survey and assess dangerous situations. Figure 1.1 depicts the demonstrator D1 during an exercise to explore a warehouse after a reported chemical spill. [28]. The UAV provides the first responders a valuable live view at the site without endangering any personnel. The images can also help operators of other robots, such as ground vehicles (GV), with more limited perspectives. Advances in autonomous assistance functions promise for the future to help first responders to perform surveying more safely even in stressful situations [47]. For this system a global positioning system is useful,e.g., to show the different robot positions.

Figure 1.1: UAV demonstrator D1 [47] during an exercise [41].

The above-mentioned industrial applications require UAVs/UGVs to locate themselves globally, primarily for higher-level applications, not concerned with controlling the robot momentarily but with bigger-picture tasks. These tasks may include flight path calculation with global estimates, coordination of multiple robots or path finding to the objective from global coordinates. An additional use-case may

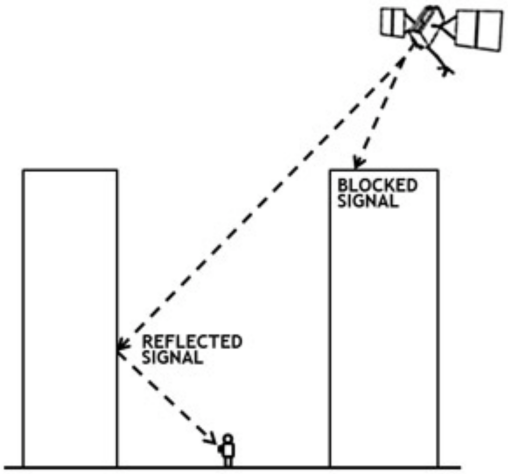


Figure 1.2: Structures like buildings interfere with GPS through signal reflection and occlusion, taken from [19].

be the access to information in priori annotated or generated maps. For these purposes, Global Navigation Satellite System (GNSS), like GPS and GLONASS, are employed to locate in current positions in global coordinate systems, such as the Universal Transverse Mercator (UTM) [37].

Satellite-based GNSSs, like GPS, are available worldwide. Nevertheless, they have two noticeable drawbacks: GNSS does not work underground, and GNSS is coarse with around 5 m [56], particularly in urban environments. In addition, it is affected by the weather phenomena such as clouds or storms. In the exemplary case of GPS, ideal conditions for GPS are a clear sky and a plain field without any obstacle that might cause interference; these conditions are only possible for limited applications such as missions high above ground or over the ocean. This thesis will focus on a more urban environment that is not conforming to these conditions.
The urban environment has a high density of obstacles, i.e., buildings. Each building may causes interference by either completely blocking the GPS signal or creating reflections that delay the signal. These two effects combined create the canyon effect and reduces accuracy even further. This causes even more coarse positioning and is most common in streets with tall building facades, as shown in fig. 1.2 [19].

Consequently, relying on GNSS alone for low-level applications, such as collision avoidance, that require localization with cm-accuracy is undesirable. Hence, local positioning is generally achieved with odometry computed from onboard sensors

like cameras, IMU, and LiDAR. For UAVs, ultrasonic sensors or barometers provide additional height information. Positioning based on odometry yields higher accuracy than GNSS in the short term. However, it has the downside that minor errors add up over time and create drift, leading to inaccurate global results. In this case, the coarser global positioning may negate these errors. For this purpose, one can identify landmark locations, such as revisited locations, since these may be used for a spline-based optimizer to revise the global position estimate.

A possible approach to refine GNSS data into a more reliable and precise localization is to find the position of the UAV in a georeferenced model of its mission environment. CityGML LOD1 model include georeferenced low-polygon building models, and digital elevation models (DEM), i.e., a low detail representation of the world. Such CityGML models are available in many German states [21]. North Rhine-Westphalia's (NRW) CityGML data [36] is available for free.
The position of the UAV in a georeferenced model may be determined via a local map generated from the UAV data, i.e., fitting the local map into the model. To reduce inconsistencies between the model and the local map, we may use semantics [8] to filter clutter.
Fitting the local map into the model may be achieved via registration methods like ICP [4], NDT [31], or registration with surfel maps [39].
Unfortunately, a single registration may lead to incorrect results. Thus one can attempt registration at multiple locations given the GNSS's uncertainty. In order to then determine the most plausible result, the quality needs to be evaluated, e.g. as done by Shetty and Gao [48], and Torres [55].

Our method streamlines the above concepts into a straightforward pipeline and proposes a novel plausibility score based on the percentage of uninterrupted rays and hits of a registered LiDAR scan. The thesis is structured as follows: We introduce related work in chapter 2. Chapter 3 presents our pipeline for robust registration, our plausibility score, and its application in pose graph optimization. We then evaluate the proposed method in chapter 4 and present the conclusion and future works in chapter 5. This thesis contains the following contributions:

1. **A robust registration approach:** Our pipeline constructs local maps and registers them against a georefrenced model at multiple potential positions to avoid getting stuck in local minima, increasing robustness.

2. **A novel intuitive plausibility score:** We base the score on the assumption that LiDAR rays can not be inside models, and meanwhile, ray end-

points should not be in free spaces.

3. **An improvement to robustness and consistency of Pose graphs:** We feed the refined GNSS results from our registration approach as additional constraints into a pose graph optimization to georeference the pose graph while reducing overall drift.

4. **An improved global positioning of pose graphs:** The graph is globally positioned, e.g. allowing the fusion of coarse environment models with more detailed LiDAR data.

5. **A thorough evaluation:** We extensively evaluate the robustness and consistency of our registrations and the pose graph on diverse flight datasets and two CityGML models.

2. Related Work

The following will discuss related works to our method. The first topic covers different urban datasets. Secondly, we introduce building information models, such as CityGML. Thirdly, we discuss different registration approaches. Then, we showcase different approaches to generate possible registration poses and present alternative plausibility evaluations to our method. Consider that some of the authors do not explicitly refer to the mentioned plausibility evaluations as such. Lastly, we introduce some alternative GNSS refinement approaches.

2.1. Datasets

In the problem statement, we briefly mentioned that one of the drawbacks of the registration against an existing model is that it has to be generated prior to the actual flight. One approach is to survey the mission site in advance with a UAV to generate a map from higher altitudes with a valid GNSS connection. Christie et al. implemented this technique and mentioned several drawbacks. The top-down perspective of the map has up to no information on walls which are crucial. In addition, high altitudes still do not ensure precise localization as explained in the introduction [11].

Another approach is to manually survey the mission location beforehand to generate highly accurate and detailed maps, as was done in the Newer College dataset by Ramezani et al. [40] for their ground truth calculations. They used a tripod-mounted stationary survey-grade BLK360 LiDAR scanner to generate millimeter-accurate maps for this task. This process is time intensive as one has to be at the mission sites beforehand, and the scanner has to be handled by experts.

In contrast to the two previous approaches, the open street map (OSM) collaborative project encompasses large regions of the world, e.g., all cities of Germany. However, this data is significantly less accurate than the model from the Newer College dataset, as the OSM data is based on crowdsourcing. Numerous users gather GPS positions and combine them into lines (e.g., streets) or polygons (e.g., buildings). In addition, users can tag the buildings and insert vital building details such as the height, roof shape, and level number. The tags can also encompass

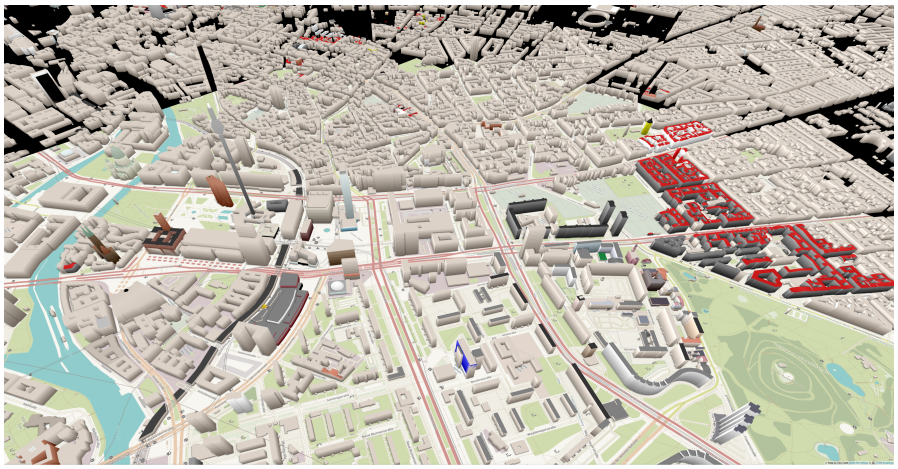


Figure 2.1: Differences in detail are obvious for Open Street Map in the city center of Berlin [33].

more advanced model details, such as color, roof style. Fig. 2.1 shows generated rough 3D models. The level of detail supported is highly dependent on the region. The Netherlands donated their state-maintained building databases to the OSM project providing detailed and accurate data [17, 22]. Although, OSM can indirectly store 3D information, it is pretty inconvenient.

A collection of datasets for cities all over Germany exists in the CityGML format [38]. Goetz [16] proposes an approach that uses OSM tags to generate highly detailed buildings for the CityGML format. However, the standard approach is that government surveying offices publish their data. The German state North Rhine-Westphalia allowed open access to CityGML datasets for every region, including over six million models [29]. However, these models are primarily prism-shaped buildings similar to fig. 2.1 . In the subsequent section, we will examine the GML format as well as others.

2.2. 3D Model Formats

This section introduces the common 3D model formats (stl, obj, and ply) and then discusses a format specifically designed to store building and city information: the CityGML format. At last, we will review two formats we used for our methods implementation: the Point Cloud 2 and Surfel Map formats.

2.2.1. STL, OBJ, And PLY

This section will introduce four 3D model file formats: STL, OBJ, PLY, and GML.

STL stores per triangle one normal vector and three vertices. This technique leads to many redundant duplicate vertices. STL can also not represent round surfaces. Instead, it requires to use an approximation with numerous triangles. Despite these two shortcomings and a lack of storage for additional information, the format is popular due to its use in numerous computer-aided design and manufacturing (CAD and CAM) softwares [34, 51].

In contrast, the OBJ format contains a wide variety of possible geometric types. The format stores 3D information by defining each model element as a key-value pair. As an example, OBJ stores vertices as individual keys with coordinates. Later on, when defining a face key, they solely refer to the vertex keys, which avoids the problem of vertice duplicates. In addition, there are keys for materials, textures, groups, and parametric surfaces, i.e., round surfaces are possible.

The ply format is very similar, although it separates the data from the description. The format is popular among academics since adding new user-defined keys is straightforward [34]. Tab. A.1 shows a triangle face representation in the three formats.

2.2.2. CityGML

The city geographic markup language (CityGML) format [21] was explicitly designed to hold building information and embed relations between models in order to model whole cities or countries, unlike the previous formats. It is used for the CityGML dataset. An additional motivation for the format was to make it reusable and accessible for many applications, e.g., autonomous driving, mobile robotics, city planning, etc.
CityGML is implemented in XML and has different levels of details (LOD) [21] based on how much information a file provides. This categorization ranges from zero to four and is listed below.

- LOD0, this LOD used to be a digital elevation model only, but a later version of GML added the representation of buildings as simple ground planes.
- LOD1 presents buildings as prism-shaped blocks. It includes . There are no further details such as vegetation.
- LOD2 adds details like roof shapes. It is also possible to color-code different materials and include vegetation.
- LOD3 is as detailed as possible (for the exterior). This LOD may also include high-resolution images.

- LOD4, this category adds to LOD3 by incorporating room layouts, stairs, and in some cases, furnishings.

Figure 2.2 showcases all five LODs. The first three LOD purpose is to represent grand-scale urban datasets, in which LOD2 exhibits landmarks only. In contrast, the last two LODs' primary use cases are individual projects.

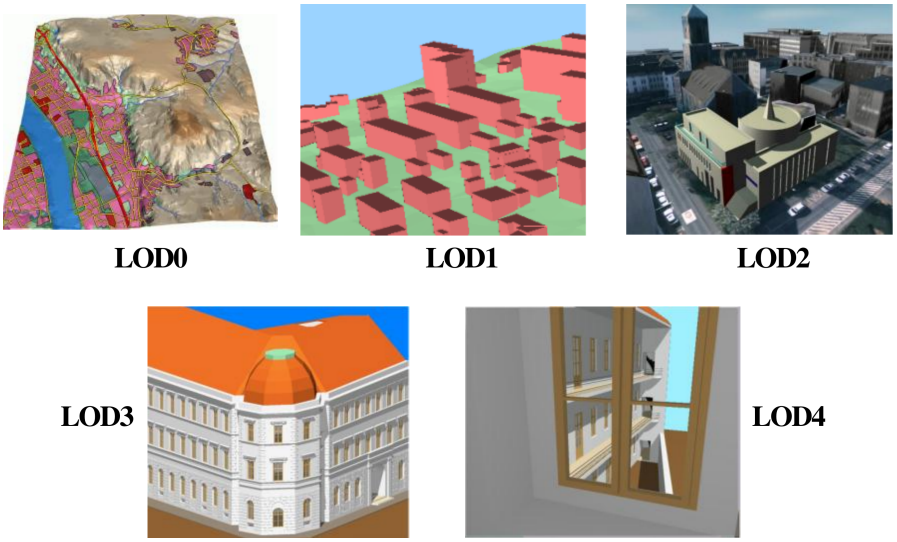


Figure 2.2: Examples for the different LODs, taken from [21].

2.2.3. Point Cloud And Multi-Resolution Surfel Map

We primarily employ two formats: point cloud [43] and multi-resolution surfel Map [39], to implement our method. We chose the former since this is the most natural way to store LiDAR data and is supported by ROS via their implementation of *sensor_msgs* PointCloud2. Meanwhile, the former increases registration efficiency considerably by combining multiple points via surfel.

The point cloud format is straightforward; it only stores 3D model information as points $p_i \in \mathbb{R}^3$. This format is ineffective in terms of storage efficiency but is convenient for conserving the output of LiDAR scanners. The point cloud implementation in *sensor_msgs* PointCloud2 can store additional information, such as the signal's intensity, the point's color, or semantics (will be expanded upon in a later section) [39, 43].

Processing raw point cloud data (e.g., for registration) becomes increasingly challenging to compute quickly due to newer LiDAR scanners' high output frequency and point density. Hence, Quenzel and Behnke [39] employ a surfel map

to reduce computing time. This approach divides the given point cloud via voxels or tetrahedra into cells. A surfel summarizes points falling into a cell by their mean μ and covariance Σ, which are calculated as follows:

$$p_i = (x, y, z)^\intercal, \tag{2.1}$$

$$N_p = |p|, \tag{2.2}$$

$$\mu = \frac{1}{N_p} \sum_{i=1}^{N_p} p_i, \tag{2.3}$$

$$Cov_{x,y} = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{N_p - 1}, \tag{2.4}$$

$$\Sigma = \begin{bmatrix} Cov_{x,x} & Cov_{x,y} & Cov_{x,z} \\ Cov_{y,x} & Cov_{y,y} & Cov_{y,z} \\ Cov_{z,x} & Cov_{z,y} & Cov_{z,z} \end{bmatrix}. \tag{2.5}$$

The surfel normal is oriented along the Eigenvector of the smallest Eigenvalue min( ) of the cell covariance.

$$det(\Sigma - \lambda E) = 0 \tag{2.6}$$

$$(\Sigma - \lambda E) \cdot x = 0 \tag{2.7}$$

In addition, the surfel map has different levels of detail. The base level one has the original cell size. Meanwhile, for each higher level of detail the cell size is divide, e.g., an original cell size of 1m becomes 0.5m in a higher level of detail. Higher detail levels are applied based on the distribution of the points [39].

2.3. Registration

In this section, the source refers to the local map that must be aligned with the static models, which will be referred to as the target.

2.3.1. ICP

The Iterative Closest Point (ICP) Algorithm by Besl and McKay [4] consists of two main steps. It first establishes correspondences between the source and target points by finding the closest point match for each point in the source. The second step minimizes the distance between each point and its correspondent by employing a least square error function that considers rotation and translation to reduce the error. The algorithm keeps iterating over the two steps until it reaches a convergence criterion and matches every point, i.e., it assumes a match-point in the target map to every point in the source map. Therefore the algorithm is less robust and prone to mismatches at the presence of many outliers, e.g., clutter [45].

2.3.2. NDT

An alternative to the ICP algorithm is the 3D Normal Distribution Transform (NDT) algorithm proposed by Magnusson et al. [31]. It uses divisions similar to the Surfel Map format approach but characterizes the subspaces by a normal distribution instead of surfels. The core loop of the algorithm is also iterative: It minimizes a score function based on how well the points from the source map fall into the normal distribution. NDT algorithm is considerably faster than ICP [31].

2.3.3. Registration With Surfel Maps

Quenzel and Behnke [39] use a Surfel Map as a basis for their registration, i.e., align surfels instead of points. The registration implements an expectation maximizer, that iteratively reduces the distances between matched surfel. Their approach considers the surfels' position and their normal and viewing direction. In addition, surfels representing the same surface are merged into one. In conclusion, this algorithm is also considerably faster than ICP [39].

2.4. Pose Search

The three registration algorithms introduced above all minimize functions iteratively. Hence, the algorithms will converge to some minima, which is not necessarily the global minimum. The authors of the ICP algorithm [4] Besl, and McKay, mention two approaches to test multiple minima to find the global one, i.e., they propose a pose search. These searches do not cover the task of determining the global minima yet. We discuss this in the following section.

