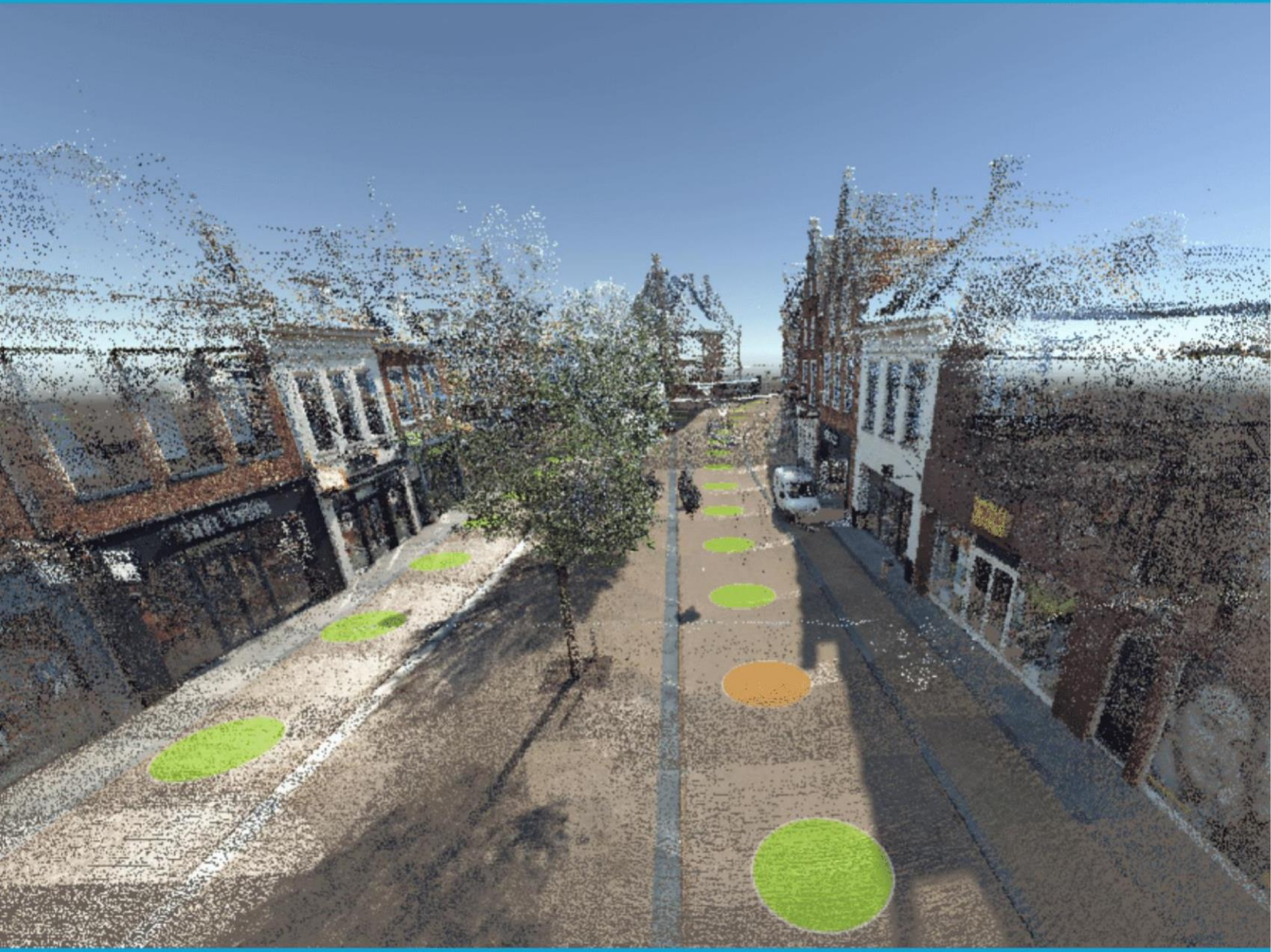
# Analyzing deformation of buildings using LiDAR point clouds obtained by a Mobile Laser Scanning System

K.R. Bos



Geoscience & Remote Sensing

Delft University of Technology

# Analyzing deformation of buildings using LiDAR point clouds obtained by a Mobile Laser Scanning System

By

K.R. Bos

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 27th June, 2023.

| | | |
|---|---|---|
| Student number: | 4589703 | |
| Project duration: | October 1, 2022 - June 27, 2023 | |
| Thesis committee: | Dr. R.C. Lindenbergh | Optical and Laser Remote Sensing group, TU Delft, chair |
| | Ir. P. Joosten | Cyclomedia Technology B.V. |
| | Dr.ir. F.J. van Leijen | Geokinematics and Radar Remote Sensing, TU Delft |

Cover: Street point cloud of a part of the Voorstraat in Franeker captured by Cyclomedia

# Abstract

Unequal deformation of the soil can cause deformation or damage to buildings, like tilted facades or cracks in walls. This research investigates how deformation of a building can be analyzed using Light Detection And Ranging (LiDAR) data. Cyclomedia captures LiDAR data yearly in the Netherlands making it possible to analyze either one or multiple epochs of data. If deformation is monitored, failure can be predicted, and repairs can be performed in time.

A subsiding area is found using the Dutch surface motion map. The data points of a single building are fetched from the LiDAR point cloud to analyze the following types of deformation: a difference in the torsion and tilt angle of the facade between two epochs of data; the tilt angle of the facade for a single epoch of data; and local deformation patterns on the facade. The data is segmented before computing the angles and analyzing the local deformation patterns. During segmentation, points that do not correspond to the facade (like a sunshade or windows) are removed. After segmentation, the facade is modeled by fitting a plane using Principal Component Analysis (PCA). The plane parameters $(A, B, C, D)$ are used to determine the torsion and tilt angles. If the torsion or tilt angles are large (above a degree), it is expected the facade has deformed. Local deformation patterns can be analyzed by using a raster containing the distance of the segmented points with an accuracy of 8 centimeters. Besides the point clouds, Cyclomedia also captures $360^o$ panoramic images (Cycloramas). These Cycloramas can be used to explain patterns that are visible in the rasters. For example, objects in front of the building like a bench or a sunshade.

This research uses Random Sample Consensus (RANSAC) to segment the data. From the results, it can be concluded RANSAC is not very predictable because random points are taken as input. So, points corresponding to the facade can be removed instead of points corresponding to windows or doors. Therefore, it is recommended to use another segmentation method for future research instead of RANSAC. Machine learning could be a good alternative to remove objects like windows and other unwanted points from the data.

# Contents

# Acronyms

| | |
|---|---|
| **BLUE** | Best Linear Unbiased Estimator<br>*BLUE is an estimator which is unbiased and linear with minimum variance.* |
| **CRS** | Coordinate Reference System<br>*Using a CRS, locations on the Earth's surface can be defined as coordinates.* |
| **DCR** | Digital Cyclorama Recording<br>*DCR is a recording system of Cyclomedia used to capture data.* |
| **DCR10** | Digital Cyclorama Recording system Version 10<br>*DCR10 is the 10th generation digital recording system.* |
| **GNSS** | Global Navigation Satellite System<br>*GNSS is a term used to describe all satellite positioning systems of the world.* |
| **HD** | High Definition<br>*HD has a high resolution making it possible to see more details.* |
| **IMU** | Inertial Measurement Unit<br>*IMU is a device measuring the acceleration and angular velocity making the determination of the orientation and position better.* |
| **InSAR** | Interferometric Synthetic Aperture Radar<br>*InSAR uses RADAR to determine the relative deformation of the surface.* |
| **LiDAR** | Light Detection And Ranging<br>*LiDAR determines the distance to an object.* |
| **MLS** | Mobile Laser Scanning<br>*MLS is when a laser scanning system is attached to a moving platform.* |
| **OLS** | Ordinary Least Squares<br>*OLS minimizes the sum of squared residuals parallel to an axis to estimate the parameters of a linear regression model.* |
| **PCA** | Principal Component Analysis<br>*PCA reduces the dimensionality of the data. This makes the data easier to interpret.* |
| **PNG** | Portable Network Graphic<br>*PNG is a file format for raster-graphics.* |
| **PSI** | Persistent Scatterer Interferometry<br>*PSI is a technique to measure and monitor displacements of the surface of the Earth.* |
| **RADAR** | Radio Detection and Ranging<br>*RADAR is used to determine the range towards an object by using radio frequency waves.* |
| **RAM** | Random Access Memory<br>*RAM is the memory of a computer.* |
| **RANSAC** | Random Sample Consensus<br>*RANSAC is a method to estimate parameters of a dataset containing outliers.* |
| **RGB** | Red Green Blue<br>*RGB is a color model to produce colors based on the primary colors red, green and blue.* |
| **RMSE** | Root Mean Squared Error<br>*RMSE measures the difference between the predicted and the actual values.* |
| **RPCA** | Robust Principal Component Analysis<br>*RPCA is a modification of PCA which is more robust to i.e. outliers.* |
| **SAR** | Synthetic Aperture Radar<br>*SAR is a technique to produce RADAR images.* |
| **SVD** | Singular Value Decomposition<br>*SVD decomposes a matrix in three different components.* |
| **TLS** | Total Least Squares<br>*TLS minimizes the sum of squared residuals in orthogonal direction to estimate the parameters of a linear regression model.* |

# 1. Introduction

In this thesis, deformation of buildings is analyzed using terrestrial LiDAR data of streets. The problem description is given in Section 1.1 and the research questions are outlined in Section 1.2. Section 1.3 gives the thesis reading guide.

## 1.1. Problem description

In the Netherlands, various causes can lead to deformation of the surface and the objects (in this thesis buildings) on the surface. Possible causes include a change in groundwater level, gas extraction, and construction work. Deforming areas can be found by using a combination of satellite-based Interferometric Synthetic Aperture Radar (InSAR) data and news articles about subsidence or damage of buildings. Some examples of deformation in the Netherlands are given in Subsection 1.1.1. The research objective is described in Subsection 1.1.2

### 1.1.1. Examples of deformation in the Netherlands

In Limburg, coal was extracted from the ground. Groundwater pumping took place to keep the mines dry. Coal mining stopped when a gas field was discovered in Groningen because extracting gas was easier than mining coal (Voncken, 2019). After closing the mines, groundwater pumping stopped stepwise (Sigaran-Loria, 2019) causing a rise in the groundwater level filling the mines. The water in the mines caused instability in the mine shafts because it caused rust on the support bars and the ground was washed away, leading to shaft failure. This caused sinkholes and deformation at some locations. ”Staatstoezicht op de Mijnen” monitors deformation using InSAR. However, it is difficult to use InSAR to monitor these deformation effects because the deformation is too local and cannot be easily detected using InSAR data (SodM, 2021). So, another method might be better.

In Delft, the groundwater level rose because the yeast factory that used groundwater for cooling was closed and taken over by DSM (a biotechnology company). DSM did not need all this water, so a schedule was made to gradually stop the groundwater pumping. However, between 2019 and 2020, the ground level began to increase causing buildings to deform. Therefore, it was decided to keep the amount of groundwater pumped out of the soil the same to prevent the surface from rising too much (Hoes, 2021).

In Amsterdam, the groundwater level dropped in peat soils, resulting in ”dancing houses” (Gemeente Amsterdam, 2020a) shown in Figure 1.1. Due to a drop in groundwater level in peat or clay soils, settlement can occur due to soil compaction. This subsidence is partly permanent and partly reversible. The irreversible effect is mainly visible in peat soil that consists of partially decayed vegetation. If the groundwater level drops, the peat above the water level comes into contact with oxygen. Then bacteria cause oxidation of the peat, which makes the peat disappear, resulting in subsidence of the ground level (Hoogland, 2012).

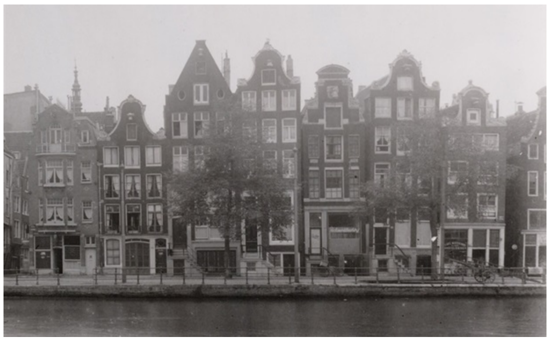


**Figure 1.1.:** *The ”dancing houses” at Amstel 100 to 112 in Amsterdam (Gemeente Amsterdam, 2020b).*

In addition to the drop in groundwater level, the Noord-Zuidlijn was constructed in Amsterdam. Some buildings subsided on the Vijzelgracht (Kuipers, 2021) during construction. So, buildings can also deform due to construction work. The construction of a tunnel can lead to the deformation of buildings due to vibrations caused by construction or soil removal.

For the construction of the railway tunnel in Delft, the Bagijnetoren blocked the construction area. Therefore, it was needed to temporarily move the tower (Delft op zondag, 2011) which might have caused a deformation when the tower was placed back at the original spot.

In Groningen, deformation occurs due to gas extraction causing subsidence of the soil. When subsidence in an area is irregular, it can cause earthquakes. These earthquakes are at a shallower depth than normal earthquakes and have a higher chance of damaging buildings (Mulder, 2018).

According to news articles, many people in Groningen have damaged homes. For example, in Appingedam, where beams were needed to support houses due to sagging walls and cracked ceilings. Sometimes buildings can be reinforced, but some buildings are too damaged and must be demolished and rebuilt (Reed, 2019).

### 1.1.2. Research objective

Deformation can cause damage to buildings, like cracks in ceilings or sagging walls. To prevent potential damage and accidents, it is important to monitor deformation making it possible to predict failure and repair damage in time. If deformation is monitored, local authorities can schedule when to raise a subsiding pavement, and buildings can be reinforced when they are deforming too much. It is also possible to search for the cause of the deformations and try to stop the deformation from happening. For example, when deformation occurs due to a change in groundwater level.

The main objective of this research is to find out if it is possible to analyze deformation of buildings using data from Cyclomedia (the data is discussed in Section 3.1). Cyclomedia is a company that owns cars with a Digital Cyclorama Recording (DCR) system on the roof. The DCR system consists of a camera system, a LiDAR scanner, an Inertial Measurement Unit (IMU), and a Global Navigation Satellite System (GNSS). So using the data of the system, colored point clouds can be produced. The data from Cyclomedia is used, because it is already available and new data is collected yearly. So, it is nice to have more applications using this dataset. If deformation of buildings can be analyzed, it is also possible to extend the research to investigate whether there is a deformation of the pavement or asphalt due to subsidence or uplifting of the soil.

To analyze deformation, there is first searched for a location where deformation occurs (explained in Section 3.3). In this area, measurements from different epochs will be considered. Different buildings in the chosen location will be compared. The buildings can be modeled by estimating a plane through the facades of the buildings. The planes can be compared by determining the normal angles (explained in Subsection 4.3.1). The data can also be compared by creating a raster of the data containing the distances (explained in Subsection 4.3.2). At the end of this research, recommendations will be written on how to improve the method that analyzes deformation with the LiDAR data of Cyclomedia.

## 1.2. Research questions

The main question for this project is defined as:

*How can street LiDAR (point cloud) and Cyclorama (images) data from the digital Cyclomedia recording system be used to analyze relative deformations of buildings?*

The project will be further split into the following subquestions:

1. How to find suitable areas where deformation occurs for which a method can be developed to analyze deformation?
2. How can relative deformation be estimated using data from multiple epochs?

3. What deformation patterns can be determined and with what confidence?
4. How can the different types of Cyclomedia data be used to analyze deformation?
5. How to evaluate the deformation measured using the planes or rasters?

## 1.3. Reading guide

In Chapter 2, background information is given which is useful for this research. First, different types of deformation are described. Second, two techniques (LiDAR and InSAR) are compared that can be used to analyze the deformation of a building. Third, there is explained how the data can be transformed from the original coordinate system to a building-based coordinate system. Fourth, different plane fitting methods (OLS, TLS and PCA) are discussed that can be used to create a model of the facade. Finally, the RANSAC algorithm is discussed which is used for the segmentation of the data. Chapter 3 describes the datasets (LiDAR and InSAR) and the study areas (Franeker and Amsterdam). Chapter 4 describes the methodology which is mainly developed using the data of the SNS building in Franeker. Chapter 5 describes the results of the study area in Franeker and Chapter 6 describes the results of the study area in Amsterdam. In Chapter 7 the results are discussed, and Chapter 8 gives the conclusions and recommendations of this research.

# 2. Background

Information is given about different types of deformation in Section 2.1. In Section 2.2, InSAR and LiDAR are compared. A description is given of how to transform the data from the original coordinate system to a building-based system in section 2.3. In Section 2.4, different plane fitting algorithms are described to model the facade of a building, and Section 2.5 describes the Random Sample Consensus (RANSAC) algorithm used to segment the data.

## 2.1. Different types of building deformation

A building has six degrees of freedom in which deformation can take place (Guéguen, 2021). Something is called deformation when the size or shape of the object (in this case the facade) changes. In this thesis, the term deformation is also used for rotation or translation of the object[1] (Truesdell, 2004). As shown in Figure 2.1, there are three translational degrees of freedom ($u_H, u_L, u_W$) and three rotational degrees of freedom ($\theta_H, \theta_L, \theta_W$). A building can move in all three directions (height, width, and length), so different types of deformation exist.

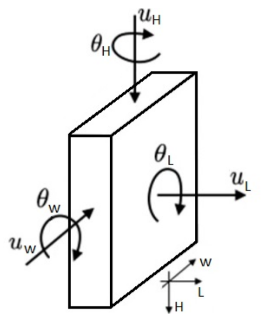


**Figure 2.1.:** *The six degrees of freedom of the facade, where H is height, W is width and L is length.*

Six types of deformation are visualized in Figure 2.2. The three types on the top row are deformations of the whole facade, which are: vertical deformation (in $u_H$ direction, see Subsection 2.1.1), tilt (in $\theta_W$ direction, see Subsection 2.1.2), and torsion (in $\theta_H$ direction, see Subsection 2.1.3). The three types of the bottom row are local deformations, which are: cracks (see Subsection 2.1.4), bulging (see Subsection 2.1.5), and bowing (see Subsection 2.1.6).

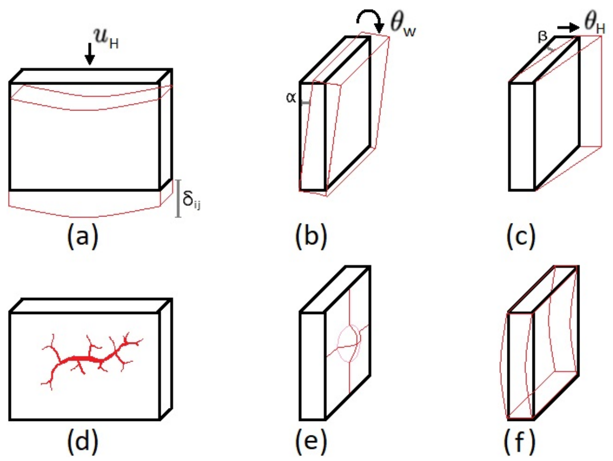


**Figure 2.2.:** *Different types of deformation of a building. (a) Vertical deformation. (b) Tilt deformation. (c) Torsional deformation. (d) Cracks. (e) Bulge. (f) Bow. (Feng, 2021)*

[1] In other research, translation and/or rotation of an object is often called displacement when the size or the shape of the object does not change. So, in case of displacement, there is only a change in position.

### 2.1.1. Vertical deformation

Vertical deformation occurs when (a part of) the foundation of a building is moving in the vertical direction. The building can move upward when the soil below the building expands, which is called heave. This can occur due to rain because then the soil swells up due to an increase in groundwater. The building can also move downward when the soil compacts or shrinks, which is known as settlement. This can occur when the amount of water in the soil decreases (Yao, 2021b). An example of heave and settlement due to rainfall is shown in Figure 2.3. In addition to a decrease in groundwater level, settlement can also occur due to placing a heavy load on the soil, decreasing the space between the ground particles and compressing the soil (Shober, 2013).

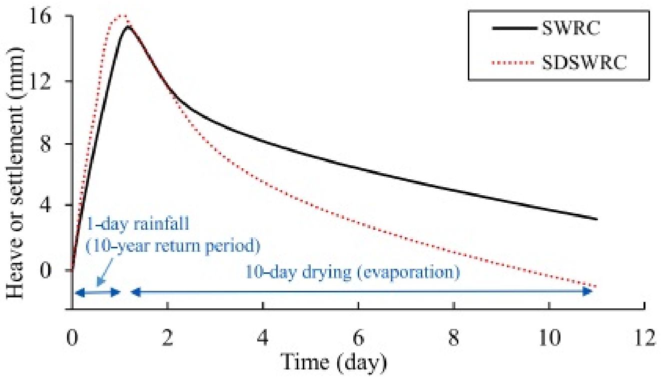


**Figure 2.3.:** *Plot of heave and settlement of soil due to rainfall (intensity of 381 mm/d) and drying. SWRC stands for soil water retention curve and SDSWRC stands for stress-dependent soil water retention curve (Yao, 2021a).*

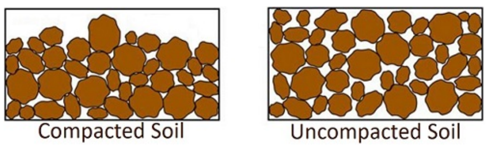


**Figure 2.4.:** *From left to right, visualization of compacted and uncompacted soil. The distance between the pores is larger for the uncompacted soil (Denny, n.d.).*

There are different ways in which soil can settle: uniform settlement and differential settlement. The building settles uniformly when the building subsides with the same amount at each location of the building. In the case of uniform settlement, there will not be a change in the normal angle of the plane of the building. However, it is more likely the building settles non-uniform, also known as differential settlement $\delta_{ij}$. In the case of differential settlement, the amount of settlement is not equally distributed along the foundation of the building. This can lead to cracks or cause problems by opening and closing windows and doors (Ferlisi, 2019). So, for this research interest lies in differential settlement because it is more likely that it causes deformation of the facade.

### 2.1.2. Tilt

Tilt is also known as tipping settlement, inclination, or rocking. In the case of tilt, a part of the soil is weak. Due to the weak soil, the building settles in one direction causing the top of the building to either move forward or backward (so in the direction of $\theta_W$ shown in Figure 2.1) depending on the part of the soil that is moving down. If the front of the building is moving downward, the building will tilt forward (positive tilt angle $\alpha$). If the back of the building is moving down, the building will tilt backward (negative tilt angle $\alpha$). Tilt does not necessarily cause cracks in the facade. However, when the tilt angle becomes too high, the building can overturn leading to failure of the building.

An example of a tilted building is the Tower of Pisa. In 2001, the tilt angle of this tower was 5.6 degrees (Zhou, 2001), but now the angle is approximately 4 degrees (CNE, 2022), see Figure 2.5a. In addition to the Pisa Tower, there are also other leaning towers. The tower of the church in Suurhusen has an angle of approximately 5.2 degrees (CNE, 2022), see Figure 2.5b. The Gau-Weinheim tower has an angle of approximately 5.4 degrees (CNE, 2022), see Figure 2.5c.

**(a)** *Tower of Pisa in Italy with a tilt angle of $4^o$ (Lintula, 2006).*

**(b)** *Tower of Suurhusen in Germany with a tilt angle of $5.2^o$ (Heymann, 2004).*

**(c)** *Gau-Weinheim tower in Germany with a tilt angle of $5.4^o$ (Roessler, 2022).*

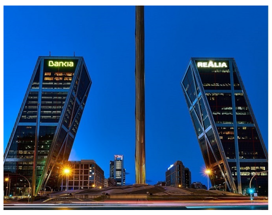

**(d)** *Gate of Europe in Spain with a tilt angle of $15^o$ (Deensel, 2017).*

**Figure 2.5.:** *Examples of towers which are tilted with a certain degree.*

Besides these buildings, there exist inclined buildings. These buildings are intentionally built under an angle. For example, the Gate of Europe, also known as the KIO Towers, which are located in Madrid in Spain. These buildings are built under an angle of 15 degrees, see Figure 2.5d.

So, it depends on the building and how large the tilt angle can be. If a building is intentionally constructed under an angle, the angle can be much larger before failure in comparison to a building that was meant to have a tilt angle of zero. However, when the angle between two years increases a lot (for example with 1 degree), it is a good idea to investigate the building and check what is happening at this location.

## 2.1.3. Torsion

In the case of torsion, one side (left or right) of the facade is moving and the other part is either motionless or moving at a different rate. Torsion in a building can occur due to an eccentric load (Oosterhoff, 2008), as shown in Figure 2.6. The load is a certain distance (indicated with $e$) from the central axis of the construction, so the load is asymmetric. This asymmetrical load causes torsion. During torsion, the building is twisted or turned due to an applied torque. Torque is a force that results in rotation of the building around the height axis ($u_H$ in Figure 2.1). Torque can for example occur due to the force of the wind (Isyumov, 1983) or due to an earthquake (Tabatabaei, 2011).

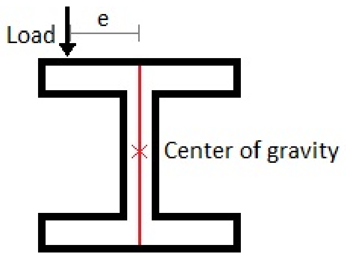


**Figure 2.6.:** *Example of an eccentric load on a steel profile.*

Due to excessive torsion, a building can fail. This can happen when the torsional moment capacity of a structure element is reached or when the lateral deflection limit of a building is reached (Gokdemir, 2013). An example of failure due to torsion is shown in Figure 2.7.

