Diploma Thesis

Joint 3D surface and subsurface imaging for geophysical archaeological prospection

submitted in satisfaction of the requirements for the degree
Diplom-Ingenieur
of the TU Wien, Faculty of Mathematics and Geoinformation

Diplomarbeit

Kombinierte Oberflächen- und Untergrundprospektion mittels Punktwolken für geophysikalische archäologische Untersuchungen

ausgeführt zum Zwecke der Erlangung des akademischen Grads
Diplom-Ingenieur
eingereicht an der TU Wien, Fakultät für Mathematik und Geoinformation

Stefan Blochberger, BSc

Matr.Nr.: 01629030

Supervision: Assoz. Prof. Dipl. geophys. **Immo Trinks**, Privatdoz. PhD

Wien, March 2025

Affidavit

I declare in lieu of oath, that I wrote this thesis and carried out the associated research myself, using only the literature cited in this volume. If text passages from sources are used literally, they are marked as such. As supportive tools for coding, the Large Language Models (LLMs) ChatGPT from OpenAI[1] and, predominantly, Mistral AI's[2] Le Chat have been utilized.

I confirm that this work is original and has not been submitted for examination elsewhere, nor is it currently under consideration for a thesis elsewhere.

I acknowledge that the submitted work will be checked electronically-technically using suitable and state-of-the-art means (plagiarism detection software). On the one hand, this ensures that the submitted work was prepared according to the high-quality standards within the applicable rules to ensure good scientific practice "Code of Conduct" at the TU Wien. On the other hand, a comparison with other student theses avoids violations of my personal copyright.

Place and Date: _____ Signature: _____

[1] https://openai.com/
[2] https://mistral.ai/

Abstract

The integration of diverse point cloud sources – such as Terrestrial Laserscanning (TLS), Image Based Modelling (IBM), and Ground Penetrating Radar (GPR) – enables the creation of comprehensive 3D models of archaeological sites, imaging both surface and subsurface structures. Each data type captures distinct information: TLS provides detailed geometry, photogrammetry offers texture and overall structure, and GPR reveals subsurface features. Merging these data sets into a unified point cloud improves the visualisation and facilitates the topographic correction of GPR data, GPR anomaly detection, and digital twin creation, which are essential for archaeological research and cultural heritage preservation.

The proposed workflow optimises and georeferences TLS data while ensuring global consistency across multiple scans. Global reference is established by using Global Navigation Satellite System (GNSS) data of distinct features and airborne drone measurements. Comparisons to transform parameters using the geo-located IBM data are conducted and consequently preferred due to visual consistency. The resulting integrated 3D models not only enhance the understanding of the investigated cultural heritage sites but also improve the accuracy of subsurface and surface mapping by correlating overlapping structures from different sensing modalities.

This project also demonstrates an automated framework, in which all data are treated as point clouds and merged following precise referencing. A dedicated C++ tool facilitates the optimisation process, and the integration is performed within a Geographic Information System (GIS) framework called Nubigon, suited for the handling of very large data sets. This type of system is pivotal in ensuring that both high-resolution surface details and subsurface anomalies are accurately represented, thereby supporting more effective conservation and research efforts. The custom developed tool is set to be published as open source software, therefore facilitating free access and the possibility of contribution by a broader user base. This encourages transparency and accessibility of software used for the specific purpose of aligning multiple point clouds and optimising the underlying pose graph.

GPR data revealed subsurface structures and features within the manor grounds. Anomalies caused by buried walls and foundations of a suspected orangery were discovered beneath the meadow of a terrace adjacent to the building complex. These 3D point cloud data sets and contained structures have been jointly visualized using Nubigon, which overlays IBM surface data with GPR subsurface data to provide a comprehensive representation.

Contents

Acronyms

AR Augmented Reality

BnB Branch and Bound

CSF Cloth Simulation Filter

DJI Da-Jiang Innovations Science and Technology Co., Ltd

GICP Generalized Iterative Closest Point

GIS Geographic Information System

GNSS Global Navigation Satellite System

GPR Ground Penetrating Radar

IBM Image Based Modelling

ICP Iterative Closest Point

IMU Inertial Measurement Unit

LiDAR Light Detection And Ranging

LM Levenberg–Marquardt algorithm

LOAM LiDAR Odometry and Mapping

MVS Multi View Stereo

PnP Perspective-n-Point

RLE Run-length encoding

SfM Structure from Motion

SVD Single Value Decomposition

TIN Triangulated Irregular Network

TLS Terrestrial Laserscanning

VR Virtual Reality

Chapter 1

Introduction

Joint 3D subsurface and surface imaging using a combination of Ground Penetrating Radar (GPR), Image Based Modelling (IBM), and Terrestrial Laser Scanning (TLS) has emerged as a powerful approach for noninvasive archaeological and historical site investigations and documentation (Schmidt et al. 2023; Merkle et al. 2020; Trinks et al. 2018; Wolf et al. 2018). Merging geoarchaeological prospection data from these methods into a unified 3D point cloud visualization offers several benefits:

Enhanced Data Integration and Interpretation

Integrating datasets allows for a multi-layered analysis where surface and subsurface features can be directly compared. This correlation of topographical details, obtained from IBM and laser scanning, with buried structures identified by GPR, enhances the spatial context. Subsurface anomalies detected by GPR can be better understood in relation to above-ground features such as walls, ditches, or terrain changes. Merging all datasets into a common 3D environment ensures better spatial alignment and reduces interpretation errors, providing a seamless overlay of different data types.

Increased Accuracy in Spatial Positioning

Combining point clouds from different sources allows for cross-validation and refinement of spatial positioning, which is particularly useful in complex terrains or forested areas where GNSS may be unreliable. Laser scanning provides high-precision georeferencing that can be used to align and correct IBM and GPR datasets, ensuring georeferencing consistency and improved data registration.

Improved Visualization and Communication

A more intuitive 3D representation allows researchers, stakeholders, and the public to better understand archaeological features by visualizing them in a realistic 3D space, including both visible and hidden structures. Point cloud-based 3D models facilitate interactive exploration, enabling sectioning and filtering of different data layers to focus on specific aspects. This enhanced visualization is a powerful tool for archaeologists, heritage managers, and the public, making discoveries more tangible and accessible, thereby improving stakeholder engagement.

Facilitating Advanced Data Processing and Analysis

Merged 3D datasets can serve as input for AI and machine learning applications, enabling automated feature recognition, segmentation, or classification using deep learning techniques.

Comparing GPR point clouds with surface data may allow for semi-automated reconstruction of stratigraphic sequences. Additionally, this integration enables quantitative analysis, such as calculating the volume of buried structures or erosion patterns.

Cross-Validation of Different Methods

The integration of different datasets allows for the confirmation of subsurface features detected by GPR by cross-checking with terrain data from laser scanning or IBM, ruling out modern disturbances. Differences between datasets can help refine interpretations, distinguishing natural geological formations from archaeological structures. Furthermore, laser scanning can provide topographic corrections for GPR, reducing errors caused by uneven terrain and improving the overall interpretation of GPR data.

Digital Twins as Investigative Tool

Digital twins of the manor houses shall serve art historians and archaeologists in their investigations. This approach is likewise applicable to the investigation of archaeological sites using non-invasive geophysical prospection methods, such as GPR. So far, the surfaces and environment in which a prospection survey is conducted are normally not recorded or documented, leading to incomplete data sets. The combined surface and subsurface mapping and imaging will lead to more comprehensive, more complete and more realistic data representations.

1.1 Choosing Imaging Methods

Image Based Modeling (IBM)

In the context of archaeological prospection, mapping the surface of sites with color and texture information is essential for digital documenting, and can be used to virtually preserving cultural heritage sites. For efficient and comprehensive acquisition of surface structures, IBM serves as a well suited, low cost method. Given most archaeological sites are geometric structures with distinct textural features, the usual pitfalls for IBM, like modeling textureless or vegetational areas, are mostly avoided. The nature of the IBM method facilitates high resolution documentation of textural features, while providing sufficient geometrical accuracy for archaeological purposes. This efficient acquisition technique is suited for large scale imaging as well as high resolution capturing of intricate textural details. Given low occlusion by surrounding objects, the GNSS receiver onboard a drone used for IBM provides global reference of the investigated object. In the context of this thesis, exterior structures of the Hafslund manor have been mapped using drone powered IBM.

Terrestrial Laser Scanning (TLS)

TLS point clouds can provide more geometrically accurate data and can serve as a reference to captured IBM data. Given sufficient overlap between adjacent scans and minimized error in Iterative Closest Point (ICP)-based matching, structural features are acquired with higher reliability compared to IBM. In this thesis, TLS data captured of the exterior and interior of the manor has been processed and analyzed.

Ground Penetrating Radar (GPR)

While TLS and IBM provide surface level data, GPR facilitates investigation of the corresponding subsurface (Conyers 2023). Given recent developments in terms of data resolution (Trinks et al. 2018), and reliable results in near-surface archaeological investigations, this method is well suited for efficient and robust subsurface data acquisition.

1.2 Manors in the Baltic Sea Region

In the framework of a collaboration between the University of Vienna, Vienna Institute for Archaeological Science (VIAS), GeoSphere Austria, and the Research Centre for Manors in the Baltic Sea Region[1] of the University of Greifswald, selected manor houses have been investigated as pilot studies. The purpose of these studies was to develop a workflow for simple and efficient digital 3D documentation of manor houses and their surroundings, as well as to use geophysical prospection techniques to survey areas in the vicinity of the main buildings. Predominantly georadar measurements have been used for the geophysical archaeological prospection surveys. One application of this joint 3D surface and subsurface imaging was attempted at Hafslund Manor in Norway, testing and showcasing its potential for the non-destructive, comprehensive documentation of cultural heritage sites.

1.2.1 Hafslund Manor

According to the Directorate for Cultural Heritage (Riksantikvaren)[2], Hafslund is one of Norway's largest manor estates (figure 1.1). The features of the surrounding landscape help explain how this power center emerged: to the west lie the Glomma River and Sarpsfossen, with its associated industry and power production; to the north and east are National Road 118 (formerly E6) and the railway – vital connections to international markets; and to the south unfolds a rich agricultural landscape. The three-winged main building, constructed between 1758 and 1762, was erected partly by reusing masonry from the structure that burned down in 1757. This masonry dates back to the 1600s and early 1700s. Today, the north wing serves as the main wing and comprises two full stories, while the east and west wings consist of a single story with an adapted attic. The main building features plastered, white-painted walls with impressive rows of windows and a high mansard roof adorned with glazed roof tiles. The external staircases, as well as the estate's fences and gates, were added during a restoration led by Arnstein Arneberg in 1936–37. Although Arneberg also influenced the interiors, much of the original, high-quality 18th-century decoration remains preserved.

As an imposing Baroque complex, Hafslund stands alongside Jarlsberg as one of Norway's premier manor estates. The design makes deliberate use of axes and symmetry in both the park and the buildings. The outbuildings conclude at the main building with two pavilions rendered in brick. Remnants of a formal parterre garden with Renaissance features and a Baroque-style landscape park with monumental linden avenues further underscore the historical significance of the estate.

Today, the manor houses meeting and conference facilities for Hafslund ASA[3].

[1] https://cdfi.uni-greifswald.de/en/history-of-art/research/research-centre-for-manors-in-the-baltic-sea-region/
[2] https://riksantikvaren.no/en/
[3] https://www.hafslund.no/en

Fig. 1.1: Hafslund manor, as captured by drone imaging. Photo: I. Trinks.

1.2.2 Other relevant manors

Other manor houses investigated with an approach similar to that used at Hafslund Manor – combining IBM, TLS, and GPR – include Fossesholm in Norway, Stola in Sweden, Kolga in Estonia, Nuhjala in Finland, and Orellen–Ungurmuiža in Latvia. The integrated surface and subsurface imaging presented in this thesis would be highly applicable to these sites. Beyond manor houses, the 3D documentation of Insula IV in Regio VII at Pompeii by the Austrian Archaeological Institute, employing the same three methods, offers a comparable scenario.

1.3 Creating and Visualising Unified 3D Point Cloud

This thesis deals with the combination and integration of the different data sets collected at Hafslund Manor by correctly referencing them to each other. All involved data sets can be regarded and presented as 3D point clouds, which is obvious for TLS and IBM data, but less so for GPR data. It was Kamp et al. (2014) who first suggested to visualise GPR data as a 3D point cloud. The process of combining these point clouds involves correctly referencing individual data sets to each other, followed by merging and visualization. The ultimate outcome of the project will be a comprehensive workflow that outlines the acquisition, processing, and integration of 3D surface and subsurface data.

1.4 Related Works

The approach by Merkle et al. 2020 was to combine LiDAR and GPR data, using two GNSS antennas in combination with an Inertial Measurement Unit (IMU) in order to reference the datasets. The main focus of this work was to provide additional subsurface information in infrastructural use cases like utility mapping, rebar localization within concrete or void detection.

Topographic correction of the GPR data, using the LiDAR data, ensures consistent ground height. Furthermore, this work shows that the LiDAR cloud as reference, increases interpretability of GPR data and allows measurements between subsurface and surface structures.

Trinks et al. 2018 states that over the past decade, the introduction of advanced multi-channel GPR antenna array systems has significantly enhanced survey efficiency and spatial sampling resolution. By utilizing GPR antenna arrays with up to 16 channels operating simultaneously, combined with automatic positioning solutions like real-time kinematic global navigation satellite systems or robotic total-stations, it is now feasible to map several hectares per day with a trace spacing of 8 cm cross-line and 4 cm in-line. This substantial increase in coverage not only reduces the costs of GPR surveys, making them more accessible for archaeological applications, but also enables high-resolution imaging of relatively small archaeological features. For instance, structures such as 25 cm wide post-holes from Iron Age buildings or the brick pillars of Roman floor heating systems can now be clearly imaged, leading to much improved archaeological interpretations of the data collected.

Based on the findings mentioned, the trend towards handling high volumes of data, potentially processed and referenced in real-time, is becoming increasingly important. Developing an efficient framework for integrating multiple datasets, particularly point clouds, is crucial for various prospecting applications. This thesis aims to cover parts of this agenda.

Chapter 2

Material and Methods

2.1 Ground Penetrating Radar (GPR)

Ground Penetrating Radar (GPR) is a non-invasive remote sensing technology that utilizes electromagnetic waves in the radar frequency range to create subsurface images by detecting reflected signals. The practical result of the radiation of electromagnetic waves into the subsurface for GPR measurements is shown by the basic operating principle that is illustrated in figure 2.1. GPR is designed primarily for investigating the shallow subsurface of the earth, building materials, and infrastructure such as roads and bridges. Developed over the past thirty years, GPR enables shallow, high-resolution investigations. It is a time-dependent geophysical method that can provide a 3D pseudo-image of the subsurface, including depth estimates for many common subsurface objects. Under favourable conditions, GPR can deliver precise information about buried objects. Additionally, GPR can be deployed in boreholes to extend the range of investigations away from the boundary of the hole.

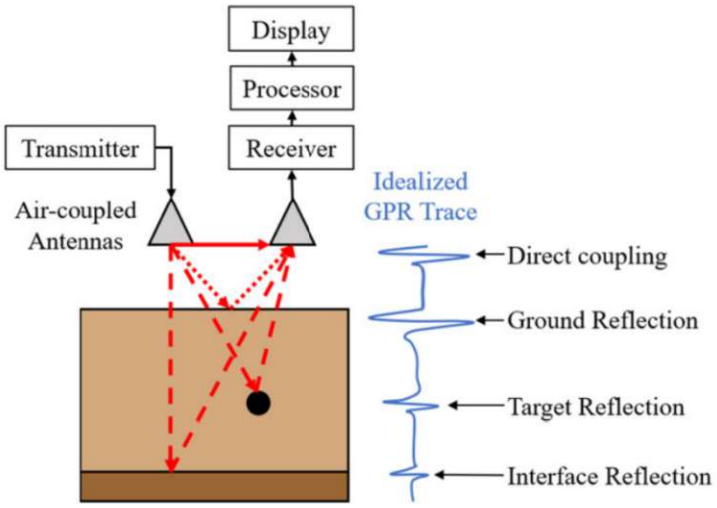


Fig. 2.1: Scheme of an air-coupled GPR system (Merkle et al. 2020).

2.1.1 Electromagnetic Theory

GPR relies on the principles of electromagnetic (EM) wave propagation, rooted in Maxwell's equations. These fundamental equations describe the physics of electromagnetic fields and their interaction with materials. Constitutive relationships link material properties to electromagnetic fields. Material can be characterized based on their response to transient EM fields. The key properties for GPR are dielectric permittivity, electrical conductivity, and magnetic permeability (Reynolds 2011).

Maxwell's Equations and Constitutive Relationships:

Maxwell's equations form the foundation of EM theory, describing how electric and magnetic fields propagate and interact with materials. The constitutive relationships are:

Dielectric Permittivity (ϵ) : measures how a material polarizes in response to an electric field, affecting how EM waves propagate through it.

$$\mathbf{D} = \epsilon\mathbf{E} = \epsilon_r\epsilon_0\mathbf{E} \tag{2.1}$$

where:

- $\mathbf{D}$ is the electric displacement vector [C/m^2]
- $\mathbf{E}$ is the electric field strength vector [V/m]
- ϵ_r is the relative permittivity
- ϵ_0 is the permittivity of free space (8.85×10^{-12} [F/m])

Electrical Conductivity (σ) : describes how free charges move to create an electric current when an electric field is applied.

$$\mathbf{J} = \sigma\mathbf{E} \tag{2.2}$$

where:

- $\mathbf{J}$ is the electric current density vector [A/m^2]
- $\mathbf{E}$ is the electric field strength vector [V/m]
- σ is the electrical conductivity [S/m]

Magnetic Permeability (μ) : measures a material's ability to support the formation of a magnetic field within itself.

$$\mathbf{B} = \mu\mathbf{H} = \mu_r\mu_0\mathbf{H} \tag{2.3}$$

where:

- $\mathbf{B}$ is the magnetic flux density vector [T]
- $\mathbf{H}$ is the magnetic field strength vector [A/m]
- μ_r is the relative permeability
- μ_0 is the permeability of free space ($4\pi \times 10^{-7}$ [H/m])

Electromagnetic Wave Propagation

In simple, isotropic, linear media, Maxwell's equations combined with the constitutive relationships describe the propagation of EM waves. The movement of electric charges generates a current $\mathbf{J}$, which produces a magnetic field $\mathbf{B}$. This field induces an electric field $\mathbf{E}$, perpetuating the propagation of EM waves through the medium (see Figure 2.2).

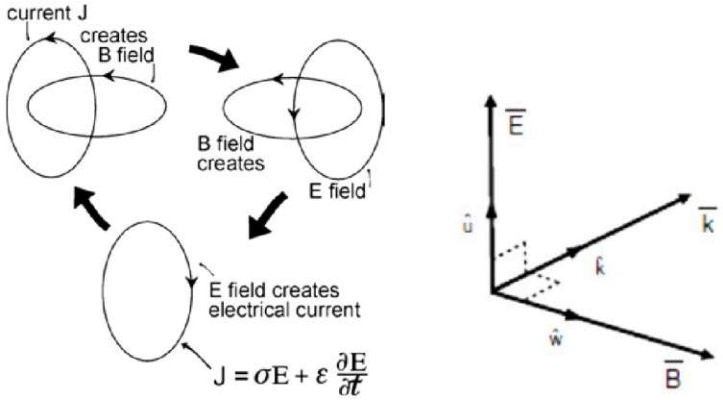


Fig. 2.2: Propagation of EM waves where changing electric fields create magnetic fields, and vice versa (Reynolds 2011).

Wave Equation: The propagation of EM waves is described by the transverse vector wave equation, derived from Faraday's law, Ampere's law, and the constitutive relations:

$$\nabla^2 \mathbf{E} - \mu\epsilon \frac{\partial^2 \mathbf{E}}{\partial t^2} = 0, \tag{2.4}$$

where $\mathbf{E}$ is the electric field, μ is the magnetic permeability, and ϵ is the dielectric permittivity of the medium (D. J. Daniels 2004). Both electric and magnetic fields satisfy this wave equation.

Wave Propagation in Frequency Domain: Transforming the wave equation into the frequency domain via Fourier transformation, assuming sinusoidal time variation, the propagation constant k is:

$$k = \alpha + i\beta = \frac{\omega}{v}\sqrt{\mu\epsilon}\left(1 + i\frac{\sigma}{\omega\epsilon}\right) \tag{2.5}$$

where:

- ω is the angular frequency
- v is the phase velocity
- α is the attenuation constant
- β is the phase constant

Lossy Media: For real earth materials, losses are often due to electrical conductivity. When losses are low, i.e.,

$$\omega\mu\sigma \ll \omega\epsilon \tag{2.6}$$

or equivalently,

$$\sigma \ll \omega\epsilon \tag{2.7}$$

the wave properties simplify to:

$$v = \frac{1}{\sqrt{\mu\epsilon}} \tag{2.8}$$

$$\alpha = \frac{\mu\sigma}{2v} \tag{2.9}$$

$$Z = \sqrt{\frac{\mu}{\epsilon}} \tag{2.10}$$

where Z is the electromagnetic impedance.

Frequency and Conductivity Dependence: The variation of wave properties with frequency and conductivity is shown in Figure 2.3. At low frequencies, wave properties are frequency-dependent. As frequency increases, water relaxation frequency effects and scattering losses become significant (Reynolds 2011).

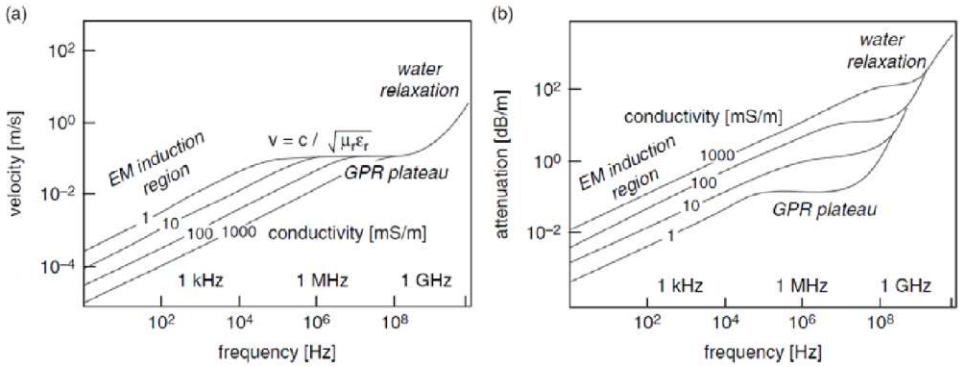


Fig. 2.3: Variation of wave velocity and attenuation with frequency for different conductivities (Reynolds 2011).

In the GPR frequency range (10 MHz – 2 GHz), velocity is generally independent of frequency and conductivity. However, attenuation increases with both frequency and conductivity, limiting penetration depth (Reynolds 2011).

2.1.2 Reflection and Transmission of EM Waves

When an EM wave encounters an interface between two materials with different electrical properties, part of the wave is reflected back and part is transmitted through the interface. This section provides a detailed scientific explanation of these processes.

Reflection Coefficient: The reflection coefficient R at an interface between two materials is given by the ratio of the reflected electric field amplitude to the incident electric field amplitude. For normal incidence, it can be expressed as:

$$R = \frac{Z_2 - Z_1}{Z_2 + Z_1} \tag{2.11}$$

where Z_1 and Z_2 are the impedances of the first and second materials, respectively. The impedance Z is related to the permittivity ϵ and permeability μ of the material by:

$$Z = \sqrt{\frac{\mu}{\epsilon}} \tag{2.12}$$

Transmission Coefficient: The transmission coefficient T is given by the ratio of the transmitted electric field amplitude to the incident electric field amplitude:

$$T = \frac{2Z_2}{Z_2 + Z_1} \tag{2.13}$$

Snell's Law: The angles of incidence θ_i and refraction θ_t at the interface between two materials are related by Snell's Law:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{v_1}{v_2} = \sqrt{\frac{\epsilon_2}{\epsilon_1}} \tag{2.14}$$

where v_1 and v_2 are the velocities of the wave in the first and second materials, respectively.

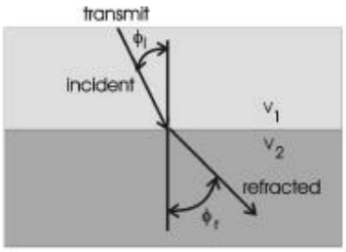


Fig. 2.4: Reflection and transmission of an EM wave at an interface (J. J. Daniels 2000).

Wave Propagation Velocity: The velocity v of an EM wave in a material is related to the permittivity ϵ and permeability μ of the material:

$$v = \frac{1}{\sqrt{\epsilon\mu}} \tag{2.15}$$

In most non-magnetic materials, $\mu \approx \mu_0$ (the permeability of free space), so the velocity is primarily determined by the permittivity ϵ.

2.1.3 Scattering Mechanisms

When an EM wave encounters an object with different permittivity, various scattering mechanisms can occur, including specular reflection, diffraction, resonant scattering, and refraction.

Specular Reflection: follows the Law of Reflection, where the angle of incidence equals the angle of reflection:

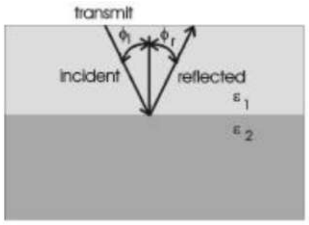
$$\theta_i = \theta_r \tag{2.16}$$

This type of scattering is common in GPR when the interface is smooth and planar (see Figure 2.5a).

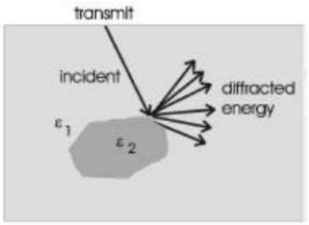
Refraction Scattering: occurs when part of the wave transmits through the interface into the second material, changing direction according to Snell's Law (see Figure 2.5b).

Diffraction Scattering: occurs when the wave encounters a sharp boundary or obstacle, causing the wave to bend and spread out. The behavior of diffracted waves can be described by Huygen's Principle (see Figure 2.5c).

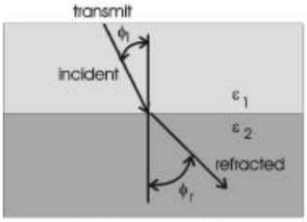
Resonant Scattering: occurs when the wave impinges on a closed object, causing the wave to resonate within the object. The resonant frequency depends on the size of the object and the material properties (see Figure 2.5d).



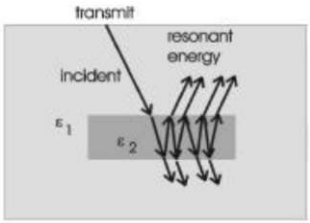
(a) spectral reflection scattering



(b) Refraction scattering



(c) Diffraction scattering



(d) Resonant scattering

Fig. 2.5: Scattering mechanisms: (a) specular reflection scattering, (b) refraction scattering, (c) diffraction scattering, and (d) resonant scattering (J. J. Daniels 2000).

2.1.4 Processing of GPR data

The processing of GPR data is crucial for accurate interpretation and involves several steps aimed at improving the signal-to-noise ratio and the resolution of the final image. This section builds upon the techniques discussed by Benedetto et al. 2017 and covers data acquisition, preprocessing, signal enhancement, migration, and interpretation.

GPR Data Representation: A-Scan, B-Scan, and C-Scan

GPR data can be represented in different formats, each providing unique insights into the subsurface structure. The most common representations are A-scan, B-scan, and C-scan.

A-Scan: (Amplitude scan) represents a single trace of reflected signal amplitudes over time. It provides a one-dimensional view of the subsurface at a single location. The vertical axis represents time (or depth), and the horizontal axis represents the amplitude of the reflected signal. The A-scan is useful for analyzing the signal characteristics at a specific point.

$$A(t) = R(t), \tag{2.17}$$

where $A(t)$ is the amplitude of the reflected signal at time t, and $R(t)$ is the recorded signal.

B-Scan: (Brightness scan) is a two-dimensional cross-sectional image of the subsurface, generated by combining multiple A-scans collected along a survey line. The horizontal axis represents the position along the survey line, while the vertical axis represents time (or depth).

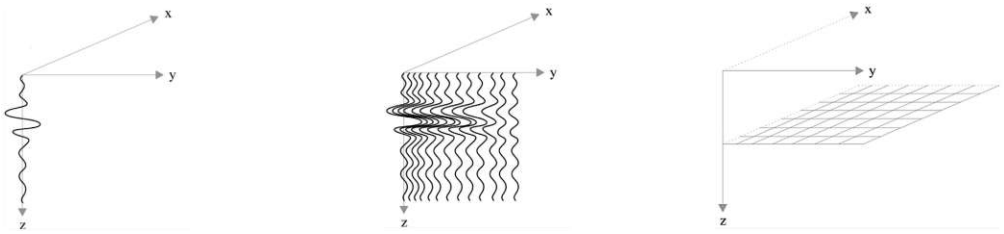
$$B(x,t) = R(x,t), \tag{2.18}$$

where $B(x,t)$ is the B-scan image, x is the position along the survey line, and t is the time (or depth). B-scans are also referred to as 2D GPR profile sections.