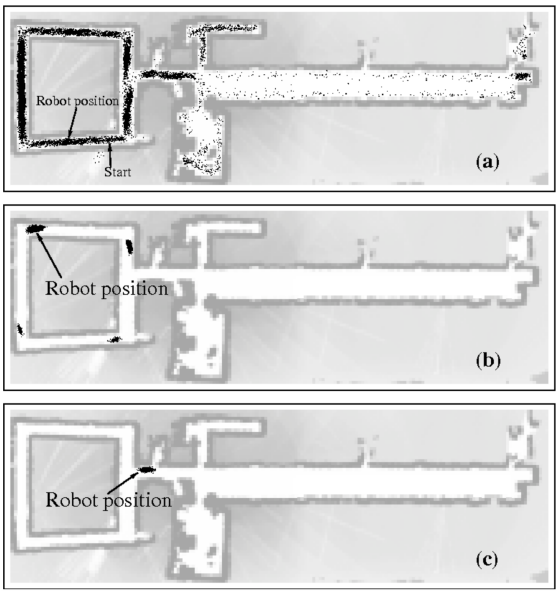


Figure 2.3: Particles, representing poses, are removed from unlikely positions and focus on certain spots, taken from [15].

2.4.1. Brute Force and Reduced Search

They explore a brute force strategy that tests almost every translation-rotation pair in a prior determined target area. Nevertheless, they conclude that the approach is computationally wasteful. On the other hand, they propose that ICP is not very sensitive to the initial translation, and thus a reduced pose search with different rotations alone will suffice. This assumption only applies to sources that include large portions of the target and is not the case for our problem [4].

2.4.2. KLD-Sampling

A popular method to compute probable poses is the Monte Carlo Localisation approach and its variation: the KLD-Sampling [15] by Fox. The general idea is to initially represent possible poses as uniformly distributed particles, remove those that are unlikely given some measurements, and add additional particles around the more likely ones. As shown in Fig. 2.3 the particle density increases in a few locations after several iterations. Fox adds adaptivity to the number of samples used to avoid computational inefficiency. This addition reduces particles when the position is unambiguous and increases the amount when it becomes uncertain again.

2.5. Plausibility Score

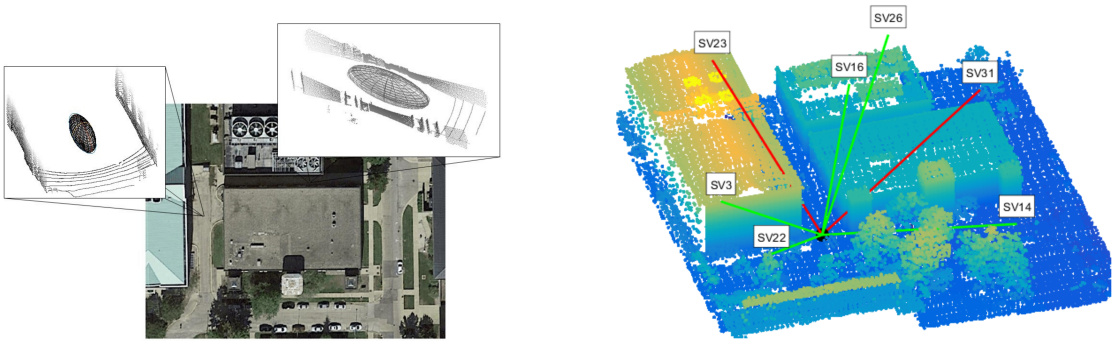
The previous section discussed different pose search approaches to find many possible poses. The topic of this section is to differentiate between more and less suitable poses, i.e., the most plausible registration. A plausibility score allows an algorithm to identify the most suitable pose that fits the local map into the model.

2.5.1. Feature Extraction Based

The following two approaches base their plausibility score on feature extraction. Shetty and Gao [48] use features to score the suitability of the registration environment, while Torres focuses on wall intersections to determine the best fit.

Shetty and Gao [48] present a plausibility score based on the environment's ability to constrain the registration. They presume that strong constraints lead to accurate ICP registrations. Consequently, they focus on the quality and quantity of these constraints. To identify constraints, they locate feature surfaces and edges (distant features are given less importance due to LiDAR being more accurate in short ranges). The features reduce the position error accordingly: for surfaces along their normals, and edges along perpendicular directions, e.g., horizontal edges would reduce the error in the vertical direction. They visualize the degree of constraint as an ellipsoids. The smaller the ellipsoid, the more confident they are with the results. Fig. 2.4a shows how building corridors constrain the position error along the wall normals. Meanwhile, the other directions have a higher uncertainty due to the lack of walls that could produce constraining feature points. In addition to this metric, Shetty and Gao use the model to determine whether any GPS signals are non-line of sight (NLOS) signals. This way, they remove signals received due to the canyon effect (Fig. 2.4b). This information generates an additional accuracy score, which Shetty and Gao also use as input for the unscented Kalman filter [48].

Torres [55] proposes a plausibility score based solely on wall intersection points. His approach is somewhat similar to ICP in that he also finds matches and pulls them closer together. Nevertheless, the match finding is more specific and detailed in Torres' case, although it only applies to two-dimensions. Torres utilizes the intersection point of walls as relevant features only. For this purpose, he first reduces the local and model to the x-y-plane, resulting in a floor plane. A RANSAC algorithm then detects walls and, subsequently, wall intersections. Torres then detects possible matches between the points by comparing the intersection angles. Then a mixed-integer linear program produces a transformation that maximizes the number of possible matches. This algorithm segment integrates the number of



(a) (b)

Figure 2.4: Shetty and Gao evaluate the local map as well as the GPS signals. The image on the left incicates the number of constraints offered by the local map as an ellipsoid, they improves the registration quality. For the figure on the right, red signals are non-line of sight (NLOS) signals. Meanwhile, Green rays are unobstructed, taken from [48].

wall intersections as a plausibility score, i.e., Torres determines that numerous wall intersection matches indicate that the transformation is correct. After applying the x-y-transformation, Torres aligns the z-axis as well. The algorithm is robust as it considers every possible match in the model, thus requiring only a very rough initial estimate. Although it can be noted that having an adequate amount of wall intersections in the local map is often not feasible.

2.5.2. Ray-Casting Based

The following two approaches simulate casting the ray from the LiDAR scan in the model and compare the results to the original measurement. We, too, employ this technique. Christie et al. [11] propose a plausibility score for the following scenario: a registration of a local map generated by an Unmanned Ground Vehicle (UGV) against a model created by a UAV. The challenge is to find features to compare that are equally well represented from a bird's eye view and a frog's perspective to evaluate a pose's plausibility. The authors define a two-dimensional descriptor system based on features detectable from both maps. The descriptor stores the results from multiple circularly cast rays from the estimated position in the height of the UGVs sensor. The results include the range of each ray and semantics, i.e., the type of obstacle that blocked the ray. The semantics are divided into obstacle categories detectable by a UGV and a UAV, e.g., buildings and vegetation. The authors generate one descriptor for the local map and all possible positions on the model. They then compare the descriptor of the local map with different descriptors from the model. The score is the normalized sum of the individual ray

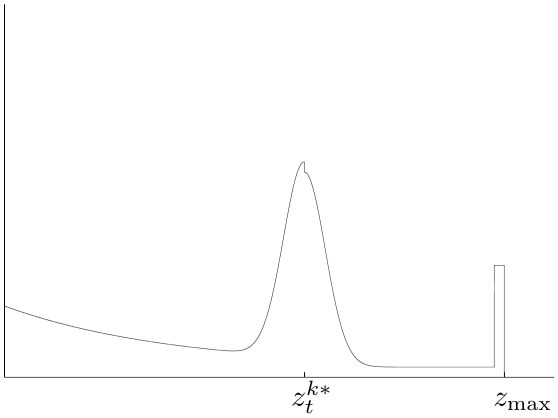


Figure 2.5: Probabilistic likelihood for a range measurement. z_{max} is the maximum measurement range of the LiDAR scanner and z_t^{k*} is the actual value, taken from [53].

scores and falls between 0 and 2, considering whether semantics agree, and the range difference between the UAVs and the UGVs descriptor [11].

The plausibility score proposed by Thrun et al. [53] considers four different measurement error types that can occur while scanning. For their score calculation, the authors compare the distance of each ray measured by the LiDAR scanner with the distance measured from a computed ray cast from the estimated position in the model. From now on, the actual value refers to measurements obtained by the model, and the measured value refers to the one acquired from the LiDAR scan. Thrun et al. solely incorporate the discrepancy between the measured and actual value. This discrepancy would be zero in an ideal scenario. The likelihood that a given discrepancy stems from a correct measurement is modeled by four distributions that represent the following four scenarios:

1. Correct measurement with noise: the sensor measured correctly but still has incorrect results due to sensor inaccuracies that were known beforehand. These errors may result in slightly varying range measurements. The likelihood of measurements being correct but being affected by a sensor inaccuracy increases the smaller the discrepancy between the real value and the measured one becomes, i.e., a zero-mean Gaussian distribution distribution around the real value can represent this error's effect.

2. An unexpected object blocking the ray scenario: the ray was unable to reach the correct length due to dynamic clutter, e.g., people. The likelihood for interference by dynamic clutter increases the shorter the measured value is. Since clutter closer to the scanner can block more rays and hide objects

behind it, furthermore, this effect cannot occur behind the obstacle. An exponential distribution can represent the error.

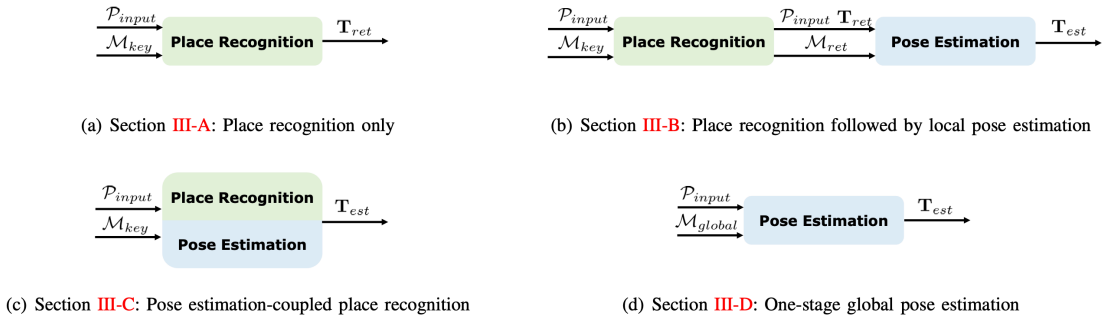
3. Rays that do not return scenario: this effect occurs when the target is out of the sensor's range or due to reflection and absorption. In all these cases, the ray does not return to the scanner, and the LiDAR will mark it this way (Thrun et al. assume that the LiDAR returns the maximum range). A small uniform distribution represents this scenario at the end of the LiDARs range z_{max}.

4. The random measurement scenario: this accounts for all other possibilities that a measurement failed, such as mirrored rays or interferences by other sensors. This error is independent of the measurement margin, and thus it is modeled with a uniform distribution.

Fig. 2.6 shows the one can see the combination of all distributions [53].

2.6. GNSS Refinement Methods Based On 3D Models

Our approach can be categorized in the Global LiDAR Localization problem. The survey by Huan et al. [61] divides approaches into four categories which are made up of two components, place recognition and pose estimation. Place recognition finds the rough current position by comparing the LiDAR with previously generated descriptors and the pose estimation refines its position an returns an exact position.



(a) Section III-A: Place recognition only

(b) Section III-B: Place recognition followed by local pose estimation

(c) Section III-C: Pose estimation-coupled place recognition

(d) Section III-D: One-stage global pose estimation

Figure 2.6: $\mathcal{P}_{input}$ indicates an input data (LiDAR point cloud). $\mathcal{M}_{key}$ and $\mathcal{M}_{global}$ represent prior keyframe-based submaps and a global metric/feature map respectively. $\mathcal{M}_{ret}$ and T_{ret} represent a keyframe and a robot pose by place retrieval in the keyframe-based submaps. $_{est}$ is a robot pose output by pose estimation module, taken from [61].

2.6.1. Lucks et al.

Lucks et al. [30] propose a pose refinement approach by employing CityGML LOD1 models. They start with a local map generated from odometry and a LiDAR scanner. Then they extract facades and streets separately and register the local map against the CityGML models with NDT. Since they use LOD1 models only, the researchers removed any ground or other clutter elements. They implemented this filter with a random forest that detects walls. Lucks et al. do not employ a pose search. Instead, to reduce local inconsistencies they segment the path trajectory and latter interpolate between them.

2.6.2. Cappelle et al.

An other pose refinement approach [9] relies on LOD3 and LOD4 models with high-resolution textures. Their approach directly relies on the map looking similar to the real world. They use a single monocular camera and compare its footage with a virtual camera placed in a model scene. The placement is based on GNSS or, in case of invalid GNSS, on odometry.

The authors then extract Harris keypoints from both images and match them. They calculate the transformations between the features and the deriving offset of the actual camera. This transformation is valuable as the virtual camera has a coordinate in the model, which is geo-referenced. The offset and the virtual camera together result in a refined GNSS.

2.6.3. Tang et al.

Tang et al. [52] employ place recognition followed by pose refinement strategy. Their map is based on google maps data. A priori they create many snapshots of the vehicle path and simulate two-dimensional LiDAR scans from their center, i.e., they ray-trace until a building is hit. They use multiple of these simulated scans to implement place recognition: The LiDAR scan of their vehicle is condensed to a two-dimensional version to achieve the cross-modality (Fig. 2.7). The reduced scan is then matched with one of the snapshots. The pose refinement is achieved with deep closest point (DCP).

2.7. Relative and Absolute Pose Graph Fusion

Our pose graph consists of many nodes and has edges from different relative measurements . Regarding absolute measurements, there are multiple modeling ap-

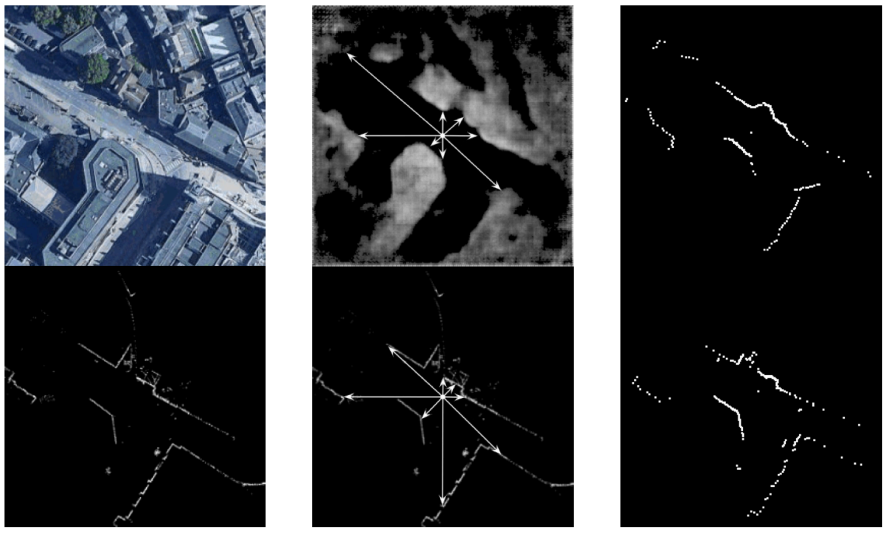


Figure 2.7: Virtual LiDAR scan extraction (top row) from google maps photo and reduction of corresponding measured LiDAR scan (bottom row), taken from [52].

proaches to incorporate them in a pose graph, [13] by Das and Dubbelman evaluates three of them. The first one represents absolute poses as further measurement edges, the authors conclude that this modeling approach is too rigid. They achieve better results by modeling the absolute poses as additional nodes. That are connected to the pose graph with edges that represent the global poses accuracy. They attempt this with the nodes position being adjustable or static. Their test show that, that adjustable nodes for GNSS data is most effective as it allows for more flexibility and thereby improves convergences.

3. Our Method

In order to refine our GNSS position estimate for the graph optimizer, we create a local map and register it against a model constructed from CityGML and DEM models. Our methods pipeline is constructed as follows.

Fig. 3.1 showcases our pipeline. We will now introduce the relevant and optional input data required for our algorithm (Sec. 3.1). Then we detail the local map creation (Sec. 3.2), before describing the registration and grid search (Sec. 3.3). Lastly, we describe how we generate relative transformations and feed them as constraints into a pose graph optimizer (Sec. 3.4).

The local map combines multiple semantically annotated LiDAR point clouds in a sliding window. Before combining the point clouds, we filter out clutter objects not represented in the model and transform the point clouds from the LiDAR frame to the map frame. After the map is created, we initialize the local map at a GNSS pose.

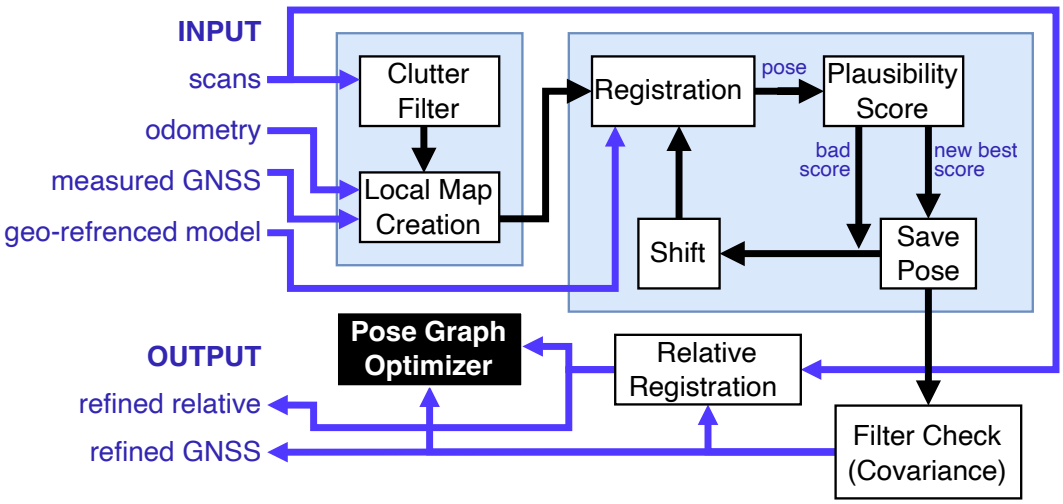


Figure 3.1: An overview of our pipeline.

