Marco Stranner, BSc

Subsurface Infrastructure Localization for GIS Data Alignment using Semantic Segmentation

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme

Computer Science

submitted to

Graz University of Technology

Supervisor

Dipl.-Ing. Dr.techn. Clemens Arth

Institute of Computer Graphics and Vision

Advisor

Dipl.-Ing. Philipp Fleck

Institute of Computer Graphics and Vision

Graz, Austria, January 2023

In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.

Douglas Adams

Abstract

The use of augmented reality in the outdoor sector is finding more and more practical applications with better sensor technology on mobile devices. In this work, we optimize our previous setup for tracking in outdoor augmented reality (AR) applications within the field of subsurface utility engineering (SUE). We address limitations in north estimation by compass and the semi-automatic initialization of tracking fusion that are perceived as impractical for users. An algorithm for alternative GPS based north estimation is presented. Additionally, the setup is enhanced with a continuous drift correction to automate tracking completely.

Despite the highly accurate tracking, the quality of data being worked with remains a challenge. GIS data documentation is often of poor quality due to a lack of regulation. To address this issue, we propose a solution for segmenting pipes in an open excavation site and aligning available displaced GIS models. Our approach is based on segmentation in image space and reprojection of these results into 3D space, allowing for the estimation of center points along the pipe.

Kurzfassung

Der Einsatz von Augmented Reality im Outdoor-Bereich findet mit besserer Sensortechnologie auf mobilen Geräten immer mehr praktische Anwendungen. In dieser Arbeit optimieren wir unser bisheriges Setup für das Tracking in AR-Anwendungen im Freien auf dem Gebiet der Untergrundversorgungstechnik (Subsurface Utility Engineering, SUE). Wir gehen auf die Schwächen bei der Nordschätzung mittels Kompass und der halbautomatischen Initialisierung der Tracking-Fusion, die von den Benutzern als unpraktisch empfunden werden, ein. Es wird ein Algorithmus für eine alternative GPS-basierte Nordschätzung vorgestellt. Zusätzlich wird das System um eine kontinuierliche Driftkorrektur erweitert, um das Tracking vollständig zu automatisieren.

Trotz sehr genauen Trackings bleibt die Qualität der Daten, mit denen gearbeitet wird, die größte Herausforderung. Die Dokumentation der GIS-Daten ist aufgrund mangelnder Vorschriften oft von schlechter Qualität. Um dieses Problem anzugehen, schlagen wir eine Lösung für die Segmentierung von Rohren in einer offenen Ausgrabungsstätte und den Abgleich verfügbarer, verschobener GIS-Modelle vor. Unser Ansatz basiert auf der Segmentierung im Bildraum und der Reprojektion dieser Ergebnisse in den 3D-Raum, was die Schätzung von Mittelpunkten entlang der Rohre ermöglicht.

Acknowledgments

I would like to express my deepest gratitude to Dipl.-Ing. Dr.techn. Clemens Arth for his invaluable guidance and supervision throughout my thesis. His mentorship, support and guidance throughout the years has been instrumental in my development and growth as a researcher in the field of computer vision, graphics, and augmented reality.

I would also like to extend my appreciation to Dipl.-Ing. Phillip Fleck for being my advisor for my master thesis. His invaluable contributions, ideas, and support have helped me to achieve my goals and learn a lot in the field.

I would also like to thank Ph.D. M.Sc. Lasse Hedegaard Hansen for his technical expertise in Subsurface Utility Engineering, for his availability and support and for providing data to allow for the realization of this work.

I would also like to thank Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg for his support and for the opportunity to work with his team at the Institute of Computer Graphics and Vision, which was a truly inspiring experience.

Lastly, I would like to thank my family and friends for their tireless support and encouragement throughout my studies.

Contents

List of Figures

List of Tables

1 Introduction

The utilization of Augmented Reality (AR) in outdoor applications and commercial products has seen substantial research in recent years. However, one area that has seen moderate to limited advancement is Subsurface Utility Engineering (SUE). In the context of this work, SUE is associated with underground infrastructures, like pipes and cables, excavation works and visualizations of such structures and corresponding metadata. In 1999, Spohrer *et al.* [5] predicted that AR SUE applications would become available in the near future once hardware capabilities had improved. Despite the current availability of accurate and accessible hardware, commercial applications are still rare today. This may have various reasons, from general interest on this topic to fundamental problems in the sector itself, which will be discussed later on. As shown in Figure 1.1, latest commercial products did not advance significantly beyond what was shown in research projects years ago. Surely, hardware got much handier, while also quality improved, but those examples point out the limited progress over the last decade.

Despite the limited progress in Subsurface Utility Engineering, AR has the potential to play a significant role in this field. As outlined in later sections, excavations can result in significant financial losses due to damage to underground infrastructure. AR applications can help to overcome these challenges by optimizing and automating excavation processes, reducing human error and leading to efficient and cost-effective excavation. Our previous research has presented a mobile setup that combines an external hardware device called CapsLoc with mobile devices such as tablets, mobile phones, or head-mounted displays. This setup enables the implementation of AR applications, such as marking utilities on the surface, as illustrated in Figure 1.2. As previously discussed in our work [4], expert interviews revealed a demand for such applications and demonstrated improvements in current workflows. The implementation of outdoor AR concepts relies on external hardware for precise localization. Localization in the context of outdoor AR applications refers to accurate positioning in the real-world coordinate space. High-accuracy positioning and north estimation are crucial for precise

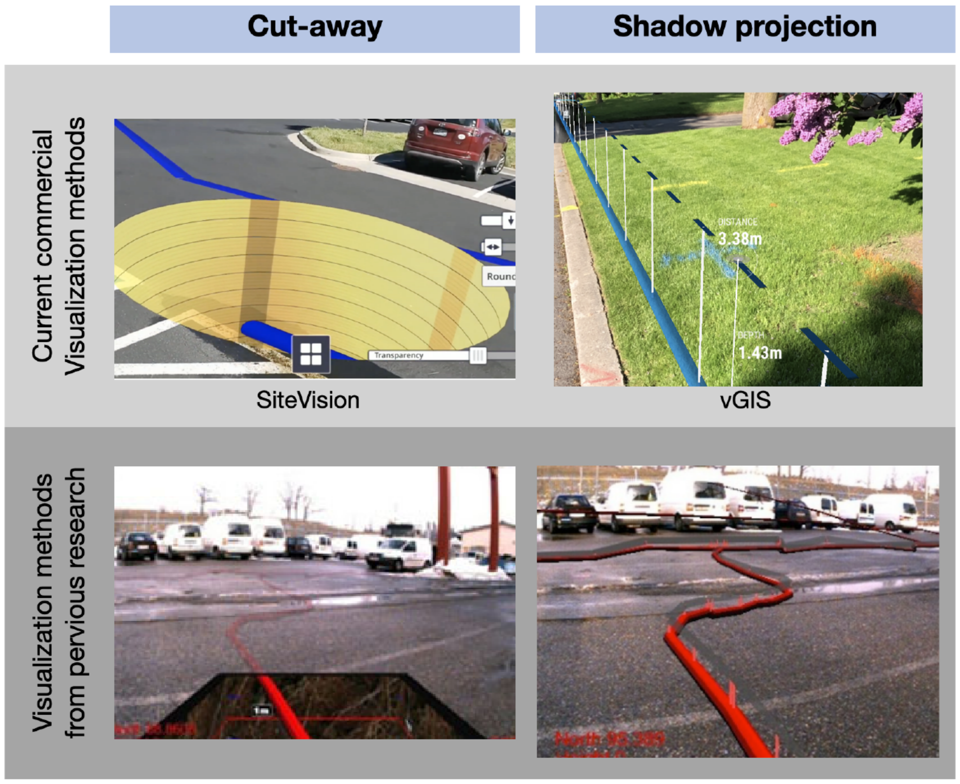

Cut-away
Shadow projection
Current commercial Visualization methods
Visualization methods from pervious research
SiteVision
vGIS
DISTANCE
3.38m
DEPTH
1.43m
Transparency
1m

Figure 1.1: Underground Structure Visualization then vs. now: Visualization of subsurface utilities. The top two images are latest commercial products from SiteVision and vGIS. Bottom images show research approaches 10 years ago [1, 2] *Comparison between visualization methods for underground infrastructure* by Hansen [3]

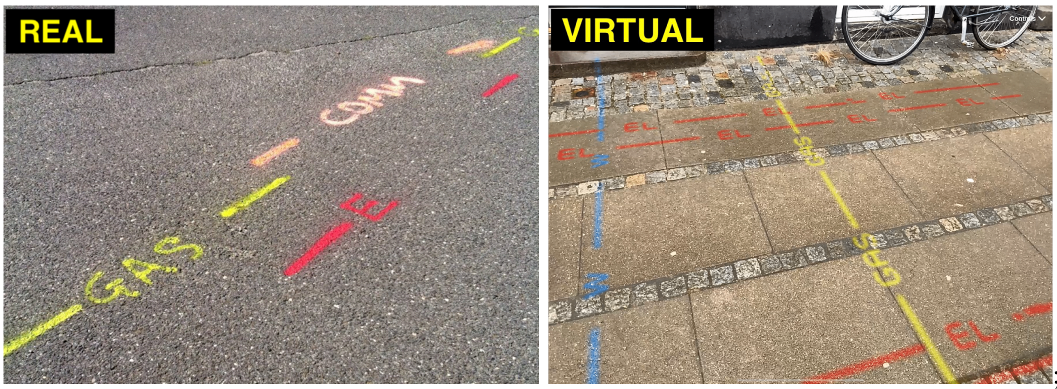
augmentations linked to real-world objects or locations. Furthermore, in excavation work, visualizations should be accurate within a tolerance of no more than the width of a shovel head, which is approximately 20 cm.

Although our current setup presents a promising approach, further user evaluations and investigations have revealed certain limitations over time, including:

(a) **Compass based north estimation** fails to deliver consistent and reliable performance in urban areas.

(b) **Tracking drift compensation**: SLAM systems like used in ARCore and ARKit drift over time, especially when leaving local areas. This phenomenon can already appear at distances of 3 to 5 meters from the tracking origin in monotonous environments, such as asphalt, which is most common for SUE applications.

(c) **Data Quality** is an always recurring issue in man-made and man-maintained structures and data records. Due to non-mandatory documentation standards, subsurface infrastructure data is not broadly available in sufficient quality for convenient AR applications.



;

Figure 1.2: Virtual Spray marking: An AR application for automated spray paint marking of SUE utilities. Left image show real spray markings on the surface, right image in contrast show the virtual spray markings. Realism of visualization can be achieved by overlay blending in combination with accurate 3D LIDAR reconstruction of the environment. *Images from our previous work [4].*

Shortcomings **(a)** and **(b)** are setup related problems, where compass based north estimation is solved by an GPS based alternative approach. This algorithm can be combined with compass based estimation or used as standalone approach. However, it requires highly accurate GPS positioning for reasonable accuracy. To address the issue of tracking drift, a continuous drift correction algorithm was implemented to achieve seamless combination of local and global tracking. GPS positioning is used to monitor drift of local tracking over time and to trigger re-initialization when drift exceeds a certain threshold. The improved AR setup is capable of robust and precise visualization. In a perfect world, infrastructure would be digitized with sufficient precision for easy use in AR applications. However, documentation in the industry is not standardized, and many companies do not

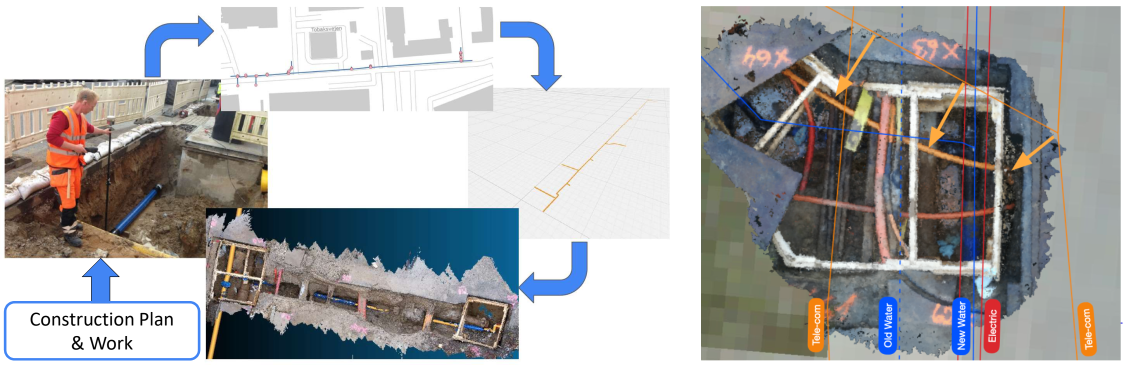
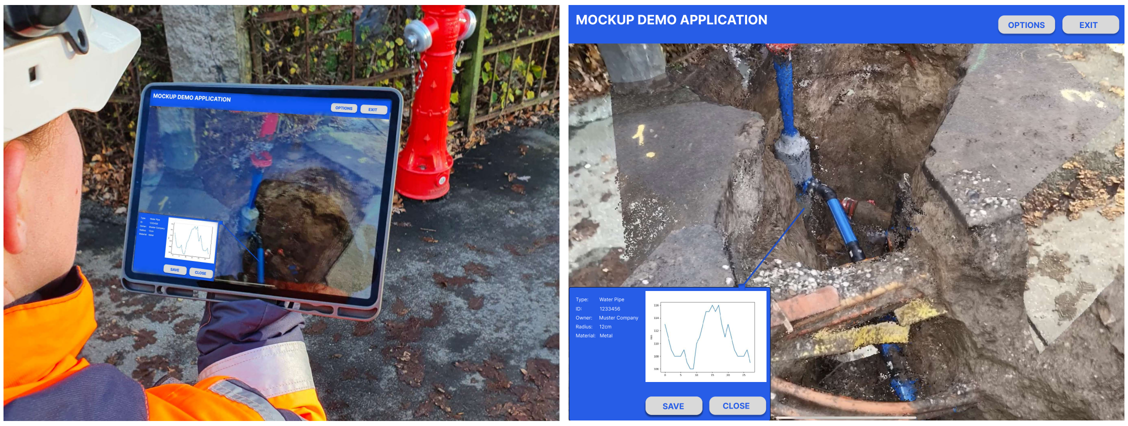


Figure 1.3: The left image illustrates an ideal workflow where utilities are surveyed and documented in a standardized, high-quality database after installation. This documentation can be used to create precise 3D geometry that aligns with real-world utilities. In contrast, the right image shows a real-world example of current documentation. In this example, the telecom asset is displaced and lacks height/depth information.

prioritize documentation after installing new infrastructure, as shown in Figure 1.3.

Our previous research has assumed the availability of accurate data, but this is not always the case. While some countries have implemented programs and platforms to improve data access, data quality is still not reliable. No quality obligations for companies lead on to poor documentation quality. Displaced utilities are not rare, furthermore information on depth of utilities is mostly not even provided, let alone some utilities are missing completely, as described later in this work. This problem is addressed by using pipe alignment with GIS data to create accurate visualizations of excavation sites, and to detect deviations from real-world objects, which can be used to update current documentation.

We aim to address the issue of poor quality GIS data (c) in the excavation setting for improved real-world registration. To achieve this, we propose a semi-automated GIS data registration algorithm. The user specifies a pipe in real world to be segmented. Once the segmentation is complete, a generated 3D model of GIS data is fitted to the segmentation. This can be seen as a complement to the perfect world example, where instead of surveying, the creation of the 3D reconstruction and the manual improvement of the current documentation is replaced by automated alignment. This enables precise augmentations on the real-world excavation scene, as shown in Figure 1.4. Additionally, the alignment step can be performed offline using geotagged 3D reconstructions, simplifying documentation without the need for experts on-site, using inexpensive handheld hardware, such as our setup.



Type: Water Pipe
ID: 1233456
Owner: Muster Company
Radius: 12cm
Material: Metal

Figure 1.4: Mockup of an on site live GIS data visualization application. Utility Asset specific meta-data can be visualized directly at excavation site. Left: User with hand-held device at excavation. Right: Application mockup.

2 Related Work

Contents

This section presents a comprehensive examination of related work. First the crucial part for outdoor visualization applications, accurate localization, will be discussed. This is split up in a general overview of outdoor AR setups, and further localization in context of this work, including local tracking and GPS based north estimation. Second, algorithms for point cloud segmentation are discussed, followed by work on further fitting of GIS data to the segmentation results. Lastly, previous work of AR applications in the field of subsurface engineering will be reviewed.

2.1 Localization

Localization is a key component in AR applications, as it enables the precise placement of virtual objects within a physical environment. However, the specific localization requirements vary depending on the nature of the project. For example, in simple games, such as Pokémon Go[1], localization may only require ground plane detection. Whereas in indoor environments, localization may need to be specific to a local space, allowing for the placement of virtual objects on walls or tables. In certain environments, such as warehouses, localization within a known environment is required.

In the context of this work, localization needs to be on a global scale. There are various methods to achieve this and an overview of these possibilities will be presented,

[1] `https://www.pokemon.com/us/app/pokemon-go/`

with a subsequent determination of the optimal approach for the context of this work. The second part will focus specifically on localization within this context.

2.1.1 Localization for outdoor Augmented Reality

Research on outdoor AR has been done since the first mobile hardware was available. A quick summary of the road on outdoor AR setups is given in the following paragraph.

- In 1997 and 1998 first outdoor AR prototypes were developed [6, 7]. Both systems featured a very similar setup, backpacks stuffed with hardware, composed of a wearable computer, a digital compass or other orientation sensors and a differential GPS. Later Thomas *et al.* [8] advanced their setup also including gloves for haptic input, to enable interaction with the environment.
- In 2004, Robertson *et al.* [9] proposed the concept of utilizing image feature matching with geotagged reference images in a database to determine the user's current pose. They presented a setup for urban navigation as an AR application. This idea using a geotagged image database is pursued in later ideas again.
- Back in 2006 Reitmayr *et al.* [10] introduced a tracking approach based on known 3D models to perform edge detection for vision tracking. While there are many approaches that utilize 3D models, for outdoor applications, the size of the 3D data can be quite large. As a result, approaches using simplified models are preferred for outdoor applications.
- In 2012, Oskiper *et al.* [11] developed a setup consisting of a camera, an IMU and GPS. The system employed an Extended Kalman Filter for fusing IMU and camera data for robust real-time tracking, complemented by the precise global positioning provided by GPS. The Kalman filter approach presented in this work is an early version of the now-common SLAM approach. From this work on SLAM based systems became the go-to solution. Especially with enhancing mobile computation power, vision tracking became the most common approach for local tracking.

Nowadays, all approaches for outdoor AR rely on vision-based tracking for stable, robust, and real-time ready local pose estimation. To achieve precise geo-referenced visualizations, tracking must also be registered on a global scale, with an accuracy level that is appropriate for the specific application. Suurinkeroinen's thesis [12], provides an overview of the current state-of-the-art tracking methods that can achieve centimeter-level accuracy for outdoor AR applications. It summarizes the advantages and disadvantages of the various methods and gives a clear indication of which approaches are best suited to the requirements for specific applications.

After all, SLAM-based systems are mostly trusted today. However, as SLAM only tracks on a local scale, it must be extended for global tracking. One simple and straightforward approach is to initialize the system at a known location, such as scanning a QR code at a surveyed position or by physically placing the device in a known position. This type of initialization is only feasible for a limited number of applications. Barcode-based initialization is a possible solution for prototypes or for limited visualizations on fixed places, like visualizations at city sights. In open-world SLAM applications, there are generally two possibilities for global registration. The first is the use of **additional geotagged data**, such as 3D models or images. The second is the extension by **external localization hardware**, such as GPS. There has been extensive research in this field, and the following paragraph will outline various approaches and their respective advantages and disadvantages.

First, let's discuss the previously stated idea of image matching using geo-referenced data [9]. In the work of Zhang *et al.* [13], positions for a given view are calculated using SIFT (Scale-Invariant Feature Transform) feature matching in a geotagged image database. This method can be integrated into a SLAM tracking approach, either by adding key frames with calculated global camera poses from this algorithm, or by incorporating it directly into a SIFT-based SLAM system. One of the main advantages of this approach is that it does not require any additional sensing equipment. It can be implemented on a mobile device with internet connectivity and a camera. To speed up the process, a rough position estimate from phone signal triangulation or GPS can be used, which is possible even on standard devices. However, the image database must be created, which can be a challenging task. There is public data available which could be used for creating such database. Websites like Flickr[2], Tumblr[3] and Instagram[4] provide millions of images, partly geotagged. This approach would require the elimination of poor-quality samples, and the quality of geotagging may not always be reliable. Projects, such as Google Street View, provide an excellent foundation for creating such databases. Zamir *et al.* [14] and Vaca-Castano et a [15] implemented systems that utilize this type of data. Since Google Street View consists of professionally taken, geotagged images, it is well-suited for this type of approach. However, it is only applicable in areas where Google Street View coverage is available.

Another vision-based approach combines SLAM with edge-based matching of buildings, which is achieved by creating building models using 2.5D maps from publicly available Open Street Maps (OSM) data [16]. Similar to the approach used by Reitmayr in 2006, edge detection is effectively used for localization. The advantage of creating 2.5D maps from OSM is that it can also be used in smaller villages, as building floor plans are often available. However, the accuracy and up-to-dateness of the data in smaller villages is not

[2]www.flickr.com
[3]www.tumblr.com
[4]www.instagram.com

guaranteed, also features such as trees, hedges, and other vegetation can interfere feature detection, making this approach more suitable for urban environments.

The current trend towards deep learning applications is also reflected in this research area. Many computer vision problems are well-suited for Convolutional Neural Networks (CNNs). Rao *et al.* [17] used a deep learning-based detection approach for image recognition and matching with a given geotagged image database, similar to the previous examples. This approach can directly process an input video stream and register the location where geo-referenced visualizations should be placed in the current view. While their approach is not yet real-time ready, steady improvements in system architecture and the increasing power of mobile hardware make it a possibility for future systems. However, it is also dependent on the availability of geo-referenced data in the region of use.

It shows that at present, vision-based approaches require additional data, which is only widely available in specific urban areas. Applications that are limited to specific environments where such databases can be provided are well-suited for such approaches. However, for SUE applications that are supposed to work nationwide and in any environment, pure image-based approaches are not practical. Therefore, a combination of SLAM and additional sensing is the most viable solution at this stage.

What can be used alongside SLAM local tracking without the need of additional datasets? The obvious choice is GPS and an orientation sensor such as an IMU or at least gravity sensors. Nevertheless, first let's look at some other options for position estimation. There are many possibilities for position estimation such as Triangulation, Trilateration, or Proximity within a given network. However, many of these approaches can be eliminated immediately because the necessary infrastructure is not available, or it is not feasible to install it, such as sonic-based localization. Mobile network infrastructure can be used for localization almost everywhere, although the current level of precision is not very high. However, 5G networks could be used to achieve sub-meter accuracy [18]. This is still ongoing research, and 5G networks are not fully developed yet. This could become a very promising possibility in the coming years, but currently, satellite-based localization is the only reasonable option.

Satellite-based position estimation (GNSS or GPS) is subject to noise in urban areas, such as shadowing or reflection of signals. Having said this, local tracking is purely vision based, hence stable high precise GPS positioning is not a must-have but rather just important for initialization and if needed re-initialization at some points. Nowadays, every smartphone has built-in GPS, but is it precise enough for reasonable geo-referenced outdoor visualizations? Standard GPS is not suitable at all, as it has an accuracy of no better than 10 m. However, raw GNSS measurements, available to developers since Android 7 in 2016, enable improved position estimates via DGPS (Differential GPS), RTK (Real-Time Kinematic) or PPP (Precise Point Positioning) positioning algorithms on android smartphones. Evaluations of optimizing

GNSS positioning on android phones show interesting results [19]. Smartphones with multi-frequency GPS receivers were used in a NRTK (network-streamed correction data) setup, achieving good accuracy even in centimeter levels with near reference stations. However, noise from built-in antennas led to poor outliers and the accuracy is not sufficient for the purpose of this work. But it is worth considering and feasible for many other use cases. The experiment of Niu *et al.* [20] with the combination of VI-SLAM and RTK positioning provided by the smartphone showed promising results. Unfortunately, this concept is still in proof-of-concept stage, but could become the go-to combination for outdoor AR setups. External GNSS systems are currently the best option. There are some works on integrating external GPS positioning into different SLAM systems. Nevertheless, using commercial local tracking software like ARCore, ARKit or Vuforia extended with external global registration seems very advantageous due to constant professional maintenance. Platform-independent approaches can be implemented using Unity [21]. Vision-based local tracking with external GPS for accurate global localization seems to be the best choice at the moment.

2.1.2 SLAM and external sensor combination

In summary, for SUE applications, the combination of SLAM or similar algorithms for local tracking and external sensors for geo-registration is the best approach. Unity Game Engine can be used as the base to achieve cross-platform AR applications. Unity's AR-Foundation provides robust local vision-based tracking which is combined with external sensors, as described in our previous work [22]. The sample applications of virtual spray paint marking and virtual daylighting, using this localization hardware, are shown in detail in our previous work [4]. The setup is described in more detail in Section 4.1.1. This section will focus on local tracking and GPS-based north estimation, which will be used to address the reliability issue of compass-based north estimation.

Local Tracking

To achieve robust and real-time capable tracking, standalone hardware-based tracking is not a viable option, as it is prone to errors and not robust enough on a real-time scale. Tracking is a major research topic in robotics and computer vision. There are various approaches for real-time tracking in different setups. The most common one in AR is SLAM, which was first developed in robotics to localize a robot (2D position and 1 orientation angle) and simultaneously creating a 2D map of the environment using a 3D scanner sensor [23].

On this basis, many different versions were developed, utilizing various sensors and providing different degrees of freedom. The most relevant research in the field of AR involves vision-based SLAM versions using single camera setups [24], stereo camera-based and RGB-D cameras. The work of Klein *et al.* [25] with early approaches using SLAM in an AR application and later extending it to a real-time version running on a smartphone [26]. ORB-SLAM [27] is a great example, capable of using all three mentioned sensors. Currently, some form of SLAM algorithm is used in almost all commercial products like Apple's ARKit, Google's ARCore, or Microsoft's MR (Hololens). Strictly speaking ARKit is not slam-based, it is built on visual odometry (VO) [28] , which is similar to SLAM but with focus on local consistency. SLAM is concerned with global map consistency, whereas VO can be used as part of a full SLAM system [29]. However, while classical VO does not aim for loop closure, ARKit is able to recognize previously visited areas. These SDKs are optimized for specific hardware. Platform-independent engines like Vuforia with Vuforia Fusion and Unity with ARFoundation provide interfaces for straightforward use of platform-specific tracking SDKs.