C-Scan: (Constant depth scan) provides a plan view (horizontal slice) of the subsurface at a constant depth. It is created by extracting data at a specific time (or depth) from multiple B-scans collected over a grid of survey lines. The intensity of each pixel in the resulting image corresponds to the amplitude of the reflected signal. The C-scan is useful for mapping the spatial distribution of subsurface features.

$$C(x, y) = R(x, y, t_0), \tag{2.19}$$

where $C(x, y)$ is the C-scan image, x and y are the coordinates of the survey grid, and t_0 is the selected time (or depth). C-scans are also referred to as GPR time-slices or GPR depth-slices.



(a) A-scan **(b)** B-scan **(c)** C-scan

Fig. 2.6: Illustration of A-scan, B-scan, and C-scan representations in GPR data (Benedetto et al. 2017).

Preprocessing

Preprocessing steps include time-zero correction, dewow filtering, and gain adjustment. These steps aim to remove noise and correct for systematic errors in the data.

Time-Zero Correction: adjusts the recorded data to ensure that the start time of each trace is aligned. This correction is necessary because the initial pulse may not coincide with the zero-time point due to system delays.

Dewow Filtering: removes low-frequency noise, typically caused by the GPR system's electronics. This is achieved by applying a high-pass filter to the data. A common approach is to use a moving average filter to subtract the low-frequency component:

$$d_{dewowed}(t) = d(t) - \frac{1}{N} \sum_{i=-N/2}^{N/2} d(t + i), \tag{2.20}$$

where $d(t)$ is the original signal and N is the window size of the moving average (Cassidy and Jol 2009).

Gain Adjustment: compensates for the attenuation of the signal with depth. This can be done using exponential or automatic gain control (AGC):

$$d_{gain}(t) = d(t) \cdot e^{\alpha t}, \tag{2.21}$$

where α is a constant that controls the gain factor (Cassidy and Jol 2009).

Signal Enhancement

Signal enhancement techniques include filtering, background removal, and deconvolution. These methods improve the clarity and resolution of the GPR data.

Filtering: Band-pass filters are often used to remove both low-frequency and high-frequency noise from the data. The filter can be represented in the frequency domain as:

$$D_{filtered}(f) = H(f)D(f), \tag{2.22}$$

where $D(f)$ is the Fourier transform of the original signal, and $H(f)$ is the filter function (D. J. Daniels 2004).

Background Removal: Background removal techniques eliminate consistent noise patterns, such as antenna ringing. One common method is to subtract the average trace from each trace in the data set:

$$d_{background\_removed}(t) = d(t) - \frac{1}{M}\sum_{i=1}^{M} d_i(t), \tag{2.23}$$

where M is the number of traces (Cassidy and Jol 2009).

Deconvolution: aims to collapse the wavelet to a spike, improving resolution. The Wiener deconvolution filter is commonly used:

$$W(f) = \frac{S^*(f)}{S(f)S^*(f) + N(f)N^*(f)}, \tag{2.24}$$

where $S(f)$ is the signal spectrum, $N(f)$ is the noise spectrum, and $*$ denotes the complex conjugate (Cassidy and Jol 2009).

Migration

Migration is the process of repositioning reflected signals to their correct locations in space, accounting for the propagation path of the waves. The Kirchhoff migration algorithm is widely used:

$$d_{migrated}(x,z) = \int\int d(x_s,t_r)\delta\left(t_r - \frac{\sqrt{(x-x_s)^2+z^2}}{v}\right)dx_s dt_r, \tag{2.25}$$

where x_s is the source position, t_r is the travel time, v is the wave velocity, and δ is the Dirac delta function (Yilmaz 2001).

Topographic Correction

The primary goal of topographic correction is to remove the distortions caused by surface elevation changes. In GPR surveys conducted over undulating terrain, the recorded signal travel times are influenced by the varying distance between the radar antenna and the subsurface reflectors. Without correction, these variations can lead to misinterpretation of subsurface features and inaccurate depth estimations (Neal 2004).

Topographic correction involves several steps, which can be summarized as follows:

1. **Surface Elevation Measurement:** Accurate measurement of the surface elevation at each GPR survey point is essential. This can be achieved using differential GPS (DGPS) or total station surveying (D. J. Daniels 2004), as well as topographic recording using image based modelling or Lidar-based sensors (see the following sections).

2. **Travel Time Adjustment:** The recorded two-way travel times (t_{obs}) are adjusted based on the measured surface elevations. The corrected travel time (t_{corr}) for a given depth is calculated using the following equation:

$$t_{corr}(x) = t_{obs}(x) - \frac{2h(x)}{v}, \tag{2.26}$$

   where $h(x)$ is the surface elevation at position x, and v is the velocity of the radar wave in the medium (Grasmueck 1996).

3. **Data Interpolation:** Interpolating the GPR data onto a regular grid that accounts for the corrected travel times. This step ensures that the subsurface reflections are accurately represented in a consistent spatial framework.

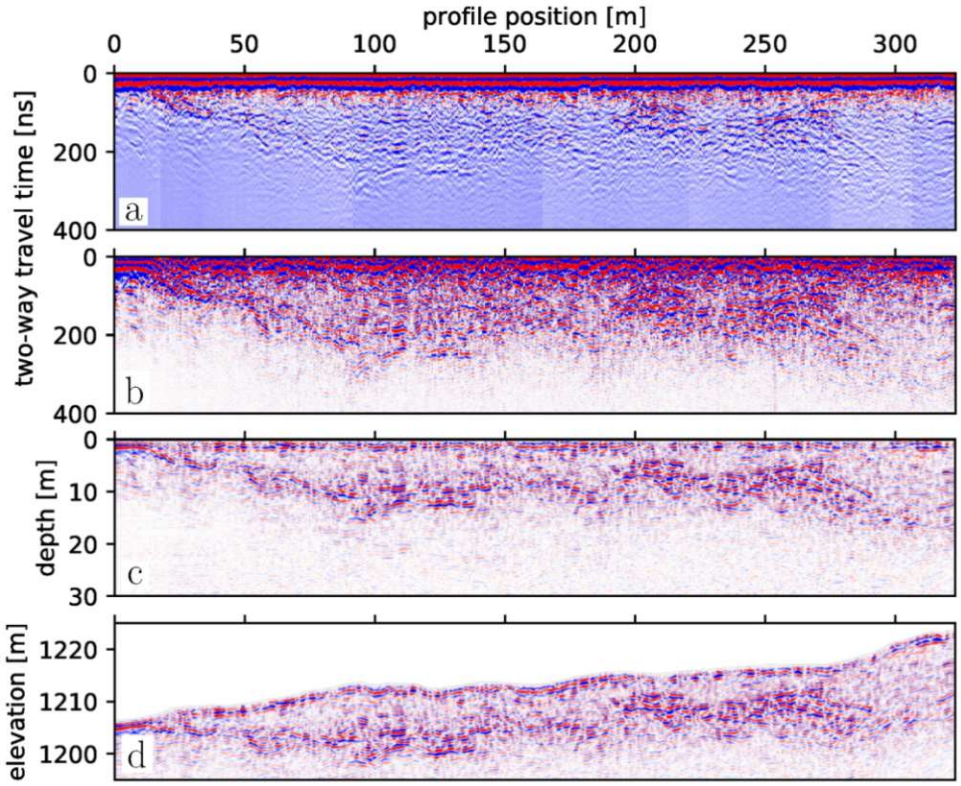


Fig. 2.7: Example for common-offset data processing steps. (a) Raw data. (b) After trace alignment, dewow, and mean trace removal. (c) After along-trace smoothing (window width 10 samples), t1.2 gain, time-to-depth conversion using an RMS velocity of 0.1 m/ns, f-k migration, and along-profile smoothing (oversampling factor 4, running mean window width 6 traces) (d) After topographic correction (Plattner 2020).

2.2 Image Based Modeling (IBM)

Image-based modeling, otherwise referred to as photogrammetry, incorporates the principles of Structure from Motion (SfM) to reconstruct three-dimensional representations from two-dimensional images. The fundamental idea is to estimate both the 3D structure of the scene and the motion of the camera.

Structure from Motion (SfM)

SfM is a foundational technique within photogrammetry that enables the estimation of camera positions and scene structure simultaneously from a set of overlapping images. This process involves identifying common features or points across multiple images, calculating their positions in 3D space through triangulation, and refining the camera parameters to minimize errors (Schonberger and Frahm 2016).

The SfM pipeline typically involves the following key steps:

1. **Feature Detection and Matching:** Extract and match distinctive features across multiple images.
2. **Geometric Verification:** Estimate the relative pose between image pairs based on the matched features.
3. **Incremental Reconstruction:** Build a sparse 3D point cloud incrementally by adding new images (see figure 2.8).
4. **Bundle Adjustment:** Refine the 3D structure and camera parameters by minimizing the reprojection error.
5. **Dense Reconstruction (Optional):** Generate a dense 3D model from the sparse point cloud.

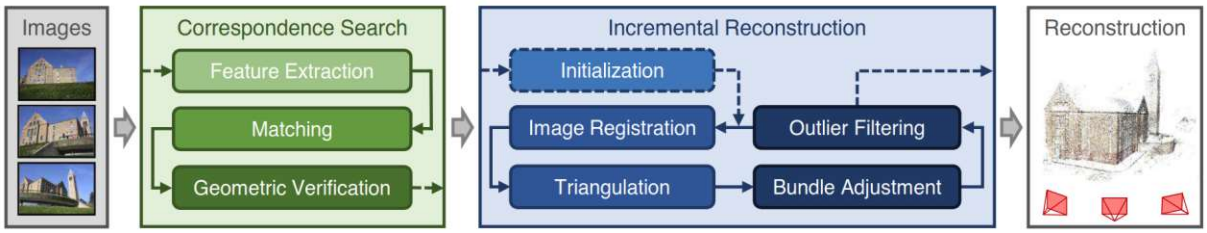


Fig. 2.8: Pipeline of an incremental SfM algorithm (Schonberger and Frahm 2016).

2.2.1 Feature Detection and Matching

The first step in SfM involves detecting and matching features across images. Reliable feature detection and matching are crucial for accurate 3D reconstruction. Common algorithms for feature detection include the Scale-Invariant Feature Transform (SIFT) (Lowe 2004) and the Speeded-Up Robust Features (SURF) (Bay et al. 2008).

Scale-Invariant Feature Transform (SIFT)

The Scale-Invariant Feature Transform (SIFT) algorithm, developed by David Lowe (Lowe 2004), is a key method for detecting and describing local features in images. SIFT features are robust to scale changes, rotations, and to some extent, affine transformations and illumination variations. The algorithm consists of several stages:

Scale-Space Extrema Detection: The first stage of SIFT is to detect potential keypoints by finding extrema in the scale-space. Scale-space is constructed by applying Gaussian blurring at different scales to the image (Figure 2.9a). The scale-space function $L(x, y, \sigma)$ is defined as:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.27}$$

where $G(x, y, \sigma)$ is the Gaussian kernel with standard deviation σ, and $I(x, y)$ is the input image. To identify keypoints, a series of Difference-of-Gaussian (DoG) images are computed by subtracting consecutive blurred images:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma), \tag{2.28}$$

where k is a constant factor. Keypoints are detected as local maxima and minima in the DoG images across both spatial and scale dimensions (Figure 2.9b).

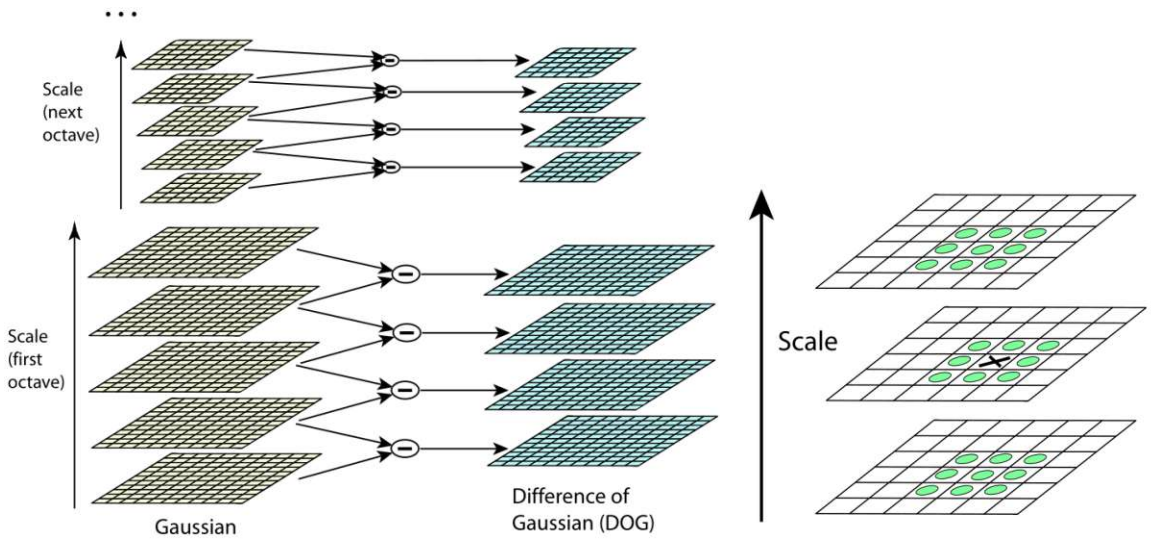


Fig. 2.9: (a) For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated. (b) Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3×3 regions at the current and adjacent scales (marked with circles) (Lowe 2004).

Keypoint Localization: Keypoints are refined to achieve sub-pixel accuracy and eliminate keypoints that are poorly localized or have low contrast. This is achieved by fitting a 3D

quadratic function to the local neighborhood of the keypoint using Taylor expansion. The function $D(\mathbf{x})$ is:

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}, \tag{2.29}$$

where $\mathbf{x} = (x, y, \sigma)^T$ is the displacement from the candidate keypoint.

Orientation Assignment: Each keypoint is assigned an orientation to achieve invariance to image rotation. The local dominant gradient angle is computed and used as a reference orientation. Then, the local gradient distribution is normalized with respect to this reference direction. Gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are calculated as:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}, \tag{2.30}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right). \tag{2.31}$$

An orientation histogram is created from the gradient orientations within a region around the keypoint, and the peak of this histogram defines the dominant orientation (see Figure 2.10).

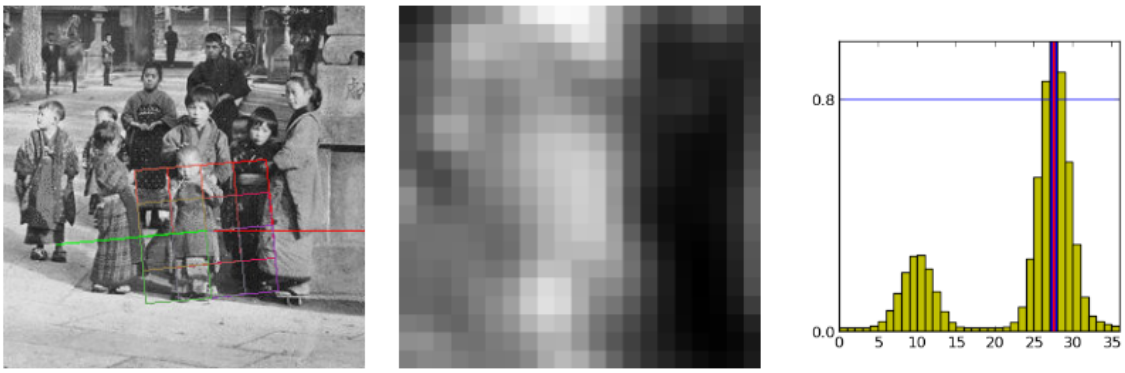


Fig. 2.10: Reference Orientation Attribution: The gradient orientations are computed from the scaled image (middle), collected into a histogram (right) and then smoothed (Rey Otero 2015).

Keypoint Descriptor Generation: The final stage involves creating a descriptor for each keypoint, which is a vector representing the local gradient distribution around the keypoint. The region around each keypoint is divided into a 4×4 grid, and within each cell, an orientation histogram with 8 bins is computed. The descriptor vector is then formed by concatenating these histograms, resulting in a 128-dimensional vector for each keypoint (see Figure 2.11).

2.2.2 Geometric Verification

Estimation of the Essential or Fundamental Matrix

Once feature matches between pairs of images are established, the next step is to estimate the essential or fundamental matrix. The choice between these matrices depends on whether the intrinsic parameters of the cameras (e.g., focal length, principal point) are known.

Fig. 2.11: SIFT descriptor construction: The patch around the detected keypoint is scaled down, normalized and oriented (middle). It is further partitioned into 16 subpatches for which orientation histograms with 8 bins are computed, resulting in a 128-dimensional vector. Features from different images are then matched by comparing their descriptors, typically using a nearest-neighbor search (Rey Otero 2015).

Essential Matrix: The essential matrix, denoted as $\mathbf{E}$, is used when the intrinsic parameters of the cameras (such as focal length, principal point, and lens distortion) are known. It encapsulates the relative rotation $\mathbf{R}$ and translation $\mathbf{t}$ between two camera positions. The relationship between corresponding points $\mathbf{x}$ and $\mathbf{x}'$ in two images can be expressed as:

$$\mathbf{x}'^{\top}\mathbf{E}\mathbf{x} = 0, \tag{2.32}$$

where $\mathbf{E}$ can be decomposed as $\mathbf{E} = [\mathbf{t}]_{\times}\mathbf{R}$. Here, $[\mathbf{t}]_{\times}$ is the skew-symmetric matrix of the translation vector $\mathbf{t}$, and $\mathbf{R}$ is the rotation matrix. This equation represents the epipolar constraint, which states that a point in one image will be mapped to a corresponding epipolar line in the second image (Figure 2.12). The essential matrix effectively describes how the camera's viewpoint changes from one image to another. It is derived from the intrinsic camera calibration and the relative pose of the cameras. The essential matrix can be estimated using the normalized eight-point algorithm (Hartley and Zisserman 2003). This algorithm involves normalizing the coordinates of the matched points, solving a linear system to find an initial estimate, and then enforcing the rank-2 constraint on $\mathbf{E}$.

Fundamental Matrix: The essential matrix serves as a foundation for the fundamental matrix, $\mathbf{F}$. Both matrices are instrumental in defining constraints between corresponding points in images. However, the fundamental matrix is applicable only when the cameras are calibrated, as it requires knowledge of the intrinsic camera parameters (matrices $\mathbf{K}$ and $\mathbf{K}'$) for normalization (Hartley and Zisserman 2003). Conversely, if the cameras are calibrated, the essential matrix is valuable for determining the relative position and orientation of the cameras, as well as the 3D positions of matching image points. The relationship between the essential matrix and the fundamental matrix is given by:

$$\mathbf{E} = (\mathbf{K}')^{\top}\mathbf{F}\mathbf{K}^{-1}. \tag{2.33}$$

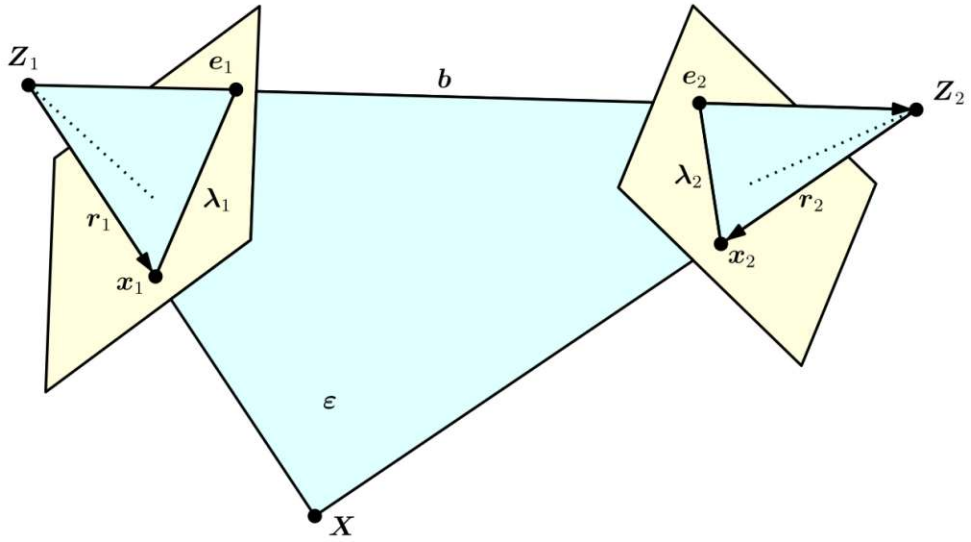


Fig. 2.12: Epipolar constraint: a point in one image will be mapped to corresponding epipolar line in the second image, forming an epipolar plane (Pfeifer et al. 2021).

Decomposition of the Essential Matrix: Once the essential matrix is estimated, it is decomposed to obtain the relative rotation and translation between the two camera poses. The essential matrix $\mathbf{E}$ can be decomposed into $\mathbf{R}$ and $\mathbf{t}$ using Single Value Decomposition (SVD):

$$\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \tag{2.34}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\mathbf{\Sigma}$ is a diagonal matrix. From $\mathbf{U}$ and $\mathbf{V}$, the possible solutions for $\mathbf{R}$ and $\mathbf{t}$ are derived:

$$\mathbf{R} = \mathbf{U}\mathbf{W}\mathbf{V}^\top \quad \text{or} \quad \mathbf{R} = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top \tag{2.35}$$

$$\mathbf{t} = \mathbf{u}_3 \quad \text{or} \quad \mathbf{t} = -\mathbf{u}_3 \tag{2.36}$$

where $\mathbf{W}$ is a specific rotation matrix and $\mathbf{u}_3$ is the third column of $\mathbf{U}$ (Arun et al. 1987).

Disambiguating the Correct Pose: Since the decomposition of the essential matrix provides four possible solutions (two for rotation and two for translation), additional constraints are needed to determine the correct pose. One common approach is to use the positive depth constraint: the correct solution ensures that the triangulated 3D points lie in front of both cameras. This is verified by reprojecting the 3D points into both images and checking the sign of the depth values (Pfeifer et al. 2021).

2.2.3 Incremental Reconstruction

Once the initial pose of the cameras is estimated, the next step in the SfM pipeline is the incremental reconstruction of the 3D scene. This step involves incrementally adding images to the reconstruction, estimating their poses, and extending the 3D point cloud (Schonberger and Frahm 2016).

Initial Triangulation: Starting with an initial pair of images with known relative pose, 3D points are triangulated from the matched feature points. Triangulation involves finding the intersection

of the rays corresponding to the matched points in the two images. Given two camera projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$, and the corresponding points $\mathbf{x}_1$ and $\mathbf{x}_2$, the 3D point $\mathbf{X}$ can be found by minimizing the **reprojection error**:

$$\min_{\mathbf{X}_j} \sum_{i,j} \|\mathbf{x}_{ij} - \mathbf{P}_i \mathbf{X}_j\|^2 , \tag{2.37}$$

where $\mathbf{x}_{ij}$ is the observed image point, $\mathbf{P}_i$ is the projection matrix for the i-th camera, and $\mathbf{X}_j$ is the j-th 3D point. The reprojection error is the difference between the observed and projected points in the images, and minimizing this error improves the accuracy of the 3D point locations (Hartley and Zisserman 2003).

Projection Matrix: The projection matrix $\mathbf{P}_i$ is defined as:

$$\mathbf{P}_i = \mathbf{K}[\mathbf{R}_i|\mathbf{t}_i], \tag{2.38}$$

where $\mathbf{K}$ is the intrinsic camera matrix, $\mathbf{R}_i$ is the rotation matrix, and $\mathbf{t}_i$ is the translation vector for the i-th camera (Hartley and Zisserman 2003).

Adding New Images with Perspective-n-Point (PnP): New images are added to the reconstruction one at a time using the PnP algorithm. Given a set of 3D points $\{\mathbf{X}_i\}$ and their image projections $\{\mathbf{x}_i\}$ in the newly added image, the goal is to find the rotation $\mathbf{R}$ and translation $\mathbf{t}$ that minimize the reprojection error (compare to 2.37):

$$\min_{\mathbf{R}_i,\mathbf{t}_i} \sum_{i,j} \|\mathbf{x}_{ij} - \mathbf{P}_i \mathbf{X}_j\|^2 , \tag{2.39}$$

This problem is typically solved using iterative algorithms such as the Levenberg–Marquardt algorithm (LM) algorithm (Lepetit et al. 2009). For details on the LM algorithm, see chapter 2.3.1.3.

2.2.4 Bundle Adjustment

Bundle adjustment is a crucial step in SfM that simultaneously refines the 3D points and the camera parameters by minimizing the reprojection error. This optimization process ensures that the 3D model is accurate and consistent across all images. Bundle adjustment minimizes the following objective function (Triggs et al. 2000):

$$\min_{\mathbf{R}_i,\mathbf{t}_i,\mathbf{X}_j} \sum_{i,j} \|\mathbf{x}_{ij} - \mathbf{P}_i \mathbf{X}_j\|^2 , \tag{2.40}$$

The optimization adjusts the camera poses $\{\mathbf{R}_i, \mathbf{t}_i\}$ and the 3D points $\{\mathbf{X}_j\}$ to minimize the reprojection error (compare to 2.37 and 2.39) across all images. Note that 2.37, 2.39 and 2.40 show the same mathematical context, yet the adjusted parameters in order to minimize the problem differ.

2.2.5 Dense Reconstruction with Multi View Stereo (MVS)

SfM techniques recover camera poses and 3D point clouds. However, the resultant 3D points are typically sparse, representing only the distinctive image features that consistently match across multiple photographs. To advance from sparse to dense 3D reconstruction, MVS algorithms

are employed. MVS algorithms enhance 3D geometric information by utilizing multiple images, analogous to how the human visual system perceives depth by integrating information from two eyes. In the MVS context, numerous images that observe the same point can be utilized for depth estimation. Figure 2.13 demonstrates the fundamental process by which a basic MVS algorithm estimates a depth value at a single pixel (Agarwal et al. 2011).

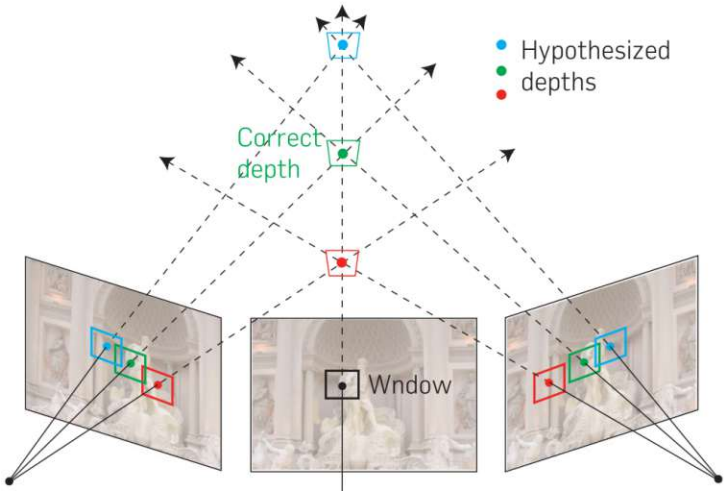


Fig. 2.13: A conventional window-based multi-view stereo (MVS) algorithm operates by hypothesizing a finite number of potential depths along the viewing ray of a given pixel and its surrounding image window. At each hypothesized depth, the image window is projected onto the corresponding locations in the other images. The algorithm evaluates the consistency of textures at these projected locations. The true depth, marked in green, corresponds to the hypothesis that yields the highest consistency score (Agarwal et al. 2011).

Point Cloud Generation: After computing the depth maps, the 3D points are extracted to form a dense point cloud (see Figure 2.14). Each depth value is back-projected into 3D space using the known camera parameters (intrinsics and extrinsics). The back-projection formula is given by:

$$\mathbf{X} = Z(\mathbf{K}^{-1}\mathbf{x}), \tag{2.41}$$

where $\mathbf{X}$ is the 3D point, Z is the depth value, $\mathbf{K}$ is the camera intrinsic matrix, and $\mathbf{x}$ is the pixel coordinate in homogeneous form.

Fig. 2.14: From left to right, sample input image, structure from motion reconstructions, and multiview stereo reconstructions (Agarwal et al. 2011).

2.3 Terrestrial Laser Scanning (TLS)

Terrestrial Laser Scanning (TLS), also known as ground-based Light Detection And Ranging (LiDAR), is a method used to capture high-resolution 3D data of objects, terrains, and infrastructure. The technology has gained widespread use due to its ability to generate detailed and accurate spatial information quickly. TLS has applications in a variety of fields, including surveying, construction, heritage conservation, forestry, and geology.

TLS operates on the principle of laser ranging, where a laser light pulse is emitted from the scanner towards a target object. The time taken for the laser to return to the scanner after reflecting off the object is measured, and this time-of-flight (TOF) is used to calculate the distance between the scanner and the object while also registering azimuth and elevation. The scanner typically rotates horizontally and vertically to capture a close to full 360-degree view of the surroundings, generating a dense point cloud that represents the 3D structure of the environment (Kraus 2004).