We then offset the initial pose using a grid search. For every pose, we register the local map against the model, selecting the result with the highest plausibility

score. Our score considers the distance between the LiDAR scanner, the hit point for each ray in the model, and the hit/miss ratio.

The refined GNSS positions are able to georeference the spline optimizer. Nevertheless, relative transformations are essential to improve spline accuracy. We first detect loop closure match candidates. The candidates are then registered to obtain a relative transformation that is fed into the pose graph optimizer.

3.1. Input Data

Our method semantically annotated point clouds, odometry, GNSS, and a georeferenced model, e.g., CityGML and DEM, of the scenery location.

The semantic module [8] fuses multiple sources, such as thermal and RGB cameras, to annotate the LiDAR scans.

We define poses as 3D poses [54], elements of the Special Euclidean group SE(3), where $\boldsymbol{R}$ is our rotation and $\boldsymbol{t}$ is our transformation:

$$SE(3) = \left\{ T = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} | \boldsymbol{R} \in SO(3), \boldsymbol{t} \in \mathbb{R}^3 \right\} \tag{3.1}$$

$$SO(3) = \left\{ \boldsymbol{R} \in \mathbb{R}^{3\times3} | \boldsymbol{R}\boldsymbol{R}^T = I, det(\boldsymbol{R}) = 1 \right\} \tag{3.2}$$

We derive our GNSS poses from a combination of a GNSS-generated translation $\boldsymbol{t}$, UTM [37] values based on longitude and latitude from the UAV, and a rotation $\boldsymbol{R}$ based on UAV measurements, i.e., the IMU's roll and pitch and the magnetometer's yaw .

Height Sensor An ultrasonic sensor may replace the GNSS for more accurate height results. However, flights in high altitudes next to buildings can produce errors for ultrasonic sensors. The sensor can misidentify the ground, e.g., skyscrapers' facades as the ground floor (as happened at MBZIRC [5]). So to avoid mistakes, higher altitude measurements are conducted by barometers.

We generate our original odometry from the raw data with MARS [39] by Quenzel and Behnke to fuse the data into odometry poses . As mentioned before, the odometry is less prone to large jumps and locally more accurate than GNSS, as shown in fig. 3.2.

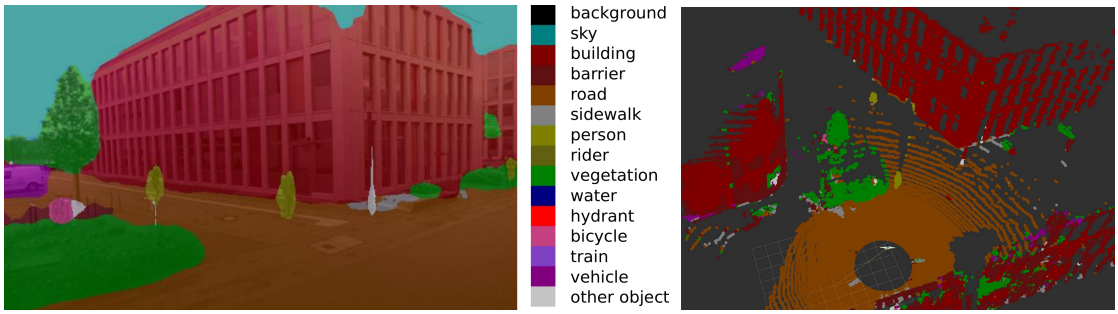


Figure 3.3: Example of semantic annotations, taken from [8].

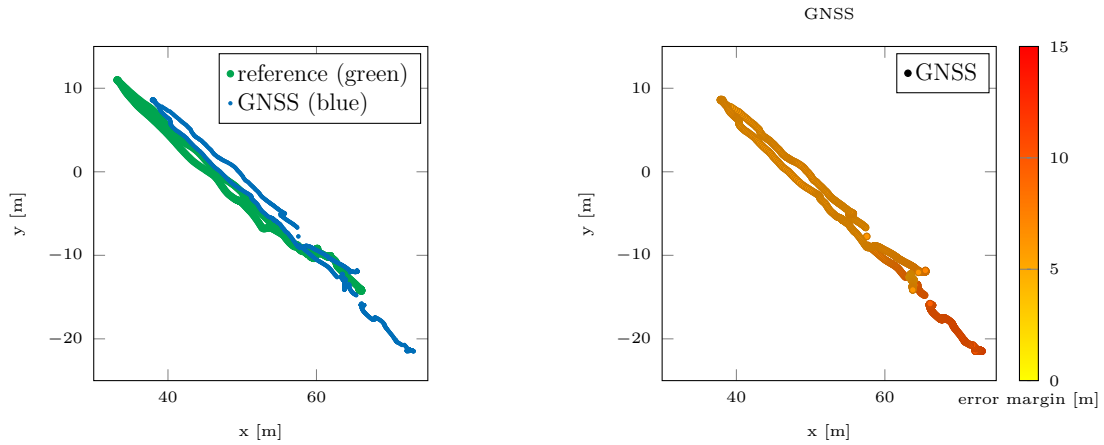


Figure 3.2: Inconsistencies between odometry and GNSS (left). Trajectory color-coded by error (right). Gaps are caused by unreliable GNSS.

3.1.1. Point Cloud Semantics

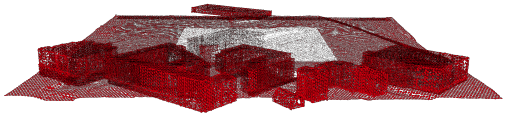
A common registration problem stems from scene measurements without correspondence in the model. The discrepancy reduces the registration quality since this results in mismatched points, possibly preventing convergence and resulting in implausible results [45].

We refer to points not present in the model as clutter. To remove these points, we segment points into different relevant semantic categories using the approach of Bultmann et al. [8]. Clutter categories include dynamic objects, such as people, means of transportation, and vegetation.

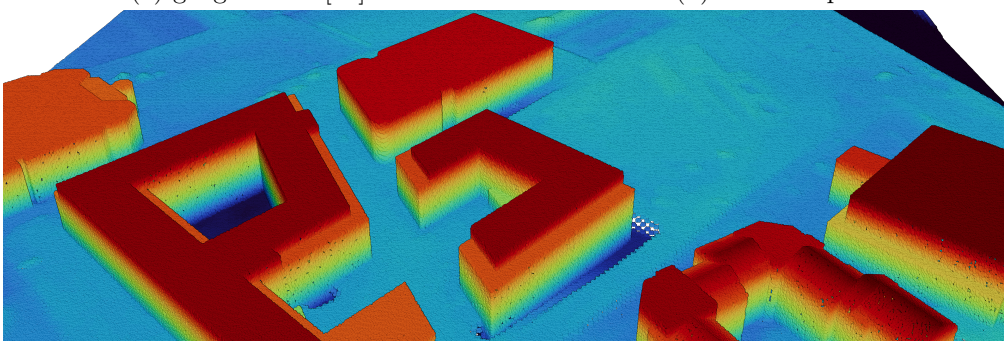
3.1.2. Model And Height Map

For the model, a point cloud is subsampled from a CityGML model and a DEM, which we show in Fig. 3.4c. Based on this point cloud, we generate a height map. The height map is a square grid with a side length h_n that we divide into cells with h_{width} intervals. Each cell stores the height of the highest point that falls into the cell. Fig. 3.4d shows an exemplary height map. Finally, the point cloud is also converted into a surfel map for registration with MARS [39] as shown in Fig. 3.4b. The surfel are calculated according to Sec. 2.2.3.

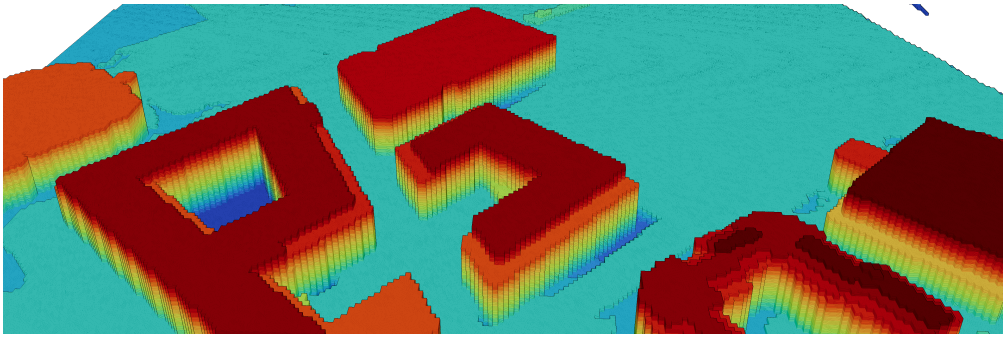
(a) google earth [18]



(b) surfel map

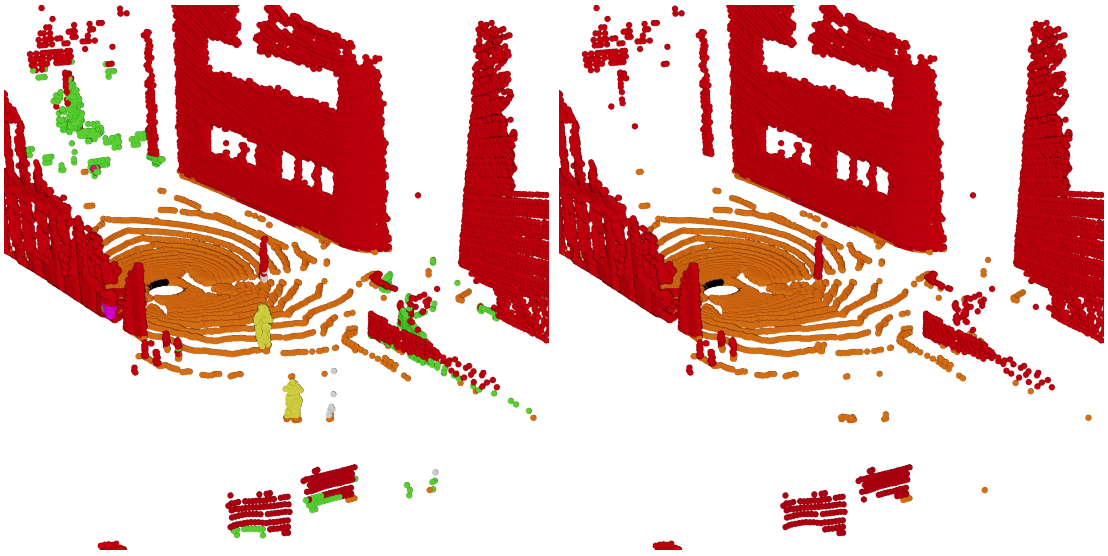


(c) point cloud



(d) height map

Figure 3.4: Fig. a shows the Poppelsdorf campus of the university of Bonn. The figs. b-c shows the same scene recreated from a CityGML model and a DEM in different formats.



(a) (b)

Figure 3.5: Point cloud with semantic categories - building (red), road (orange), people (yellow), vegetation (green). (a) before semantically-assisted clutter removal and (b) after the removal, mainly the person, the lamppost, and the tree, were removed.

3.2. Local Map Creation

This section discusses our local map creation. On the one hand, it would be possible to only register a single scan, they will include less features, may be obstructed, and have a restricted field of view. On the other, local maps avoid or reduce these flaws by aggregating multiple scans, reducing the effect of outliers.

3.2.1. Semantically-assisted Clutter Removal

A modern LiDAR creates an organized point cloud with precise measurements. The point data is in spherical coordinates with two angles and the measured range. Due to the organized structure, we have to discard any invalid measurements, e.g., too close or too far.

We then remove all points with semantic categories that are either dynamic entities or are too detailed for the LOD2 CityGML model we used in our experiments.

Thus, we choose to filter points from the following categories: *person, rider, vegetation, sky, water, hydrant, motor-/bicycle, train, vehicle, and foreground objects*. Fig. 3.5a shows the color-coded semantic point cloud before (3.5a) and after removal (3.5b).

3.2.2. Transformation

The following transformations are needed to transfer a point cloud from the LiDAR frame to the map frame:

1. The transformation T_{b_l} from the LiDAR scanner to the $base_link$ of the UAV.

2. An odometry transformation T_{m_b} to locally position the UAV (this contains drift error).

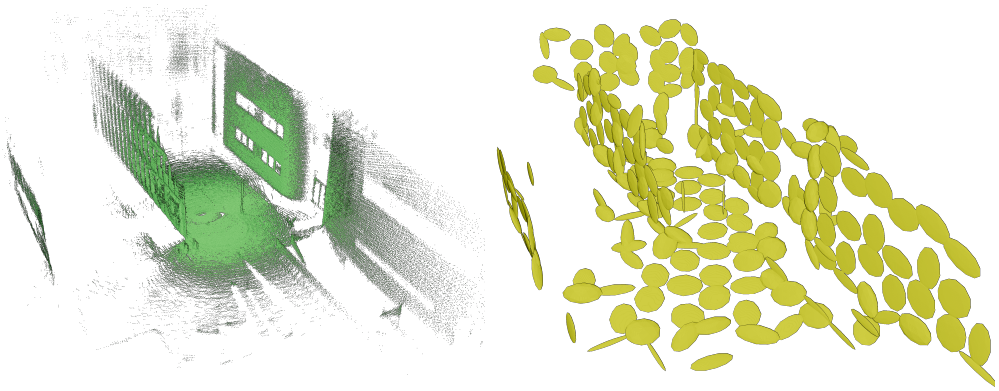
The uncertainty of the local estimates increases over time (due to accumulated drift). Therefore, it is vital to reduce the uncertainty with global GNSS position estimates.
After applying all transformations, the point cloud is ready to be inserted in the local map.

3.2.3. Extending the Local Map

Our local map is a collection of N_w point clouds with a Euclidean distance of d_w. We employ a map window, i.e., limit the number of simultaneously observed LiDAR measurements. A low N_w keeps the computational time reasonable by limiting the number of points to consider in the plausibility evaluation while reducing the effect of drift. Meanwhile, a larger N_w contains more scene information and, thus, more registration constraints. In a similar vein, a large N_w is prone to include drift errors and a small N_w only covers little scene information.

We build a local map with N_w and d_w for each point cloud with a valid GNSS signal (the GNSS health provided by the sensor fulfills the criterion $\leq \theta_{GNSS}$). Before employing our pose estimation, we evaluate the condition number κ of the local map. The κ describes how well the surfel map (Sec. 2.2.3.) is constrained and is based on the normal distribution of the observed surfels. For this purpose, we calculate the covariance matrix Σ of the surfel maps normals and take the smallest and largest eigenvalues λ_{min} and λ_{max}. Local maps where the ratio between minimum and maximum surpasses a τ_κ are discarded:

$$\kappa = \frac{\lambda_{min}(\Sigma)}{\lambda_{max}(\Sigma)} \leq \kappa_{th}. \tag{3.3}$$



(a) point cloud (b) surfel map

Figure 3.6: We use the point cloud on the left (a) for our plausibility check and the surfel map format on the right (b) for registration.

We then position the map at the GNSS estimate, that is temporally closest to the first cloud of the local map (this was determined before building the local map). Fig. 3.6 shows an example result from our tests.

3.3. Relative Pose Estimation

To estimate the relative GNSS offset, we employ a grid search followed by fine registration. Each iteration within the grid search consists of offsetting the raw GNSS pose, registering the current local map against the model, and evaluating the outcome with our plausibility score.
We do not solely rely on registration because the convergence radius of MARS is smaller than the GNSS error margin. Hence, the registration may converge to a local minimum. To identify the global minimum among the local ones, i.e., the best-fitting pose, we shift the starting position for the registration by moving the local map along a grid. After each shift, we register the local map against the model, and then we calculate our plausibility score and select the registration-adjusted pose with the highest score to obtain a more robust result in a wide user-defined area, e.g., a 16-meter radius as shown in Fig. 3.7.

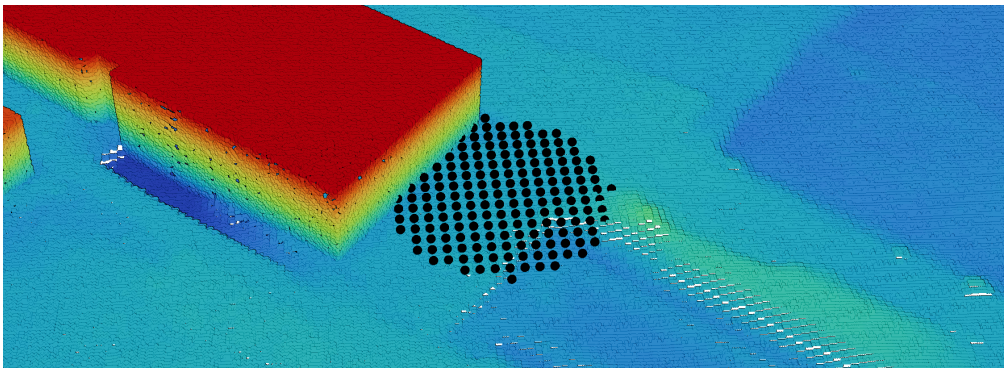


Figure 3.7: Our search grid (black) with a radius r_g=16m and a cell size δ_g=2m overlayed on the model point cloud.