LIDAR bases SLAM: LIDAR sensors are becoming increasingly popular in mobile devices, enabling more accurate and robust SLAM implementation with highly accurate environment maps. While SLAM alone does not require highly accurate 3D reconstructions for good tracking, when high-quality reconstructions are required, external 3D ranging sensors are very common. Single camera-based setups can also provide good 3D reconstructions via structure from motion [30], but this is more time-consuming and cannot be run simultaneously with SLAM in real-time.

GPS based North Estimation

Magnetometer-based north estimation can be affected by metal objects. An alternative approach using GPS is commonly used in mobile robotics and vehicle heading estimation. The general idea is shown in Figure 2.1. Double GPS-Antenna setups are used to calculate the angle difference (β) between two points in time (t_n). In this simplified illustration, this is straight forward using trigonometric functions. In practice, there are more considerations to keep in mind. Single GPS measurements will have some

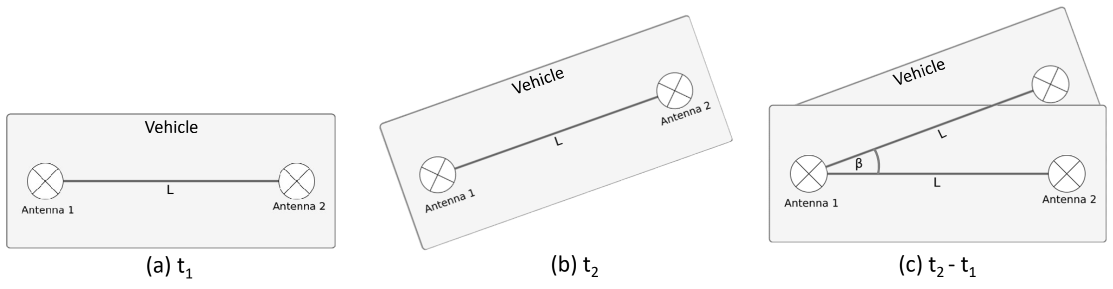


Figure 2.1: The illustration shows a simplified version of heading calculation over two time frames. The heading is represented by the change in direction of the vehicle, as measured by the two points of the GPS antennas. This is represented by angle α in subfigure (c).

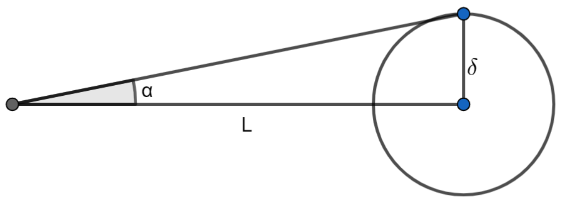


Figure 2.2: The heading angle error α, is a function of both the baseline L and the GNSS error δ. By increasing the baseline L, the total error can be minimized as the GNSS error δ becomes less significant.

degree of error. However, using a two-antenna GNSS setup with a specific baseline can be integrated effectively in a model of vehicle kinematics to estimate vehicle sideslip. Covaciu *et al.* [31] demonstrate a time-varying function based on "bicycle model" kinematics in a vehicle equipped with two independent GPS receivers. This approach is commonly used with Extended Kalman filters for estimation, as it provides accurate results when well-defined kinematic models are available [32, 33].

In addition to GNSS accuracy, the baseline length has the greatest impact on the quality of the estimation. In an experimental setup [34] using two DGPS receivers with baselines ranging from 1 m to 8 m, the error was shown to decrease with an increase in baseline length, as predicted by the Equation 2.1.

$$\alpha = tan^{-1}(\frac{\delta}{L}) \tag{2.1}$$

In this work, a combination of local tracking positioning and GPS measurements from a single antenna setup is used for north angle calculation. This idea is derived from the concept of the common two antenna approaches. To the best of our knowledge, no similar works have been conducted yet. The algorithm is described in Section 4.2.1.

2.2 Segmentation of 3D point clouds

Segmentation is a broad field in computer vision, with many well-established approaches. In this work, we focus on segmenting 3D point clouds. Segmentation of point clouds can be challenging due to high redundancy, uneven sampling density, and lack of explicit structure. Normally segmentation is the task to classify multiple homogeneous regions, however in the case of this work we are only interested in one specific class. There is a wide range of segmentation algorithms in 2D or 3D space, both have advantages and disadvantages. Additionally, access to prior knowledge of GIS metadata, especially the radius of pipes, could be useful for the segmentation task.

An overview of classical segmentation approaches in point clouds is given by Nguyen *et al.* [35]. Since point cloud data is mostly noisy, sparse and unorganized, segmentation can be quite challenging. The data also lacks a statistical distribution, features are sharp and the foreground is often intertwined with the background. Some common approaches for segmenting point clouds include:

- **Edge Based:** Regions are found by detecting boundaries. These methods allow fast segmentation, but come with accuracy problems due to their sensitivity to errors. These methods are not suitable for this work as the area of interest lacks distinct edges.
- **Region Based:** These methods make use of neighborhood information to cluster similar regions. They can handle noisy data effectively, but may result in over- or under-segmentation. These methods can be either seeded, where starting points are provided, or unseeded, where points are selected randomly or based on certain criteria. This approach can be used for the given data in this work, but it can be computationally intensive.
- **Attribute Based:** Regions are clustered based on specific characteristics of the point cloud. They are robust but highly dependent on the quality of derived attributes, require careful parameter selection, and are computationally intensive.
- **Model Based:** Model based methods make use of primitive geometric shapes for grouping. Algorithms such as RANSAC are very robust at detecting simple structures and cope very well with outliers. Therefore, it is likely to be a fast method for detecting cylindrical objects like the pipes in this work.
- **Graph based:** Point clouds are represented as graphs for efficient search in unstructured data. This allows fast segmentation based on features such as color or surface normals. Such an approach could also be quite interesting in the context of this work.

Among these different segmentation techniques that can be applied to 3D point clouds, the following three turn out to be suitable: region-based, model-based, and graph-based. Given the cylindrical shape of the target objects, such as pipes and cables, the model-based approach is considered as the most promising method.

Model Based : Gelfand *et al.* [36] proposed a local slippage analysis-based approach for classifying slippage shapes, such as spheres, helix, planes, cylinders, and linear extensions. This approach utilizes ”slippage signatures” to cluster points and identify the most likely shapes. These signatures consist of rigid motion sets, which can be applied to specific shapes to create sliding transformed copies of the original without creating any gaps. However, a significant challenge in the field of SUE is that pipes are not always reconstructed perfectly, often due to data collection from above the excavation ground. As a result, only the top portion of underground assets is captured, and depending on the depth of the structure and the 3D reconstruction approach used, the reconstructed pipes tend to be rather flat than cylindrical, thus this approach cannot be used in this case.

Region Based and Graph Based: There are several Region-Based approaches to detect geometrical features by clustering normal vectors of similar direction to planar shapes [37, 38]. In this work, since the radius of pipes is known, neighbor points can be clustered using normal vectors. The normals should change according to the distance between the points derived by the given radius, starting with a seed point on the pipe. However, due to the issue of poor reconstruction shape, this approach is not feasible. A more appropriate approach for this scenario would be Color-based segmentation, as pipes typically have different colors compared to dirt in excavation scenes. Strom *et al.* [39] proposed a Graph-based segmentation based on color and surface normals for clustering. A similar approach could be used in this work, where color values are given the highest weight in clustering, while normals and distance to neighbors can be used to detect outliers. This could be implemented as a seeded region growing approach using a KNN-Tree structure for efficient point cloud representation.

It is also important to consider **machine learning** approaches for 3D point cloud segmentation. 3D features can be extracted and machine learning can be used to learn different classes of object types. In complex scenes, such approaches tend to be more effective than classical algorithms. The rise of Deep learning has also brought a significant amount of research in this area. One popular architecture is PointNet [40], which has been used in various applications. The architecture can be trained for 3D shape classification and shape part segmentation. However, it should be noted that Deep Learning applications require suitable training data, which is not available for this particular problem.

Segmentation in 2D: Segmentation can also be performed in 2D image space. Point clouds can be rendered, and virtual images can be taken from any desired viewpoint. By moving from 3D space to 2D space, segmentation can be accelerated and some noise present in the 3D segmentation will not be present in the 2D projection. However, the results must be reprojected back into the original coordinate space. Color-based approaches are suitable as the assets are well differentiated from the background and typically have simple shapes in image space. Some interesting approaches make use of the watershed algorithm for seed point identification and subsequent region growing for segmentation [41]. When using color features for detection, Hue values in HSV Image space are effective [42]. Points of interest from the segmentation process can be re-projected back into the 3D scene, as described in the book *Multiple View Geometry* [43].

2.3 Fitting GIS models to Segmentation

Depending on the chosen segmentation approach, there are various options for model fitting. 3D models can be generated from GIS data, where only polylines are provided. Alternatively, 3D polylines can be used directly for fitting.

Iterative Closest Point (ICP) is a widely used algorithm for registration of 3D objects, as it is compatible with a wide range of different representations of geometrical data. As the name suggests, ICP is an optimization algorithm that iteratively finds the best possible fit of points to a given model. Besl *et al.* [44] proposed one of the first versions, which is a least-square fitting approach. Nowadays, there are many different implementations of the ICP algorithm, customized for various scenarios [45]. However, a major disadvantage in the context of this work is that ICP requires a good initialization of a reasonable initial fit and relies on accurate segmentation to avoid excessive noise.

Principal Component Analysis (PCA) is typically used in classification and compression techniques for projecting data onto other orthogonal basis. However, it can also be used for rough registration between two point clouds by aligning the eigenvectors of their covariance matrices. This approach is very sensitive to outliers and also relies on point clouds with similar distributions. If point correspondences between both clouds are available, **Singular Value Decomposition (SVD)** can be used to minimize Euclidean distances between these points, which is solved as a least-squares problem [46]. Both algorithms are not feasible for this work. Another common approach for general primitive fitting is **RANSAC** (RANdom SAmple Consensus), which can be used for fitting cylinder shapes, representing pipes in this context. Schnabel *et al.* [47] proposed a robust and efficient approach for primitive shape fitting based on RANSAC. The latest primitive fitting approaches are based on **deep learning**. Li *et al.* [48] proposed a supervised primitive fitting algorithm which outperforms state-of-the-art RANSAC-based methods. The state-of-the-art point cloud registration algorithm is presented by Yang *et al.* [49] with *TEASER*. The algorithm is highly robust against outliers and shows superior performance and accuracy when compared to ICP and RANSAC approaches. The approach is based on truncated least square estimation relaxed to a semi-definite program (SPD). However, solving large SPDs is computationally expensive, therefore they also propose a faster approach, TEASER++, using graduated non-convexity for rotation estimation.

In conclusion, given that our segmentation algorithm computes center-points along the pipe, there are two options for model fitting. The first option is to extract all points within a specific distance to the segmented center points and run an algorithm such as TEASER to match a 3D model created from GIS data. The second option is to directly fit the segmented center-points to the given polylines from GIS. This can be achieved through a simple least-square fitting in 3D space. While the first method would likely yield higher accuracy, the second approach would be significantly faster. Therefore, in an offline scenario, a more complex algorithm could be used for fitting, while a real-time approach should utilize simple least-square fitting.

2.4 Augmented Reality in Subsurface Utility Engineering

There is limited research on the application of AR in the context of SUE. Most of the existing work focuses on visualizing subsurface data on the surface prior to excavation. Smart Vidente [1] is an application for real-world interaction with a GIS database in an AR setup. The system consists of a tablet PC equipped with a camera, a 3DoF orientation sensor, and a DGPS receiver for achieving centimeter-level accuracy. Earlier, Schall *et al.* [2, 50] presented predecessor systems also built on GNSS and orientation sensors. However, Smart Vidente incorporated a laser range finder, enabling 3D interaction. Differential correction data is streamed via the internet to achieve accuracy of less than 10 cm and the orientation sensor provides accuracy up to one degree. The Interactive GIS Registration application presented by Sun *et al.* [51] also relied on GPS and IMU positioning. They combined a head-mounted display for visualization and a small keyboard for interaction. To overcome displacements, position corrections can be made by the user. A more recent approach for visualizing GIS data is the LARA project [52]. Global position is also fully estimated by external sensors. GNSS and IMU are fused to a 6DOF Camera pose when calibrated before use. Local tracking for AR and visualization is provided by Unity Rendering Engine. Another approach, though controversial, is using outside-in tracking, which is not commonly used in outdoor systems [53]. This approach focuses on tracking multiple devices at road maintenance work by using several Laser Range Finders to track people walking in the scene. They achieve an accuracy of less than 25 cm, which is sufficient for road maintenance work.

Overall, the most reliable approaches for SUE combine GPS and IMU. While there is some interesting research on visualization techniques for SUE in AR [1–3, 54], these methods could be improved using current visualization standards. Additionally, all of these works assume perfect documentation. This work aims to present an approach for pipe segmentation and fitting that deals with poor-quality data commonly found on excavation sites.

3 Motivation and Requirements

Contents

The chapter on motivation and requirements provides an overview of the benefits of using AR for SUE and establishes specific requirements for successful implementation. The section on localization in outdoor AR is reviewed in depth, followed by an analysis of the problem of poor data quality and the corresponding requirements for its resolution.

3.1 Tracking for AR subsurface utility engineering tasks

Recent hardware advancements, such as the 2020 models of Apple's iPads and iPhones, incorporate high-precision sensing capabilities. The integration of LIDAR sensors allows precise 3D reconstruction of the local environment. By extending this localization technique to a global approach, applications that involve geotagged visualizations with centimeter-level precision can be developed. There is a wide range of potential outdoor AR applications, particularly within the construction industry. For example, the use of highly accurate on-site 3D models can enhance spatial perception of architectural plans. Thus facilitating communication of the designer's intent to clients within the physical context. In this context, precision is particularly important when visualizing plans for building extensions. Another example is the utilization of AR for providing assistance at construction sites, where geotagged visualizations can be employed throughout the construction process. This can include the use of on-site 3D plan visualizations to speed up the execution of construction plans or aid in decision-making processes. However, this work will focus on the specific use cases within the field of subsurface utility engineering, which will be discussed later on. The constraints and requirements for such tasks can vary, but the overall goal is to achieve highly accurate localization. This can be accomplished through various methods, depending on the specific environment. In general, the combination of LIDAR and GPS can provide a sufficient solution for achieving real-time 6 Degrees of Freedom (6DoF) tracking on a global level. The next chapter will dig into the technical specifications for subsurface utility engineering.

3.1.1 Current workflow for excavations

Excavation of subsurface utilities is a complex task. The cost of excavation can vary depending on the specific utilities being uncovered and can become quite costly when encountering hazardous utility lines, such as gas pipelines. Therefore, excavation must be done with great care in these situations. Current best practice for excavation work consists of:

1. **Spray marking:** Manually spraying markings is a common and inexpensive method in most excavation setups. Locations of underground utilities are marked on the ground surface, using information provided by a GIS database or sometimes on paper maps. These markings are done by hand by the workers, making it challenging to ensure they align exactly with the plans.
2. **Enhanced Sensors:** To ensure marking placement is correct and to estimate depth, special sensors are used. Usually, Ground Penetrating Radar is used for scanning subsurface assets.
3. **Daylighting:** One of the most efficient ways to do this is by using a hydro-vacuum excavator, which exposes utilities using a combination of pressure, water, and air. However, this method can be expensive and may not be suitable for all environments. Currently, it is common to combine manual labor and the use of excavators.

Especially steps two and three do come along with additional cost. However, to bypass those steps with alternative solutions precise and also complete documentation would be needed, which is not the case at the moment.

3.1.2 Utility Strikes

Damage to utilities cause billions of costs every year, to put that into numbers, \$50-100 billion in the US, more than €1 billion in the Netherlands and also more than £1 billion in UK [55]. DIRT Report (damage information reporting tool) of 2019 reported damage of more than \$ 30 billion in North America [56], where 90% of damage cause was split into around 30% cause due to locate problem of facility, around 30% no locate data requested and around 30% excavation practice indebted. In the UK alone, more than 60,000 utility strikes were reported in 2016 [57]. Reasons leading to damages are reported as following:

- Statutory reports are outdated and do not always reflect as-build information, most companies said provided data by utility companies and clients are not adequate mostly and cannot be trusted.

- Poor training of workers.
- Inaccurate marking of assets.
- Physical site conditions at excavation site.
- Wrong chosen excavation technique.
- Bad scanning of asset depth by geophysical surveying.
- Planning with inaccurate documentation, therefore updated information after geophysical survey should be used.

The true costs of damages are often unknown by most companies. Including indirect costs, such as road disruption, plant damage, traffic delays, environmental damage, street regulation, atmospheric damage, loss of service provision, and related damages, the ratio of overall costs to cost of repair is calculated to be 29:1 [58]. Applying this factor to the average estimated cost of £3600 (€ 4120) per strike gives an average damage of £100.000 (€ 114.520) in the UK [59]. Another important consideration is that evaluations of marking tasks have shown that gas and water utilities are marked more accurately than other facilities, such as TV, internet, or electricity. Additionally, inaccurate marking and the lack of depth information have been identified as significant problems[60].

3.1.3 Using AR to improve SUE workflow

The previous section highlighted the huge impact of utility strikes. Given that human error is a significant contributing factor to damage, the utilization of AR for error prevention should be considered. This section will discuss the potential for AR to support current best practice excavation work.

Spray marking: The process of marking utility assets can be directly replaced or supported by visualizations, as previously demonstrated in our work [4]. This would eliminate human error in inaccurate marking of underground structures, particularly the issue of carelessness when marking less critical utilities.

Enhanced Sensors: This step cannot be replaced if feasible documentation is not present. However, if accurate documentation is provided, this step would not be necessary in the first place.

Daylighting: Assistance for the excavation process can be achieved through various approaches, such as providing annotations of the height distance from the current excavation level to the asset, or utilizing 3D visualizations of the utilities, such as cut-away or shadow projections. However, determining the most effective approach would require significant evaluation and research.

In addition to the excavation process, there are other areas where visualizations can be beneficial. One such example is the use of previously captured 3D data or accurate 3D models for informed decision-making with on-site AR visualizations. A prototype system was used by Hansen *et al.* [3] to evaluate its usefulness for workers at excavation sites. Interviews with utility owners and constructors revealed that this could help prevent excavation damages. Visualizations can also help provide clear scene understanding, reducing misunderstandings between foremen and the excavation team.

3.1.4 Requirements for AR in Subsurface Utility Engineering

Suitable AR applications must ensure the precise overlay of visual augmentations on the physical environment, requiring accurate 3D reconstruction of the terrain. This is particularly critical for the task of segmentation, where a high degree of precision is necessary for accurate depth estimation of pipes, with errors no greater than 5 cm at a scanning distance of 3 m [**R1.1**]. Furthermore, the precise positioning of geotagged data visualization is crucial for SUE, with a minimum accuracy of 20 cm required to ensure the detection of utilities during excavation with a shovel [**R1.2**]. In addition, accurate north estimation is crucial for displacement error reduction, as errors increase with distance from the object of interest, as shown in Figure 3.1. For example, an error of just one degree can result in a displacement of 90 cm at a distance of 50 m. In this use case of SUE, north estimation should be reasonably small, to achieve an accurate visualization in relation to the local environment reconstruction. Here we assess that with the expected sizes of the excavations a north estimation error of less than 1 degree should be aimed for [**R1.3**]. Furthermore, interviews [61] showed that most participants see manual realignment of localization as a major concern in outdoor AR setups. This highlights the need of an automated initialization and re-initialization logic [**R1.4**].

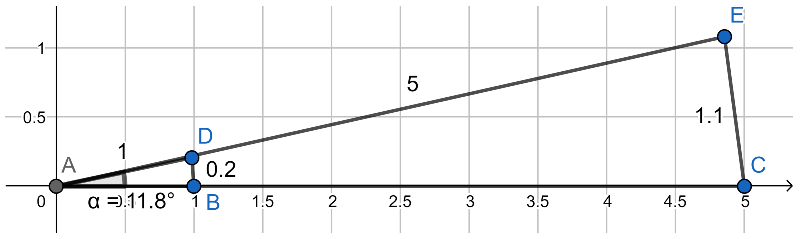


Figure 3.1: Small north estimation errors lead to large displacement at distance. An angle error of 11.8 degrees cause a displacement of 20 cm at 1 m distance (Point B to D) and over 1 m displacement at 5 m distance (Point C to E).

3.2 The Problem of poor GIS data quality

Most AR scenarios for SUE ate heavily dependent on the quality of documentation. As an example, spray marking visualizations are only useful when based on accurate data, as visualizing displaced or incorrect utilities can harm the excavation process. The previous chapter demonstrate how precise geo-referenced visualizations can be used for supporting SUE work. However, the current state of documentation is not sufficient for commercial applications. Subsequent sections will provide further insight into documentation.

3.2.1 Current state of SUE documentation and how it could be improved

Workers in the field of SUE raise complaint about current state of documentation [61]. Depth information of assets is often unreliable or not provided and only few companies survey their newly installed utilities. Most of the time reporting of wrong documentation is only done when damaged or completely wrong, that is the case because proof of duty is on their side. In Denmark, private secondary utility lines are often not provided at all, as they are not legally required to do so. Currently, documentation requirements only necessitate the reporting of 2D data, and there is a lack of standardization in digital formats. Some documentation may still be in the form of scanned drawings, which cannot be easily imported into GIS systems. However, several countries, including Denmark, the Netherlands, and the United Kingdom, are implementing programs and platforms to address the current challenges in documentation. Despite these efforts, the accuracy and reliability of data remains a concern, as demonstrated in Figure 1.3.

Hansen *et al.* [61] explored methods for improving the quality of records in subsurface utility engineering. An areal-based mapping approach using ground penetrating radar was found to be infeasible due to high costs and difficulties in nationwide matching to existing GIS data. Instead, they proposed utilizing ongoing construction and maintenance projects to improve data quality. Some companies already employ survey methods to locate, verify and map existing assets, but this information is often not shared beyond the specific project. The authors suggest the use of photo geometry or laser scanning for on-site documentation and utilizing geotagged 3D reconstructions for accurate documentation. Hand-held hardware can be used by workers for reconstructions, and semi-automated segmentation and matching algorithms can be used for utility data verification, mapping or correction.

3.2.2 Requirements for dealing with poor documentation data

The specific hardware requirements for a given use-case can vary. 3D reconstructions can be performed using a smartphone and photogeometry algorithms in an offline approach. Alternatively, laser scanners can be used for real-time 3D reconstruction. To achieve a precise geotagged 3D scan, the hardware specifications outlined in the previous section also apply here. When working with low-quality GIS data, it is necessary to locate utility assets within the scene. A segmentation algorithm should be developed to identify the assets in the 3D data. The segmentation algorithm should be user-initialized and semi-automated. It would be really challenging to identify assets autonomously in the 3D scan. However, when providing enough knowledge about the specific appearance of pipes, it may be possible but still hard to assign them to the correct asset in the database. Therefore, the segmentation should be easily triggered by the user either on-site in a live application or afterward in a specific software, such as through an integration into common GIS Software. The accuracy should also be less than 20 cm here. [**R2.1**]. The segmentation should then enable a fitting approach to verify existing documentation or calculate an error measurement. The alignment process should deliver a transformation that fits the displaced data as closely as possible to the segmentation. Here we should also aim for an accuracy in the range of centimeters [**R2.2**]. The process of 3D scanning, segmentation, and fitting should be able to achieve accuracy within the SUE standard of an error less than the width of a shovel. With the corrected data, documentation can be updated and used for on-site visualizations as described in the previous Section 3.1.3. Due to the lack of standardization in documentation, an API for GIS data handling must be implemented [**R2.3**]. It should be easily extendable to accommodate different data standards of different companies, to provide asset plans and metadata in a suitable format. Furthermore, an application for 3D model creation using given GIS data should be implemented to enable 3D visualizations and, if necessary, be used for the fitting approach [**R2.4**].

In conclusion, the requirements can be divided into two groups: [**R1**] those specific to setting up accurate outdoor AR localization, and [**R2**] those for implementing software to address the issue of poor documentation. All the requirements discussed in this chapter are summarized in Table 3.1.

Requirement	**Describtion**
R1	Localization for accurate outdoor AR visualizations
R1.1	Accurate environment reconstruction
R1.2	Global positioning with maximum error of 10 cm
R1.3	Robust and accurate North Estimation
R1.4	Automate (Re) Initialization
R2	Overcome bad and incomplete GIS data
R2.1	Utility Segmentation in 3D reconstruction data
R2.2	Fitting Algorithm for segmented utilities
R2.3	GIS data API to deal with non standardized documentation
R2.4	3D model creation from GIS data

Table 3.1: Requirement List: A compact representation of all found requirements, grouped into localization related as R1 and GIS data related as R2.

4 Setup Implementation

Contents

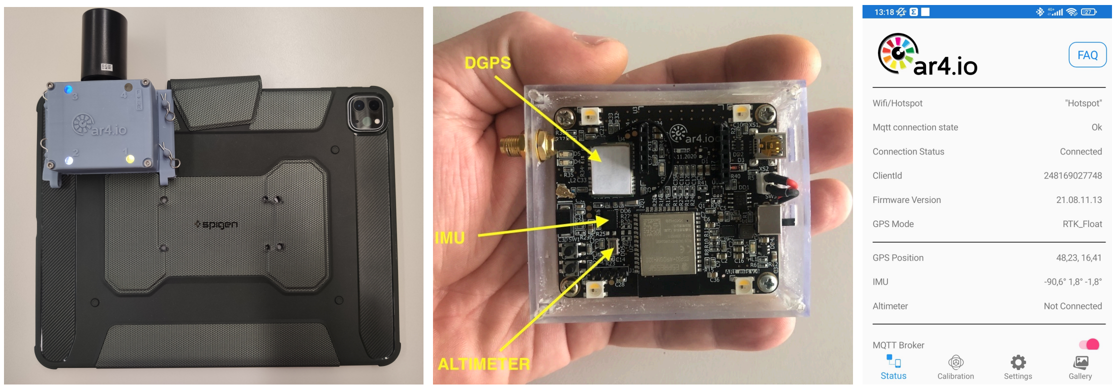
This section presents the implementation of specific enhancements to the setup. Initially, a brief overview of the current state is provided, with further details available in our previous work [4, 22]. Subsequently, the section describes the implementation of improvements to the current state.

4.1 Initial Setup

This section introduces the current state of the setup. The setup has undergone multiple iterations since the initial state during my bachelor thesis, leading to the current prototype.