Components of a TLS System: a typical TLS system consists of several key components:

- **Laser Source:** The laser source emits a coherent light pulse, usually in the infrared spectrum, towards the target. The wavelength and power of the laser determine the scanning range and accuracy.
- **Receiver:** The receiver captures the reflected laser signal, and the time-of-flight is measured to determine the distance.
- **Rotating Mirror/Scanner:** This component allows the laser beam to be directed across the scene in both horizontal and vertical planes, enabling the acquisition of a comprehensive high-resolution 3D point cloud.
- **Control Unit:** The control unit processes the data collected by the scanner and manages the scanning process.
- **Data Storage:** The collected data is stored in the form of a 3D point cloud, which can be processed and analysed later.

Data Processing

The raw data obtained from a TLS system typically consists of a large set of 3D points, each with x, y, and z coordinates, collectively known as a point cloud. Additional point fields for color or intensity information are common. Color is often assigned to 3D points by capturing additional digital photographs with a camera integrated into or mounted on the laser scanner system. This process involves mapping the pixel colors from the photographs to the corresponding points in the 3D scan (Abdelhafiz 2013). A survey might consist of several *scan positions*, each representing the recording of an independent point cloud of the same object. The consequential overlap of those *scans* being conducted from varying perspectives is a prerequisite to align and merge consecutive clouds to a consistent data set, and to eliminate shadow zones (Pfeifer 2018).

2.3.1 Registration and Georeferencing in Terrestrial Laser Scanning

Registration and georeferencing are critical steps in processing TLS data. Registration involves aligning multiple point clouds from different scans into a single coordinate system, while georeferencing refers to placing this registered point cloud into a global coordinate system. This

section focuses on two key methods: the Iterative Closest Point (ICP) algorithm and specifically Generalized ICP (GICP) for registration and georeferencing via targets.

2.3.1.1 Iterative Closest Point (ICP) Algorithm

the ICP algorithm, developed by Besl and McKay (Besl and McKay 1992), is a widely used method for aligning 3D point clouds. The core idea behind ICP is to iteratively minimize the distance between two point clouds by finding the best rigid transformation, consisting of rotation and translation, that aligns them (see Figure 3.1).

Algorithm Workflow: The ICP algorithm operates in the following steps:

1. **Initial Alignment:** A rough initial alignment of the two point clouds is required. This can be achieved through manual alignment or using an algorithm that provides a coarse registration.
2. **Correspondence Matching:** For each point in the source point cloud, the algorithm identifies the closest point in the target point cloud.
3. **Transformation Estimation:** Using the matched pairs, the algorithm computes the best rigid transformation (rotation and translation) that minimizes the distance between corresponding points.
4. **Application of Transformation:** The estimated transformation is applied to the source point cloud.
5. **Iteration:** Steps 2 to 4 are repeated iteratively until the alignment converges, i.e., the change in the alignment error between iterations falls below a predefined threshold.

Challenges and Variations: While ICP is powerful, it has limitations. A major challenge is its reliance on a good initial alignment; without it, the algorithm may converge to a local minimum, resulting in incorrect registration. Additionally, ICP assumes that the point clouds overlap significantly and that one point cloud is a subset of the other, which may not always be the case, leading to false correspondences and poor alignment. To address these issues, several variations of ICP have been developed. The most relevant adaptions for the scope of this thesis are explained in the following chapters.

2.3.1.2 Point-to-Plane ICP

Unlike the standard point-to-point ICP, which minimizes the Euclidean distance between corresponding points, point-to-plane ICP minimizes the error along the surface normal at each correspondence, thereby increasing robustness to noise and outliers (Chen and Medioni 1992).

Instead of minimizing the sum of squared Euclidean distances:

$$T = \arg\min_{T} \sum_{i} ||T \cdot b_i - m_i||^2, \tag{2.42}$$

where m_i is the corresponding point in the reference cloud, the point-to-plane algorithm minimizes the projection of the residual error onto the surface normal:

$$T = \arg\min_T \sum_i w_i ||\eta_i \cdot (T \cdot b_i - m_i)||^2, \tag{2.43}$$

where η_i is the surface normal at m_i, and w_i is an optional weighting term that can be used to model uncertainty (Segal et al. 2009).

This modification improves convergence by favoring alignment along locally estimated surface structures rather than treating all dimensions equally. The approach is particularly effective when dealing with structured environments, such as indoor mapping and robotic perception, where planar surfaces are common.

2.3.1.3 Generalized Iterative Closest Point (GICP)

Generalized Iterative Closest Point (GICP) extends the traditional Iterative Closest Point (ICP) algorithm by incorporating probabilistic models for point cloud alignment (Segal et al. 2009). Unlike standard ICP, which minimizes the Euclidean distance between corresponding points, GICP refines the optimization by attaching a probabilistic model to the transformation step while maintaining efficiency.

Formulation of GICP: Given two point clouds, $A = \{a_i\}_{i=1}^N$ and $B = \{b_i\}_{i=1}^N$, GICP assumes that each measured point a_i and b_i is generated from an underlying true set $\hat{A}$ and $\hat{B}$ according to Gaussian noise:

$$a_i \sim \mathcal{N}(\hat{a}_i, C_i^A), \quad b_i \sim \mathcal{N}(\hat{b}_i, C_i^B), \tag{2.44}$$

where C_i^A and C_i^B are covariance matrices describing the uncertainty of each point. Assuming perfect correspondences and an ideal transformation T^*, we obtain:

$$\hat{b}_i = T^* \hat{a}_i. \tag{2.45}$$

The alignment is then performed by solving for the transformation T that maximizes the likelihood of the observed correspondences, which leads to minimizing the Mahalanobis distance:

$$T = \arg\min_T \sum_i d_i^{(T)^\top} (C_i^B + T C_i^A T^\top)^{-1} d_i^{(T)}, \tag{2.46}$$

where $d_i^{(T)} = b_i - T a_i$ is the residual error.

Special Cases and Interpretation: The standard point-to-point ICP algorithm emerges as a special case of GICP when:

$$C_i^B = I, \quad C_i^A = 0, \tag{2.47}$$

which reduces the cost function to:

$$T = \arg\min_T \sum_i ||d_i^{(T)}||^2. \tag{2.48}$$

Similarly, the point-to-plane ICP variant, which minimizes distances along surface normals, is derived by setting:

$$C_i^B = P_i^{-1}, \quad C_i^A = 0, \tag{2.49}$$

where P_i is the projection onto the local plane at b_i.

Plane-to-Plane Generalization: GICP can be further extended to plane-to-plane alignment by incorporating covariance information from both scans. Given surface normals μ_i at b_i and ν_i at a_i, the covariance matrices are constructed as:

$$C_i^B = R_{\mu_i} \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} R_{\mu_i}^\top, \quad C_i^A = R_{\nu_i} \begin{bmatrix} \varepsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} R_{\nu_i}^\top, \tag{2.50}$$

where ε is a small constant representing uncertainty along the surface normal. Intuitively, this can be interpreted as knowing the positions of the sampled points along the surface normals with very high confidence, while low confidence is attributed to the location along its local plane. This model improves alignment robustness by leveraging local surface structure from both scans. Figure 2.15 demonstrates the algorithm's behavior in an extreme scenario. Here, all points along the vertical section of the green scan are erroneously associated with a single point in the red scan. Due to inconsistent surface orientations, the plane-to-plane method automatically discounts these matches.

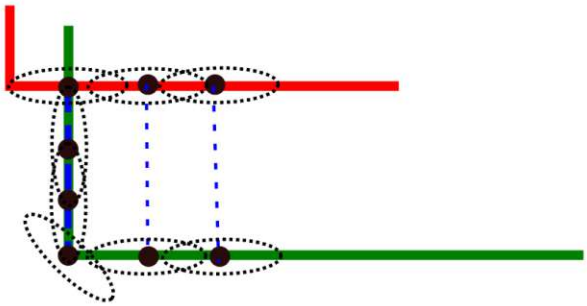


Fig. 2.15: Illustration of the plane-to-plane generalization (Segal et al. 2009).

Optimization and Implementation: The transformation T is iteratively computed using non-linear optimization techniques such as LM (Levenberg 1944, Marquardt 1963). Each iteration refines T by linearizing the error function and solving the resulting system:

$$(H + \lambda I)\Delta x = -b, \tag{2.51}$$

where H is the system Hessian, representing the second order derivative of the error function, b is the gradient, and λ is the damping factor in the LM framework. The final transformation is obtained as:

$$T^* = \hat{T} + \Delta x^*. \tag{2.52}$$

The LM algorithm is a widely used approach for solving nonlinear least squares problems. It combines the Gauß-Newton method, which approximates the Hessian as $H = J^T J$, where J is the Jacobian of the error function, with gradient descent, which introduces a damping factor λ to improve convergence in ill-conditioned cases. The update step is given by:

$$(J^T J + \lambda I)\Delta x = -J^T r, \tag{2.53}$$

where r represents the residual vector. The damping term λ is adaptively adjusted:

- If the error decreases, λ is reduced, making the update more Gauß-Newton-like for faster convergence.
- If the error increases, λ is increased, shifting towards gradient descent, ensuring stability.

This balance allows LM to handle highly nonlinear problems efficiently. GICP leverages LM optimization to refine transformations, ensuring robust and accurate point cloud alignment. The method has been successfully applied in robotic perception and SLAM (Segal et al. 2009).

2.3.1.4 Optimization of the pose graph

In this thesis, the approach specified by Kümmerle et al. 2011 is used to optimize the pose graph in order to minimize cumulative alignment errors. This is a necessary step to combine a large number of scans required for surveys on sizable objects, such as buildings. The mathematical problem can be formulated as minimizing a nonlinear least-squares objective function:

$$F(\mathbf{x}) = \sum_{(i,j)\in\mathcal{E}} \mathbf{e}(x_i, x_j, z_{ij})^T \Omega_{ij} \mathbf{e}(x_i, x_j, z_{ij}), \tag{2.54}$$

where:

- $\mathbf{e}_{ij}(x_i, x_j)$ is a vector error function that measures how well the parameter blocks $\mathbf{x}_i$ and $\mathbf{x}_j$ satisfy the constraint $\mathbf{z}_{ij}$. It is 0 when $\mathbf{x}_i$ and $\mathbf{x}_j$ perfectly match the constraint,
- Ω_{ij} is the information matrix associated with the measurement uncertainty. Ω_{ij} is usually provided as the hessian of the error function of the GICP alignment (see formula 2.51). Practical consequences and problems with weighing measurements based on the respective hessians will be discussed in chapter 3.1.2.
- $\mathcal{E}$ is the set of edges representing constraints between state variables.

This problem can be effectively represented by a directed graph, where nodes correspond to parameter blocks, and edges encode constraints (see fig. 2.16).

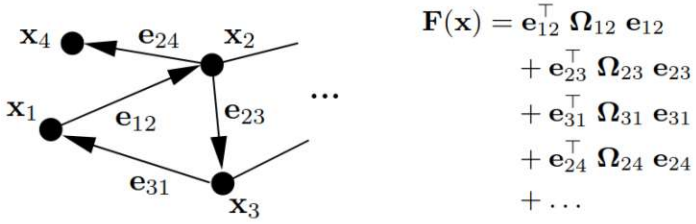


Fig. 2.16: Illustration of mathematically representing a graph based problem by constructing an objective function $F(x)$ (Kümmerle et al. 2011).

Least-Squares Optimization: If an initial estimate $\hat{x}$ is available, numerical optimization methods such as Gauss-Newton or Levenberg-Marquardt (LM) can be employed to solve:

$$x^* = \arg\min_x F(x). \tag{2.55}$$

Using a first-order Taylor expansion around the estimate $\hat{x}$:

$$e_{ij}(\hat{x} + \Delta x) \approx e_{ij} + J_{ij}\Delta x, \tag{2.56}$$

where J_{ij} is the Jacobian of $e_{ij}(x)$ at $\hat{x}$. Substituting this into the objective function gives a quadratic approximation:

$$F(\hat{x} + \Delta x) \approx c + 2b^\top \Delta x + \Delta x^\top H \Delta x. \tag{2.57}$$

Minimizing this quadratic form leads to the normal equations:

$$H\Delta x^* = -b, \tag{2.58}$$

where H is the information matrix. The solution is updated iteratively:

$$x^* = \hat{x} + \Delta x^*. \tag{2.59}$$

In the LM algorithm, a damping term λ is introduced to control convergence:

$$(H + \lambda I)\Delta x^* = -b. \tag{2.60}$$

The damping factor λ is adjusted dynamically based on error reduction in each iteration (Levenberg 1944; Marquardt 1963). If the error decreases, λ is reduced; otherwise, λ is increased, and the step is reverted.

For more details on the implementation of LM in this framework, see Kümmerle et al. 2011.

2.3.1.5 Georeferencing

The custom C++ tool, discussed in chapter 2.4, allows users to define target points for transforming the merged point cloud into a global coordinate system. Given a set of corresponding points in both the local frame and the global (reference) frame, the goal is to compute the optimal rigid transformation (rotation and translation) that minimizes the alignment error using (Arun et al. 1987).

Let the local points be $\mathbf{p}_i \in \mathbb{R}^3$ and their corresponding global coordinates be $\mathbf{q}_i \in \mathbb{R}^3$, where $i = 1, \ldots, N$. The optimal transformation consists of a rotation matrix $\mathbf{R} \in SO(3)$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$, such that:

$$\mathbf{q}_i \approx \mathbf{R}\mathbf{p}_i + \mathbf{t}, \quad \forall i.$$

To determine $\mathbf{R}$ and $\mathbf{t}$, the centroids of both point sets are first computed:

$$\bar{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{p}_i, \quad \bar{\mathbf{q}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{q}_i.$$

The point sets are then centered by defining:

$$\mathbf{p}'_i = \mathbf{p}_i - \bar{\mathbf{p}}, \quad \mathbf{q}'_i = \mathbf{q}_i - \bar{\mathbf{q}}.$$

The cross-covariance matrix is computed as:

$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{p}'_i \mathbf{q}'^\top_i.$$

Applying SVD to $\mathbf{H}$, we obtain:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

The optimal rotation is given by:

$$\mathbf{R} = \mathbf{V}\mathbf{U}^\top.$$

If $\det(\mathbf{R}) < 0$, ensuring a proper rotation requires adjusting $\mathbf{V}$ by flipping its last column:

$$\mathbf{V}' = [\mathbf{v}_1, \mathbf{v}_2, -\mathbf{v}_3],$$

$$\mathbf{R} = \mathbf{V}'\mathbf{U}^\top.$$

Finally, the optimal translation is determined as:

$$\mathbf{t} = \bar{\mathbf{q}} - \mathbf{R}\bar{\mathbf{p}}.$$

This method ensures the least-squares optimal alignment of the point cloud in the presence of noise (Arun et al. 1987).

2.4 Tool for visualizing, optimizing and georeferencing of TLS data

In order to facilitate the processing and analysis of large volumes of TLS data without reliance on costly commercial software[1], a specialized C++ tool has been developed, applying several robust open-source libraries. Because other open source software solutions like *Cloud Compare* provide a broad spectrum of point cloud processing capabilities but rather lack in performance and stability for large datasets, this tool was developed with the goal of avoiding boilerplate[2] and overhead code by using specialized lightweight libraries. The following chapter provides an overview of the tool's functionality and the underlying open-source libraries integrated into its pipeline.

2.4.1 Overview of the Tool

The tool supports the efficient handling of large TLS datasets by incorporating the following core functionalities:

- **Point Cloud Management:** Reading and processing point clouds from E57 file format.
- **Alignment:** Aligning point clouds using Generalized Iterative Closest Point (GICP).
- **Pose Graph Optimization:** Optimizing the pose graph by implementing the least squared method.
- **Global Georeferencing:** Allowing users to pick target points and georeference the merged cloud to a global coordinate system.
- **Visualization:** Real-time rendering of point clouds, scan positions, and connections.

[1] Both commonly used commercial IBM tools *AgiSoft MetaShape* and *Reality Capture* where at the onset of this work priced at approximately 3500 € for the professional license, both permitting the merging of IBM and TLS point clouds. In the meantime, *Reality Capture* has been made available free of charge by *EPIC GAMES*.

[2] Refers to code that is repeated in multiple places with little to no variation. When using languages that are considered verbose, the programmer must write a lot of boilerplate code to accomplish only minor functionality (Lämmel and Jones 2003).

2.4.2 Libraries and Dependencies

The following libraries form the backbone of the tool:

Pangolin: *Pangolin* is a versatile and lightweight library developed by Steven Lovegrove and Richard Newcombe. It is designed to streamline the development of 3D, numerical, and video-based applications, particularly in the field of Computer Vision. Its primary goal is to minimize platform-specific boilerplate while providing a simple and flexible interface for data visualization.

Pangolin simplifies tasks such as window management and video handling through intuitive interfaces and factory methods. Additionally, it offers a range of interactive debugging tools, including 3D manipulation, plotting utilities, adjustable variables, and a drop-down Quake-style console for Python scripting and real-time parameter tuning (Lovegrove and Newcombe 2024).

libE57: *libE57Format* is a C++ library providing read and write support for the ASTM-standard E57 file format. 3D point cloud data are stored in binary format including fields for color and intensity as well as extensive header information for individual scans. Additionally, 2D images (taken by a 3D imaging system) can be stored (Maloney 2024).

small gicp: *small gicp* is a header-only C++ library offering efficient and parallelized algorithms for precise point cloud registration, including ICP, Point-to-Plane ICP, GICP, VGICP, and more. It represents a refined and optimized evolution of its predecessor, *fast gicp*, re-engineered from the ground up to enhance computational efficiency and accuracy in point cloud alignment tasks (Koide 2024).

g2o (General Graph Optimization): *g2o* is an open-source C++ framework for optimizing graph-based nonlinear error functions. It has been designed to be adaptable to wide range of problems, unusually complex problems can be defined with the addition of a few new lines of code (Kümmerle et al. 2011).

JSON for Modern C++ : *JSON for Modern C++* handles JSON input/output for saving and reading pose information and configuration data (Lohmann 2023).

2.4.3 Core Functionalities and Pipeline

2.4.3.1 Point Cloud Import and Subsampling

The newly developed tool reads point clouds from E57 files, leveraging `libE57` for efficient parsing. To handle large datasets, a subsampling routine is implemented, reducing the number of points while retaining geometric fidelity. Customizable selections of scans can be read and visualized. The functionality of the file reading can be viewed in the Appendix 7.1.1. Random subsampling has been implemented, with a standard point cloud size of 500,000 points (see Appendix 7.1.1.1). This method has been chosen in order to avoid spatial queries during the subsampling process that potentially decrease processing speed.

2.4.3.2 Visualization and Interaction

The tool features a real-time visualization interface built with `Pangolin`, allowing users to view point clouds, connections, and target points. Controls enable point picking, rendering adjustments, and transformation previews. The current main interface is shown in figure 2.17.

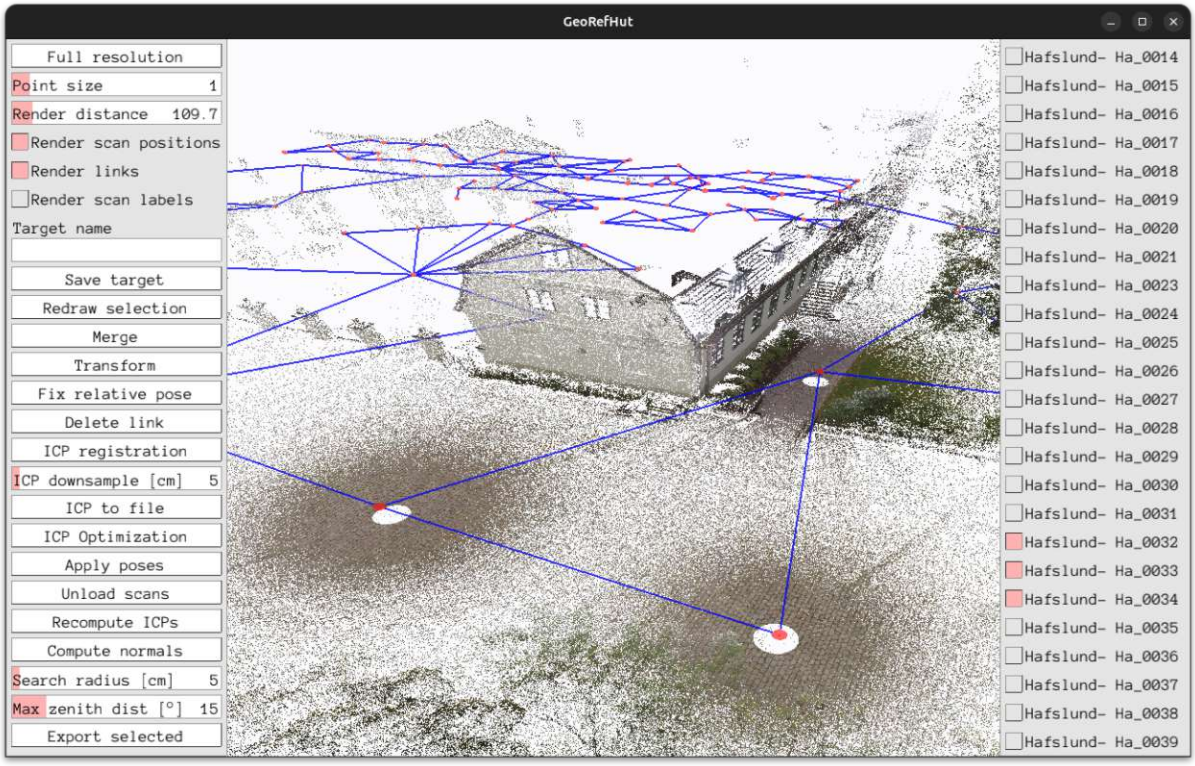


Fig. 2.17: Interface of custom C++ tool, after selecting E57 files required for processing.

Metadata is monitored by displaying verbose information about various processing steps in a terminal window.

Visualization Pipeline

- A display is initialized using `Pangolin`, rendering the loaded scans and their respective connections.
- Users can adjust rendering settings, including point size, distance thresholds, and display of scan positions or connections.
- Custom handlers allow for point picking and the creation of georeferencing targets.

2.4.3.3 Alignment via GICP

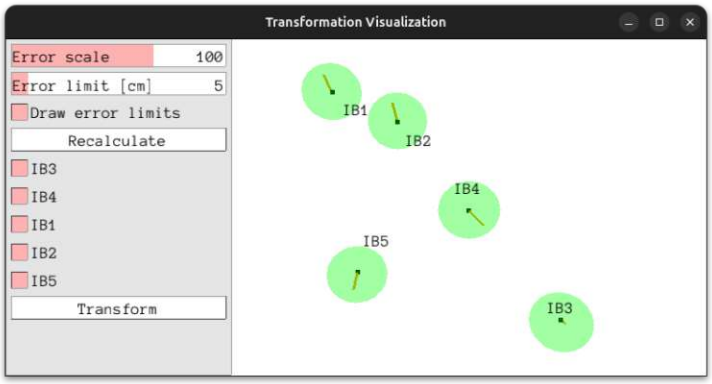
Pairwise alignment between scans is performed using the `small gicp` library, which allows for downsampling and iterative optimization of correspondences (see Chapter 2.3.1.3). The tool organizes these transformations into a pose graph by saving GICP results to a JSON file. The code used for this step is specified in Appendix 7.1.2.

2.4.3.4 Optimization of the pose graph

The established pose graph can be optimized by approaching the mathematical problem as specified by Kümmerle et al. 2011 (see Chapter 2.3.1.4). The tool saves the resulting parameters such as optimized poses and covariances to a JSON file, enabling further analysis and visualization. The corresponding code segment for this step can be found in Appendix 7.1.3.

2.4.3.5 Georeferencing

By selecting corresponding points in the local TLS frame using the main interface (figure 2.17) and providing ASCII information of the geopositioning of the targets, the computation of the transformation parameters is executed as specified in Chapter 2.3.1.5. An interface is provided to streamline the selection of included targets (figure 2.18). The implementation of this step is described in Appendix 7.1.4.



(a)

```
local centroid: -5.82220000
 1.74260000
-2.02100000

global centroid:  621750.10620000
6572504.36580000
    125.02400000

Transformation Matrix:
     -0.61923096      -0.78515624       0.00909392  621747.88750557
      0.78504100      -0.61929652      -0.01350738 6572509.98835341
      0.01623724      -0.00122509       0.99986742     127.14140332
      0.00000000       0.00000000       0.00000000       1.00000000
IB3     |diff: 0.014 -0.010 -0.015 [m] | abs. diff:  0.023 [m]
IB4     |diff: 0.019 -0.026  0.031 [m] | abs. diff:  0.045 [m]
IB1     |diff:-0.018  0.036 -0.013 [m] | abs. diff:  0.042 [m]
IB2     |diff:-0.010  0.032 -0.007 [m] | abs. diff:  0.035 [m]
IB5     |diff:-0.006 -0.032  0.003 [m] | abs. diff:  0.033 [m]
```

(b)

Fig. 2.18: Interface of the custom C++ tool for computing transformation parameters (a) and complementary terminal output (b).

2.5 NUBIGON

NUBIGON[3] is a software tool designed for the visualization and analysis of large point cloud data sets, particularly in the field of reality capture. It enables the display, examination, and presentation of point cloud datasets with great efficiency, accessing in each case only those points that are necessary to fill the display screen. This capability is beneficial for the joint visualization of surface and subsurface structures, especially in archaeological contexts.

[3] https://www.nubigon.com/

2.5.1 Octree Generation

Upon initial loading, *NUBIGON* converts point cloud data into an octree structure. This hierarchical data format optimizes rendering performance and visual quality. The octree generation process occurs only once at data import, allowing the software to manage very large datasets consisting of many billions of 3D points effectively.

2.5.2 X-Ray Mode

NUBIGON's X-ray mode allows users to visualize hidden structures and details within point clouds by automating transparency in respect to the camera position. In the context of this thesis, it was utilized to simultaneously visualize surface and subsurface features in order to characterize archaeological sites.

2.6 Data

2.6.1 TLS data

The TLS dataset from Hafslund Manor was acquired using the BLK360 G1 stationary terrestrial laser scanner (see figure 2.19), developed and manufactured by Leica Geosystems[4]. The technical details of the scanner are summarized in table 2.1.

Fig. 2.19: BLK360 laser scanner positioned on table in Hafslund. Photo: I. Trinks.

Using the Cyclone Field app, provided by Leica Geosystems, the point clouds from individual scans can be coarsely aligned through the application of an ICP algorithm. The aligned point clouds can be exported to an E57 file and visualized in the custom C++ tool (fig. 2.21). The coarse cloud-to-cloud registrations result in misalignments of structural features, with deviations reaching magnitudes of up to several decimeters (fig. 2.22).

[4] https://leica-geosystems.com/products/laser-scanners/scanners/blk360

Fig. 2.20: Panoramic image captured by the BLK360's internal camera.

Terrestrial laser scanners (TLS) equipped with in-built RGB cameras, such as the BLK360, capture both range measurements and spherical panoramic images. The laser scanner generates a point cloud with 3D coordinates, while the camera provides color information (fig. 2.20). Once registered, the color information from the panoramic images is mapped onto the point cloud. This is achieved by intersecting the line connecting each point in the point cloud with the scanner's position and the corresponding pixel in the panoramic image. The color of the intersecting pixel is then assigned to the point (Abdelhafiz 2013).

Category	Specification
SCANNING	
Distance measurement system	High speed time of flight enhanced by Waveform Digitizing (WFD) technology
Laser class	Class 1 (in accordance with IEC 60825-1:2014)
Wavelength	830 nm
Field of view	360° (horizontal) / 300° (vertical)
Range	Min. 0.6 – up to 60 m at 78% albedo
Point measurement rate	Up to 360,000 pts/sec
Ranging accuracy	4 mm @ 10 m / 7 mm @ 20 m
Measurement modes	3 user-selectable resolution settings
IMAGING	
Camera system	15 Mpixel 3-camera system, 150 Mpx full dome capture, HDR, LED flash
	Calibrated spherical image, 360° x 300°
Thermal camera	FLIR technology-based longwave infrared camera
	Thermal panoramic image, 360° x 70°
PERFORMANCE	
Measurement speed	$<$ 3 min for complete fulldome scan, spherical image, thermal image
3D point accuracy	6 mm @ 10 m / 8 mm @ 20 m all at 78% albedo

Tab. 2.1: Specifications of the BLK360 G1 stationary laser scanner. The table summarizes scanning, imaging, and performance features of the device. All parameters were taken from the official BLK360 G1 specification sheet provided by Leica Geosystems. Accuracy data is presented in terms of standard deviation.