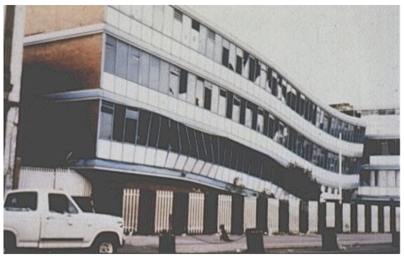
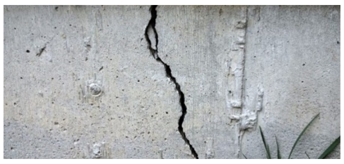


**Figure 2.7.:** *A building in Mexico City which failed due to torsion (Earthquake Engineering ANNEXES, 2007).*

### 2.1.4. Cracks

Facades can be made of different materials, for example, bricks, concrete, or plaster. In all these types of facades, cracks are a common defect. In Figure 2.8 some examples of cracks are visible. A crack is an unnaturally elongated gap, so a crack splits the wall.

**(a)** *Crack in a brick wall (Dotson, n.d.).*


**(b)** *Crack in a concrete wall (Kumar, 2021).*

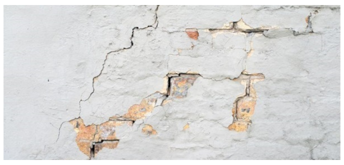

**(c)** *Crack in plaster wall (Coating, 2019).*

**Figure 2.8.:** *Examples of cracks in different types of facades.*

There are different reasons why cracks could occur, for instance, due to constructive causes or due to degradation agents (Bauer, 2018). Constructive causes depend on the construction process and the material quality. For example, drying shrinkage can occur when the mortar, concrete, or plaster is hardening due to the loss of water. Drying shrinkage cracks can occur even four years after the construction of a building (Day, 2003). Besides shrinkage, cracks can also occur due to poor workmanship or a bad structural design. The workmanship is poor when the constructors do not mix the materials correctly. A structural design can be bad when the strength and stability calculations are not made correctly or when not all environmental aspects, such as the soil type, are considered when making the design (SmartBricks, 2018).

Degradation agents are causes due to solar radiation, temperature, and rain. A temperature change can cause thermal movement because building materials can expand or contract depending on the temperature. Large daily differences in temperature have a higher chance of damaging a building because they are more rapid (SmartBricks, 2018). Rain makes the facade wet, and the humidification and drying of the facade can cause expansion of the cementitious materials. These movements of the materials increase the chance of cracking because the materials have restricted movement. The movement is restricted because the wall is attached to the base or support layer (Bauer, 2018).

Cracks can also occur due to vegetation because the roots of plants can grow on the wall. An example of a plant is ivy, which can grow inside the cracks damaging the mortar. When pulling ivy from the wall, the mortar or other materials can come with it causing more damage (Viles, 2011).

### 2.1.5. Bulging facade

Bulge is a local deformation of the facade (Diaz, 2020). Sometimes, bulge can also be intentional for architectural reasons, as shown in Figure 2.9. In the case of bulging, a part of the wall swells and has an outward displacement. This displacement can occur due to different reasons. For example, due to water pressure in the soil around the foundation or due to water seeping behind the wall.

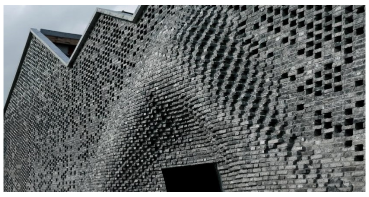


**Figure 2.9.:** *Bulging building in Shanghai (Griffiths, 2016).*

Water pressure, also known as hydrostatic pressure, occurs when water gets absorbed by the soil around a building. The water places pressure and weight against the foundation wall and this can eventually lead to bulge of the foundation of a structure (Foundations & Waterproofing, 2022). Besides water pressure affecting the foundation, water can seep through the mortar joints of outdoor brick walls. When the water is behind the wall, the moisture swells, causing the bricks to bulge outwards.

This mainly happens during winter moisture in combination with freezing and thawing. In the warmer seasons, water evaporates through the mortar joints, making the bulges more prominent and noticeable (Lu, 2018).

### 2.1.6. Bowing facade

Bow occurs when the center of a facade is pushed either inward or outward, and the top and bottom of the facade do not move, as shown in Figure 2.10. It can occur due to different reasons. For example, due to expansive soil, hydrostatic pressure, frost-heaving and lateral pressure from heavy objects (Petty, 2021), as shown in Figure 2.11. Besides these reasons, bow can also occur on a new brick wall when concrete framing and brick veneer[2] are combined. Over time, concrete shrinks and brick expands causing the wall to bow. This can be prevented by using relief joints when using a combination of different building materials.

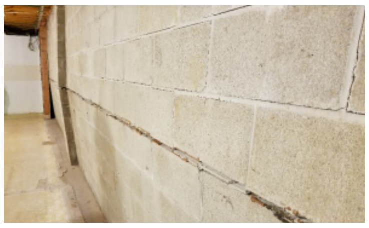


**Figure 2.10.:** *Example of a bowing wall (Kater, 2019).*

Expansive soil is for example clay. This soil type can swell or shrink based on the moisture content. During rain, the soil swells. This causes the soil to expand and push against the wall. When the sun shines, water evaporates allowing the wall to fall back outward. The change between swelling and shrinking can cause the foundation walls to bow. Hydrostatic pressure is like expansive soil, but instead of soil, water pushes against the wall. This mostly occurs in areas with a high water table. Frost-heaving occurs when the soil freezes. This can increase the volume of the soil, causing the soil to push against the wall. Besides a volume increase, the soil can rise when it freezes. During thaw, the soil settles and this movement of the soil upwards and downwards causes pressure on the foundation walls. Lateral pressure occurs when a heavy object is placed on the soil. A possible heavy object is a vehicle. The weight of this vehicle pushes down the soil, causing pressure against the foundation.

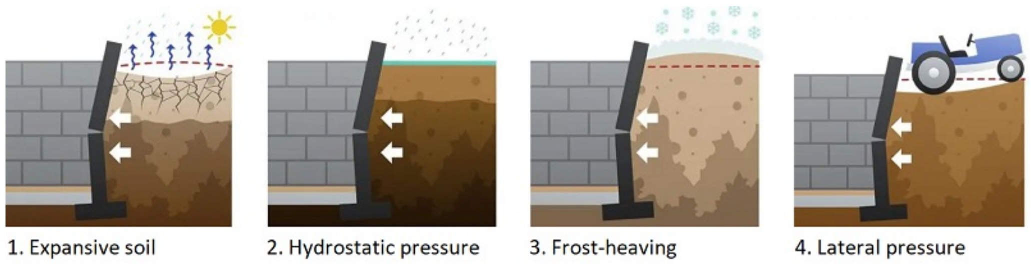


**Figure 2.11.:** *Causes of bowing walls (Werner, 2022).*

## 2.2. Two techniques to analyze deformation of a building

Deformation can be analyzed by comparing at least two sets of measurements spread over time. It is also possible to analyze deformation by comparing a single measurement with an ideal plane. The ideal plane is an estimation of the facade just after construction, so there is no torsion, tilt, or local deformation present. So, the ideal plane is assumed to be a plane perpendicular to the ground plane[3] as shown in Figure 2.12.

[2]A brick veneer wall is not built to support the building. The building is supported by a backup system. The brick veneer wall is attached to the backup system and its main purpose is aesthetics (Martins, 2017).

[3]The normal vector of the ground plane is assumed to be (0,0,1) and the normal vector of the ideal plane is assumed to be either (0,1,0) or (1,0,0).

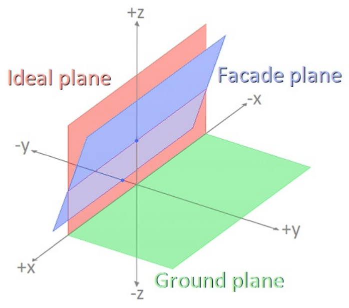


**Figure 2.12.:** *Example of an ideal plane (red) perpendicular to the ground plane (green) and an example of a plane going through a tilted facade (blue) (CPM, 2014).*

Multiple geomatic methods can be used to analyze deformation, for example, taking measurements with a total station (measuring angles and distances between certain points). However, taking measurements with a total station is very time-consuming (Vacca, 2016), because it takes a while and the data is not already available for all locations. Deformation can also be analyzed using data that is already captured for other purposes, for example, data obtained with InSAR or LiDAR. These two techniques are compared in Table 2.1.

**Table 2.1.:** *The differences between InSAR and LiDAR.*

| InSAR | LiDAR |
|---|---|
| Data collected using satellites | Data collected using e.g., a vehicle, a tripod, a plane, or a drone |
| Measurement time and location are fixed | Measurement time and location are adaptable |
| Covers a large area in one scan | Covers a smaller area in one scan |
| Density is 1.0 * $10^5$ points per $km^2$ (SkyGeo, 2022) | Density is around 1.8 * $10^9$ points per $km^2$ (while using the data of Cyclomedia[4]) |
| Effective in all weather conditions (Stilla, 2003) | Rain, fog, frost, or ice can disturb measurements (Stilla, 2003) |
| Difficult to determine the exact location of a measurement point | Use GNSS and IMU to determine the location of a measurement point |
| Prime observable is the displacement of a specific point (scatterer) between two or more epochs | Prime observable is the three-dimensional position of a point. Each measurement contains different points. So, a model, grid, or interpolation is needed to compare epochs |

In this thesis, the focus lies on finding a study area (see Subsection 2.2.1) and analyzing deformation of the facade of a building (see Subsection 2.2.2). There is determined which technique, either InSAR or LiDAR, is suitable for a certain application.

### 2.2.1. Find a study area with InSAR

In the case of finding a study area, it is interesting to know at which location deformation occurs. This location can be determined globally using a low density of points because it is just an indication of an area that could contain deformation. However, it is useful to have data that is already displayed on a map, so time will not be wasted by preprocessing data of areas that have a low chance of finding deformation.

Looking at the advantages and disadvantages of the two techniques described in Table 2.1, it can be concluded preference lies in using InSAR instead of LiDAR for selecting an area of interest. InSAR data is already visualized (Dutch surface motion map), so this data can easily be used to find an interesting location. For InSAR it is hard to determine the exact location of a scatterer. However, the exact position is not needed because InSAR is only used to have a higher chance of finding a building that is deforming.

[4]The density is calculated using a measurement area of 50 by 50 m containing approximately 4.5 million points.

### 2.2.2. Analyze deformation of a building with LiDAR

In the case of analyzing deformation of the facade of a building, it is useful to know the location of the measurement points. Knowing the location of the points makes it possible to determine which specific building is deforming. It is also convenient to have a high density of points. If the density of the point cloud is higher, the distance between the points will be smaller, and then the location of the deformation can be determined more accurately.

Looking at the information described in Table 2.1, it can be concluded preference lies in using LiDAR instead of InSAR for analyzing deformation of a specific building because while using LiDAR the location of deformation can be determined. Thereby, LiDAR has a higher point density in comparison to InSAR. The Dutch motion map (created with InSAR data) only contains points located on hard surfaces like rocks, buildings, or other infrastructure. These hard surfaces have an undisturbed Radio Detection and Ranging (RADAR) signature and a similar reflection characteristic which makes it possible to compare data from different epochs (SkyGeo, 2022). While using LiDAR, a laser signal is reflected from objects resulting in the data points. Besides points from hard objects, LiDAR also measures things like vegetation resulting in a higher point density.

Two types of LiDAR could be used: Terrestrial and Aerial. When using an airborne (or aerial) system, the resulting point cloud will mainly contain points located on the roof of a building instead of the facade making aerial LiDAR less suitable to analyze horizontal deformation (Young, 2010). Using terrestrial LiDAR it is much easier to capture points on the facade. Therefore, the focus lies on analyzing deformation using terrestrial laser scanning.

Some research is already performed using a terrestrial laser scanner, for example, to measure deformation of a wooden or concrete beam. Targets were placed on these beams and then a load was applied to get some deformation. This resulted in vertical deformation of the beams in a couple of hours (Gordon, 2005). The research with the beams is slightly different in comparison with deformation of facades. Targets are not placed on the facades and deformation happens over a longer period instead of a couple of hours. Thereby, deformation can occur in different directions, for example, horizontal or vertical instead of only vertical.

A terrestrial laser scanner can also be used to determine the roughness of a surface when scanning with high accuracy (in the order of millimeters (Corso, 2017)). Therefore, the type of material can be identified while using Red Green Blue (RGB) images in combination with images of lighting techniques. By comparing the two images, it might be possible to check if there is stone decay. So, this is on a smaller scale than deformation of the whole facade for which an accuracy in the order of centimeters is already enough.

## 2.3. Transformation of the coordinate system

In this thesis, the original Cartesian coordinate system $(x, y, z)$ is transformed to a building-based coordinate system $(W, L, H)$[5]. In the original coordinate system, the $y$-coordinate points North, the $x$-coordinate points East, and the $z$-coordinate points upwards. The building-based coordinate system depends on the orientation of the building with respect to the original coordinate system. A single building has a certain width ($W$), length ($L$), and height ($H$). So, for one single building in the original coordinate system $(x, y, z)$, the width could be in the direction of the $x$-coordinate, but for another building, the width could be in the direction of the $y$-coordinate, as shown in Figure 2.13. However, the origin of the coordinate system and the $z$-coordinate (called height ($H$) in the building-based system) are the same for the two coordinate systems. The orientation of the building-based coordinate system with respect to the original coordinate system is shown in Figure 2.14.

The 4x4 transformation matrix $\mathbf{M}$ consists of two parts: the rotation matrix $\mathbf{R}_\theta$ and the translation matrix $\mathbf{T}$. The building-based coordinate system can be obtained by a rotation around the $z$-axis of the original coordinate system, or by a rotation around the $H$-axis of the building-based coordinate system because the $z$-axis and the $H$-axis are aligned.

[5]How to compute the rotation angle to get from the original coordinate system to the building-based coordinate system is explained in Subsection 4.1.4

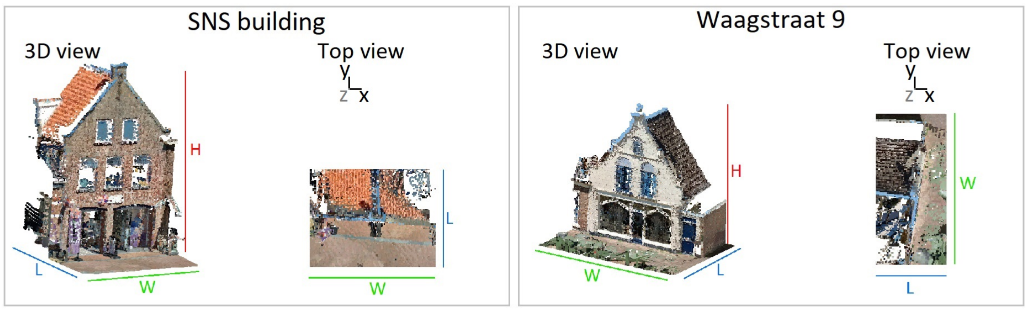


**Figure 2.13.:** *Left, a building with the width in x direction and right a building with the width of the building in y direction.*

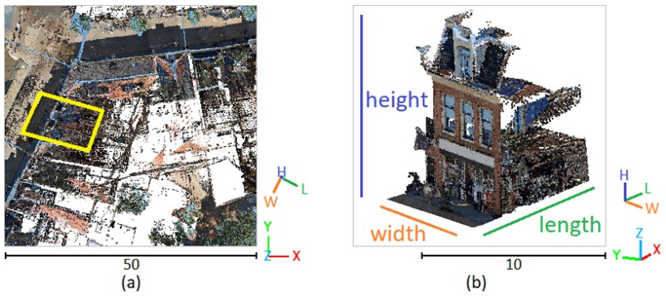


**Figure 2.14.:** *Orientation of the building-based coordinate system (W,L,H) with respect to the original coordinate system (x, y, z). Figure (a): Top view of the dataset. The building of Figure (b) is highlighted in yellow. Figure (b): three-dimensional visualization of a building containing the width, length, and height components of the building.*

This can be done using the following matrix (Popescu, 2014) in which $\theta_H$ represents the rotation angle around the $H-$axis of the building:

$$\mathbf{R}_{\theta_H} = \mathbf{R}_z(\theta_H) = \mathbf{R}_H(\theta_H) = \begin{bmatrix} \cos(\theta_H) & -\sin(\theta_H) & 0 \\ \sin(\theta_H) & \cos(\theta_H) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

When the rotation matrix is applied, the data points are rotated around the origin (0,0,0). Due to the rotation matrix, the center of the bounding box of the building chances (this also changes the mean of the width and length). So, after rotating the data, the data points are translated to back the original center of the bounding box (for example the yellow dot in Figure 2.15). This translation can be performed using the translation matrix.

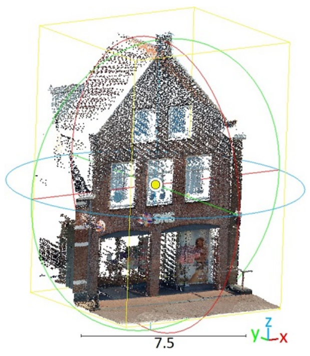


**Figure 2.15.:** *Bounding box of the data of a single building. The center of the points is displayed with a yellow dot.*

The data is rotated around the $z$-axis (or $H$-axis) to transform the $x$ and $y$-coordinates to the building-based data points $W$ and $L$. The translation in the $z$ direction is assumed to be zero because the data points are rotated around this axis. So, the translation matrix will be:

$$\mathbf{T}(t) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

The original data points are $p = (x_i, y_i, z_i)_{i=0...n-1}$ and $\mathbf{R}_{\theta_H}$ is the rotation matrix around the $H-$axis of the building. So, the rotated data $(W_i, L_i, H_i)_{i=0...n-1}$ is matrix $\mathbf{R}_{\theta_H}$ multiplied by $p$. So $t_x$ and $t_y$ are:

- $t_x$ is the mean of the rotated data point $(W_i)_{i=0...n-1}$, subtracted by the mean of $(x_i)_{i=0...n-1}$
- $t_y$ is the mean of the rotated data point $(L_i)_{i=0...n-1}$, subtracted by the mean of $(y_i)_{i=0...n-1}$

So, the coordinate system of a single building can be obtained by multiplying the original coordinates with the following transformation matrix (combination of the rotation and translation matrix):

$$\mathbf{M} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & t_x \\ \sin(\theta) & \cos(\theta) & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

## 2.4. Different plane fitting methods

A facade can be modeled by fitting a plane through the data points of the facade of a building. In other research, the coordinates of data points are described using $x, y, z$ (the original coordinate system (EPSG:28992)). However, this research uses the building-based coordinate system (discussed in Section 2.3). So, $x, y, z$ in the plane equation are replaced with $W, L, H$.

A plane through the facade can be fitted by estimating the plane parameters $(A, B, C, D)$ to get the equation of the plane. A plane is defined as:

$$AW + BL + CH + D = 0 \tag{2.4}$$

The following algorithms can be used to estimate the plane parameters: Ordinary Least Squares (see Subsection 2.4.1), Total Least Squares (see Subsection 2.4.2), and Principal Component Analysis (see Subsection 2.4.3). In this Section, two epochs of data are used to create some example results, which are the point clouds of 2021 and 2021 of the SNS building (Voorstraat 58, Franeker).

### 2.4.1. Plane fitting using Ordinary Least Squares (OLS)

A plane can be fitted using the Ordinary Least Squares (OLS) algorithm, which is a linear regression technique. OLS minimizes the sum of the squared horizontal residuals $S$ (Tiberius, 2021). So, the estimated length coordinate $(\hat{L_i})$ is subtracted from the actual length $(L_i)$. The plane parameters $(A, B, C, D)$ can be estimated using an equation for the data points $(W_i, L_i, H_i)_{i=0...n-1}$, which is the general equation for a linear function with two variables ($W$ and $H$ in this case). In the equation for OLS, the value of B is assumed to be -1. So, the plane equation for OLS (Eberly, 2000) will become:

$$f(W, H) = L = \frac{AW + CH + D}{-B} = AW + CH + D \tag{2.5}$$

The function for the minimized sum of squared residuals is:

$$S = \arg\min \sum_{i=0}^{n-1} (L_i - \hat{L_i})^T (L_i - \hat{L_i}) \tag{2.6}$$

Where $L_i$ is the length component of a data point and $\hat{L_i}$ is the length component of a point on the plane estimated with the plane equation for OLS.

Using the plane equation for OLS (Eq. 2.5) and the coordinates of the data points, the values of the plane parameters can be determined. The plane parameters $A, C, D$ are estimated to best fit the data points $(W_i, L_i, H_i)_{i=0...(n-1)}$, so the sum of squared errors is minimized.

The following matrix equation (Eberly, 2000) is used to estimate the plane parameters:

$$\underbrace{\begin{bmatrix} W_1 & H_1 & 1 \\ W_2 & H_2 & 1 \\ \vdots & \vdots & \vdots \\ W_i & H_i & 1 \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} A \\ C \\ D \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ L_i \end{bmatrix}}_{\mathbf{b}} \tag{2.7}$$

The shape of matrix **M** is *nxm* and the shape of matrix **b** is *nx1*, where *m* represents the number of parameters to be estimated $(A, C, D)$ and *n* represents the amount of data points (the coordinates of the LiDAR points). However, in the case of fitting a plane through a facade, there are many data points. If the amount of data points is above twenty thousand points, too much memory needed is to perform the matrix multiplication. The plane parameters $A, C, D$ can be solved using Best Linear Unbiased Estimator (BLUE). So, the matrix equation (Teunissen, 2000) which is used to find the best estimation is :

$$\hat{\mathbf{x}} = (\mathbf{M}^T \mathbf{Q}_{yy}^{-1} \mathbf{M})^{-1} \mathbf{M}^T \mathbf{Q}_{yy}^{-1} \mathbf{b} \tag{2.8}$$

The matrix Equation for BLUE contains $\mathbf{Q}_{yy}$, which represents the variance of the length component. This variance matrix can be computed by determining the standard deviation of the length component of the dataset $\sigma_L$, which is:

$$\mathbf{Q}_{yy} = \sigma_L^2 I \tag{2.9}$$

If the amount of data points is large, there are two possibilities. The first option is to take a random subset of the data. This subset will contain ten thousand points. Another option is to use a summation method. The matrix for this summation method is (Eberly, 2001):

$$\underbrace{\begin{bmatrix} \sum_{i=0}^{n-1} W_i^2 & \sum_{i=0}^{n-1} W_i H_i & \sum_{i=0}^{n-1} W_i \\ \sum_{i=0}^{n-1} W_i H_i & \sum_{i=0}^{n-1} H_i^2 & \sum_{i=0}^{n-1} H_i \\ \sum_{i=0}^{n-1} W_i & \sum_{i=0}^{n-1} H_i & \sum_{i=0}^{n-1} 1 \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} A \\ C \\ D \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \sum_{i=0}^{n-1} W_i L_i \\ \sum_{i=0}^{n-1} H_i L_i \\ \sum_{i=0}^{n-1} L_i \end{bmatrix}}_{\mathbf{b}} \tag{2.10}$$

When using the summation method, the matrix **M** is a square matrix. So, the plane parameters are computed with:

$$\hat{\mathbf{x}} = \mathbf{M}^{-1} \mathbf{b} \tag{2.11}$$

After computing the values of $A, C$, and $D$, the plane can be constructed. In Python, this can be done by creating a meshgrid using the minimum and maximum values of the $W$- and $H$-coordinates. The length coordinates of the plane $L_p$ are obtained by inserting the values of the meshgrid into the following equation:

$$L_{p,OLS} = Aw + Ch + D \tag{2.12}$$

After computing the length coordinate of the plane, the plane is plotted as is shown in Figure 2.16. When the plane is estimated, it is interesting to know how well the inserted data points fit the plane and how accurate the fitted plane is.

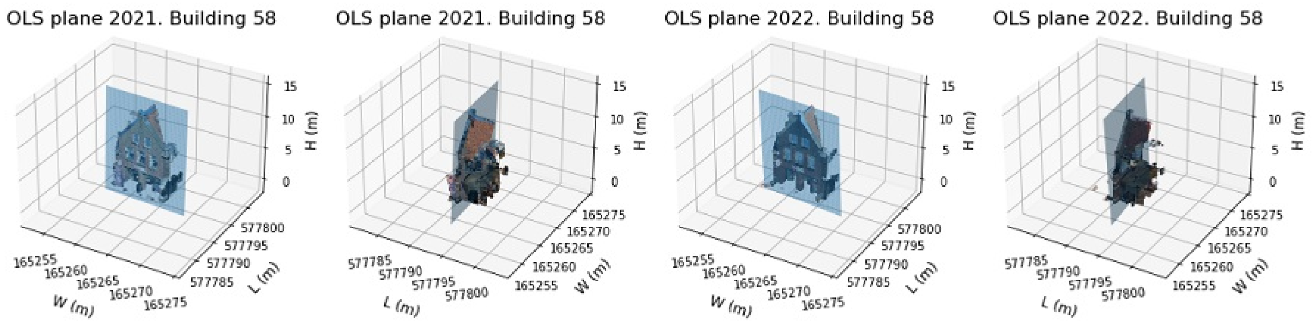


**Figure 2.16.:** *Result of plane fitting with Ordinary Least Squares (SNS building, Franeker).*

How well the data points fit the plane can be determined by calculating the distance between the data points and the plane, which is also known as the residuals $\hat{e}$. So, the residuals are the difference between the length coordinate $L$ and the predicted length coordinate of the plane $L_p$ which is:

$$\hat{\mathbf{e}}_{OLS} = \mathbf{L} - \mathbf{L}_{p,OLS} \tag{2.13}$$

When the distances between the points and the plane are computed, an image is created where the colors indicate the distance to the plane, as shown in Figure 2.17. Using this image, the distance between the facade points and the plane can be visualized. Thereby, it is possible to remove the points which are further away from the location of the facade, which are assumed to be outliers.