4.1.1 Hardware Setup

The hardware component utilized in this work, referred to as CapsLoc, was developed during our previous work. Figure 4.1 illustrates the hardware configuration used in our experiments, which includes an iPad Pro equipped with a LIDAR sensor and the CapsLoc localization device. Evaluations comparing 3D reconstructions of industrial LIDAR scanner and 2020 iPad Pro [62], showed that latest mobile devices with built-in LIDAR can keep up with professional devices. The range of up to 5 m for scanning was also verified in independent experiments [63]. Results showed an average precision of about 2 cm with an accuracy of about 4 cm. This fulfills requirements [**R1.1**]. The CapsLoc device main components are an ublox high precision GNSS RTK module for global position estimation, an IMU with compass for orientation estimation and an altimeter for height estimation



Wifi/Hotspot	"Hotspot"
Mqtt connection state	Ok
Connection Status	Connected
ClientId	248169027748
Firmware Version	21.08.11.13
GPS Mode	RTK_Float
GPS Position	48,23, 16,41
IMU	-90,6° 1,8° -1,8°
Altimeter	Not Connected
MQTT Broker	

Figure 4.1: Setup used in our experiments: Left: iPad Pro with LIDAR sensor combined with CapsLoc sensor cube. Middle: Close shot of CapsLoc board with important sensors labeled. Right: Screenshot of CapsLoc App used for data streaming from sensor case to iPad.

since GPS has its greatest error in altitude. Detailed Evaluations and further explanations are found in our previous work [22]. Data is sent via a local Hotspot (WiFi access point) from CapsLoc to the mobile device using MQTT. MQTT [1] is a standard messaging protocol for Internet of Things (IoT). For achieving best possible accuracy the setup hast to be calibrated beforehand. For this purpose a semi-automated calibration routine was implemented using Furgale *et al.* [64, 65] work and provided calibration tool *kalibr*[2] from the Autonomous Systems Lab of ETH Zurich. The calibration approach is described in detail within our previous work [4]. This yields an IMU to Camera calibration, however if CapsLoc is used with a directly mounted antenna like in our setup the camera to GPS calibration can be derived from it. Later calibration is not of biggest importance considering the given accuracy requirement of around 20 cm, however it is necessary to achieve best possible accuracy.

4.1.2 Real Time Kinematic GPS in practice

Accurate GPS positioning is the most important component for our setup. GPS is a satellite based positioning system, which provides the user a 3D position with a maximum accuracy to 10 m. Accuracy improvements can be achieved using Differential GPS or Real Time Kinematic (RTK). Both of these applications make use of observations from reference stations to compute better position estimations. Depending on distance, similar environment and common satellites in sight, position accuracy is optimized. The achievable accuracy is described in Table 4.1. Correction Data is sent as RTCM (Radio Technical Commission for Maritime Services) messages, normally via radio transmission or over so called NTRIP Servers (Networked Transport of RTCM via Internet Protocol). The latter are provided by companies and can usually be accessed via a paid subscription.

[1]https://mqtt.org/
[2]https://github.com/ethz-asl/kalibr

	min Sattelites	max Base distance[km]	Accuracy[m]
GPS	3-4	X	10
DGPS	4	100	1
RTK Float	5	10	0.20
RTK Fixed	6	10	0.02

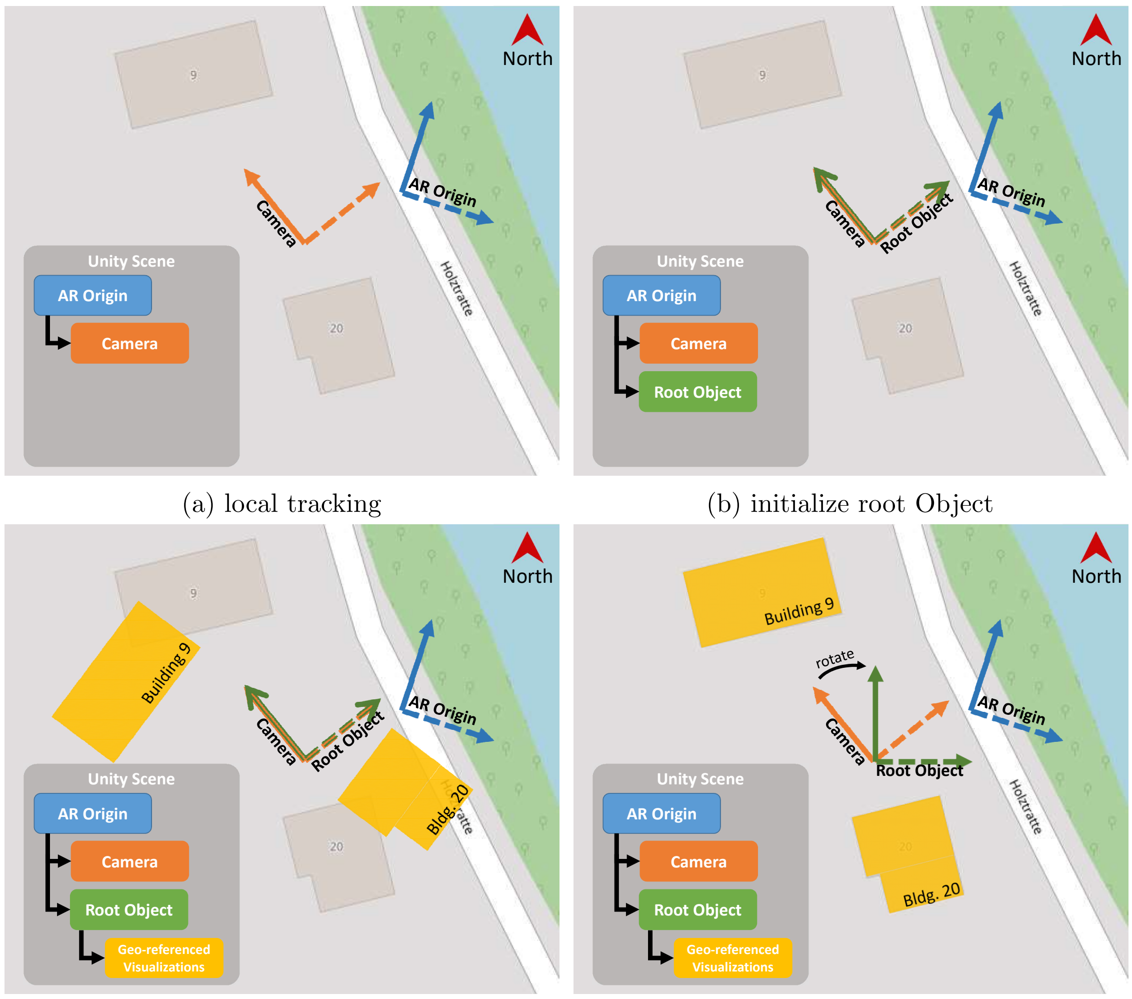
Table 4.1: GPS comparison and accuracy requirements: different GPS systems with given requirements of at least shared satellites in view, maximal distance to base station and consequential accuracy to be achieved.

However, there is also the option to set up a base station on your own quite easily with ublox receivers [66].

Nevertheless, we have observed some relevant properties when using GPS in practice. High accurate position estimates can be achieved using cheap patch Antennas. Drawbacks using cheaper hardware are, RTK fixed mode is hardly achievable when moving and require ground planes for better signals. Ground planes are metallic discs with a diameter of at least 14 cm (the more, the better). Mounting such planes onto a mobile setup decrease handiness and limits mobility, since it should always stay in a rather horizontal position. Furthermore, time to accomplish RTK fixed mode is significantly longer. Helix antennas, which are commonly used with drones, work best in free moving setups. As they are also used in our current setup, shown in Figure 4.1. The distance to the reference station has a direct impact on the time required to reach RTK fixed mode. Additionally, weather conditions also play a role in the accuracy of position estimation. Our observations have shown that RTK positioning can be unstable during cloudy or rainy weather conditions, even when using a high-quality antenna. This is probably due to variations in signal distortion at the rover and the base stations. Minimizing the distance between rover and base is recommended to reduce the influence of weather conditions on position estimation. Best practice for achieving RTK fixed mode is to initially place the equipment in an open space, such as a parking lot, for a few minutes prior to starting actual work. To maintain fixed mode, it is important to perform smooth motions and ensure that the antenna has a clear view.

4.1.3 Unity Integration

Unity is used as rendering Engine. Cross-platform support and integration of device-specific AR APIs enable flexibility in device selection. Building Localization on top of Unitys ARFoundation simplifies maintainability and ensures best performance, because the local tracking will always be up-to-date. The goal is to implement an application capable of fusing drift prone local tracking with global positioning provided by the CapsLoc device.



(a) local tracking (b) initialize root Object

(c) add geo-referenced objects (d) north correction

Figure 4.2: Illustration of the fusion algorithm broken down into 2D Top view to present it in a more understandable way. Coordinate systems are shown by two arrows, consistent line for north/y axis and dotted line for east/x axis. The blue coordinate system is AR origin, which is the local tracking origin. Orange is the current camera (user) position, moving inside the blue AR Origin space. The green coordinate system (Root Object) is used to align local space to real world space, which is represented in UTM. **(a)** Starting with local tracking in the scene. **(b)** RootObject is initialized with current GPS position and aligned with current camera position. **(c)** Geo-referenced virtual objects are added and placed as children of the RootObject. Here, buildings are displaced due to north error given by the local tracking. **(d)** The Root Object is rotated to align with real world north. Now geo-referenced visualizations align with the real world objects.

Fusion of Local and Global Positioning

The necessary steps to align both coordinate spaces are sketched in Figure 4.2 (a-d). Since ARFoundation provides sandboxed tracking, we cannot directly integrate global pose updates into the tracking algorithms. In Unity, there is a main coordinate space where

GameObjects are placed. In addition, each GameObject has its own local coordinate space. GameObjects can also be attached as child of other GameObjects, then they are placed within the local coordinate space of the parent GameObject, where the parent Object defines the origin. Local tracking origin (`AR Origin`) is initialized on startup and cannot be changed. The `Camera` (current user pose) is a child of `AR Origin` and is moving within this coordinate space **(a)**. To fuse global positioning into local tracking space, a new GameObject is introduced called `RootObject`. This Object is placed as child of the `AR Origin`). To align global objects correctly into local tracking, the `RootObject` is aligned with the current camera position **(b)**. Therefore, the position of the `RootObject` is set to the current `Camera` position and the latest global position of the CapsLoc system is saved as reference position. At the time of this alignment process, the camera is at this known global position. This allows placing geo-referenced objects in correct local distance to this global reference point, since local and global difference is the same in reference to this point **(c)**. Furthermore, the `RootObjects` Y axis rotation (north) is set by the difference of `AR Origins` Y (local tracking north) angle and the latest north estimation angle **(d)**. Step **(c)** and **(d)** can also happen in different order. New geo-referenced Objects can be placed at any time as child of the `RootObject`.

As previously stated, this state of the system suffers from errors due to north estimation via compass. Furthermore, the mapping of global and local pose is triggered by the user. Hence, drift of local tracking has to be adjusted manually.

4.2 Enhancing Accuracy to enable in-field use

In this section, we will discuss strategies for enhancing the current experimental setup. As previously stated, the reliability of compass based north estimation is limited under certain conditions. Thus, GPS-based north estimation is implemented as a supplementary method. Additionally, the ultimate goal is to achieve fully autonomous global localization. Therefore, the global-local mapping component (Tracking Fusion) has been revised and optimized. The data flow diagram in Figure 4.3 illustrates the integration of all components used in the setup.

4.2.1 GPS Based North Estimation

The concept of GPS based north angle estimation is a widely used technique in the fields of autonomous driving and robotics. Typically, this method utilizes a two-antenna setup to calculate the bearing angle between them. In contrast to traditional two-antenna setup, we present a single antenna approach utilizing the UTM coordinate system for north angle estimation.

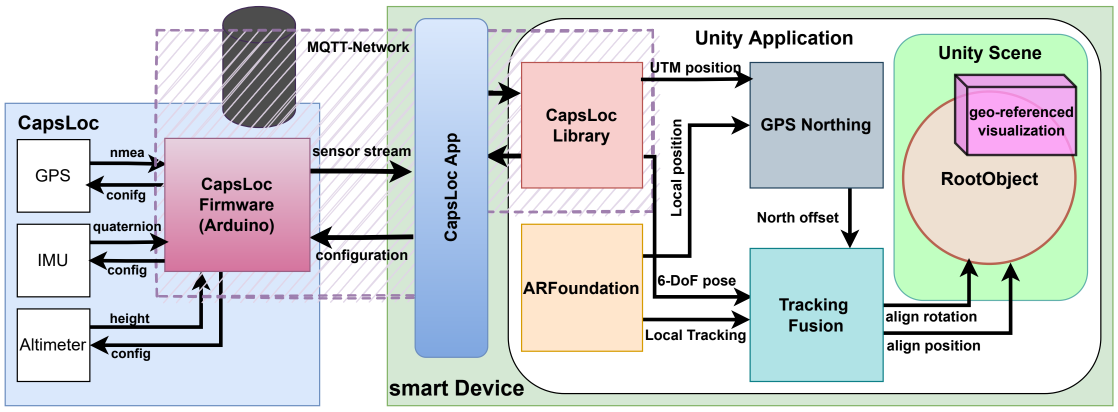
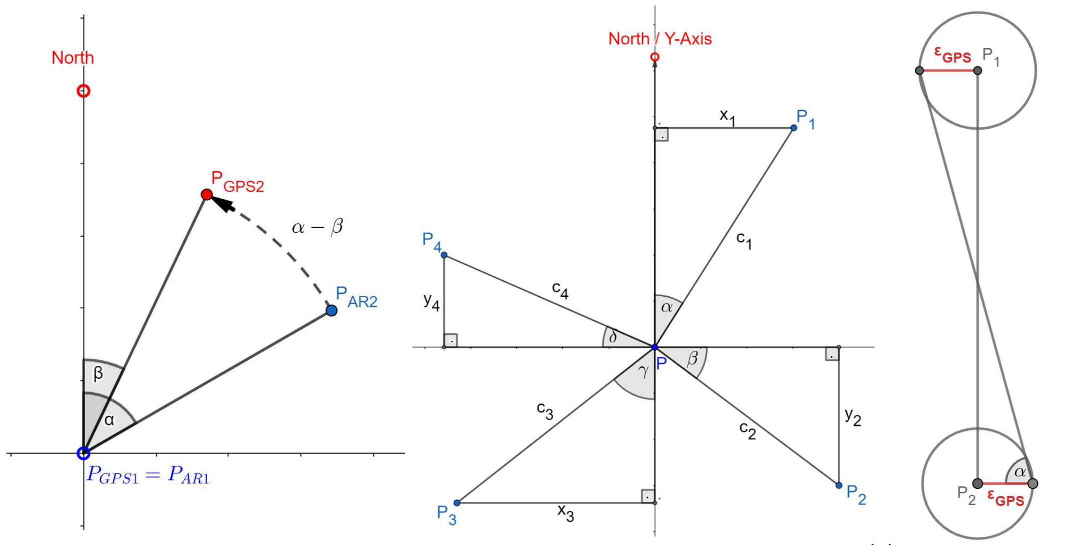


Figure 4.3: This figure shows the dataflow diagram of the whole (upgraded) Setup. The two Devices in the setup are on the left side the CapsLoc device (light blue) and on the right side the smart device (green). CapsLoc App (blue) is hosting an MQTT Server (purple dashed) to stream data between all parties. CapsLoc Firmware (purple) is responsible to collect data from the sensors (white inside CapsLoc) and handle configurations. In the Unity Application (white inside smart device) the CapsLoc Library (red) is responsible for communication with the CapsLoc device via MQTT. Local tracking is provided by ARFoundation (yellow). The GPS Northing (grey) component uses UTM position from GPS observations and local positions to calculate north offset of the local tracking. The tracking Fusion Algorithm (turquoise) gathers local poses, poses from the CapsLoc device and the north offset to manipulate the RootObject (orange) GameObject in the Unity Scene (light green) to align local tracking with global tracking. The final result is accurate visualization of geo-referenced Objects (light purple).

North angle calculation

In UTM coordinate space the Y-axis is aligned with true North. This enables the possibility to directly calculate the north angle between two given points. The angle can be calculated using trigonometric functions as shown in Equation 4.1. Depending on where the second point is located in relation to the first point, the calculation is slightly different. The four possible relations are sketched in Figure 4.4 (b). A right triangle is drawn between the two points and either the X or Y axis. Within the triangle the *asin* of either x_i or y_i (distance between both points on axis X/Y) and c_i (euclidian distance between both points) is used to calculate the angle between these two sides of the triangle. The offset of the used axis to the positive X-axis, pointing towards north is added onto this angle to get the final north estimation α.

$$\alpha = \begin{cases} asin(\frac{x_1}{c_1}) \cdot \frac{180}{\pi}, & \text{if } P_1 \\ asin(\frac{y_2}{c_2}) \cdot \frac{180}{\pi} + 90, & \text{if } P_2 \\ asin(\frac{x_3}{c_3}) \cdot \frac{180}{\pi} + 180, & \text{if } P_3 \\ asin(\frac{y_4}{c_4}) \cdot \frac{180}{\pi} + 270, & \text{if } P_4 \end{cases} \tag{4.1}$$



(a) Align local to true North (b) North angle estimation (c) Maximum error

Figure 4.4: (a) **Local and global tracking north alignment:** To align local tracking north to true north, difference of north angles between ground truth of two GPS positions (β) and corresponding local tracking positions (α) is used. (b) **North angle calculation:** This geometry sketch exhibits the trigonometry to calculate north angle between two Points using UTM Y-Axis pointing towards north. Depending on the section the second point is in, calculation differs as shown in equations 4.1. (c) **North estimation error:** α is given by position estimation error at both Points (P_1 and P_2) ϵ_{GPS} and the distance between both points.

Algorithm: In order to perform north angle estimation, a significant number of local and global position pairs are required. Each pair consists of a GPS position in UTM format, as well as the position of the local tracking at that corresponding time. Both the GPS position and the local tracking position are represented as two-dimensional vectors, with x and y values. These position pairs are stored in an array (`point_pairs`) for computations.

With each incoming GPS position, a new position pair is generated by combining the GPS position with the most recent local position. Also, calculation of possible angle corrections is initiated. By continuously performing these calculations, the total computation load is distributed over multiple points in time, thus avoiding any potential blocking of the main thread. The calculation of a single correction value is done as follows: Two position pairs are selected and the North Angle calculation, as previously described, is performed for each of the GPS positions and local positions. The angle between the two GPS positions should correspond to the true north angle. Additionally, as illustrated in Figure 4.4 (a), the difference between the two calculated angles should correspond to the correction value that must be applied to align the local north to the geographic north.

The accuracy of corrections is depending upon various factors. As depicted in Figure 4.4 (c), smaller GPS errors lead to more accurate angle estimation. As such, the use of RTK fixed GPS position is recommended. Additionally, the illustration indicates that an increase in the distance between points results in a decrease in the impact of GPS errors on angle calculation. Timing is also a crucial consideration, as the frequency of local positioning is typically higher than that of GPS. As such, proper time synchronization is essential, as poor timing can result in a misalignment between local position and corresponding GPS positions, leading to errors in correction value calculation. Additionally, the possible drift in local tracking should be considered too. To minimize its influence, position pairs for calculation should not be too far apart in both spatial and temporal distance.

When a new correction value calculation cycle is triggered, the algorithm calculates all potential corrections between the newly acquired point pair and a subset of point pairs selected from the `point_pairs` array. These new corrections are added to a list of all values calculated so far (`correction_values`). The selection of point pairs is determined by following properties: the maximum number of correction values to be used, as specified by `max_corrections_len` which balances the trade-off between accuracy and computational efficiency; the maximum number of point pairs to keep for calculating the correction values, as specified by `max_comparable_points`, which limits the number of calculations performed at each iteration; and the acceptable range of distances between point pairs, as defined by `min_distance` and `max_distance`, which addresses potential errors arising from both local tracking drift and GPS position estimates. Additionally, the maximum time offset for point pairs to be retained for calculations is specified by `max_time_offset`.

A final estimate of the angle correction is derived from the accumulated correction values stored in the `correction_values` list. These values form a normal distribution, with a low standard deviation of a few degrees in optimal cases. To eliminate outliers, values outside the first standard deviation are removed by sorting the `correction_values` list and discarding the first and last 15% of elements. The final angle correction is determined by the mean of the remaining values. The settings that yield optimal results are reported in Table 4.2.

Condition	**Value**
`max_corrections_len`	10000
`max_copmareable_points`	300
`min_distance`	1 m, $\epsilon_{max} = 2.3°$
`max_distance`	3 m, $\epsilon_{max} = 0.8°$
`max_time_offset`	20000 ms

Table 4.2: Specifications of the properties utilized for north estimation in this work. These settings have been identified as optimal based on the results of our evaluations.

4.2.2 Continuous Drift Correction

Fusion of ARFoundation and CapsLoc tracking is primarily depending on accurate north alignment. This alignment can be continuously refined through the incorporation of accurate GPS positioning. The alignment process involves synchronizing the local and global positioning of the device, which enables the use of UTM coordinate space for visualizations. LIDAR sensing allows for highly accurate ground estimation, eliminating the need for precise height estimation of the device itself. The concept of alignment, as previously described and illustrated in Figure 4.2, is solved using a root GameObject in Unity. The rotation of this object is updated via either GPS-based north estimation or compass-based estimation, depending on the availability of data. The root object is positioned at the current AR Camera position, and the corresponding real-world position is saved as a reference position. With the help of accurate GPS tracking, it is continuously checked whether the local tracking is moving in the same way in the reference space. In cases where the difference between these two tracks exceeds a certain threshold, re-initialization is triggered, the root object is repositioned at the current local position, and the reference position is updated. To correctly place objects in this reference space, positions are calculated through subtraction of the reference point, as given in Equation 4.2.

$$\begin{aligned} x_{dis} &= x - x_{ref} \\ y_{dis} &= y - y_{ref} \end{aligned} \tag{4.2}$$

When re-initialized, local positions of the geotagged objects are not valid anymore and have to be recalculated using the new reference position. Visual examples of this process can be seen later on in the evaluation chapter.

4.3 Evaluation of setup improvements

In this section, experiments performed with the improved setup are presented. The primary focus of the evaluation is on GPS-based north estimation, as the second part of continuous drift correction is relatively straightforward.

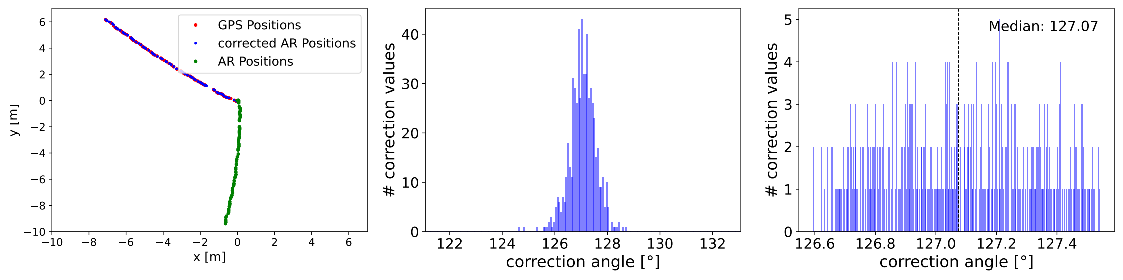
4.3.1 GPS Based North Estimation

To evaluate the implementation of GPS based north estimation, movements were tracked for different scenarios. To ensure that the evaluation was conducted in environments that are representative of SUE construction sites, all recordings were obtained on surfaces that are commonly found in urban environments, such as asphalt or concrete. In the most favorable scenario, a continuous RTK fixed mode was present. Further experiments with RTK float and local tracking without LIDAR sensor were performed. All data was collected using the presented Unity approach, utilizing a tablet device in combination with the CapsLoc sensor cube. North estimation was then performed offline to evaluate the impact of different properties of the algorithm.

Records under the best conditions

In these experiments, the setup described in Section 4.1.1 was used. The utilization of RTK fixed GPS positioning minimizes positioning error to 2 cm, thereby enabling the obtained position estimates to serve as a ground truth measurement. As previously demonstrated in Figure 4.4 (c), the accuracy of angle estimation is dependent on the positioning error and the distance between two points. In theory, a distance of 3 m between two GPS positions results in a maximum angle error of 0.8 degrees, satisfying requirements.

Best practice: As demonstrated in our experiments, the most favorable results are obtained when the user performs straight and smooth walks over a distance of at least 8 m. This is in line with the typical length of excavation sites, and therefore this type of movement can be performed in real-world scenarios. In the case of linear movement, local tracking drift is not a significant concern, as local tracking tends to be consistent in direction. This is because drift in the distance between two points does not affect angle calculation. Figure 4.5 illustrates an example of an optimal north estimation movement. In this calculation, a minimum distance of 1 m and a maximum distance of 3 m between points were used. The histogram in Figure 4.5 (c) shows a first standard deviation within the range of only one degree, indicating a highly accurate estimation. The theoretical maximum error is given by 2.3 degrees with a distance of 1 m which can be seen in the whole normal distribution in the histogram (b).