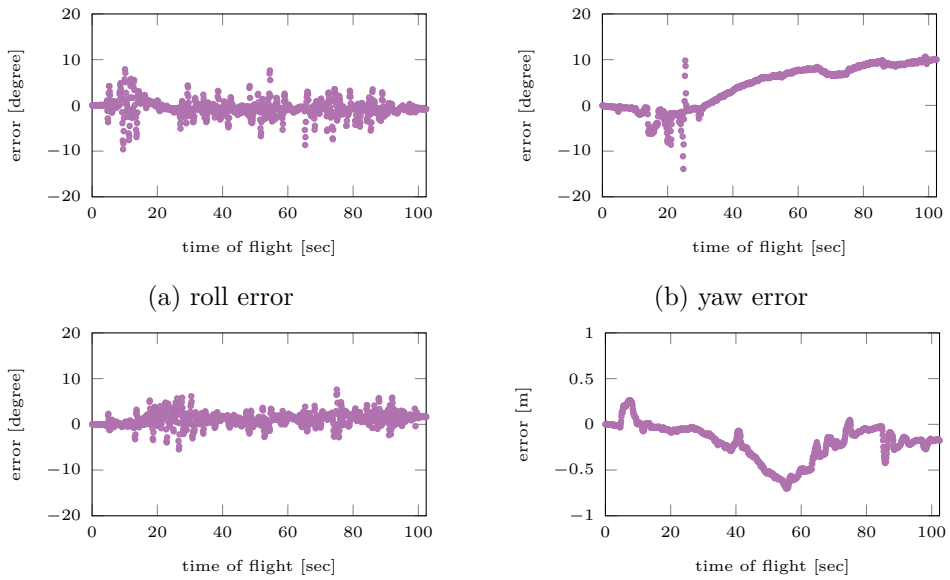
3.3.1. Grid Search

For our initial pose search, we employed a grid search (Fig. 3.7). We center the circular grid around the original GNSS pose and exclusively change the translation of the pose, excluding the height estimate, since it is adequate due to the barometers' small angle error. Our search's circle grid radius size r_g depends on the typical upper bound for the GNSS error, e.g. 16 m. Meanwhile, the search grid cell size δ_g depends on the registrations' basin of covergence. For all initial estimates within the same basin, the registration will converge to the same minima. This way, we can find and compare all minima to identify the global one.

Usually, this approach would require testing many possible poses, c.f., AMCL [42]. This approach is unnecessary for our problem scenario due to its many constraints. Unlike AMCL, our search area only includes a small region within the model since we only have to check an area around the GNSS estimate. In addition, the registration mitigates minor orientation errors and the height of the UAV. A slightly different orientation does not have to be evaluated explicitly e.g. with a modified rotation R of the GNSS pose, i.e., modifying the rotation R of the GNSS pose. We graphed the error of a test flight (Fig. 3.8) to confirm that the angle and height error is in the convergence basin. More flights are represented in Fig. B.1.

3.3.2. Single Ray Plausibility Score

The different poses generated by our grid search require a score for plausibility to determine the most fitting placement of the local map in the model. We propose an intuitive metric that compares the measured LiDAR ray with a projected ray at the estimated position in the model. The plausibility score returns values between



(a) roll error

(b) yaw error

(c) pitch error

(d) height error

Figure 3.8: Error margins recorded for our fine tuning flight 16-44

zero (highly implausible) and one (highly plausible) The metric uses two evaluation criteria:

- The **ray criterion** ray_s compares the number of cells along the measured ray in cells L and the number of unobstructed cells along the projected ray within the model M:

$$ray_s = \min\left(\frac{M}{L}, 1\right). \tag{3.4}$$

- The **hit criterion** hit_s considers whether a projected ray overlaps with the model, i.e., the endpoint's height h_p is below or above the model's height h_m:

$$hit_s = \begin{cases} 1 & if h_m > end h_p, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Each criterion rewards good placements and penalizes implausible placements differently. However, both enforce a basic assumption: a LiDAR ray can only intersect with the terrain once, specifically at its reflection point. Therefore, any premature overlap of a projected ray with the model is implausible since a LiDAR

scanner would have detected a shorter range in the real world. Respectively, no intersections would have yielded a longer range.

The ray criterion considers this assumption by penalizing poses that place the projected rays so that they overlap with the model prematurely. Meanwhile, the hit criterion rewards placements that ensure that the endpoint of the projected rays hit.

We now detail how we compute the ray and hit scores. In order to detect the collision of a projected ray, we use Bresenham's line algorithm [7] to efficiently calculate the height map cells passed by the ray.

Ray criterion We calculate the ray criterion score for every ray of the clouds in the local map window using the previously generated height map (Sec. 3.1.2). ray_s is calculated as follows:

$$ray_s = \frac{M}{L} \tag{3.6}$$

Where M is the number of cells from sensor origin until the first intersection with the model and L is the number of total cells, i.e., all height map cells along the ray. To calculate M, we utilize Bresenham's line algorithm and analyze all cells between the sensor origin and the endpoint until a collision is detected. For each cell at x and y, we compare the lowest point in the cell $h_r(x, y)$ of the ray with the model elevation $h_m(x, y)$: $h_r(x, y) > h_m(x, y)$.

We calculate $h_r(x, y)$ at the cell's border, where the ray is lowest (based on the rays slope). If a ray segment does not fulfill the height condition, we detect a collision and terminate, returning the number of cells passed L until now.

Fig. 3.9 shows an example of how the rays traverse the height map cells and stop at the first model intersection.

Hit criterion Similarly, we calculate the hit criterion for every ray of the scan in the local map window. This time, we only examine the ray's end point. For this purpose, we again employ the height map to detect hits and misses. A miss requires the end points' height map cell to be lower than the end point's height. In contrast, a hit requires the end point to be above the cell height.. In addition, we check whether the endpoint is close to a cell border. If this applies, we extend the ray and try a second attempt, i.e., compare the height map cell height with the end point height because the cell borders are arbitrary. The indicator function for a single ray is as follows:

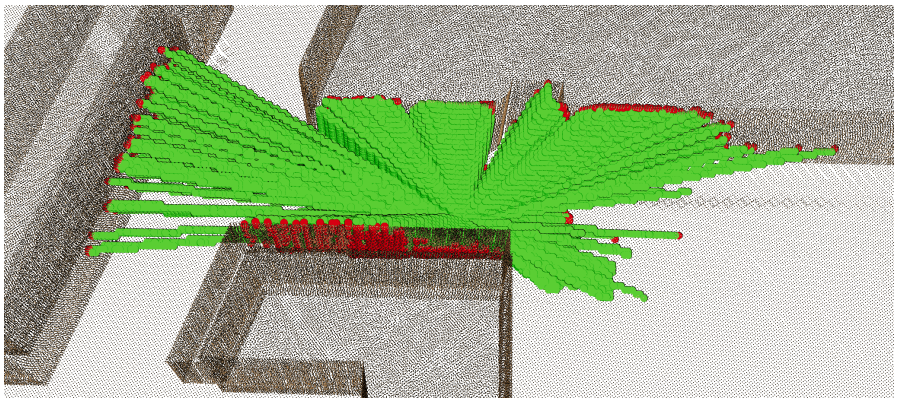


Figure 3.9: The rays are segmented based on the height map cells. Collision (red) with the model (black) occurs when the ray segment intersects the map. Ray segments above the map are collision-free (green).

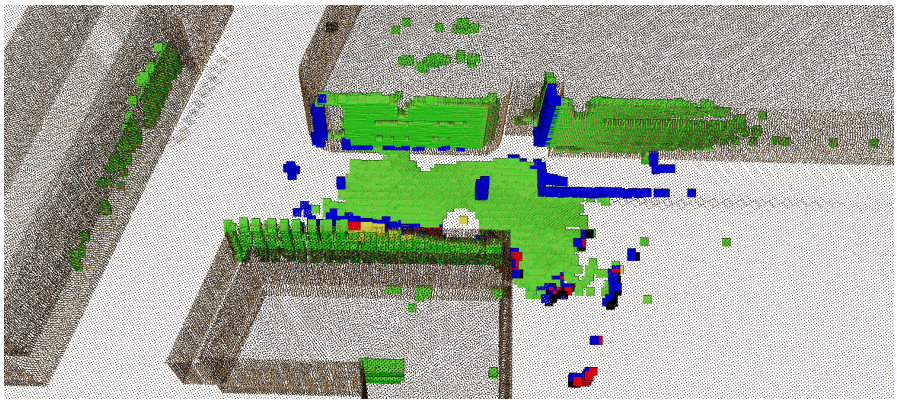


Figure 3.10: The figure indicates different endpoint scenarios: hits(green), initial misses(blue), second attempt hits(yellow), and second attempt misses(red).

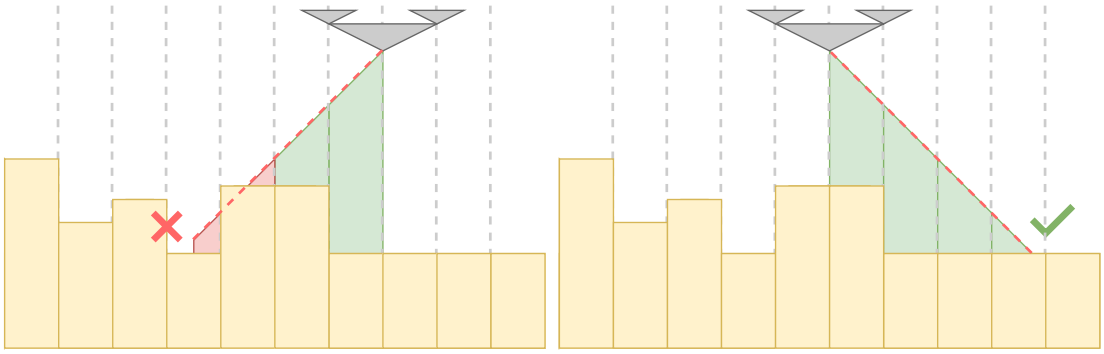
$$hit_s = \begin{cases} 0, & \text{miss} \\ 1, & \text{hit} \end{cases} \tag{3.7}$$

Fig. 3.10 shows the color coded end points.

3.3.3. Complete Point Cloud Plausibility Score

We combine the results of the individual criterias for a single ray as a weighted sum with weight hit_w as follows:

$$single_i = ray_{s_i} \cdot hit_w + hit_{s_i} \cdot (1 - hit_w), hit_w \in [0, 1]. \tag{3.8}$$



(a) $ray_s = \frac{2}{4}$, $hit_s = 0$, $R_i = 0.25$ (b) $ray_s = \frac{4}{4}$, $hit_s = 1$, $R_i = 1$

Figure 3.11: Simplified side view: The map model is divided into cells (gray dotted), and the area under the ray is colored based on whether an intersection occurred: unobstructed (green) or obstructed (red). The endpoint criterion is based on the overlapping of the ray's end.

We continue to calculate the per cloud results of the clouds and then of the local map with the mean to get a score between zero and one:

$$c_i = \frac{1}{|p|} \sum_{i=1}^{|p|} single_i, s_{local\,map} = \frac{1}{|w|} \sum_{i=1}^{|w|} c_i. \tag{3.9}$$

Figs. 3.11 showcase the score critria. Fig. 3.11a shows a sub optimal placement. The ray overlaps early with the model. The ray criterion ray_s is, $\frac{2}{4}$ and since the endpoint is not overlapping with the model, the endpoint criterion hit_s is zero. In the ideal scenario in fig. 3.11b the ray is passing all cells unobstructed, and the endpoint is connected with the model resulting in an R_i of 1.

In conclusion, the calculation of our metric is an intuitive approach to measuring the plausibility of a given positioning that detects estimated poses that disconnect or overlap the local maps with the model.

3.4. Graph Optimization With Absolute Poses

The graph optimizer [20] is a separate component to our pose refinement pipeline. The graph consists of nodes X_i and edges e_{ij}, which are initialized with multiple multi-modal measurements. The nodes are poses and the edges are measured constraints. These constraints may contradict each other, e.g., due to drift error. The optimizer will then attempt to reduce the overall error, i.e., adjust the poses so that the error w.r.t. all measurements is minimized. This is done iteratively each iteration adjusts the poses until the optimizer converges.

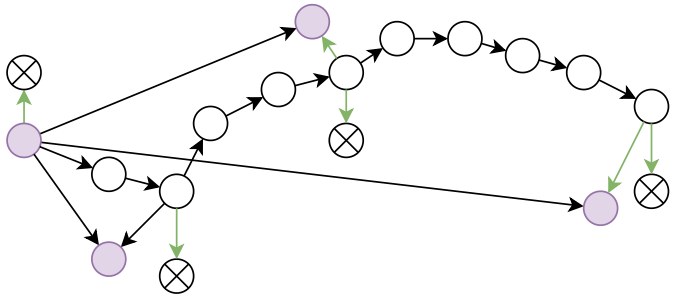


Figure 3.12: An abstract pose representation. Key-frames, i.e., our registration results are marked with purple, relative measurements, e.g., raw odometry are marked with white. Global nodes, e.g. GNSS poses are marked with a cross.

For this thesis, we use an optimizer implementation by Imbusch and Quenzel, that optimizes a continuous-time B-spline trajectory [49] using a pose graph. Time continuity allows to query the pose at any intermediate time without explicit representation of all intermediate poses. We feed our graph optimizer a number of inputs, including global poses. This enables us to add accurate georeferenced context to UAV positions. The global poses are modeled as additional graph nodes that are connected to the graph with an edge that represents the global poses uncertainty. Fig. 3.12 showcases an abstract graph representation of our pose graph. Regarding input, we use absolute refined GNSS and raw relative poses from the integrated sensors. In addition, we use MARS [39] odometry to increase local accuracy.

3.4.1. Relative Loop Closure Transformations

The following section explains a supplementary approach we implemented and evaluated. We add a new constraint by generating relative transformations between our global poses.

To increase in the accuracy of the optimizer, we process our optimized GNSS poses further. After each successful global pose refinement, we store the surfel map of the first LiDAR scan along with the refined pose. We then compare the new scan with previously refined scans based on a time and distance threshold. We use the time threshold τ_t to reduce redundancy, i.e., odometry on short distances is very accurate, and improvement is unlikely. The distance between refined GNSS poses is thresholded with τ_d to ensure a reasonable overlap between the two scans, such that an appropriate registration is possible. When two scans fulfill both thresholds and τ_κ, we register them against each other with initialization from their refined poses. The overlap between the two has to also fulfill τ_κ. We then store the relative

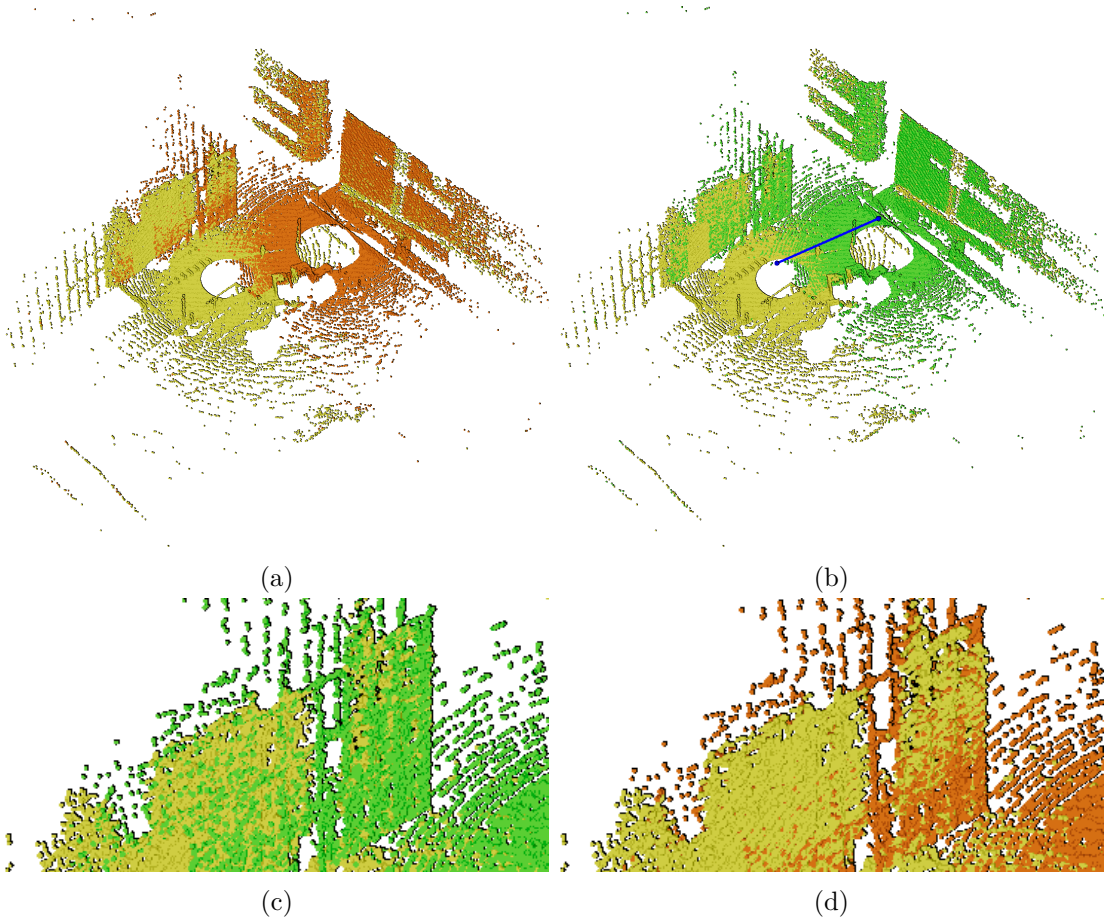
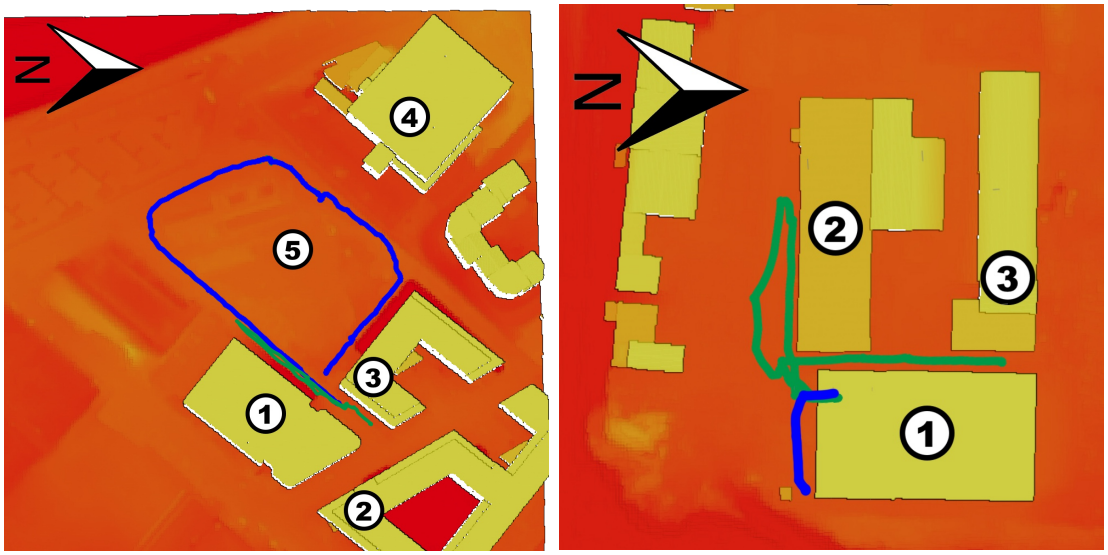


Figure 3.13: Two scans with refined global poses, clear the our criteria and are being registered. (a) The previous scan (yellow) and new scan before registration (orange). b) We registered the previous scan (yellow) against the new scan (green) and stored the relative transformation (blue). The close-ups before (c) and after (d) showcase the fine aliment of a wall.

transformation. This transformation has loop-closing qualities since τ_d allows us to detect recurring positions based on the refined GNSS. Lastly, store the relative results along the original refined poses with time stamps and covariance from the registration. Fig. 3.13 exemplifies the registration process.