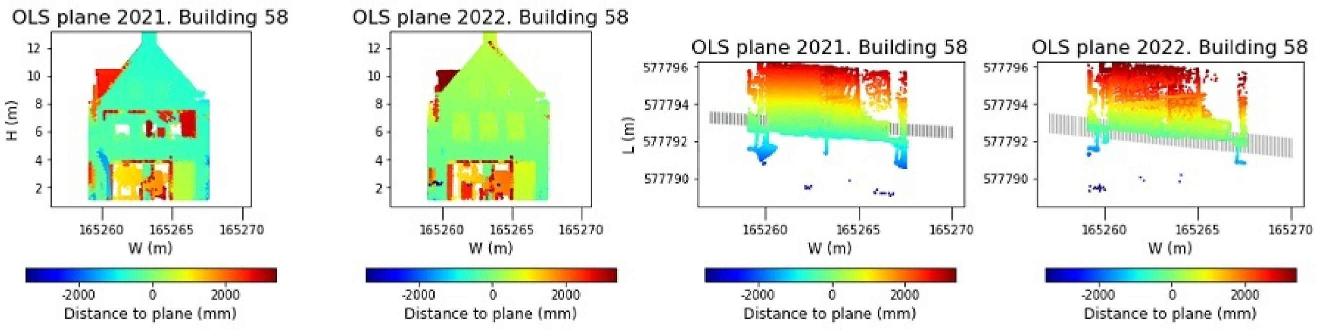


**Figure 2.17.:** *Result of the distances from the points towards the plane using Ordinary Least Squares (SNS building, Franeker).*

The Root Mean Squared Error (RMSE) indicates the precision of the residuals, and it is based on the plane parameters. It is a value that can be used to compare the quality of different areas with each other. Assuming the data is unbiased, the RMSE is the same as the standard deviation of the residuals and has meters as their unit. So, the RMSE can be computed with:

$$RMSE_{OLS} = \sigma_{\hat{\mathbf{e}},OLS} = \sqrt{\frac{\hat{\mathbf{e}}_{OLS}^T \hat{\mathbf{e}}_{OLS}}{n-m}} = \sqrt{\frac{\hat{\mathbf{e}}_{OLS}^T \hat{\mathbf{e}}_{OLS}}{n-3}} \tag{2.14}$$

In this equation, $n$ denotes the amount of data points, and $m$ is the number of parameters that are estimated, which is 3 in this case.

The quality of a plane estimated with the OLS algorithm $\mathbf{Q}_{\hat{x}\hat{x},OLS}$ is computed with (Teunissen, 2000):

$$\mathbf{Q}_{\hat{x}\hat{x},OLS} = (\mathbf{M}^T \mathbf{Q}_{yy,OLS}^{-1} \mathbf{M})^{-1} \tag{2.15}$$

This equation contains the covariance matrix of the observables[6] $\mathbf{Q}_{yy,OLS}$ which is:

$$\mathbf{Q}_{yy,OLS} = \sigma^2_{\hat{\mathbf{e}},OLS} I_n \tag{2.16}$$

The covariance matrix of the observables represents how well the points determine the plane parameters (Tiberius, 2021). So, this covariance matrix is the identity matrix $I$ scaled by the noise level $\sigma^2_{\hat{\mathbf{e}},OLS}$. The quality of planar fitting can be determined by computing $\hat{\sigma}^2_{m,OLS}$ with (Soudarissanane, 2016):

$$\hat{\sigma}^2_{m,OLS} = A_m \mathbf{Q}_{\hat{x}\hat{x},OLS} A_m^T \tag{2.17}$$

Due to using the center of gravity (the mean of the data points $A_m = [\bar{W}, \bar{L}, \bar{H}]$), the quality of planar fitting represents the range quality of the area.

### Example quality calculations Ordinary Least Squares

The RMSE and the quality of plane fitting $\hat{\sigma}^2_{m,OLS}$ can be computed for the SNS building in Franeker for the two years of data. As input data, a random set of five thousand data points is used.

For the dataset of 2021, the RMSE is

$$RMSE_{21} = \sigma_{\hat{\mathbf{e}},21} = 1.092[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,21} = \sigma^2_{\hat{\mathbf{e}},21} I_n = 1.193 I_n$$

The estimated parameters $A, B, C, D$ are

$$[-1.0 * 10^{-1}, 1.0, 4.1 * 10^{-2}, 5.9 * 10^5]$$

For the dataset of 2022, the RMSE is

$$RMSE_{22} = \sigma_{\hat{\mathbf{e}},22} = 0.929[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,22} = \sigma_{\hat{\mathbf{e}},22} I_n = 0.864 I_n$$

The estimated parameters $A, B, C, D$ are

$$[-1.4 * 10^{-1}, 1.0, -6.5 * 10^{-2}, 6.0 * 10^5]$$

[6]Observables are the inserted coordinates.

The covariance matrix of estimated parameters $(A, C, D)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},21} = \begin{bmatrix} 3.3*10^{-5} & 8.5*10^{-7} & -5.5 \\ 8.5*10^{-7} & 3.2*10^{-5} & -1.4*10^{-1} \\ -5.5 & -1.4*10^{-1} & 9.1*10^{5} \end{bmatrix}$$

The mean of matrix $\mathbf{M}$ is

$$A_{m,21} = [165263.13, 4.56, 1.00]$$

So, the quality of the planar parameters while estimating a plane with OLS is

$$\hat{\sigma}^2_{m,21} = 2.25*10^{-4}[-]$$

The covariance matrix of estimated parameters $(A, C, D)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},22} = \begin{bmatrix} 2.8*10^{-5} & -5.7*10^{-7} & -4.7 \\ -5.7*10^{-7} & 2.9*10^{-5} & 3.8*10^{-2} \\ -4.7 & 9.2*10^{-2} & 7.8*10^{5} \end{bmatrix}$$

The mean of matrix $\mathbf{M}$ is

$$A_{m,21} = [165263.31, 4.45, 1.00]$$

So, the quality of the planar parameters while estimating a plane with OLS is

$$\hat{\sigma}^2_{m,22} = 1.88*10^{-4}[-]$$

### 2.4.2. Plane fitting with Total Least Squares (TLS)

Instead of the OLS algorithm, a plane can also be estimated using Total Least Squares (TLS) which is based on homogeneous equations (the equation is equal to zero). In this case, the plane parameters $(A, B, C, D)$ are estimated using the plane equation (Eq. 2.4) for the data points $(W_i, L_i, H_i)_{i=0...n-1}$. So, the following matrix equation is used to estimate the plane parameters:

$$\underbrace{\begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{bmatrix}}_{\mathbf{M}_{[nxm]}} \underbrace{\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}}_{\mathbf{x}_{[mx1]}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{0}_{[nx1]}} \tag{2.18}$$

The residual will be smaller for TLS in comparison to OLS because the orthogonal distance to the plane is minimized. So, it is expected, this method will fit a plane that is slightly better than a plane that is fitted using OLS. The function for the minimized sum of squares is (Soudarissanane, 2016):

$$S = \arg\min ||\mathbf{Mx} - \mathbf{0}|| \tag{2.19}$$

The plane parameters are solved by computing the eigenvectors and eigenvalues using Singular Value Decomposition (SVD). SVD decomposes the matrix into three components, which are (Alter, 2000):

$$M_{[nxm]} = \underbrace{U_{[nxm]}}_{\text{Left singular vector}} \underbrace{S_{[mxm]}}_{\text{Singular values}} \underbrace{V^H_{[mxm]}}_{\text{Right singular vector}} \tag{2.20}$$

The eigenvalues are obtained by squaring the diagonal values of matrix $S_{[mxm]}$. The eigenvectors of $M^H M$ are the rows of $V^H$ because their shape is $m$x$m$ and $H$ represents the Hermitian transpose[7]. The eigenvector with the lowest eigenvalue contains the plane parameters. The columns of $U$ represent the eigenvectors of $MM^H$.

The plane parameters are the lowest eigenvalues and the components $W, H$ are a meshgrid that is created using the minimum and maximum values of the $W$- and $H$-coordinates. This results in a plane as is shown in Figure 2.18. The parameters of the plane can be used to compute the length coordinates of the plane which are:

$$L_{p,TLS} = \frac{AW + CH + D}{-B} \tag{2.21}$$

Next, the orthogonal distances to the plane, also known as the residuals, are computed with:

$$\hat{\mathbf{e}}_{TLS} = \frac{AW + BL + CH + D}{\sqrt{A^2 + B^2 + C^2}} \tag{2.22}$$

As mentioned before, the residuals of TLS are slightly different from the residuals of OLS because for TLS the distance is orthogonal to the plane instead of the difference in the length direction. The result of the computation of the orthogonal distance to the plane is shown in Figure 2.19.

[7] $H$ stands for the Hermitian transpose, which means that the matrix is first conjugated and then transposed. However, if the matrix contains only real values, the Hermitian transpose it the same as the transpose. So V$^H$ = V$^T$ (Hestenes, 1961).

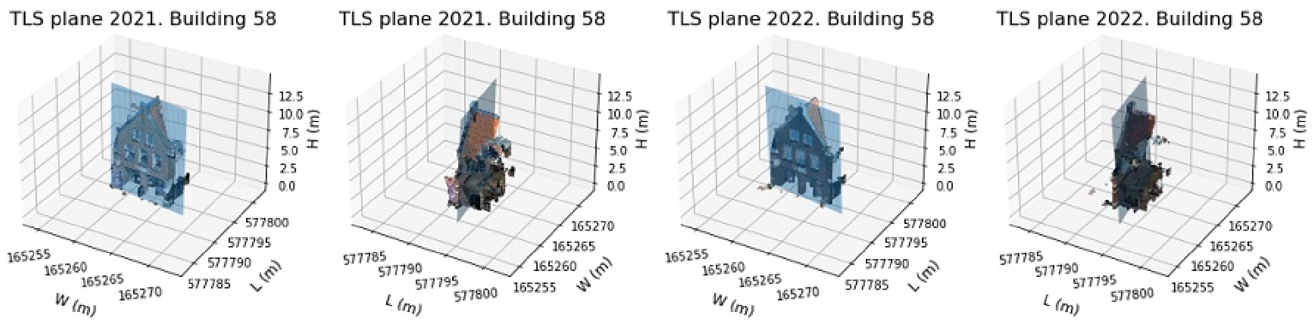


**Figure 2.18.:** *Result of plane fitting with Total Least Squares (SNS building, Franeker).*

The RMSE can be computed with (Soudarissanane, 2016):

$$RMSE_{TLS} = \sqrt{\frac{\hat{\mathbf{e}}_{TLS}^T \hat{\mathbf{e}}_{TLS}}{n-m+1}} = \sqrt{\frac{\hat{\mathbf{e}}_{TLS}^T \hat{\mathbf{e}}_{TLS}}{n-3}} \tag{2.23}$$

Where $n$ represents the amount of data points and $m$ represents the number of parameters that are estimated using this algorithm, which is 4 in this case because all the plane parameters $(A, B, C, D)$ are estimated with the same equation.

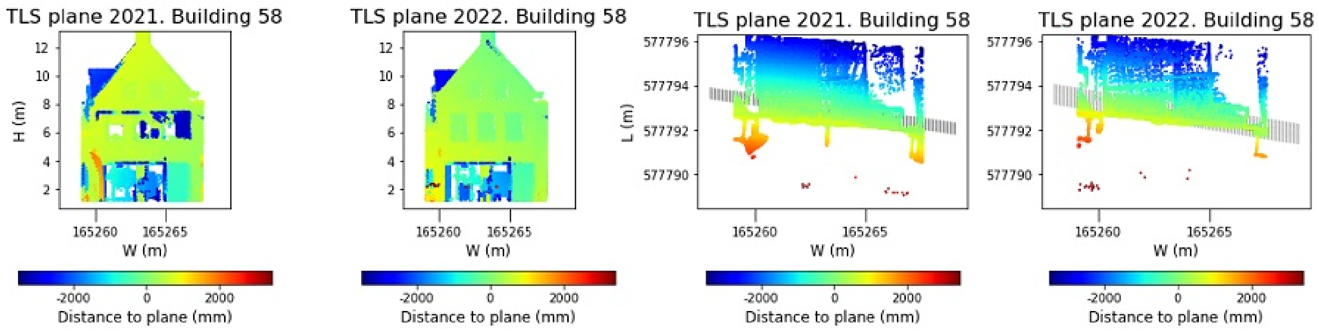


**Figure 2.19.:** *Result of the distances from the points towards the plane using Total Least Squares (SNS building, Franeker).*

The quality of a plane estimated with the TLS algorithm $\mathbf{Q}_{\hat{x}\hat{x},TLS}$ is computed with (Teunissen, 2000):

$$\mathbf{Q}_{\hat{x}\hat{x},TLS} = (\mathbf{M}^T \mathbf{Q}_{yy,TLS}^{-1} \mathbf{M})^{-1} \tag{2.24}$$

The covariance matrix $\mathbf{Q}_{yy,TLS}$ is:

$$\mathbf{Q}_{yy,TLS} = \sigma^2_{\hat{\mathbf{e}},TLS} I_n \tag{2.25}$$

The quality of planar fitting is determined by computing $\hat{\sigma}^2_{m,TLS}$. The matrix $A_m$ is the mean of the dataset, so $A_m = [\bar{W}, \bar{L}, \bar{H}, 1]$. So, the quality of planar fitting is computed with:

$$\hat{\sigma}^2_{m,TLS} = A_m \mathbf{Q}_{\hat{x}\hat{x},TLS} A_m^T \tag{2.26}$$

## Example quality calculations Total Least Squares

The RMSE and the quality of plane fitting $\hat{\sigma}^2_{m,TLS}$ can be computed for the SNS building in Franeker for the two years of data. As input data, a random set of five thousand data points is used.

For the dataset of 2021, the RMSE is

$$RMSE_{21} = \sigma_{\hat{\mathbf{e}},21} = 1.083[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,21} = \sigma^2_{\hat{\mathbf{e}},21} I_n = 1.172 I_n$$

For the dataset of 2022, the RMSE is

$$RMSE_{22} = \sigma_{\hat{\mathbf{e}},22} = 0.929[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,22} = \sigma_{\hat{\mathbf{e}},22} I_n = 0.864 I_n$$

The estimated parameters $A, B, C, D$ are

$$[-2.5 * 10^{-7}, -1.7 * 10^{-6}, 6.5 * 10^{-8}, 1.0]$$

The covariance matrix of estimated parameters $(A, B, C, D)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},21} = \begin{bmatrix} 3.7 * 10^{-5} & 2.0 * 10^{-5} & 4.9 * 10^{-8} & -1.8 * 10^{1} \\ 2.0 * 10^{-5} & 2.0 * 10^{-4} & -8.1 * 10^{-8} & -1.2 * 10^{2} \\ 4.9 * 10^{-8} & -8.1 * 10^{-8} & 3.4 * 10^{-5} & 4.6 \\ -1.8 * 10^{1} & -1.2 * 10^{2} & 4.6 & 7.1 * 10^{7} \end{bmatrix}$$

The mean of matrix $\mathbf{M}$ is

$$A_{m,21} = [165263.13, 577792.87, 4.56, 1.00]$$

So, the quality of the planar parameters while estimating a plane with TLS is

$$\hat{\sigma}^2_{m,21} = 2.34 * 10^{-4} [-]$$

The estimated parameters $A, B, C, D$ are

$$[-2.8 * 10^{-7}, -1.7 * 10^{-6}, -1.1 * 10^{-7}, 1.0]$$

The covariance matrix of estimated parameters $(A, B, C, D)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},22} = \begin{bmatrix} 2.5 * 10^{-5} & 2.7 * 10^{-5} & 1.3 * 10^{-6} & -2.0 * 10^{1} \\ 2.7 * 10^{-5} & 2.0 * 10^{-4} & 1.3 * 10^{-5} & -1.2 * 10^{2} \\ 1.3 * 10^{-6} & 1.3 * 10^{-5} & 2.3 * 10^{-5} & -7.6 \\ -2.0 * 10^{1} & -1.2 * 10^{2} & -7.6 & 7.1 * 10^{7} \end{bmatrix}$$

The mean of matrix $\mathbf{M}$ is

$$A_{m,21} = [165263.31, 577792.94, 4.45, 1.00]$$

So, the quality of the planar parameters while estimating a plane with TLS is

$$\hat{\sigma}^2_{m,22} = 1.43 * 10^{-4} [-]$$

### 2.4.3. Plane fitting with Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is an algorithm that can be used to reduce the dimensionality of a dataset by calculating vectors (Ringnér, 2008). The calculated vectors, also known as principal components, are in the direction of the maximal variation of the point cloud. Besides the principal components, the vectors of the plane can be computed. There are plane vectors in three different directions, two of them are in the direction of the plane, and the normal vector is perpendicular to the plane. The normal vector can be used to plot the plane and by computing the angle between two normal vectors of different epochs, it can be checked if deformation occurs.

Before applying the PCA algorithm, the data needs to be centralized because PCA is sensitive to variances. The data is centralized by subtracting the mean from the coordinates. The centralized data is represented with $\mathbf{X}_C$. $\mathbf{X}$ are the coordinates of the points $[W, L, H]$ and $\mathbf{X}_M$ is the mean of the data $[\bar{W}, \bar{L}, \bar{H}]$. So, the centralized data is computed with:

$$\mathbf{X_C} = \mathbf{X} - \mathbf{X_M} \tag{2.27}$$

Table 2.2 shows an example of possible coordinates and the result after centralization of the data. The mean was first around 577800 and after centralizing, it is around 8. If the mean of a feature (a certain coordinate) is high, it will have a greater impact on the results of PCA, because in PCA higher values will have greater importance. After centralizing, the coordinates will have the same scale, so the importance of the different coordinates will be more similar than before.

**Table 2.2.:** *An example of the effect of centralizing the data. The values have meters as their unit.*

| Coordinate | Max value | Min value | Mean value | Min value after centralizing | Max value after centralizing |
|---|---|---|---|---|---|
| **W** | 165259.5 | 165267.3 | 165263.4 | -3.9 | 3.8 |
| **L** | 577792.0 | 577792.8 | 577792.4 | -0.4 | 0.4 |
| **H** | 1.0 | 13.2 | 5.5 | -4.5 | 7.6 |

After centralizing the data, the relationship between the three-dimensional data points is identified by computing the covariance matrix (Pishro-Nik, 2016), which is the following 3x3 symmetric matrix:

$$C = \frac{1}{n-1}\mathbf{X}_c\mathbf{X}_c^T = \begin{bmatrix} Cov(W,W) & Cov(L,W) & Cov(H,W) \\ Cov(W,L) & Cov(L,L) & Cov(H,L) \\ Cov(W,H) & Cov(L,H) & Cov(H,H) \end{bmatrix} = \begin{bmatrix} Var(W) & Cov(L,W) & Cov(H,W) \\ Cov(W,L) & Var(L) & Cov(H,L) \\ Cov(W,H) & Cov(L,H) & Var(H) \end{bmatrix} \tag{2.28}$$

If the sign of the covariance matrix is positive, the variables are correlated and when the sign is negative, the variables are inversely correlated. The covariance can be computed by multiplying $\mathbf{X}_c$ with its transpose $\mathbf{X}_c^T$. The result of this matrix multiplication is divided by the length of the dataset minus one (Soudarissanane, 2016).

After computing the covariance matrix, the eigenvectors and eigenvalues are computed with the *np.linalg.eig* function in Python. After using this function, the eigenvectors are ordered from the highest eigenvalue (highest variance) to the lowest eigenvalue (lowest variance).

The eigenvectors of the covariance matrix are the principal components, which are the directions of the axes that capture the most information. The eigenvalues give the amount of variance carried out in each principal component.

The ordered eigenvectors are used to fit a plane through the data. The normal vector of the plane is the eigenvector with the lowest eigenvalue. This eigenvector will represent the parameters $A$, $B$ and $C$ in the following plane equation:

$$AW + BL + Ch = D \tag{2.29}$$

Where $D$ is a constant value that can be calculated by multiplying the centroid of the plane $(\bar{W}, \bar{L}, \bar{H})$ with the plane parameters $A, B, C$. Plane parameter $D$ is computed as follows:

$$D = -[\bar{W}, \bar{L}, \bar{H}][A, B, C] \tag{2.30}$$

Now, all the plane parameters $A, B, C, D$ are computed. Using the plane parameters and the constructed meshgrid (containing the minimum and maximum values of $W$ and $H$), the length coordinate of the plane can be estimated with:

$$L_{p,PCA} = \frac{AW + CH + D}{-B} \tag{2.31}$$

The result of plane fitting is shown in Figure 2.20.

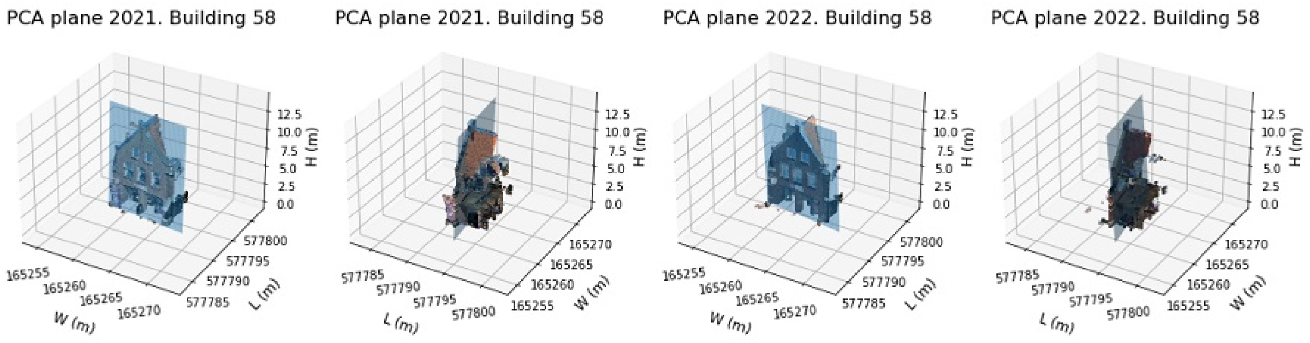


**Figure 2.20.:** *Result of plane fitting with Principal Component Analysis (SNS building, Franeker).*

The distance towards the plane is visualized in Figure 2.21. The orthogonal distances to the plane, also known as the residuals, are (Soudarissanane, 2016):

$$\hat{\mathbf{e}}_{PCA} = \frac{AW + BL + CH + D}{\sqrt{A^2 + B^2 + C^2}} \tag{2.32}$$

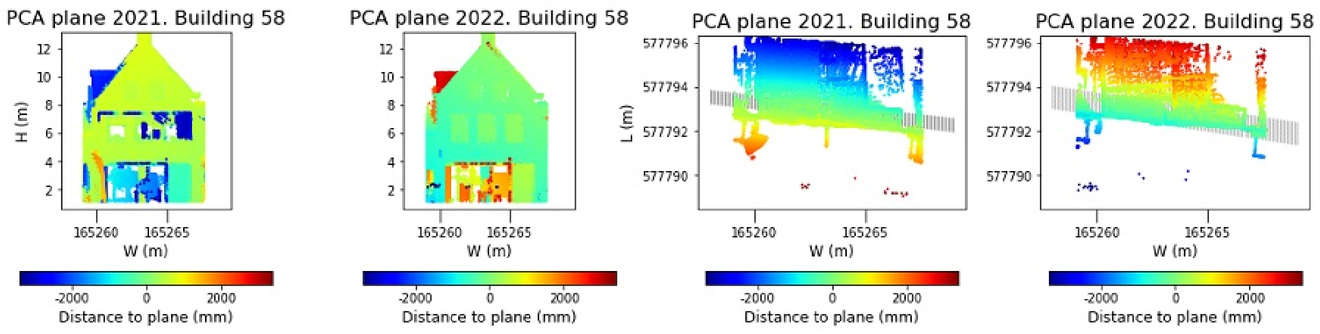


**Figure 2.21.:** *Result of the distances from the points towards the plane using Principal Component Analysis (SNS building, Franeker).*

The RMSE is computed with (Soudarissanane, 2016):

$$RMSE_{PCA} = \sqrt{\frac{\hat{\mathbf{e}}_{PCA}^T \hat{\mathbf{e}}_{PCA}}{n - m + 1}} = \sqrt{\frac{\hat{\mathbf{e}}_{PCA}^T \hat{\mathbf{e}}_{PCA}}{n - 2}} \tag{2.33}$$

Where $n$ represents the amount of data points and $m$ represents the number of parameters that are estimated using this algorithm, which is 3 $(A, B, C)$.