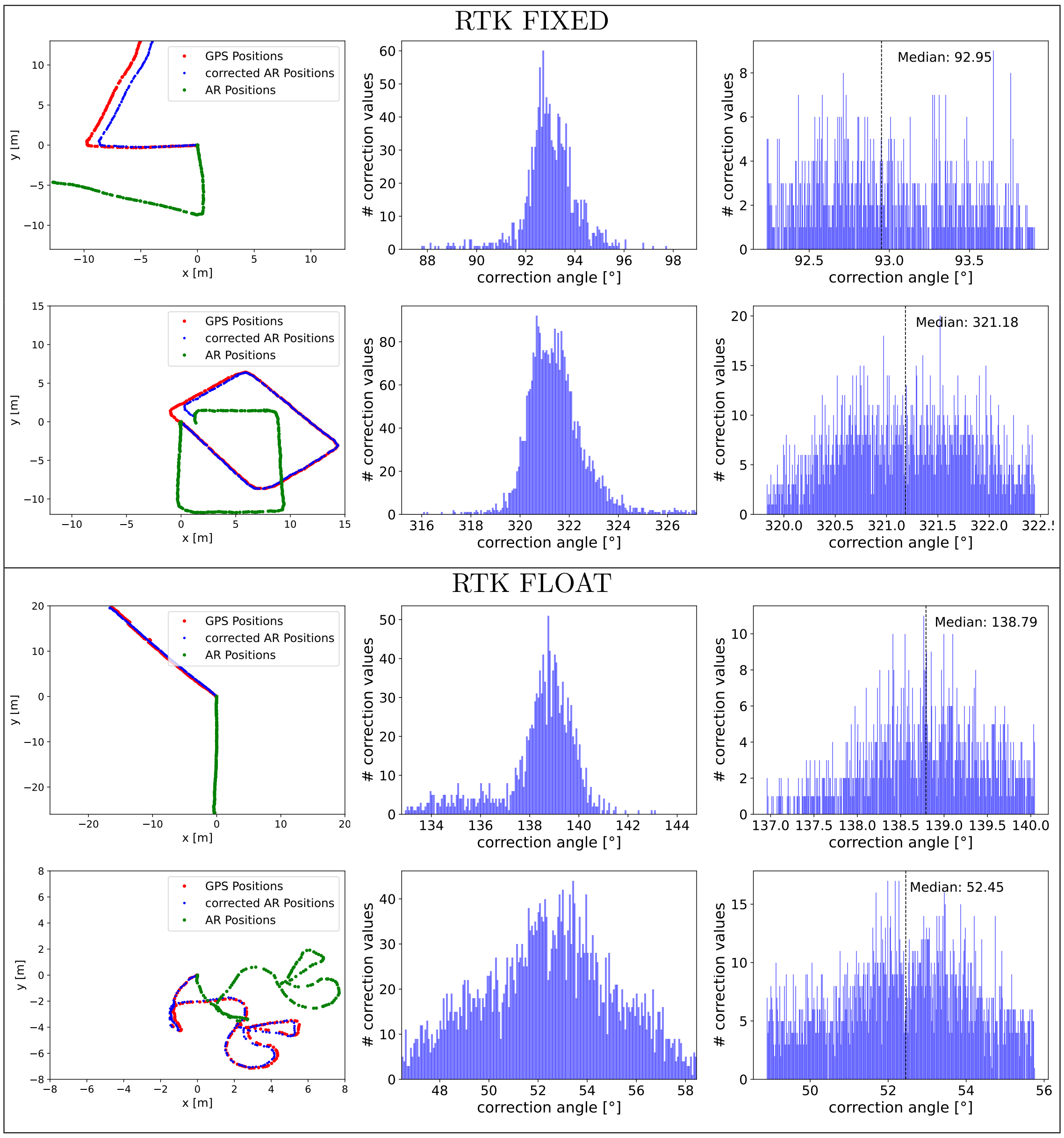
(a) Position Tracks (b) full Histogram (c) first standard deviation

Figure 4.5: Best practice north estimation: These plots show an example where positions from local and global tracking were recorded to evaluate the GPS north estimation algorithm. GPS positions (red) are assumed as ground truth, for corrected local tracking (blue), due to their high level of accuracy and use of north-aligned UTM coordinate system. Raw local tracking is visualized in green (a). The North correction algorithm uses minimum distance of 1 m between two points, yielding max error of 2.3 degrees, which can be observed in the histogram of all correction values (b). A maximum distance of 3 m should yield a maximum error of 0.8 degree, which is almost the case for corrections inside the first standard deviation (c).

How movements affect the result: The evaluation of various movements revealed that straight walks show the best performance. Additional examples of this can be observed in the top two Figures 4.6, which illustrates tracks of in the first row two linear movements over approximately 10 m and in the second row a rectangular shaped walk. These movements can be compared to walks around an excavation site. Furthermore, movements over 10 m are no problem in outdoor applications. The correction histograms show first standard deviation in the range of 2 degrees. Results from additional experiments are presented in Table 4.3. In the case of random movements, the normal distribution is significantly wider, with a first standard deviation distributed over more than 6 degrees. This is likely due to increased drift in local tracking. Nevertheless, the rotational alignment remains reasonable. Additionally, a scenario involving very fast straight movement was performed. Due to the limited number of positions, fewer correction values were calculated. However, a first standard deviation in the range of 1.5 degrees was still achieved. These results were obtained using the best currently available properties listed in Table 4.2.

Is RTK float mode sufficient for north estimation?

Since RTK fixed mode cannot be achieved in all circumstances, experiments with less accurate RTK float mode were performed. The setup was identical to that of the RTK fixed mode experiments, but in these tracks, fixed mode was not achieved and the receiver consistently operated in float mode. The maximum estimation error, given 20 cm GPS positioning accuracy, is 21 degrees with a distance of 1 m and 7.5 degrees with a distance of 3 m in theory. However, only the correction estimation histogram of the random walk track, shown in the last row of Figures 4.6 (b), had a normal distribution exceeding 20



(a) Position Tracks (b) full Histogram (c) first standard deviation

Figure 4.6: GPS-based north estimation experiments are illustrated. 2D positions in UTM coordinates (Y-axis=North) are shown in (a) where red represents GPS positioning, green represents original local tracking, and blue represents north-corrected local tracking. The distribution of correction values is depicted in a histogram (b) and corrections within one standard deviation are shown in (c). Linear movements with GPS RTK fixed positioning (2 cm accuracy) were performed in the first two tracks, while the last two tracks were performed with RTK float mode. The standard deviation in linear walk is sufficient for accurate north determination, while the broad standard deviation in the random walk track does not ensure accurate north correction.

degrees. The other tracks with linear movements had smaller maximum errors. Nonetheless, the rectangular track had a range of 6 degrees in the first standard deviation, as seen in Table 4.3. Notably, the straight walk track (shown in the third row of Figures 4.6) and the two straight motions track yielded a standard deviation of 3 degrees. Although the entire normal distribution is wider than in RTK fixed mode experiments, the results are still accurate enough for reliable north estimation.

Monocular local tracking

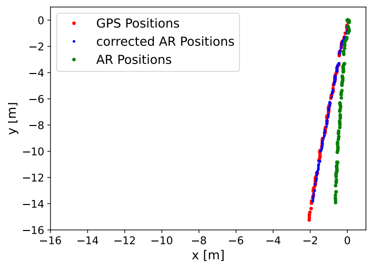
The final experiment involved mounting the CapsLoc sensor on a Samsung Tab6 and utilizing the same Unity application for recording. However, since the tablet runs on Android, local tracking was based on ARCore, which relies on a single RGB camera and the device's built-in orientation sensors. In this setup, the drift in local tracking was even more noticeable during random motions. Nonetheless, best practice still yielded good results. Figures 4.7 shows the recording of a linear motion, which showed a range of only 2 degrees in the first standard deviation.

Conclusions

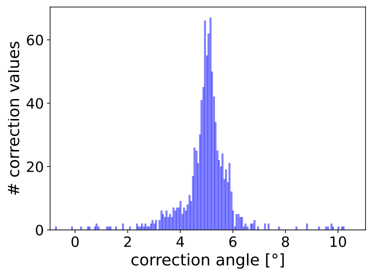
Sparse local tracking drift leads to best results, particularly in terms of high accuracy orientation estimation during linear movements. Initialization of rotation can be accomplished through a simple straight walk of 10 m. Excavations are typically of similar size, making the application of best practice in real-world scenarios manageable. Virtual spray marking can be initialized using compass north estimation and refined through GPS-based estimation during movement along markings. Plausible results of the algorithm with both RTK float and monocular-based local tracking further enhances its usability.

Performed Motion	GPS Mode	1σ [°]
straight walk	**fixed**	**<1**
two straight motions	**fixed**	**2**
rectangular walk	fixed	3
random walk	fixed	>6
fast straight walk	fixed	**2**
straight walk	**float**	**$\approx$ 3**
two straight motions	**float**	**$\approx$ 3**
rectangular walk	float	5
random walk	float	>7

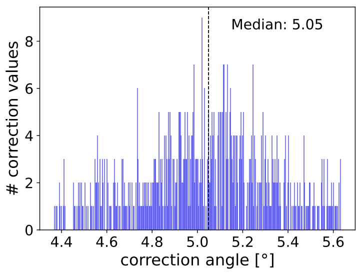
Table 4.3: North estimation results obtained from various movements and GPS fix conditions. The first standard deviation represents the range of the mean within the 70th percentile. Results with a value less than 3 degrees are considered as good estimates.



(a) Position Tracks



(b) full Histogram



(c) first standard deviation

Figure 4.7: This example was performed with monocular local tracking and GPS RTK fixed mode. 2D positions in UTM coordinates (Y-axis=North) are shown in (a) where red represents GPS positioning, green represents original local tracking, and blue represents north-corrected local tracking. Plots (b) and (c) show the histogram of calculated correction values. (b) shows the whole normal distribution, while (c) exhibits the first standard deviation. A variation of less than 2 degrees in the first standard deviation is sufficient for accurate north estimation.

4.3.2 Continuous Drift Correction

This evaluation examines tracking using the combination of north estimation and local drift correction. The optimal setup is employed and tracks are recorded to assess not only north alignment, but also the alignment of local and global tracking. A track featuring circular motion and significant local tracking drift serves as an example to demonstrate the effectiveness of the algorithm. This is illustrated in Figure 4.8, where the trigger for drift correction is set to 0.5 m. The four plots depict four successive instances when drift correction was activated. The red track represents GPS positions, serving as the ground truth. The blue track represents raw local tracking positions, while the black track reflects the positions after drift correction. The green crosses indicate the positions at which re-initialization was triggered and served as the current reference position until the next trigger point. As seen in the final frame (4), there is a high degree of overlap between the corrected positions and the GPS positions.

The experiments demonstrate that the requirements outlined in Table 3.1 [R1] are met. This leads to high-accuracy tracking for geo-referenced applications. In addition to applications such as spray paint marking, this capability also allows precise 3D reconstructions to tackle the problem of inaccurate utility documentation.

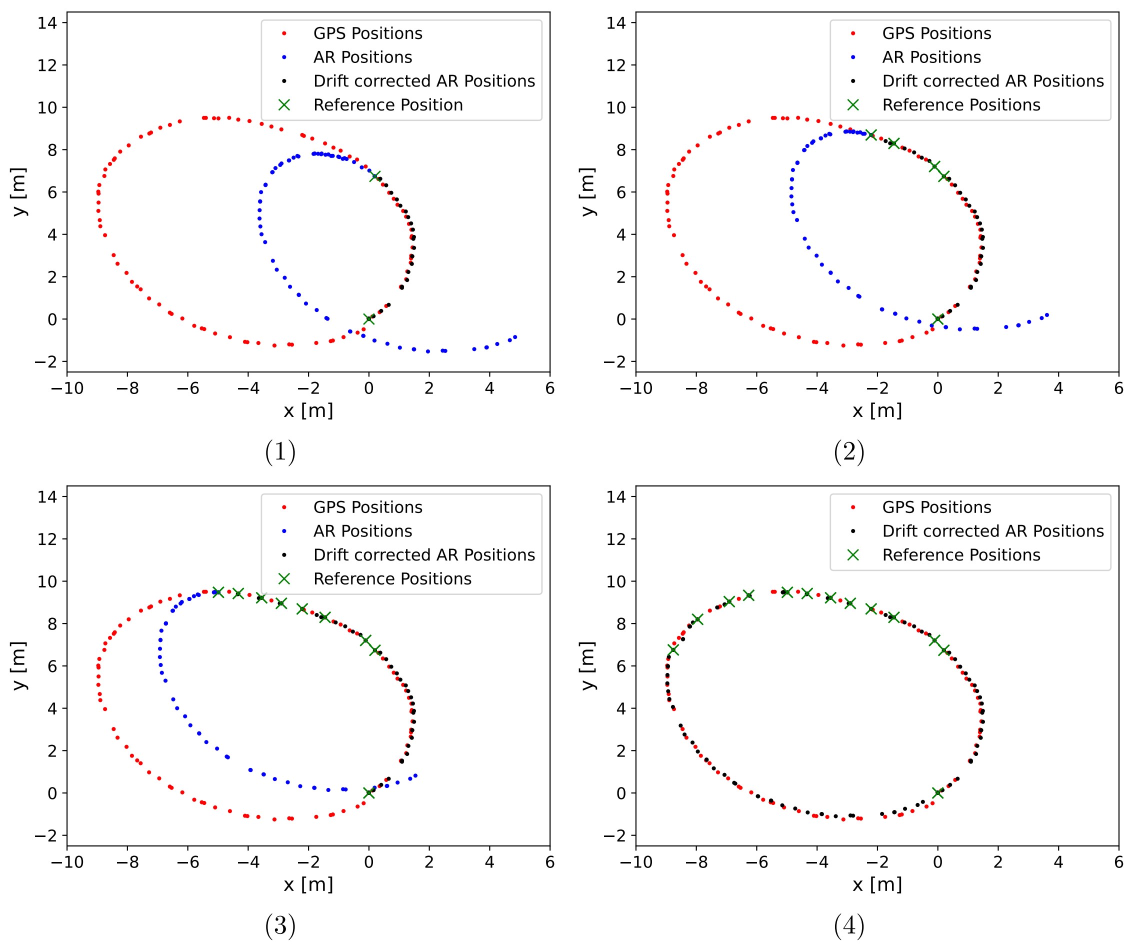


Figure 4.8: These four plots illustrate the chronological progression (1-4) of continuous drift correction applied to a track with substantial local tracking drift. The red points represent ground truth GPS positions, while the blue points depict the rotation-corrected local tracking positions. The green crosses indicate the positions used for re-initialization (drift correction), and drift correction is triggered when the drift exceeds 0.5 m. The black track subsequently represents the drift-corrected local tracking. The final outcome (4) demonstrates a high level of alignment between the corrected positions and the GPS positions, indicating an accurate alignment of local and global tracking.

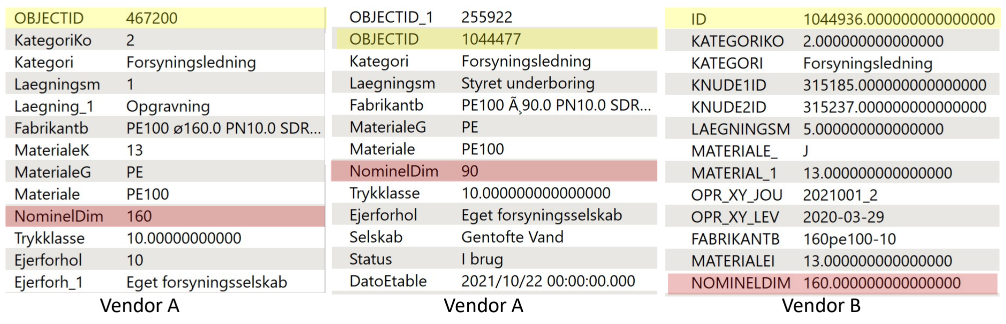
5 Semantic Segmentation using GIS Data

Contents

In this chapter, a segmentation and fitting approach for identification of Utility Assets, specifically pipes, in open excavation sites is presented. Initially, an interface is introduced for the acquisition of GIS data, which is subsequently standardized for further processing. This standardization is crucial for the alignment of segmentation results with the acquired data and the generation of accurate 3D models, providing better understanding of the excavation site for geo-referenced visualizations. The segmentation process is applied to 3D reconstructions of the excavation site, beginning at a user-specified starting point on the pipe. This is done via image renderings along the pipe, resulting in the identification of pipe center points. Since the segmentation only yields individual points and not a subset of the entire 3D reconstruction, the alignment problem can be solved through a simple least squares minimization. Furthermore, the GIS model can be directly aligned as a polyline structure to the segmentation points. This eliminates the need to create a 3D model explicitly for the alignment process.

5.1 GIS Data API and 3D model creation

The process of parsing Utility Asset documentation and 3D model creation is split into two independent components. The first component, an adaptable data parsing component, is required to handle non-standard data representation, which is then converted and structured for further processing. The second component, an expandable module, allows for the creation of 3D models in various styles and levels of detail.



OBJECTID	467200
KategoriKo	2
Kategori	Forsyningsledning
Laegningsm	1
Laegning_1	Opgravning
Fabrikantb	PE100 ø160.0 PN10.0 SDR...
MaterialeK	13
MaterialeG	PE
Materiale	PE100
NominelDim	160
Trykklasse	10.00000000000
Ejerforhol	10
Ejerforh_1	Eget forsyningsselskab

Vendor A

OBJECTID_1	255922
OBJECTID	1044477
Kategori	Forsyningsledning
Laegningsm	Styret underboring
Fabrikantb	PE100 Ã¸90.0 PN10.0 SDR...
MaterialeG	PE
Materiale	PE100
NominelDim	90
Trykklasse	10.000000000000000
Ejerforhol	Eget forsyningsselskab
Selskab	Gentofte Vand
Status	I brug
DatoEtable	2021/10/22 00:00:00.000

Vendor A

ID	1044936.000000000000000
KATEGORIKO	2.000000000000000
KATEGORI	Forsyningsledning
KNUDE1ID	315185.000000000000000
KNUDE2ID	315237.000000000000000
LAEGNINGSM	5.000000000000000
MATERIALE_	J
MATERIAL_1	13.000000000000000
OPR_XY_JOU	2021001_2
OPR_XY_LEV	2020-03-29
FABRIKANTB	160pe100-10
MATERIALEI	13.000000000000000
NOMINELDIM	160.000000000000000

Vendor B

Figure 5.1: Comparison of three sections of GIS metadata (screenshots taken in QGIS). The first two sections are from the same utility provider, but from different cities. Although the most important values are equally labeled, documentation is not even standardized within the same vendor. The third section shows documentation for the same utility type (water pipe) by another vendor. The important values of pipe ID (yellow) and the diameter (red) are highlighted here. Labels are not the same for different data provider.

5.1.1 GIS Data API

GIS data can either be predefined handcrafted plans in QGIS[1] or streamed from a data provider like ESRI[2]. However, pipe data is not provided in a standardized manner, varying in format and accuracy depending on the facility provider. As such, a versatile interface for data import must be implemented to handle different formats and data structures.

As seen in Figure 5.1, metadata is described differently by each provider. Pipes are typically represented using polylines, which are connected series of line segments, with each segment representing a line between two points. Some companies provide polylines in 3D, others provide height information in separate files, while some do not provide height information at all. Figure 5.2 illustrates our parsing pipeline, where different formats are

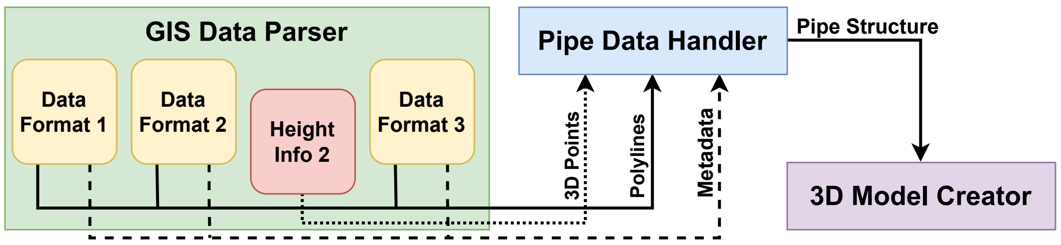


Figure 5.2: This sketch illustrates the process of creating a model using GIS data, which is provided in different formats (yellow) and includes polyline data for the pipes and associated metadata. The height information is often provided separately and not integrated in the utility documentation (red). The GIS Data Parser component (green) is responsible for parsing different formats and the Pipe Data Handler (blue) converts the information into a unified format that can be used for model generation. The height information is also interpolated to obtain valid 3D polylines. The 3D Model Creator component creates geometries from this data.

[1]https://www.qgis.org
[2]https://www.esri.com

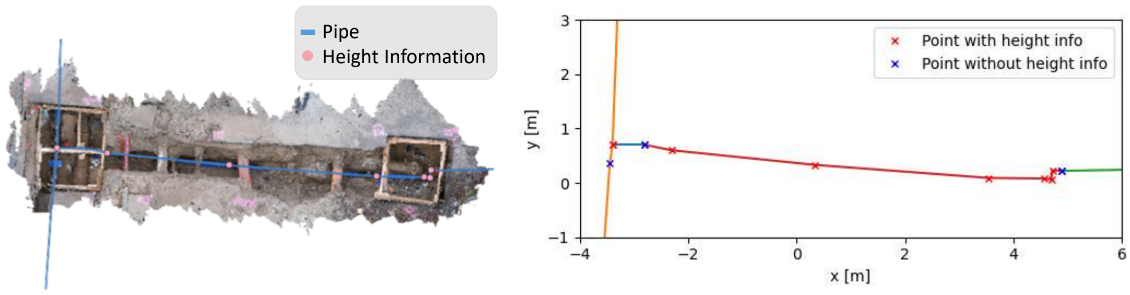


Figure 5.3: On the left is an extract from QGIS showing an excavation with overlapping GIS data. The blue lines represent pipes and the pink points represent the given height information. On the right, a plot of the same GIS data is shown, where individual pipes (with unique IDs) are visualized in different colors. The red dots indicate points with given height information, while blue dots indicate points for which no height information is provided. This plot represents the state of the Pipe Data Handler before height interpolation. To enable 3D model creation, all points must have appropriate height information.

read in the GIS Data parser, correctly assigned and passed to the Pipe Data Handler. The Pipe Data Handler creates a unified structure from this information and generates additional knowledge from the given data, which is needed for further processing. This knowledge includes correct linkages between all individual pipes and to complete height information throughout the entire structure.

GIS Data Parser

The focus of this component is on parsing data standards from different vendors. One approach to achieve this is through the use of QGIS scripting to standardize different formats, however, this requires the installation of QGIS and manual execution by the user. As an alternative, we have integrated this functionality directly into the pipeline for GIS data parsing and 3D model generation. Despite variations in overall data structure, the placement of pipes is consistently represented as polylines in at least 2D (UTM) space. To enable the creation of 3D models, it is crucial to correctly link metadata, such as pipe radius, material, heights, and other attributes, which may not be labeled uniformly. Additionally, height information may also be provided through a series of single measurement points in an additional file or database table, as seen in Figure 5.3(a). This information must be parsed and linked correctly to the corresponding polyline points.

This component loads pipe structures within a defined radius around a reference point. Polylines that only partially fall within the area of interest are cut off using this circle. The polyline points are transformed from global UTM coordinates to relative coordinates using the reference point. Metadata is parsed and collected in a dictionary with standardized labels. Each pipe is then passed individually to the pipe data handler as an array of points linked with their unique ID and additional metadata of interest.

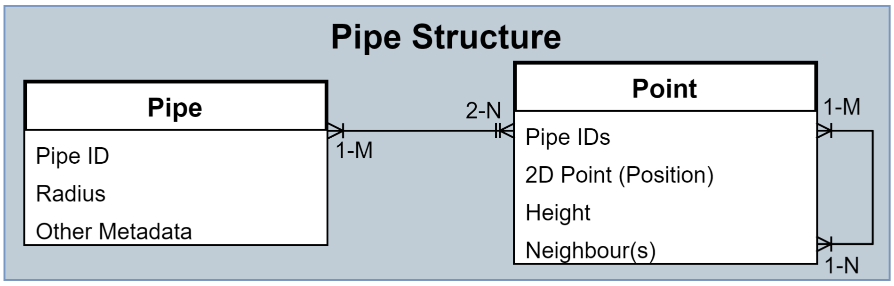


Figure 5.4: This Entity Relationship Diagram illustrates the connections within the data structure generated by the Pipe Data Handler. Each pipe is identified by a unique pipe ID and contains pipe-specific metadata, such as radius, material, etc. Pipes consist of at least two and up to any number of points. Each point must be part of at least one pipe, but can also belong to multiple pipes, such as in the case of an intersection. Points also contain a 2D position and an optional height. The points are linked to one another to represent the correct ordering. Therefore, each point must have at least one neighboring point, but in the case of a crossing it can have any number of neighbors.

If height information is not included in the metadata or polylines, a separate height information file is parsed. Instead of polylines, single points are specified. The unique pipe ID is used to correctly assign height measurements. The height points are then passed on to the Pipe Data Handler.

Pipe Data Handler

The Pipe Data Handler generates uniform pipe structures for further processing. Data structure is illustrated in Figure 5.4. Pipes contain specific metadata, including a unique Pipe ID. Each pipe can have any number of points, but must have at least two. Points have at least one neighboring point, and can be connected to multiple points to create pipe structures. Points must be associated with at least one pipe. Branch points are linked to all associated pipes. Points store their position as a 2D vector and their height information, which is not mandatory and can be left empty initially. The links between points beyond individual pipes enable traversal of the entire pipe structure, which is necessary for height interpolation. The pipe structure is constructed incrementally by successively adding individual pipes (from the GIS Data Parser), each consisting of an array of points, a unique ID, and a dictionary of metadata. The aim is to create a link between the individual pipes and to avoid duplicated 3D points. Therefore, before a point is created, it is checked if a point from another pipe already exists at the same position. A small epsilon ($\epsilon > 1cm?$) serves as the maximum distance to identify point matches. If the point already exists, links to the new pipe and neighboring points are added. Height information, if available in a separate file, can be added individually afterwards. In the best case, height measurements are assigned to a pipe using the unique pipe ID, otherwise, all points must be searched to determine to which point the height measurement belongs.

At this stage, a fully linked pipe system is established. But in most cases, only isolated points are provided with height values, as shown in Figure 5.3 (b). Therefore, the remaining elevation values have to be estimated through interpolation. The linked point structure is used to go through the pipe system using a recursive algorithm, as outlined in Algorithm 1. The algorithm starts at any endpoint (`p_current`), with the parameter for the previous point (`p_prev`) passed as a `NULL` value and `to_interpolate` passed as an empty list.

5.1.2 Implementation Details

All three components, GIS Data Parser, Pipe Data Handler, and 3D Model Creator, are implemented using Python. For mobile operation, this system runs on a server and provides 3D models through web requests. Since the GIS data is only available in the shapefile format (ESRI data format), the Python Shapefile Library (PyShp)[3] is used for file reading and parsing. If data is provided through a database, the PyShp library is replaced with appropriate database requests. The data is in a similar format as ESRI also provides an online service.

5.1.3 3D Model Creation

By standardizing the received information with the Pipe Data Handler, we can generate 3D geometry. As previously stated, the goal is to make model creation as independent and interchangeable as possible. To achieve this, we chose to implement the geometry creation part using ESRI's CityEngine[4], a powerful tool for automatically generating 3D geometries from simple 2D or 2.5D data through procedural modeling. Rule files determine how to generate 3D models from certain 2D/2.5D shapes. These shapes can be given attributes, such as the radius of the pipes in our case. The rule files enable easy exchange or extension of the 3D model generation. Additionally, CityEngine allows the use of existing CAD models of the individual pipe parts for automated 3D geometry generation, resulting in the most realistic model possible.