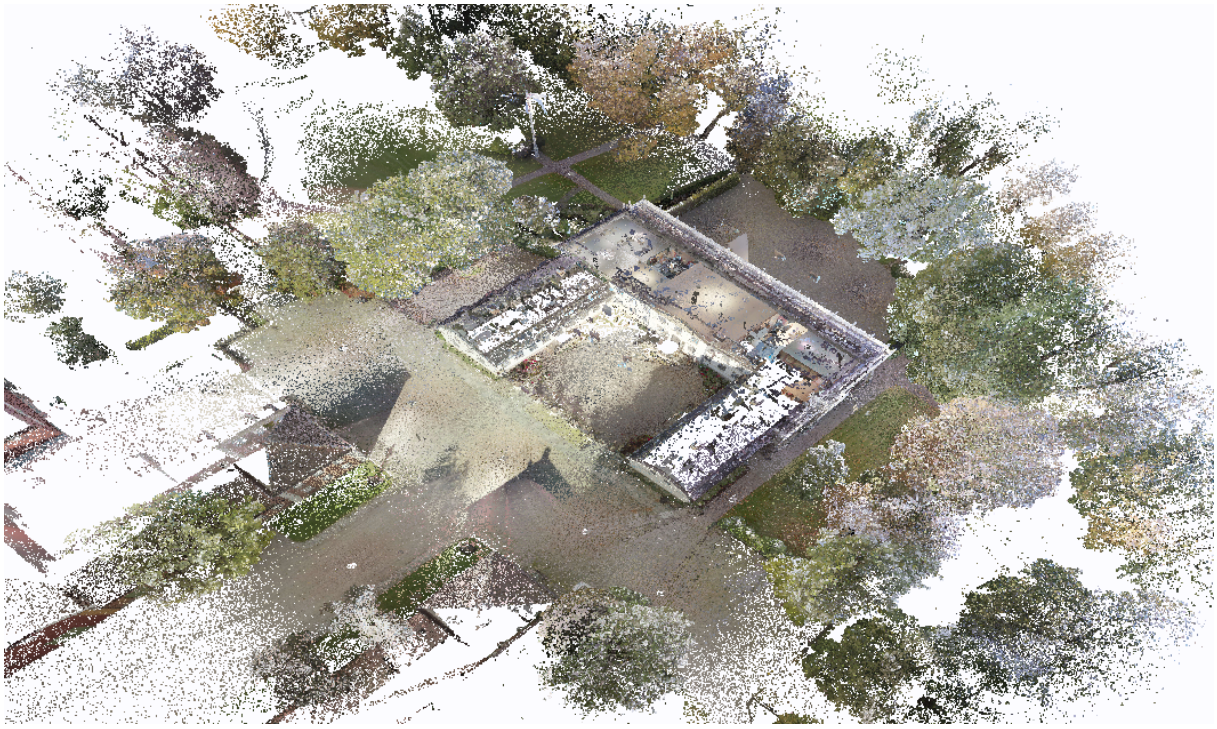


Fig. 2.21: TLS data as exported from Cyclone Field (Leica Geosystems) and visualized using the custom C++ tool.

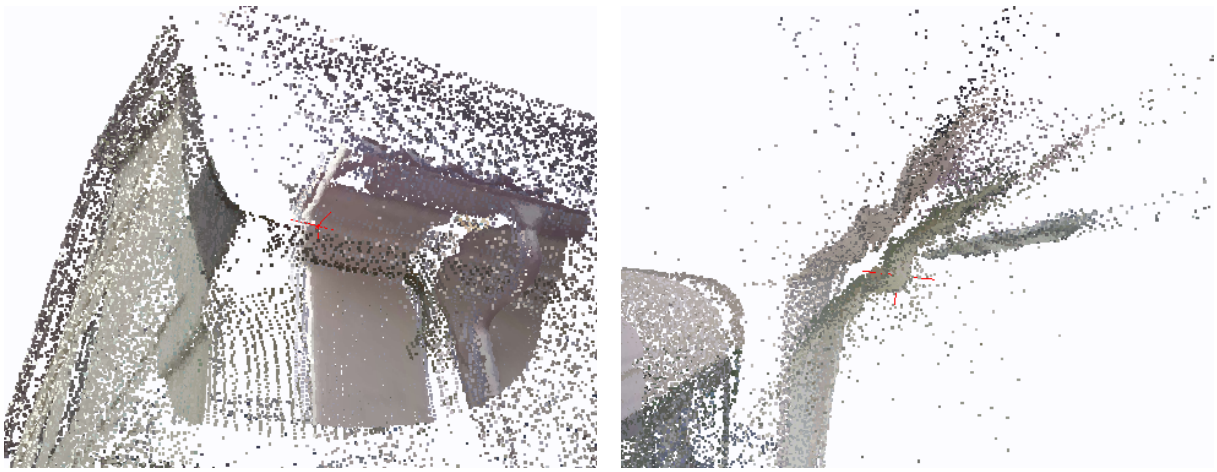


Fig. 2.22: Exemplary alignment issues in the exported raw TLS data.

2.6.2 GNSS data

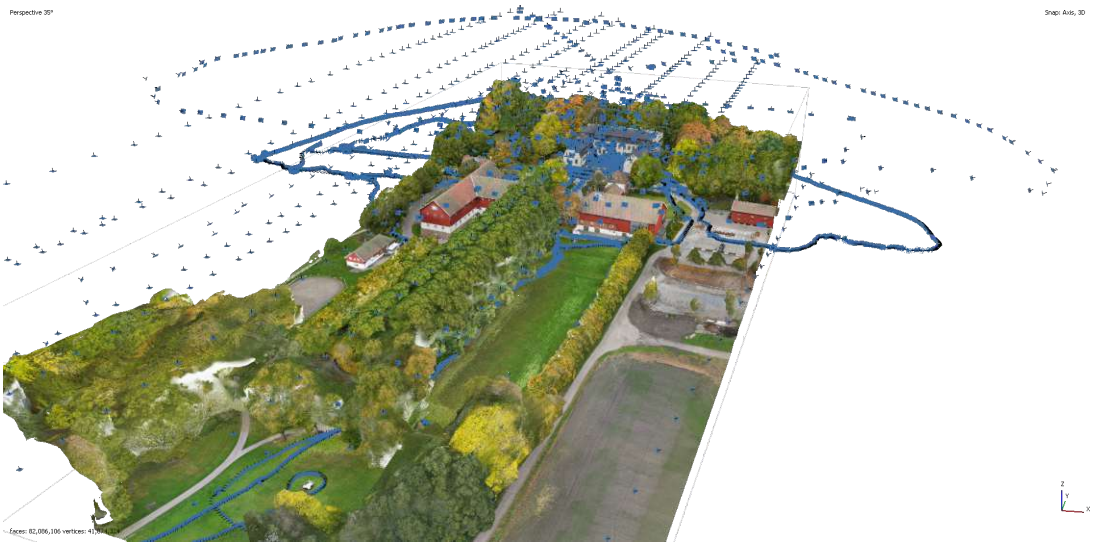
During the measuring campaign at Hafslund manor several distinct features have been used as targets for global referencing (fig. 2.23). By additionally extracting the corresponding local coordinates of these features from the TLS point cloud, a global reference can be established (see Chapter 2.3.1.5).

Fig. 2.23: GNSS measurements used as targets for transforming TLS data.

2.6.3 IBM data

Based on geopositioned RGB images taken from an sub-249 g heavy airborne *DJI*[5] *Mini 3 Pro* drone complemented by terrestrial imaging using a handheld *Sony ZV-1* photo camera, a globally referenced point cloud of the manor and its surroundings was computed using *Agisoft Metashape professional*[6] (fig. 2.25a). The theoretical approach to acquire 3D information from images has been specified in Chapter 2.2. Figure 2.24 illustrates the estimated camera poses computed by Metashape, overlaid onto the resulting surface model. The visualization also includes the sparse point cloud and the resulting colored dense point cloud, providing a view of the data integration process (see Chapter 2.2).

[5] Da-Jiang Innovations Science and Technology Co., Ltd (DJI)
[6] https://www.agisoft.com/



(a)

(b)

(c)

Fig. 2.24: IBM surface model created in Agisoft Metashape and estimated camera poses as blue rectangles (a), sparse point cloud (b) and dense point cloud (c).

2.6.4 GPR data

The topographically corrected GPR data is provided in the 3D field format FLD. This format has been developed by Alois Hinterleitner from GeoSphere Auastria, formerly Zentralanstalt für Meteorology and Geodynamics, Archeo Prospections® and LBI ArchPro, and is used to store GPR data as a 3D data block in a single binary file. Depth-slices of gridded GPR prospection data, computed by ApRadar (ZAMG Archeo Prospections®), are stored and extended with geodetic information. The format enables a lossy compression and Run-length encoding (RLE) of no data values.

The FLD file was converted to a LAZ point cloud (Isenburg 2013) by using a python script, based on the open source project ApInsight, a Ground Penetrating Radar Data Analysis Tool (Nau 2024). The algorithm developed for this thesis involves omitting zero values received by the matrix of the FLD data format.The code for this step is specified in Appendix 7.2.

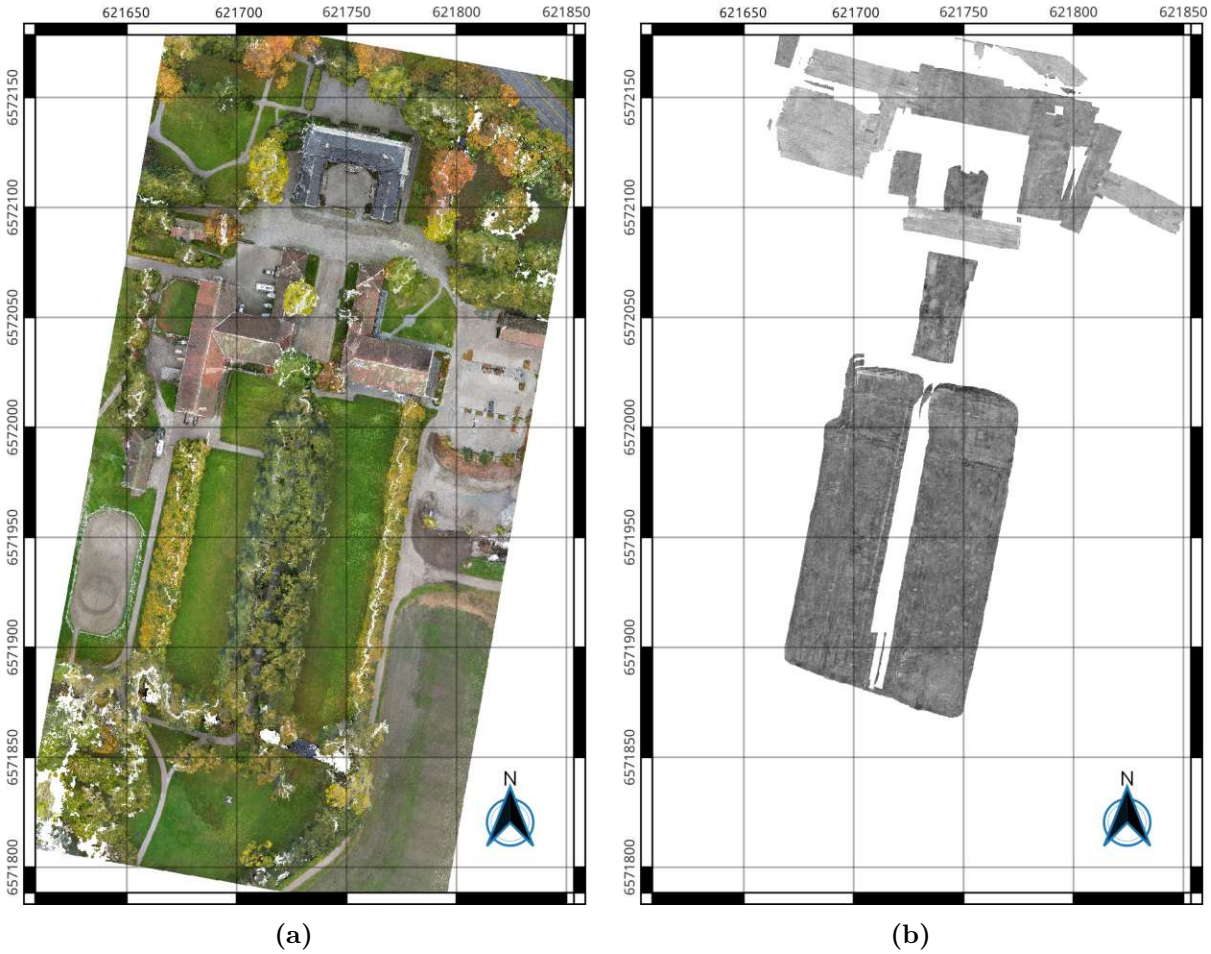


Fig. 2.25: IBM point cloud created in Agisoft Metashape (visualized in CloudCompare) (a) and GPR data (C-scan) of the study area Hafslund manor at surface level (b).

Chapter 3

Results

3.1 TLS

3.1.1 GICP alignment

Based on the .e57 files received from the Leica Cyclone Field app, coarse alignments of the scans showed discrepancies in distinct structural features (fig. 2.22 and fig. 3.1a). Without metadata on ICP linkage between scans, manual selection of potential matches was necessary. GICP was applied to the respective pairs of scans and, provided convergence was achieved (compare to Chapter 2.3.1.3), visually verified as appropriate. Figure 3.1 visualizes the effect of successful convergence to the global minimum of the error function 2.46. The LM algorithm starts off with higher values for λ, which is typical of the optimization behavior of the gradient descent method, which shifts the pose parameters in the direction of the steepest decline and therefore prioritizes stability over convergence speed in the early iterations. During the algorithm, λ is being decreased, providing the error decreases. The update becomes more Gauß-Newton-like, accelerating convergence (compare to paragraph 2.3.1.3). The initial and new errors are depicted as well in figure 3.1c, while the initial error describes the euclidean distances between point correspondences and the new error the distances of the same correspondences d_i after minimizing the error function 2.46. Note that in each iteration new correspondences are established, consequently reducing the initial error of the next iteration compared to the new error of the previous iteration. The overall rapid decrease in initial and new errors especially in the early iterations demonstrates the effectiveness of the LM-algorithm in finding the global minimum of the error function. The stabilization of error parameters, translational and rotational increments suggests convergence to an optimal solution.

GICP was applied to all possible matches and the final residuals were scaled by the number of point correspondences (see fig. 3.2 left). The magnitude of the residuals is mostly in accordance with the scanners ranging accuracy (compare to table 2.1) when also considering influences from parameters like point density on overlapping features, scanner pose distances and overall geometric feature richness of the scanned objects. The scatter plot in figure 3.2 shows a trend of increasing scaled errors with larger pose distances, while some higher error values also occur in close proximity scan alignments.

3.1.2 Pose graph optimization

By aligning all scans to their respective matches by using GICP a pose graph was established. Alignment parameters computed by GICP represent edges or constraints while the poses of the scans can be abstracted as nodes. The optimization algorithm as described in chapter 3.1.2 demands a-priori stochastic information used as weighing factors for edges. In some cases, the initial estimates of the relative poses were of higher quality than the resulting GICP alignment. For these circumstances, the initial estimates were fixed by applying artificially low uncertainty

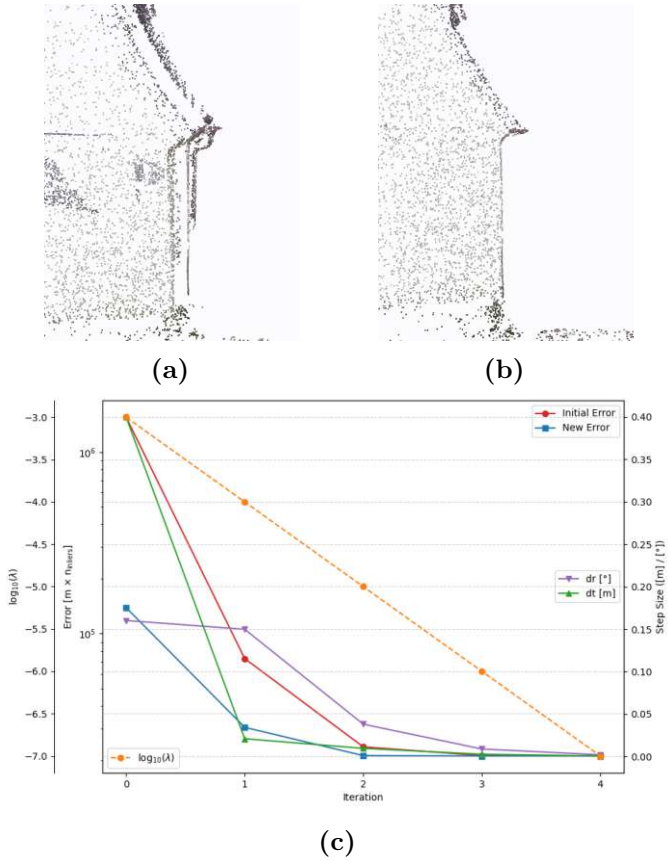


Fig. 3.1: Before (a) and after (b) applying GICP with the C++ package small gicp (Koide 2024) on two scans of the Hafslund dataset. (c) shows parameter development during the LM algorithm. The expected tendencies of decreasing values in λ, cumulative distance error of correspondences (red and blue) and step sizes in translation and rotation (green and magenta) are presented.

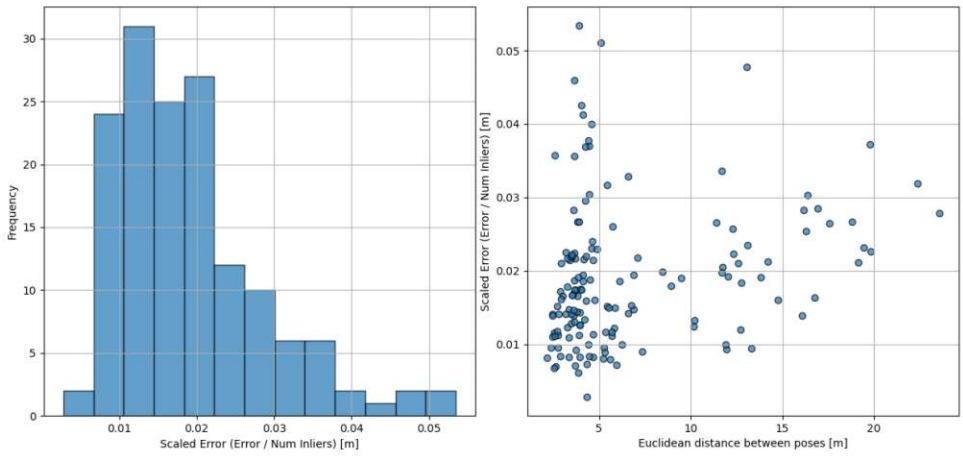


Fig. 3.2: Scaled residuals of all GICP alignments (left) and residuals against pose distances (right).

values. In order to assess values for the remaining matrices Ω_{ij} in formula 2.54, two approaches were tested.

3.1.2.1 Weighing edges by corresponding Hessians

By using the resulting Hessian matrix $\mathbf{H}_{ij}$ from formula 2.51 of the GICP algorithm, full information on the alignment behavior can be included in the pose graph optimization process. $\mathbf{H}_{ij}$ describes the curvature of the error surface with respect to the transformation parameters. It captures the second-order partial derivatives of the error function (formula 2.46), providing information about how the error changes as the parameters vary. The Hessian matrix $\mathbf{H}_{ij}$ with respect to the pose parameters $\mathbf{p}_{ij} = [x_{ij}, y_{ij}, z_{ij}, \alpha_{ij}, \beta_{ij}, \gamma_{ij}]$, is given by:

$$\mathbf{\Omega}_{ij} = \mathbf{H}_{ij} = \begin{bmatrix} \frac{\partial^2 E_{ij}}{\partial x_{ij}^2} & \frac{\partial^2 E_{ij}}{\partial x_{ij}\partial y_{ij}} & \frac{\partial^2 E_{ij}}{\partial x_{ij}\partial z_{ij}} & \frac{\partial^2 E_{ij}}{\partial x_{ij}\partial \alpha_{ij}} & \frac{\partial^2 E_{ij}}{\partial x_{ij}\partial \beta_{ij}} & \frac{\partial^2 E_{ij}}{\partial x_{ij}\partial \gamma_{ij}} \\ \frac{\partial^2 E_{ij}}{\partial y_{ij}\partial x_{ij}} & \frac{\partial^2 E_{ij}}{\partial y_{ij}^2} & \frac{\partial^2 E_{ij}}{\partial y_{ij}\partial z_{ij}} & \frac{\partial^2 E_{ij}}{\partial y_{ij}\partial \alpha_{ij}} & \frac{\partial^2 E_{ij}}{\partial y_{ij}\partial \beta_{ij}} & \frac{\partial^2 E_{ij}}{\partial y_{ij}\partial \gamma_{ij}} \\ \frac{\partial^2 E_{ij}}{\partial z_{ij}\partial x_{ij}} & \frac{\partial^2 E_{ij}}{\partial z_{ij}\partial y_{ij}} & \frac{\partial^2 E_{ij}}{\partial z_{ij}^2} & \frac{\partial^2 E_{ij}}{\partial z_{ij}\partial \alpha_{ij}} & \frac{\partial^2 E_{ij}}{\partial z_{ij}\partial \beta_{ij}} & \frac{\partial^2 E_{ij}}{\partial z_{ij}\partial \gamma_{ij}} \\ \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}\partial x_{ij}} & \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}\partial y_{ij}} & \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}\partial z_{ij}} & \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}^2} & \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}\partial \beta_{ij}} & \frac{\partial^2 E_{ij}}{\partial \alpha_{ij}\partial \gamma_{ij}} \\ \frac{\partial^2 E_{ij}}{\partial \beta_{ij}\partial x_{ij}} & \frac{\partial^2 E_{ij}}{\partial \beta_{ij}\partial y_{ij}} & \frac{\partial^2 E_{ij}}{\partial \beta_{ij}\partial z_{ij}} & \frac{\partial^2 E_{ij}}{\partial \beta_{ij}\partial \alpha_{ij}} & \frac{\partial^2 E_{ij}}{\partial \beta_{ij}^2} & \frac{\partial^2 E_{ij}}{\partial \beta_{ij}\partial \gamma_{ij}} \\ \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}\partial x_{ij}} & \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}\partial y_{ij}} & \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}\partial z_{ij}} & \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}\partial \alpha_{ij}} & \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}\partial \beta_{ij}} & \frac{\partial^2 E_{ij}}{\partial \gamma_{ij}^2} \end{bmatrix}$$

where E_{ij} is the error function for the constraint of the i-th and j-th poses being minimized, and x_{ij}, y_{ij}, z_{ij} represent the translational parameters, while $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$ represent the rotational parameters around the x-, y-, and z-axes, respectively.

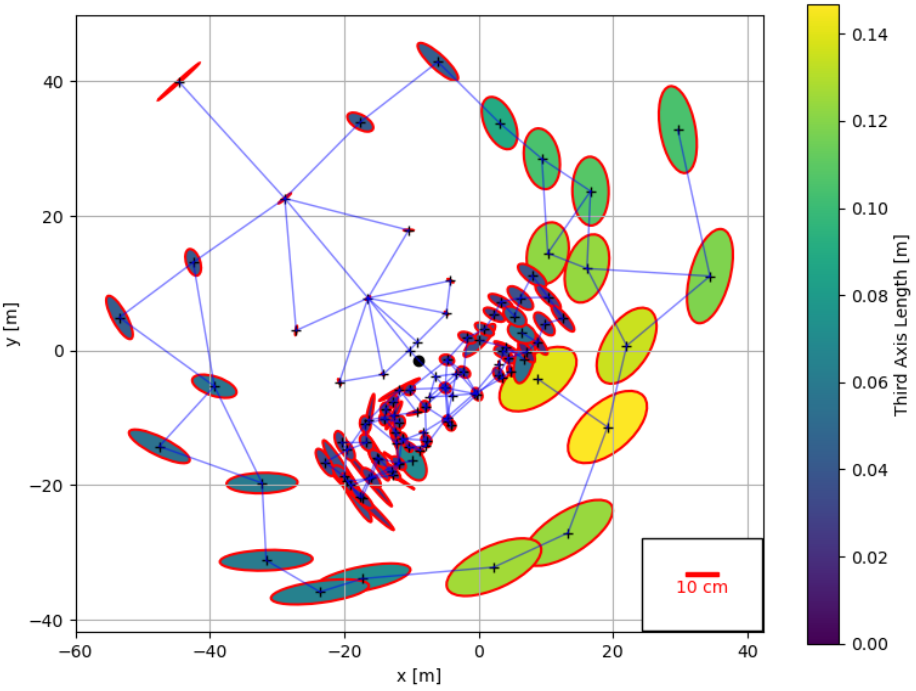


Fig. 3.3: Scaled local uncertainty ellipses and third axis color coded after optimization.

Figure 3.3 visualizes the local uncertainty ellipses (according to 95% confidence) of the poses after optimization by g2o. Note that pose graph estimation was performed in the local frame.

The rank deficiency in formula 2.54 refers to the ambiguity in global pose estimation, effectively allowing the entire graph to be translated or rotated without violating relative constraints. By fixing the first node of the graph, a reference frame is established, reducing the degrees of freedom and ensuring a unique solution. The fixed node is marked as a black dot in the center of the graph in figure 3.3. As a consequence of fixing on node and error propagation, a wide range of uncertainty ellipsoid axis length can be observed with smaller uncertainties at poses closer to the fixed node and higher uncertainties at poses with a greater edges path length between the nodes.

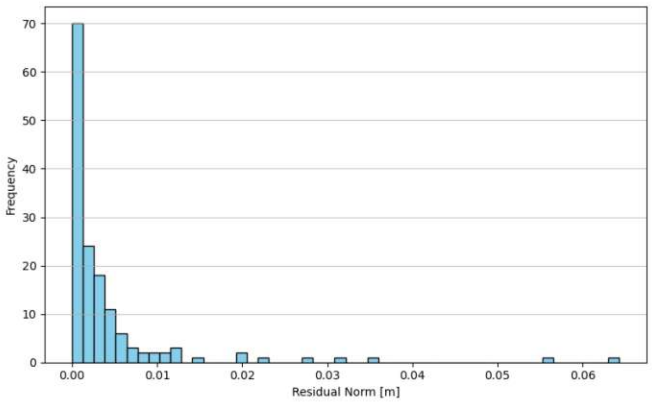


Fig. 3.4: Residuals of edges after pose graph optimization.

Figure 3.4 visualizes the residuals of the edges following optimization. These residuals represent the discrepancies between the relative positions derived from GICP and the relative positions of the optimized estimates, expressed as distances. Comparing these values to table 2.1 and figure 3.2, similar magnitudes of the a-posteriori residuals, GICP alignment errors and scanner ranging accuracy can be observed. Figure 3.4 exhibits higher frequencies in the millimeter range and a more pronounced decline towards higher magnitudes compared to figure 3.2.

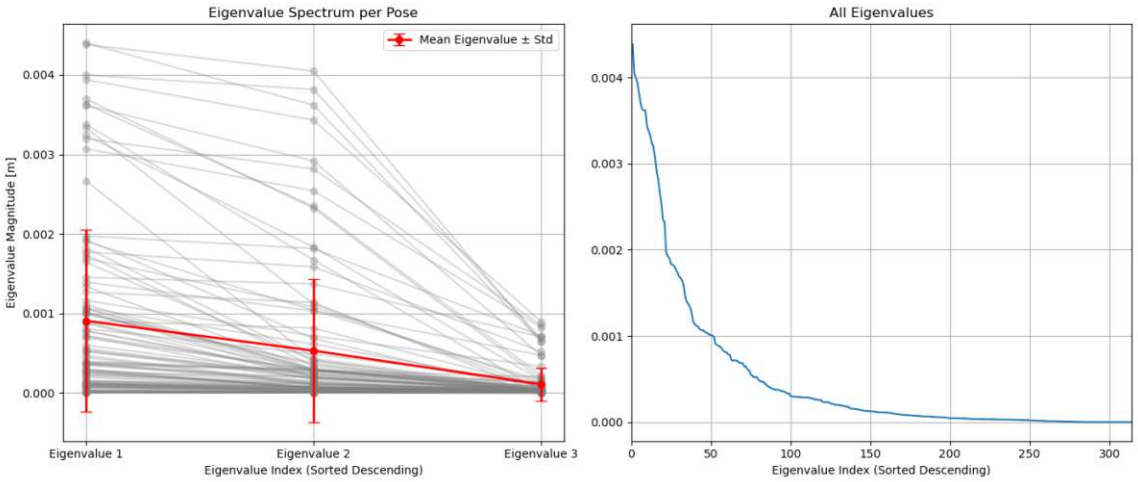


Fig. 3.5: Eigenvalue Spectrum per pose (left) and Eigenvalue Spectrum of the whole pose graph (right).

The Eigenvalues of the pose uncertainty ellipsoids are presented in figure 3.5. The spectrum on the left depicts a trend of insignificant differences in Eigenvalue magnitudes per pose with few outliers in this regard. The spectrum to the right shows a steep decline in Eigenvalue magnitude towards higher Eigenvalue indices, when sorted by value.