The quality of a plane estimated with the PCA algorithm $\mathbf{Q}_{\hat{x}\hat{x},PCA}$ is computed with Teunissen, 2000:

$$\mathbf{Q}_{\hat{x}\hat{x},PCA} = (\mathbf{X}^T \mathbf{Q}_{yy,PCA}^{-1} \mathbf{X})^{-1} \tag{2.34}$$

The covariance matrix $\mathbf{Q}_{yy,PCA}$ is:

$$\mathbf{Q}_{yy,PCA} = \sigma^2_{\hat{\mathbf{e}},PCA} I_n \tag{2.35}$$

The quality of planar fitting is determined by computing $\hat{\sigma}^2_{m,PCA}$. The matrix $A_m$ is the mean of the dataset $(A_m = [\bar{W}, \bar{L}, \bar{H}])$. So, the quality of planar fitting is computed with:

$$\hat{\sigma}^2_{m,PCA} = A_m \mathbf{Q}_{\hat{x}\hat{x},PCA} A_m^T \tag{2.36}$$

#### Example quality calculations Principal Component Analysis

The RMSE and the quality of plane fitting $\hat{\sigma}^2_{m,PCA}$ can be computed for the SNS building in Franeker for the two years of data. As input data, a random set of five thousand data points is used

For the dataset of 2021, the RMSE is

$$RMSE_{21} = \sigma_{\hat{\mathbf{e}},21} = 1.080[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,21} = \sigma^2_{\hat{\mathbf{e}},21} I_n = 1.167 I_n$$

The estimated parameters $A, B, C, D$ are

$$[1.2 * 10^{-1}, 9.9 * 10^{-1}, -4.8 * 10^{-2}, -5.9 * 10^{5}]$$

The covariance matrix of estimated parameters $(A, B, C)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},21} = \begin{bmatrix} 3.2 * 10^{-5} & -9.2 * 10^{-6} & 1.2 * 10^{-6} \\ -9.2 * 10^{-6} & 2.6 * 10^{-6} & -3.5 * 10^{-7} \\ 1.2 * 10^{-6} & -3.5 * 10^{-7} & 3.3 * 10^{-5} \end{bmatrix}$$

The mean of matrix $\mathbf{X}$ is

$$A_{m,21} = [165263.13, 577792.87, 4.56]$$

So, the quality of the planar parameters while estimating a plane with PCA is

$$\hat{\sigma}^2_{m,21} = 2.33 * 10^{-4}[-]$$

For the dataset of 2022, the RMSE is

$$RMSE_{22} = \sigma_{\hat{\mathbf{e}},22} = 0.844[m]$$

The covariance matrix of the observables is

$$\mathbf{Q}_{yy,22} = \sigma_{\hat{\mathbf{e}},22} I_n = 0.713 I_n$$

The estimated parameters $A, B, C, D$ are

$$[1.5 * 10^{-1}, 9.9 * 10^{-1}, 7.2 * 10^{-2}, 5.9 * 10^{5}]$$

The covariance matrix of estimated parameters $(A, B, C)$ is

$$\mathbf{Q}_{\hat{x}\hat{x},22} = \begin{bmatrix} 2.5 * 10^{-5} & -5.6 * 10^{-6} & -7.8 * 10^{-7} \\ -5.6 * 10^{-6} & 1.6 * 10^{-6} & 2.2 * 10^{-7} \\ -7.7 * 10^{-7} & 2.2 * 10^{-7} & 2.2 * 10^{-5} \end{bmatrix}$$

The mean of matrix $\mathbf{X}$ is

$$A_{m,21} = [165263.31, 577792.94, 4.45, 1.00]$$

So, the quality of the planar parameters while estimating a plane with PCA is

$$\hat{\sigma}^2_{m,22} = 1.43 * 10^{-4}[-]$$

## 2.5. Random Sample Consensus (RANSAC) for segmentation

Random Sample Consensus (RANSAC) is designed to deal with a large proportion of outliers in the input data (Derpanis, 2010). The RANSAC algorithm can be used to segment data. RANSAC uses the smallest possible set of data to fit a geometric primitive to a point cloud and enlarges this with consistent data points. There are different geometric primitives, for example, cylinders, spheres, cuboids, or a plane (Mariga, 2021). In this thesis, the focus lies on RANSAC using a plane as a geometric primitive.

The RANSAC algorithm consists of different steps (Derpanis, 2010), which are shown below.

1. A specific amount of initial points (*ransac n*) are randomly selected from the dataset. These randomly selected points are used to determine the parameters of the model.

2. The parameters of the model are estimated.

3. There is checked how many points fit within the predefined threshold value (*distance threshold*). The points within the threshold value are the inliers and the other points are outliers.

4. When the number of inliers with respect to the total number of points is above a certain threshold $\tau$, the model parameters are estimated again by using all the identified inliers.

5. If the threshold $\tau$ is not reached, steps 1 to 4 are repeated. The steps can only be repeated with a maximum amount specified by the number of iterations (*num iterations*).

### Example of RANSAC implementation in Python

The RANSAC algorithm can be used for model-based segmentation. This model-based method is implemented using the *segment plane* function from the Open3D library in Python (Poux, 2021) computing the plane parameters *(A, B, C, D)* and the indices of the inliers. The function takes the three-dimensional coordinates of the point cloud as input, together with three parameters: *distance threshold*, *ransac n*, and *num iterations*. An example of a possible output is shown in Figure 2.22.

The *distance threshold* is the maximum distance to the plane. If the distance between a point and the plane is above this threshold, the point will be considered as an outlier. *Ransac n* represents the minimal number of points needed to create a plane. So, this value needs to be at least three, because a plane can be defined by three points if these points are not on the same line. *Num iterations* stands for how many iterations are performed before the optimal plane is selected, which is set to 1000. If the number of iterations is too low, the optimal plane might not be found. If the number of iterations is too high, it might take longer to get the optimal plane. The optimal plane is the plane through the points of interest, for example, points on the facade or points on the ground.

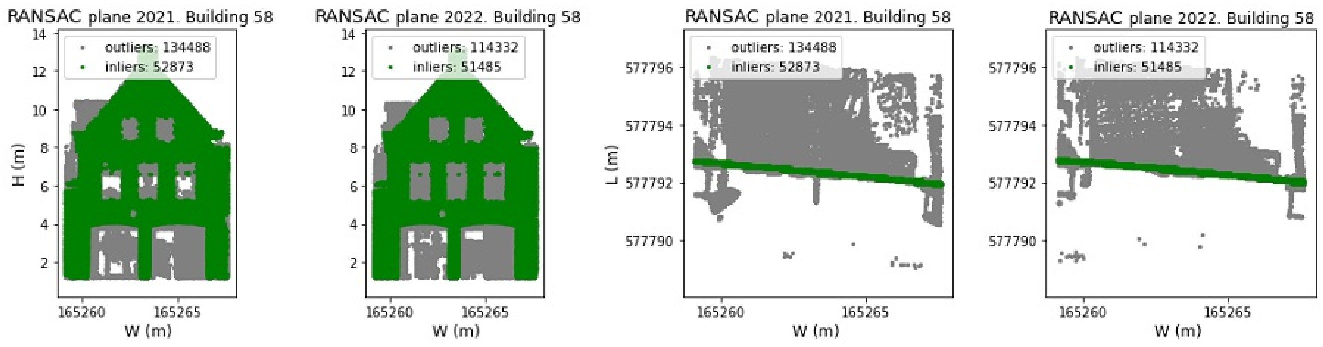


**Figure 2.22.:** *Inliers (in green) and outliers (in grey) of RANSAC. The distance threshold is 0.04 m, ransac n is 3, and num iterations is 1000.*

# 3. Dataset and Study areas

This chapter first describes the LiDAR dataset in Section 3.1. Secondly, the InSAR dataset is described in Section 3.2. Thirdly, in Section 3.3, a description is given of how possible study areas can be found and which study areas are chosen for this research.

## 3.1. Cyclomedia datasets and tools (LiDAR)

Every year, Cyclomedia captures images and street LiDAR data from roads and buildings in the Netherlands with its own Mobile Laser Scanning (MLS) system described in Subsection 3.1.1. Street LiDAR data of Cyclomedia is stored in two different formats: point clouds (see Subsection 3.1.2) and depth maps (see Subsection 3.1.3). In addition to these two formats, it is also possible to use Street Smart (an application of Cyclomedia) which is described in Subsection 3.1.4.

### 3.1.1. Mobile Laser Scanning System

Point clouds are obtained using street LiDAR, which is measured using a MLS system. The laser scanner sends out pulses of light to a surface and measures how long it takes for this signal to return. So, LiDAR measures the time between emitting and receiving the signal. These measurements are used to determine the position of points from the object to the laser scanner. All the points together form a point cloud.

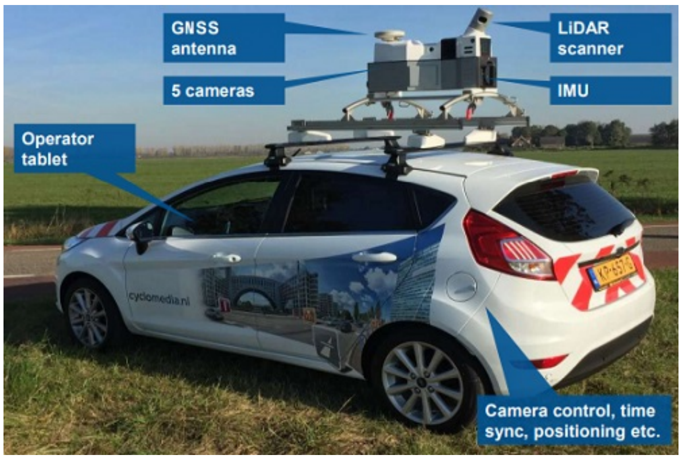


**Figure 3.1.:** *HD-Cyclorama recording system: DCR10.*

Cyclomedia has its own data capture system which consists of vehicles collecting street-level data, see Figure 3.1. The vehicles are driven by the operators of Cyclomedia, and have a system on the roof, called the HD-Cyclorama recording system: Digital Cyclorama Recording system Version 10 (DCR10). The system consists of a GNSS antenna (measuring the position), a LiDAR scanner (collecting the point cloud), 5 cameras (creating a panoramic image, called Cyclorama), and an IMU (for the inertial measurements and orientation).

When taking measurements with a vehicle, the exact scanning location will be different for each epoch. By using a combination of GNSS and IMU data, the location of the vehicle is determined. This makes it possible to compare data from different epochs. The GNSS antenna is used to determine the position of the vehicle to at least four different satellites (Hein, 2020), resulting in the $x, y, z$ coordinates of the vehicle. In some cases, there are not enough satellites in range. For example, when driving in a city with high buildings or trees blocking the signal from the satellites. So, besides GNSS, an IMU is needed which measures acceleration and angular velocity (Constant, 2021). Using the data from the IMU, the position of the vehicle can continuously be determined even when the GNSS receiver does not obtain a signal from the satellites (Cahyadi, 2019).

In the Netherlands, the time difference between the epochs is approximately a year because recordings are taken once a year on every road with a measurement interval of 5 meters. The measurement values have a certain precision and accuracy. The precision (or relative accuracy) depends on random errors in the data and the accuracy (or absolute accuracy) depends on systematic errors.

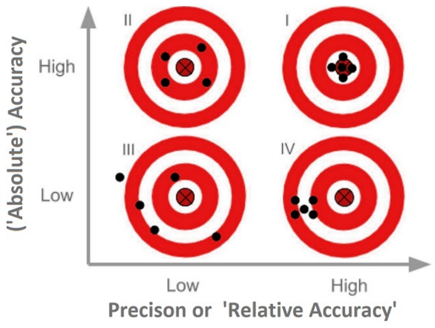


**Figure 3.2.:** *Accuracy and precision.*

The data is precise if multiple measurements performed under equal conditions give a similar output (Hofer, 2005). For example, when measuring the location of a house number plate. The measurement points are precise if they are all located on this number plate, but they are also precise if they are all around the handle of the door. So, the precision indicates the maximum distance between the different data points. If the points are far away from each other, the data is not precise.

The data is accurate when the data points are located in the area of interest, for example, the house number plate. If the data points are on the number plate or if they are at a certain distance around the number plate, the data is accurate. The data is not accurate when the data points are centered around another object, for example, the door handle. So, the accuracy is based on the distance from the data points to the object of interest.

The LiDAR measurements have an average accuracy of 10 centimeters and a precision of 2 centimeters (Boom, 2019). The quality of the data is influenced by the GNSS reception. So, at some locations, the quality of the data could be even better than the mentioned average. In comparison, while using InSAR the precision of displacement velocity over time is around 1 millimeter per year and the accuracy of the data points is between 10 and 15 meters. However, during the processing of the data, the uncertainty of the location can be reduced to 1 meter (SkyGeo, 2022).

Besides the GNSS reception, the quality is also influenced by other factors. For example, weather conditions and the amount of light. The quality of the LiDAR data decreases if it is raining very hard, as mentioned in Section 2.2. Besides the LiDAR measurements, the quality of the pictures also decreases due to water drops on the lens of the camera or raindrops in the picture blocking the view. A lack of light only reduces the quality of the pictures. However, the pictures are an important part of the data so no measurements are taken between sundown and sunrise.

If the quality of the data is bad (low precision, low accuracy, or bad image), the operator gets a notification on the operator's tablet or a call from the office. In both cases, the operator needs to capture the data again to maintain the promised quality.

### 3.1.2. Street LiDAR point clouds (LAS file)

A point cloud is a set of points in a three-dimensional coordinate system. The points represent the shape of objects, like buildings, trees, or cars. Besides the coordinates, point clouds can also contain additional information, like the color of a point (stored in RGB format). A point cloud can be stored as a LAS or LAZ file (compressed LAS file) file that contains the data points. The LAS files are stored in cells with a size of 50 by 50 meters (in $x$- and $y$- direction). The LAS file contains different features, but only the coordinates and colors are used, see Figure 3.3. The GNSS system and the IMU are used to measure the $x$-, $y$-, and $z$-coordinates, which pinpoint the location of a certain point. The RGB colors are obtained by the 5 cameras and assigned to the three-dimensional points.

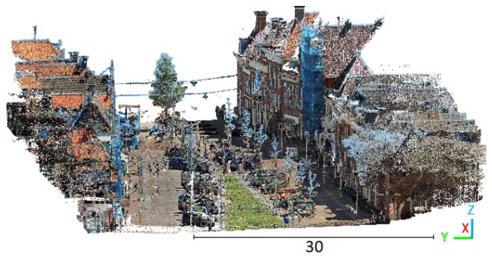


**Figure 3.3.:** *Visualization of RGB point cloud data from Cyclomedia captured in Franeker.*

### 3.1.3. Street LiDAR depth maps (PNG file)

A depth map is an image that contains information about the distance (length component) of objects measured from a certain angle stored as a Portable Network Graphic (PNG) file. Each pixel (height, width) in this image represents a distance between the reference point to a certain object (Jamwal, 2016). For example, the distance from the scanner to the facade. In a depth map, the distances can be visualized as a greyscale image or they can be represented as a colored image. Figure 3.4 shows an example of a depth map. Points that are close to the camera (so points in the foreground) are indicated in blue, and points far away from the camera (so points in the background) are indicated in red.

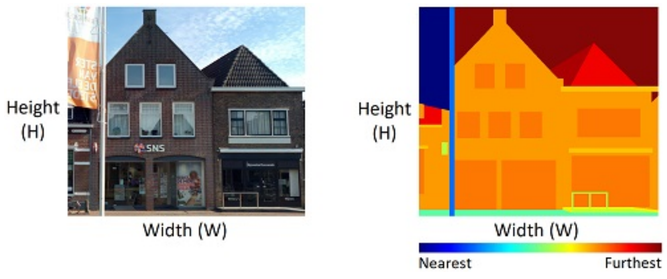


**Figure 3.4.:** *Example of an image captured by Cyclomedia (left) and a corresponding depth map (right).*

A depth map is different from a point cloud because a depth map has a fixed viewpoint (so from a particular angle). In the case of a point cloud, the data can be visualized from different viewing angles. So, while using a point cloud, it is possible to look behind objects when measurements are taken from different positions. A depth map only contains the distance to the scanner for different locations instead of the three-dimensional position of a point. A point cloud can also contain the color of the data points, while a depth map only contains the distance.

The availability of depth maps differs from the availability of LAS files. LAS files captured by Cyclomedia are only available from the last 2 to 3 years, but depth maps are available from the last 4 to 5 years. So, when a larger period is needed to check if there is deformation present, it might be better to use depth maps. The colors in the depth map can be converted to the distance to reconstruct the point cloud. The quality of a depth map might differ from the quality of a LAS file, so it is a good idea to compare the quality of these datasets[1]. The dataset can be compared to check if using a different representation gives a different result of the amount of deformation that is present.

### 3.1.4. Street Smart

Street Smart is an application of Cyclomedia. It is a website containing the data captured by the operators of Cyclomedia. So, the website contains the Cyclorama images, the point clouds, and the depth maps. The website has different tools which can be used. For example, the green circle shown in Figure 3.5 which drapes over the three-dimensional surface of the image. So, using this circle, it is possible to see the shape of an object in a two-dimensional image. Besides the shape of an object, the green circle can be used to get the coordinates of a certain point in the Cyclorama image (a 360 degrees panoramic image). The coordinates are displayed in the left bottom corner of the image.

[1]This is out of the scope of this thesis.

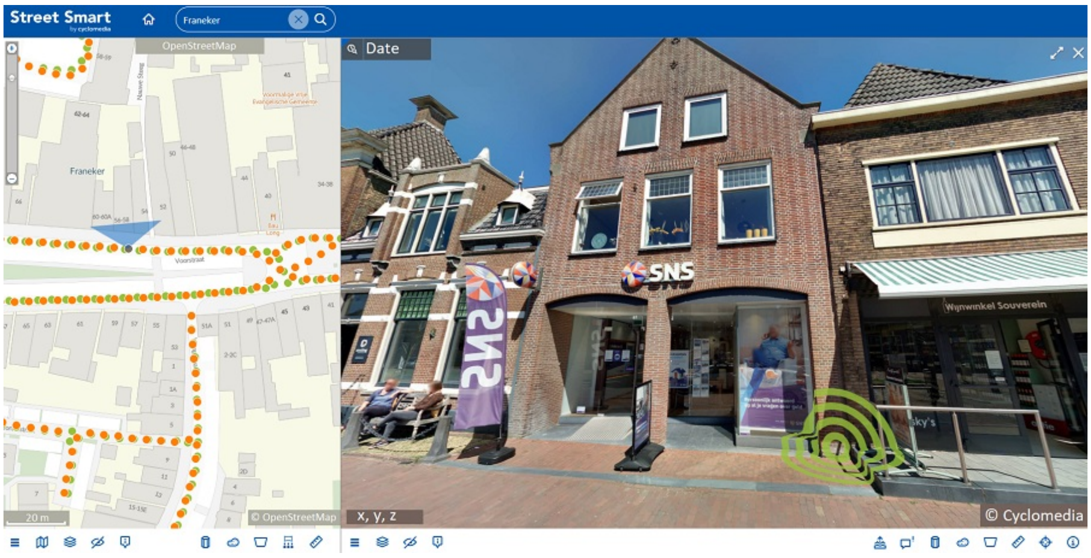


**Figure 3.5.:** *Depth map image in Street Smart.*

The extracted coordinates can be used to estimate a plane through the facade. Besides the coordinates, it is also possible to perform the following measurements: location, distance, surface, orthogonal, and height. So, Street Smart can be a useful tool to check if the generated code works.

## 3.2. Surface and object motion map (InSAR)

The Dutch surface and object motion map (abbreviated to Dutch motion map) is a dataset containing information about surface deformation. The map is constructed using the InSAR technique. InSAR is based on the principle of radio detection and ranging (RADAR). A RADAR signal is sent in the line-of-sight direction of the satellite towards the Earth's surface. The signal is sent to the Earth under an angle, so the sensor will be more sensitive to vertical deformation than to horizontal deformation (Tofani, 2013) when the incidence angle is small. After reaching the Earth, the signal is scattered and reflected to the satellite, also known as backscatter. This backscatter is recorded as the amplitude and the phase of the signal (Helz, 2021) which are stored as Synthetic Aperture Radar (SAR) images. The amplitude contains information about the roughness of the Earth's surface. The phase is used to determine deformation of the surface by measuring the fraction of the last phase cycle. It is possible to determine whether deformation occurs between two epochs by comparing the phase differences between two SAR images.

For the Dutch motion map, the SAR-images are captured by the Sentinel-1a and Sentinel-1b satellites. GNSS data is used to determine the position of the satellite (NCG, 2023a) and the position of the satellites is used to determine the location of the data points. The two Sentinel satellites fly in the same orbital plane, as shown in Figure 3.6. The Sentinel satellites have a repeat cycle of 12 days, so every 6 days a satellite flies over the Netherlands (Geudtner, 2012a). The Dutch motion map is only updated a couple of times per year (NCG, 2023b). However, while using the same data, a more dedicated data analysis can be made, at a higher frequency.

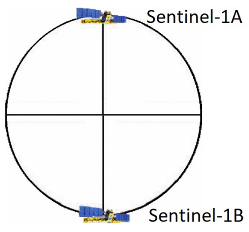


**Figure 3.6.:** *The orbital configurations of Sentinel-1 Geudtner, 2012b.*

The deformations in the Dutch motion map are just estimates. So, the Dutch motion map is not suited for specific technical applications because the map is a simplification of reality. In addition to the Dutch surface motion map, a more detailed analysis is possible using InSAR data. For example, with the Persistent Scatterer Interferometry (PSI) technique, which combines multiple SAR images of an area, to create a map containing deformation (Crosetto, 2016).

## 3.3. Study areas

For this research, an interesting study area should contain deformation. To find deforming locations, it is possible to use news articles with complaints of people about damage to their homes as a reference, but it is also possible to use InSAR data. The Dutch surface motion map shows InSAR data of the Netherlands. The InSAR data in this map is an estimate of the actual displacement, which is based on various assumptions (SkyGeo, 2022). However, this is assumed to be good enough for this application because these locations are chosen to increase the chance of finding actual deformation.

Besides InSAR data, information on the soil type can be used. In Figure 3.7, different soil types in the Netherlands are displayed. When the groundwater level decreases, subsidence occurs in cohesive soils. Cohesive soils are peat, clay, loam, and löss soils (Tromp, 2008). So, in areas with these soil classes, there is a higher chance of deformation.

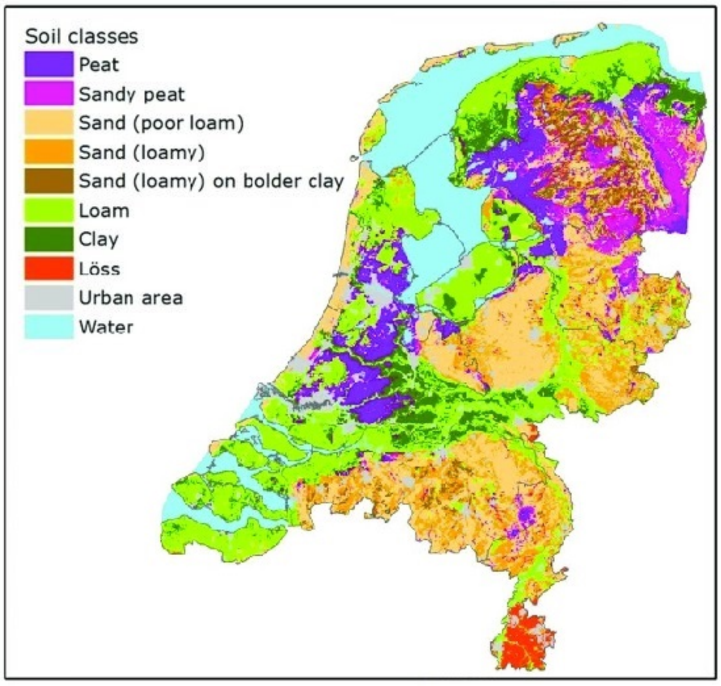


**Figure 3.7.:** *Map containing the soil types in the Netherlands (Tiktak, 2017).*

When using InSAR or the soil type, a subsiding area can be found. However, a subsiding area does not necessarily lead to deformation or damage to a building. If a building is subsiding uniformly, it is not likely that the building is damaged (the building will just settle uniformly). However, when the settlement is differential, the amount of settlement is not equally distributed along the foundation. Differential settlement can lead to cracks in the facade (as mentioned in Subsection 2.1.1).

So, interest lies in areas with gradients. These areas could cause damage to buildings because a part of the building could be subsiding, and the other part might be stable or rising due to differential settlement. So, the most interesting areas are locations with a subsiding building on one side and a rising building on the other side of the road. Another interesting thing is when there is deformation (subsidence or rising) at one side and no deformation (stationary) on the other side, which might be hard to find. Using locations like this, it might be more likely that the settlement of a foundation is differential instead of uniform which makes the chance of detecting deformation higher.

It is also important to check the availability of the data because at some locations the number of available LAS files is lower. Locations with a higher amount of data (3 epochs of data available) might be preferred over locations with less data (2 epochs of data available). Besides the availability of data, preference lies in areas with a limited amount of vegetation, because trees and other plants might be in front of the buildings making it harder to find the facade of a building.

For the circles displayed in Figure 3.8, the following cities and streets are considered to be interesting[2]:

- Circle 1: Franeker. Voorstraat and Waagstraat (gas extraction and salt mining (Esfahany, 2010))
- Circle 2: Several villages in Groningen, for example:
  - Kruisweg in Loppersum (earthquake in July 2020 due to gas extraction (Darwinkel, 2020))
  - Burgemeester Meesstraat 32-40 in Appingedam (reinforcement of houses in 2018/2019 due to effects of gas extraction (Numan, 2018))
- Circle 3: Amsterdam. Possible interesting locations are:
  - 'Dancing' houses at Amstel 100-112 (subsidence of peat soil (Gemeente Amsterdam, 2020a))
  - Vijzelgracht 4-8 and 24-26 (construction of the Noord-Zuidlijn. These buildings were reinforced in 2016 (Meershoek, 2016))
- Circle 4: Almere. Chamoisstraat or Bankierbaan (subsidence of the street due to weak sea clay (Maas, 2021))
- Circle 5: De Lier. Meidoornhof or Eikenhof (subsidence of peat soil due to groundwater extraction for greenhouses (College Westland, 2022))
- Circle 6: Kerkrade. Franciscanerstraat 5-12 (collision of a mine shaft in 2020) (Vos, 2020)

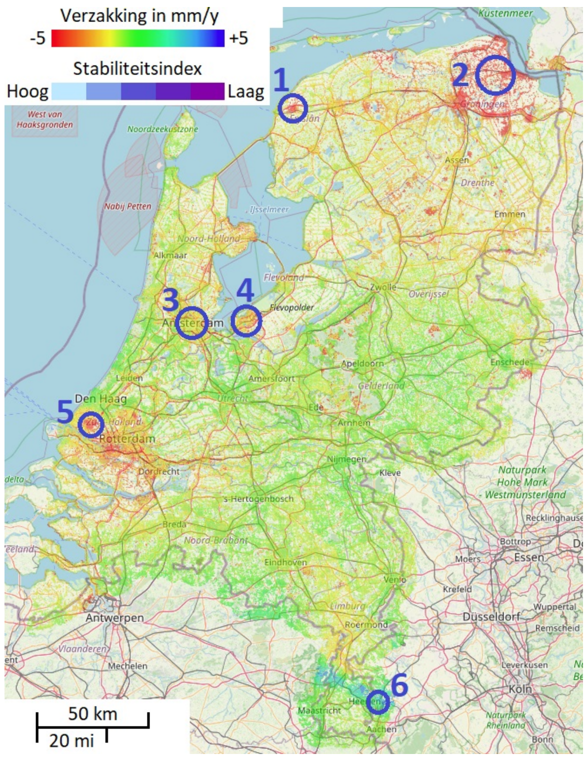


**Figure 3.8.:** *Dutch surface motion map. Interesting locations are indicated with a circle (SkyGeo, 2022).*

[2]Other locations might also be interesting. The mentioned locations are just some examples.