4. Evaluation

This chapter covers the evaluation of our GNSS refinement pipeline and its effect on the optimizer. We separate these evaluations into two parts, beginning with the GNSS improvement. In both cases, we use datasets collected with a UAV at the Poppelsdorf (POP) Campus at the University of Bonn and the test center of the German Rescue Robotics Center (DRZ) [28]. The flights are listed in Tab. 4.1.



(a) (b)

Figure 4.1: On the left is the Poppelsdorf (POP) site (left): lecture hall (1), b-it building (2) and diet and simulations institute(3), Mensa (4) and the plain (5). We marked flight "16-38" (blue) and flight "16-44" (green). On the right is the German Rescue Robotics Center (DRZ) site (right) the flights revolve around three halls: main hall (1), long hall (2) and small hall(3). We also marked the flight "18-42" (blue) and "18-33" (green). Please consider that the flights were roughly overlayed for visualization purposes and may differ from actual results.

Model	Flight-name	Description	Scan msgs	GNSS msgs	Duration [s]
POP	16-44	A straight flight along the lecture hall (1) wall facing the simulations institute (3).	1030	5148	102
	14-18	Starts at the t-section between lecture hall (1), simulations institute (2), and the b-it building (3) and ends at the plain (5)	2241	12028	240
	16-38	A loop around the field (5) starting and ending at the intersection of the lecture hall (1) and the simulations institute (2).	2599	12996	259
DRZ	18-33	In front of the large hall (4) and entering it.	1197	2993	60
	18-34	From the front of the large hall (4) to along the long building (2)	1671	4179	83
	18-42	This flight starts behind the main hall at (5) and goes back and forth to (4)	9004	22511	367

Table 4.1: Path description of the flights on the Poppelsdorf (POP) campus and the German Rescue Robotics Center (DRZ) test site. The numbers refer to the maps in fig. 4.1

4.1. GNSS Reference Generation

For our GNSS accuracy refinement evaluation, we generated reference poses. For this, we rely on two point clouds. The first cloud is computed by precisely aligning the LiDAR scan point clouds with FAST-LIO [60] by Xu et al. and combining them into one. The second cloud is based on terrestrial laser scanning (TLS) data. The TLS data was acquired with a stationary survey-grade LiDAR scanner and converted into a point cloud by the Institute of Geodesy at the University of Bonn[1]. We register the surfel maps of the aggregated LiDAR cloud from FAST-LIO against the TLS data, similarly to the Newer College dataset [40]. We then apply the resulting transformation to the optimized odometry from the FAST-LIO. This yields the reference poses for the following evaluation. Unfortunately, no TLS data was available for the DRZ site. Instead, we used the long "18-42" flight after positioning it in the model and registering it. This approach may impact the reference accuracy negatively. Both resulting references are oriented east north up and have a UTM origin.

[1] $https://www.gib.uni-bonn.de/?set_language=en$

4.2. Model Conversion

We produced the model by combining a DEM and building models. Both models are available for free in North Rhine-Westphalia [36]. The DEM is based on airborn Laser Scanning (ALS) data and stores the terrain height in a grid of one meter cell size. The DEM is updated every five years, and buildings have been removed. The building models are stored in the CityGML format and are updated yearly. They are based on floor plans of the government cadastral office with additional height and roof shape are derived from ALS data. The shape of the roof is approximated with a typical roof shape that best fits the scanner data. This might cause intricate roof details to be lost.

Both models are converted to point clouds and combined, as shown in fig. 4.2. The DEM [27] cell width is one meter. Hence, bilinear interpolation on a 2D grid is used to obtain a denser ground model in point cloud. The building models [26] are first transformed from CityGML to CItyJson with [6] and then to OBJ [3]. Then polygons of the OBJ data are transformed into the point cloud format as follows. The point cloud is then subsampled with [12, 32] with a point density of 400 per m^2 of mesh surface area.

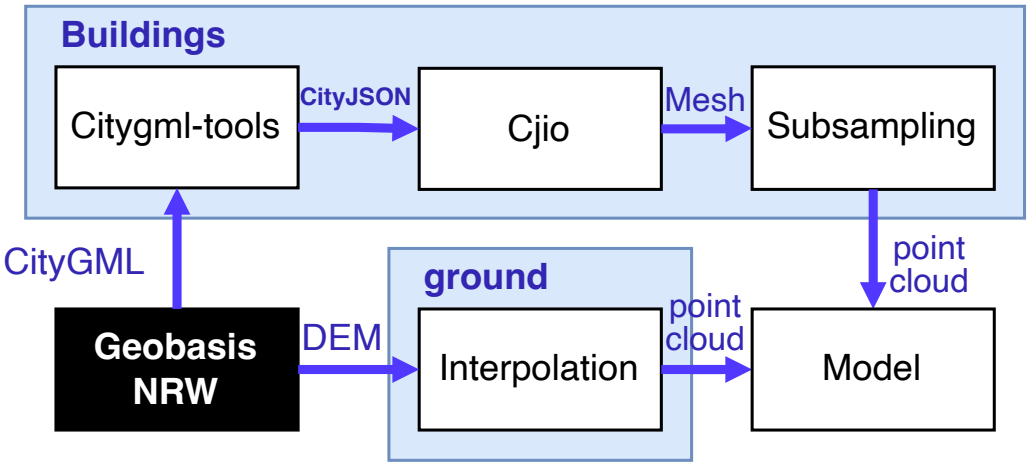


Figure 4.2: Model conversion pipeline

4.3. Evaluation Implementation

For our implementation, we chronologically read the LiDAR scans and GNSS messages. We then use two buffers: One for all scans with valid GNSS and one for all scans relevant to creating a local map for each, i.e., scans with correct d_w relative to a valid GNSS scan. A local map is created whenever a window is possible with a

leading scan with valid GNSS and $N_w - 1$ further scans that fulfill the d_w criterion. We remove clutter based on the CNN semantic annotations [8].

To test how well the local map is constrained, we calculate the local map's κ and check against a τ_κ. We position it at the raw GNSS estimated and start a grid search. Since individual shifts are independent from one another, we run the registration and plausibility evaluation in parallel. After the registration, we also check the κ of the model area that overlaps with the local map, to confirm that the registration was well constrained. We then store the resulting estimated poses and evaluate them with our plausibility score. We then apply the plausibility as descriped in Sec. 3.3.3. and select the refined pose with the best score. For comparison to our score, we also included the beam model scoring introduced in Sec. 2.5.2. We implemented the beam model proposed by Thrun et al. [53] from the ROS AMCL package [42]. The parameters for the beam model use the recommendations by Laet et al. [14].

4.4. Hyperparameter Evaluation

First, we analyzed the adjustable parameters (Tab. 4.2)and determined the most effective values, the calculated root-mean-square deviation (RSME) for all value sets is shown in Tab. A.2. We used the Popelsdorf "16-44" flight for hyperparamater estimation. Fig. B.2 shows the discrepancies between the reference and the GNSS estimates on this flight. Additionally, during the flight, there was a large dirt pile on the plain, highlighted in Fig. B.2a. This compelled the registration to match points from the pile with the plain in the CityGML model.

Local Map Parameter

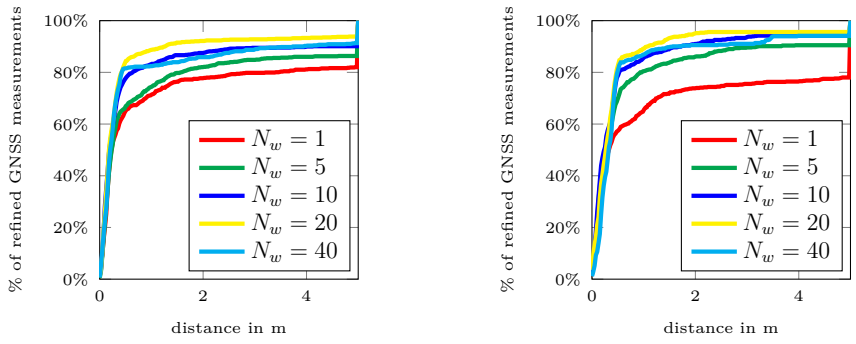
The defining parameters of our local map are N_w and d_w. Both affect the distance covered by the map, the distance is euqal to the product of the two. We expect to see the following correlations: too long distances introduce too much drift and too short distances do not cover enough features. To effectively compare the result we will from now on use the euclidean distance between the reference and the refined results. We used the cumulative sum and then divided it by the total measurements.

Parameter	description	tested	selected
N_w	Number of scans in the map window, i.e, how large is the window	1/5/10/20/40	20
d_w	Euclidean distance between the scans in the window	0/0.5/1(m)	0.5
τ_κ	Threshold on well-posedness of translation	5/10/15/20	15
semantics	Filter of clutter, e.g., trees and people	true/false	true
M_{res}	The size of the largest cell size of the surfel map	1/2/4	2
M_{len}	The side length of the square area covered by the surfel map	512(m)	512(m)
M_{lvl}	Additional level allow surfel to be included with half the size of previous levels beginning with M_{res}	1/2/3	3
r_g	The radius length of the circle area of the grid search	16(m)	16(m)
δ_g	The grid interval size	1/2/4	2
h_n	Number of intervals in the Height map	300	300
h_{width}	Size of one interval of the Height map	0.4(m)	0.4(m)
τ_d	Maximum distance covered by a local map	$N_w \cdot 0.5$(m)	$N_w \cdot 0.5$(m)
τ_t	Minimum time covered by two scans before they are compared	50(sec)	50(sec)

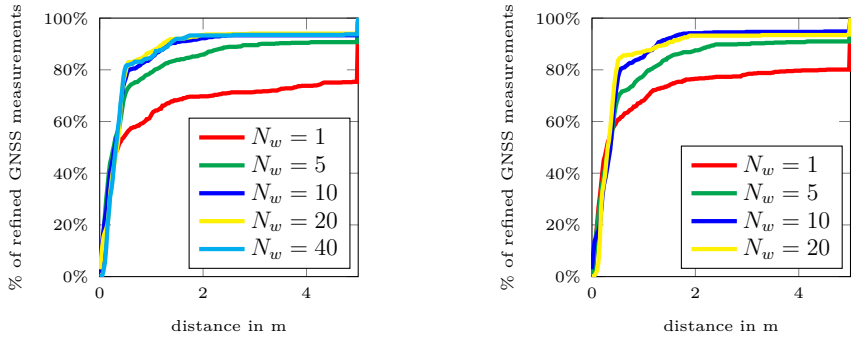
Table 4.2: The Hyperparameters of the experiment and their tested alternatives.

The results show the trends mentioned above. One can see drastic changes in accuracy, especially between the first three parameters of N_w. In addition, $N_w = 40$ appears to reduce accuracy. We receive the best result with a d_w of 0.5 and a N_w of 20, covering exactly $10m$ (RMSE=1.252).

The MARS map used for the registration has the three parameters M_{res}, M_{len}, and M_{lvl}. M_{len} has to be a power of two due to implementation restrictions and be large enough to ensure that the target position is included. Therefore, we made M_{len} conservatively sized. We picked $512m$.

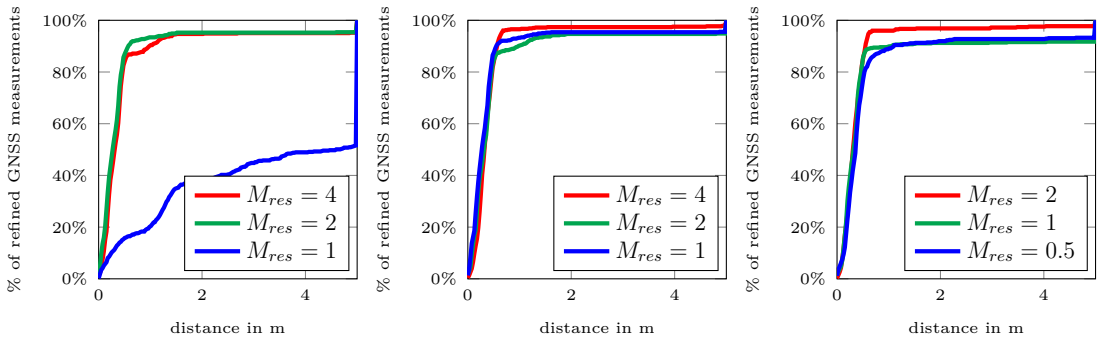


(a) $d_w = 0.0$m (b) $d_w = 0.5$m



(c) $d_w = 1.0$m (d) $d_w = 2.0$m

Figure 4.3: The graphs showcases the euclidean distance between the reference and our refined GNSS with adjusted N_w and w_{margin}.



(a) $M_{lvl} = 3$ (b) $M_{lvl} = 2$ (c) $M_{lvl} = 1$

Figure 4.4: The figure showcases the euclidean distance between the reference and our refined GNSS with adjusted M_{lvl} and M_{res}.

Meanwhile, M_{res} is adjustable as it decides the granularity of largest cell size, more granular cell sizes are add with M_{lvl}. Each level of the model will have double

the resolution of the previous level,e.g., in our case we have cell sizes $2.0m$, $1.0m$ and $0.5m$. We expect that too fine surfels will need more points from the clouds to benefit the registration. Meanwhile, a too-coarse resolution will likely simplify the scene too much. Unexpectedly 2m and 4m where more accurate than $2m$, this happend since the dirt pile was not well represented, i.e., it effect was reduced. This was unintended, therefore the more consistent M_{lvl} of 1 and M_{res} of 2 was chosen.

Next, we tested the effectiveness of the semantics. Since we introduce the semantics to reduce clutter, we expect a positive change in accuracy results. Fig. 4.5a shows that this is indeed the case (RMSE=1.134). However, we can also see that in case of errors, these tend to be larger than without semantics.

The inaccuracy of the semantic filter-enabled results occurs around the dirt pile. The dirt is not a category and is falsely classified as a wall. Unfortunately, we remove the grass around the dirt since it falls into the vegetation category, which we filter due to it also including trees. Thus, registration matches the points from the dirt pile with the plain.

Pipeline Paramter

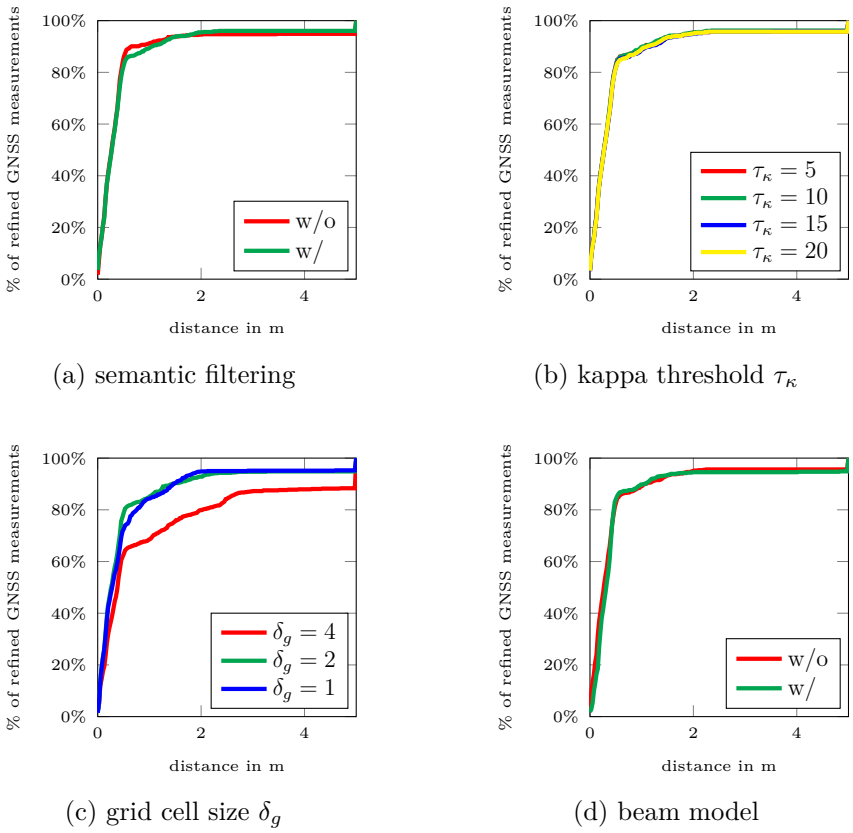
A criterion for the pipeline is the τ_κ. Since κ evaluates the well-constrained local map, we expect a negative correlation between τ_κ and accuracy results.