In the scatter plot presented in Figure 3.6, a trend of increasing axis lengths across all ellipsoid dimensions is evident. Specifically, within the range of 0.18 m to 0.05 m for the major axis, the second axis – which can be interpreted as the minor axis in the 2D context – appears to plateau for a significant number of poses.

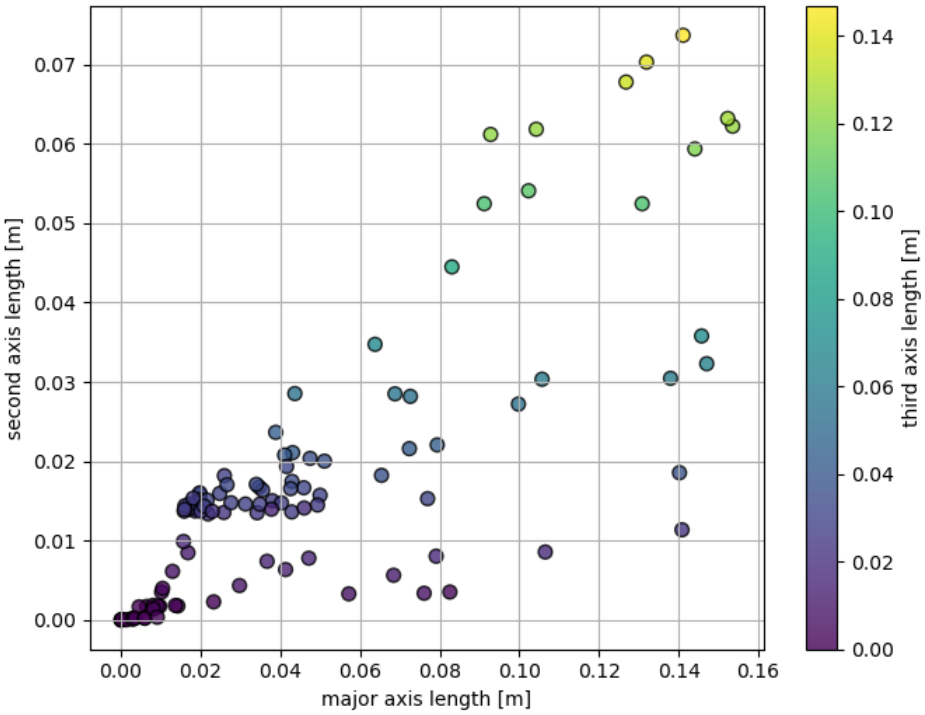


Fig. 3.6: Relation of confidence ellipses after optimization using corresponding Hessian matrices.

A negative consequence of optimizing the pose graph with the GICP-derived $\mathbf{H}_{ij}$ as a-priori stochastic information $\mathbf{\Omega}_{ij}$ (see formula 2.54) is depicted in figure 3.7. Persistent misalignment in structural features is evident across the optimized point clouds, particularly with respect to rotational inconsistencies. Furthermore, Figure 3.8 illustrates a sharp decline in χ^2 alongside a systematic reduction in λ. After approximately 25 iterations, no substantial improvement in χ^2 is noticable. The final χ^2/DoF ratio, where DoF refers to the degrees of freedom in the pose graph, is given by:

$$\frac{\chi^2}{\text{DoF}} = 260.01$$

with $\chi^2/\text{DoF} \gg 1$ being strongly indicative of overfitting (Bevington and Robinson 1969).

Fig. 3.7: Alignment issue after optimization using GICP Hessians as a-priori stochastic information. In GICP alignments of TLS point clouds congruency of overlapping features is expected. This visualization exhibits displacement of those features.

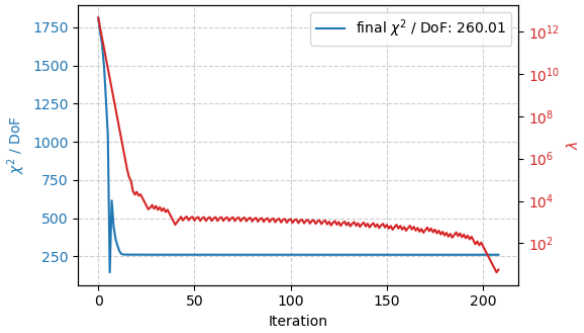


Fig. 3.8: Optimization process using the GICP-derived $\mathbf{H}_{ij}$ as a-priori stochastic information $\mathbf{\Omega}_{ij}$.

3.1.2.2 Weighing edges using adjusted stochastic model

To address the issue of overfitting, the underlying stochastic model was adjusted. The following matrix was applied to formula 2.54:

$$\mathbf{\Omega}_{ij} = \begin{bmatrix} \frac{1}{\sigma_t^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{\sigma_t^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sigma_t^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\sigma_r^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sigma_r^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\sigma_r^2} \end{bmatrix}$$

where $\sigma_t = 0.008$ (given in table 2.1) and $\sigma_r = 0.00023$, in order to apply low uncertainty to rotations which, from experience, are well estimated. By omitting a-priori correlations between rotation and translation, this approach reduces the risk of overparametrisation and overfitting, leading to a more flexible and generalizable model. The rotational component σ_r was adjusted to achieve a fitting a-priori stochastic model by $\chi^2/\text{DoF} \approx 1$. This behavior is visualized in Figure

3.9, where convergence is achieved at $\chi^2/\text{DoF} = 1.05$. Initially, there is a steep decline in χ^2 over the first five iterations, accompanied by a logarithmic decrease in λ values until iteration 13. After iteration 10, no significant change in χ^2 is observed. Towards the end of the optimization process, λ values exhibit a sharp increase, indicating that the algorithm is struggling to find parameter adjustments that further reduce the error. This, combined with the stabilization of χ^2, triggers the termination of the optimization. The lack of improvement in χ^2 suggests that the algorithm has likely reached a minimum, potentially the global minimum.

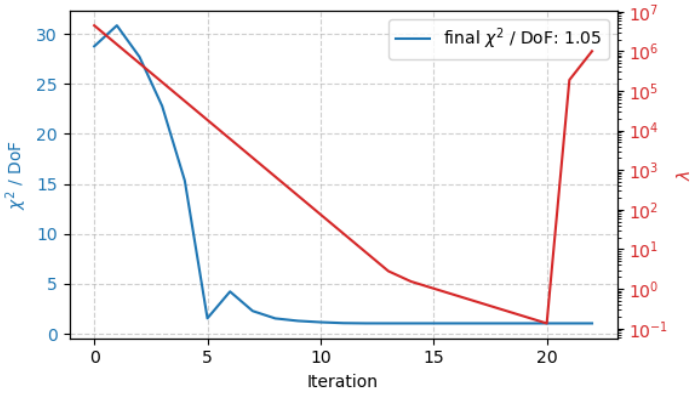


Fig. 3.9: Optimization process using the adjusted $\mathbf{\Omega}_{ij}$ as a-priori stochastic information.

Figure 3.10 illustrates the confidence ellipses at a 95% significance level following pose graph optimization using $\mathbf{\Omega}_{ij}$, where only the diagonal elements are non-zero. The ellipses exhibit reduced eccentricity compared to those in Figure 3.3. Additionally, the negative error propagation is less pronounced, as evidenced by the more uniform sizes of the ellipses.

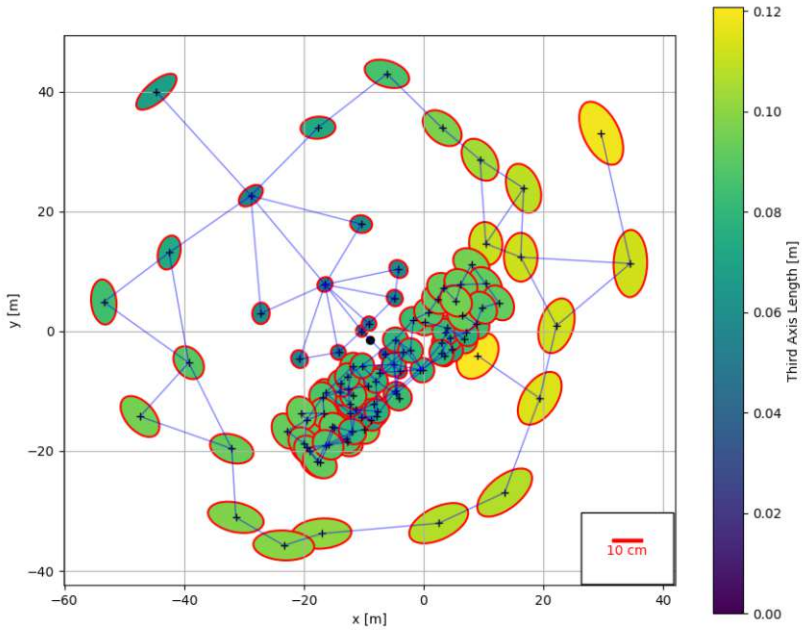


Fig. 3.10: Scaled local uncertainty ellipses and third axis color coded after optimization using $\mathbf{\Omega}_{ij}$.

Figure 3.11 illustrates the residuals of the edges following optimization, using $\mathbf{\Omega}_{ij}$ as a-priori stochastic information. The discrepancies between the GICP results and the optimization results are reduced compared to optimization using $\mathbf{H}_{ij}$ (compare to figure 3.4), as indicated by the high frequency of residuals below 1 mm. The distribution exhibits a sharp decline towards higher residuals, with the most extreme values remaining under 1 cm. Upon further comparison of these residuals with the GICP-derived errors depicted in Figure 3.2, it is evident that the post-optimization residuals exhibit a significantly lower magnitude than the measurement errors.

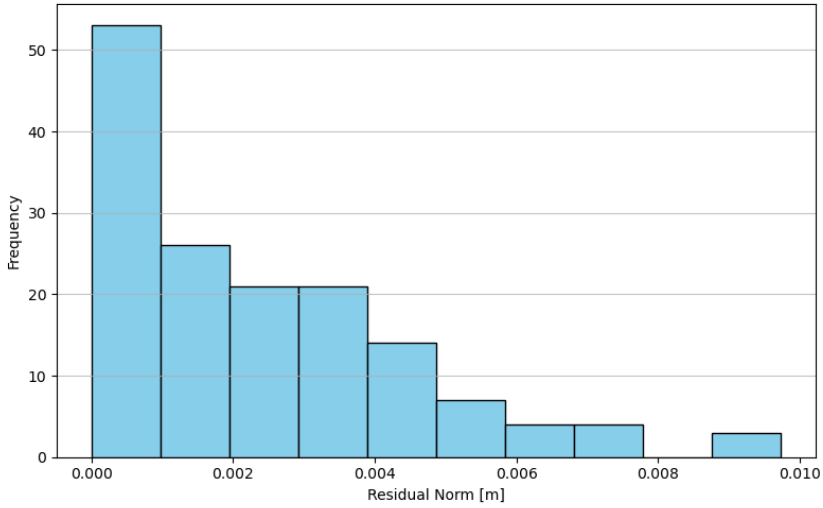


Fig. 3.11: Residuals of edges after pose graph optimization using $\mathbf{\Omega}_{ij}$.

Figure 3.12 illustrates a more homogeneous Eigenvalue spectrum compared to the spectrum obtained using $\mathbf{H}_{ij}$ as a-priori stochastic information. On the left, the eigenvalues of individual poses exhibit a somewhat proportional relationship, with a significantly lower mean and standard deviation. On the right, there is a steep decline in Eigenvalues at lower indices, followed by a more evenly distributed tail.

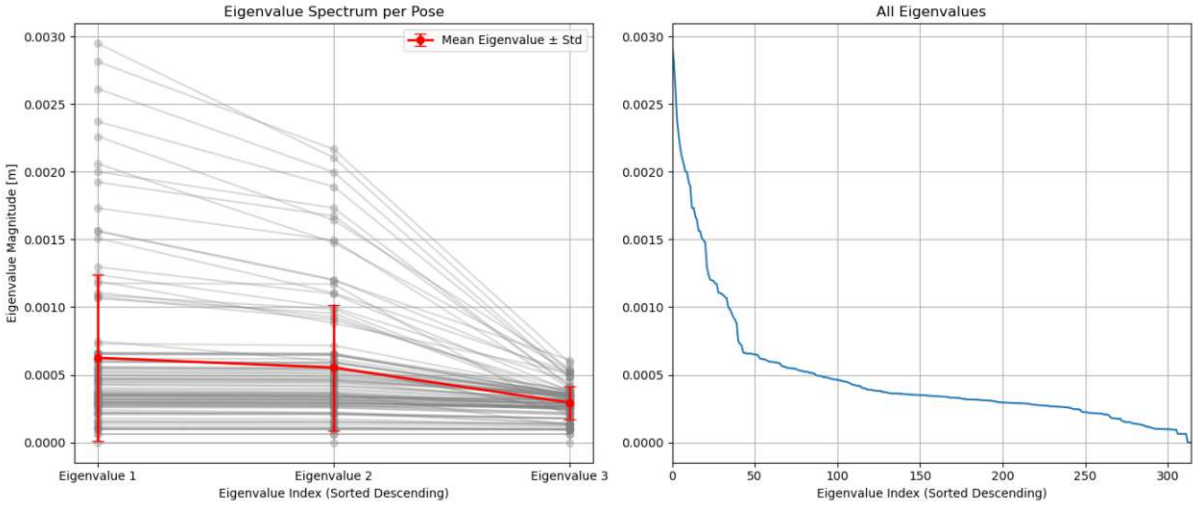


Fig. 3.12: Eigenvalue Spectrum per pose (a) and Eigenvalue Spectrum of the whole pose graph (b). Both spectra visualize the Eigenvalues in respect to the positions of the scans.

In the scatter plot depicted in Figure 3.13, the grouping of data points occurs at higher values compared to Figure 3.6. Notably, the magnitudes of the minor and major axes, both for individual poses and across the entire pose graph, appear more evenly distributed. The plateauing effect observed in Figure 3.6 is not evident in Figure 3.13. However, there is a higher visual density in the range of 0.04m to 0.06m of the major axis and 0.04m to 0.05m of the second axes. The extreme value at [0.0, 0.0, 0.0] corresponds to the fixed pose used to address global ambiguity.

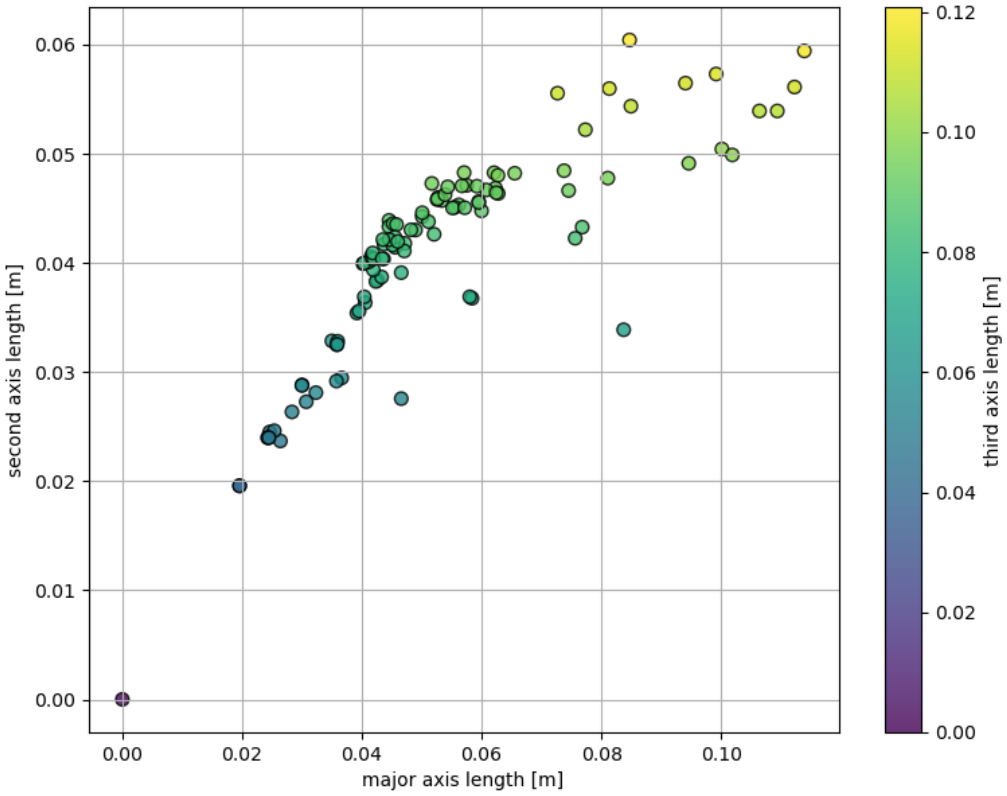


Fig. 3.13: Relation of confidence ellipses after optimization using $\mathbf{\Omega}_{ij}$.

Assessment of positioning quality through wall structure comparison To evaluate the quality of the pose graph over extended edge path distances, an analysis of the northern facade was conducted and compared to the inner wall structure. By assuming parallel surfaces on both the interior and exterior, it was possible to compare unconnected scans, thereby assessing the overall quality of the pose graph. Specifically, scans from the top floor were compared to exterior scans to the north, which are significantly distant in terms of edge path. Figure 3.14 displays the segmented point cloud, highlighting the wall structure. The color coding indicates the distances between the interior and exterior surfaces, offering a clear visualization of spatial variations. Note that in the central section, higher distances are visible due to the thicker wall structure in this area.

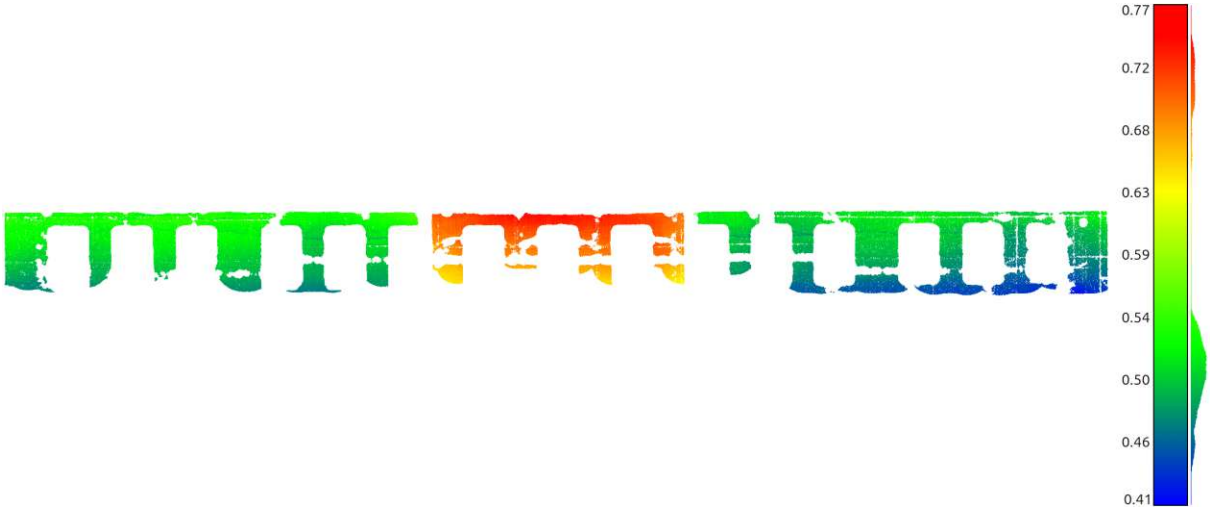


Fig. 3.14: Wall structure of the northern wall as seen from the interior. The color bar to the right refers to the cloud-to-cloud distances in relation to the exterior facade.

The variation in distances with increasing height is evident in Figure 3.15 on the left. Note that the central section mentioned above was removed for the purpose of highlighting the suspected trend. The trend indicates a difference in wall thickness of approximately 10 cm from the floor to the ceiling on the top floor. Additionally, a lateral deviation of about 5 cm is detectable across the approximately 40 m long wall front. The degree to which the walls taper is noted above the scatter plots.

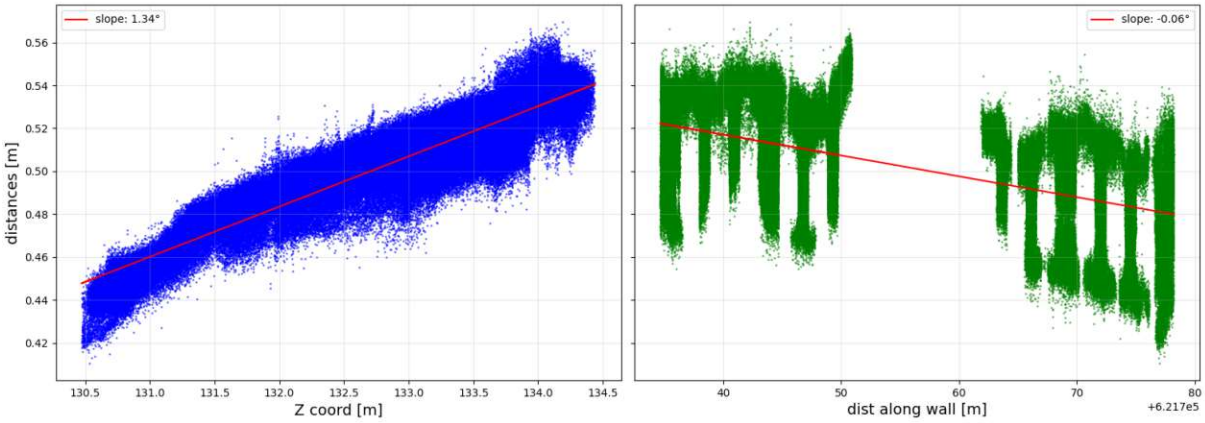


Fig. 3.15: Wall structure trend regarding its thickness.

3.1.3 Georeferencing the Terrestrial Laser Scanner (TLS) cloud

To reference the optimized TLS point cloud to the global frame, GNSS measurements were utilized. The targets were visually selected in the optimized local point cloud, corresponding to the features measured by GNSS, and the residuals were calculated as shown in table 3.1. Through an iterative process of deselecting the targets with the highest residuals, a set of four residuals, each below 5 cm, was achieved.

Point	ΔX [m]	ΔY [m]	ΔZ [m]	$\lvert\Delta\rvert$ [m]
TP002	0.015	-0.028	-0.028	0.042
TP003	-0.035	0.017	0.026	0.046
TP005	-0.012	0.007	-0.003	0.014
TP007	0.031	0.004	0.005	0.032
not selected				
TP004	-0.180	-0.222	-0.038	0.289
TP006	-0.133	-0.056	-0.569	0.587
TP008	-0.007	-0.344	-0.092	0.356
TP018	-1.010	0.412	2.065	2.336

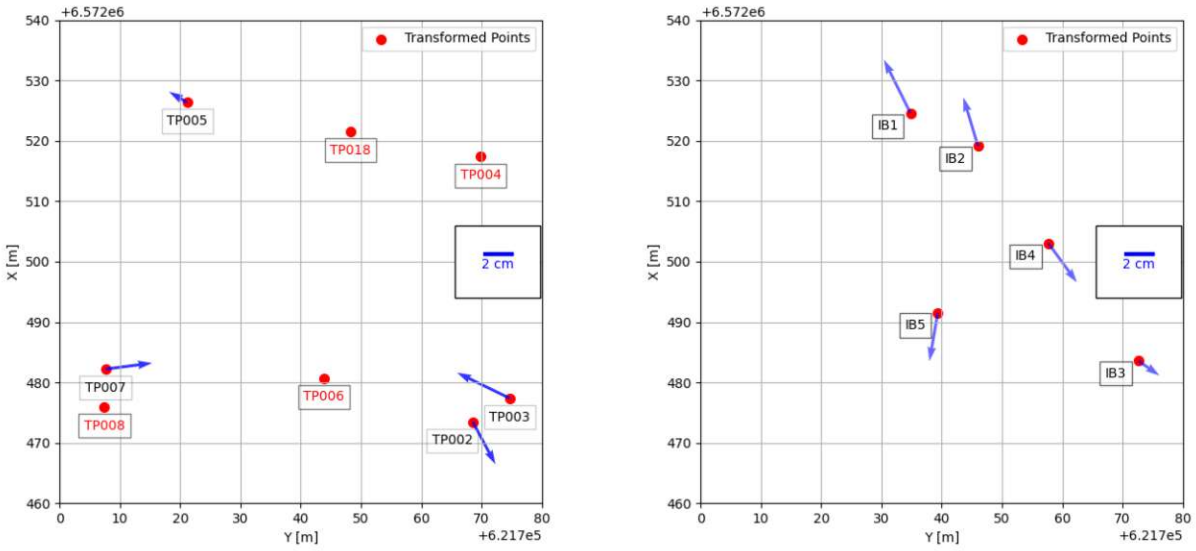
Tab. 3.1: Transformation residuals based on GNSS targets.

By utilizing the geo-located IBM point cloudfor target references, more precise transformation parameters were obtained, as demonstrated by the residuals in table 3.2. This approach allowed for the inclusion of additional targets, as the residuals for all selected targets were lower compared to table 3.1.

Point	ΔX [m]	ΔY [m]	ΔZ [m]	$\lvert\Delta\rvert$ [m]
IB1	-0.018	0.036	-0.013	0.042
IB2	-0.010	0.032	-0.007	0.035
IB3	0.014	-0.010	-0.015	0.023
IB4	0.019	-0.026	0.031	0.045
IB5	-0.006	-0.032	0.003	0.033

Tab. 3.2: Transformation residuals based on IBM targets.

Figure 3.16 visualizes the previously mentioned residuals. While the GNSS-targets better encompass the study area, the IBM-targets contribute to a more reliable transformation result as more targets could be included.



(a) **(b)**

Fig. 3.16: Transformation residuals using (a) GNSS targets and (b) IBM targets. In the left plot (a), targets not used for the computation of the transformation parameters are colored red (see residuals in table 3.1).

3.2 IBM data and TLS comparisons

As a consequence of the IBM-target-based transformation, the point clouds of the IBM approach and the transformed TLS data were analyzed. A simple cloud-to-cloud distance visualization is presented in figure 3.17.

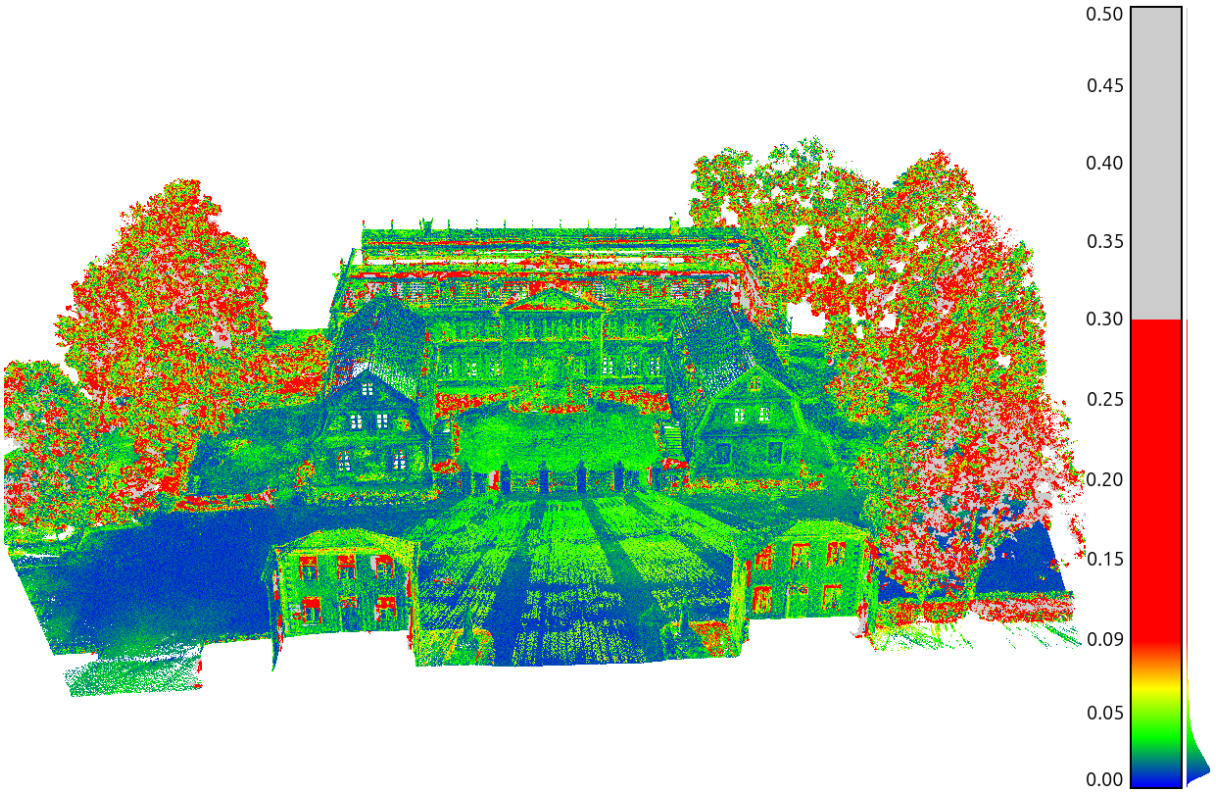
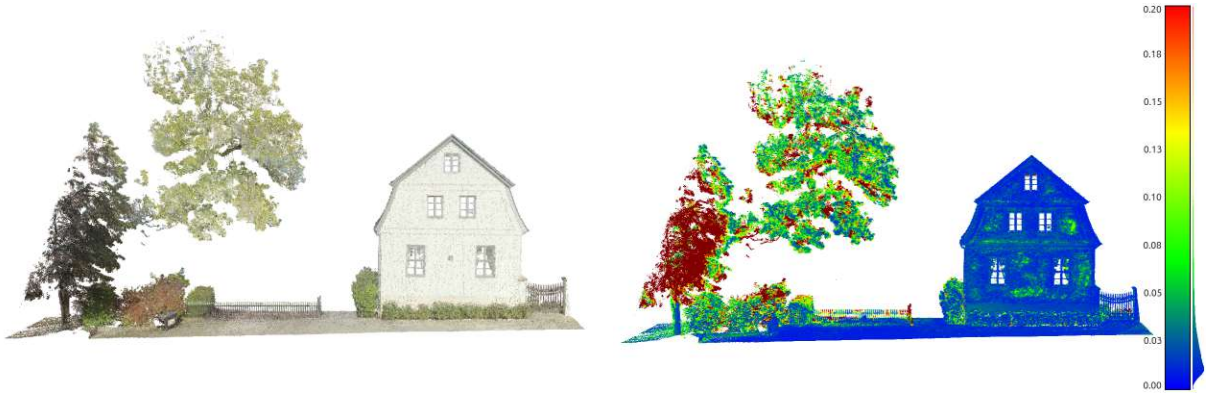


Fig. 3.17: Cloud-to-cloud distances (in meters) of the IBM point cloud, referenced against the TLS point cloud, computed in CloudCompare post-georeferencing.