[3]https://github.com/GeospatialPython/pyshp

[4]https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview

Algorithm 1 Recursive height interpolation

```
 1: function INTERPOLATE_HEIGHT(p_current, p_prev, to_interpolate)      ▷ Point,
    Point, List<Point>
 2:     if p_current has height information then
 3:         if to_interpolate not empty then
 4:             if to_interpolate[0] has height and to_interpolate[-1] has height then
 5:                 INTERPOLATE(to_interpolate)
 6:             else
 7:                 Set all Points in to_interpolate to height of to_interpolate[0]
 8:             end if
 9:         end if
10:         clear to_interpolate
11:     end if
12:     add p_current to to_interpolate
13:     p_neighbours ← neighbours of p_current
14:     if length of p_neighbours == 1 then           ▷ p_prev == p_neighbours[0]
15:         if p_neighbours[0] has no height then
16:             Set all Points in to_interpolate to height of to_interpolate[0]
17:         end if
18:         return              ▷ end of the recursion in this branch of the pipe structure
19:     end if
20:     for all p_i ∈ p_neighbours do
21:         if p_prev == p_i then
22:             Skip this Point                              ▷ coming from this direction
23:         end if
24:         if p_i is first neighbour then
25:             INTERPOLATE_HEIGHT(p_i, p_current, to_interpolate)
26:                                          ▷ Interpolation of this Branch is finished here
27:             assert p_current has a height value now
28:         else
29:             add p_current to to_interpolate ▷ to_interpolate is empty, p_current
    first point in next branch to interpolate
30:             INTERPOLATE_HEIGHT(p_i, p_current, to_interpolate)
31:         end if
32:     end for
33: end function
```

```
// radius attribute, default value if not specified
attr radius = 0.4

@StartRule
Pipe-->//to align cylinder correct, GeometryScope is aligned to longest side
        alignScopeToGeometry(yUp,0,longest)
        center(xyz)
        //cylinder rotated 90 deg because would go up not on ground
        rotateScope(0,0,90)
        //create cylinder along shape primitiveCylinder(sides,radius,height)
        primitiveCylinder(32, radius ,scope.sy)
```

Table 5.1: Rule code for CityEngine used for creation of the simple cylindrical 3D models.

CityEngine is required to compile the rule files, but thanks to PyPrt[5] it can be executed directly from Python without the need of a CityEngine installation. For model creation, polynomial shapes are generated, which are then passed to the PyPrt library for further processing. To achieve this, all points in each pipe are iterated and a rectangular 2.5D polynomials are created from every two consecutive points. All these shapes and their attributes, such as the radius in this case, are passed to the library as a list. Additionally, the compiled rule files to be used are specified, and an encoder is selected. Several encoders are available, we use the `OBJEncoder`. Rules are linked to individual polygons to determine how 3D geometry is created. In the simplest case, the pipe consists of a set of straight polygons from one point to another, with only the radius as an additional attribute. Therefore, more complex pipes are subdivided into straight lines, and the 3D structure is created as a cylinder along the polygon with the specified radius. The Rule file used for simple pipe geometry is illustrated in Table 5.1.

During initial tests, we observed that strange artifacts can appear when using real UTM coordinates, which can become very large numbers, to create the models. This can be seen in Figure 5.5 (top left). Therefore, it is recommended to work in a smaller referenced area. Examples of generated 3D models are shown in Figure 5.5. The models match well with the pipes in the referenced 3D scans. Height interpolation appears to be successful, as all models match perfectly in height with the ground truth 3D scans. This implementation satisfies Requirements [R2.3] and [R2.4], as specified in Table 3.1.

[5]https://github.com/Esri/pyprt

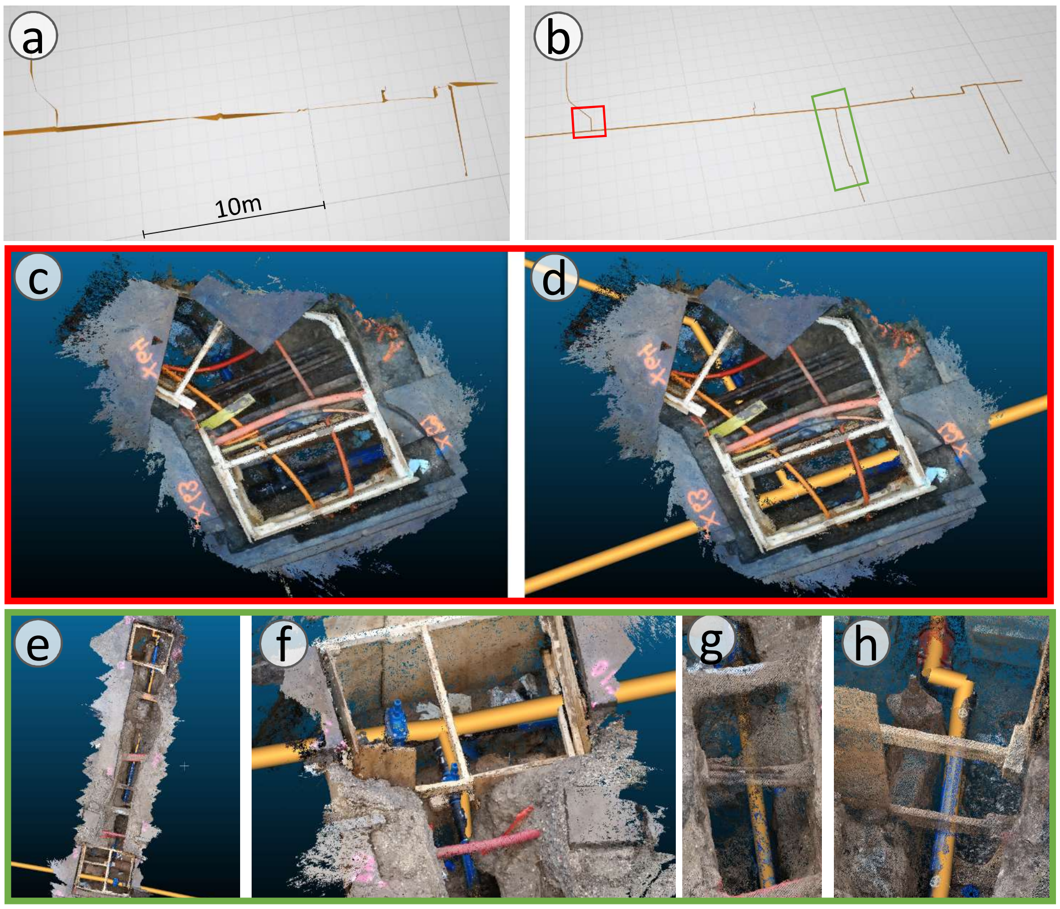


Figure 5.5: Image (a) shows GIS data model created in UTM space where artifacts arise, while on (b) the same data is used in a local space. Images (c-h) are examples withing the ground truth dataset. Those excavations are part of the model shown in the top image. In Image (d) created 3D model is superimposed into the 3D scan. Images (e-h) are four sections from an excavation where the generated model was superimposed on the 3D scanned data. The created geometry fits over the real pipes. These examples show heights in the created model are correct, indicating that height interpolation logic works.

5.2 Segmentation

The proposed segmentation approach utilizes image-based segmentation on renderings of a 3D reconstruction obtained via virtual cameras. The segmentation results in image space are then projected into the 3D space. The advantage of this approach is that 3D representation format does not matter and the 3D scan noise is usually negligible. The concept is illustrated in Figure 5.6, where the thick red line, L, represents a pipe in the 3D scene. Images are rendered from various poses around the pipe. Camera positions are indicated by C_1 and C_2. Segmentation results in image space are denoted by both red lines l'_1 and l'_2. Two arbitrary points on each segmented line can be taken for later 3D projection (x_1, x_2 and y_1, y_2). Rays from the camera position through the segmented

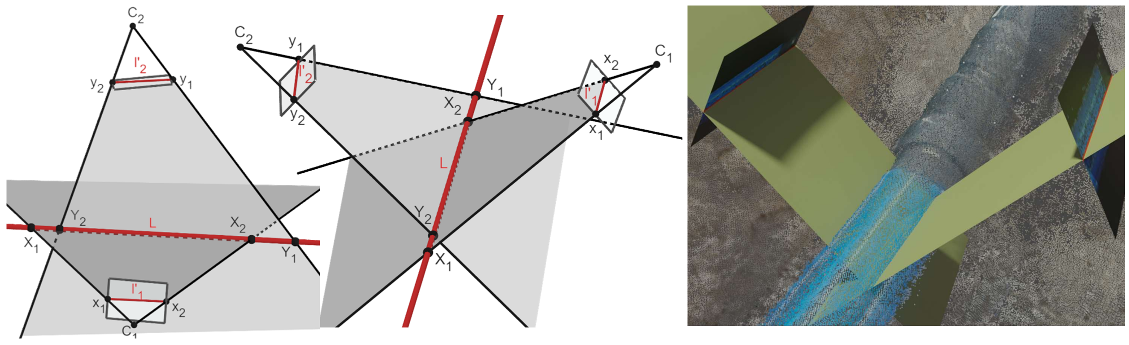


Figure 5.6: The segmentation is performed in 2D image space, rendered from different views. The left and middle sketch describe the geometry behind the approach and the right image is a 3D rendering to illustrate it in a real setup. From different camera positions (C_1 and C_2) the pipe (L) is segmented in image space (l'_1 and l'_2). Rays from the camera through any point in the segmented line of the image space (x_1,x_2,y_1,y_2) should hit the pipe in 3D space (X_1,X_2,Y_1,Y_2). We use this fact to span a plane for each view over the camera position and the segmentation points in the image plane (gray and light gray). Intersection of these planes yields a line in 3D space representing the pipes center.

points in image space should pass through the real pipe in the 3D scene, represented by X_1, X_2, Y_1, and Y_2. To project segmentation from image space to 3D space, planes are created from each camera position and the corresponding segmented points in the image. The intersection of these planes creates a line that should be in the center of the pipe.

5.2.1 Algorithmic Implementation

In order to implement our proposed segmentation concept, we describe the procedure in nine distinct steps. Additionally, we provide insight on the implementation of the crucial steps within the algorithm.

1. Start the segmentation process using a point on the pipe and the color of the pipe (there can be several colors).

2. The next item from the list of positions to check (3D positions where a segmentation cycle should be performed) is selected. If no estimate of the pipe direction is provided or if the last estimation did not result in a correct segmentation, an estimate of the pipe direction (Z/height-axis angle) is made at this stage. To achieve this, the pipe is rendered from a top-view perspective and a rectangular fit is performed on the color mask. The pipe heading angle is approximated by the rotation of this rectangular fit.

3. With the direction estimate, six images are rendered from different perspective around the pipe. The camera positions are determined based on the specified distance and angles ranging from -30 to 30 degrees.

4. Pipes are segmented in image space for each perspective. If not at least 4 out of 6 views are successful, it is assumed that there is no pipe at the current position, and the segmentation process for that branch is terminated. If the segmentation fails, two retries are triggered. The first retry skips the current position and moves on to the next position (+`walk_distance`). If nothing is found, the current position is reused, but the heading angle is discarded to reinitialize it. If still nothing is found, the segmentation process is considered finished. If there are positions for possible junctions, these are used to initiate new segmentation processes from these points (back to step 1) in order to distribute points of different pipe branches into individual groups.

5. In the contour image of the color segmentation, it is checked whether a junction is present. If the junction is confirmed in 4 out of 6 views, a 3D position on the outgoing branch is saved for later initialization of the segmentation of that branch.

6. Planes are calculated using two points on the segmented line in image space and the corresponding camera position.

7. The planes are intersected to calculate a line in 3D space, the center point of the pipe at the current segmentation position and a direction from that point are determined.

8. Based on the calculated new center point and direction, the next point for segmentation is determined and added to the list of points to be checked. In the initial step, the next point is used in both directions of the pipe, then in one direction to ensure that nothing is checked twice.

9. Go back to Step 2).

Rendering

Rendering can be performed with any Rendering Engines. The principle is always the same, but it is crucial to ensure that the camera position is calculated in the correct coordinate system. In this work, Unity is used as an external rendering engine, and it is important to note that Unity uses a left-handed coordinate system. The `LookAt` functionality is used to calculate the pose of the camera, which requires three parameters: `Center`, the point where the camera should look, `Eye`, the position of the camera itself, and `Up`, which describes the vector up in the rotation of the camera. The relevant segmentation attributes that affect rendering are: image height and width, field of view, and camera distance. The image size can be kept relatively small to improve the segmentation performance in the image space. The field of view is somewhat dependent on the distance of camera (`Eye`) to the pipe (`Center`). The closer the camera, the larger the field of view can be. A significant advantage of the virtual 3D scan representation is that we can use near and far clipping to selectively hide the background and foreground in the rendering. This way, only the pipe is usually rendered, and possible disturbances are avoided during the rendering process. For the first directional estimation of the pipe, the first image is shot from the top. Once the orientation is known, other images are taken in a quarter-circle around the pipe.

Segmentation in Image space

Segmentation in image space is based on a color-based approach. First, a basic image blur is applied to reduce potential noise. Color-based filtering is performed in the HSV color space, as it is the most suitable. Depending on the quality of the input, a simple range filter in the Hue range will suffice in the quick procedure. Otherwise, a region growing algorithm is used, which segments regions based on the similarity of adjacent pixels in a certain range of the given color. The region growing approach only needs to segment a region where the starting point is given by the current step in segmentation, either centered in the image or centered on the side of the last segmentation position. Morphological transformations can be used to ensure a closed segmentation region. The output of the color filter is shown in Figure 5.7 (a). To initialize pipe heading direction, the top view rendering is used. It is assumed that the starting position is at a straight section of pipe and not at an intersection. After color filtering, a minimal rectangle is fitted around the outline of the pipe. The angle between the rectangle and the bottom of the image is used as an estimate of the pipe direction. This angle can now be used to walk along the pipe for segmentation.

The next step is to determine the center of the pipe. This is achieved by using the distance transform function, which assigns grayscale intensities to pixels based on their distance to the nearest boundary, as shown in Figure 5.7 (b). The brightest values in the image indicate the center of the pipe. To ensure that the central estimate is not affected by outliers or intersections, a binary filter is applied which only returns true if the values above and below the center of the filter window are equal, as shown in Figure 5.7 (c). This can be done because the pipe is assumed to be relatively vertical in the image. The result of this filter is a vertical line that corresponds to the central estimate of the pipe. A 2D line fitting algorithm can now be performed on this line estimate (OpenCV provides this functionality), as seen in Figure 5.7 (d). Using the fitted line in image space, any two points on this line can be used to construct a plane in 3D space. In this case, we use the two points on the left and right edges of the image.

Reproject Segmentation to 3D space

The projection of image space segmentation into the 3D scene is achieved by defining a plane through the camera position and two points on the segmented line in the image. This process is repeated for all six rendered frames surrounding the pipe. The intersection of all planes create a 3D line in the middle of the pipe.

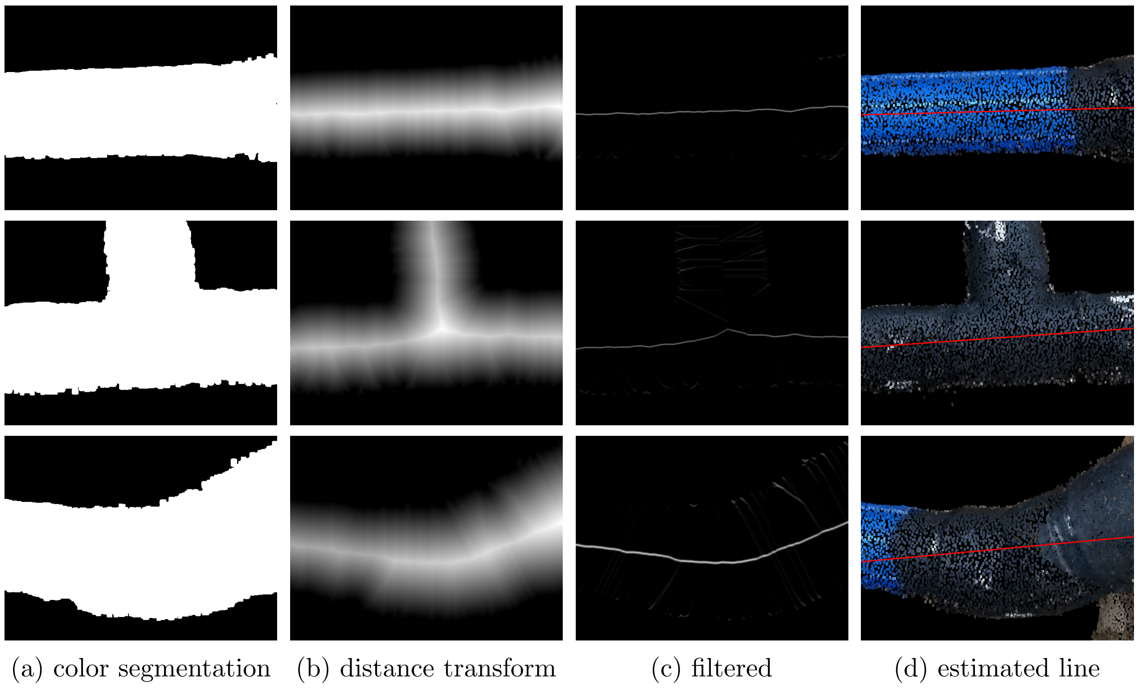


Figure 5.7: Main steps of segmentation in image space: (a) Color based segmentation to mark pipe area in the image. (b) Perform distance transform to highlight the middle part of the tube. (c) Individual filtering to detect horizontal line. This is used to eliminate disturbance of junction parts or other noise in the color segmentation. (c) Visualization of the segmentation result in the given examples. The red lines are used for reprojection into 3D space. Three different scenarios are shown, the top row depicts a straight pipe, the second row shows a T-junction, and the third row depicts a slightly curved pipe part.

The initial step involves the computation of two rays starting from the camera position (p_{cam}) and passing through two points on the segmented line in image space. To obtain a single ray, the image space pixels (p_x and p_y) are transformed into camera space coordinates using equation 5.1. This calculation takes into account both the image dimensions (width: w_{img}, height: h_{img}) and the camera's vertical field of view (fov_v). First the aspect ratio ($aspect\_ratio$) is calculated and then the position of the pixels in the camera space (x and y).

$$\begin{aligned} aspect\_ratio &= \frac{w_{img}}{h_{img}} \\ x &= (2 * \frac{p_x + 0.5}{w_{img}} - 1) * tan(\frac{fov_v}{2} * \frac{\pi}{180}) * aspect\_ratio \\ y &= (1 - 2 * \frac{p_y + 0.5}{h_{img}}) * tan(\frac{fov_v}{2} * \frac{\pi}{180}) \end{aligned} \tag{5.1}$$

The camera position (p_{cam}) and the pixel position (p_{pixel}) in camera space are represented by equation 5.2. As the image plane is placed at a distance of one unit from the camera, the z-coordinate is initialized to -1.

$$\begin{aligned} p_{cam} &= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \\ p_{pixel} &= \begin{pmatrix} x \\ y \\ -1 \end{pmatrix} \end{aligned} \tag{5.2}$$

The points in camera space are then transformed to world space using the camera view matrix (M_{view}), yielding the camera world position ($p'cam$) and the pixel world position ($p'pixel$). The ray (ray_i) is obtained by subtracting these two points, as per equation 5.3.

$$\begin{aligned} M_{view} &= \begin{pmatrix} R_{view} & t_{view} \\ 0 & 0 \end{pmatrix} \\ p'_{cam} &= M_{view} * p_{cam} \\ p'_{pixel} &= M_{view} * p_{pixel} \\ ray_i &= p'_{pixel} - p'_{cam}; \end{aligned} \tag{5.3}$$

The next step is construction of a plane in 3D space, which can be represented by the equation $Ax + By + Cz = D$. The calculation of this plane from the two rays and the camera position (p'_{cam}) is described in equation 5.4. The normal vector of the plane is obtained by taking the cross product of the two rays (ray_1 and ray_2). This normal vector provides the values of A, B and C in the plane equation. The value of D is calculated by taking the dot product of the normal vector (n_i) and the camera position vector ($p'cam$).

$$\begin{aligned} &\mathbf{n_i} = \mathbf{ray_1} \times \mathbf{ray_2} \\ &D = \mathbf{n_i} * p'_{cam} \\ &Plane => Ax + By + Cz = D \quad \text{where} \quad \mathbf{n_i} = \begin{pmatrix} A \\ B \\ C \end{pmatrix} \end{aligned} \tag{5.4}$$

The straight forward solution to cut two planes would be to equate their formulas 5.5.

$$A_1x + B_1y + C_1z - D_1 = A_2x + B_2y + C_2z - D_2 \tag{5.5}$$

This solution provides the complete 3D line, however we are interested in a single 3D Point representing the center-point at the current segmentation step. An alternative approach is to compute the direction of the line and a point on the line that has the minimum distance from a given reference point using Lagrange multipliers[67]. In this case, the two normal vectors of the planes ($\mathbf{n_1}$ and $\mathbf{n_2}$) and two points on the planes, in our case the camera positions (p_1 and p_2), are used to define the planes. The intersection line of both planes is represented by a point (p) and a direction vector ($\mathbf{n}$). As the point (p) must lie on both planes, two Lagrange constraints are given by equation 5.6.

$$\begin{aligned}(p - p_1) * \mathbf{n_1} = 0\\ (p - p_2) * \mathbf{n_2} = 0\end{aligned} \tag{5.6}$$

The point (p) should now be as close as possible to the reference point (p_0). Then the distance equation 5.7 is used for minimization in Lagrange multipliers.

$$\|p - p_0\|^2 = (p_x - p_{0x})^2 + (p_y - p_{0y})^2 + (p_z - p_{0z})^2 \tag{5.7}$$

With the use of the objective function 5.7 and the two constraints 5.6, Lagrange multipliers can be applied effectively. In this case, the midpoint between the two camera positions (p_1 and p_2) is chosen as the reference point (p_0), as given by equation 5.8.

$$p_0 = \frac{p_1 + p_2}{2} \tag{5.8}$$

These five linear equations represented in matrix form 5.9 follow, in which λ and μ are the Lagrange multipliers of the two constraints. Solving this system of equations yields the desired point p.

$$\begin{pmatrix} 2 & 0 & 0 & n_{1x} & n_{2x} \\ 0 & 2 & 0 & n_{1y} & n_{2y} \\ 0 & 0 & 2 & n_{1z} & n_{2z} \\ n_{1x} & n_{1y} & n_{1z} & 0 & 0 \\ n_{2x} & n_{2y} & n_{2z} & 0 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} 2p_{0x} \\ 2p_{0y} \\ 2p_{0z} \\ p_1\mathbf{n_1} \\ p_2\mathbf{n_2} \end{pmatrix} \tag{5.9}$$

The direction of the intersection line is calculated using the cross product of the two normal vectors 5.10.

$$\mathbf{n} = \mathbf{n_1} \times \mathbf{n_2} \tag{5.10}$$

At each segmentation step, all planes are intersected and an average point and direction are calculated. The point of intersection should represent the center of this pipe section. The direction found is used for the next segmentation steps. This completes the reprojection for this segmentation step. An alternative approach could be to use an optimization algorithm to directly compute the optimal estimate of the intersection of all planes, which may yield more precise results.

How to identify junctions

In reality, there are only two types of junctions, namely T-junctions and cross junctions. Curves that are sharp in nature are also detected in this step. The detection of junctions is done by analyzing the contours of the colored mask.

1. Calculate contours of the color mask image.
2. Go through the upper part of the contour:
   - 2.1 Search for a sequence of pixels with increasing y values until the maximum image height (`img_height`), followed by a sequence of decreasing y values until the starting point of the initial increasing sequence is reached.
   - 2.2 Upon identifying this pattern, mark it as a potential upward-facing junction. If only increasing or decreasing part of the pattern was found, mark it as potential curve and go to step 3.
   - 2.3 Determine the midpoint between the last point of the increasing sequence and the first point of the decreasing sequence, which serves as the center point of the detected intersection.
   - 2.4 Estimate the direction angle by calculating the angles using the first and last points of the two sequences, and by taking the average of these two angles.
3. Go through the lower part of the contour:
   - 3.1 Search for a sequence of pixels with decreasing y values until 0, followed by a sequence of increasing y values until the starting point of the initial decreasing sequence is reached.
   - 3.2 Upon identifying this pattern, mark it as a potential downward-facing junction. If only increasing or decreasing part of the pattern was found, mark it as a potential curve.
   - 3.3 If possible junction:
     - 3.3.1 Determine the midpoint between the last point of the decreasing sequence and the first point of the increasing sequence, which serves as the center point of the detected intersection.
   - 3.4 If possible curve:
     - 3.4.1 Determine the midpoint between the last two points of the found sequences, which serves as the center point of the detected intersection.
   - 3.5 Estimate the direction angle by calculating the angles using the first and last points of the two sequences, and by taking the average of these two angles.

This check is always performed in the initial top rendering. If at least one branch is identified in this step, the process is also applied to all other views in the current segmentation step. Figure 5.8 provides two examples of the segmentation process. The four points (P_1 - P_4) depict the two identified sequences of increasing and decreasing values in the contour that define a junction. P_{mid} is given as the midpoint

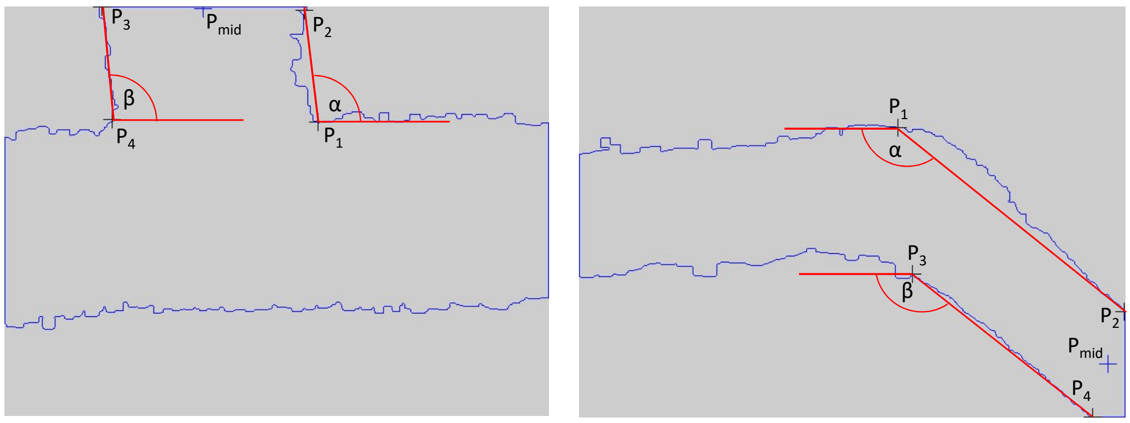


Figure 5.8: Two results of the junction detection algorithm. On the left image, a T-junction is identified and on the right image, a sharp curve is detected. The blue colored segmentation contours are traversed to identify sequences of increasing and decreasing pixel values. The starting points (P1 and P2) and ending points (P3 and P4) of these sequences are used to compute a new point (P_{mid}) and the heading direction ($\frac{\alpha+\beta}{2}$) of the outgoing pipe branch, which serves as initialization for the segmentation of the outgoing branch.

between, whether P_2 and P_3 or P_2 and P_4, depending on the detected junction type. The angles α and β indicate the angle at which the identified branch leaves the pipe. After identifying possible midpoints in all images, they are projected into 3D space.