For this research, a Python code is written to analyze deformation of a facade. The code is tested on different buildings to make the code general. If the code is general, the code could be used to check if there is deformation on all the different locations mentioned in the list above.

Two of the six study areas are selected to examine. The first study area is Franeker (see Subsection 3.3.1), and the second study area are the dancing houses in Amsterdam (see Subsection 3.3.2). The data of Franeker is used to create the Python code. The generated code is applied to the data of Amsterdam to determine the current tilt angle of the dancing houses.

### 3.3.1. Voorstraat and Waagstraat in Franeker

The Voorstraat is a street that goes through the center of the historical part of Franeker while the Waagstraat is perpendicular to the Voorstraat. The surface motion map of both streets is shown in Figure 3.9. From the map, it can be seen that Franeker is subsiding. There are two possible causes, which are gas extraction and salt mining.

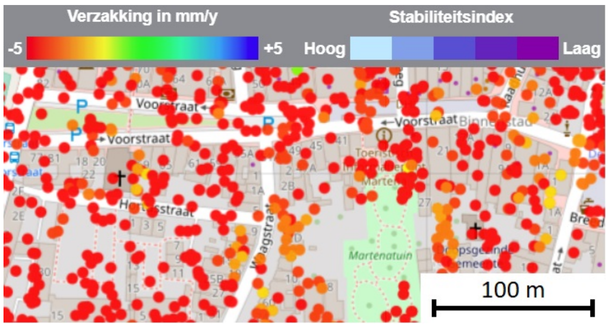


**Figure 3.9.:** *Surface motion map of the Voorstraat and Waagstraat in Franeker (SkyGeo, 2022).*

In 1988 people started extracting gas from the Harlingen gas field (located west of the city of Franeker). Besides gas extraction, salt mining started in 1995 northwest of the gas field. In 2003 a second mine became operational north of the gas field. The subsidence mainly occurred due to gas extraction, salt mining only had a small contribution. The gas extraction near Franeker stopped in July 2008, because the amount of subsidence was 3 centimeters larger than the allowed 12 centimeters (Binnenlands Bestuur, 2008). Even after stopping the gas extraction, the soil kept subsiding and it is expected this will continue until 2038 (Houtenbos, 2010). So, the after-effects of the gas extraction might still be visible in the data of 2022.

Point cloud data from 2021 and 2022 can be compared to check if there are subsiding effects visible. The point cloud from 2021 was captured in the afternoon on the 1st of June, and the point cloud from 2022 was captured in the morning on the 16th of May. The colors of the images might be dissimilar due to the position of the sun. Both point clouds use the EPSG:28992 Coordinate Reference System (CRS), which is commonly used in the Netherlands.

There are many buildings in Franeker, but only a few of them are selected to examine if deformation occurs. Buildings constructed before 1970 have a shallow foundation (Stichting Kennis Centrum Aanpak Funderingsproblematiek, 2022) and therefore these buildings are more likely to deform in comparison to buildings with a deep foundation. So, there is chosen to mainly look at buildings constructed before 1970. The house numbers and the construction years of the buildings of interest are visualized in Figure 3.10. The exact construction year is shown in Table 3.1. The buildings will be compared to analyze the behavior of the street and to check if the code is general enough to work for different buildings.

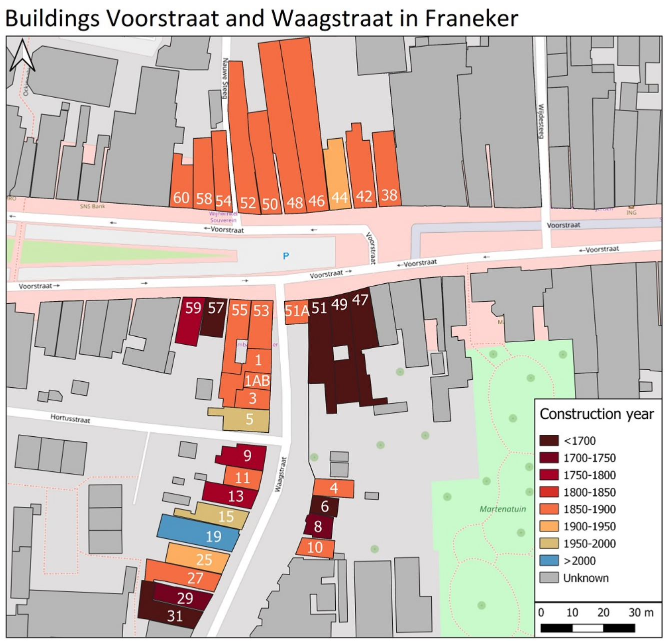


**Figure 3.10.:** *Construction year (from onegeo.co) and house numbers of the buildings in the Voorstraat and Waagstraat, Franeker.*

**Table 3.1.:** *The construction year of the buildings of interest in Franeker (from onegeo.co).*

| Address | City | Construction year | Address | City | Construction year |
|---|---|---|---|---|---|
| Voorstraat 38 | Franeker | 1900 | Waagstraat 1 | Franeker | 1900 |
| Voorstraat 40 | Franeker | 1900 | Waagstraat 1AB | Franeker | 1900 |
| Voorstraat 44 | Franeker | 1905 | Waagstraat 3 | Franeker | 1900 |
| Voorstraat 46 | Franeker | 1900 | Waagstraat 4 | Franeker | 1899 |
| Voorstraat 47 | Franeker | 1650 | Waagstraat 5 | Franeker | 1974 |
| Voorstraat 48 | Franeker | 1900 | Waagstraat 6 | Franeker | 1600 |
| Voorstraat 49 | Franeker | 1658 | Waagstraat 8 | Franeker | 1715 |
| Voorstraat 50 | Franeker | 1900 | Waagstraat 9 | Franeker | 1800 |
| Voorstraat 51A | Franeker | 1900 | Waagstraat 10 | Franeker | 1900 |
| Voorstraat 51 | Franeker | 1658 | Waagstraat 11 | Franeker | 1900 |
| Voorstraat 52 | Franeker | 1900 | Waagstraat 13 | Franeker | 1800 |
| Voorstraat 53 | Franeker | 1900 | Waagstraat 15 | Franeker | 1988 |
| Voorstraat 54 | Franeker | 1900 | Waagstraat 19 | Franeker | 2001 |
| Voorstraat 55 | Franeker | 1900 | Waagstraat 25 | Franeker | 1930 |
| Voorstraat 56 | Franeker | 1900 | Waagstraat 27 | Franeker | 1900 |
| Voorstraat 57 | Franeker | 1615 | Waagstraat 29 | Franeker | 1750 |
| Voorstraat 59 | Franeker | 1796 | Waagstraat 31 | Franeker | , 1700 |
| Voorstraat 60 | Franeker | 1900 | | | |

## 3.3.2. Dancing houses in Amsterdam

The dancing houses in Amsterdam are located beside the Amstel canal. The first houses along this canal were built in 1200. However, due to the peat soil, the buildings subsided. Occasionally, the ground needed to be raised due to the subsidence of the soil. Then there was discovered wooden piles could be used for the foundation of a building. The wooden piles had a depth of 7 meters. However, the strong sand layer is on a depth of approximately 13 meters (De Gans, 2011).

In 1612, the city wall of Amsterdam was shifted. Due to the shift of the city wall, the houses along the Amstel were now also inside the borders of the city. Around the 17th century, the dancing houses were built, which are located at Amstel 100-112 (Gemeente Amsterdam, 2020a). It is expected, the foundation of these buildings consists of wooden piles. However, the piles are most like not deep enough, which caused the buildings to tilt and subside over the years. This makes the buildings look like they are dancing, resulting in the name "dancing houses".

The surface motion map of Amsterdam is shown in Figure 3.11. The InSAR data in this map is from the period between 2017 and 2022. Looking at the surface motion map, it can be seen many dots are green. So, at the Amstel, the displacement is expected to be below 5 millimeters per year. This makes the chance of finding deformation in this area low. Therefore, there is only looked at the current tilt angles of the buildings at the Amstel. The buildings of interest are shown in Figure 3.12 and the construction year of these buildings are shown in Table 3.2.

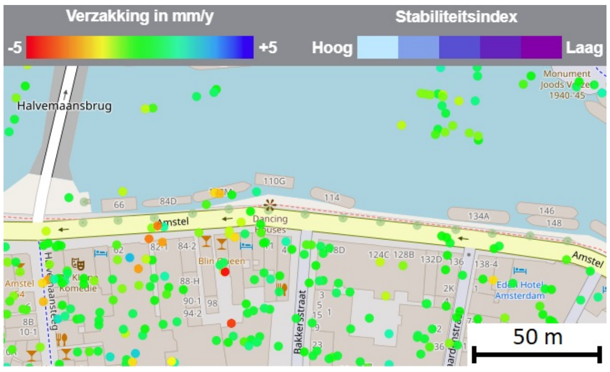


**Figure 3.11.:** *Surface motion map of the Amstel in Amsterdam (SkyGeo, 2022).*

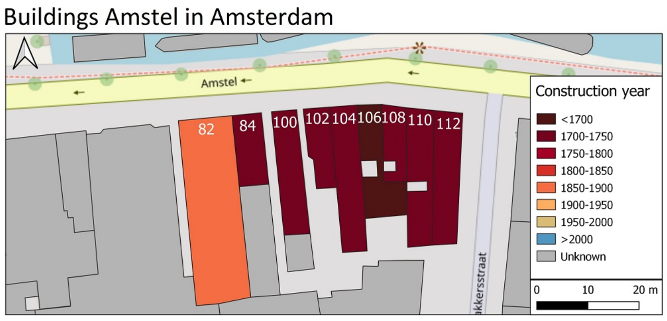


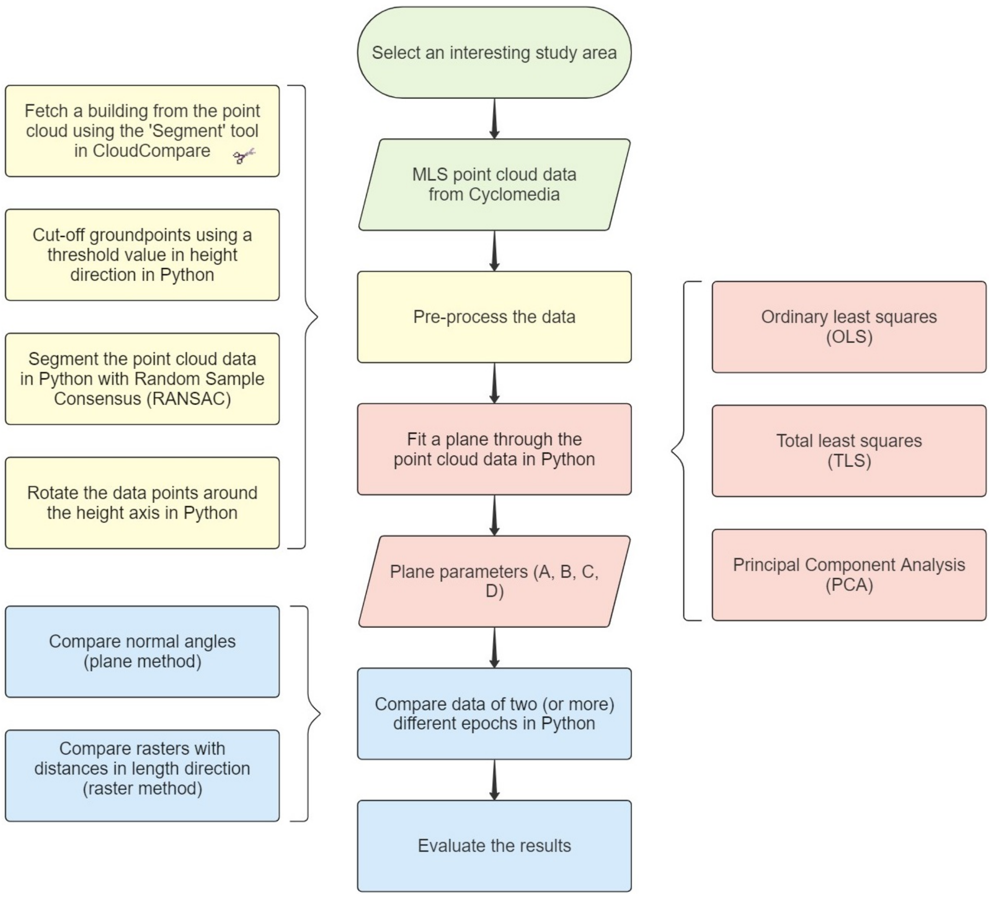
**Figure 3.12.:** *Construction year (from KadastraleKaart.com and Rijksmonumenten.nl) and house numbers of the buildings at the Amstel, Amsterdam.*

**Table 3.2.:** *The construction year of the buildings of interest in Amsterdam (from KadastraleKaart.com and Rijksmonumenten.nl).*

| Address | City | Construction year |
|---|---|---|
| Amstel 82 | Amsterdam | 1898 |
| Amstel 84 | Amsterdam | 1744 |
| Amstel 100 | Amsterdam | 1742 |
| Amstel 102 | Amsterdam | 1710 |
| Amstel 104 | Amsterdam | 1710 |
| Amstel 106 | Amsterdam | 1659 |
| Amstel 108 | Amsterdam | 1725 |
| Amstel 110 | Amsterdam | 1706 |
| Amstel 112 | Amsterdam | 1727 |

# 4. Methodology

This chapter describes the methodology of this research (see Figure 4.1). First, the data is preprocessed in Section 4.1. Then, a plane fitting method is chosen in Section 4.2. Finally, the data of different epochs is compared in Section 4.3. Example results are created with the data of the SNS building (see Figure 4.2). The data from epoch 1 is collected in 2021 and the data from epoch 2 is collected in 2022.



Select an interesting study area
MLS point cloud data from Cyclomedia
Pre-process the data
Fit a plane through the point cloud data in Python
Plane parameters (A, B, C, D)
Compare data of two (or more) different epochs in Python
Evaluate the results
Fetch a building from the point cloud using the 'Segment' tool in CloudCompare
Cut-off groundpoints using a threshold value in height direction in Python
Segment the point cloud data in Python with Random Sample Consensus (RANSAC)
Rotate the data points around the height axis in Python
Ordinary least squares (OLS)
Total least squares (TLS)
Principal Component Analysis (PCA)
Compare normal angles (plane method)
Compare rasters with distances in length direction (raster method)

**Figure 4.1.:** *Workflow of this thesis.*

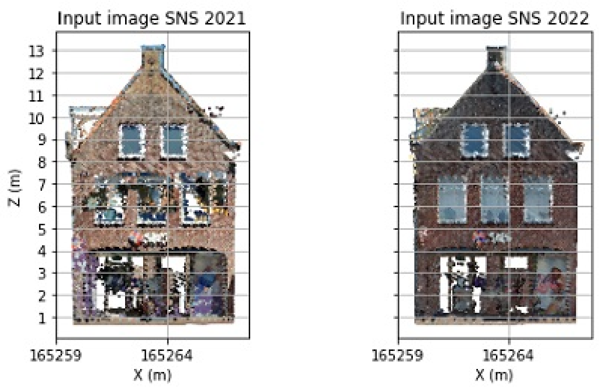


**Figure 4.2.:** *From left to right, side view of the point cloud of the SNS building (Voorstraat 58, Franeker) of 2021 and 2022.*

## 4.1. Preprocessing

After finding an interesting study area, the point cloud data is downloaded and preprocessed. First, a building is fetched from the point cloud using CloudCompare (see Subsection 4.1.1). Secondly, the data is preprocessed using Python. During preprocessing in Python, the data from the LAS file is first transformed to the EPSG:28992 CRS. Then, the ground points are removed from the dataset (see Subsection 4.1.2). After ground removal, segmentation is performed to remove points that are considered to be outliers (see Subsection 4.1.3).

### 4.1.1. Fetch a building from the point cloud

CloudCompare is used to select a specific part of the data. Using CloudCompare, it is possible to merge point clouds of different locations into one big point cloud. It is also possible to cut the data of different epochs by using the 'Segment' tool, as shown in Figure 4.3. It is important, that the planes of two different epochs are created similarly. So, using the 'Segment' tool in CloudCompare, the data of two different epochs need to be cut out the same. In Figure 4.4, an example is given when one dataset contains more points than another one. A plane is fitted through the data using Ordinary Least Squares (OLS) and from Figure 4.4 it can be seen, the plane parameters are slightly different.

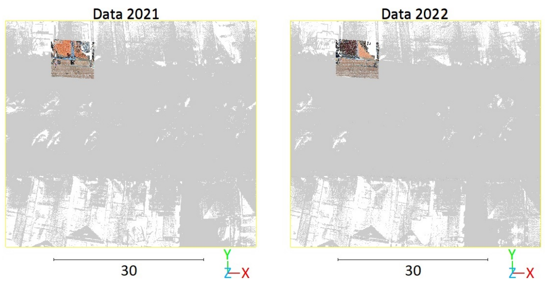
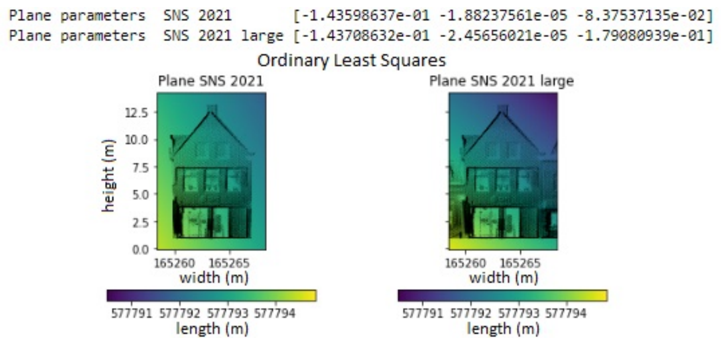


**Figure 4.3.:** *Segmented dataset of a single building (in color) cut from the original dataset (in grey) which is 50 by 50 meters.*



```
Plane parameters  SNS 2021       [-1.43598637e-01 -1.88237561e-05 -8.37537135e-02]
Plane parameters  SNS 2021 large [-1.43708632e-01 -2.45656021e-05 -1.79080939e-01]
```

**Figure 4.4.:** *Comparison of plane fitting with differently segmented point clouds. The left dataset is larger than the right dataset.*

### 4.1.2. Removing ground points from the data

There are two different methods considered that could be used to remove the ground points from a point cloud. The first method uses a threshold value below which the points are considered to be ground points. The second method uses segmentation. After applying the two methods, they are compared and the best one is used for this research.

#### Removal of the ground points using a threshold value

The first method is to remove the points on the ground using a threshold value of the $z$-coordinate which is assumed to be the height of the street in front of the building. This threshold value can be found by creating a histogram, as shown in Figure 4.5. The resulting threshold value depends on three factors: the bin width, the height difference between the bins, and the cutoff value of the histogram.

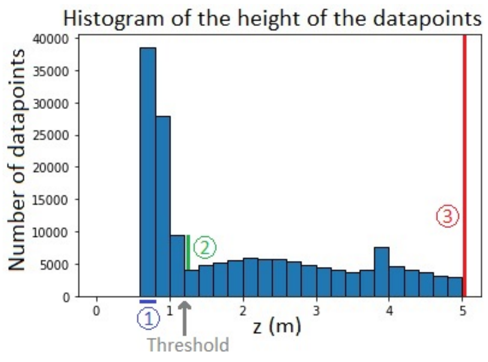


**Figure 4.5.:** *The histogram containing the height of the data points (SNS building 2021) used to determine the height threshold. The resulting threshold value depends on three factors: the bin width (1), the height difference between the bins (2), and the cut-off value of the histogram (3). The cut-off value is set to 5 m because it is expected the ground points are below this height.*

The bin width of the histogram is set to 0.2 meters. It is also possible to use a narrower or a wider bin width. An extremely narrow bin width, such as 0.01 meters, needs to be avoided since a significance level of 0.1 meters is assumed to be sufficient. Besides the significance of the threshold value, the difference in bin height might be too low if the bin width is either too small or too large. Instead of a histogram with a large bin width (above 1 meter), the threshold value could also be visually estimated by using an image of the building containing grid lines.

The maximum height of the ground points is determined by using the height difference between the bins. If a left bar of the histogram is twice as large, the points in this left bar are considered to be ground points. This assumption is based on the density of the point cloud. The density of a point cloud is higher closer to the scanner and the point cloud is less dense further away (Bakula, 2016). The ground points are assumed to be closer to the scanner, so it is assumed the density of the ground points is twice as high as the density of the points on the facade.

Besides the ground points, other things can also cause a peak in the histogram. For example, in the case of a large horizontal surface on the building, like a balcony or a shelter. It is possible that these peaks are also twice as large as the next bin. A cut-off value is introduced to remove these bins from the histogram. This cut-off value is set to 5 meters because it is not expected the ground level is above 5 meters.

For the building shown in Figure 4.2, the threshold value for ground removal is 1.2 meters. From Figure 4.5, it can be seen the bar of 1.2 meters is twice as high as the bar of 1.4 meters. For the dataset from 2021, it means that 36% of the data is removed and for the dataset from 2022, 38% of the data is removed, as shown in Figure 4.6. So, ground points are a large part of the data. Therefore, the ground points can affect the results when they are not removed before plane fitting.

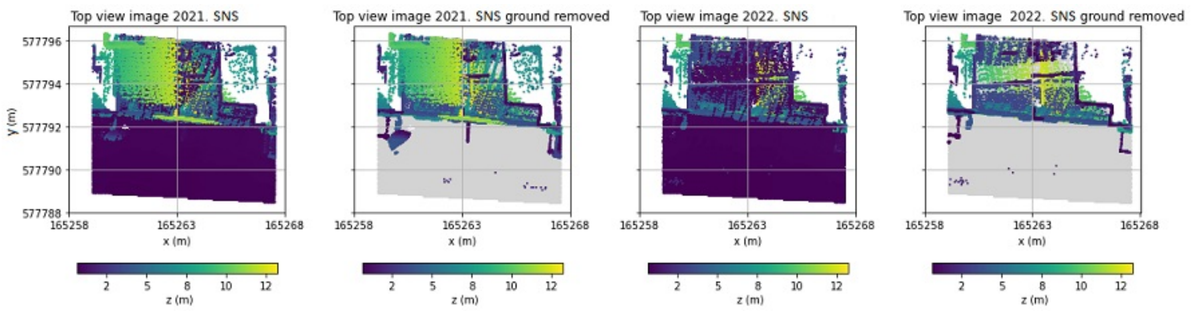


**Figure 4.6.:** *Removal of the ground points in the dataset with a threshold value of 1.2 m. The removed ground points are displayed in grey.*

**Removal of the ground points using segmentation**

Segmentation can be used to classify the points of a point cloud. During segmentation, data is divided into different groups resulting in a group of ground points and a group of non-ground points. The data is segmented using the RANSAC algorithm, as explained in Section 2.5. So, for segmentation, the *segment plane* function in Python is used. This function has the following inputs: three-dimensional data points, *distance threshold*, *ransac n*, and *num iterations*.

In the case of segmenting a ground plane, the *distance threshold* is 0.04 meters, *Ransac n* is 3 points, and *Num iterations* is 1000 iterations. The optimal plane is the plane that goes through the ground points, as shown in Figure 4.7. For the dataset from 2021, 28% of the data is considered to be ground points and this is 30% for the dataset from 2022.

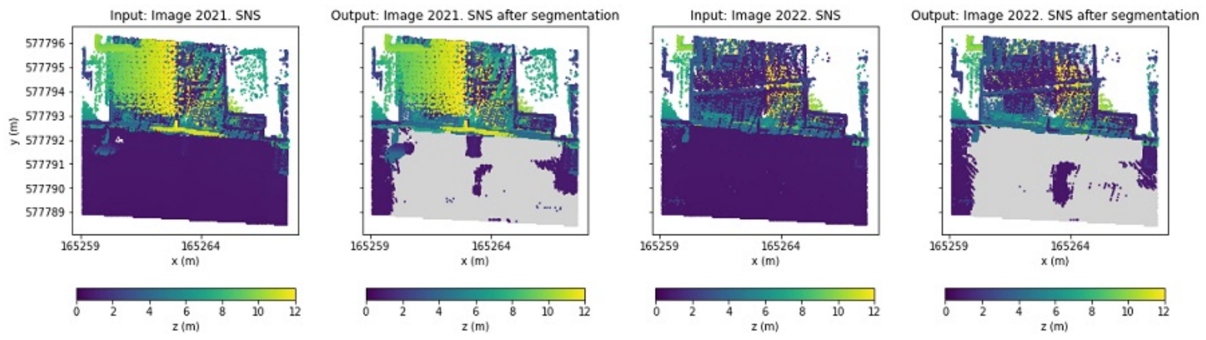


**Figure 4.7.:** *Result of the segment plane function with a threshold value of 0.04 m. The determined ground points are displayed in grey.*

**Comparison of two ground removing methods**

If segmentation is performed before removing the ground points, the *segment plane* function might fit a plane through the ground points instead of the facade as shown in Figure 4.7. Therefore, the ground points must be removed before fitting a plane.