(a) **(b)**

Fig. 3.18: Segmented TLS point cloud visualized with RGB information (a) and cloud-to-cloud distances of the same section (b) with distances referred to in meters. Points with values above 0.2 m are colored in dark red.

Figure 3.18 displays a section of the TLS point cloud, highlighting parts of the facade and tree structures. Significantly higher residuals are observed in vegetated areas, particularly tree crowns (see also figure 3.17). Surfaces with low textural topography exhibit residuals around 10 cm, while fragile structures such as fences also show elevated residuals.

3.3 IBM data and GPR comparisons

To compare ground features captured by IBM and GPR, ground points from both datasets were extracted. The gridded nature of the GPR data and consequently the point cloud, allowed for determination of the highest data point in each column, representing a ground point. By creating a Triangulated Irregular Network (TIN) from the ground points, a ground mesh for the GPR data was generated. Additionally, using the Cloth Simulation Filter (CSF) method by W. Zhang et al. 2016, a corresponding ground mesh for the IBM data was computed, as illustrated in figure 3.19. Vertical mesh to mesh distances were computed in CloudCompare. Figure 3.20 illustrates the resulting height residuals to the right. Discrepancies of approximately 20 cm are noticable across most areas, with some regions exhibiting significantly higher differences exceeding 50 cm. Notable patterns are evident around the small path perpendicular to the central avenue and in other areas featuring embankments.

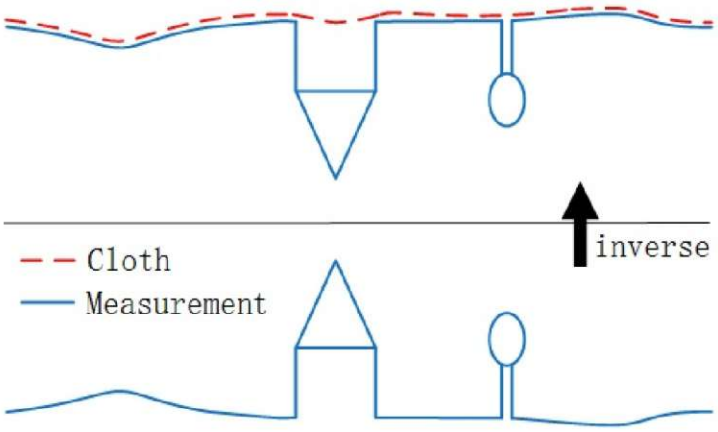


Fig. 3.19: Illustration of the CSF algorithm. The simple physical process of a cloth pulled onto the inverted surface points is simulated to extract the ground points of the IBM data (W. Zhang et al. 2016).

3.4 Visualization of merged 3D surface and subsurface data

The GPR data reveals distinct buried anomalies of strongly reflective and geometric nature, suggesting an anthropogenic origin. These anomalies exhibit clear patterns indicative of structured, human-made features rather than natural formations. Specifically, the data highlights wall-like structures arranged in a perpendicular formation, which is characteristic of architectural design. This configuration suggests the presence of building remains and related structures, such as pipes (see fig. 3.21).

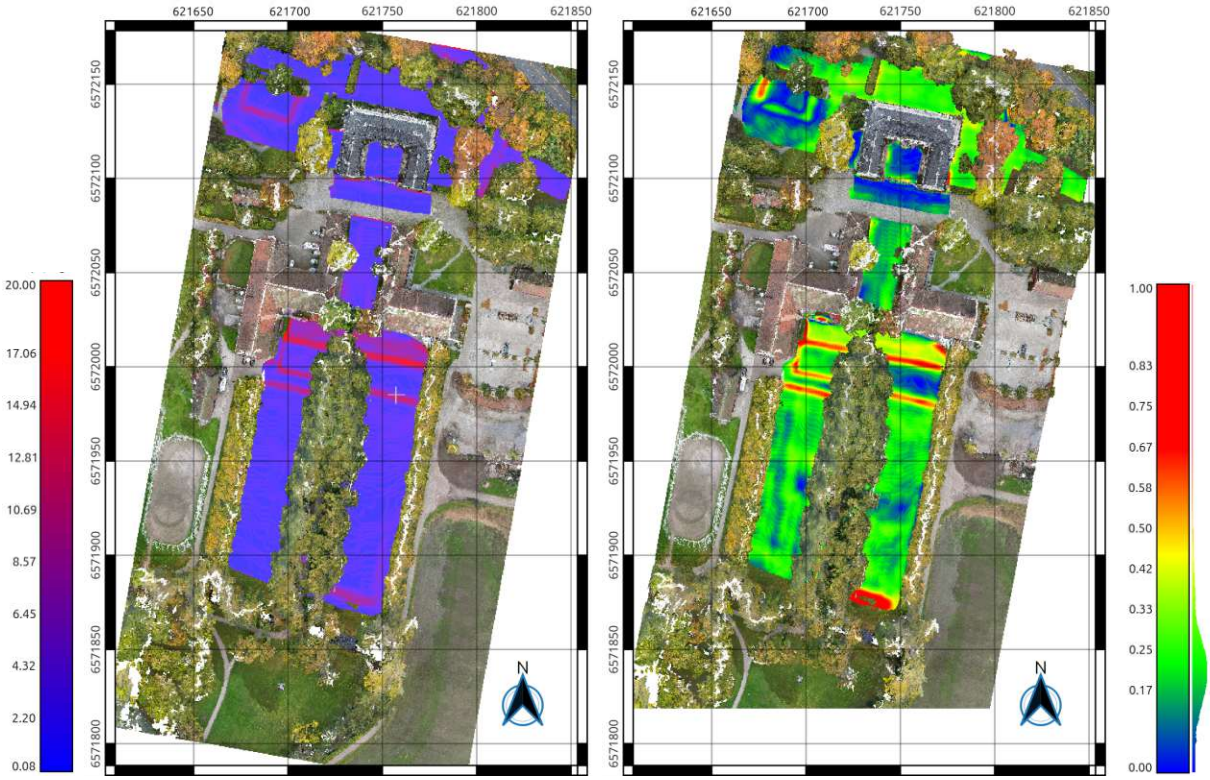


Fig. 3.20: Top view of IBM point cloud with superimposed slope map [degrees] (a) and mesh to cloud distances between IBM surface points and top GPR mesh [m] (b).

Fig. 3.21: Visualization of merged 3D GPR and IBM data in Nubigon.

Chapter 4

Discussion

4.1 TLS

4.1.1 GICP alignments

By selecting the Generalized ICP (GICP) as the alignment technique, more robust computation of alignments was achieved compared to using the conventional ICP algorithm. GICP's consideration of surface structure allows for the discarding of faulty point correspondences, resulting in more robust outcomes (Segal et al. 2009). However, coarse alignment of scans remains necessary to ensure successful convergence to a global minimum. The TLS data used in this thesis meets this requirement, as coarse manual alignment was performed using the Cyclone Field app prior to data export. Alignment errors, as illustrated in Figure 3.2, are influenced by several factors, including point density, structural characteristics in overlapping areas, scanner pose distances, and ranging accuracy. The highly featured structures within the manor grounds, such as vegetation and gravel surfaces, present challenges for accurate capture and alignment. These surfaces exhibit intricate structural compositions that are difficult to reliably and accurately match across adjacent scans. The limitations in ranging accuracy and point density of the sensor used further worsen these issues, leading to inaccuracies in the GICP alignment process.

4.1.2 Pose graph optimization

By using the Hessian matrices obtained in the GICP algorithm as a priori stochastic information for pose graph optimization, intricate details about the translational and rotational parameters and their correlations can be included for pose estimation. Intuitively, this approach assigns lower weights for alignments with less accuracy and higher weights to those with greater precision (Niemeier 2008). Distinctions in weighting within the parameter space define the quality of the GICP alignment in relation to environmental conditions. For instance, lengthy corridors contribute to higher accuracy perpendicular to the corridor axis, while accuracy parallel to the corridor axis is lower. The variation in weighting is evident in Figure 3.4, where high residuals of approximately 6 cm indicate an edge with low weight, necessitating a more significant adjustment. Meanwhile, the histogram displays the expected behavior across most other areas. Analyzing the Eigenvalue spectrum in figure 3.5, the pose graph exhibits notable characteristics. A sharp decline in Eigenvalues, when sorted by decreasing value per pose, indicates low isotropy, resulting in a confidence ellipse with higher eccentricity. Additionally, the sharp decline in the Eigenvalue spectrum to the right of figure 3.5 is significant. The more evenly distributed portion of the spectrum corresponds to interior poses, characterized by shorter pose distances, higher overlaps, and favorable structures. Conversely, the sharply inclined portion to the left corresponds to exterior poses, which exhibit opposite characteristics. This behavior is typical for unfavorable error propagation, a consequence of fixing a single pose to eliminate global ambiguity (Niemeier 2008). Confidence ellipses of poses within the interior of the manor exhibit an increase in the major axis, while the magnitudes of the second and third axes remain stable (Figure 3.6). This suggests a relationship

between the nature of interior structures and error propagation. The adjustments at edges with lower weight, combined with the over-parameterization from correlating translational and rotational parameters, result in undesired structural misalignments, as seen in Figure 3.7. This is particularly evident in rotational discrepancies. The resulting $\chi^2/\text{DoF} \gg 1$ indicates a poor model fit, suggesting an underestimation of the measurement covariances (Bevington and Robinson 1969). Consequently, the Hessian matrices appear to apply an overly optimistic weighting scheme.

Addressing the mentioned shortcomings, a simpler stochastic model was applied, with no correlation between translational and rotational parameters (see Chapter 3.1.2.2). Notably, convergence is achieved at significantly less iterations with a final $\chi^2/\text{DoF} = 1.05$. Figure 3.12 on the right shows confidence ellipses with less eccentricity and a less asymptotic path of the Eigenvalue spectrum. The residuals exhibit no significant outliers, all being below 1 cm. Error propagation is moderate because all edges are weighted equally, regardless of their relative pose distances. This approach, chosen experimentally, achieves satisfying visual congruency of structural features and a $\chi^2/\text{DoF} = 1.05$. This method is justified because alignments with greater pose distances can serve as stabilizing control elements, unlike close-proximity pose setups. This is valid when there is sufficient coverage of critical features for matching, with ranging accuracy and resolution being the only limiting factors. Consequently, the scatter plot in Figure 3.13 reveals a more uniform distribution of axis lengths throughout the pose graph.

4.1.3 Assessment of Wall Structure Discrepancies

In the assessment of wall structure discrepancies, a notable disparity was identified between the exterior facades and their corresponding interior wall structures. Assuming parallel surfaces in situ, significant rotational inconsistencies were observed between poses on the interior of the first floor and those on the exterior. This is expected due to the high edge path distances between these poses, which result in unfavorable error propagation.

4.2 IBM data and TLS comparison

The comparison of transformed TLS data and IBM data reveals no systematic discrepancies between the two point clouds. The exceptions are vegetational structures, reflective areas such as windows, and other fine structures like fences. Given that photogrammetry struggles to capture these features accurately, these discrepancies are expected. Additionally, strong variations in residuals are notable in areas with little texture, such as parts of the facade, which provide insufficient texture for feature extraction and depth estimation.

4.3 IBM data and GPR comparison

In a comparative analysis of IBM and GPR data consistency with respect to geometry, cloud-to-mesh distances have been calculated between the IBM surface points and the GPR mesh of the surface data points. The residuals are notably higher in areas characterized by embankments. A plausible explanation for this discrepancy is inaccurate topographic correction of the GPR data, which introduces inconsistencies, particularly in sloped regions such as embankments.

4.4 Visualization of merged data in Nubigon

GPR data uncovered subsurface structures and features beneath the manor grounds. Anomalies of buried walls, foundations and utilities were identified. Figure 3.21 illustrates these structures, which are situated beneath the grass covered terraces adjacent to the main building complex, and which are suspected to be part of an former orangery. The visibility and clarity of these features in the GPR data underscore the effectiveness of this non-invasive method in detecting and delineating subsurface archaeological remains. The identification of these structures contributes significantly to our understanding of the site's historical layout and potential past uses, providing valuable insights for further archaeological investigation and interpretation. The ability to render the GPR and surface point cloud data jointly for interactive 3D inspection in Nubigon permits novel insights into the data and better understanding of the spatial relations. Figures 4.1 – 4.5 exhibit additional visualizations, illustrating improved interaction with the data set.

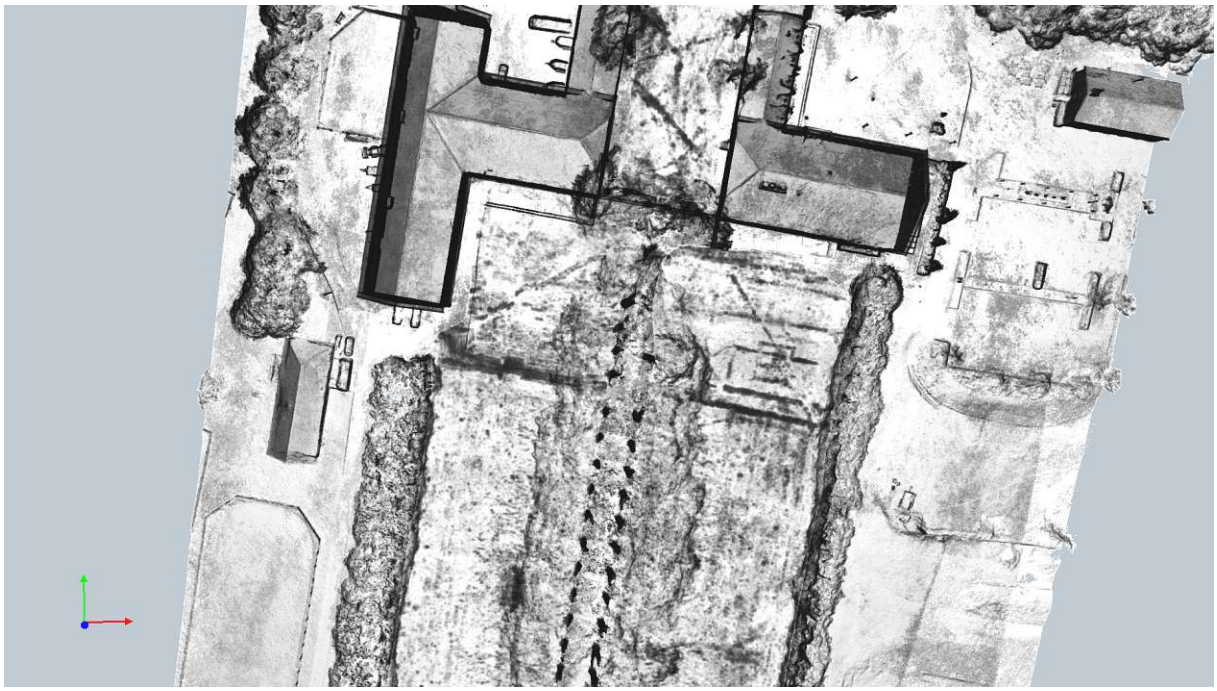
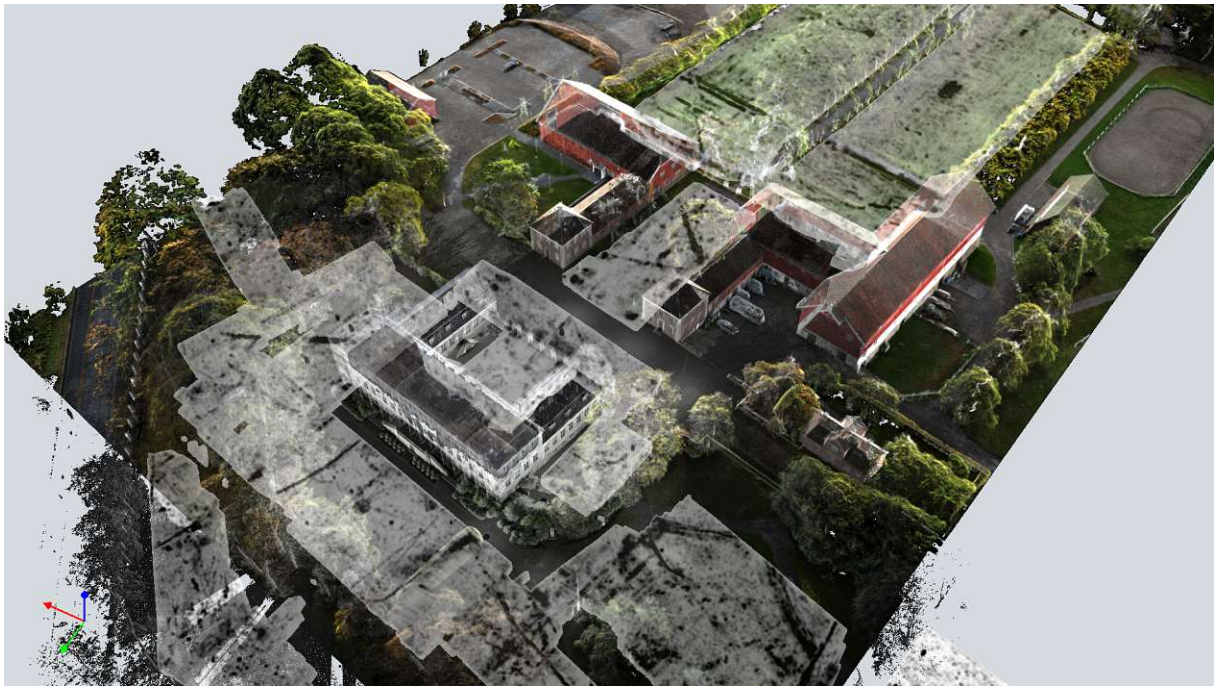


Fig. 4.1: IBM and GPR data as 3D point cloud visualisation. The x-ray mode permits the imaging of the buried structures on the terraces south of the manor house.

Fig. 4.2: Buried walls can be seen as dark reflections in the GPR 3D point cloud data, with low reflection intensities rendered transparent. Overlapping structures suggests different building phases. A linear diagonal feature is likely to be caused by a pipe.

Fig. 4.3: View of the same data as depicted in figure 4.2 from the side, showing the different anomalies in the GPR data at different depths. By interactively moving and rotating the data sets in Nubigon, the anomalies and their relationship to each other become easily recognizeable.

(a)



(b)

Fig. 4.4: (a) IBM and TLS 3D point cloud showing Hafslund manor house from NW. In the garden and in front of the main building the TLS scan positions are visible. (b) Additionally, the GPR 3D point cloud integrated with the IBM and TLS point clouds, showing subsurface features in 3D.

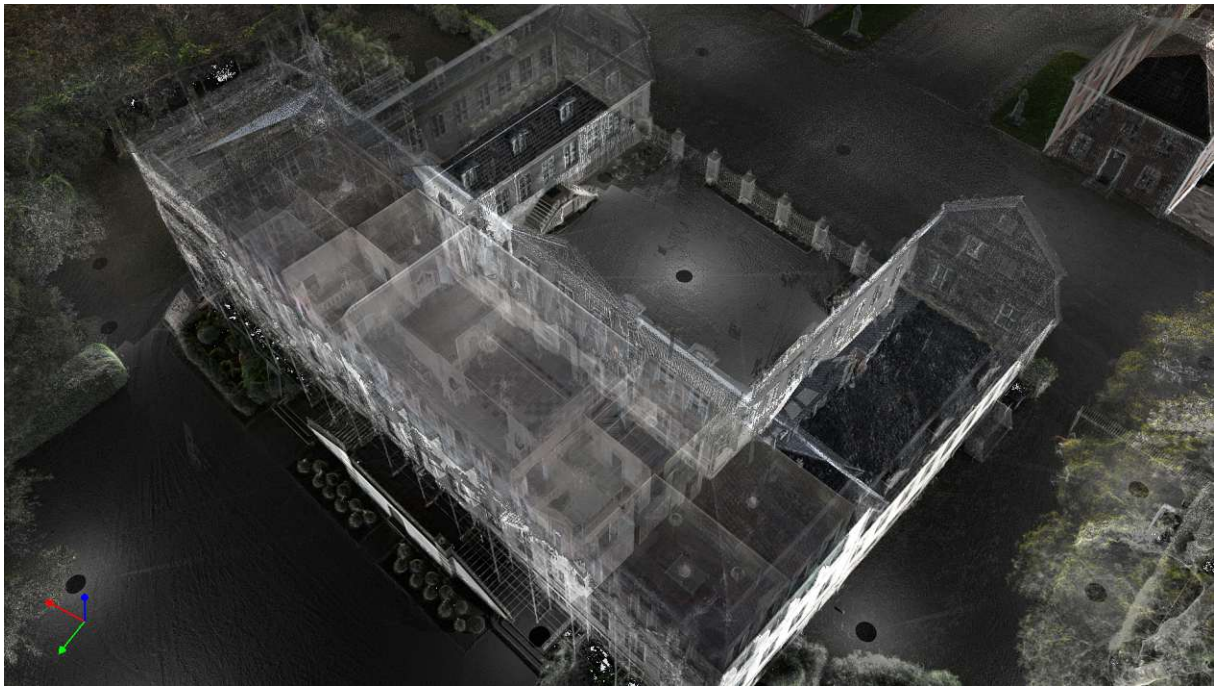


Fig. 4.5: TLS und IBM 3D point cloud visualisation in Nubigon using the x-ray presentation, which permits insights into the interior of the building

4.5 Proposed Workflow

With the current capabilities in sensors, platforms, and software available for this thesis, a workflow has been established for the efficient, comprehensive, and reliable processing of surface and subsurface 3D point cloud data (fig. 4.6). Extensive coverage of the analyzed site using drone images with GNSS information facilitates reliable georeferencing, provided occlusions are minimized in a significant number of images. Incorporating images from handheld cameras at ground level enhances the resolution and accuracy of the resulting IBM 3D point cloud. Pose graph optimization and georeferencing of the TLS data are executed using the custom developed C++ tool by selecting local targets within the TLS point cloud. The global reference

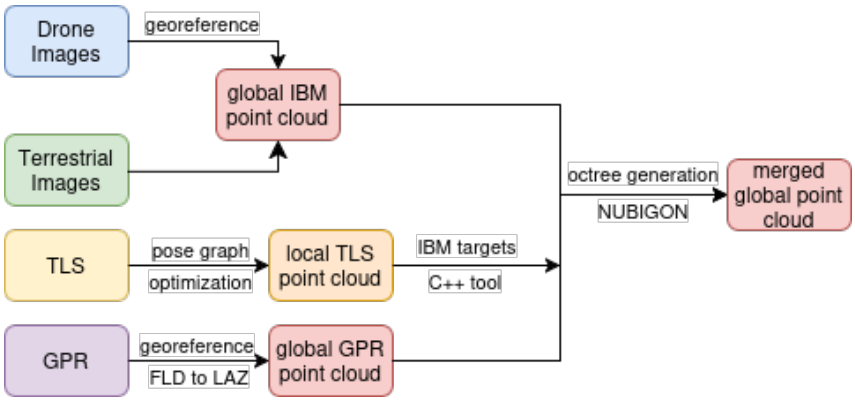


Fig. 4.6: Workflow established for use in future measurement campaigns on archaeological sites, given current sensor and software capabilities available for this thesis.

is established by aligning these targets with corresponding points in the global IBM cloud. GPR data, already globally referenced and topographically corrected, is converted from FLD to LAZ format, with GPR reflection amplitudes stored as intensity values, to facilitate merging with the TLS and IBM datasets as point clouds. Currently, this merging is accomplished by importing these datasets into NUBIGON, where the generation of octrees enables a smooth interaction and further analysis of the merged dataset.

Chapter 5

Conclusion

Unifying geoarchaeological prospection data from image-based modeling (IBM) point clouds, laser scanning (TLS), and georadar (GPR) into a single 3D point cloud visualization presents a number of advantages. By integrating these datasets, multi-layered analysis is possible, enhancing spatial context through the correlation of topographical details from IBM and laser scanning with buried structures identified by GPR. This methodology improves the understanding of subsurface anomalies in relation to surface features and ensures better spatial alignment, thereby reducing possible interpretation errors.

The combination of point clouds from various sources facilitates cross-validation and refinement of spatial positioning, which is especially valuable in complex terrains and where GNSS reliability may be compromised. Laser scanning offers high-precision georeferencing, enabling the alignment and correction of IBM and GPR datasets, ensuring consistency and enhancing data registration.

Realistic digital 3D representations enable researchers, site managers, stakeholders, and the general public to better grasp archaeological features and contexts by visualizing them in an common 3D environment. Point cloud-based models support interactive exploration, making discoveries more tangible and accessible, thus enhancing stakeholder engagement.

Merged 3D datasets can be utilized for automated feature recognition and quantitative analysis, such as calculating the volume of buried structures. The integration of different datasets permits the cross-verification of subsurface features detected by GPR with terrain data from laser scanning or IBM, excluding modern disturbances and refining geophysical and archaeological interpretations. Additionally, laser scanning and IBM data can provide topographic corrections for GPR, mitigating errors caused by uneven terrain.

This thesis outlines the methods and challenges in processing point cloud data acquired through IBM, TLS, and GPR. Merging this data can provide advanced means of visualization and further benefit the extended geoarchaeological characterization of imaged anomalies. The combined surface and subsurface mapping and imaging can result in more comprehensive, complete, and realistic data representations. The collaboration between the Vienna Institute for Archaeological Science (VIAS) of the University of Vienna and GeoSphere Austria, along with the Research Centre for Manor Houses in the Baltic Sea Region of Greifswald University, benefits from open-source software solutions to efficiently process these large datasets. The outlined methods for analyzing TLS data are proposed to qualify as such tools, with the possibility of further development to create a comprehensive workflow for processing data acquired for geophysical and archaeological prospecting.

While TLS , IBM and GPR data where collected independent from each other in this case

study, in future we are likely to see integrated survey solutions in which IBM cameras and TLS sensors are mounted on GPR systems for semi-autonomous mapping.

Chapter 6

Outlook

To increase the quality of the data and streamline the processing, some improvements have been researched and will be discussed here.

6.1 TLS aquisition and processing

6.1.1 Pose graph interconnectivity

To achieve higher reliability within the TLS point cloud, interconnectivity of poses within the pose graph should be increased. Avoiding loops with lengthy edge path lengths contributes to lower relative covariances between disonnected poses, providing better congruency of structural features throughout the point cloud. This can be facilitated by positioning the scanner in windows for example, consequently allowing GICP alignments of interior and exterior (see figure 6.1).

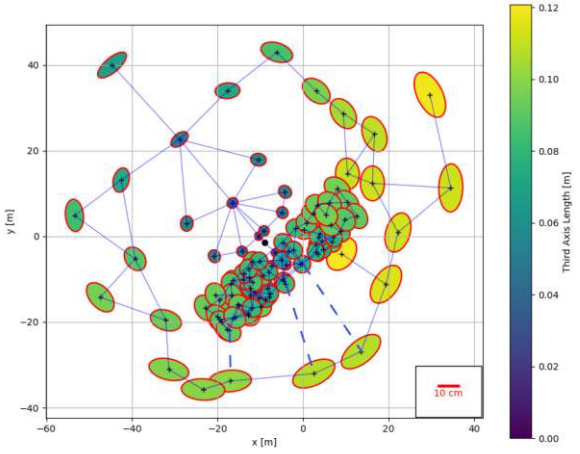


Fig. 6.1: Additional GICP matches (dashed lines) will decrease unfavorable error propagation.