A ray (**ray**) is shot from the camera (p'_{cam}) through the crossing point P_{mid} as previously described in equations 5.1 to 5.3. By utilizing the Symmetric Form representation of a line in 3D, as outlined in equation 5.11, we can compute a point on the line at a specific height. In this form, the point on the line corresponds to (x_0, y_0, z_0) and the direction corresponds to (a, b, c). In our case, the point is the camera position (p'_{cam}) and the direction is the calculated ray (**ray**).

$$\frac{x - x_0}{a} = \frac{y - y_0}{b} = \frac{z - z_0}{c} \tag{5.11}$$

Using the previously estimated height of the current segmentation step, a point is calculated on the ray up to this height 5.12. This point should represent the midpoint in the intersection found.

$$\begin{aligned} y &= center_y \quad \text{where y is height axis in unity} \\ x &= \frac{y - p'_{camy}}{ray_y} * ray_x + p'_{camx} \\ z &= \frac{y - p'_{camy}}{ray_y} * ray_z + p'_{camx} \end{aligned} \tag{5.12}$$

The final point in 3D space is computed as an average of all the points calculated from the various camera positions. The calculated direction angles of the identified intersection are also averaged. These values are used to initialize the segmentation algorithm for the outgoing pipe branch. In this case, the heading estimate for the initial top rendering is already known, and the direction from which the intersection originates is also provided, allowing the segmentation process to proceed in a single direction.

Software tools

The segmentation is implemented as a stand-alone C++ library. As external libraries OpenCV[6] is used for processing image data and RapidJson[7] for parsing input and return values. Additionally, two implementations of rendering are available. The first option utilizes Open3D[8] for background rendering, which requires the use of Open3D as an external library. The second option is the possibility that rendering is done by the application calling the segmentation. In this case, images are requested from the library via callbacks. The latter approach has the advantage of not requiring a separate render platform if the calling program has already loaded the 3D model.

Unity Integration

For the implementation of segmentation in Unity, the potree Unity library [9] is used to represent pre-recorded point cloud data. However, since the segmentation is performed in image space, the representation of the 3D data can be managed at will. Unfortunately, the option of obtaining 3D data directly from the LIDAR sensor is currently not straightforward. In the current state (iOS 16), it is only possible to acquire depth data without associated color information. While real-time meshing is available, it is not textured at all. There are apps available in the store that enable textured 3D scanning with the LIDAR sensor (e.g., 3D Scanner App [10]), but implementing this functionality in your own Unity app would require additional development effort.

The interaction of the segmentation library and Unity is done via a callback from the C++ level. The segmentation script is assigned to its own GameObject. During initialization, the GameObject creates a new camera with the desired parameters for the segmentation, such as the vertical field of view and the near and far clipping planes. To share image data with the C++ library, a byte array of size `img_width` * `img_height` * 3 (RGB) is created and the pointer to the array is shared. Additionally, a rendering function is specified at Unity level and passed as a function pointer to the C++ level. The segmentation process is executed as a separate thread to avoid blocking the main Unity thread. The `RenderCallback` is not called within the main thread, but Unity

[6]`https://github.com/opencv/opencv`
[7]`https://github.com/Tencent/rapidjson`
[8]`https://github.com/isl-org/Open3D`
[9]`https://github.com/SFraissTU/BA_PointCloud`
[10]`https://3dscannerapp.com/`

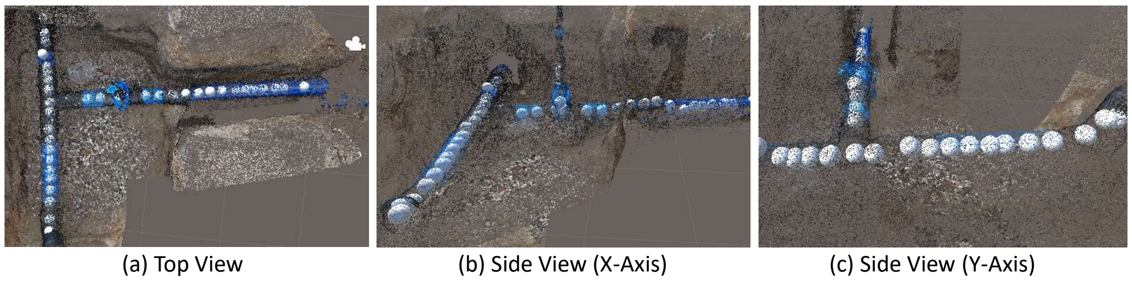


Figure 5.9: These are three screenshots of a segmented Point Cloud in Unity. Segmented midpoints are visualizes as white Spheres. The samples show results from all three Axis, (a) Top view, (b) X-Axis, (c) Y-Axis (in real world, but actually Z-Axis in Unity)

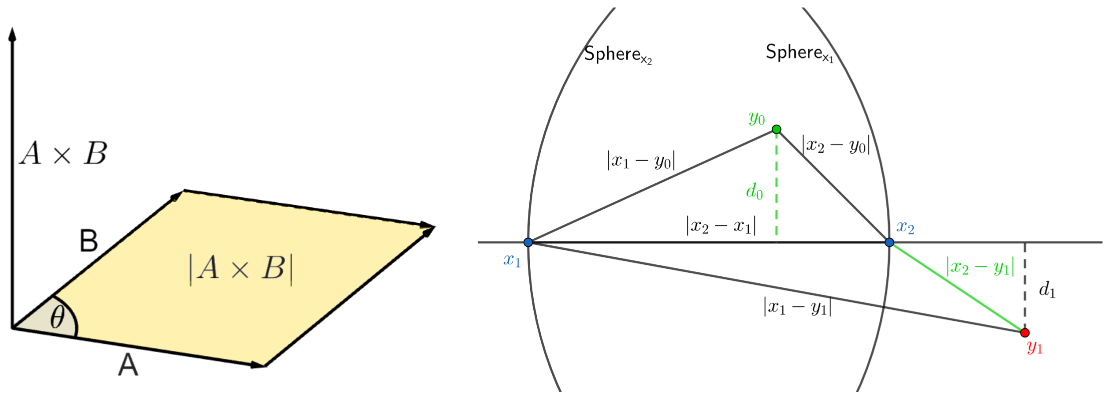
only allows the main thread to manipulate objects in the scene or render an image. Therefore, moving the camera and rendering the image must be done in the main thread. So there must be a synchronization between the main thread and the segmentation thread. This is achieved by using two action queues and an `EventWaitHandle` (both .Net components). One queue is processed in `Update` and one in `LateUpdate`. Transformation work packages are processed in `Update` and rendering in `LateUpdate`. This has the advantage that in `LateUpdate` all transformations in the scene have definitely already been completed. When the `RenderCallback` is fired, it pushes a camera transformation work package to the first queue and then waits using `EventWaitHandle(WaitOne(int millisecondsTimeout))`. In the `Update` function of the main Unity thread, the action queue is processed. When a camera move has been made, the image rendering workpackage is pushed to the second queue. So in the next `LateUpdate` call, an image is rendered, the image data is written to the byte array, and then the wait state is resolved by `EventWaitHandle(ewh.Set())`. This allows the segmentation thread to continue with the new image data. When the segmentation is completed, the library returns a list of center points. Figure 5.9 shows an example of the segmentation result in the Unity implementation.

5.3 Fitting

The goal of fitting is to align displaced GIS data with the segmented points. This is achieved by solving a least squares minimization problem, where the objective is to minimize the distance between individual segmented points and the 3D polylines provided in the GIS data. As a result, a transformation matrix should be obtained, which moves the model to the segmentation points.

Distance Calculation

The minimum distance between the segmented point (y_i) and a polyline segment (x_1 to x_2) can be calculated by constructing a triangle between the three points in 3D space, as



(a) cross product correspondence (b) distance to segment

Figure 5.10: Figure (a) is an illustration of the correspondence between the length of the cross product and the area of the parallelogram. (b) Calculation of the distance between a segmentation point (y_i) and a polyline segment (x_1 to x_2). Depending on whether the point is inside (green point) the space of intersection of two spheres given by the ends of the segment, or outside (red point), the distance (green) is given by the height inside an extended triangle or the distance between two points. The illustration is presented in 2D space for better visualization of the triangle formed by the three points.

illustrated in Figure 5.10(b). The distance from point y_i to the line defined by points x_1 and x_2 can be determined by calculating the triangle height. To calculate the area of a triangle in 3D space, one can use the fact that the normalization of the vector product between two vectors corresponds to the area of the parallelogram spanned by those two vectors. This fact is illustrated in figure 5.10 (a). Halving this area gives the area of the triangle (A). Once the area of the triangle is known, the distance (d_{line}) from y_i to the line can be calculated, as stated in equation 5.13.

$$\begin{aligned} d_{line} &= 2\frac{A}{|x_2 - x_1|} \\ A &= \frac{|(x_2 - x_1) \times (x_1 - y_i)|}{2} \\ d_{line} &= \frac{|(x_2 - x_1) \times (x_1 - y_i)|}{|x_2 - x_1|} \end{aligned} \tag{5.13}$$

The distance shown in this context refers to the direct distance from a point to the line passing through the two endpoints of a segment. It should be noted that a point that is not situated between these two points, as illustrated in Figure 5.10(b) with the point y_2, would not yield the distance to the segment, but rather to the line. In order to determine the minimum distance to the segment ($d_{segment}$), it is necessary to take into consideration three distinct cases, as schown in equation 5.14. We define that a point (y_i) is situated between the segment points if it falls within the intersection of two spheres, with the segment endpoints (x_1 and x_2) serving as the midpoints. The radius of these spheres is equivalent to the distance between the two endpoints ($|x_2 - x_1|$). In the event that the point is located outside this intersection, the distance between the point (y_i) and the closest segment endpoint (x_1 or x_2) is used as the smallest distance. This decision is highlighted in green in Figure 5.10 (b).

$$d_{segment} = \begin{cases} d_{line}, & \text{if } |(x_2 - y_i)| < |(x_2 - x_1)| \text{ and } |(x_1 - y_i)| < |(x_2 - x_1)| \\ |x_2 - y_i|, & \text{else if } |(x_2 - y_i)| < |(x_1 - y_i)| \\ |x_1 - y_i|, & \text{else} \end{cases} \tag{5.14}$$

Points that are situated between the segment points, yet outside the spheres are assigned a greater distance than the actual shortest distance. This can be interpreted as an added penalty in the least squares problem.

Implementing Least-Square Fitting

The fitting process is integrated into the C++ library for segmentation. Google's Ceres Solver[11] is used to solve the least squares minimization. The `AutoDiffCostFunction` is employed as the cost function template. This way, derivatives are calculated by Ceres. The parameters for the cost function include a 3D segmentation point and a pointer to an array of 3D model polyline points, along with the number of segments in the polyline. The template only requires implementation of the `operator()` function, which, in this case, takes in rotation and translation parameters and calculates the residuals. Rotation and translation are modified by Ceres solver during the minimization process. The residuals are the minimum distance between the segmented point and any segment of the polyline. To compute the residuals, all segments are iterated through, translated according to current rotation and translation values, then distance from the point is calculated as previously described. The smallest distance found during this iteration is defined as the residual.

The Ceres problem is constructed by iterating over all segmentation points and adding a new block to the problem via `AddResidualBlock`, using the standard squared loss as loss function. All blocks share the same reference to rotation and translation. Both rotation and translation are initialized with 0 values. If specified, any of the parameters can be fixed, e.g. if only the translation is to be calculated. A dense Cholesky factorization normal equations approach is selected as the solver type (`SPARSE_NORMAL_CHOLESKY`).

[11] http://ceres-solver.org/

6 Evaluation of Segmentation and Fitting

This chapter deals with evaluation of the segmentation and the fitting approach. First, the segmentation accuracy in a given dataset is evaluated against the ground truth. The alignment approach is then tested with synthetically generated data based on GIS documentation. Finally, the combination of segmentation from the given dataset and fitting the segmentation to ground truth data is analyzed.

6.1 Segmentation

First, the given data set containing of 3D point coulds of excavations and corresponding (ground truth) GIS documentation is discussed. It is explained how ground truth data are generated. Next, problem cases encountered during the evaluation of the segmentation algorithm are pointed out and strategies to overcome them are discussed. Finally, the segmentation is evaluated against the ground truth.

6.1.1 Ground Truth Data Analysis

We have a limited number of 3D scans with GIS data available, consisting of 30 photogrammetrically captured 3D reconstructions, all of which are limited to water pipes only. All types of pipe structures are listed in Table 6.1. Simple straight pipes or T-junctions are common in excavations, while curves, fire hydrants, or S curves are less so. However, our data set is evenly distributed. Figures 6.1 show different examples of given pipe structures.

Different Pipe Structures	
Straight Pipe	T Junction (Double T)
Curved Pipe	S-Curve Part
with Fire Hydrant	without Fire Hydrant

Table 6.1: Listing of the different pipe structures which are available in the scenes.

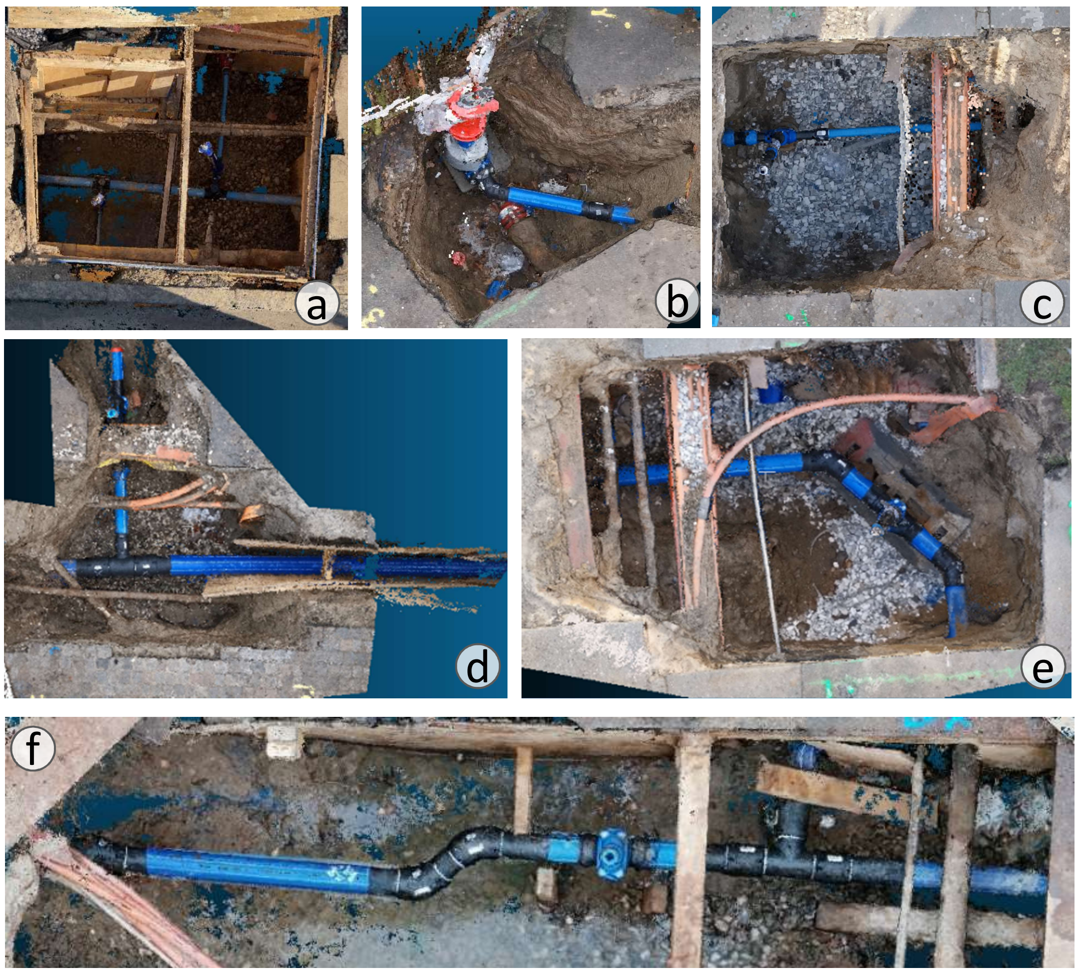


Figure 6.1: Examples of different pipe structure setups given in the 3D cloud dataset. (a) is a double T-junction with fire hydrants on both junctions. (b) a straight pipe ending in a fire hydrant. (c) is a straight pipe with fire plug. (d) a T-junction. (e) shows a curved pipe example with two 45 degree curves with a fire hydrant inbetween. (f) is a big Junction with a small S shift, a fire hydrant and a T junction.

The ground truth data provided is not always precise. In the field of SUE, errors within 20 cm are considered acceptable. However, for the purpose of evaluation, highly accurate ground truth is required. Some of the data is already well aligned, but others require improvement. An example of such GIS data can be seen in Figure 6.2 (left). Since it is already quite well aligned, ICP can be used for registration in such cases. However, ICP still has some limitations due to the presence of noise in 3D scan data. With ICP and some manual pre-processing, 3D models can be registered with high accuracy, as shown in Figure 6.2 (right). GIS data correction was applied to all data to create sufficient ground truth. In most cases, a height shift of approximately the radius of the pipe was observed. This is likely due to the fact that measurements were taken at the top of the pipes and

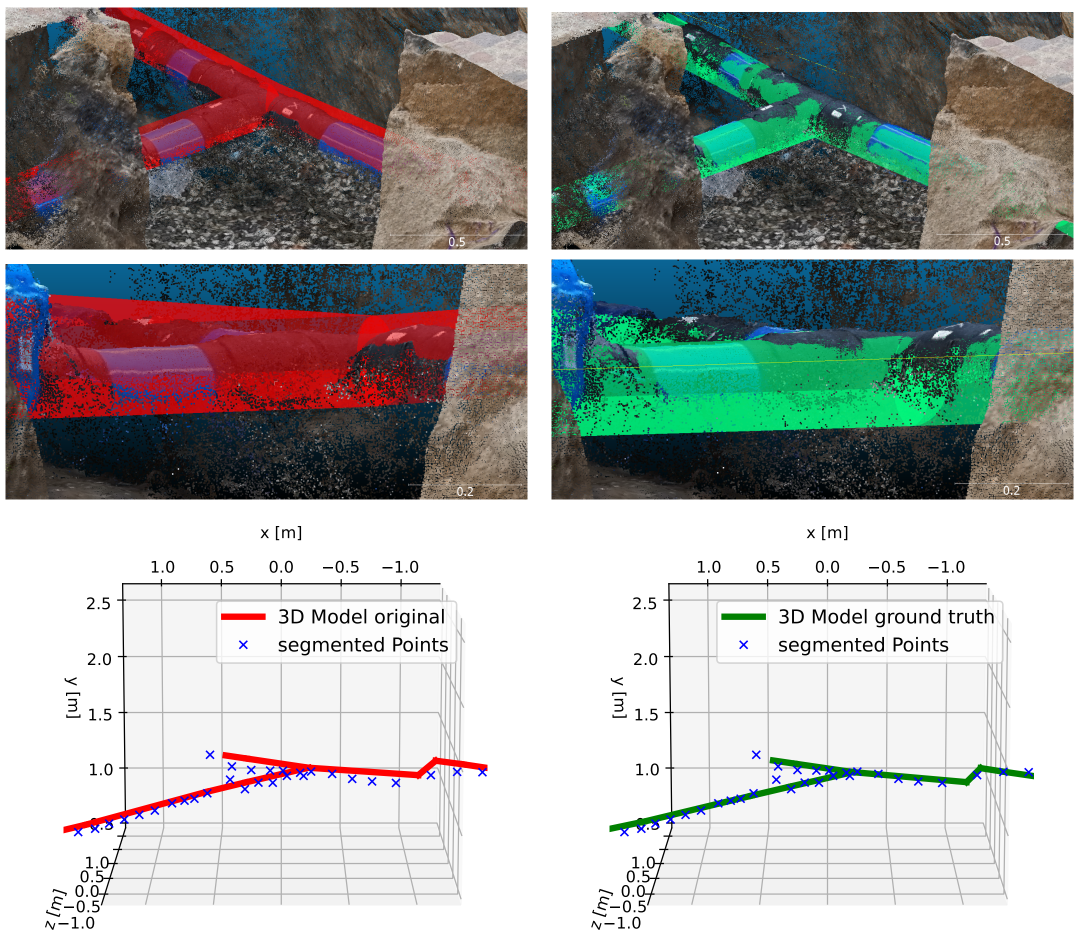


Figure 6.2: The left side shows given ground truth data. Top two screenshots are visualized pipes (red) in the corresponding 3D scan and the bottom left plot is a segmentation outcome (blue marks) compared to the given ground truth (red). After fine-tuning of given data, the new aligned ground truth is shown on the right side. Top two right screenshots are visualized pipes (green) in the corresponding 3D scan and the bottom left plot is a segmentation outcome (blue marks) compared to the updated ground truth (green). Given ground truth is good enough for typical SUE usecases, but not precise enough for evaluation of a segmentation algorithm. The updated data should fit well enough for evaluation.

taken directly. During this process, it was also noted that ICP introduces the largest error in height estimation, due to the relatively flat reconstruction of pipes in 3D space.

6.1.2 Findings during testing on different 3D data

A number of problematic scenarios emerged during the evaluation. Fire plugs moving towards the camera affect the spatial segmentation of the image. For straight pipes, they are usually not a big deal, if so, just the segmentation around can be a bit worse,

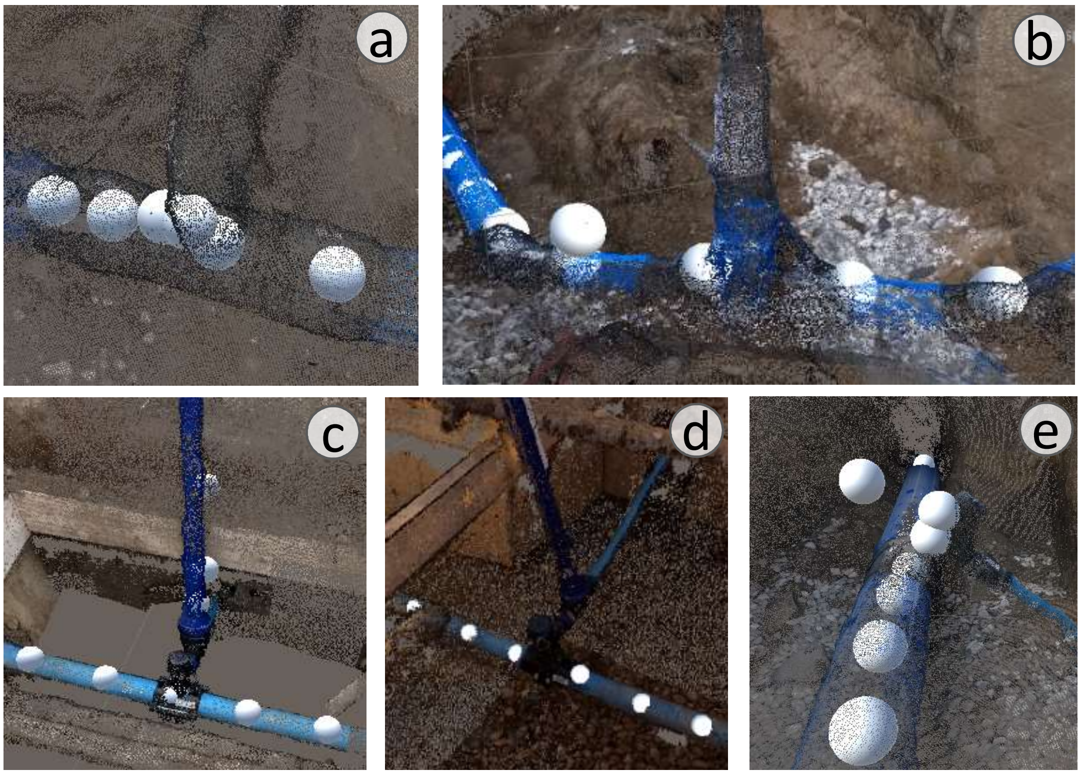


Figure 6.3: Poor segmentation examples, as seen in Unity screenshots showing 3D scan with segmented points represented as spheres. Scene (a) shows an undetected intersection, where the foreground crop causes the junction to go undetected. Figure (b) depicts larger errors around the fire hydrant. Figure (c) shows a fire hydrant angled rather than mostly straight, being wrongly detected as a pipe. Figure (d) illustrates an intersection that is not detected due to the presence of a fire hydrant. Figure (e) depicts a detected junction but poorly segmented branch, where the combination of poor 3D reconstruction and a steep slope causes issues for the segmentation algorithm.

as shown in the Figure 6.3 (c). But they can interfere with the correct detection of branches, or if they rise a little obliquely, they can themselves be segmented as part of the pipe. These cases can be seen in Figures 6.3 (c and d). Furthermore, limiting the culling space (near and far clipping), which actually reduces noise, can also be problematic. For example, it prevents the detection of the T-junction in Figure 6.3 (a). It also causes problems with very steep pipes as in Figure 6.3 (e). It should also be taken into account that we assume that pipes run rather vertically and do not have too steep slopes. Nevertheless, better reconstructed (and also thicker) pipes with a slope of about 45 degrees are segmented perfectly in another scene (See Figure 6.2 (bottom)).

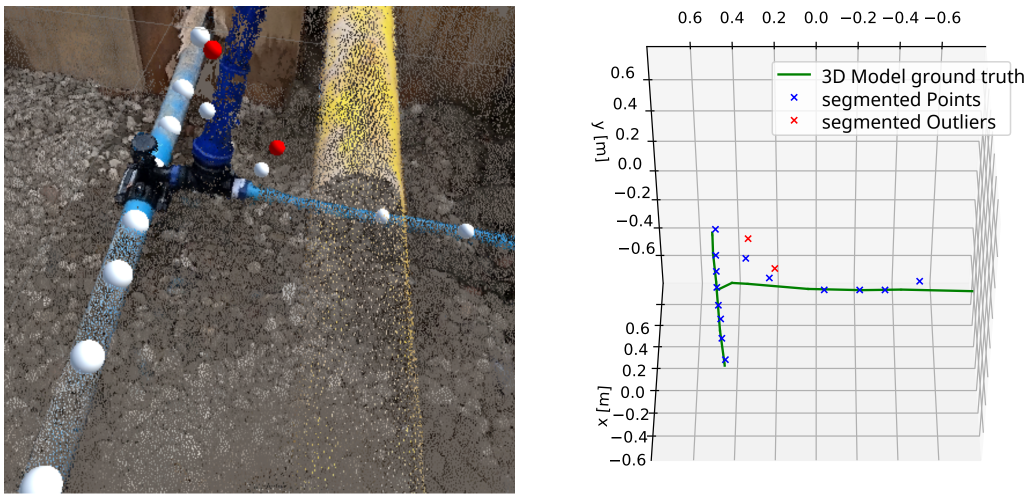


Figure 6.4: In this example, a fire hydrant is incorrectly segmented as part of the pipe structure, as seen in the Unity screenshot on the left and the ground truth comparison on the right. The left image displays segmentation points as spheres and corresponding points on the right plot are marked in blue/red. This can occur when the hydrant is not oriented vertically, but at an angle. Points where the vector from the previous segmentation point to the current one shows the highest value in the up axis are classified as outliers (red).