For 2021, 36% of the data is removed while using the threshold method, and 28% of the data is removed while using segmentation. For 2022, 38% of the data is removed while using the threshold method, and 30% of the data is removed while using segmentation. So, the results of removing ground points with a threshold value and removing the ground points with segmentation are slightly different, as shown in Figure 4.8. It can be seen that the threshold method removes more points in front of the building which are assumed to be ground points.

In the example shown in Figure 4.8, the threshold value is assumed to be better than the segmentation method for removing the ground points, because it removes a higher percentage of points. However, the amount of points that are removed depends on the quality of the points. If the density of the ground points is higher, the percentage of removed points increases for the threshold method. Thereby, the threshold method has no chance of accidentally removing a lot of points on the facade instead of the ground points, because only points below a certain height are removed. In the case of the segment method, points on the facade can accidentally be removed if points on the facade are selected as initial points. So in this research, a threshold value will be used to remove the ground points.

### 4.1.3. Segmenting the facade

Segmentation can also be used to determine which points are on the facade of a building. For example, using the *segment plane* function explained in Section 2.5. However, in this case, the optimal plane is the plane that goes through the facade of the building. The most difficult part is to determine when enough points are removed, so choosing the *distance threshold*.

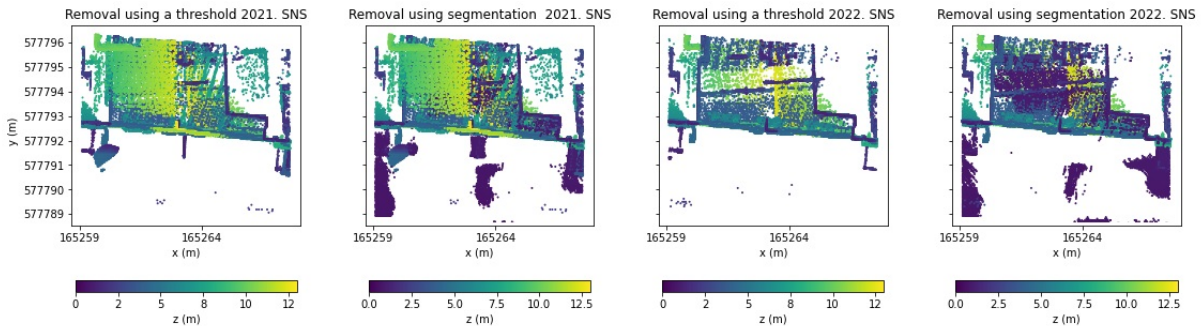


**Figure 4.8.:** *Comparison of two different methods to remove ground points: threshold value vs segment plane function.*

### Choosing the distance threshold value

After removing the ground points with the threshold method, the *distance threshold* is determined for the *segment plane* function. The distance must be set small enough, such that most outliers are removed. Possible points which can be considered to be outliers are flags, sunshades, roof tiles, windows, or doors. Two possible methods are considered to determine the distance threshold value for segmentation. The first method uses a histogram to determine the location of the facade. For the second method, multiple threshold values are tried and compared.

A histogram is constructed as shown in Figure 4.9. This histogram contains the number of points with a certain $y$ (e.g. width) coordinate. The highest peak in this histogram is expected to correspond to the points on the facade of the building. However, looking at the histogram, there are multiple peaks visible. It is difficult to determine which of these peaks corresponds to the facade. This makes it hard to determine which distance threshold value is needed to remove the outliers on the facade (e.g. sunshades, windows, etc.).

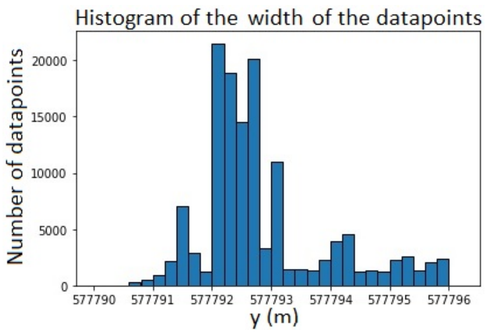


**Figure 4.9.:** *The histogram containing the number of data points with a certain width value (SNS building 2021). The histogram could be used to determine the distance threshold. However, there are too many peaks so it is unsure which peak corresponds to the facade of the building.*

Instead of the histogram, different *distance threshold* values are tried and compared by using a loop in Python. For example, a distance threshold of 2, 4, and 6 centimeters, which are shown in Figure 4.10. In the results of a threshold of 2 centimeters, vertical stripes are visible in the data. This results in a loss of data points of the facade, so a threshold of 2 centimeters is not preferred. For the threshold of 4 centimeters, there are a couple of facade points missing in the output image of 2022. However, this is not as extreme as the stripes when using a threshold of 2 centimeters. Comparing the threshold of 4 centimeters with the threshold of 6 centimeters, it can be seen the points at the top of the doors are removed with the lower threshold value.

When more data is removed, stripes can be seen in Figure 4.10. This is mainly visible for a threshold of 2 centimeters. When measuring the distance between the data points, it can be seen the distance in $y$ direction is around 6 centimeters and the distance in $x$ direction is around 1.5 centimeters. So, the points are not equally distributed, as shown in Figure 4.11 which shows the segmented data in CloudCompare. It is visible the data points are stored with a certain structure (rows and columns). If too much data is removed, this results in stripes as some of the rows and columns will be empty.

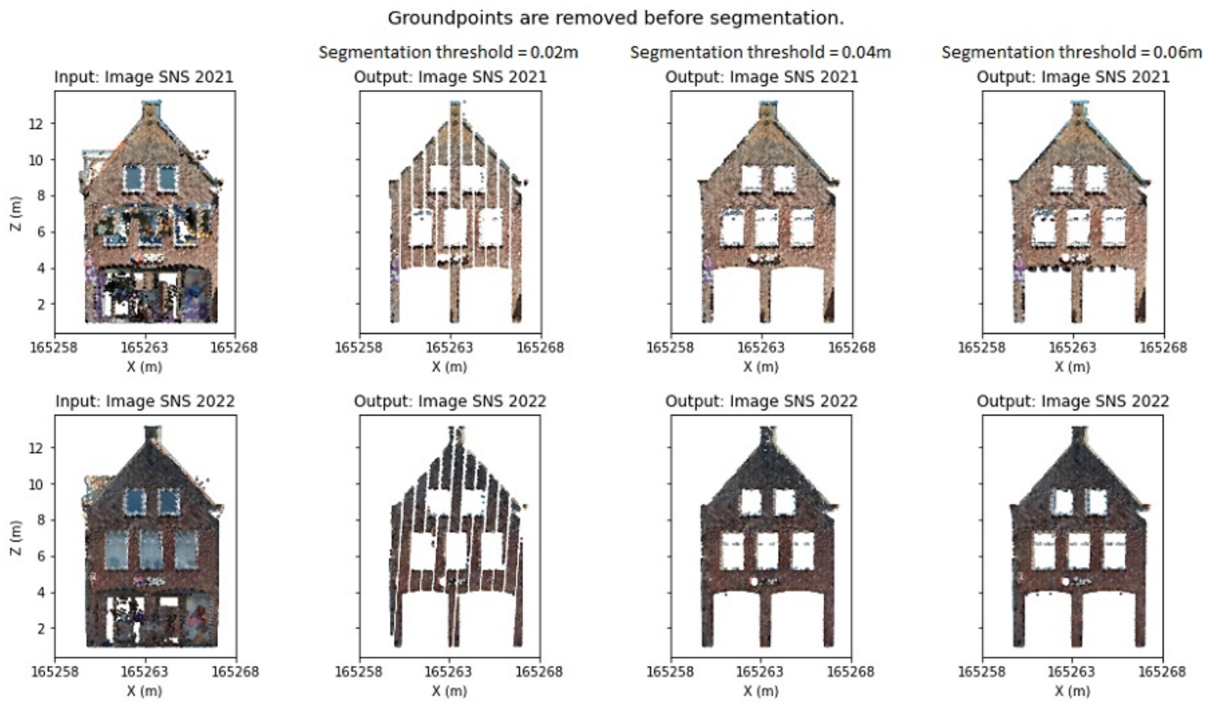


**Figure 4.10.:** *From left to right, comparison of three different distance threshold values, which are 2, 4, and 6 cm. The top row contains the dataset from 2021 and the bottom row the dataset from 2022.*

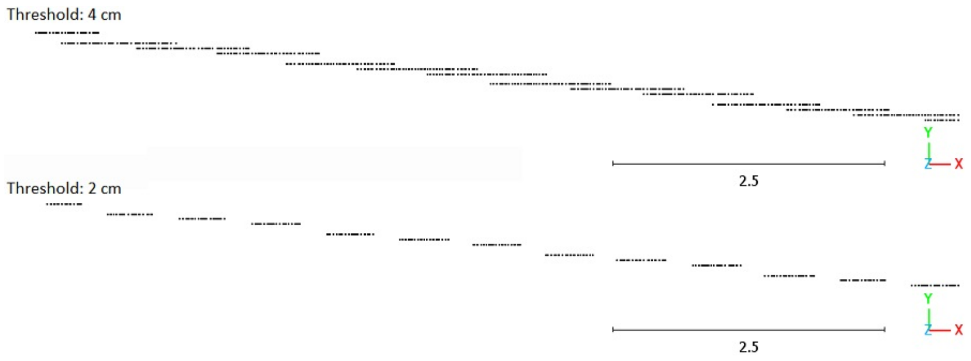


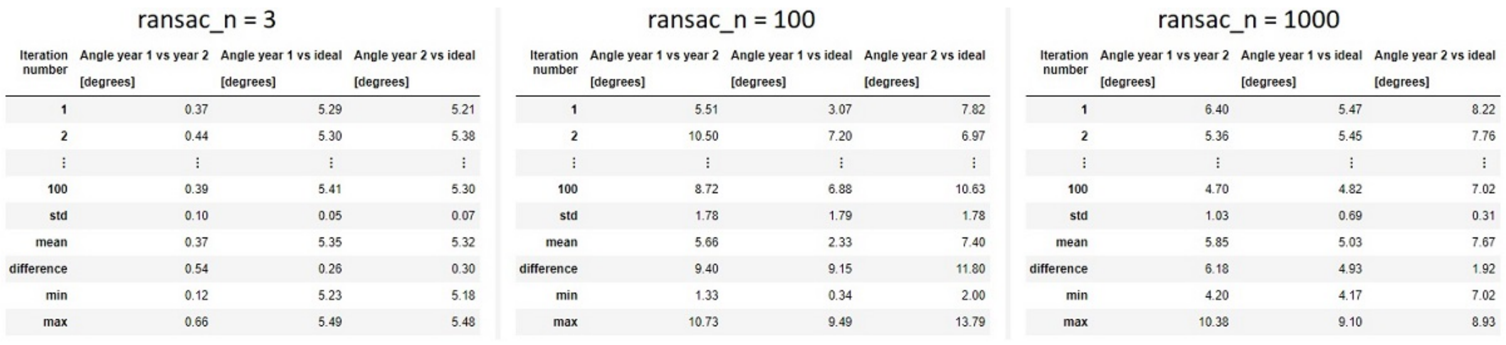
**Figure 4.11.:** *From top to bottom, an example of the segmented SNS dataset of 2021 with a threshold of 4 cm and 2 cm.*

### Limitations of the segment plane function

The *segment plane* function has a downside. Each time the function is run, the results will be slightly different due to the RANSAC function that takes random points as initial values and this can affect the results. Something might seem like deformation, but this may be caused by how the data is segmented. Therefore, the code must be run multiple times to check if the computed angle is deformation or just a large angle because of the different results from the *segment plane* function.

Figure 4.12 shows the results of iterations of the *segment plane* function. From the results, it can be seen a *ransac n* value of 3 will give the smallest difference between the maximum and the minimum obtained angles and the smallest standard deviation compared to a *ransac n* of 100 or 1000 points.

According to the results of the iterations, the smallest angle between the different epochs is 0.1 degrees. However, the *segment plane* function gives a different result each time the function is run, because the RANSAC function takes the initial points randomly. So, in the worst case, the angle between the planes of different epochs is higher than the real angle between the planes. In the case of the example from Figure 4.12, the angle could be 0.5 degrees higher for a different iteration. So, when looking at the results keep in mind the precision of the data is around 0.5 degrees.



ransac_n = 3

| Iteration number | Angle year 1 vs year 2 [degrees] | Angle year 1 vs ideal [degrees] | Angle year 2 vs ideal [degrees] |
|---|---|---|---|
| 1 | 0.37 | 5.29 | 5.21 |
| 2 | 0.44 | 5.30 | 5.38 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 100 | 0.39 | 5.41 | 5.30 |
| std | 0.10 | 0.05 | 0.07 |
| mean | 0.37 | 5.35 | 5.32 |
| difference | 0.54 | 0.26 | 0.30 |
| min | 0.12 | 5.23 | 5.18 |
| max | 0.66 | 5.49 | 5.48 |

ransac_n = 100

| Iteration number | Angle year 1 vs year 2 [degrees] | Angle year 1 vs ideal [degrees] | Angle year 2 vs ideal [degrees] |
|---|---|---|---|
| 1 | 5.51 | 3.07 | 7.82 |
| 2 | 10.50 | 7.20 | 6.97 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 100 | 8.72 | 6.88 | 10.63 |
| std | 1.78 | 1.79 | 1.78 |
| mean | 5.66 | 2.33 | 7.40 |
| difference | 9.40 | 9.15 | 11.80 |
| min | 1.33 | 0.34 | 2.00 |
| max | 10.73 | 9.49 | 13.79 |

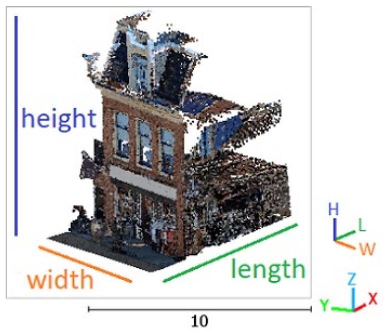
ransac_n = 1000

| Iteration number | Angle year 1 vs year 2 [degrees] | Angle year 1 vs ideal [degrees] | Angle year 2 vs ideal [degrees] |
|---|---|---|---|
| 1 | 6.40 | 5.47 | 8.22 |
| 2 | 5.36 | 5.45 | 7.76 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 100 | 4.70 | 4.82 | 7.02 |
| std | 1.03 | 0.69 | 0.31 |
| mean | 5.85 | 5.03 | 7.67 |
| difference | 6.18 | 4.93 | 1.92 |
| min | 4.20 | 4.17 | 7.02 |
| max | 10.38 | 9.10 | 8.93 |

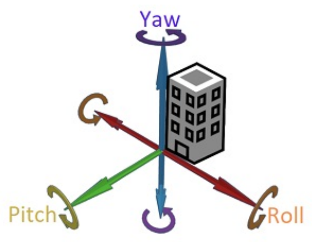
**Figure 4.12.:** *Table containing the results of iteration of the segment plane function with different values for ransac n.*

## 4.1.4. Transformation of the coordinate system

In the original coordinate system, the $y$-coordinate points North and the $x$-coordinate points East. This original coordinate system is transformed into a building-based system (see Section 2.3) containing the width $W$, length $L$, and height $H$ of the building, as shown in Figure 4.13a. The data is transformed to the coordinate system of the building after segmentation to remove a degree of freedom from the dataset. In Figure 4.13b, the three different rotational degrees of freedom are visible. Assuming there is no pitch, there is still a chance of movement in the direction of yaw (torsion) or roll (tilt).



**(a)** *Orientation of the building*



**(b)** *Rotational degrees of the building*

**Figure 4.13.:** *Left, the orientation of the transformed coordinate system W, L and H with respect to the original coordinate system x, y and z (image from CloudCompare). Right, the three possible rotational degrees of freedom of a building.*

When transforming the segmented data of two different epochs with the same transformation matrix, the torsion angle of epoch 1 becomes zero. This makes it possible to compute the angle of torsion between epoch 1 and epoch 2. When the torsion angle is removed from the dataset, only one rotational degree is left, which is rotation in the roll direction indicating tilt. The matrix used to transform the dataset in the yaw direction is discussed in Section 2.3 (Eq. 2.3). Multiplying the data points with this transformation matrix results in a rotated dataset corresponding to a single building.

The rotation angle $\theta_H$ used to transform the data from the original coordinate system to a building-based system is computed using the plane parameters to determine the vectors of the plane. The plane parameters are computed during the segmentation of the data with the *segment plane* function. The angle $\theta_H$ is computed with respect to an ideal plane. The orientation of this ideal plane is discussed in Section 2.2. The plane equation of an ideal plane is:

$$AW + BL + CH + D = 0$$
$$L = D \tag{4.1}$$

So, the plane parameters $A$ and $C$ are equal to zero, and plane parameter $B$ is equal to one. $D$ is equal to the mean length component of the building.

The angle between two vectors of the plane can be computed with (Strang, 2006):

$$\theta = \cos^{-1}\left(\frac{\mathbf{a}.\mathbf{b}}{|\mathbf{a}||\mathbf{b}|}\right) \tag{4.2}$$

Where **a** and **b** represent the vectors. The corresponding values of **a** and **b** are:

$$\mathbf{a} = \begin{bmatrix} B_1 \\ A_1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4.3}$$

Vector **a** is the vector of the data points and vector **b** is the vector of the ideal plane. In vector **a**, $A_1$ and $B_1$ represent the plane parameters obtained by the *segment plane* function of epoch 1.

## 4.2. Choose a plane fitting method

As discussed in Section 2.4, there are three different methods for plane fitting. For each method, the RMSE, covariance matrix $\mathbf{Q}_{yy}$ and quality of planar fitting $\hat{\sigma}^2_m$ are computed for the segmented data of the two epochs of data, see Table 4.1. The advantages and disadvantages of different methods are shown in Table 4.2. One of the three plane fitting methods is chosen to use in this research.

**Table 4.1.:** *The Root Mean Squared Error (RMSE), the covariance matrix $\mathbf{Q}_{yy}$ and the quality of planar fitting $\hat{\sigma}^2_m$ of the segmented data for epoch 1 and epoch 2 for plane fitting with Ordinary Least Squares (OLS), Total Least Squares (TLS) and Principal Component Analysis (PCA).*

| Algorithm | RMSE $[m]$ | | $\mathbf{Q}_{yy}\,[m^2]$ | | $\hat{\sigma}^2_m\,[m^2]$ | |
|---|---|---|---|---|---|---|
| | Epoch 1 | Epoch 2 | Epoch 1 | Epoch 2 | Epoch 1 | Epoch 2 |
| **OLS** | 0.164 | 0.473 | $2.7 * 10^{-2}$ | $2.2 * 10^{-1}$ | $7.8 * 10^{-8}$ | $1.5 * 10^{-7}$ |
| **TLS** | 0.019 | 0.020 | $3.6 * 10^{-4}$ | $3.8 * 10^{-4}$ | $6.6 * 10^{-8}$ | $7.2 * 10^{-8}$ |
| **PCA** | 0.019 | 0.020 | $3.6 * 10^{-4}$ | $3.8 * 10^{-4}$ | $7.1 * 10^{-8}$ | $7.6 * 10^{-8}$ |

**Table 4.2.:** *Advantages and disadvantages of different plane fitting methods (Soudarissanane, 2016): Ordinary Least Squares (OLS), Total Least Squares (TLS), and Principal Component Analysis (PCA).*

| Method | Advantages | Disadvantages |
|---|---|---|
| **OLS** | - Simple to calculate the plane parameters | - Residuals only minimized in length direction<br>- Sensitive to outliers<br>- Asymptotic bound is O(n$^2$log(n)) |
| **TLS** | - Orthogonal distance is minimized<br>- Residuals are independent of the coordinate system | - Limited amount of data can be processed due to complexity bounded by O(mn$^2$) with m=4 |
| **PCA** | - Orthogonal distance is minimized<br>- Residuals are independent of the coordinate system<br>- Method is more efficient than TLS, complexity is O(mn$^2$) with m=3 instead of m=4 | - Sensitive to outliers<br>- Not robust for exceptional cases (e.g. points aligned as a line)<br>- Efficiency depends on data structure |

Comparing the RMSE of the algorithms, OLS performs the worst because the RMSE is the largest. The RMSE is larger because the distance from the data points to the plane is measured in a horizontal direction instead of the orthogonal direction. In addition, the residuals are only minimized in the length direction (Soudarissanane, 2016).

Looking at the quality of the planar parameters $\hat{\sigma}^2_m$, the values in Table 4.1 are lower for TLS than for PCA. However, the quality of the planar parameters was also determined in Subsections 2.4.2 and 2.4.3. When the data is not segmented, the quality is approximately the same for the two different methods[1].

TLS and PCA have the same RMSE and $\mathbf{Q}_{yy}$ according to the table. For both methods, the orthogonal distance is minimized, which is the shortest distance from a point to the plane. However, PCA is more efficient than TLS. For TLS, the input data can not be too large, so there is a limit to the amount of data points that can be processed. If this limit is reached, there is not enough memory to estimate the plane parameters. The amount of memory Python can use depends on the Random Access Memory (RAM) of the used computer. Therefore, the data points are multiplied by their transpose[2]. This matrix multiplication reduces the needed memory.

[1]For TLS the values were $\hat{\sigma}^2_{m,21} = 2.34 * 10^{-4}$ and $\hat{\sigma}^2_{m,22} = 2.43 * 10^{-4}$. For PCA the values were $\hat{\sigma}^2_{m,21} = 2.33 * 10^{-4}$ and $\hat{\sigma}^2_{m,22} = 2.43 * 10^{-4}$

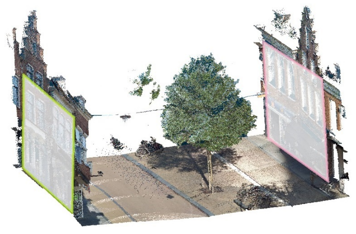
[2]By multiplying a matrix with its transpose, a matrix is created which is comparable to matrix **M** (see Eq. 2.5), explained in 2.4.1. For example, consider a dataset with 20 data points that are three-dimensional. The data is stored in a 5x3 matrix A. So, matrix multiplication $A^T A$ results in a 3x3 matrix.

Using PCA, less memory is needed. PCA is more efficient because this method estimates only three parameters $(A, B, C)$ instead of four. The fourth parameter $(D)$ is computed using the data points in combination with the parameters that are already estimated. So, in this research PCA is used to estimate the plane parameters and the corresponding plane equation.

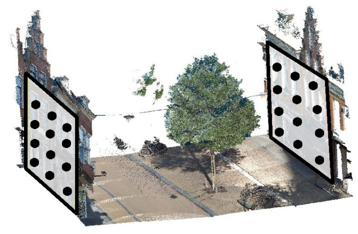
Instead of PCA, Robust Principal Component Analysis (RPCA) could be applied. As stated before, PCA is sensitive to outliers and suited for points that are aligned as a line. However, RPCA is not sensitive to outliers and noise in the data (Nurunnabi, 2012). In addition, RPCA can also be applied to nonlinear data (Lyu, 2019). However, in this research PCA is used because the outliers are removed from the dataset before applying plane fitting.

## 4.3. Compare data of different epochs

Different methods can be used to compare the data of different epochs. For example the plane method (see Subsection 4.3.1) and the raster method (see Subsection 4.3.2). In Figure 4.14, an example result is given for the different methods.



**(a)** *Plane method*



**(b)** *Raster method*

**Figure 4.14.:** *Example results of possible methods to analyze deformation using LiDAR data.*

### 4.3.1. Plane method to determine torsion and tilt angles

For the plane method, a building is modeled by fitting a plane through the facade. This fitted plane is compared with the ideal plane, which is shown in Figure 4.15, by determining the torsion and tilt angles. The torsion and tilt angles can be presented in a table, but they can also be plotted on a map where colors indicate the angle size.

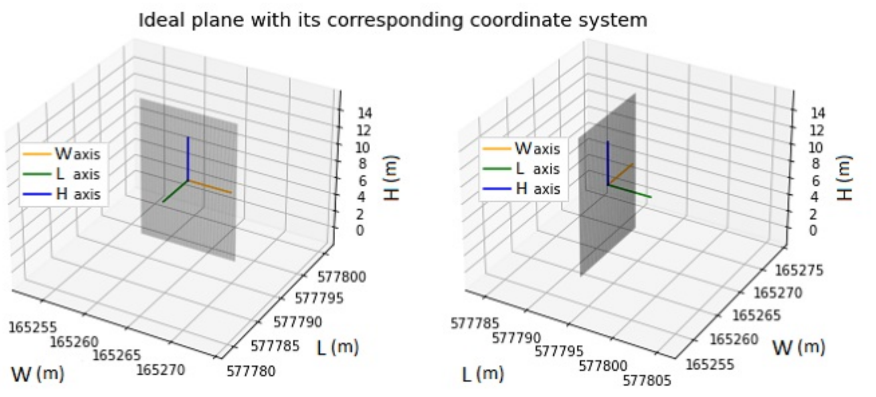


**Figure 4.15.:** *Example of an ideal plane with its corresponding coordinate system.*

Before plane fitting, the ground points are removed and the data is segmented using RANSAC. The outliers are removed from the data, which are points that can cause uncertainty. For example, the street in front of the building (also known as ground points), windows, doors, or objects attached to the building (e.g. a sunshade or a flag). After outlier removal, a plane is fitted through the data points using Principal Component Analysis (PCA) (see Subsection 2.4.3). Then, the plane parameters are used to check if torsion or tilt occurs.

The estimated planes through the facade can be compared using the vectors of the plane, which can be obtained by using the plane parameters $A, B, C, D$. By computing the difference in angle between the vectors, the torsion and tilt angle can be determined (see Subsection 4.1.4). The torsion angle is a rotation around the height axis of the building. So, in the case of torsion, the left side of the building moves in a different direction than the right side. The tilt angle is a rotation around the width axis of the building. So, in the case of tilt, the top of the building can either lean forward or backward.