6.1.2 Scan matching

To automate scan matching and pose graph optimization of TLS point clouds, initial coarse alignment of the point clouds must also be automated. Go-ICP (Yang et al. 2015) is a robust and efficient algorithm for global registration of 3D point clouds. It addresses the challenge of aligning point clouds without initial pose estimates by leveraging a Branch and Bound (BnB) optimization framework. BnB is an algorithmic strategy used for solving optimization problems, particularly in discrete and combinatorial contexts. It systematically explores the solution space by dividing it into smaller subsets (branching) and calculating bounds on the optimal solution within each subset (bounding). The algorithm excludes subsets that cannot contain the optimal solution, thereby reducing the search space and improving efficiency. BnB guarantees

finding the global optimum by exhaustively searching the feasible region while eliminating non-promising candidates (see figure 6.2). Go-ICP guarantees the discovery of the globally optimal solution within a specified error bound, making it highly reliable for applications requiring precise registration. The algorithm is designed to handle significant levels of noise and outliers, ensuring robust performance in real-world scenarios. By implementing this approach in a custom portable software solution, such as a monitoring measure on a tablet, on-site pre-registration can be automated. This eliminates the need for commercial software like the Cyclone Field app, providing a more flexible and cost-effective solution for point cloud registration tasks.

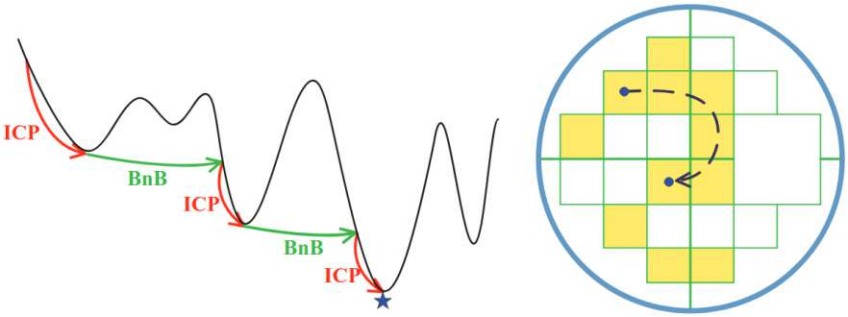


Fig. 6.2: Collaboration of BnB and ICP. Left: BnB and ICP collaboratively update the upper bounds during the search process. Right: with the guidance of BnB, ICP only explores un-discarded, promising cubes with small lower bounds marked up by BnB (Yang et al. 2015).

6.1.3 Pose graph optimization and global reference

To address unfavorable error propagation and resolve global ambiguity in pose graph optimization, the inclusion of landmarks is recommended. These landmarks can be GNSS targets, provided their measurement accuracy is appropriate for the scale, or other targets acquired through methods with significantly lower uncertainty parameters, compared to GICP error propagation. By attributing stochastic information to these landmarks, the pose graph can be adjusted proportionally, transforming it to the global frame based on the landmarks' coordinates and stochastic data. This approach can be directly integrated into the g2o framework, using the methodology used in bundle adjustment. For further details, please refer to Kümmerle et al. 2011.

6.2 LOAM

In recent years, LiDAR sensors have been extensively researched for use on moving platforms, leading to significant improvements in autonomous mapping accuracy and position estimation. The deployment of such sensors on moving platforms is particularly advantageous for surveying vast areas, especially when autonomously driven. Consequently, automated positioning and navigation within the surveyed space become critical priorities. LiDAR Odometry and Mapping (LOAM), introduced by J. Zhang and Singh 2014, is a real-time method for odometry and mapping that utilizes range measurements from a 2-axis LiDAR operating in 6-DOF. The primary challenge addressed by LOAM is the asynchronous nature of range measurements, which can lead to motion estimation errors and subsequent misalignment in the point cloud. Unlike off-line batch methods that rely on loop closure to correct drift, LOAM achieves low-drift and low-computational complexity without the need for high-accuracy ranging or inertial measurements (see fig. 6.3).

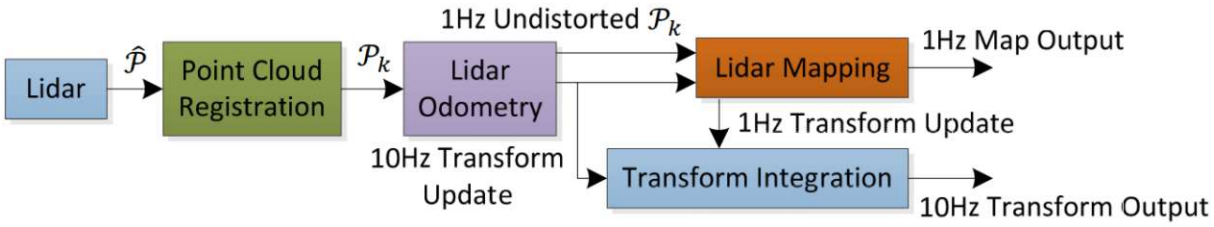


Fig. 6.3: Diagram of the LOAM software system (J. Zhang and Singh 2014).

The method innovatively divides the complex problem of simultaneous localization and mapping into two complementary algorithms: one for high-frequency, low-fidelity odometry to estimate LiDAR velocity, and another for lower-frequency, fine matching and registration of the point cloud. This dual approach enables real-time mapping capabilities. Extensive experiments and evaluations on the KITTI odometry benchmark have demonstrated that LOAM achieves accuracy comparable to state-of-the-art offline batch methods. A particularly compelling application of such a system is as a localization method for environments with limited GNSS coverage, such as forests. The deployment of LOAM in these settings offers a promising solution for accurate positioning and mapping. This is especially relevant for archaeological investigations and forestry applications (Butnor et al. 2003), where magnetometry mapping of subsurface anthropogenic structures and potentially GPR mapping for root investigations are crucial.

6.3 VR and AR applications

Merged 3D datasets can be utilized for immersive visualization in Augmented Reality (AR) and Virtual Reality (VR) environments. AR overlays digital information onto the real world, while VR creates a fully immersive digital environment. These technologies provide valuable tools for both research and public engagement by allowing users to interact with and explore archaeological sites in a more engaging manner, as well as remotely. The generated virtual 3D models can be augmented with additional information in form of texts, images, videos, or further 3D content.

6.4 Closing Remarks

By offering open-source access to the developed C++ tool for processing TLS data, users are empowered to deploy the mentioned enhancements and contribute to the project. This approach also encourages discourse about potential improvements, fostering collaboration and innovation within the community. As a result, the tool's capabilities can be collectively refined and expanded. Applying the suggested changes to the current workflow of conducting the measurements and processing will increase efficiency, accuracy and reliability of the resulting point cloud stemming from TLS, as well as encourage more engaging interaction with the merged data set, consequently improving the quality of the derived information from the digital twin. A well-structured 3D dataset serves as a digital archive, preserving both archaeological and cultural heritage contexts and landscape settings for future research and analysis. By continuing to incorporate mapping technologies like LOAM and potentially more, the acquisition of data can be improved in terms of flexibility, enabling the execution of such measurements in more diverse and challenging environments. The use of AR and VR enables more engaging interaction and better communication of the acquired results.

Ultimately, enhancing data interaction will improve data acquisition, and vice versa. This thesis aims to encourage this development, contributing to advancing archaeological research using geospatial data and enhancing public engagement with cultural heritage.

Bibliography

Abdelhafiz, A. (2013). "Laser scanner point cloud colouring algorithm applied on real site". In: *Survey Review* 45.332, pp. 343–351. ISSN: 0039-6265.

Agarwal, S., Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski (Oct. 2011). "Building Rome in a day". In: *Commun. ACM* 54.10, pp. 105–112. ISSN: 0001-0782. DOI: 10.1145/2001269.2001293. URL: https://doi.org/10.1145/2001269.2001293.

Arun, K. S., T. S. Huang, and S. D. Blostein (1987). "Least-squares fitting of two 3-D point sets". In: *IEEE Transactions on pattern analysis and machine intelligence* 5, pp. 698–700. ISSN: 0162-8828.

Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool (June 2008). "Speeded-Up Robust Features (SURF)". In: *Similarity Matching in Computer Vision and Multimedia* 110.3, pp. 346–359. ISSN: 1077-3142. URL: https://www.sciencedirect.com/science/article/pii/S1077314207001555.

Benedetto, A., F. Tosti, L. B. Ciampoli, and F. D'amico (2017). "An overview of ground-penetrating radar signal processing techniques for road inspections". In: *Signal processing* 132, pp. 201–209. ISSN: 0165-1684.

Besl, P. J. and N. D. McKay (1992). "Method for registration of 3-D shapes". In: vol. 1611. Spie, pp. 586–606.

Bevington, P. and D. Robinson (1969). "Error Analysis for the Physical Sciences". In.

Butnor, J. R., J. A. Doolittle, K. H. Johnsen, L. Samuelson, T. Stokes, and L. Kress (2003). "Utility of ground-penetrating radar as a root biomass survey tool in forest systems". In: *Soil Science Society of America Journal* 67.5, pp. 1607–1615. ISSN: 0361-5995.

Cassidy, N. J. and H. M. Jol (2009). "Ground penetrating radar data processing, modelling and analysis". In: *Ground penetrating radar: theory and applications*, pp. 141–176.

Chen, Y. and G. Medioni (1992). "Object modelling by registration of multiple range images". In: *Image and vision computing* 10.3, pp. 145–155. ISSN: 0262-8856.

Conyers, L. B. (2023). *Ground-Penetrating Radar for Archaeology*. Rowman & Littlefield Publishers. ISBN: 9781538179352.

Daniels, D. J. (2004). *Ground penetrating radar*. Vol. 1. Iet.

Daniels, J. J. (2000). "Ground penetrating radar fundamentals". In: *Prepared as an appendix to a Report to the US EPA, Region V*, pp. 1–21.

Grasmueck, M. (1996). "3-D ground-penetrating radar applied to fracture imaging in gneiss". In: *Geophysics* 61.4, pp. 1050–1064. ISSN: 0016-8033.

Hartley, R. and A. Zisserman (2003). *Multiple view geometry in computer vision*. Cambridge university press.

Isenburg, M. (2013). "LASzip: lossless compression of LiDAR data". In: *Photogrammetric engineering and remote sensing* 79.2, pp. 209–217. ISSN: 0099-1112.

Kamp, N. K., S. Russ, O. Sass, G. Tiefengraber, and S. Tiefengraber (2014). "A Fusion of GPR-and LiDAR-Data for Surveying and Visualisation of Archaeological Structures – a case example of an archaeological site in Strettweg, District of Murtal, Austria". In: *Geophysical Research Abstracts*. Vol. 16. EGU General Assembly 2014-12621. EGU.

Koide, K. (Aug. 2024). "small_gicp: Efficient and parallel algorithms for point cloud registration". In: *Journal of Open Source Software* 9.100, p. 6948. ISSN: 2475-9066. DOI: https://doi.org/10.21105/joss.06948.

Kraus, K. (2004). *Photogrammetrie, 7. Auflage*. Walter de Gruyter Verlag, Berlin, Germany.

Kümmerle, R., G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard (2011). "g 2 o: A general framework for graph optimization". In: *2011 IEEE international conference on robotics and automation*. IEEE, pp. 3607–3613.

Lämmel, R. and S. P. Jones (2003). "Scrap your boilerplate: a practical design pattern for generic programming". In: *ACM SIGPLAN Notices* 38.3, pp. 26–37. ISSN: 0362-1340.

Lepetit, V., F. Moreno-Noguer, and P. Fua (Feb. 2009). "EPnP: An Accurate O(n) Solution to the PnP Problem". In: *International Journal of Computer Vision* 81.2, pp. 155–166. ISSN: 1573-1405. URL: https://doi.org/10.1007/s11263-008-0152-6.

Levenberg, K. (1944). "A method for the solution of certain non-linear problems in least squares". In: *Quarterly of applied mathematics* 2.2, pp. 164–168. ISSN: 0033-569X.

Lohmann, N. (Nov. 28, 2023). *JSON for Modern C++*. Version 3.11.3. URL: https://json.nlohmann.me.

Lovegrove, S. and R. Newcombe (Aug. 2024). *Pangolin*. https://github.com/stevenlovegrove/Pangolin.

Lowe, D. G. (Nov. 2004). "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2, pp. 91–110. ISSN: 1573-1405. URL: https://doi.org/10.1023/B:VISI.0000029664.99615.94.

Maloney, A. (June 2024). *libE57Format*. https://github.com/asmaloney/libE57Format.

Marquardt, D. W. (June 1963). "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". In: *Journal of the Society for Industrial and Applied Mathematics* 11.2, pp. 431–441. ISSN: 2168-3484. DOI: https://doi.org/10.1137/0111030.

Merkle, D., C. Frey, and A. Reiterer (Nov. 2020). "Fusion of ground penetrating radar and laser scanning for infrastructure mapping". In: *Journal of Applied Geodesy* 15.1, pp. 31–45. DOI: 10.1515/jag-2020-0004.

Nau, E. (2024). *ApInsight: Ground Penetrating Radar Data Analysis Tool*. URL: https://github.com/erichnau/ApInsight.

Neal, A. (2004). "Ground-penetrating radar and its use in sedimentology: principles, problems and progress". In: *Earth-science reviews* 66.3-4, pp. 261–330. ISSN: 0012-8252.

Niemeier, W. (2008). *Ausgleichungsrechnung: Statistische Auswertemethoden*. Walter de Gruyter.

Pfeifer, N. (2018). *Photogrammetrie*. University Lecture 122.429. TU Wien.

Pfeifer, N., C. Ressl, and W. Karel (2021). *Photogrammetrie Vertiefung*. University Lecture 120.027. TU Wien.

Plattner, A. M. (May 2020). "GPRPy: Open-source ground-penetrating radar processing and visualization software". In: *The Leading Edge* 39.5, pp. 332–337. ISSN: 1070-485X. DOI: 10.1190/tle39050332.1. URL: https://doi.org/10.1190/tle39050332.1.

Rey Otero, I. (Sept. 2015). "Anatomy of the SIFT method". Theses. École normale supérieure de Cachan - ENS Cachan. URL: https://theses.hal.science/tel-01226489.

Reynolds, J. M. (2011). *An introduction to applied and environmental geophysics*. 2. ed., 1. publ. ISBN 9780471485360. Chichester [u.a.]: Chichester [u.a.] : Wiley-Blackwell, XII, 696 S., Ill., graph. Darst., Kt. URL: https://permalink.catalogplus.tuwien.at/AC08527103.

Schmidt, A., T. Sparrow, C. Gaffney, V. Gaffney, A. S. Wilson, and R. A. E. Coningham (2023). "4D with accuracy: why bother?" In: *Non-Intrusive Methodologies for Large Area Urban Research*. Archaeopress Publishing Ltd, pp. 106–113.

Schonberger, J. L. and J.-M. Frahm (2016). "Structure-from-motion revisited". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113.

Segal, A., D. Haehnel, and S. Thrun (2009). "Generalized-icp." In: vol. 2. 4. Seattle, WA, p. 435.

Triggs, B., P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon (2000). "Bundle Adjustment – A Modern Synthesis". In: *Vision Algorithms: Theory and Practice*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 298–372.

Trinks, I., A. Hinterleitner, W. Neubauer, E. Nau, K. Löcker, M. Wallner, M. Gabler, R. Filzwieser, J. Wilding, H. Schiel, V. Jansa, P. Schneidhofer, T. Trausmuth, V. Sandici, D. Ruß, S. Flöry, J. Kainz, M. Kucera, A. Vonkilch, T. Tencer, L. Gustavsen, M. Kristiansen, L.-M. Bye-Johansen, C. Tonning, T. Zitz, K. Paasche, T. Gansum, and S. Seren (2018). "Large-area high-resolution ground-penetrating radar measurements for archaeological prospection". In: *Archaeological Prospection* 25.3, pp. 171–195. DOI: https://doi.org/10.1002/arp.1599. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/arp.1599. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/arp.1599.

Wolf, J., S. Discher, L. Masopust, S. Schulz, R. Richter, and J. Döllner (2018). "Combined visual exploration of 2D ground radar and 3D point cloud data for road environment". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W10, pp. 231–236. DOI: 10.5194/isprs-archives-XLII-4-W10-231-2018. URL: https://isprs-archives.copernicus.org/articles/XLII-4-W10/231/2018/.

Yang, J., H. Li, D. Campbell, and Y. Jia (2015). "Go-ICP: A globally optimal solution to 3D ICP point-set registration". In: *IEEE transactions on pattern analysis and machine intelligence* 38.11, pp. 2241–2254. ISSN: 0162-8828.

Yilmaz, Ö. (2001). *Seismic data analysis*. Vol. 1. Society of Exploration Geophysicists Tulsa.

Zhang, J. and S. Singh (2014). "LOAM: Lidar odometry and mapping in real-time." In: vol. 2. 9. Berkeley, CA, pp. 1–9.

Zhang, W., J. Qi, P. Wan, H. Wang, D. Xie, X. Wang, and G. Yan (2016). "An easy-to-use airborne LiDAR data filtering method based on cloth simulation". In: *Remote sensing* 8.6, p. 501. ISSN: 2072-4292.

Chapter 7

Appendix

7.1 Custom C++ tool for TLS point cloud processing

The critical functions used for the custom C++ tool are attached in this chapter. Note that the code presented here reflects the state of the tool as of February 2025 and is subject to change.

7.1.1 E57 file reading

```cpp
void readE57(const std::string& filename, const std::string& scanName, size_t maxPoints = 500000) {
    std::string extension = filename.substr(filename.find_last_of(".") + 1);

    if (extension != "e57") {
        std::cout << "Unsupported file format: " << extension << std::endl;
        return;
    }

    e57::Reader e57Reader(filename, e57::ReaderOptions());
    std::cout << "Reading file: " << filename << std::endl;
    std::cout << "Scans in file:" << e57Reader.GetData3DCount() << std::endl;

    for (size_t scanIndex = 0; scanIndex < e57Reader.GetData3DCount(); ++scanIndex) {
        e57::Data3D scanHeader;
        e57Reader.ReadData3D(scanIndex, scanHeader);

        if (scanHeader.name != scanName) {
            if (!(e57Reader.GetData3DCount() == 1)) {
                continue;
            }
        }
        std::cout << "Reading Scan: " << scanHeader.name << std::endl;
        std::string scanKey = filename + "_" + scanHeader.name;

        size_t nPoints = scanHeader.pointCount;
        std::cout << "Number of points: " << nPoints << std::endl;
        std::vector<float> xData(nPoints), yData(nPoints), zData(nPoints);
        std::vector<uint16_t> rData(nPoints), gData(nPoints), bData(nPoints);
        std::vector<double> intensityData(nPoints);
        std::vector<Eigen::Vector3f> points, colors, subsampledPoints, subsampledColors;

        // Set up the buffer for reading the data
```

```
33          e57::Data3DPointsFloat buffers;
34          buffers.cartesianX = xData.data();
35          buffers.cartesianY = yData.data();
36          buffers.cartesianZ = zData.data();
37          buffers.colorRed = rData.data();
38          buffers.colorGreen = gData.data();
39          buffers.colorBlue = bData.data();
40          buffers.intensity = intensityData.data();
41
42          // Read the data from the scan
43          e57::CompressedVectorReader dataReader = e57Reader.SetUpData3DPointsData(scanIndex,
44              nPoints, buffers);
45          dataReader.read();
46          dataReader.close();
47          std::cout << "Reading finished " << std::endl;
48
49          // read poses from scanPositions:
50          Eigen::Matrix4d localPoseMatrix;
51          for (size_t i = 0; i < scanPositions.size(); i++) {
52              if (scanPositions[i].key == scanKey) {
53                  localPoseMatrix = scanPositions[i].LocalPose;
54              }
55          }
56
57          points.reserve(nPoints);  // Pre-allocate memory for points
58          colors.reserve(nPoints);  // Pre-allocate memory for colors
59
60          // Parallelized loop for transforming points
61          #pragma omp parallel for
62          for (size_t i = 0; i < nPoints; ++i) {
63              // Apply the transformation to the point
64              Eigen::Vector4f point(xData[i], yData[i], zData[i], 1.0f);
65              Eigen::Vector4f transformedPoint = localPoseMatrix.cast<float>() * point;
66              Eigen::Vector3f transformedPoint3f = transformedPoint.head<3>();
67
68              // If color is invalid, use intensity for grayscale rendering
69              bool isColorInvalid = (rData[i] == 255 && gData[i] == 255 && bData[i] == 255);
70              Eigen::Vector3f color;
71              if (isColorInvalid) {
72                  float grayscale = static_cast<float>(intensityData[i]);
73                  color = Eigen::Vector3f(grayscale, grayscale, grayscale);
74              } else {
75                  // Use RGB values as they are
76                  color = Eigen::Vector3f(rData[i] / 255.0f, gData[i] / 255.0f, bData[i] / 255.0f);
77              }
78
79              #pragma omp critical
80              {
81                  points.push_back(transformedPoint3f);
82                  colors.push_back(color);
```

```
            }
        }

        // Subsample points and colors
        SubsamplePointCloud(points, colors, subsampledPoints, subsampledColors, maxPoints);
        size_t subsize = subsampledPoints.size();

        // Create PangoPointCloud using only points and colors
        auto pangoCloudSub = std::make_unique<PangoPointCloud>(subsampledPoints, subsampledColors);
        auto pangoCloudFull = std::make_unique<PangoPointCloud>(points, colors);

        // Store scan data in the loadedScans map
        ScanData scanData;
        scanData.subsampledCloud = std::move(pangoCloudSub);
        scanData.fullCloud = std::move(pangoCloudFull);
        scanData.isRendered = true;

        loadedScans[scanKey] = std::move(scanData);

        std::cout << "Scan " << scanHeader.name << " loaded with " << subsize
            << " subsampled points.\n" << std::endl;
    }
}
```

7.1.1.1 Subsampling

```
void SubsamplePointCloud(const std::vector<Eigen::Vector3f>& points,
                         const std::vector<Eigen::Vector3f>& colors,
                         std::vector<Eigen::Vector3f>& subsampledPoints,
                         std::vector<Eigen::Vector3f>& subsampledColors,
                         size_t maxPoints) {
    size_t n = points.size();
    if (n <= maxPoints) {
        subsampledPoints = points;
        subsampledColors = colors;
        return;
    }

    subsampledPoints.resize(maxPoints);
    subsampledColors.resize(maxPoints);

    std::random_device rd;
    std::default_random_engine rng(rd());

    #pragma omp parallel for
    for (size_t i = 0; i < maxPoints; ++i) {
        size_t randomIndex = rng() % n; // Pick a random point
        subsampledPoints[i] = points[randomIndex];
        subsampledColors[i] = colors[randomIndex];
```

```
    }
}
```

7.1.2 Generalized ICP registration

```
void ICP(const std::vector<FileEntry>& fileEntries, const int downsample) {

    ProcessSelectedScans(fileEntries);

    // Ensure exactly 2 scans are rendered
    int renderedCount = 0;
    std::vector<std::string> renderedScans; // To store the keys of the rendered scans
    for (const auto& [key, scanData] : loadedScans) {
        if (scanData.isRendered) {
            renderedScans.push_back(key);
            if (++renderedCount > 2) {
                std::cerr << "Error: Selection of only 2 scans allowed for ICP." << std::endl;
            }
        }
    }

    if (renderedCount != 2) {
        std::cerr << "Error: Exactly 2 scans must be rendered for ICP." << std::endl;
    }
    DrawAll();
    DrawScanPositions(scanPositions);
    pangolin::FinishFrame();

    // Randomly assign source and target initially
    std::srand(std::time(nullptr)); // Seed random number generator
    int sourceIndex = std::rand() % 2;
    int targetIndex = 1 - sourceIndex;
    std::string extractedSource = ExtractFilepathAndSuffix(renderedScans[sourceIndex]).second;
    std::string extractedTarget = ExtractFilepathAndSuffix(renderedScans[targetIndex]).second;

    std::cout << "Source cloud (transform): " << renderedScans[sourceIndex] << "\n";
    std::cout << "Target cloud (fixed): " << renderedScans[targetIndex] << "\n";

    if (sourceIndex < 0 || sourceIndex >= renderedScans.size() ||
        targetIndex < 0 || targetIndex >= renderedScans.size()) {
        std::cerr << "Error: Invalid source/target index.\n";
    }

    // Access the selected scans
    auto& sourceCloud = loadedScans[renderedScans[sourceIndex]];
    auto& targetCloud = loadedScans[renderedScans[targetIndex]];

    std::vector<Eigen::Vector3f> source_points = sourceCloud.fullCloud->getVertices();
```

```
44      std::vector<Eigen::Vector3f> target_points = targetCloud.fullCloud->getVertices();
45
46      if (source_points.empty() || target_points.empty()) {
47          std::cerr << "Error: No points in source or target cloud.\n";
48      }
49
50      small_gicp::RegistrationSetting setting;
51      unsigned int nThreads = std::thread::hardware_concurrency();
52      std::cout << "Number of threads available: " << nThreads << std::endl;
53      setting.num_threads = nThreads;
54      float down_float = downsample / 100.0f;
55      setting.downsampling_resolution = down_float;
56      setting.max_correspondence_distance = 0.4;
57      setting.max_iterations = 10;
58      setting.verbose = true;
59
60      std::cout << "Source points size: " << source_points.size() << "\n";
61      std::cout << "Target points size: " << target_points.size() << "\n";
62
63      Eigen::Isometry3d init_T_target_source = Eigen::Isometry3d::Identity();
64
65      small_gicp::RegistrationResult result = small_gicp::align(target_points, source_points,
66          init_T_target_source, setting);
67
68      Eigen::Isometry3d T = result.T_target_source;  // Estimated transformation
69      size_t num_inliers = result.num_inliers;       // Number of inlier source points
70      Eigen::Matrix<double, 6, 6> H = result.H;      // Final Hessian matrix (6x6)
71      bool conv = result.converged;
72      double err = result.error;
73      Eigen::Matrix<double, 6, 1> b = result.b;
74
75      std::cout << "ICP complete. Transformation matrix:\n" << T.matrix() << "\n";
76      std::cout << "Number of inliers: " << num_inliers << "\n";
77      std::cout << "Converged: " << conv << "\n";
78      std::cout << "Error: " << err << "\n";
79      std::cout << "b: " << b << "\n";
80      std::cout << "Iterations: " << result.iterations << "\n";
81
82      // Apply the transformation
83      std::vector<Eigen::Vector3f> subsampled_points = sourceCloud.subsampledCloud->getVertices();
84      for (auto& point : subsampled_points) {
85          Eigen::Vector4f homogenousPoint(point.x(), point.y(), point.z(), 1.0f);
86          Eigen::Vector4f transformedPoint = T.matrix().cast<float>() * homogenousPoint;
87          point = transformedPoint.head<3>();
88      }
89
90      for (auto& point : source_points) {
91          Eigen::Vector4f homogenousPoint(point.x(), point.y(), point.z(), 1.0f);
92          Eigen::Vector4f transformedPoint = T.matrix().cast<float>() * homogenousPoint;
93          point = transformedPoint.head<3>();
```

```
94      }
95
96
97      // Update the source cloud vertices with transformed points
98      sourceCloud.subsampledCloud->setVertices(subsampled_points);
99      sourceCloud.fullCloud->setVertices(source_points);
100
101     Eigen::Matrix4d relPose;
102
103     for (auto& scanPosit : scanPositions) {
104         if (scanPosit.name == extractedSource) {
105             Eigen::Matrix4d updatedLocalPoseSource = T.matrix() * scanPosit.LocalPose;
106             Eigen::Matrix4d updatedGlobalPoseSource = globalShift * updatedLocalPoseSource;
107
108             scanPosit.LocalPose = updatedLocalPoseSource;
109             scanPosit.GlobalPose = updatedGlobalPoseSource;
110             scanPosit.positionLocal = updatedLocalPoseSource.block<3, 1>(0, 3).cast<float>();
111
112             std::cout << "Updated scan position for source (" << scanPosit.name << "):\n";
113             std::cout << "New global pose:\n" << scanPosit.GlobalPose << "\n";
114
115             for (auto& scanPositT : scanPositions) {
116                 if (scanPositT.name == extractedTarget) {
117                     // relPose in respect to the targets local coordinate frame!!
118                     relPose = scanPositT.LocalPose.inverse() * scanPosit.LocalPose;
119                 }
120             }
121
122         }
123     }
124
125     // Saving results to map and updating json file:
126     std::string icpKey = renderedScans[sourceIndex] + "<->" + renderedScans[targetIndex];
127     ICPResult icpR;
128     icpR = {renderedScans[sourceIndex],renderedScans[targetIndex],result,relPose};
129     icpResults[icpKey] = icpR;
130     currICP = {icpKey, icpR};
131
132     // Redraw the scene to show the transformed source cloud
133     lines = updateLinks(scanPositions);
134     pangolin::FinishFrame();
135     DrawAll();
136     DrawScanPositions(scanPositions);
137     pangolin::FinishFrame();
138 }
```