To reduce the impact of poor segmentation on fire hydrants, points with steep slopes relative to the previous point are treated as outliers. As demonstrated in Figure 6.4, points whose direction vector has a gradient of more than 60 degrees from the previous point are considered outliers. Inaccurate segmentation around fire hydrants on straight pipes can be neglected, because the few smaller outliers should not cause any major problems in the fitting process. In an attempt to relax the intersection detection properties, many incorrect intersections were classified, so this is not adjusted. For now, to solve this problem, we simply choose a second starting point on the adjacent branch and segment the two pipes independently.

Segmentation property selection: Empirically, we found an image size of 160x120 most suited for the segmentation approach. This size allows for effective segmentation while also ensuring faster processing of images. A `walk_distance` value of 0.2 m or 0.15 m is typically appropriate. However, too large steps can result in missed junctions and more points are beneficial for the fitting algorithm. On the other hand, smaller steps also decrease the efficiency of the segmentation process, as more points need to be segmented. The camera distance (`cam_distance`) depends on the scene and should take into account the radius of the pipes, with thicker pipes requiring a larger distance. A distance of 0.6 m has been found to be suitable. The camera's field of view performed well at 40 degrees, but this value should be adjusted according to the camera distance, with a closer distance often requiring a wider field of view.

6.1.3 Distance based Evaluation

To measure the accuracy of the segmentation, the distance of the segmented points from the ground truth is used. The distance is calculated as described in Section 5.3. Each segmented point is thus evaluated based on its distance to the ground truth. Figure 6.5 provides an example of point distance calculation.

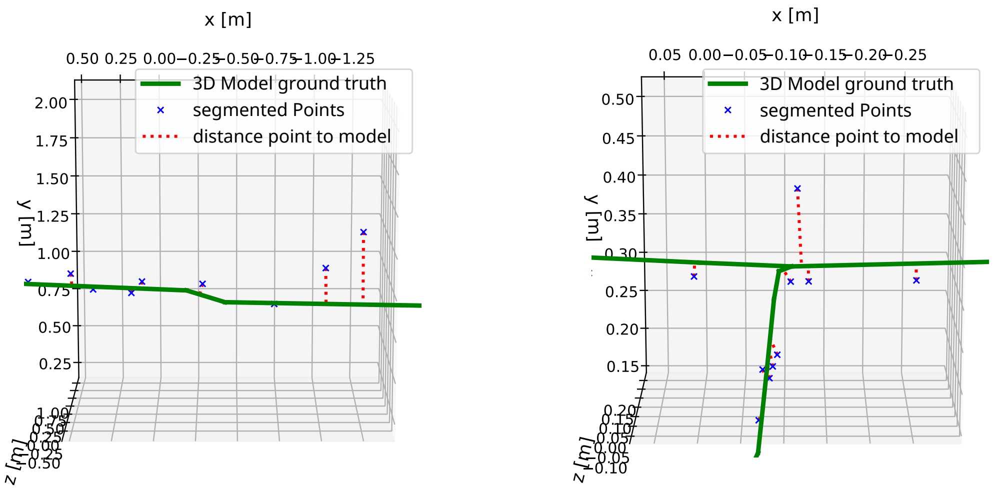


Figure 6.5: These examples visualize the point-to-model distance evaluation. For the evaluation of the segmentation algorithm, the shortest distance (red dotted) from the segmented points (blue) to the ground truth model (green) is calculated. Both plots show segmentations on 3D scans of the previously described data set.

All existing excavations were segmented for evaluation. Each excavation has been initiated from at least two different start-points. The segmentation algorithm parameters have been slightly modified for some scenes, but are generally used as described above. All individual segmentation points were stored and then evaluated against the ground truth. The results are shown in Table 6.2. At evaluation time different scenes are subdivided to see if different pipe structures lead to different results. For a better overview, the results are also visualized as a boxplot in Figure 6.6. As already mentioned, incorrect height segmentations are already filtered out at this stage. Table 6.2 presents the average and maximum values for all segmentation points, as well as the average, median, and maximum of the best 98% to exclude outliers. The results indicate that the errors tend to be less than 5 cm, which is considered sufficient for reasonable segmentation. The boxplot shows that the maximum error in all configurations is less than 10 cm, and there are only a few outliers above that. These results fulfill requirement [R2.1]. However, the problem cases mentioned in the previous paragraph should be examined more closely in order to eliminate them as far as possible.

		all values		best 98%		
Scene Type	#	**Avg. [m]**	**Max [m]**	**Avg. [m]**	**Median [m]**	**Max [m]**
Straight Pipe	10	0.039033	0.300554	0.031225	0.028427	0.096917
T Junction	18	0.034915	0.273999	0.028473	0.024416	0.099826
Curve	**2**	0.056413	0.112391	0.053401	0.051858	0.107196
w. fire hydrant	17	0.035632	0.300554	0.028870	0.024108	0.100664
w/o fire hydrant	13	0.042254	0.173779	0.037045	0.032090	0.096917
Overall	30	0.036906	0.300554	0.030443	**0.026050**	**0.100664**

Table 6.2: Evaluation of the accuracy of the segmentation algorithm. The distances of the individual segmentation points to the ground truth model are shown in this table. Grouped in the different pipe structure configurations available. Segmentation was performed at least twice per scene and the number of available scenes is provided in column two. The results indicate an overall segmentation accuracy of less than 5 cm, with maximum errors at less than 10 cm. Larger errors are only observed as outliers. The curved dataset has the largest median error, which may be attributed to the small sample size.

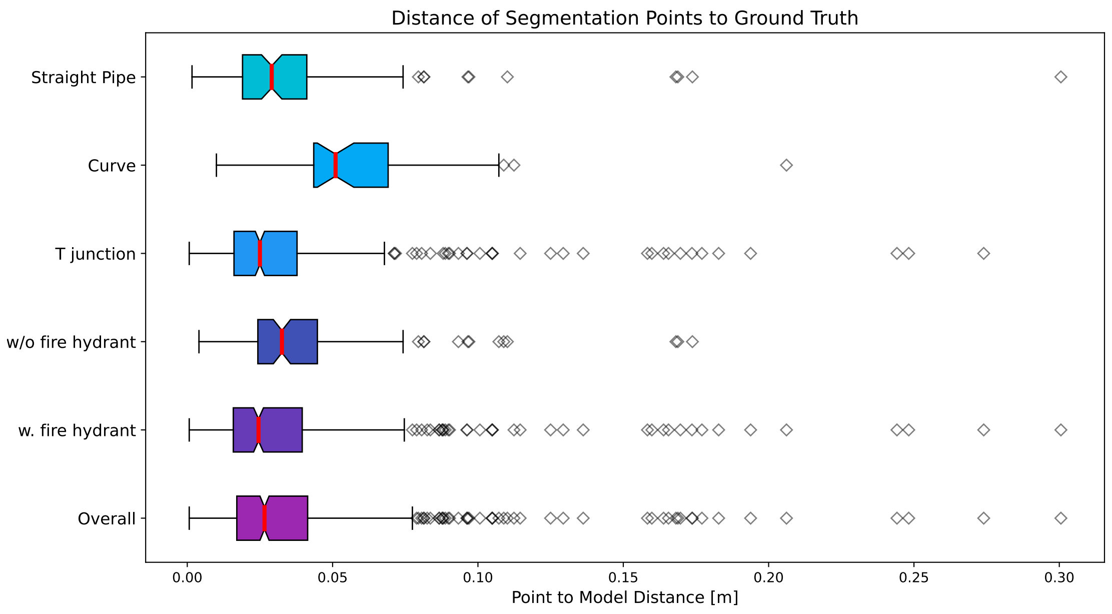
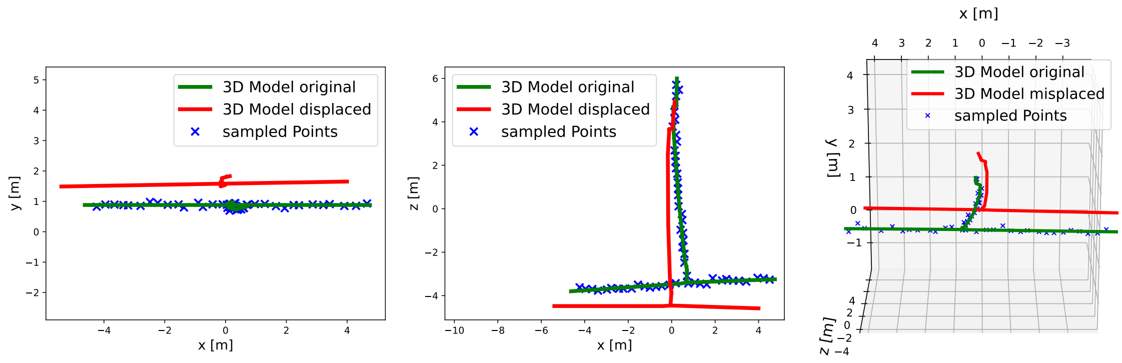


Figure 6.6: The evaluation shows that the different pipe structure setups have similar errors, suggesting that the structure does not affect the accuracy of the segmentation. The Curve scenario appears to have a slightly larger error, but it must be noted that it is based on only two scenes, which reduces the strength of the conclusion. Overall, the mean error is around 3 cm and more than 75% of all segmentation points have a distance from the ground truth model of less than 5 cm. When outliers are excluded, the errors are clearly below 10 cm, which is within the desired range.



(a) Side View (c) Top View (d) 3D View

Figure 6.7: The synthetic input for evaluation is created by sampling points (blue) along the ground truth model (green) and applying Gaussian noise to simulate segmentation error. The model is transformed by random $[R|t]$ to simulate model displacement (red).

6.2 Fitting

The evaluation of fitting is divided into two parts: Evaluation on synthetic data and on real segmentation data. First, using synthetically generated data to test it independently of the segmentation algorithm. Second, the combination of segmentation results on the ground truth dataset from the previous chapter is used to evaluate the fitting approach.

6.2.1 Experiments on synthetic Segmentation Points

The GIS Data API is used to retrieve pipe data. Using this pipe structure, points are sampled along the pipes at a specified frequency (`sample_distance`, which is the equivalent in synthetic data generation, to `walk_distance` in the segmentation algorithm). Gaussian noise ($\sigma = 0.1m$) is applied to the sampled points to simulate the inaccuracies of the segmentation algorithm. The pipe structure is then offset with arbitrary transformations ($[R_{true}|t_{true}]$) to simulate the disalignment of GIS data, as illustrated in Figure 6.7.

Using the sampled points and the displaced pipe structures, the fitting algorithm is executed. The resulting transformations ($[R_{res}|t_{res}]$) can be compared to initial transformations for displacement to estimate the accuracy of the alignment result 6.1.

$$\begin{aligned} R_{error} &= R_{true} - R_{res} \\ t_{error} &= t_{true} - t_{res} \end{aligned} \tag{6.1}$$

Another indicator for evaluating the result is the distance between the segmentation points and the line segments. The average distance of all points should be similar to the applied noise ($\sigma \approx avg\_distance$). In fact, it should be smaller. As the points can also be moved along the pipe after the noise is added, this does not increase the distance to the line segment. The error from the final step of the least-squares optimization can be used as the distance metric.

	Rotation [°]		Translation [m]	
Axis	Minimum	Maximum	Minimum	Maximum
x	-2	2	-2	2
y	-5	5	-2	2
z	-2	2	-2	2

Table 6.3: In order to evaluate the performance of the fitting algorithm, a benchmark dataset is generated by applying random transformations within predefined bounds to the ground truth data. Bounds for translations are determined based on observations of real-world displaced GIS documentation. Assuming potential edge cases, we cover more rotational displacement variance than would be anticipated in real data because rotational displacements are not as significant in real data. This allows for a more comprehensive assessment of the algorithms ability to handle rotational alignment.

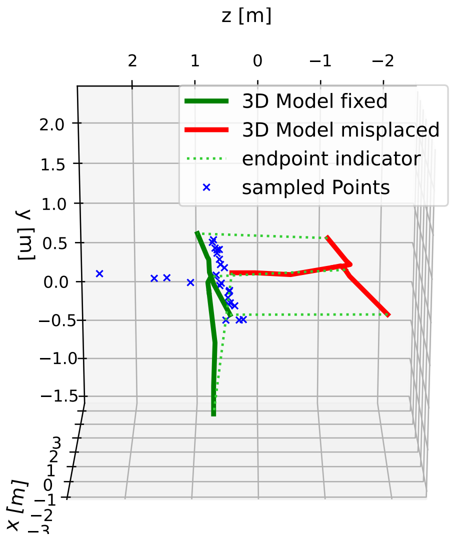
To use the pipe structure in the evaluation, we draw from ten random junctions with a radius around the central node of 7 m. The segmentation points are simulated as previously described and the pipe structure is randomly transformed to generate displacement as given in Table 6.3.

These ranges are realistic deviations that can be expected in real excavations. This is repeated many times to obtain results to evaluate the quality of the fitting approach. Only pipe constructions with a junction or significant directional change are used in this experiment. This is because there is no unique solution for the alignment process of a line in 3D space. The rotation along the line cannot be defined unambiguously.

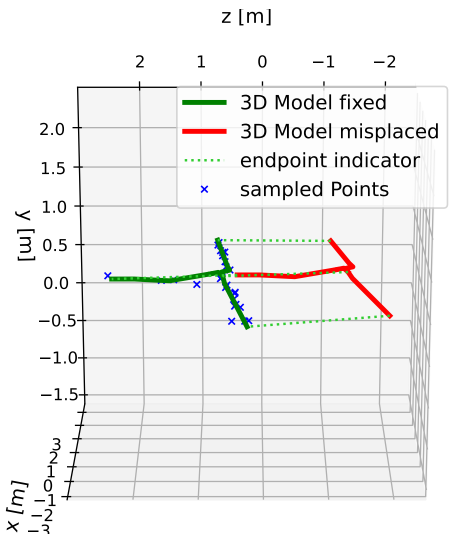
During the evaluation, we noticed that configurations with one branch, in which the outgoing branch is represented by only a few segmentation points, often failed in the rotation estimation. Such a case is illustrated in Figure 6.8 (a). The translation estimate is good, but there is a large error of almost 90 degrees in the X-axis rotation correction. The problem is probably that the four points of the outgoing branch are treated as outliers during the optimization process. One solution is to attach the fitting twice. During the first pass, the rotation is fixed and only the displacement in translation is calculated. This translation serves as an initialization in the second pass. This procedure fixes the rotation error as shown in Figure 6.8(b).

If the initial fitting results in an average distance greater than 0.3 m, or a rotation error greater than 5° in any of the x, y, or z axis, then the fitting algorithm will be re-executed using the two-step approach. The overall results, as described in Table 6.4, show no rotation errors greater than 5°. This indicates that reinitialization can effectively eliminate such poor fits.

With this change, we performed an evaluation over 1000 random samples with a `sample_distance` of 0.3 m. The results are listed in the top half of Table 6.4. The results are split into two parts, first all experiments including the problematic junction



(a) fitting error



(b) fixed fitting error

Figure 6.8: These plots exhibit the fitting results, where the red lines represent misplaced 3D polylines, and the blue markers are simulated segmentation points. The green line represents the corrected pipe model, using the transformation $[R|t]$ calculated in the least-squares minimization process. Plot (a) shows one run where rotation and translation are calculated simultaneously. This leads to an error in this example with only four samples on the short side branch. In Plot (b), this estimation was done in two runs. First, only translation was calculated and used to initialize the translation error in the second run. The second run then performed optimization on both translation and rotation variables.

(with edge case) and second, these experiments are omitted (w/o edge case). The average distance between sample points and the pipe line segments is smaller than the σ of the Gaussian noise, indicating a good fitting result. The maximum error of almost 5 degrees on the X-Axis estimates indicates that there is still a problem with a few points on a side branch. Since only 4 points do not contribute significantly to the overall distance error used in the optimization process, we must accept this limitation. The median translational error is less than 0.05 m for both data sets, and the median rotational error is less than 0.3 degrees in both cases. Maximum errors have higher peaks in the X-Axis for both data sets. These outliers probably come from the fact that the pipes in our data tend to be oriented along the X-Axis. Thus, junctions with few data points on the adjacent branch (as described before) are not weighted enough in the alignment process. Since these points could also be outliers, it is not feasible to increase the weighting of such points.

Further experiments, with a three times higher sampled point frequency (`sample_distance` = 0.1 m), are shown in the lower half of Table 6.4. Results indicate that, on average, the rotational error is almost halved. However, notable outliers remain in the X-axis rotation. Translation error is also reduced, but only slightly, which doesn't make a significant difference. A direct comparison of the two evaluations is shown in Figure 6.9. The improvement in the accuracy of rotation estimation is evident, whereas

			Rotation [°]		**Translation [m]**	
		Axis	**Median**	**Max**	**Median**	**Max**
sample_distance = 0.3m	edge case	**Avg. Distance**	-	-	0.069825	0.088471
		error (x)	0.279028	**4.546382**	0.037170	0.238620
		error (y)	0.107049	0.571547	0.014570	0.105548
		error (z)	0.109813	1.490714	0.038234	0.217004
	w/o edge case	**Avg. Distance**	-	-	0.070222	0.085800
		error (x)	0.170775	1.215341	0.038621	0.238621
		error (y)	0.094429	0.536870	0.013768	0.068598
		error (z)	0.097739	0.675852	0.039110	0.200522
sample_distance = 0.1m	edge case	**Avg. Distance**	-	-	0.071563	0.079621
		error (x)	0.121080	**2.241106**	0.033723	0.178539
		error (y)	0.054118	0.546967	0.011802	**0.069970**
		error (z)	0.062881	1.442006	0.033414	0.182977
	w/o edge case	**Avg. Distance**	-	-	0.071636	0.079161
		error (x)	0.090692	0.822003	**0.033285**	0.178539
		error (y)	0.049876	0.466418	**0.011569**	0.062050
		error (z)	0.058663	0.626382	**0.034008**	0.174351

Table 6.4: Result of four evaluation passes. The upper two were performed with synthetic segmentation points at a sample distance of 0.3 m and the lower two at 0.1 m. The two scenarios are both divided into one run with and one without the problematic junction (edge case). The rotation errors in the examples with the problematic junction are higher, because few samples are on the outgoing branch. With a higher sample rate and thus more points on the branch, this error decreases. Overall, median errors are more than sufficiently small enough. Here it is noticeable that with the higher sample rate the rotation errors are almost halved, while the translation error is not significantly smaller. It is also noticeable that the translation error on the Y (height) axis is clearly the smallest in each scenario. This is probably due to the fact that the pipe structures almost lie on a flat surface and therefore the Y value is very similar in all samples.

the improvement in translation estimation is minimal. Overall, it can be seen that the estimation error for rotation can be determined with an accuracy of less than 0.6 degrees, and translation within 15 cm. It is also noticeable that the error in the height axis (Y) is smaller. This is probably due to the fact of similarity in the distribution of points in the height axis as the pipe structures are level. Three randomly selected plots from the evaluation process are shown in Figure 6.10, which demonstrate that the displaced 3D models can be aligned very well to the synthetic segmentation points.

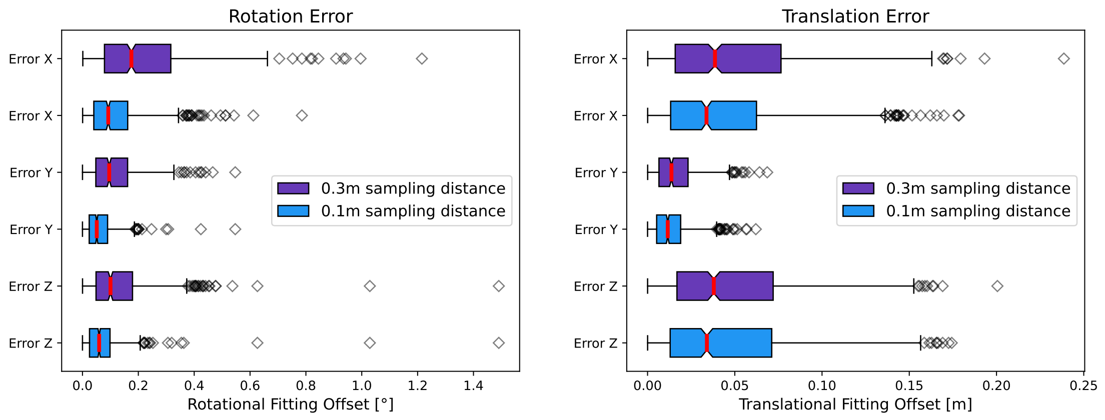


Figure 6.9: The direct comparison between results of higher (blue) and smaller (purple) sample rate is plotted. Rotational error is shown on the left and translational error on the right. A higher sample rate of segmentation points is observed to reduce the error of the alignment process, particularly in the rotational case. The overall results, with a sample distance (`sample_distance`) of 0.3 m, already demonstrate sufficient accuracy with 95% of rotational alignment error within 0.6 degrees. The translational alignment is similar for both sample sizes. The trend towards the lowest error in the height (Y) axis can be attributed to the similarity in the distribution of points in the height axis as the pipe structures are level.

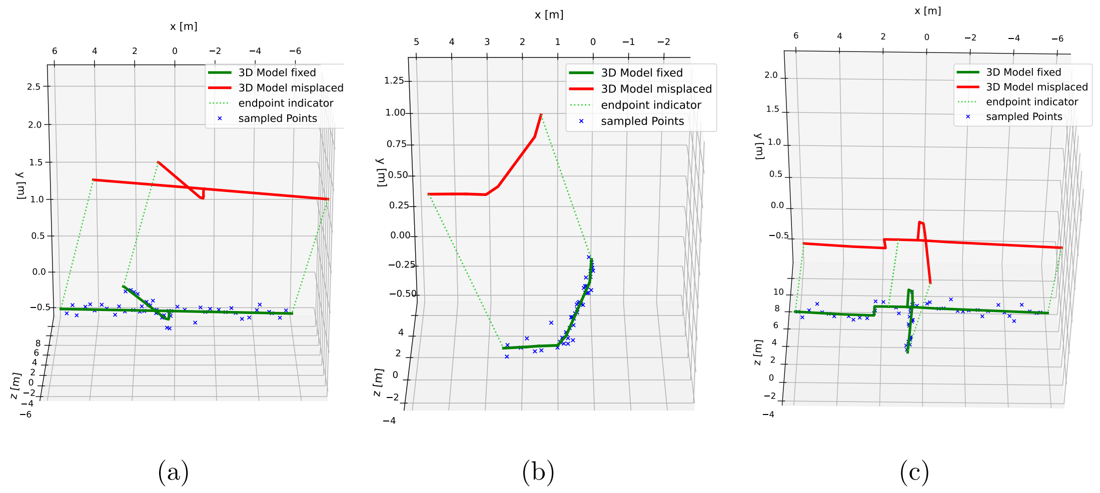


Figure 6.10: Three randomly chosen evaluation examples on synthetic data. The displaced 3D model is seen in red and the sample points to align this model are marked in blue. The result of the fitting algorithm is visualized in green. Dashed light green indicates model endpoints to highlight the transformation of the displaced 3D model. Example (a) with sample rate of 0.3 m comes with a rational error of: $x = 0.261352; y = 0.184; z = 0.048°$ and a translational error of $x = 0.048m; y = 0.023m; z = 0.052m$. Example (b) with sample rate of 0.1 m comes with a rational error of: $x = 0.166; y = 0.168; z = 0.353$ and a translational error of $x = 0.033m; y = 0.025m; z = 0.073m$. Example (c) with sample rate of 0.3 m comes with a rational error of: $x = 0.076964; y = 0.145; z = 0.114$ and a translational error of $x = 0.072m; y = 0.001m; z = 0.094m$.

Conclusion

Except for a few outliers, the fitting approach satisfies requirements for aligning the displaced GIS documentation with given segmentation points. This fulfills requirements [R2.2] as listed in Table 3.1. The results indicate that increasing sampling frequency of segmentation points improves the accuracy of the alignment process. As the optimization only aligns a relatively small number of points on line segments, instead of an entire subpart of the 3D scan on a full 3D model, alignment calculation is performed more efficiently. An increase in the number of segmentation points would not be a problem, as the number of samples would rather be limited by the execution time of the segmentation algorithm.

6.2.2 Experiments on real Segmentation Points

To evaluate the alignment approach with real data, we use the results of the segmentation evaluation. As rotation in three-dimensional space can only be accurately determined when an intersection is present, we only consider examples with junctions. The evaluation process is similar to the previous chapter with synthetic data, but actual segmentation results are used and ground truth models are randomly displaced. The results of aligning a particular segmentation stood out, as the rotational error of about 7 degrees along the x-axis remained consistent regardless of how the ground truth model was shifted. It was observed that the average distance from the segmentation points to the aligned model was less than 0.03 m compared to the ground model at 0.04 m, suggesting that the fit aligns better with segmentation points than with the ground truth. The error is caused by an increasing error in height estimation at the end of the outgoing branch of the pipe. This problem can be seen in Figure 6.11 (a). This error is present in multiple examples, but is not as serious in others. Segmentation of this larger error is excluded from the remaining evaluation. The excavation sites have a size of up to 10 m, and since the rotation is applied at the center of the scene, the displacement error is given by the rotation-caused displacement at a distance of 5 m. In this case, we aim for a rotation accuracy of less than 1 degree to ensure adequate visualization. Greater errors are too height for a usable application. Nevertheless, translation remains precise enough even in cases with poor segmentation at the corners, which is the most critical aspect.

During the evaluation of 1000 randomly displaced models, another error in the fitting approach was identified. For ten of the 1000 samples, the optimization procedure found a local minimum that did not correspond to the correct solution, an example of which can be seen in Figure 6.11 (b). This can be corrected by using a different initialization. To address this issue, we now reattempt fitting four times with the previously found translation, but shifted by 1 m in the X and Z axes ([+1,0],[-1,0],[0,-1],[0 ,+ 1]) if the average point distance is still greater than 0.05 m after initial alignment. The result with the lowest average distance is then used. While this currently solves the problem, a more efficient approach may be developed in the future.