Different torsion and tilt angles can be computed when there are two epochs of data available. The three possible angles are the angle of the plane of epoch 1 with respect to the ideal plane (epoch 1 vs ideal), the angle of the plane of epoch 2 with respect to the ideal plane (epoch 2 vs ideal), and the difference in angle between the plane of epoch 1 and 2 (epoch 1 vs epoch 2).

When the computed angle is high (e.g. 1 degree), it might indicate that deformation is present. This is proved by an example of the SNS building. The SNS building has a height of approximately 12 meters (so $H = 12$). The facade of this building is manually rotated with an angle of 1 degree around the $W$ axis of the building. This is done with the 'Apply transformation' tool in CloudCompare. The rotation angle is set to 1 degree and the rotation axis is the $x$-axis (width of the building). After rotation, a translation is applied with the same tool. Due to the translation, the dataset without and with a tilt angle get the same bounding box center. The result is a dataset with a tilt angle of 1 degree (so $\theta = 1$) which can be compared with the dataset without a tilt angle.

Using the sine ratio, it is possible to compute the difference in distance at the top of the building $\Delta$ which is shown in Figure 4.16. The change in the distance at the top of the building is computed between the dataset of the facade without a tilt angle and the dataset of the facade with a tilt angle of 1 degree using the following equation:

$$\Delta = \sin(\theta) * H = \sin(1 \text{ [degree]}) * 12 \text{ [m]} = 0.2 \text{ [m]} = 20 \text{ [cm]} \tag{4.4}$$

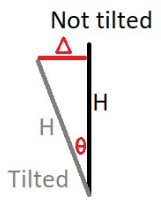


**Figure 4.16.:** *Visualization of the difference in distance at the top of the building ($\Delta$). $H$ represents the height of the building and $\theta$ is the tilt angle of the building.*

According to the computation, the difference in distance at the top of the building will be 20 centimeters. This can be checked by constructing a raster of the data as shown in Figure 4.17. Looking at the raster containing the difference in length $L$ between the facade without and with tilt, it can be seen a tilt angle of 1 degree indeed results in a change in distance of 20 centimeters (according to the red color at the top of the building in Figure 4.17).

Besides the comparison of the data without and with a tilt angle, it is also possible to compare the data of epoch 1 with epoch 2. The vectors of the torsion and tilt angles are shown in Figure 4.18. For the computation of the angles, the vectors are considered to be two-dimensional instead of three-dimensional. So, the torsion angle represents the angle of rotation around the $h$ axis, not considering the movement in $h$ direction (all three vectors have the same value for the height component). The tilt angle represents the angle of rotation around the $w$ axis, not considering the movement in $w$ direction (all three vectors have the same value for the width component).

Only two dimensions are considered, because then there is only one degree of freedom, so the computed angle will only be in one specific direction. The result of the computation is shown in Table 4.3. The first angle is the angle between the vector of epoch 1 with respect to the vector of the ideal plane, the second angle is the angle between the vector of epoch 2 with respect to the vector of the ideal plane, and the third angle is the angle between the vector of epoch 1 with respect to the vector of epoch 2. So, the third angle represents the difference in angle between the two epochs.

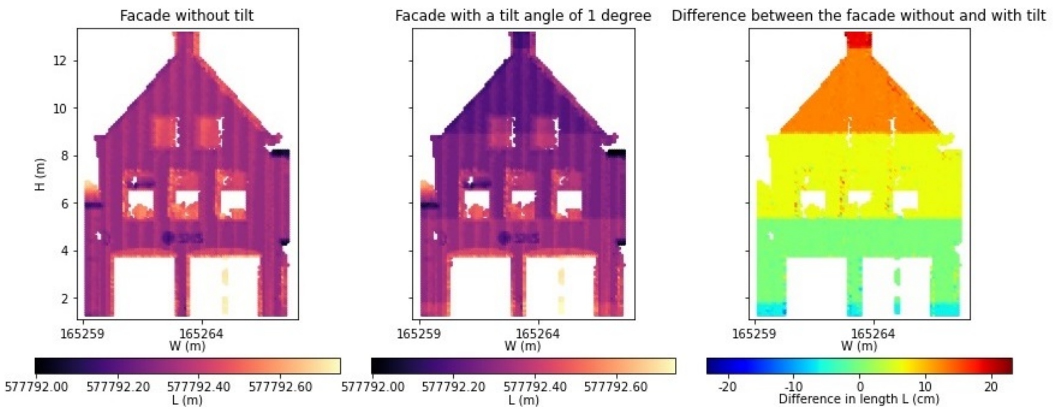


**Figure 4.17.:** *From left to right, the raster of the facade of the SNS building in 2021 without tilt, the raster of the facade of the SNS building in 2021 which is tilted manually by an angle of 1 degree, and the difference between these two rasters.*

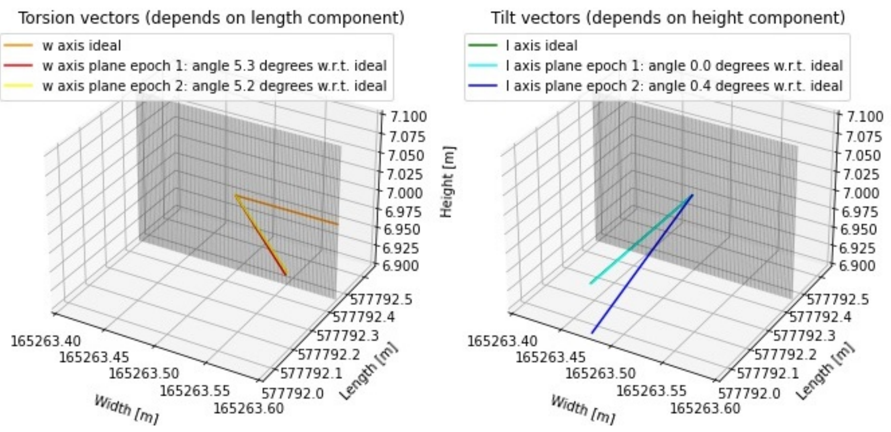


**Figure 4.18.:** *Possible vectors of the planes. The displayed vectors are the vector of the ideal plane, the vector of the plane of epoch 1, and the vector of the plane of epoch 2. The displayed plane is the ideal plane. The displayed torsion vector only depends on the length component, the width and height component are constant for all three vectors. The displayed tilt vector only depends on the height component, the width and length components are constant for all three vectors.*

Using the computed torsion and tilt angles, the data can be rotated using a transformation matrix. The transformation matrix for the torsion angle (rotation around the $H$ axis) is (Popescu, 2014):

$$C_{torsion} = \begin{bmatrix} \cos(\theta_{torsion}) & -\sin(\theta_{torsion}) & 0 & tx \\ \sin(\theta_{torsion}) & \cos(\theta_{torsion}) & 0 & ty \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

The transformation matrix for the tilt angle (rotation around the $W$ axis) is (Popescu, 2014):

$$C_{tilt} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_{tilt}) & -\sin(\theta_{tilt}) & ty \\ 0 & \sin(\theta_{tilt}) & \cos(\theta_{tilt}) & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

The transformation matrix for the torsion angle is used to remove the torsion angle from the data, and the transformation matrix for the tilt angle is used to remove the tilt angle from the data.

The torsion $\theta_{torsion}$ and tilt $\theta_{tilt}$ angles are computed with the equation mentioned in Subsection 4.1.4 (Eq. 4.2), and $t_x$, $t_y$ and $t_z$ are the mean of the rotated data points subtracted by the mean of the original data points. So $t_x$, $t_y$ and $t_z$ are

- $t_x$ is the mean of the rotated data point $(W_i)_{i=0...n-1}$, subtracted by the mean of $(x_i)_{i=0...n-1}$
- $t_y$ is the mean of the rotated data point $(L_i)_{i=0...n-1}$, subtracted by the mean of $(y_i)_{i=0...n-1}$
- $t_z$ is the mean of the rotated data point $(H_i)_{i=0...n-1}$, subtracted by the mean of $(z_i)_{i=0...n-1}$

**Table 4.3.:** *Example of possible torsion and tilt angles for the SNS building (Voorstraat 58, Franeker). A one in the subscript corresponds to a parameter of epoch 1 and a two in the subscript corresponds to a parameter of epoch 2.*

| | Torsion angle [degrees] | | | Tilt angle [degrees] | | |
|---|---|---|---|---|---|---|
| | **a** | **b** | $\theta_{torsion}$ | **a** | **b** | $\theta_{tilt}$ |
| **Epoch 1 vs ideal** | $[B_1, A_1]$ | [1,0] | -5.3 | $[B_1, C_1]$ | [1,0] | 0 |
| **Epoch 2 vs ideal** | $[B_2, A_2]$ | [1,0] | -5.2 | $[B_2, C_2]$ | [1,0] | -0.4 |
| **Epoch 1 vs epoch 2** | $[B_1, A_1]$ | $[B_2, A_2]$ | 0.1 | $[B_1, C_1]$ | $[B_2, C_2]$ | -0.4 |

Applying the transformation matrices above, the data is rotated. Figure 4.19 shows the original dataset, the dataset rotated with the torsion angle and the dataset rotated with the tilt angle. Besides the data points, the corresponding planes are plotted.

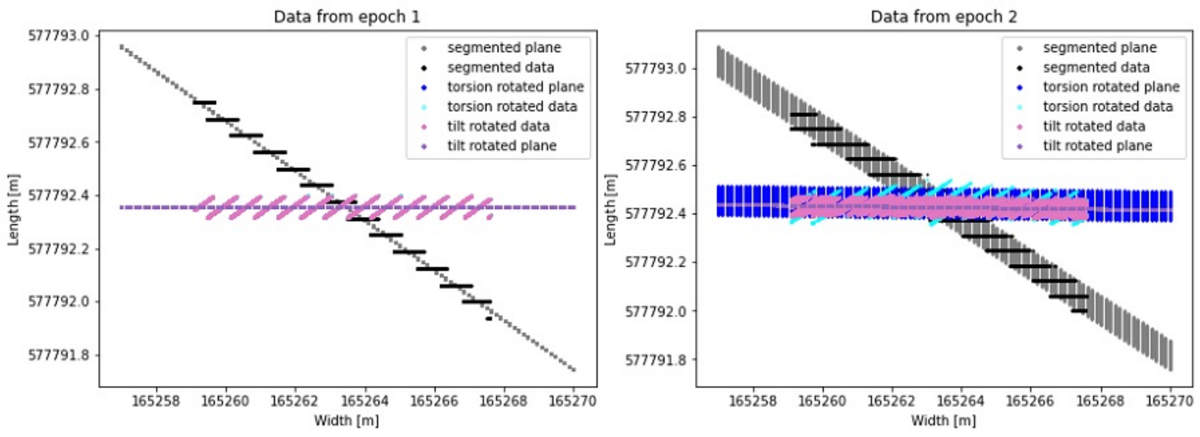


**Figure 4.19.:** *From left to right, the data points and planes of the original and the transformed data for epoch 1 and 2.*

### Quality of the estimated planes

Two methods can be used to determine the quality of the estimated plane. The first method selects three data points on the facade of a building using Street Smart. These three points are used to estimate a plane and this plane is compared with the plane obtained with the PCA algorithm. This method can also be used to verify the plane estimated using PCA. The second method runs the code multiple times and checks if the found torsion and tilt angles are similar.

For the first method, three points are selected using the 'location' tool in Street Smart. With these three points $((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3))$, a plane equation is constructed (Kurgalin, 2021):

$$\begin{bmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{bmatrix} = 0$$

$$\begin{gathered} A_3 = ((y_2 - y_1) * (z_3 - z_1)) - ((y_3 - y_1) * (z_2 - z_1)) \\ B_3 = -((x_2 - x_1) * (z_3 - z_1)) - ((x_3 - x_1) * (z_2 - z_1)) \\ C_3 = ((x_3 - x_1) * (y_2 - y_1)) - ((x_2 - x_1) * (y_3 - y_1)) \\ D_3 = A_3 * -x_1 + B_3 * -y_1 + C_3 * -z_1 \\ A_3 x + B_3 y + C_3 z + D_4 = 0 \end{gathered} \tag{4.7}$$

The selected points and the corresponding plane equations are shown in Table 4.4. The selected points are visualized in Figure 4.20.

**Table 4.4.:** *Three manually selected points on the facade and the corresponding plane Equation.*

| | Epoch 1 | | | Epoch 2 | | |
|---|---|---|---|---|---|---|
| | X (m) | Y (m) | Z (m) | X (m) | Y (m) | Z (m) |
| **Point 1 (top)** | 165263.34 | 577792.38 | 10.37 | 165263.34 | 577792.51 | 10.37 |
| **Point 2 (left)** | 165260.10 | 577792.66 | 3.20 | 165260.10 | 577792.71 | 3.20 |
| **Point 3 (right)** | 165266.80 | 577792.04 | 2.60 | 165266.80 | 577792.09 | 2.60 |
| **Plane Equation** | $-4.61X-49.98Y+0.13Z+2.964*10^7=0$ | | | $-4.57X-49.98Y+0.67Z+2.963*10^7=0$ | | |

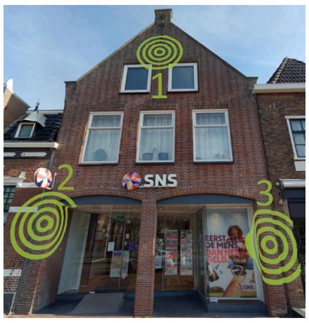


**Figure 4.20.:** *Location of the three points on the facade mentioned in Table 4.4.*

Now, the angles of the plane constructed with the three points are compared to the angles of the plane constructed using PCA. The computed angles are shown in Table 4.5 and the vectors of the torsion and tilt angles are shown in Figure 4.18. So, the first two rows are the angles with respect to the ideal plane and the last row contains the angle between the two epochs. From the table, it can be concluded the torsion angles are quite similar, but the tilt angles are different. The large difference can occur due to the points that are selected as the random points on the facade, or due to the way the data is segmented which influences the input data used to estimate the plane parameters with PCA.

**Table 4.5.:** *Comparison of the difference in torsion and tilt angles of a plane fitting using either PCA or 3 manually selected points on the facade.*

| | Torsion angle [degrees] | | Tilt angle [degrees] | |
|---|---|---|---|---|
| | $\theta_{torsion,PCA}$ | $\theta_{torsion,points}$ | $\theta_{tilt,PCA}$ | $\theta_{tilt,points}$ |
| **Epoch 1 vs ideal** | 5.3 | 5.3 | 0.0 | 0.2 |
| **Epoch 2 vs ideal** | 5.2 | 5.2 | 0.4 | 0.8 |
| **Epoch 1 vs epoch 2** | 0.1 | 0.1 | 0.4 | 0.6 |

To check if the difference in result depends on the random points which are chosen, three different points[3] are selected for epoch 2. This results in a torsion angle of 5.1 degrees and a tilt angle of 0.6 degrees. So, it can be concluded the difference in the result between PCA and 3 random points depends on which random points are selected.

The second method to check the quality of the torsion and tilt angles is to calculate the angles multiple times. Each time the *segment plane* function is run, a different result is generated. This results in different input data for the plane fitting function, leading to a different plane equation. The torsion and tilt angles use the plane parameters from the plane equation, so a different output of the *segment plane* function can result in a different torsion or tilt angle.

To check the influence of the *segment plane* function on the computed angles, the *segment plane* function is run 500 times. For each time, the plane parameters are estimated. Using these plane parameters, the torsion and tilt angles are determined. The result of these calculations is shown in Table 4.6. The maximum range is approximately 0.6 degrees. So, when computing the angles, the angle could in reality be either 0.5 degrees higher or lower than calculated.

The precision of the data is the mean plus or minus the range of the data which is $\pm$ 0.6 degrees. The accuracy of the computed angles is 0.1 degrees because the accuracy is the standard deviation of the data. So, it is expected the difference in angle between two epochs needs to be at least 1 degree for the algorithm to divide deformation from a deviation occurring in the computation of an angle.

### 4.3.2. Raster method to visualize distances

Local deformation patterns can be analyzed by constructing a raster of the data. This raster (or grid) is created using the width and the height of the building, as shown in Figure 4.21. The location of the raster cells (width and height of the raster points) are the same for different epochs making it easier to compare the rasters. The value assigned to the raster cell is the distance in length direction.

[3]Point 1: (165262.87, 577792.53,11.15), Point 2: (165260.30, 577792.70,4.84), Point 3: (165266.83, 577792.08,1.43)

**Table 4.6.:** *Computation of the torsion and tilt angles after performing the segment plane function multiple times. In this Table, the letter E represents epoch.*

| | Torsion angle [degrees] | | | Tilt angle [degrees] | | |
|---|---|---|---|---|---|---|
| Iteration | E1 vs ideal | E2 vs ideal | E1 vs E2 | E1 vs ideal | E2 vs ideal | E1 vs E2 |
| **1** | 5.37 | 5.38 | 0.01 | 0.03 | 0.51 | 0.48 |
| **...** | ... | ... | ... | ... | ... | ... |
| **500** | 5.39 | 5.26 | 0.13 | 0.05 | 0.40 | 0.36 |
| **Max** | 5.47 | 5.49 | 0.26 | 0.23 | 0.63 | 0.57 |
| **Min** | 5.21 | 5.12 | 0.00 | 0.00 | 0.07 | 0.02 |
| **Range** | 0.26 | 0.37 | 0.26 | 0.23 | 0.56 | 0.55 |
| **Mean** | 5.35 | 5.32 | 0.07 | 0.05 | 0.36 | 0.31 |
| **Std** | 0.05 | 0.07 | 0.05 | 0.04 | 0.10 | 0.10 |

This distance can be determined by taking the mean of the 10 nearest neighbors of the raster cell location. If the distance between the raster cell and the neighbors is larger than the size of the raster cell, in either width or height direction, the mean will not be calculated because there are not enough data points inside this raster cell.

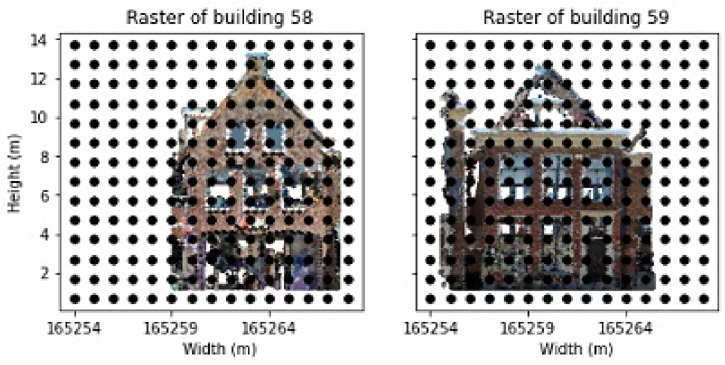


**Figure 4.21.:** *Example of a constructed raster (no values are assigned to the raster cells).*

Besides the location of the raster points, the different rasters will have the same point density. In the original point cloud, one building can contain more points than another one. Due to the construction of the raster, the distance between the points is equal, which also makes it easier to compare data from different epochs. Thereby, while using a raster it is easier to see deformation patterns on the facade instead of using the whole point cloud. While using the whole point cloud, there are more data points which could make it more difficult to interpret the data.

It is possible to use either one building or two opposite buildings for the raster method. When using one building, the raster values will be the location of the facade of this building. In the case of two opposite buildings, the raster values will be the relative distance between the facades of the buildings. However, the raster containing the distance between two opposite buildings is not easy to interpret, so there is only looked at the difference between two epochs for a single building.

To analyze deformation over time, rasters of different epochs are compared with each other. The raster of epoch 2 is subtracted from the raster of epoch 1, resulting in a raster containing the deformation. If the raster cell value is zero, the location of the point is constant and not deforming over time.

An example of a possible raster is shown in Figure 4.22. The colors of the first and second raster represent the mean length value of the 10 closest points of a single epoch. The third raster shows the difference in length between the two epochs. From the image, it can be seen the difference between the dataset is approximately 8 centimeters on the facade of the building. When looking at the top view of the two datasets, this difference is also visible as shown in Figure 4.23.

Looking at the data in Table 4.4, it can be seen the difference in the length component at the top of the building is approximately 10 centimeters, and at the left and right pillar, the difference in the length component is 5 centimeters. This is also visible in the raster image. So, it is expected the constructed raster makes sense. Thereby, the difference in distance is also visible in Figure 4.23.

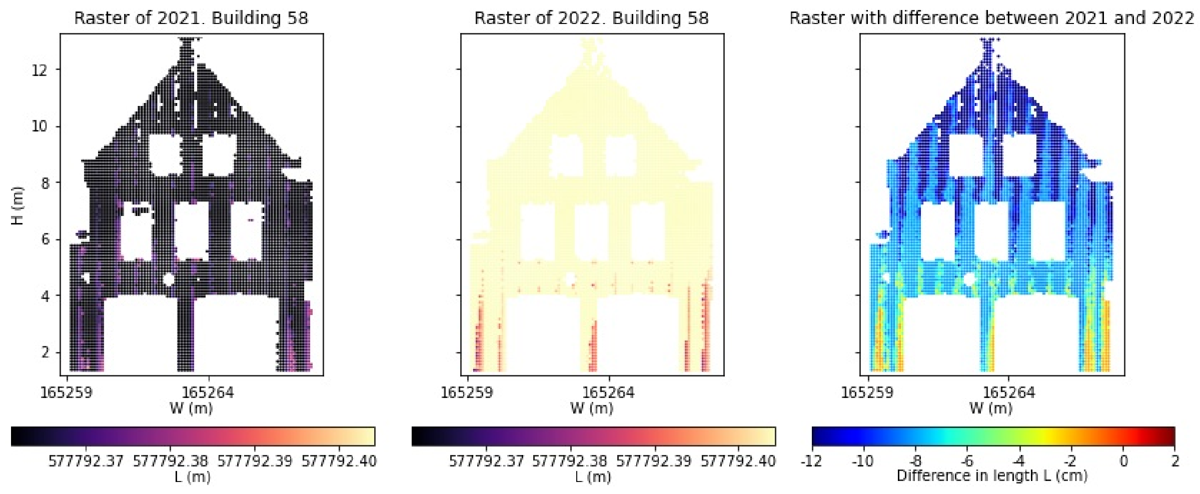


**Figure 4.22.:** *Example of a possible raster for the SNS building in Franeker.*

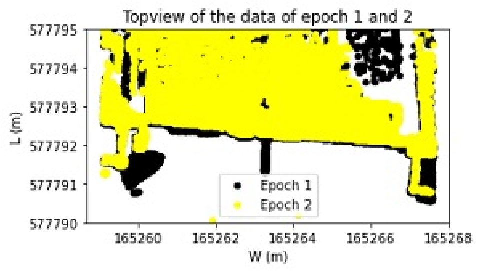


**Figure 4.23.:** *Top view of the SNS dataset of the two different epochs.*

There are two possible explanations for the difference in the length coordinate. The first option is that the facade of the building has deformed between two epochs. Another explanation is an error during data processing, which could result in a misalignment of the data.

Besides the misalignment of the data, the top part of the building has a higher difference in distance than the bottom part. So, according to the raster, the top of the building has tilted backward. The difference in distance between the bottom part and the top part is approximately 4 centimeters. So, according to the raster, the tilt angle is (see Figure 4.16):

$$\theta = \arcsin \frac{\Delta}{H} = \arcsin \frac{0.04[\text{m}]}{12[\text{m}]} = 0.2[\text{degrees}] \tag{4.8}$$

This angle is smaller than the difference in tilt angle computed using the planes of epoch 1 and 2 estimated with PCA, which resulted in an angle of 0.4 degrees (see Table 4.5).

# 5. Results study area 1: Franeker

In this chapter, the results of the study area in Franeker are described. In Franeker, two different streets were analyzed: Waagstraat and Voorstraat. For these streets, two epochs of data are available. The data of epoch 1 is captured on the 1st of June 2021 and the data of epoch 2 is captured on the 16th of May 2022.

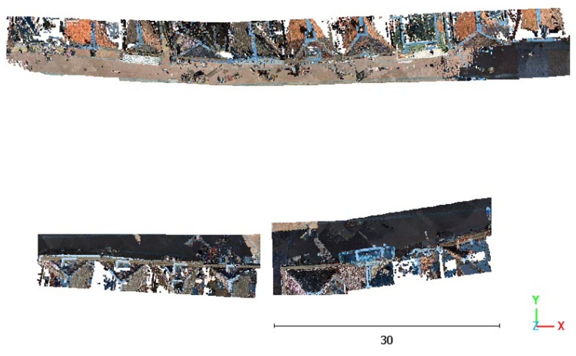
The data is preprocessed in Section 5.1. In Section 5.2, the torsion and tilt angles are computed and displayed. The buildings are examined further by creating a raster of the buildings containing the distance in the length direction in Section 5.3. A summary of the results is given in Section 5.4.

## 5.1. Preprocessing of the data

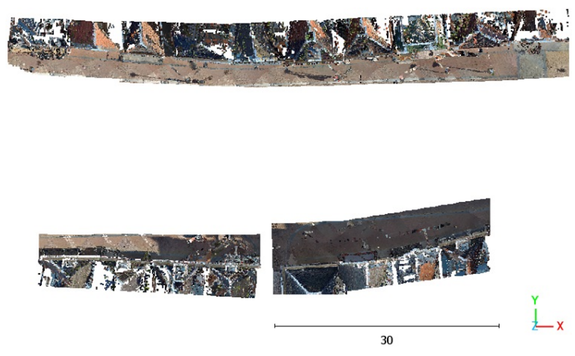
As discussed in Section 4.1, there are four different preprocessing steps. First, a building is fetched from the point cloud in Subsection 5.1.1. Then the ground points are removed from the data in Subsection 5.1.2. After removing the ground points, the data is segmented in subsection 5.1.3. Finally, the data is transformed from the original coordinate system to the building-based coordinate system in Subsection 5.1.4.