Fig. 4.5b showcases no significant differences between the different τ_κ values. Upon closer inspection of the data, only 12 scans did not qualify for a τ_κ of 5, which did for 10. The flight is alongside a wall, presumably resulting in well-constrained local maps.

The grid search has two parameters: r_g and δ_g. The accuracy of the GNSS dictates the former. We chose 16 m, although we never observed such coarse results in our data sets. Therefore, we only focus on the latter. We expect δ_g to affect how well the grid search covers all local minima. Fig. 4.5c shows precisely this (RSME=1.266). We settled 2m due reduce computation time (RSME=1.294).

beam model vs. Our Plausibility Score

Fig. 4.5c shows that our method (RMSE=1.179) is slightly better than the beam model (RMSE=1.252). Nevertheless, both perform very well, which we attribute to the beam model being very similar to our approach, despite the approaches to find them being very different. Thrun et al. [53] considered the physical attributes of a LiDAR scanner to conceive the beam model. Meanwhile, we focused on the intersection of the local map and the model.



(a) semantic filtering

(b) kappa threshold τ_κ

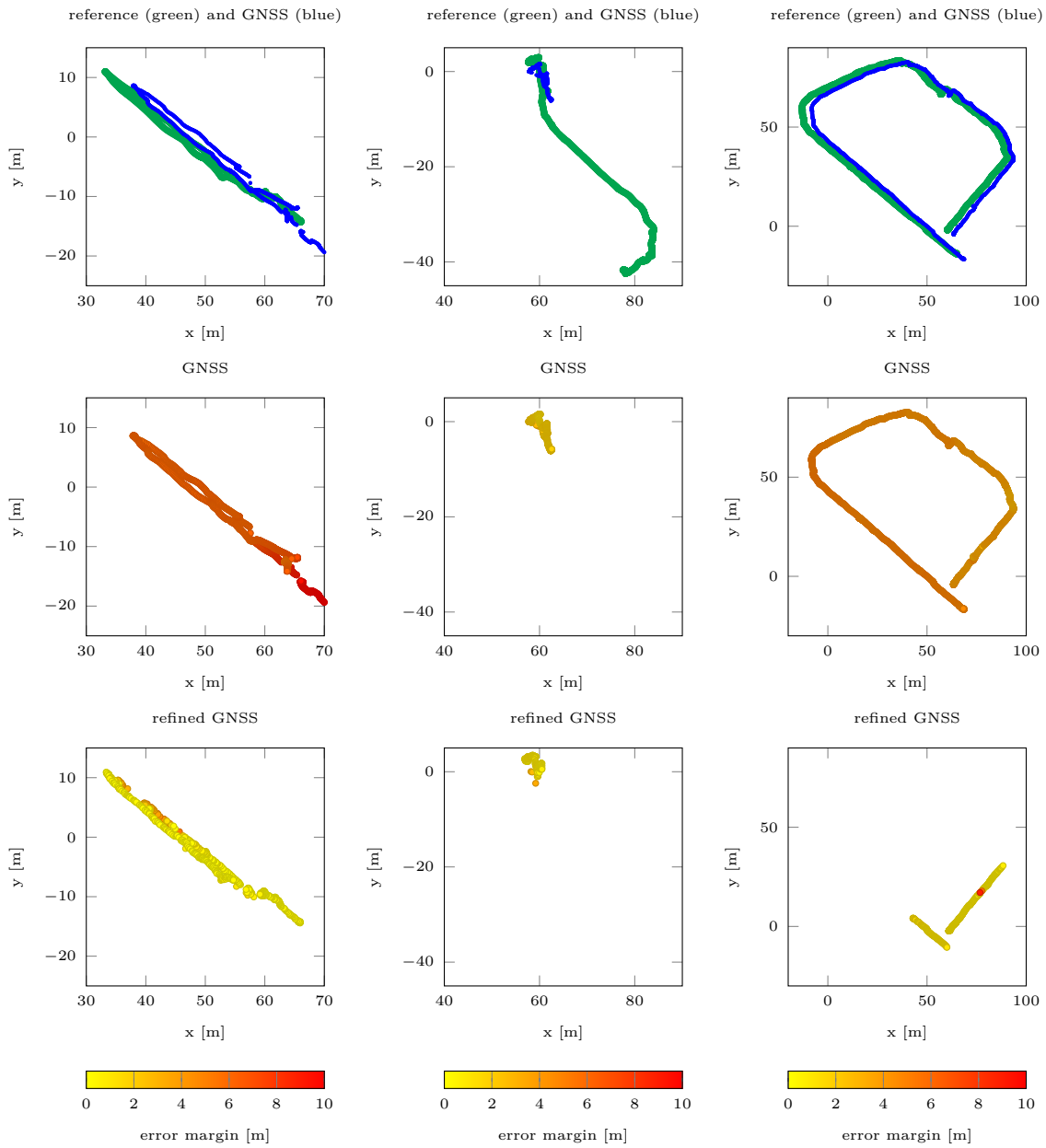
(c) grid cell size δ_g

(d) beam model

Figure 4.5: The graphs showcases the euclidean distance between the reference and our refined GNSS with adjusted parameters.

4.5. Flight Results

In the following section, we will present the results of our refinement evaluation. Fig. 4.6 and Fig. 4.7 show the six flight refinement results. We expect to observe three main trends: significant improvements in accuracy around buildings, decreases in accuracy when clutter is not filtered correctly, and extreme decreases in accuracy when indoor environments are included in the flight data.



a) POP 16-44 b) POP 14-18 c) POP 16-38

Figure 4.6: The figs. show our test flights from Poppelsdorf (POP) campus measured GNSS (blue), reference (green), and our results colored based on euclidean distance to reference.

Poppelsdorf Campus (POP)

We will now go through the flights taken on the Poppelsdorf campus site.

The first flight, ”16-44”, is the one we used for hyperparameter estimation. The

GNSS estimates have an offset of five to ten meters. This showcases our method's robustness as it does not consider the distance to the raw position in score calculation. Therefore, the results significantly improve, although some measurements are slightly off. These errors are likely caused by a large dirt pile misclassified as a wall (Fig. B.2).

The second flight, "14-18", showcases notable increases in accuracy to about half a meter. However, the GNSS measurements do not cover the first part of the flight. This, however, highlights our method's single-shot characteristic, which allows for accurate results, despite no initial positioning.

The third flight, "16-38", shows excellent results. Nevertheless, it also captures the downside of our approach, a reliance on up-to-date models and the inherent assumption of registration results that the model and local map contain the same information. In this case, the rest of the flight contains container buildings not represented in the model map and open terrain where no relation could be established. The effects of the container building can be seen in Fig. B.2c.

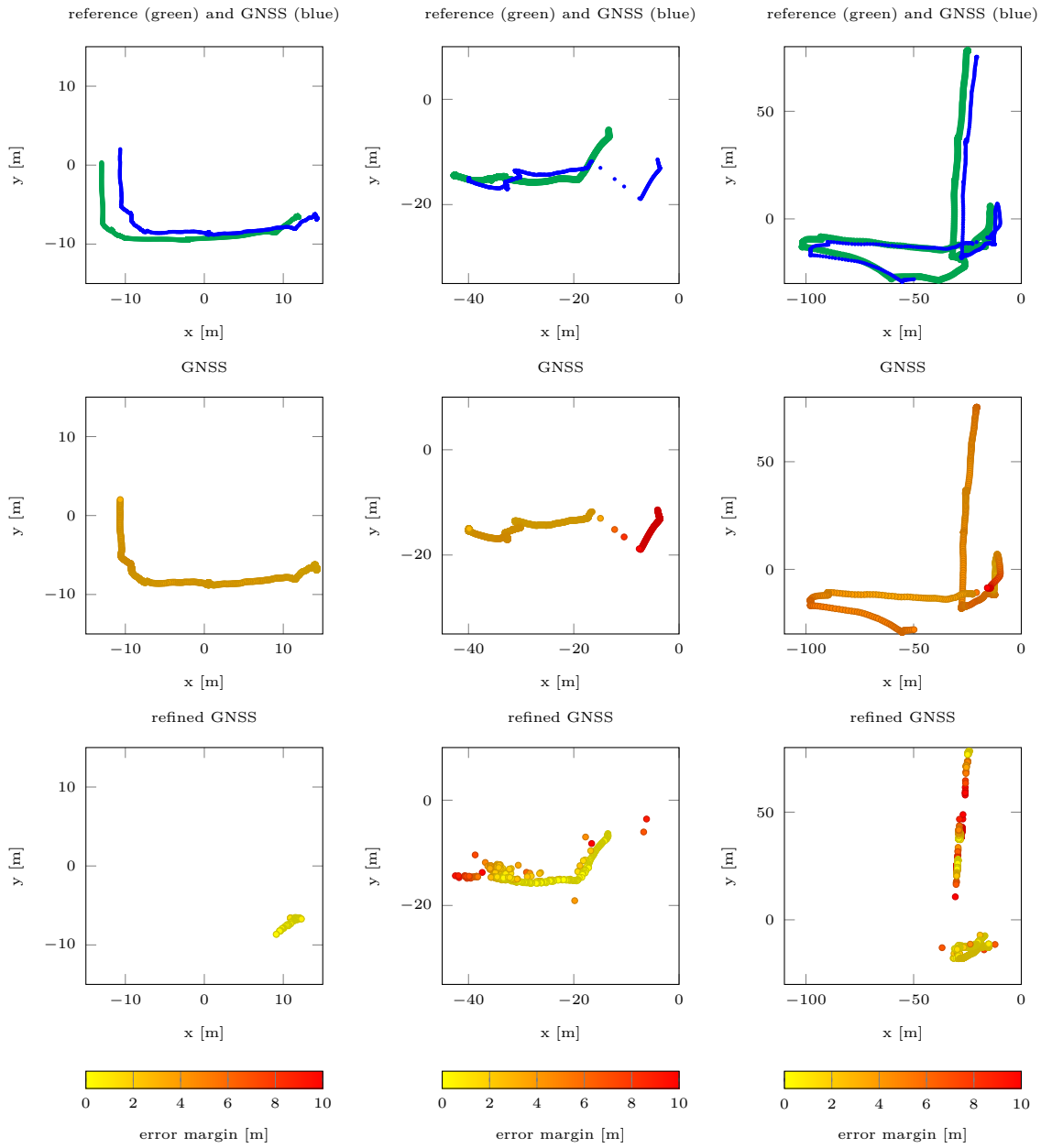
German Rescue Robotics Center (DRZ)

We will now go through the flights taken on the DRZ site. It is noteworthy that this site had a lot of clutter elements, such as trees, cars, and a large gearwheel (Fig. 4.8). In addition, most flights ended in the main hall, i.e., interiors that are not covered by the model.

The first flight, "18-33", is remarkable for the number of discrepancies between the model and reality. The LiDAR scanned a truck, trees, a large gearwheel, and a hut adjacent to the main hall, and they are all not represented in the model. In addition, the flight finishes inside the large hall, and the result is that only the first meters were able to pass our different thresholds. We also notice a quality drop in our semantics, e.g., a small cabin is wrongly classified as a foreground object.

The first section of the "18-34" flight is unavailable as it mostly contains the interiors of the main hall not covered in the model map. The latter portion, however, is quite well-constrained and yields good results.

The third flight, "18-42", is by far the longest. The initial results are promising. However, accuracy decreases when the local map covers only two parallel walls, and the interior section produces substantial errors. In addition, a large portion of the flight has only one wall resulting in largely poor constrained local maps.



(a) DRZ 18-33 (b) DRZ 18-34 (c) DRZ 18-42

Figure 4.7: The figs. show our test flights from Poppelsdorf (POP) campus measured GNSS (blue), reference (green), and our results colored based on euclidean distance to reference.

Figure 4.8: Clutter not represented in model include trees, cars (blue), a large gearwheel (red), and the hut (yellow), taken from [23]

4.5.1. Runtime Analysis

Above, we present the improvements our method can offer and will now analyze the runtime results for our pipeline and its components. We conducted the tests on a computer with a 3.6 GHz Intel i9-9900K CPU with 16 cores and 64 GiB of RAM. We will show the results with and without parallelization, and the parallelization uses 10 threads. The test were conducted on the first 500 scans of the ”16-38” flight. The mean results can be seen in table 4.3. Registration covers a single registration of a local map against the model. The registration loop are 197 that are required for every grid search. The plausibility check includes the calculation of the criteria. The plausibility check loop covers all 197 calculations. The ”complete pipeline” encompasses building the local map, registering it at different poses and evaluating the plausibility.

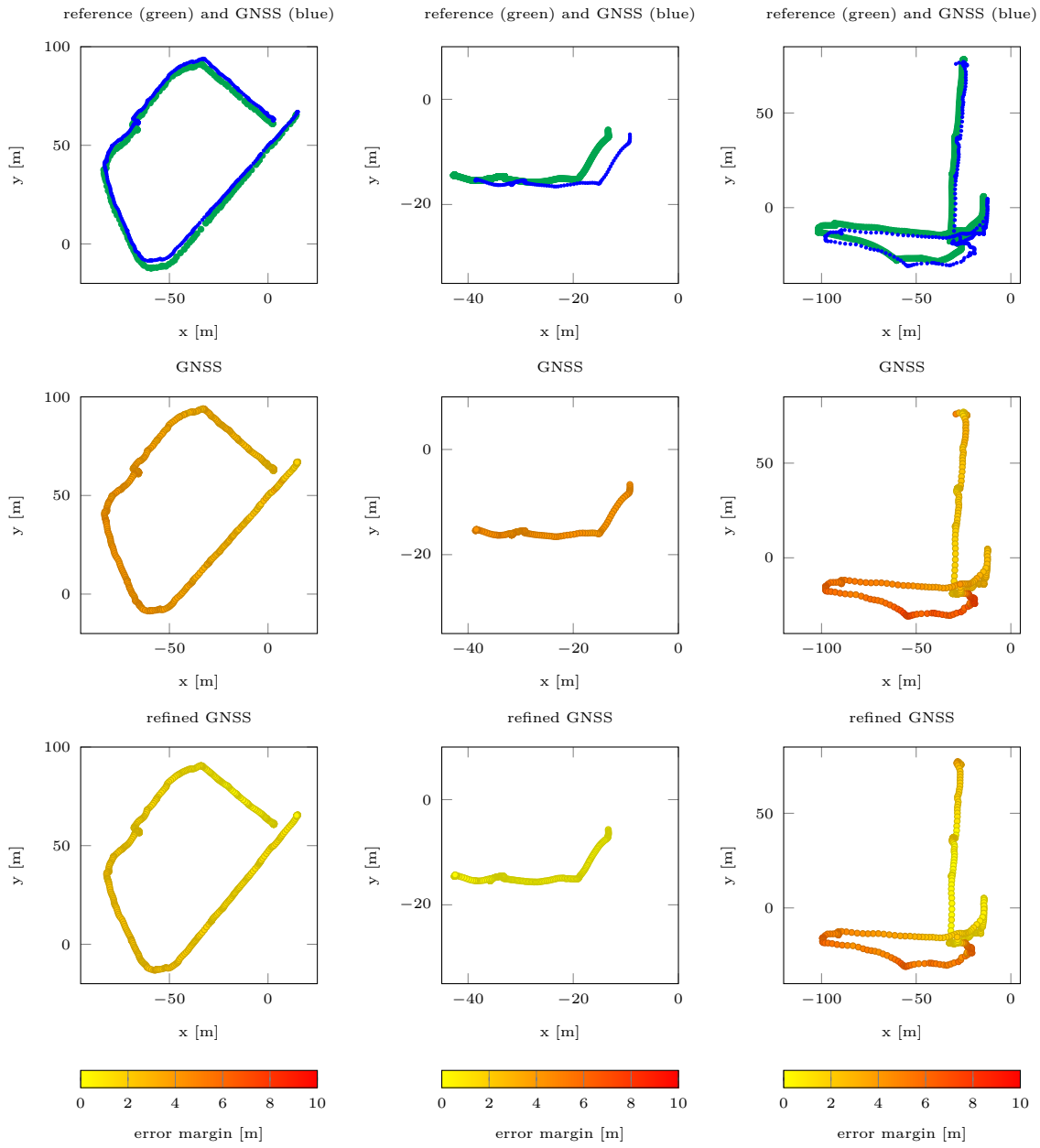
The results indicate that running our method online will be possible, but only occasionally.

name	single core [sec] (1x threads)	multi core [sec] (10x threads)
registration	0.019	–
registration loop	3.744	0,795
plausibility check	0.121	–
plausibility check loop	13.010	2,234
complete pipeline	17,133	3,416

Table 4.3: Runtime analysis with and without parallelization

4.6. Optimization Improvement Evaluation

In the final evaluation, we used a continuous-time spline-based graph optimization implementation by Imbusch.