7.1.3 Pose Graph Optimization

```cpp
void OptimizePoseGraph(const std::unordered_map<std::string, ICPResult>& icpR,
    const std::string& outputFilename, std::pair<double,double> stoch_a_priori) {
    g2o::SparseOptimizer optimizer;
    ConfigureOptimizer(optimizer);

    // Map scan names to node IDs
    std::map<std::string, int> nodeMap; // Maps scan names to graph node IDs

    DrawAll();
    DrawScanPositions(scanPositions);
    pangolin::FinishFrame();

    nlohmann::json optimizationResults = nlohmann::json::object();

    // Add nodes (scans) to the graph
    for (size_t i = 0; i < scanPositions.size(); ++i) {
        const auto& scan = scanPositions[i];
        g2o::VertexSE3* vertex = new g2o::VertexSE3();
        vertex->setId(i);


        std::vector<Eigen::Matrix4d> poses;
        // Check if poses are available, otherwise use the default LocalPose
        if (!poses.empty()) {
            vertex->setEstimate(Eigen::Isometry3d(ComputeMeanPose(poses)));
        } else {
            vertex->setEstimate(Eigen::Isometry3d(scan.LocalPose));
        }

        if (i == 0) vertex->setFixed(true);  // Fix the first node as the reference
        optimizer.SparseOptimizer::addVertex(vertex);
        nodeMap[scan.key] = i;

        // Save initial vertex data
        optimizationResults["vertices"][scan.key] = {
            {"initialPose", MatrixToJson(scan.LocalPose)},
            {"optimizationTransform", {}},
            {"optimizedPose", {}}, // Will be updated later
            {"globalPose", {}},
            {"covariance", {}}
        };
    }

    for (const auto& result : icpR) {
        const std::string& sourceName = result.second.sourceName;
        const std::string& targetName = result.second.targetName;

        int sourceId = nodeMap[sourceName];
```

```
49          int targetId = nodeMap[targetName];
50
51          // Create a new edge
52          g2o::EdgeSE3* edge = new g2o::EdgeSE3();
53
54          edge->setVertex(0, optimizer.vertex(sourceId));
55          edge->setVertex(1, optimizer.vertex(targetId));
56          edge->setMeasurement(Eigen::Isometry3d(result.second.relPose.inverse()));
57
58          // Set the edge information matrix
59          if (result.second.result.error == 0.0) {
60              info = cov.inverse() * 1e9;  // Fixed pose → high weight
61          } else {
62              cov.block<3,3>(0,0) = Eigen::Matrix3d::Identity() *
63                  stoch_a_priori.first*stoch_a_priori.first;
64              double sigma_squared_rot_custom = (stoch_a_priori.first / d_avg) *
65                  (stoch_a_priori.first / d_avg);
66              cov.block<3,3>(3,3) = Eigen::Matrix3d::Identity() *
67                  stoch_a_priori.second*stoch_a_priori.second;
68              info = cov.inverse();
69          }
70
71          // Add the edge to the optimizer
72          optimizer.SparseOptimizer::addEdge(edge);
73
74          // Save edge data
75          optimizationResults["edges"].push_back({
76              {"source", sourceName},
77              {"target", targetName},
78              {"relativePose", MatrixToJson(result.second.relPose.inverse().matrix())},
79              {"Hessian", MatrixToJson(medianHess)},
80              {"residualNorm", {}}, // Will be updated later
81          });
82      }
83
84
85      // Perform optimization
86      optimizer.setVerbose(true);
87      optimizer.computeInitialGuess();
88      std::cout << "Initial chi2: " << optimizer.chi2() << std::endl;
89
90      optimizer.initializeOptimization();
91      optimizer.optimize(1000);
92
93      std::cout << "Final chi2: " << optimizer.chi2() << std::endl;
94
95      for (const auto* edge : optimizer.edges()) {
96          const auto* e = dynamic_cast<const g2o::EdgeSE3*>(edge);
97          if (!e) continue;
98
```

```
99          // Get the two vertices associated with the edge
100         const auto* v1 = dynamic_cast<const g2o::VertexSE3*>(e->vertex(0));
101         const auto* v2 = dynamic_cast<const g2o::VertexSE3*>(e->vertex(1));
102
103         // Get their optimized estimates
104         Eigen::Isometry3d T1 = v1->estimate();
105         Eigen::Isometry3d T2 = v2->estimate();
106         std::string source;
107         std::string target;
108         std::string sourceP;
109         std::string targetP;
110         for (auto& node : nodeMap) {
111             if (node.second == v1->id()) {
112                 source = ExtractFilepathAndSuffix(node.first).second;
113                 sourceP = node.first;
114             } else if (node.second == v2->id())
115             {
116                 target = ExtractFilepathAndSuffix(node.first).second;
117                 targetP = node.first;
118             }
119
120         }
121
122         // Compute the measured transformation from ICP
123         Eigen::Isometry3d T_measured = e->measurement();
124
125         // Compute the relative transformation from optimized poses
126         Eigen::Isometry3d T_optimized = T1.inverse() * T2;
127
128         // Compute the residual
129         Eigen::Isometry3d errorTransform = T_measured.inverse() * T_optimized;
130         Eigen::Matrix< double, 6, 1 > residual = g2o::internal::toVectorMQT(errorTransform);
131
132         // Compute the norm of the residual (optional)
133         double residualNorm = residual.norm();
134
135         // Debug output
136         std::cout << "Edge between vertices " << source << " and " << target <<
137             " has residual norm: " << residualNorm << std::endl;
138
139         // Threshold to identify outliers
140         if (residualNorm > 0.05) {
141             std::cout << "High residual detected for edge: " << source << " -> " << target <<
142                 std::endl;
143         }
144
145         // Update JSON with residuals
146         for (auto& edgeJson : optimizationResults["edges"]) {
147             if (edgeJson["source"] == sourceP &&
148                 edgeJson["target"] == targetP) {
```

```
149                 edgeJson["residualNorm"] = residualNorm;
150                 break;
151             }
152         }
153     }
154 
155     std::unordered_map<std::string, Eigen::Matrix4d> trafos;
156     // Update scan positions with optimized poses
157     nlohmann::json optimizedData = nlohmann::json::object();
158 
159     for (size_t i = 0; i < scanPositions.size(); ++i) {
160         g2o::VertexSE3* vertex =
161             static_cast<g2o::VertexSE3*>(optimizer.vertex(nodeMap[scanPositions[i].key]));
162         // prevScanPoses.emplace_back(scanPositions[i].LocalPose);
163         trafos[scanPositions[i].key] = vertex->estimate().matrix() *
164             scanPositions[i].LocalPose.inverse();
165         Eigen::Matrix4d initialPose = scanPositions[i].LocalPose;
166         scanPositions[i].LocalPose = vertex->estimate().matrix();
167         scanPositions[i].GlobalPose = globalShift * scanPositions[i].LocalPose;
168         scanPositions[i].positionLocal =
169             scanPositions[i].LocalPose.block<3, 1>(0, 3).cast<float>();
170 
171         // Add to JSON data
172         nlohmann::json scanJson;
173         scanJson["ID"] = scanPositions[i].key;
174         scanJson["GlobalPose"] = MatrixToJson(scanPositions[i].GlobalPose);
175         optimizedData[scanPositions[i].key] = scanJson;
176 
177         // Update JSON data:
178         optimizationResults["vertices"][scanPositions[i].key] = {
179             {"initialPose", MatrixToJson(initialPose)},
180             {"optimizedPose", MatrixToJson(scanPositions[i].LocalPose)}, // Will be updated later
181             {"globalPose", MatrixToJson(scanPositions[i].GlobalPose)}
182         };
183 
184         g2o::SparseBlockMatrix<Eigen::MatrixXd> spinv;
185         Eigen::MatrixXd hessianBlock;
186         if (optimizer.computeMarginals(spinv, vertex)) {
187             auto block = spinv.block(vertex->hessianIndex(), vertex->hessianIndex());
188             if (block) {
189                 hessianBlock = *block;
190                 // std::cout << "Hessian block for vertex:\n" << hessianBlock << std::endl;
191             }
192         }
193         const Eigen::MatrixXd covariance = hessianBlock;
194         optimizationResults["vertices"][scanPositions[i].key]["covariance"] = MatrixToJson(covariance);
195     }
196 
197     // Save updated poses to a new JSON file
198     std::ofstream outFile(outputFilename);
```

```
    if (!outFile) {
        throw std::runtime_error("Failed to open file for writing: " + outputFilename);
    }
    outFile << optimizedData.dump(4);
    outFile.close();

    for (const auto& scan : loadedScans) {
        if (!scan.second.subsampledCloud->getVertices().empty()) {
            std::vector<Eigen::Vector3f> subsampled_points = scan.second.subsampledCloud->getVertices();
            for (auto& point : subsampled_points) {
                Eigen::Vector4f homogenousPoint(point.x(), point.y(), point.z(), 1.0f);
                Eigen::Vector4f transformedPoint = trafos[scan.first].cast<float>() * homogenousPoint;
                point = transformedPoint.head<3>();
            }
            std::vector<Eigen::Vector3f> full_points = scan.second.fullCloud->getVertices();
            for (auto& point : full_points) {
                Eigen::Vector4f homogenousPoint(point.x(), point.y(), point.z(), 1.0f);
                Eigen::Vector4f transformedPoint = trafos[scan.first].cast<float>() * homogenousPoint;
                point = transformedPoint.head<3>();
            }
            scan.second.subsampledCloud->setVertices(subsampled_points);
            scan.second.fullCloud->setVertices(full_points);

        }
    }

    // Save results to file
    std::string outFileG2O = "pose_optimization_results.json";
    std::ofstream outFileG2On(outFileG2O);
    if (!outFileG2On) {
        throw std::runtime_error("Failed to open file for writing: " + outFileG2O);
    }
    outFileG2On << optimizationResults.dump(4); // Pretty print with 4 spaces

    std::cout << "Optimization results saved to " << outFileG2O << std::endl;
    outFileG2On.close();

    lines = updateLinks(scanPositions);
    DrawAll();
    DrawScanPositions(scanPositions);
    pangolin::FinishFrame();
}
```

7.1.4 Georeferencing

7.1.4.1 Transformation matrix computation

```
Eigen::Matrix4d computeTransformationMatrix(const std::vector<Target> matchedTargets) {
    // Temporary vectors to store coordinates of selected targets
```

```
    std::vector<Eigen::Vector3d> localPoints;
    std::vector<Eigen::Vector3d> globalPoints;

    // Filter for selected points
    for (const auto& target : matchedTargets) {
        if (target.selected) {
            localPoints.push_back(target.local);
            globalPoints.push_back(target.global);
        }
    }

    // Check for sufficient selected points
    if (localPoints.size() < 3) {
        std::cerr << "Insufficient selected points for transformation." << std::endl;
        std::cout << "Transformation Matrix:\n" << Eigen::Matrix4d::Identity() << std::endl;
        return Eigen::Matrix4d::Identity();
    }

    // Compute centroids of selected points
    Eigen::Vector3d centroidLocal = Eigen::Vector3d::Zero();
    Eigen::Vector3d centroidGlobal = Eigen::Vector3d::Zero();

    for (size_t i = 0; i < localPoints.size(); ++i) {
        centroidLocal += localPoints[i];
        centroidGlobal += globalPoints[i];
    }
    centroidLocal /= localPoints.size();
    centroidGlobal /= globalPoints.size();

    std::cout << "local centroid: " << centroidLocal << "\n" << std::endl;
    std::cout << "global centroid: " << centroidGlobal << "\n" << std::endl;

    // Compute centered vectors
    std::vector<Eigen::Vector3d> localCentered(localPoints.size());
    std::vector<Eigen::Vector3d> globalCentered(globalPoints.size());
    for (size_t i = 0; i < localPoints.size(); ++i) {
        localCentered[i] = localPoints[i] - centroidLocal;
        globalCentered[i] = globalPoints[i] - centroidGlobal;
    }

    // Compute cross-covariance matrix
    Eigen::Matrix3d H = Eigen::Matrix3d::Zero();
    for (size_t i = 0; i < localPoints.size(); ++i) {
        H += localCentered[i] * globalCentered[i].transpose();
    }

    // Singular Value Decomposition
    Eigen::JacobiSVD<Eigen::Matrix3d> svd(H, Eigen::ComputeFullU | Eigen::ComputeFullV);
    Eigen::Matrix3d rotation = svd.matrixV() * svd.matrixU().transpose();

```

```
    // Ensure a proper rotation (determinant = 1)
    if (rotation.determinant() < 0) {
        Eigen::Matrix3d V = svd.matrixV();
        V.col(2) *= -1;
        rotation = V * svd.matrixU().transpose();
    }

    double scale = 1.0; // fix scale to 1

    // Compute translation
    Eigen::Vector3d localTranslation = -rotation * centroidLocal;
    Eigen::Vector3d translation = centroidGlobal + localTranslation;

    // Form the 4x4 transformation matrix
    Eigen::Matrix4d transformation = Eigen::Matrix4d::Identity();
    transformation.block<3, 3>(0, 0) = scale * rotation;
    transformation.block<3, 1>(0, 3) = translation;

    std::cout << "Transformation Matrix:\n" << transformation << std::endl;

    return transformation;
}
```

7.1.4.2 GUI and E57 handling

```
void transformE57(std::vector<FileEntry> fileEntries) {
    try {
        pangolin::DestroyWindow("GeoRefHut");

        // Read local points from the local.txt file
        std::vector<Point> localPointsMap = readPointsFromFile("local.txt");
        if (localPointsMap.empty()) {
            std::cerr << "No local points found." << std::endl;
            return;
        }

        // Select the target coordinates file using tinyfd
        const char* targetCoordinatesFile = tinyfd_openFileDialog(
            "Select target coordinates file",
            curr_path.c_str(), // Default path (empty for user to choose)
            1, // Number of filter patterns
            nullptr, // No specific filter
            "Text files (*.txt)", // Single filter description
            0 // Don't allow multiple selects
        );


        if (!targetCoordinatesFile) {
            std::cerr << "No target file selected." << std::endl;
```

```
25              return;
26          }
27          std::filesystem::path TfilePath(targetCoordinatesFile);
28          curr_path = TfilePath.parent_path();
29
30          // Step 3: Read global points from the selected target file
31          std::vector<Point> globalPointsMap = readPointsFromFile(targetCoordinatesFile);
32          if (globalPointsMap.empty()) {
33              std::cerr << "No global points found." << std::endl;
34              return;
35          }
36
37          std::vector<Target> matchedTargets;
38
39          for (const auto& localPoint : localPointsMap) {
40              for (const auto& globalPoint : globalPointsMap) {
41                  if (localPoint.name == globalPoint.name) {
42                      // Create a Target instance with the matched local and global points
43                      Target target;
44                      target.name = localPoint.name;
45                      target.local = localPoint.coordinates;
46                      target.global = globalPoint.coordinates;
47                      target.selected = true;
48
49                      // Add the target to the matched targets list
50                      matchedTargets.push_back(target);
51                  }
52              }
53          }
54
55          if (matchedTargets.size() < 3) {
56              std::cerr << "Insufficient corresponding points for transformation." << std::endl;
57              return;
58          }
59
60          // Compute transformation matrix
61          Eigen::Matrix4d initialTransformationMatrix = computeTransformationMatrix(matchedTargets);
62
63          for (const auto& point : matchedTargets) {
64              Eigen::Vector4d localHomog(point.local(0), point.local(1), point.local(2), 1.0);
65              Eigen::Vector3d transformed = (initialTransformationMatrix* localHomog).head<3>();
66              transformedPoints.push_back(transformed);
67              Eigen::Vector3d difference = point.global - transformed;
68              double absDifference = difference.norm();
69
70              // Format and align output
71              std::cout << std::fixed << std::setprecision(3);
72              std::cout << std::left << std::setw(8) << point.name << "|"  // Left-align point name
73                      << "diff:"
74                      << std::right << std::setw(6) << difference(0) << " "
```

```
75                      << std::setw(6) << difference(1) << " "
76                      << std::setw(6) << difference(2) << " [m] | "
77                      << "abs. diff: " << std::setw(6) << absDifference << " [m]"
78                      << std::endl;
79          }
80
81          // Recompute transformation matrix based on target selection
82          Eigen::Matrix4d transformationMatrix = trafoViz(initialTransformationMatrix,matchedTargets);
83
84          size_t guidCounter = 1;
85
86          std::unordered_map<std::string, Eigen::Matrix4d> optPoses;
87          std::ifstream fin("optimized_poses.json");
88          if (!posesOptimized && fin) {
89              optPoses = LoadGlobalPoses("optimized_poses.json");
90          } else {
91              for (auto& pose : scanPositions) {
92                  optPoses[pose.key] = pose.GlobalPose;
93              }
94          }
95
96          // Process each input E57 file
97          for (const auto& file : fileEntries) {
98
99              std::filesystem::path firstInputPath(file.filepath);
100             std::string outputFilename;
101             if (fileEntries.size() < 2) {
102                 outputFilename = firstInputPath.stem().string() + "_georef.e57";
103             } else {
104                 outputFilename = firstInputPath.stem().string() + "_georef.e57";
105             }
106
107             std::filesystem::path outputPath = firstInputPath.parent_path() / outputFilename;
108             std::cout << "Writing E57 to: " << outputPath << std::endl;
109
110             // Create the E57 Writer for the single output file
111             e57::WriterOptions writerOptions;
112             e57::Writer e57Writer(outputPath.string().c_str(), writerOptions);
113             if (!e57Writer.IsOpen()) {
114                 std::cerr << "Failed to open E57 writer for " << outputFilename << std::endl;
115                 return;
116             }
117
118             // Open the current E57 file for reading
119             e57::Reader e57Reader(file.filepath, e57::ReaderOptions());
120             std::cout << "Reading E57 file: " << file.filepath << std::endl;
121
122             size_t scanCount = e57Reader.GetData3DCount();
123             std::cout << "Number of scans in the file: " << scanCount << std::endl;
124
```

```
125             // Process each scan in the current file
126             for (size_t scanIndex = 0; scanIndex < scanCount; ++scanIndex) {
127                 e57::Data3D scanHeader;
128                 e57Reader.ReadData3D(scanIndex, scanHeader);
129
130                 std::string scanKey = file.filepath + "_" + scanHeader.name;
131
132                 // Apply the transformation to the scan pose
133                 Eigen::Matrix4d poseMatrix = optPoses[scanKey];
134                 Eigen::Matrix4d transformedPoseMatrix = transformationMatrix * poseMatrix;
135
136                 // Update the scan header with the new pose
137                 Eigen::Quaterniond eigenQuat(transformedPoseMatrix.block<3, 3>(0, 0));
138                 e57::Quaternion e57Quat;
139                 e57Quat.w = eigenQuat.w();
140                 e57Quat.x = eigenQuat.x();
141                 e57Quat.y = eigenQuat.y();
142                 e57Quat.z = eigenQuat.z();
143                 scanHeader.pose.rotation = e57Quat;
144                 scanHeader.pose.translation.x = transformedPoseMatrix(0, 3);
145                 scanHeader.pose.translation.y = transformedPoseMatrix(1, 3);
146                 scanHeader.pose.translation.z = transformedPoseMatrix(2, 3);
147
148                 // Assign a unique GUID for each scan
149                 scanHeader.guid = generateUniqueGUID(guidCounter);
150                 std::cout << "scan GUID: " << scanHeader.guid << std::endl;
151                 guidCounter++;
152
153                 std::cout << "Point count in scan header: " << scanHeader.pointCount << std::endl;
154
155                 // Read the point data
156                 e57::Data3DPointsData_t<double> buffers(scanHeader);
157
158                 e57::CompressedVectorReader dataReader =
159                     e57Reader.SetUpData3DPointsData(scanIndex, scanHeader.pointCount, buffers);
160                 dataReader.read();
161
162
163                 // Write the transformed scan data to the output file
164                 int64_t dataIndex = e57Writer.NewData3D(scanHeader);
165                 e57::CompressedVectorWriter dataWriter =
166                     e57Writer.SetUpData3DPointsData(dataIndex, scanHeader.pointCount, buffers);
167                 dataWriter.write(scanHeader.pointCount);
168                 if (dataIndex != scanHeader.pointCount) {
169                     std::cerr << "Error: Not all points were written!" << std::endl;
170                 }
171                 dataWriter.close();
172
173                 std::cout << "Processed scan index " << scanIndex << " from file: " << file.filepath
174                     << std::endl;
```

```
                // Close the writer after all scans are written
                e57Writer.Close();
                if (fileEntries.size() < 2) {
                    std::cout << "Transformed E57 point cloud written to " <<
                        outputPath.string() << std::endl;
                } else {
                    std::cout << "Merged and transformed E57 point cloud written to " <<
                        outputPath.string() << std::endl;
                }
            }
        }

    } catch (const e57::E57Exception& e) {
        std::cerr << e.what() << ": " << e.errorStr() << std::endl;
        return;
    } catch (const std::exception& e) {
        std::cerr << "Standard exception: " << e.what() << std::endl;
    } catch (...) {
        std::cerr << "Unknown exception occurred." << std::endl;
    }
}

```

7.2 Convert FLD to LAZ

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 13 17:11:13 2024

@author: stefa
"""
# from read_fld_optimized import read_fld
from read_fld import read_fld
import time
from PyQt5.QtWidgets import QApplication, QFileDialog
import numpy as np
import laspy
import os


def select_file():
    app = QApplication([])
    file_paths, _ = QFileDialog.getOpenFileName(None, "Select GPR-fld file")
    return file_paths

def replace_extension(file_path, ext):
    base_name, _ = os.path.splitext(file_path)
```

```
24      return base_name + ext
25
26  def create_xyz_amplitude_array(fld):
27      #fld_data, xpixels, ypixels, zpixels, pixelsize, y_coor, x_coor, pixelsize_z
28      rows, cols, z = fld[2], fld[1], fld[3]
29      matrix = fld[0]
30      pixel_size_xy = fld[4]
31      pixel_size_z = fld[7]
32      initial_y = fld[5]
33      initial_x = fld[6]
34      initial_z = 0 #for now!!!
35
36      # Find the positions of nonzero elements
37      nonzero_indices = np.nonzero(matrix)
38
39      # Extract the x, y coordinates from the nonzero indices
40      z_coords, y_coords, x_coords = nonzero_indices
41
42      # Extract intensity values at nonzero positions
43      intensity_values = matrix[nonzero_indices]
44
45      #transform intensity values to [0,255]:
46      intensity_values = (intensity_values/np.max(intensity_values))*255
47
48      #mirror values around 255/2:
49      intensity_values = 255-intensity_values
50
51      # compute coordinates:
52      x_coords = initial_x + x_coords*pixel_size_xy
53      y_coords = initial_y - y_coords*pixel_size_xy
54      z_coords = initial_z - z_coords*pixel_size_z
55
56      return x_coords, y_coords, z_coords, intensity_values
57
58
59  # file = sys.argv[1]
60  file = select_file()
61  start_time = time.time()
62  fld = read_fld(file)
63
64  file_new = replace_extension(file, ".laz")
65
66  # Create a Las
67  las = laspy.create(point_format=0)
68
69  las.x, las.y, las.z, las.intensity = create_xyz_amplitude_array(fld)
70
71  laspy.compression.LazBackend(0)
```

```
las.write(file_new)
print(f"Time taken: {time.time() - start_time:.2f} seconds")
```

7.2.1 Read FLD file

This function has been taken from Nau 2024.

```
import struct
import concurrent.futures
import numpy as np
import os
import matplotlib.pyplot as plt
import time

def read_fld(file_path):
    base_name = os.path.splitext(file_path)[0]
    output_npy = base_name + '.npy'

    with open(file_path, mode='rb') as f:
        file_content = f.read()

    xpixels, ypixels, zpixels, pixelsize, y_coor, x_coor, pixelsize_z = read_fld_header(file_content)

    data_start = read_fld_data_specs(file_content, zpixels)
    start_bits, stop_bits, number_of_values = get_start_stop_bits(file_content, zpixels, data_start)

    fld_data = read_fld_with_threads(file_content, xpixels, ypixels, zpixels, start_bits, stop_bits,
                                     number_of_values, output_npy)

    plt.imshow(fld_data[0], cmap='Greys', vmin=4, vmax=24)
    plt.show()

    return fld_data, xpixels, ypixels, zpixels, pixelsize, y_coor, x_coor, pixelsize_z

def read_fld_header(file_content):
    header_data = np.frombuffer(file_content[:40], dtype=np.dtype('>i4, >i4, >i4, >i4, >i4, >i4,
        >i4, >i4, >i4, >i4'))
    _, xpixels, ypixels, zpixels, x_start, x_end, _, _, z_start, z_end = header_data[0]
    _, xpixels, ypixels, zpixels, x_start, x_end, y_start, y_end, z_start, z_end = header_data[0]

    pixelsize = ((x_end-x_start) / xpixels) / 100
    #pixelsize = ((x_end-x_start) / xpixels) / 100
    pixelsize_z = ((z_end - z_start) / zpixels) / 100

    x_coor = np.frombuffer(file_content[198:206], dtype='>f8')[0]
    y_coor = np.frombuffer(file_content[214:222], dtype='>f8')[0]

    x_coor_end = np.frombuffer(file_content[206:214], dtype='>f8')[0]
    y_coor_end = np.frombuffer(file_content[222:230], dtype='>f8')[0]
```

```
43
44      pixelsize = round(((x_coor_end-x_coor) / xpixels), 2)
45
46      return xpixels, ypixels, zpixels, pixelsize, y_coor, x_coor, pixelsize_z
47
48
49
50  def read_fld_data_specs(file_content, z_size):
51      start_b = 246
52      start_num = start_b + z_size * 4 + 4
53      stop_num = start_num + 4
54
55      num_dt = np.frombuffer(file_content[start_num:stop_num], dtype=np.dtype('>i4'))[0]
56
57      start_num_velo = stop_num + (num_dt * 4 * 2) * 2
58      stop_num_velo = start_num_velo + 4
59
60      num_velo = np.frombuffer(file_content[start_num_velo:stop_num_velo], dtype=np.dtype('>i4'))[0]
61
62      data_start = stop_num_velo + num_velo * 2 * 4 + 4
63
64      return data_start
65  def get_start_stop_bits(file_content, number_of_layers, data_start):
66      start_bits = []
67      stop_bits = []
68      number_of_values = []
69
70      for layer_index in range(number_of_layers):
71          # Determine the size of the layer
72          number_of_values_layer = struct.unpack('q', file_content[data_start:data_start + 8])[0]
73          number_of_values.append(number_of_values_layer)
74
75          # Determine the end position of the layer data
76          data_stop = data_start + 16 + number_of_values_layer * 2
77
78          start_bits.append(data_start)
79          stop_bits.append(data_stop)
80
81          # Update the start position for the next layer
82          data_start = data_stop
83
84      return start_bits, stop_bits, number_of_values
85
86  def process_layer(layer_index, file_content, x_size, y_size, start_bits, stop_bits):
87      data_start = start_bits[layer_index]
88      data_stop = stop_bits[layer_index]
89
90      min1, max1 = struct.unpack('>f', file_content[data_start + 8:data_start + 12])[0], \
91          struct.unpack('>f', file_content[data_start + 12:data_start + 16])[0]
92
```

```
93      data_values = np.frombuffer(file_content[data_start + 16:data_stop], dtype='>i2')
94      data_values = np.where(data_values > 0, min1 + (data_values - 1) *
95          ((max1 - min1) / 32760), data_values)
96      data_unpacked = []
97
98      # start_time = time.time()
99      for value in data_values:
100         if value < 0 and value.is_integer():
101             data_unpacked.extend([0] * int(abs(value)))
102         else:
103             data_unpacked.append(value)
104     # print(f"Time taken: {time.time() - start_time:.2f} seconds")
105     data_unpacked = np.asarray(data_unpacked, dtype=float)
106
107     data_unpacked = data_unpacked.reshape(y_size, x_size)
108     # print(f"Time taken: {time.time() - start_time:.2f} seconds")
109     return layer_index, data_unpacked
110
111 def read_fld_with_threads(file_content, x_size, y_size, number_of_layers,
112     start_bits, stop_bits, number_of_values, output_npy):
113     data = np.empty((number_of_layers, y_size, x_size), dtype=np.float32)
114
115     with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
116         futures = [executor.submit(process_layer, layer_index, file_content, x_size, y_size,
117             start_bits, stop_bits) for layer_index in range(number_of_layers)]
118
119         for future in concurrent.futures.as_completed(futures):
120             layer_index, data_unpacked = future.result()
121             data[layer_index, :, :] = data_unpacked
122
123     data = data.transpose((0, 1, 2))
124     data = np.flip(data, axis=1)
125
126     return data
```