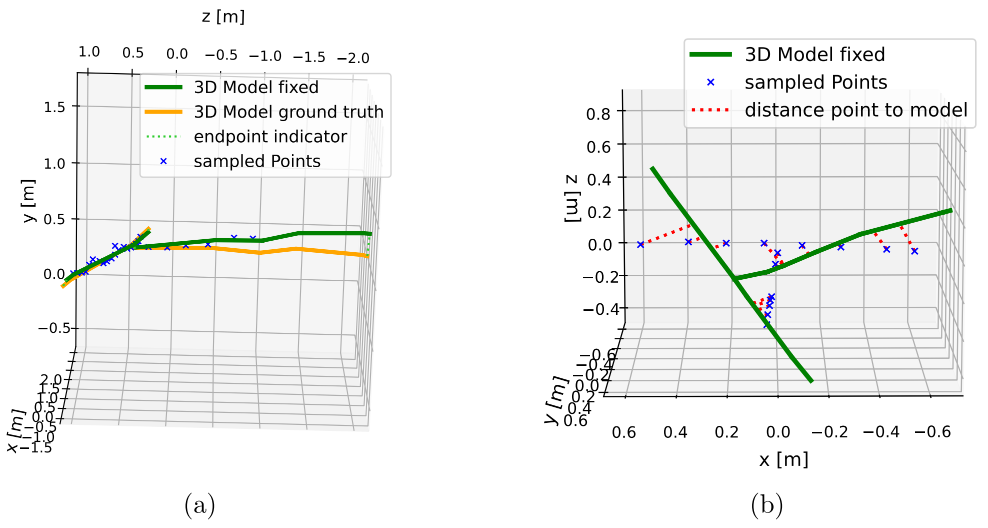


Figure 6.11: Figure (a) illustrates a major drawback when poor segmentation occurs at the ends of the pipe. The rotation error of the fitted model (green) with respect to the ground truth (orange) is approximately 7 degrees. Despite this, the alignment works well, with an average point-model distance of 0.03 m after fitting as compared to 0.04 m to the ground truth. The problem originates from the incorrect height estimate of the two last segmented points on the outgoing branch. This illustrates that poor segmentation of edge points can lead to inaccurate estimates. Figure (b) shows a fitting result (green) to segmentation points (blue) in which the Ceres solver found a local minimum during the optimization process, which was not the best solution. The distance (dotted in red) from the points to the model is minimized, but the resulting transformation is wrong.

The final results of the 1000 sample evaluation are presented in Table 6.5. The average distance between the sample points and the model is smaller compared to the evaluation with synthetic data. This was expected, since the error of the actual segmentation is more constant, rather than distributed with Gaussian noise. The small error indicates a functional fit. Average translation values of less than 5 cm are more than sufficiently accurate. Despite this, the median values of the rotation error are quite high. In Figure 6.12, the results are visualized using a boxplot. The distribution of translation errors is similar to that of synthetic data. Rotation errors, on the other hand, vary significantly from experiments with synthetic data. This is likely due to two reasons. First, in real excavations, the junctions are often only visible for a short distance and quickly disappear into the ground. This results in fewer segmented points on these branches, contributing to worse results in rotation estimation, as described in the previous section. Additionally, as shown in the example in Figure 6.11 (a), we have seen that errors at the end of a pipe significantly affect the alignment.

A rotation error of 3 degrees would result in a displacement of approximately 28 cm at a distance of 5 m. As larger rotational errors are more likely to occur in smaller excavations where there are fewer segmentation points, the desired accuracy of a shovel head is still

	Rotation [°]		Translation [m]	
Axis	**Median**	**Max**	**Median**	**Max**
Avg. Distance	-	-	0.031317	**0.041186**
error (x)	0.602720	3.355823	0.039110	0.193550
error (y)	0.963099	2.600000	0.019330	0.172001
error (z)	0.305951	3.057889	0.043835	0.205524

Table 6.5: The results of the alignment process of real segmentation points show a lower average distance from the model. As expected, since segmentation errors are not artificially introduced. This indicates that the fitting algorithm is also effective with real values. The translation alignment is sufficiently accurate, with a median error of less than 5 cm and a maximum error of less than 20 cm. The distribution of translation results is similar to that of the synthetic segmentation data experiments. However, the rotation errors can be quite large in some cases, which may not be sufficient. Poor results in rotation estimation can come from view bad segmentation points at the end of the disclosed pipes.

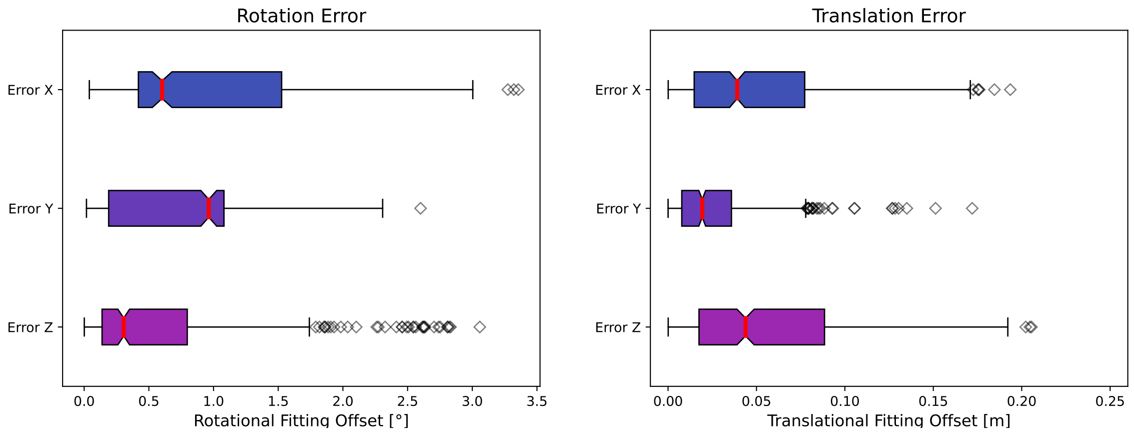


Figure 6.12: Results on real segmentation data show suitable median values. The distribution of translational alignment errors is similar to that of synthetic data. A median error of less than 5cm and maximum error of less than 20cm is considered adequate for practical applications. However, a few bad segmentation points at the end of pipes and a significant number of real-world scenarios with only small exposed parts of the outgoing branches at junctions lead to higher rotational errors. Despite this, a majority of the rotation results are still within one degree offset, which is considered accurate enough.

likely to be met in these scenarios. The rotation of the model takes place at the center of the excavation, and small excavations do not exceed 5 m, thus the displacement offset must remain small enough. The translation accuracy is highly accurate and can be used to improve GIS data. In Figure 6.13, 9 randomly selected visualizations of evaluation results are displayed, showcasing that the fitting approach also works effectively with real segmentation examples.

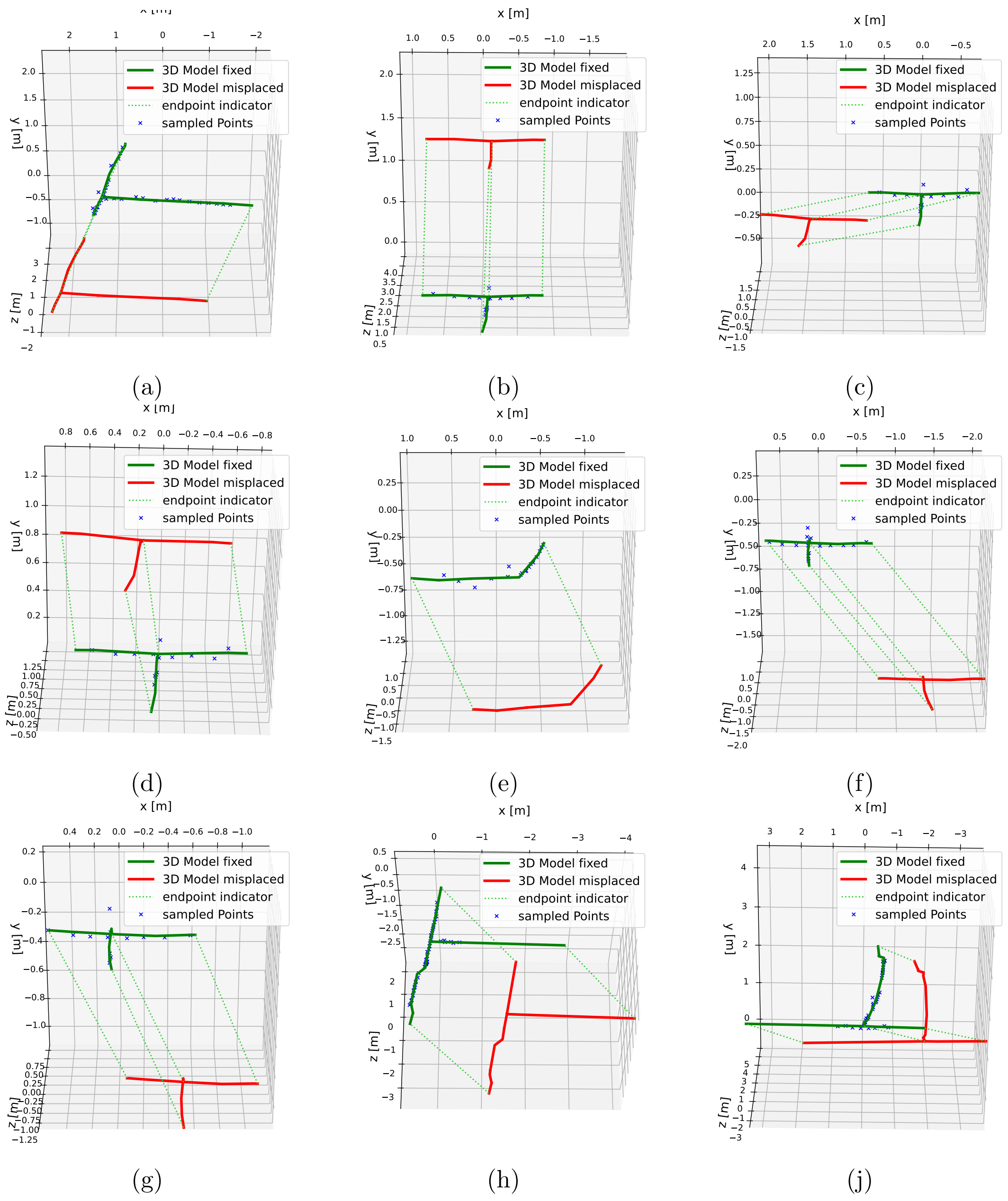


Figure 6.13: Nine randomly selected plots from the 1000 examples of the evaluation with real segmentation points are shown. The fitting approach is consistent throughout. Higher errors in the statistics of the comparison between the fitting result and the ground truth are attributed to issues in the segmentation itself.

7 Conclusions and Future Work

In this work, we present a complete solution for real-time visualizations in subsurface utility engineering (SUE) works. First, we discuss the improvements to our outdoor AR setup presented in previous work. Such as the implementation of north estimation with RTK GPS measurements, which makes it independent of compass to achieve highly accurate and reliable outdoor tracking. Furthermore, we have implemented a continuous drift correction algorithm that allows both local and global tracking to be combined robustly and consistently over a longer period of time. Evaluations show that tracking accuracy below the target of 20 cm is achieved. Additionally, real-time LIDAR reconstruction enables the placement of visualizations with sub 5 cm accuracy at distances up to 5 m. This enables the reliable and accurate implementation of applications such as virtual spray marking tools. Of course, highly accurate tracking is also the key to generate useful geo-referenced 3D reconstructions, which are crucial for the segmentation algorithm.

With poor GIS documentation, SUE visualizations are not very useful. The current workflow requires a significant amount of additional effort to align displaced data. To address this issue, we propose a segmentation algorithm using semantic segmentation to identify pipes in open excavation sites and uses this information to align displaced data, improving the workflow efficiency. Our approach is based on segmentation in the 2D image domain, followed by the reprojection of the segmentation results into 3D space. Benefits of segmentation in the image space are the reduced influence of 3D reconstruction noise and the flexibility in the representation formats of the 3D reconstruction. The implementation of the segmentation was done in an independent library which can be used with any rendering engine. This enables a wide range of applications, including real-time on-site applications, and background renderings for GIS processing software like QGIS. In order to align the displaced 3D models to the segmentations, we have implemented a fitting algorithm. Since the segmentation provides single 3D points along the pipe structure, this process could be implemented using a simple leas squares minimization method. The limited number of segmentation points, as opposed to entire sub-parts of the 3D reconstruction, results in a small minimization problem. The algorithm calculates a transformation of the displaced 3D polyline to align the segmentation points as accurately as possible. The evaluation of our proposed segmentation approach shows an accuracy of less than 10 cm, which meets the set requirements. Further evaluation of the fitting algorithm using synthetic data indicates that it allows for highly accurate alignment of the segmentations. Translation errors were found to be within the targeted 20 cm range, and rotation errors were also below the targeted 1 degree. The alignment experiments performed on real segmentation data also yielded good results, with translation errors remaining within the target range. The rotation errors, however, have shown larger outliers to the compared ground truths. These errors occur especially with bad segmentations at pipe endings. Overall the results are usable in translation, but an improvement of the segmentation at the edge of excavations where pipes disappear into the ground should be considered. In real-world scenarios, rotational shifts are less likely to occur as pipes are typically laid on a relatively straight plane and deviations from the plan are usually translational.

Future Work

There are some possible improvements and extensions to the approached segmentation algorithm. Additionally, several promising potential applications for this work are worth exploring. One area of focus for further research is to investigate the cause of the recurrent issues in height estimation of the segmentation at the borders of excavations. Our analysis suggests that the Distance Transform filtering method may contribute to this problem, particularly in images where pipes do not occupy the entire width of the image. Alternative methods or modifications to the current approach should be considered to avoid such behavior. While the option of skipping edge points may appear to be a viable solution, it could lead to branches with a limited number of segmentation points, which could negatively impact the alignment process. In addition, a revision with the aim of optimizing performance would be a good idea. The current implementation is not optimized in terms of speed and memory consumption. After that, comparisons could be made with other possible segmentation algorithms in 3D space. This would be a crucial step in preparing the work for practical application. Furthermore, the development of a real-time system is needed for the practical implementation of the work. Although the described components are functional on mobile devices, the current iOS version (16) only allows for LIDAR 3D reconstruction using non-textured meshes. Thus, the implementation of a real-time capable mesh texturing in Unity is necessary to finalize the prototype. Another interesting application area is GIS software integration. This would enable geo-referenced 3D reconstructions captured by excavation workers to be integrated with GIS software for data optimization using our segmentation and alignment approaches. One could select a starting point in a 2D top view of the 3D reconstruction, and using the background rendering, the 2D starting point can be projected as a ray into the 3D scene to obtain a starting point for segmentation. Afterwards segmentation and fitting can be performed, and suggestions for adjustments to the GIS data can be made within the software. In any case, this work would certainly be useful for supporting the construction of a large database to enable the creation of deep learning based segmentation methods. Currently, a lack of sufficient data is a significant challenge that hinders the feasibility of such approaches. However, in the future, such techniques may become viable as they are well-suited for computer vision problems and have the potential to produce fast and accurate results.

A

List of Acronyms

API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
DGPS	Differential GPS
DoF	Degrees of Freedom
fov	field of view
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HSV	Hue, Saturation, Value
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IoT	Internet of Things
KNN	K-Nearest Neighbors
LIDAR	Light Detection and Ranging
MQTT	MQ Telemetry Transport, MQ is reference to IBM MQ
NTRIP	Networked Transport of RTCM via Internet Protocol
NRTK	Network-streamed correction data
OSM	Open Street Maps
PCA	Principal Component Analysis
PPP	Precise Point Positioning
RANSAC	RANdom SAmple Consensus
RGB(-D)	Red, Green, Blue (plus Depth)
RTK	Real-Time Kinematic
RTCM	Radio Technical Commission for Maritime Services
SIFT	Scale-Invariant Feature Transform

SLAM	Simultaneous Localization And Mapping
SUE	Subsurface Utility Engineering
SVD	Singular Value Decomposition
UTM	Universal Transverse Mercator
VI-SLAM	Visual-Inertial SLAM
VO	Visual Odometry
QR	Quick Response

Bibliography

[1] Gerhard Schall, Stefanie Zollmann and Gerhard Reitmayr. Smart vidente: advances in mobile augmented reality for interactive visualization of underground infrastructure. *Personal and Ubiquitous Computing*, page 1533–1549, 2012. (page xv, 2, 15)

[2] Gerhard Schall, Erick Mendez, and Dieter Schmalstieg. Virtual redlining for civil engineering in real environments. pages 95–98, 09 2008. (page xv, 2, 15)

[3] Lasse Hedegaard Hansen. *Augmented Reality for Subsurface Utility Engineering: Exploring and developing 3D capture and AR visualization methods for subsurface utilities.* PhD thesis, 2021. PhD supervisor: Assoc. Prof. Erik Kjems, Aalborg University. (page xv, 2, 15, 20)

[4] Lasse H. Hansen, Philipp Fleck, Marco Stranner, Dieter Schmalstieg, and Clemens Arth. Augmented reality for subsurface utility engineering, revisited. *IEEE Transactions on Visualization and Computer Graphics*, 27(11):4119–4128, November 2021. Publisher Copyright: © 1995-2012 IEEE. (page xv, 1, 3, 9, 19, 23, 24)

[5] J. C. Spohrer. Information in places. *IBM Systems Journal*, 38(4):602–628, 1999. (page 1)

[6] Steven Feiner, Blair Macintyre, Tobias Höllerer, and Anthony Webster. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. volume 1, pages 74–81, 12 1997. (page 6)

[7] B. Thomas, V. Demczuk, W. Piekarski, D. Hepworth, and B. Gunther. A wearable computer system with augmented reality to support terrestrial navigation. In *Digest of Papers. Second International Symposium on Wearable Computers (Cat. No.98EX215)*, pages 168–171, 1998. (page 6)

[8] W. Piekarski and B.H. Thomas. Tinmith-metro: new outdoor techniques for creating city models with an augmented reality wearable computer. In *Proceedings Fifth International Symposium on Wearable Computers*, pages 31–38, 2001. (page 6)

[9] Duncan Robertson and Roberto Cipolla. An image-based system for urban navigation. 01 2004. (page 6, 7)

[10] Gerhard Reitmayr and Tom Drummond. Going out: Robust model-based tracking for outdoor augmented reality. pages 109–118, 10 2006. (page 6)

[11] Taragay Oskiper, Supun Samarasekera, and Rakesh Kumar. Multi-sensor navigation algorithm using monocular camera, imu and gps for large scale augmented reality. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 71–80, 2012. (page 6)

[12] Samu Suurinkeroinen. Centimeter-level real-time position tracking options with a short initialization time for outdoor applications. Master's thesis, LUT University, 05 2020. (page 6)

[13] Wei Zhang and Jana Kosecka. Image based localization in urban environments. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 33–40, 2006. (page 7)

[14] Amir Zamir and Mubarak Shah. Accurate image localization based on google maps street view. pages 255–268, 09 2010. (page 7)

[15] Gonzalo Vaca-Castano, Amir Roshan Zamir, and Mubarak Shah. City scale geo-spatial trajectory estimation of a moving camera. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1186–1193, 2012. (page 7)

[16] Clemens Arth, Christian Pirchheim, Jonathan Ventura, Dieter Schmalstieg, and Vincent Lepetit. Instant outdoor localization and slam initialization from 2.5d maps. *IEEE transactions on visualization and computer graphics*, 21, 07 2015. (page 7)

[17] Jinmeng Rao, Yanjun Qiao, Fu Ren, Junxing Wang, and Qingyun Du. A mobile outdoor augmented reality method combining deep learning object detection and spatial relationships for geovisualization. *Sensors*, 17:1951, 08 2017. (page 8)

[18] Ying Liu, Xiufang Shi, and Shibo He. Prospective positioning architecture and technologies in 5g networks. *IEEE Network*, PP, 06 2017. (page 8)

[19] Marco Fortunato, Joshua Critchley-Marrows, Malgorzata Siutkowska, Maria Ivanovici, E. Benedetti, and William Roberts. Enabling high accuracy dynamic applications in urban environments using ppp and rtk on android multi-frequency and multi-gnss smartphones. pages 1–9, 04 2019. (page 9)

[20] Zun Niu, Xinyang Zhao, Junren Sun, Lin Tao, and Bocheng Zhu. A continuous positioning algorithm based on rtk and vi-slam with smartphones. *IEEE Access*, 8:185638–185650, 2020. (page 9)

[21] Frank Fong Ling, Carmine Elvezio, Jacob Bullock, Steve Henderson, and Steven Feiner. A hybrid rtk gnss and slam outdoor augmented reality system. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1044–1045, 2019. (page 9)

[22] Marco Stranner, Clemens Arth, Dieter Schmalstieg, and Philipp Fleck. A high-precision localization device for outdoor augmented reality. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 37–41, 2019. (page 9, 23, 24)

[23] Randall Smith, Matthew Self, and Peter Cheeseman. A stochastic map for uncertain spatial relationships. pages 467–474, 01 1987. (page 9)

[24] Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1403–1410 vol.2, 2003. (page 9)

[25] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007. (page 9)

[26] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pages 83–86, 2009. (page 10)

[27] Carlos Campos, Richard Elvira, Juan J. Gomez, Jose M. M. Montiel, and Juan D. Tardos. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. (page 10)

[28] D. Nister, O. Naroditsky, and James Bergen. Visual odometry. volume 1, pages I–652, 01 2004. (page 10)

[29] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robot. Automat. Mag.*, 18:80–92, 12 2011. (page 10)

[30] Johannes Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. 06 2016. (page 10)

[31] Dinu Covaciu, Ion Preda, Dragos Dima, and Chiru Anghel. Study on the possibility to estimate the vehicle side slip using two independent gps receivers. *Applied Mechanics and Materials*, Vol. 822:321–330, 01 2016. (page 11)

[32] Jong-Hwa Yoon and Huei Peng. Vehicle sideslip angle estimation using two single-antenna gps receivers. volume 2, 09 2010. (page 11)

[33] Guilherme Trigo, Duarte Donas-Boto, Claudio Silva, and Jose E. Sanguino. Vehicle heading estimation using a two low-cost gps receiver configuration. In *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2011. (page 11)

[34] João Reis, Jose Sanguino, and António Rodrigues. Baseline influence on single-frequency gps precise heading estimation. *Wireless Personal Communications*, 64, 05 2012. (page 11)

[35] Anh Nguyen and Bac Le. 3d point cloud segmentation: A survey. pages 225–230, 11 2013. (page 12)

[36] N. Gelfand and Leonidas Guibas. Shape segmentation using local slippage analysis. volume 2004, pages 219–228, 01 2004. (page 12)

[37] P. Dorninger and Clemens Nothegger. 3d segmentation of unstructured point clouds for building modelling. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36, 01 2007. (page 13)

[38] D Tóvári and Norbert Pfeifer. Segmentation based robust interpolation- a new approach to laser data filtering. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36, 05 2012. (page 13)

[39] J. Strom, Andrew Richardson, and E. Olson. Graph-based segmentation for colored 3d laser point clouds. pages 2131 – 2136, 11 2010. (page 13)

[40] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. 12 2016. (page 13)

[41] Jun Tang. A color image segmentation algorithm based on region growing. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 6, pages V6–634–V6–637, 2010. (page 13)

[42] D. Crevier. Hue-based segmentation of color images. In *Proceedings of Canadian Conference on Electrical and Computer Engineering*, pages 1250–1253 vol.2, 1993. (page 13)

[43] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, New York, NY, USA, 2 edition, 2003. (page 13)

[44] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. (page 14)

[45] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. (page 14)

[46] Ben Bellekens, Vincent Spruyt, Raf Berkvens, and Maarten Weyn. A survey of rigid 3d pointcloud registration algorithms. 08 2014. (page 14)

[47] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, 2007. (page 14)

[48] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J. Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2647–2655, 2019. (page 14)

[49] Heng Yang, Jingnan Shi, and Luca Carlone. Teaser: Fast and certifiable point cloud registration. 2020. (page 14)

[50] Gerhard Schall, Daniel Wagner, Gerhard Reitmayr, Elise Taichmann, Manfred Wieser, Dieter Schmalstieg and Bernhard Hofmann-Wellenhof. Global Pose Estimation using Multi-Sensor Fusion for Outdoor Augmented Reality. *International Symposium on Mixed and Augmented Reality*, pages 153–162, 2009. (page 15)

[51] Min Sun, Liu Lei, Wei Huang, and Xiang Ren. Interactive registration for augmented reality gis. pages 246–251, 12 2012. (page 15)

[52] Gregorio Soria, Lidia Ortega, and Francisco Feito. Augmented and virtual reality for underground facilities management. *Journal of Computing and Information Science in Engineering*, 18:041008, 07 2018. (page 15)

[53] Ching-Tzun Chang, Ryosuke Ichikari, Koji Makita, Takashi Okuma, and Takeshi Kurata. [poster] road maintenance mr system using lrf and pdr. pages 204–205, 09 2015. (page 15)

[54] Stefanie Zollmann, Raphael Grasset, Gerhard Reitmayr, and Tobias Langlotz. Image-based x-ray visualization techniques for spatial understanding in outdoor augmented reality. 12 2014. (page 15)

[55] Geoff Zeiss and Dr. Sakura Shinoaki. Reducing damage to underground utility infrastructure during excavation. Technical report, Geospatial Information & Technology Association, 2020. (page 18)

[56] Dirt annual report of the common ground alliance. Technical report, 2019. (page 18)

[57] Lewis Makana, Nicole Metje, Ian Jefferson, and Chris Rogers. What do utility strikes really cost? 01 2016. (page 18)

[58] Lewis O Makana, Nicole Metje, Ian Jefferson, Margaret Sackey, and Chris D F Rogers. Cost estimation of utility strikes: towards proactive management of street works. *Infrastructure Asset Management*, 7(2):64–76, 2020. (page 19)

[59] Nicole Metje Richard Broome, Stephen Crossland and Andy Rhoades. Utility strike damage report. Technical report, 2020. (page 19)

[60] Ahmed Al-Bayati and Louis Panzer. Reducing damage to underground utilities: Lessons learned from damage data and excavators in north carolina. *Journal of Construction Engineering and Management*, 145:04019078, 09 2019. (page 19)

[61] Lasse H. Hansen, Erik Kjems, and Simon Wyke. Towards ar-enabled informed decision-making in subsurface utility projects through visualising 3d capture data of as-built utilities. Workingpaper, 2021. (page 20, 21)

[62] Maximilian Vogt, Adrian Rips, and Claus Emmelmann. Comparison of ipad pro®'s lidar and truedepth capabilities with an industrial 3d scanning solution. *Technologies*, 9:25, 04 2021. (page 23)

[63] Alessandra Spreafico, Filiberto Chiabrando, Lorenzo Teppati Losè, and Fabio Giulio Tonolo. The ipad pro built-in lidar sensor: 3d rapid mapping tests and quality assessment. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B1-2021:63–69, 06 2021. (page 23)

[64] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 2013. (page 24)

[65] Paul Furgale, Timothy D. Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. In *2012 IEEE International Conference on Robotics and Automation*, pages 2088–2095, 2012. (page 24)

[66] C94-m8p application board setup guide. `https://content.u-blox.com/sites/default/files/C94-M8P-Appboard-Setup_QuickStart_%28UBX-16009722%29.pdf`. (page 25)

[67] John Krumm. Intersection of two planes. May 2000. (page 51)