(a) POP 16-38 (b) DRZ 18-34 (c) DRZ 18-42

Figure 4.9: The figs. show our test flights from Poppelsdorf (POP) campus measured GNSS (blue), reference (green), and our results colored based on euclidean distance to reference.

4.6.1. Implementation

The pose graph optimization utilizes IMU, odometry, and raw GNSS. It weights the inputs according to their covariance. We replaced the raw GNSS with our refined GNSS and a covariance Cov_{GNSS} derived from the product of the registration covariance matrix $\boldsymbol{Cov_r}$ and the plausibility score value $S_{local\,map}$:

$$Cov_{GNSS} = \left\{ \boldsymbol{M} = \boldsymbol{Cov_r} \cdot S_{local\,map} | \boldsymbol{Cov_r} \in [0,1]^{6\times6}, S_{local\,map} \in [0,1] \right\}. \quad (4.1)$$

4.6.2. Optimization Results

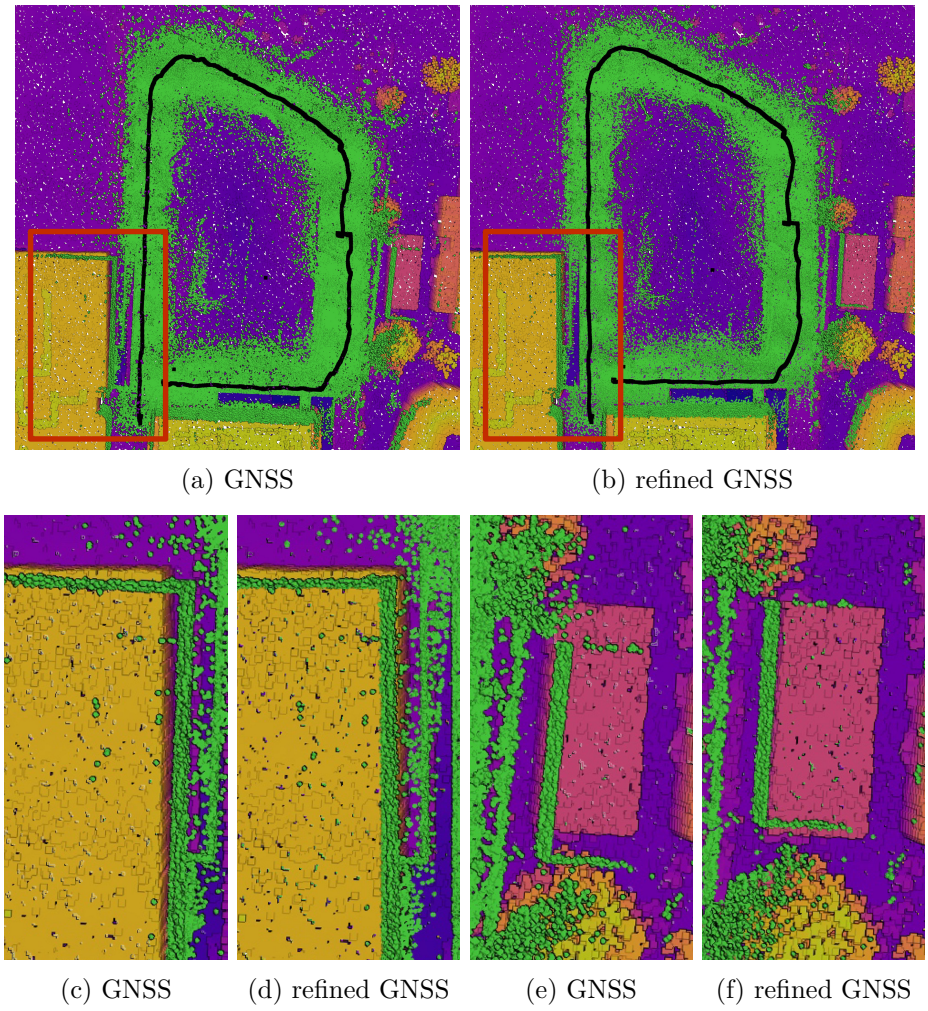
For the result analysis, consider that the local odometry is predominantly a result of the odometry and IMU. Meanwhile, the placement of the path in the global context is the result of either GNSS or our refined GNSS.

Regarding the results, one can see significantly better accuracy due to our method. The error distance falls below one meter. We can see, however, that in the "16-38" flight, errors are caused by differences between local odometry between the reference and our graph optimizer result. These errors might fall into the margin of error of our reference. Figs. 4.10 shows the impact our method has on the accuracy of the global spline placement. One can see that considering the model algins the graph significantly better with buildings.

Tab. 4.4 shows the quantitative improvement of our refinement method over GNSS.

Seq.	GNSS	mean [m]	std [m]	min [m]	$25th$ percentiles [m]	$75th$ percentiles [m]	max [m]
POP 16-38	raw	3.079	0.753	1.221	2.493	3.764	4.305
	refined	1.281	0.602	0.031	0.676	1.818	2.928
	ref. + trans.	**0.502**	**0.338**	**0.002**	**0.180**	**0.802**	**1.172**
DRZ 18-34	raw	4.117	0.125	3.837	4.028	4.203	4.465
	refined	**0.142**	**0.059**	**0.016**	**0.104**	**0.182**	**0.332**
DRZ 18-42	raw	3.374	1.749	1.189	2.011	5.002	7.699
	refined	2.307	1.850	0.124	1.002	4.194	**6.280**
	ref. + trans.	**1.799**	**1.448**	**0.0120**	**0.747**	**2.567**	6.881

Table 4.4: Results of the optimizer with raw and refined GNSS. The values are the mean, std, min, max of the euclidean error distance between the optimizer results and the reference. Our goal is to minimize the error. All data is presented in meters.



(a) GNSS (b) refined GNSS

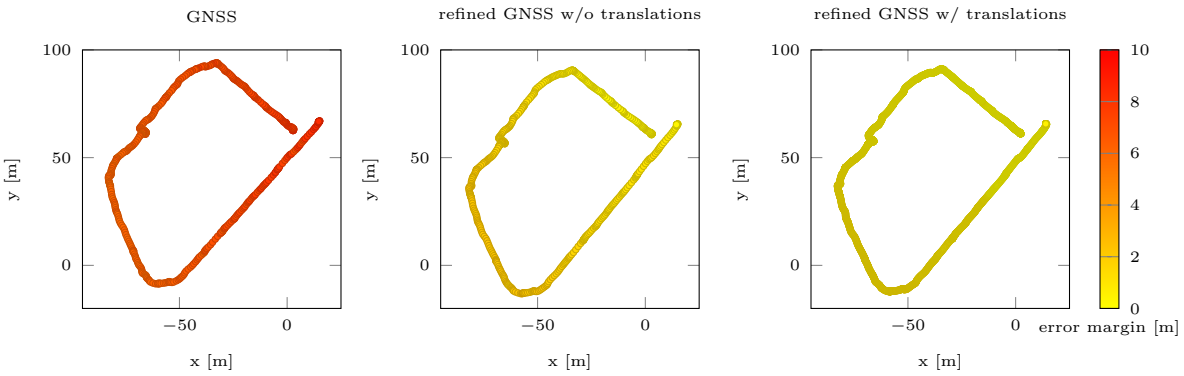
(c) GNSS (d) refined GNSS (e) GNSS (f) refined GNSS

Figure 4.10: Result of the graph optimizer with GNSS and our refined GNSS as input. The flight path is indicated in black. The aggregated LiDAR point cloud (green) is overlayed with a height-colored ALS model. The significant difference is highlighted in red. (a) shows optimizer result with GNSS, that do not align with buildings. (b) shows Optimizer result with our refined GNSS. Since the model was considered, our global placement aligns significantly better with the buildings. (c) and (d) are a close-up of the lecture hall with raw and refined GNSS. (e) and (f) showcase a close-up the container building with raw and refined GNSS.

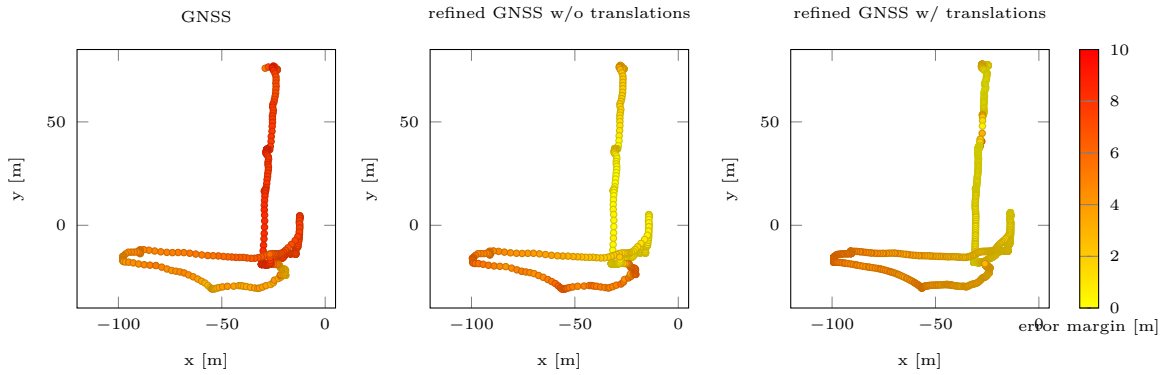
4.6.3. Relative Transformation Optimization

For the final evaluations we included our relative transformation introduces in Sec. 3.4.1. We evaluated the two longest flights ”18-42” and ”16-36”, as they are most affected by drift.

Regarding the results, one can see more significant improvements in Tab. 4.4. We consider ”16-38” improvement especially relevant as large middle sections have been improved. For ”18-42” we can see a mean improvement as well, but outliers are placed incorrectly, again these errors might fall into the margin of error of our reference. Fig. 4.11 shows the significant improvement over raw GNSS.



(a) POP 16-38



(b) DRZ 18-42

Figure 4.11: The Figs. showcases all three tested inputs for the optimizer on ”18-42” and ”16-38”, the results are colored based on euclidean distance to reference.

5. Conclusion

This thesis refines coarse GNSS estimates in urban environments utilizing georeferenced CityGML and DEM models and use these results to improve the global positioning of a pose graph.

First, we presented our GNSS refinement pipeline, combining LiDAR point clouds, semantics, and odometry data to generate semantic-filtered local maps. Second, we used a grid search to place a local map in multiple positions around the GNSS estimate and determined the most plausible placement. Lastly, we evaluated the plausibility with our intuitive plausibility score and fed the result into a graph optimizer. We also tested our results thoroughly.

We achieved significant GNSS refinement in the urban environment. The improvement was significant when registration conditions were forthcoming, i.e., walls that constrain the map in multiple axis. On the one hand, poorly constrained local maps tended to slide along walls resulting in one-directional errors and were filtered out. On the other hand our method assumption of constrains means that plains as well as single walls do not yield results. The semantic filter proofed useful, however the limited amount of categories resulted in occasional worse registration, e.g., we could not distinguish between grass and tree leaves as they are both labeled as "vegetation", resulting in false removal.

Our plausibility score achieved only slightly better results than the one proposed by Thrun et al. [53].

We were able to supply the pose graph optimizer with more accurate global poses, allowing it to become more accurate than with regular GNSS results.

The geo-refrenced data can now be used for many high level applications, such as visualizing the flight path on a map to drone operators. In addition, loop closures can be detected to increase trajectory accuracy, as was achieved with our relative transformations. Possible applications in tandem with the model entail: change detection, e.g. the construction of a new building could be detected. Detailed LiDAR scans may also be used to refine the model. Model addition or substation are also possible to reflect changes, e.g., constitution or demolition of buildings.

For future work, in regards to semantics, we suggest implementing more categories or adjust certain categories, e.g., move grass to the floor category from the vegetation category. We will also implement and test the prospect of relative

transformation as explained in our method chapter. The less supervised Aerial Laser scans (ALS) may also be used. As well as other registration approaches that support single-direction constraints. Real-time deployment may also be tested.

Our implementation showcases that CityGML models and DEM can be used to refine GNSS and enhance results of pose graph optimizer. Especially the latter might proof useful for applications such as positioning in global maps, allowing more reliable flight path estimations for maneuvers with drone swarm and ease of use for pilots.

A. Tables

Format name	Code for a triangle face

STL	`facet normal 0.0 0.0 1.0`
	`  outer loop`
	`  vertex 0.0 0.0 0.0`
	`  vertex 1.0 0.0 0.0`
	`  vertex 0.0 1.0 0.0`
	`  endloop`
	`endfacet`

OBJ	`v 0.0 0.0 0.0`
	`v 0.0 1.0 0.0`
	`v 1.0 0.0 0.0`
	`f 1 2 3`

PLY	`ply format ascii 1.0`
	`element vertex 3`
	`property float x`
	`property float y`
	`property float z`
	`element face 1`
	`property list uchar int vertex_indices`
	`end_header`
	`0.0 0.0 0.0`
	`0.0 1.0 0.0`
	`1.0 0.0 0.0`
	`3 0 1 2`

Table A.1: Example code for defining a triangle face in STL, OBJ, and PLY formats.

Hyperparameter and values	RMSE
$N_w = 1$, $d_w = 0.0$m	2.303
$N_w = 5$, $d_w = 0.0$m	2.004
$N_w = 10$, $d_w = 0.0$m	1.691
$N_w = 20$, $d_w = 0.0$m	1.379
$N_w = 40$, $d_w = 0.0$m	1.699
$N_w = 1$, $d_w = 0.5$m	2.523
$N_w = 5$, $d_w = 0.5$m	1.722
$N_w = 10$, $d_w = 0.5$m	1.385
$N_w = 20$, $d_w = 0.5$m	1.179
$N_w = 40$, $d_w = 0.5$m	1.434
$N_w = 1$, $d_w = 1.0$m	2.676
$N_w = 5$, $d_w = 1.0$m	1.720
$N_w = 10$, $d_w = 1.0$m	1.408
$N_w = 20$, $d_w = 1.0$m	1.349
$N_w = 40$, $d_w = 1.0$m	1.371
$N_w = 1$, $d_w = 2.0$m	2.379
$N_w = 5$, $d_w = 2.0$m	1.680
$N_w = 10$, $d_w = 2.0$m	1.296
$N_w = 20$, $d_w = 2.0$m	1.377
$\tau_\kappa = 5$	1.153
$\tau_\kappa = 10$	1.122
$\tau_\kappa = 15$	1.162
$\tau_\kappa = 20$	1.168
w/o *semantics*	1.216
w/ *semantics*	1.134
$M_{res} = 4$, $M_{lvl} = 3$	1.207
$M_{res} = 2$, $M_{lvl} = 3$	1.157
$M_{res} = 1$, $M_{lvl} = 3$	missing
$M_{res} = 4$, $M_{lvl} = 2$	0.885
$M_{res} = 2$, $M_{lvl} = 2$	1.238
$M_{res} = 1$, $M_{lvl} = 2$	1.141
$M_{res} = 2$, $M_{lvl} = 1$	0.898
$M_{res} = 1$, $M_{lvl} = 1$	1.510
$M_{res} = 0.5$, $M_{lvl} = 1$	1.424
$\delta_g = 4$	1.979
$\delta_g = 2$	1.294
$\delta_g = 1$	1.266
w/o beam model	1.179
w/ beam model	1.252

Table A.2: Root-mean-square deviation for the different hyper parameters.

B. Figures

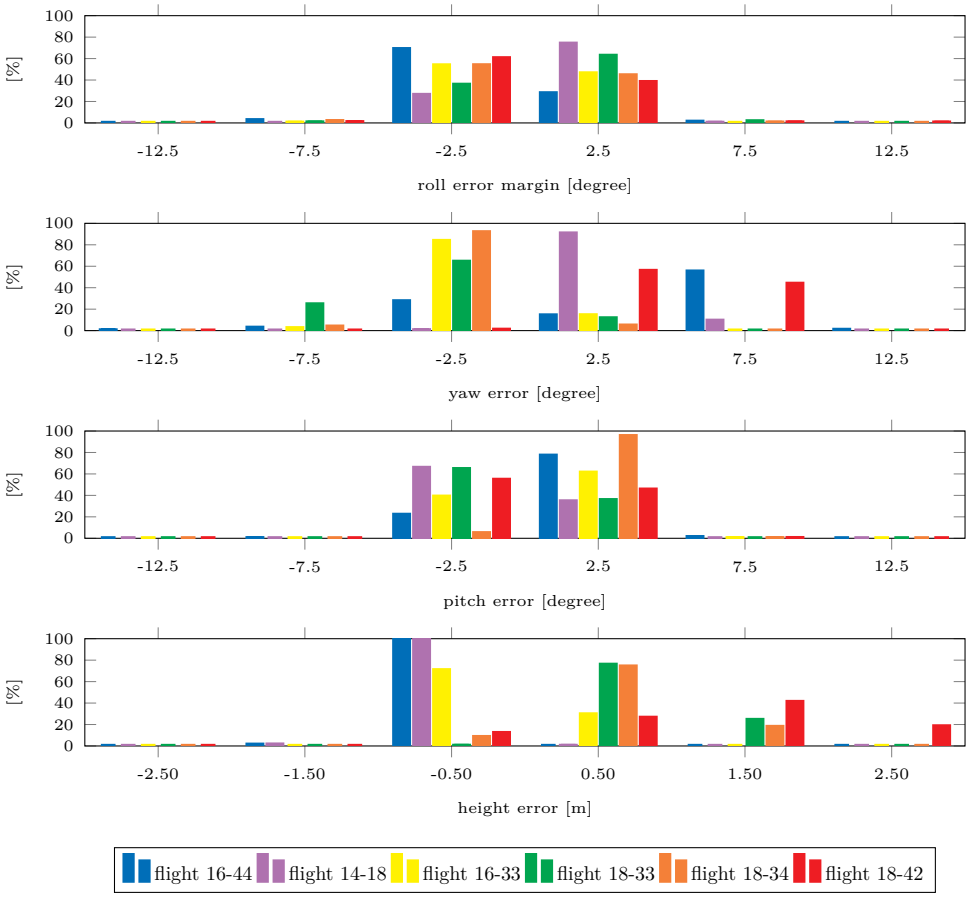


Figure B.1: The plots show the error margins that where measured between the reference and our input. The data is collected from our test flights.

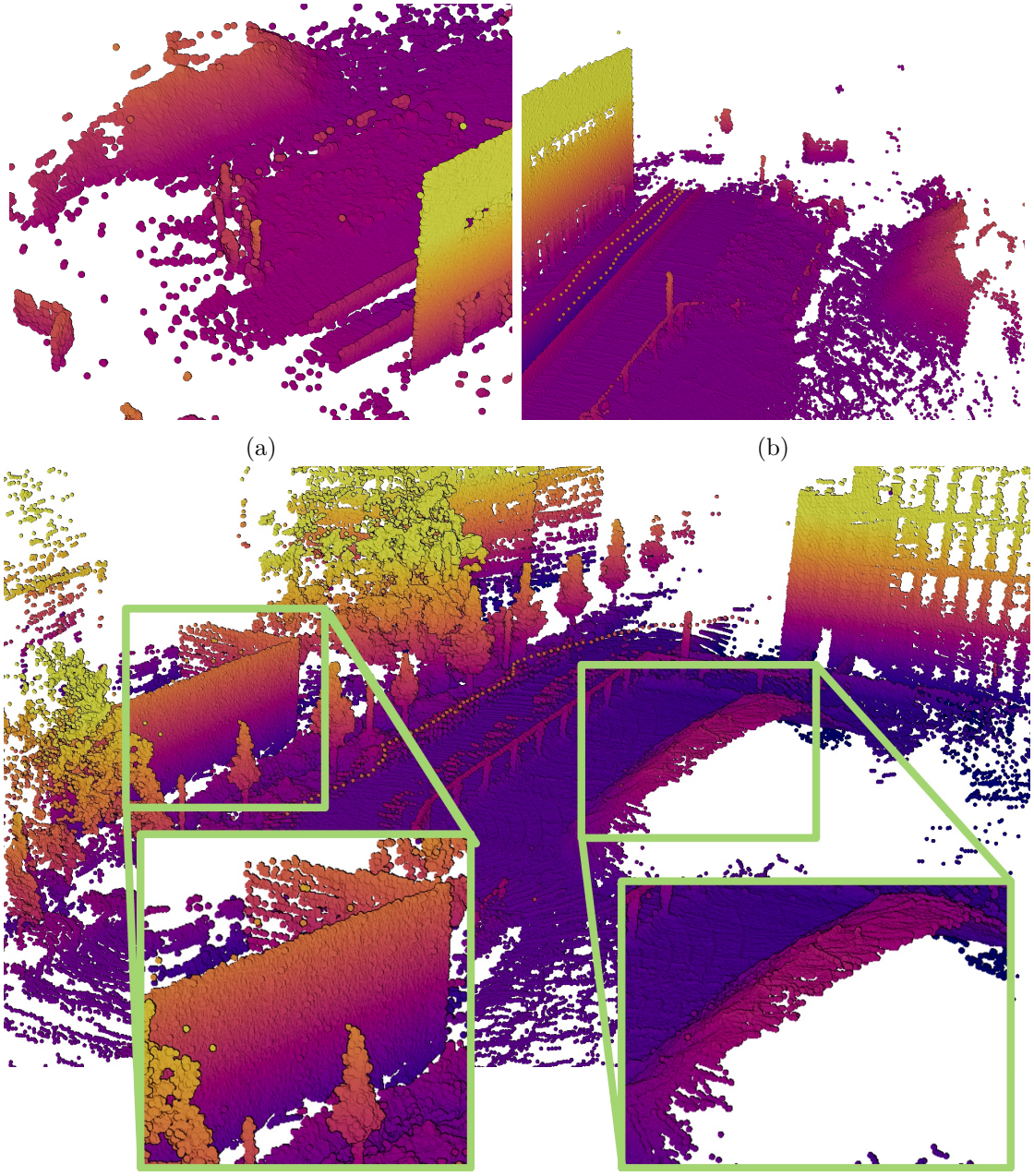


Figure B.2: (a) and (b) showcase a dirt pile on ”16-44” that is only on the recorded data and not the CityGML model. This results in incorrect matching. (c) showcases another dirt pile (right) on ”16-38”. The LiDAR data also includes a container (left) that was not represented in the model, therefore the registration may align the container wall with a different building. Those result are being discarded.

List of Figures

List of Tables

Bibliography

[1] Moulay A. Akhloufi, Andy Couturier, and Nicolás A. Castro. "Unmanned aerial vehicles for wildland fires: sensing, perception, cooperation and assistance." In: *Drones* 5.1 (2021), p. 15.

[2] Saeed Hamood Alsamhi, Alexey V. Shvetsov, Santosh Kumar, Svetlana V. Shvetsova, Mohammed A. Alhartomi, Ammar Hawbani, Navin Singh Rajput, Sumit Srivastava, Abdu Saif, and Vincent Omollo Nyangaresi. "UAV Computing-Assisted Search and Rescue Mission Framework for Disaster and Harsh Environment Mitigation." In: *Drones* 6.7 (2022), p. 154.

[3] unknown author. *CityJSON/io*. URL: `https://github.com/cityjson/cjio`.

[4] 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.

[5] Marius Beul, Max Schwarz, Jan Quenzel, Malte Splietker, Simon Bultmann, Daniel Schleich, Andre Rochow, Dmytro Pavlichenko, Radu Alexandru Rosu, Patrick Lowin, Bruno Scheider, Michael Schreiber, Finn Süberkrüb, and Sven Behnke. "Target Chase, Wall Building, and Fire Fighting: Autonomous UAVs of team NimbRo at MBZIRC 2020." In: *CoRR* abs/2201.03844 (2022). arXiv: `2201.03844`. URL: `https://arxiv.org/abs/2201.03844`.

[6] Filip Biljecki and Ken Arroyo Ohori. "Automatic Semantic-preserving Conversion Between OBJ and CityGML." In: *Eurographics workshop on urban data modelling and visualisation 2015*. Delft, Netherlands, Nov. 2015, pp. 25–30.

[7] Jack E. Bresenham. "Algorithm for computer control of a digital plotter." In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[8] Simon Bultmann, Jan Quenzel, and Sven Behnke. "Real-time multi-modal semantic fusion on unmanned aerial vehicles." In: *European Conference on Mobile Robots (ECMR)*. IEEE. 2021, pp. 1–8.

[9] Cindy Cappelle, Maan E El Najjar, François Charpillet, and Denis Pomorski. "Virtual 3D city model for navigation in urban areas." In: *Journal of Intelligent & Robotic Systems* 66 (2012), pp. 377–399.

[10] Nived Chebrolu, Philipp Lottes, Thomas Läbe, and Cyrill Stachniss. "Robot localization based on aerial images for precision agriculture tasks in crop fields." In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 1787–1793.

[11] Gordon A. Christie, Garrett Warnell, and Kevin Kochersberger. "Semantics for UGV registration in gps-denied environments." In: *Corr* abs/1609.04794 (2016). arXiv: 1609.04794. URL: http://arxiv.org/abs/1609.04794.

[12] Cloud Compare. *Sample points*. URL: https://www.cloudcompare.org/doc/wiki/index.php/Mesh%5CSample_points.

[13] Anweshan Das and Gijs Dubbelman. "An experimental study on relative and absolute pose graph fusion for vehicle localization." In: *2018 ieee intelligent vehicles symposium (iv)*. IEEE. 2018, pp. 630–635.

[14] Tinne De Laet and Herman Bruyninckx. "A rigorously Bayesian Beam Model and an adaptive full scan model for range finders in dynamic environments." In: *JAIR* 33 (Jan. 2014).

[15] Dieter Fox. "KLD-sampling: Adaptive particle filters." In: *Advances in neural information processing systems* 14 (2001).

[16] Marcus Goetz. "Towards generating highly detailed 3D citygml models from OpenStreetMap." In: *International Journal of Geographical Information Science* 27.5 (2013), pp. 845–865.

[17] Marcus Goetz and Alexander Zipf. "OpenStreetMap in 3D–detailed insights on the current situation in Germany." In: *International Conference on Geographic Information Science*. Vol. 2427. 2012, p. 2427.

[18] Google. *Google Earth*. URL: https://earth.google.com/web/.

[19] U.S. government. URL: www.gps.gov/systems/gps/performance/accuracy. (accessed 31.12.2021).

[20] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. "A tutorial on graph-based SLAM." In: *Intelligent transportation systems magazine* 2.4 (2010), pp. 31–43.

[21] Gerhard Gröger, Thomas H. Kolbe, Claus Nagel, and Karl-Heinz Häfele. *City geography markup language (citygml) encoding standard*. OGC, 2012.

[22] Mordechai Haklay and Patrick Weber. "Openstreetmap: User-generated street maps." In: *IEEE Pervasive computing* 7.4 (2008), pp. 12–18.

[23] Google inc. *Google maps*. URL: https://www.google.com/maps/place/DRZ+-+Deutsches+Rettungsrobotik+Zentrum/@51.5484306,7.3734607,144m/data=!3m1!1e3!4m6!3m5!1s0x47b91bd27abcffc1:0xad65bc0cfede6b73!8m2!3d51.5487916!4d7.3738784!16s%2Fg%2F11mv5csdgw.

[24] Engineering Council of India and pwc. *Flying high: Drones drive jobs in the construction sector*. 2018.

[25] Masiri Kaamin, Siti Nooraiin Mohd Razali, Nor Farah Atiqah Ahmad, Saifullizan Mohd Bukari, Norhayati Ngadiman, Aslila Abd Kadir, and Nor Baizura Hamid. "The application of micro UAV in construction project." In: *AIP Conference Proceedings*. Vol. 1891. 1. AIP Publishing LLC. 2017, p. 020070.

[26] Bezirksregierung Köln. *3D-Gebäudemodelle*. URL: `www.bezreg-koeln.nrw.de/brk_internet/geobasis/3d_gebaeudemodelle/index.html`.

[27] Bezirksregierung Köln. *Höhenmodelle*. URL: `www.bezreg-koeln.nrw.de/brk_internet/geobasis/hoehenmodelle/index.html`.

[28] Ivana Kruijff-Korbayová, Robert Grafe, Nils Heidemann, Alexander Berrang, Cai Hussung, Christian Willms, Peter Fettke, Marius Beul, Jan Quenzel, Daniel Schleich, et al. "German Rescue Robotics Center (DRZ): A Holistic Approach for Robotic Systems Assisting in Emergency Response." In: *International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2021, pp. 138–145.

[29] Robert Kulawik, Arne Schilling, and Alexander Zipf. "Landesweite 3D-Stadtmodelle im Internet auf Basis offener Standards des Open Geospatial Consortiums (OGC)-das Beispiel Nordrhein-Westfalen 3D." In: *Real corp* (2009), pp. 293–302.

[30] Lukas Lucks, Lasse Klingbeil, Lutz Plümer, and Youness Dehbi. "Improving trajectory estimation using 3D city models and kinematic point clouds." In: *Transactions in GIS* 25.1 (2021), pp. 238–260.

[31] Martin Magnusson, Achim Lilienthal, and Tom Duckett. "Scan registration for autonomous mining vehicles using 3D-NDT." In: *Journal of Field Robotics* 24.10 (2007), pp. 803–827.

[32] Andy Maloney and Daniel Girardeau-Montaut. *Mesh Sampling Tool*. URL: `https://github.com/CloudCompare/CCCoreLib/blob/50511c47792693d6f53b4592d261a2684src/MeshSamplingTools.cpp#L387`.

[33] Open Street Map. URL: `www.osmbuildings.org/?lat=52.52394&lon=13.41578&zoom=16.8&rotation=314&tilt=23`. (accessed 9.3.2022).

[34] Kenton McHenry and Peter Bajcsy. "An overview of 3D data content, file formats and viewers." In: *National Center for Supercomputing Applications* 1205 (2008), p. 22.

[35] C. A. Mücher, S. Los, G. J. Franke, and C. Kamphuis. "Detection, identification and posture recognition of cattle with satellites, aerial photography and UAVs using deep learning techniques." In: *International Journal of Remote Sensing* (2022), pp. 1–16.

[36] Geobasis NRW. URL: www.geoportal.nrw/opendatadownloadclient. (accessed 8.7.2022).

[37] National Oceaning and Atmospheric Administration. *Universal Transverse Mercator Coordinates*. URL: www.geodesy.noaa.gov/TOOLS/utm.html.

[38] Open Geospacial Consortium. *City GML*. URL: https://www.ogc.org/standard/citygml/.

[39] Jan Quenzel and Sven Behnke. "Real-time multi-adaptive-resolution-surfel 6D LiDAR odometry using continuous-time trajectory optimization." In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5499–5506.

[40] Milad Ramezani, Yiduo Wang, Marco Camurri, David Wisth, Matias Mattamala, and Maurice Fallon. "The newer college dataset: Handheld lidar, inertial and vision with ground truth." In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4353–4360.

[41] DRZ - Deutsches Rettungsrobotik-Zentrum. *DRZ Image-Teaser*. Youtube. 2020. URL: https://www.youtube.com/watch?v=anZtJAajQy0,. (accessed 12.1.2022).

[42] Open Robotics. *AMCL Overview*. 2022. URL: http://wiki.ros.org/amcl.

[43] Open Robotics. *PCL Overview*. 2022. URL: http://wiki.ros.org/pcl.

[44] Radu Alexandru Rosu, Jan Quenzel, and Sven Behnke. "Reconstruction of textured meshes for fire and heat source detection." In: *International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2019, pp. 235–242.

[45] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm." In: *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE. 2001, pp. 145–152.

[46] Dimitrios Sainidis, Dimitrios Tsiakmakis, Konstantinos Konstantoudakis, Georgios Albanis, Anastasios Dimou, and Petros Daras. "Single-handed gesture UAV control and video feed AR visualization for first responders." In: *International Conference on Information Systems for Crisis Response and Management (ISCRAM)*. 2021, pp. 23–26.

[47] Daniel Schleich, Marius Beul, Jan Quenzel, and Sven Behnke. "Autonomous flight in unknown GNSS-denied environments for disaster examination." In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2021, pp. 950–957.

[48] Akshay Shetty and Grace Xingxin Gao. "Covariance estimation for GPS-lidar sensor fusion for UAVs." In: *International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2017)*. 2017, pp. 2919–2923.

[49] Christiane Sommer, Vladyslav Usenko, David Schubert, Nikolaus Demmel, and Daniel Cremers. "Efficient derivative computation for cumulative b-splines on lie groups." In: *Conference on Computer Vision and Pattern Recognition*. IEEE. 2020, pp. 11148–11156.

[50] Vojtech Spurny, Vaclav Pritzl, Viktor Walter, Matej Petrlik, Tomas Baca, Petr Stepan, David Zaitlik, and Martin Saska. "Autonomous firefighting inside buildings by an unmanned aerial vehicle." In: *IEEE Access* 9 (2021), pp. 15872–15890.

[51] Márta Szilvśi-Nagy and G. Y. Matyasi. "Analysis of stl files." In: *Mathematical and Computer Modelling* 38.7-9 (2003), pp. 945–960.

[52] Tim Y. Tang, Daniele De Martini, and Paul Newman. "Get to the point: Learning LiDAR place recognition and metric localisation using overhead imagery." In: *Robotics: science and systems, 2021* (2021).

[53] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005. ISBN: 0262201623 9780262201629.

[54] William P. Thurston. *Three-dimensional geometry and topology, volume 1*. Ed. by Silvio Levy. Princeton Mathematical Series. Princeton University Press, 1997.

[55] Alan O. G. Torres. "Registering and visualizing point cloud data with existing 3D CityGML Models." MA thesis. Hochschule Bonn-Rhein-Sieg, 2021, p. 37.

[56] Frank Van Diggelen and Per Enge. "The world's first gps mooc and worldwide laboratory using smartphones." In: *international technical meeting of the satellite division of the institute of navigation (ION GNSS+)*. 2015, pp. 361–369.

[57] Carlos Viegas, Babak Chehreh, José Andrade, and João Lourenço. "Tethered UAV with combined multi-rotor and water jet propulsion for forest fire fighting." In: *Journal of Intelligent & Robotic Systems* 104.2 (2022), pp. 1–13.

[58] Junjie Wan, Lijun Qi, Hao Zhang, Jiarui Zhou, et al. "Research status and development trend of UAV broadcast sowing technology in China." In: *ASABE Annual International Virtual Meeting*. American Society of Agricultural and Biological Engineers. 2021, p. 1.

[59] Chen Wang, Yutong Tang, Mukhtar A. Kassem, and Zhenquan Chen. "UAV Application for Typhoon Damage Assessment in Construction Sites." In: *Applied Sciences* 12.13 (2022), p. 6293.

[60] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry." In: *Transactions on Robotics* 38.4 (2022), pp. 2053–2073.

[61] Huan Yin, Xuecheng Xu, Sha Lu, Xieyuanli Chen, Rong Xiong, Shaojie Shen, Cyrill Stachniss, and Yue Wang. "A Survey on Global LiDAR Localization." In: *Arxiv preprint arxiv:2302.07433* (2023).

[62] Chika Yinka-Banjo and Olasupo Ajayi. "Sky-farmers: Applications of unmanned aerial vehicles (UAV) in agriculture." In: *Autonomous Vehicles* (2019), pp. 107–128.