### 5.1.1. Fetch a building from the point cloud

The buildings of interest are cut out from the dataset using the 'Segment' tool in CloudCompare. This results in a separate las-file for each building. These las-files are imported in Python to process the data. The cut-out point clouds of the Voorstraat are shown in Figure 5.1 and the point clouds of the Waagstraat are shown in Figure 5.2.

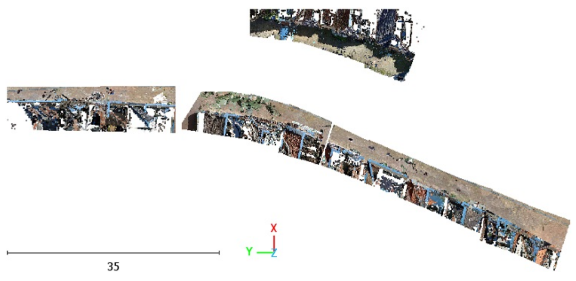


**(a)** *Epoch 1*

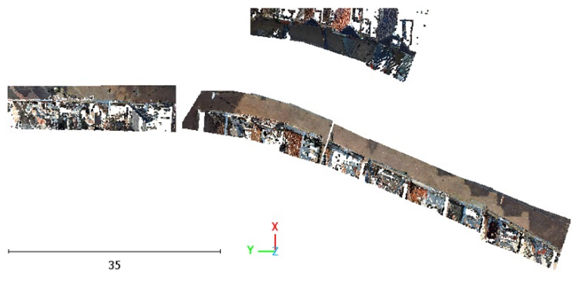


**(b)** *Epoch 2*

**Figure 5.1.:** *Point cloud data of the Voorstraat in Franeker of epoch 1 and epoch 2.*



**(a)** *Epoch 1*



**(b)** *Epoch 2*

**Figure 5.2.:** *Point cloud data of the Waagstraat in Franeker of epoch 1 and epoch 2.*

### 5.1.2. Removing the ground points from the data

The point clouds are imported in Python. When the data is imported in Python, the ground points need to be removed because they can influence the result of the segmentation function. The ground points are removed by using a histogram to determine the threshold below which points are considered to be ground points. Each building has a different threshold value for the removal of ground points which is determined using a histogram. The threshold values for each building are shown in Table 5.1.

**Table 5.1.:** *Result of the computed threshold for removing the ground points of the Voorstraat and the Waagstraat. Marked in yellow, is a high threshold value of 4 meters for Voorstraat 50. After examining the histogram and adjusting the range to 3 meters, the threshold value of Voorstraat 50 is reduced to 1 meter.*

| Building | Threshold [m] | Building | Threshold [m] |
|---|---|---|---|
| Voorstraat 38 | 1.0 | Waagstraat 1 | 1.0 |
| Voorstraat 40 | 1.0 | Waagstraat 1AB | 0.8 |
| Voorstraat 44 | 1.0 | Waagstraat 3 | 1.0 |
| Voorstraat 46 | 1.0 | Waagstraat 4 | 0.6 |
| Voorstraat 47 | 1.2 | Waagstraat 5 | 1.0 |
| Voorstraat 48 | 1.0 | Waagstraat 6 | 0.6 |
| Voorstraat 49 | 1.2 | Waagstraat 8 | 0.6 |
| Voorstraat 50 | 4.0 $\to$ 1.0 | Waagstraat 9 | 1.0 |
| Voorstraat 51A | 1.2 | Waagstraat 10 | 0.6 |
| Voorstraat 51 | 1.2 | Waagstraat 11 | 0.8 |
| Voorstraat 52 | 1.0 | Waagstraat 13 | 1.0 |
| Voorstraat 53 | 1.2 | Waagstraat 15 | 0.8 |
| Voorstraat 54 | 1.2 | Waagstraat 19 | 0.8 |
| Voorstraat 55 | 1.2 | Waagstraat 25 | 0.8 |
| Voorstraat 58 | 1.2 | Waagstraat 27 | 0.6 |
| Voorstraat 57 | 1.2 | Waagstraat 29 | 1.2 |
| Voorstraat 59 | 1.2 | Waagstraat 31 | 0.6 |
| Voorstraat 60 | 1.0 | | |

Looking at the data in the table containing the threshold values, it is clear that the majority of the threshold values range between 0.6 and 1.2 meters. However, there is one outlier: the threshold value for Voorstraat 50 (marked in yellow in the table). For this building, the threshold value is equal to 4 meters. This is odd because it is expected the threshold value would be below 1.2 meters, just like the other buildings on this street. In Figure 5.3, the histogram containing the height of building 50 is shown. Besides the histogram, there is also looked at the image and raster of the building to discover what causes the peaks at a height between 3.6 and 4 meters.

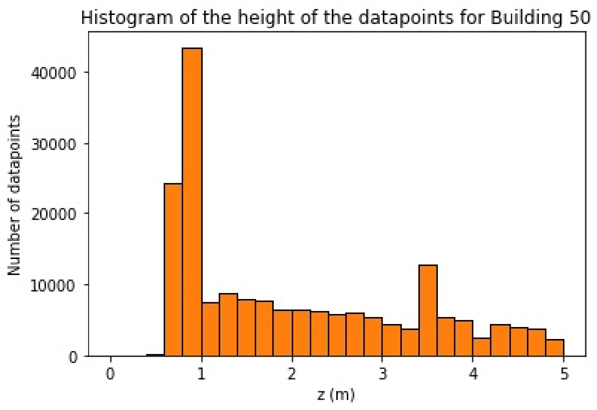


**Figure 5.3.:** *Histogram of the height of the data points of Voorstraat 50.*

Looking at the histogram, it can be seen that the bin of the data at a height of 4 meters is indeed twice as large as the next one. Besides this peak at a height of 4 meters, there is also a peak at 3.6 meters. So, it might be better to lower the range of 0 to 5 meters to determine the ground points to a range of 0 to 3 meters. This will result in a height threshold of 1 meter for Voorstraat 50.

The raster containing the distances and an image of the building are shown in Figure 5.4. Looking at this figure, it can be seen that the building has a canopy at a height of 3.6 to 4 meters (displayed in purple in Figure 5.4). So, the canopy of this building is causing the peaks in the histogram.

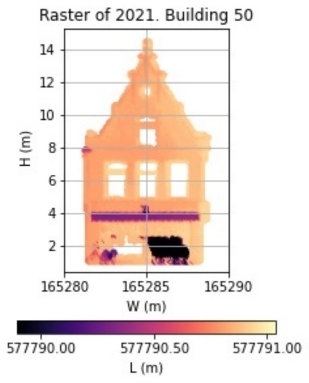
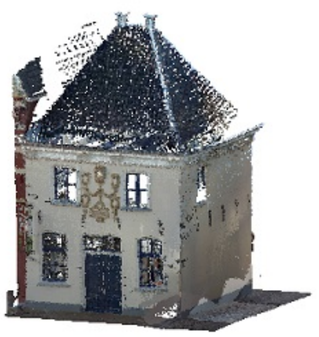


**Figure 5.4.:** *From left to right, raster containing the length component of the data points of Voorstraat 50, and an image of the building.*

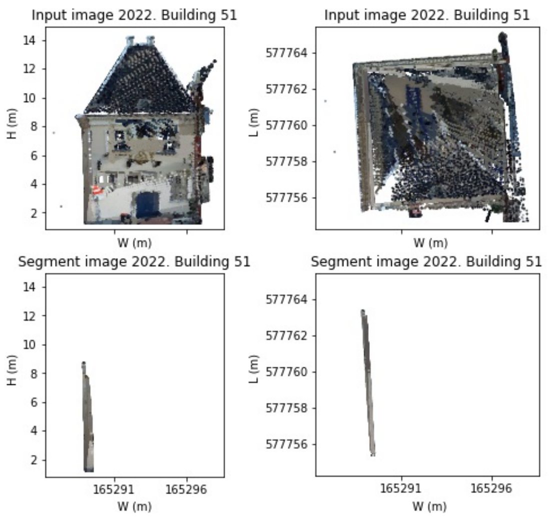
## 5.1.3. Segmenting the facade

Besides removing the ground points, a part of the data is removed during the segmentation of the data. The data is segmented using the RANSAC algorithm. This algorithm gives a plane model and the corresponding inliers as output. Sometimes, the *segment plane* function fits a plane through the side of the building instead of the facade. This can happen at buildings for which the sides are scanned. Examples of such buildings are Voorstraat 51A, Voorstraat 53, Waagstraat 5, and Waagstraat 9. Between Voorstraat 51A and 53, there is some space with a road to the Waagstraat. Between Waagstraat 5 and 9 is some space with a road to the Hortusstraat.

In Figure 5.5, an example is given where the *segment plane* function fits a plane through the side of the building instead of the front of the building. It makes sense because the side of the building is more smooth than the facade of the building which is facing the Voorstraat. In this research, the solution is to cut off the side of the building in CloudCompare. If the wall is not there, the *segment plane* function can not take this wall as the ideal plane. Another possible solution is to use a function that estimates a plane through both the front and the side of the building. Having two possible planes, the one of interest could be selected for the next steps.



**(a)** *Point cloud*

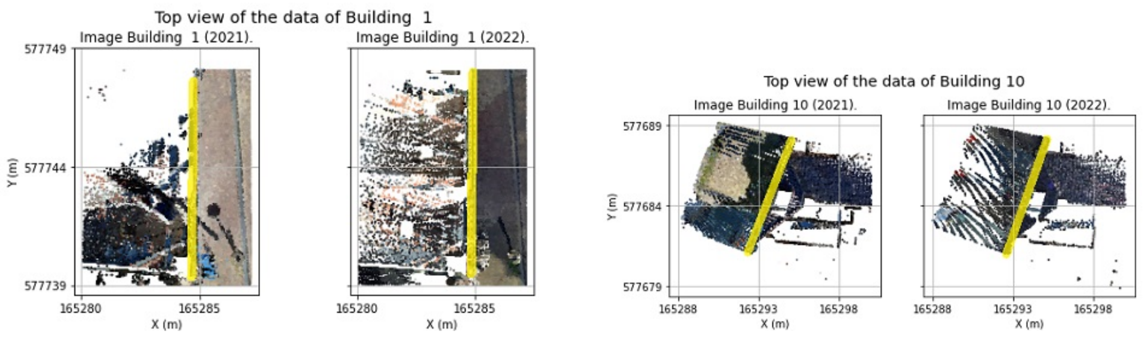


**(b)** *Results after segmentation*

**Figure 5.5.:** *Result of the segment plane function of epoch 2 (plane at the side of a building). Left, is the point cloud of Voorstraat 51A. Right, is the result of segmentation (top not segmented, bottom segmented data).*

### 5.1.4. Transformation of the coordinate system

After segmentation, the data is transformed into a building-based coordinate system. The data is transformed based on the plane model obtained during segmentation. The parameters of the plane model are used to rotate the data around the $z$-axis (also known as the height axis). If the angle of rotation is above 90 degrees, 90 is subtracted from the rotation angle. The resulting rotation angles are displayed in Table 5.2.



**(a)** *Waagstraat 1* **(b)** *Waagstraat 10*

**Figure 5.6.:** *Example of the facade of two buildings with respect to the original coordinate system. The facades of the buildings are marked with yellow.*

In Figure 5.2, the original data of the Waagstraat is shown. Some buildings have their facade parallel to the $y$-axis. However, some buildings have an angle with respect to the $y$-axis. For example the building at the most left part of the image, which is Waagstraat 1 shown in Figure 5.6a. This building seems to be parallel to the $y$-axis, so the transformation angle is assumed to be low. Another example is the building most right at the top, which is Waagstraat 10 shown in Figure 5.6b. This building seems to have an angle with respect to the original coordinate system, so the transformation angle of this building is assumed to be higher than the transformation angle of Waagstraat 1. Looking at the data in Table 5.2, it can be seen Waagstraat 1 has a transformation angle of 0.3 degrees and Waagstraat 10 has a transformation angle of 20.3 degrees, which is indeed much higher.

**Table 5.2.:** *The angle needed to transform the data from the original system $(x, y, z)$ to a building-based system $(W, L, H)$.*

| Building | Transform angle [degrees] | Building | Transform angle [degrees] |
|---|---|---|---|
| Voorstraat 38 | 2.4 | Waagstraat 1 | 0.3 |
| Voorstraat 40 | 3.8 | Waagstraat 1AB | 2.5 |
| Voorstraat 44 | 6.0 | Waagstraat 3 | 2.1 |
| Voorstraat 46 | 5.5 | Waagstraat 4 | 8.8 |
| Voorstraat 47 | 14.8 | Waagstraat 5 | 2.5 |
| Voorstraat 48 | 3.8 | Waagstraat 6 | 9.3 |
| Voorstraat 49 | 13.8 | Waagstraat 8 | 16.5 |
| Voorstraat 50 | 0.6 | Waagstraat 9 | 8.1 |
| Voorstraat 51A | 3.7 | Waagstraat 10 | 20.3 |
| Voorstraat 51 | 6.9 | Waagstraat 11 | 13.1 |
| Voorstraat 52 | 3.6 | Waagstraat 13 | 22.2 |
| Voorstraat 53 | 1.2 | Waagstraat 15 | 24.0 |
| Voorstraat 54 | 5.7 | Waagstraat 19 | 23.8 |
| Voorstraat 55 | 3.2 | Waagstraat 25 | 24.0 |
| Voorstraat 58 | 5.3 | Waagstraat 27 | 23.0 |
| Voorstraat 57 | 4.4 | Waagstraat 29 | 29.2 |
| Voorstraat 59 | 0.5 | Waagstraat 31 | 28.8 |
| Voorstraat 60 | 4.3 | | |

## 5.2. Torsion and tilt angles

The plane method is used to determine the torsion and tilt angles of an estimated plane through the dataset. This estimated plane is a simplified model of the facade of a building.

The plane model is obtained by fitting a plane with the PCA algorithm. First, the difference in torsion and tilt angle between epoch 1 and epoch 2 are displayed in Figure 5.8. Second, the tilt angles (with respect to the ideal plane) per epoch are shown in Figure 5.9. An overview of the angles is displayed in Table 5.3.

There are three different axes around which the building can rotate, as shown in Figure 5.7. The torsion angle is a rotation around the height axis of the building. So, when standing in front of a building a side of the facade can move towards or away from you. The tilt angle is a rotation around the width axis of the building (roll in Figure 5.7). So, when standing in front of a building, the top of the building can either lean forward or backward. Tilt around the length axis of the building (pitch in Figure 5.7) is also possible in some cases. However, for the houses in Franeker, it is assumed that there is no tilt around the length axis (so the pitch angle is assumed to be zero).

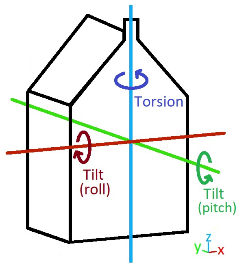
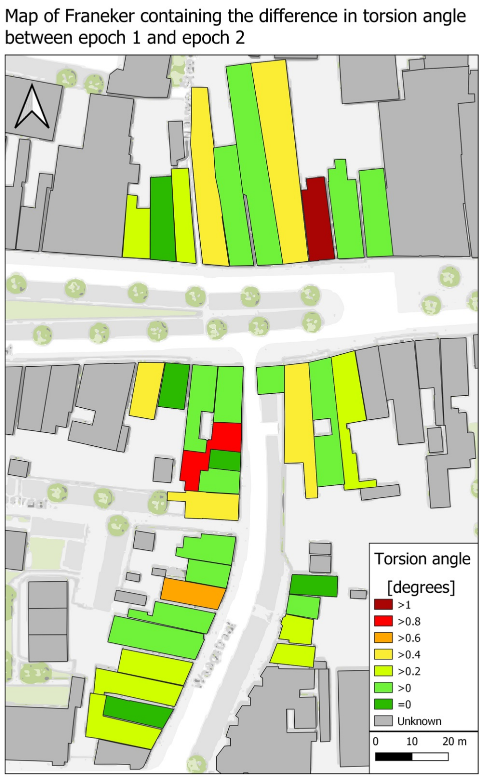


**Figure 5.7.:** *The three rotation degrees: torsion, tilt in roll direction, and tilt in pitch direction.*

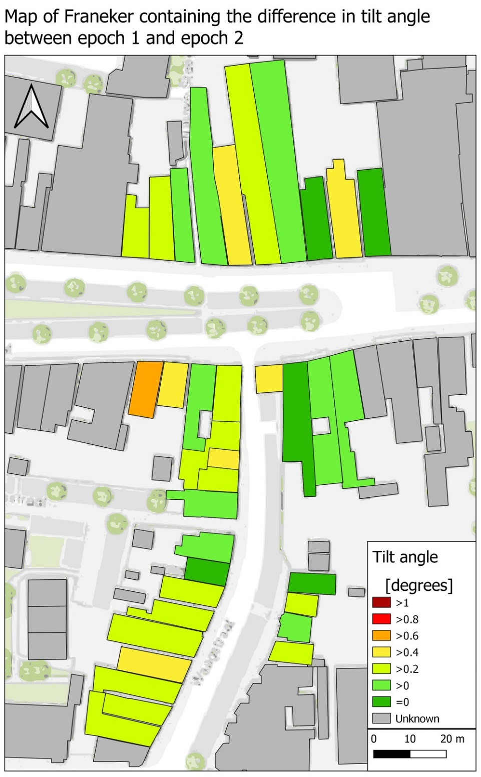
Looking at the colors in Figure 5.8 and the values in Table 5.3, four buildings are assumed to be interesting. These buildings have a difference in either torsion or tilt angle of at least 0.6 degrees. These four buildings are Voorstraat 44, Voorstraat 59, Waagstraat 1 and Waagstraat 13. Looking at the colors in Figure 5.9, there are two buildings with a tilt angle above 1 degree. These two buildings are Waagstraat 9 and Waagstraat 25. So, six buildings are examined to check what causes these larger angles. The exact angles are displayed in Table 5.3. The buildings are examined further in Subsection 5.3.2 by creating a raster of these buildings to find possible causes of the larger angles in comparison to the other buildings. The rasters could also be used to check if other deformation patterns are visible, like absolute non-vertical tilt.

**Table 5.3.:** *Computed torsion and tilt angles of buildings at the Voorstraat and Waagstraat. Angle differ is the difference in torsion or tilt angle between two epochs. The tilt angles are the tilt angles with respect to an ideal plane for a single epoch. An angle difference of at least 0.6 degrees is marked in yellow and a tilt angle of at least 1 degree is marked in orange.*

| | Angle differ [degrees] | | Tilt angles [degrees] | | | Angle differ [degrees] | | Tilt angles [degrees] | |
|---|---|---|---|---|---|---|---|---|---|
| Building | $\theta_{torsion}$ | $\theta_{tilt}$ | $\theta_{epoch1}$ | $\theta_{epoch2}$ | Building | $\theta_{torsion}$ | $\theta_{tilt}$ | $\theta_{epoch1}$ | $\theta_{epoch2}$ |
| Voorstraat 38 | 0.1 | 0.0 | 0.1 | 0.1 | Waagstraat 1 | 0.8 | 0.3 | 0.2 | 0.5 |
| Voorstraat 40 | 0.1 | 0.4 | 0.5 | 0.1 | Waagstraat 1AB | 0.0 | 0.5 | 0.0 | 0.5 |
| Voorstraat 44 | 1.4 | 0.0 | 0.0 | 0.1 | Waagstraat 3 | 0.1 | 0.3 | 0.2 | 0.5 |
| Voorstraat 46 | 0.1 | 0.1 | 0.5 | 0.4 | Waagstraat 4 | 0.0 | 0.0 | 0.2 | 0.2 |
| Voorstraat 47 | 0.2 | 0.1 | 0.2 | 0.1 | Waagstraat 5 | 0.4 | 0.1 | 0.2 | 0.1 |
| Voorstraat 48 | 0.1 | 0.3 | 0.6 | 0.3 | Waagstraat 6 | 0.1 | 0.3 | 0.8 | 0.6 |
| Voorstraat 49 | 0.1 | 0.1 | 0.4 | 0.3 | Waagstraat 8 | 0.2 | 0.1 | 0.5 | 0.6 |
| Voorstraat 50 | 0.1 | 0.5 | 0.1 | 0.6 | Waagstraat 9 | 0.1 | 0.1 | 1.4 | 1.5 |
| Voorstraat 51A | 0.1 | 0.5 | 0.5 | 0.0 | Waagstraat 10 | 0.2 | 0.2 | 0.2 | 0.0 |
| Voorstraat 51 | 0.5 | 0.0 | 0.1 | 0.0 | Waagstraat 11 | 0.1 | 0.0 | 0.4 | 0.4 |
| Voorstraat 52 | 0.4 | 0.1 | 0.0 | 0.1 | Waagstraat 13 | 0.7 | 0.3 | 0.8 | 0.4 |
| Voorstraat 53 | 0.1 | 0.3 | 0.0 | 0.3 | Waagstraat 15 | 0.1 | 0.3 | 0.0 | 0.4 |
| Voorstraat 54 | 0.3 | 0.1 | 0.1 | 0.1 | Waagstraat 19 | 0.1 | 0.3 | 0.0 | 0.3 |
| Voorstraat 55 | 0.1 | 0.1 | 0.1 | 0.2 | Waagstraat 25 | 0.2 | 0.5 | 0.5 | 1.1 |
| Voorstraat 58 | 0.0 | 0.3 | 0.0 | 0.3 | Waagstraat 27 | 0.2 | 0.2 | 0.1 | 0.3 |
| Voorstraat 57 | 0.0 | 0.4 | 0.0 | 0.4 | Waagstraat 29 | 0.0 | 0.2 | 0.3 | 0.2 |
| Voorstraat 59 | 0.5 | 0.6 | 0.2 | 0.7 | Waagstraat 31 | 0.2 | 0.3 | 0.6 | 0.3 |
| Voorstraat 60 | 0.2 | 0.3 | 0.5 | 0.1 | | | | | |

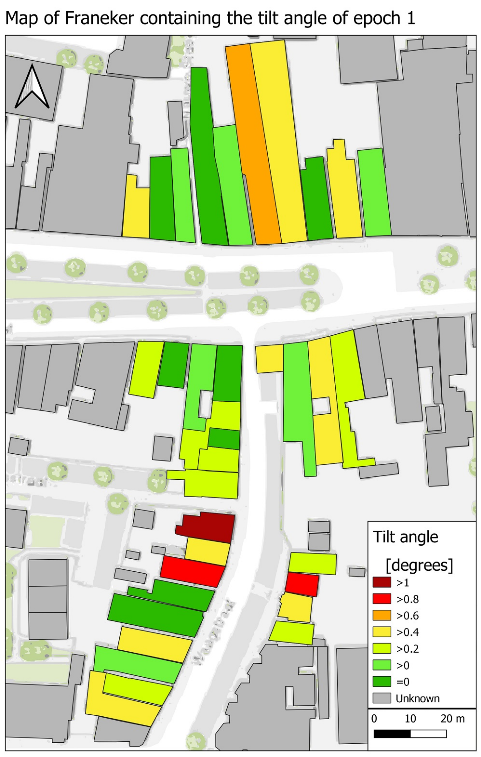

Map of Franeker containing the difference in torsion angle between epoch 1 and epoch 2
Torsion angle
[degrees]
>1
>0.8
>0.6
>0.4
>0.2
>0
=0
Unknown
0
10
20 m

**(a)** *Difference in torsion angle*

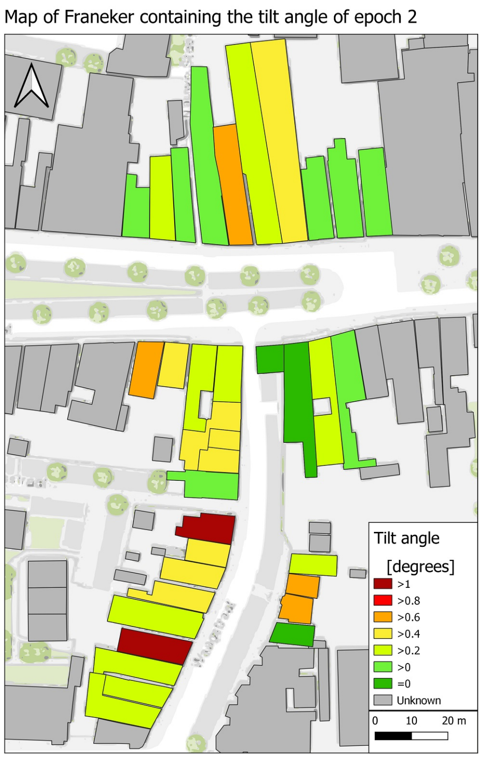

Map of Franeker containing the difference in tilt angle between epoch 1 and epoch 2
Tilt angle
[degrees]
>1
>0.8
>0.6
>0.4
>0.2
>0
=0
Unknown
0
10
20 m

**(b)** *Difference in tilt angle*

**Figure 5.8.:** *From left to right, a map containing the difference in torsion and tilt angle between epoch 1 and epoch 2.*



**(a)** *Tilt angle epoch 1*



**(b)** *Tilt angle epoch 2*

**Figure 5.9.:** *From left to right, map containing the tilt angle of epoch 1 and epoch 2.*

## 5.3. Raster containing the distances of the buildings

For the raster method, a raster with distances of the building is constructed. For the construction of the rasters, the segmented data is used. There is chosen for the segmented data because objects in front of the building are already removed. When these objects are not removed, they will be visible in the data. For example, a car parked in front of the building. Interest does not lie in these objects but in the facade. Therefore it makes sense to use the segmented data.

The values of the raster cell are the distances. The distance can be the difference in length component between the two epochs, or the distance can be the length component corresponding to a single epoch (either epoch 1 or 2). So, there are two types of raster and each one has its own application.

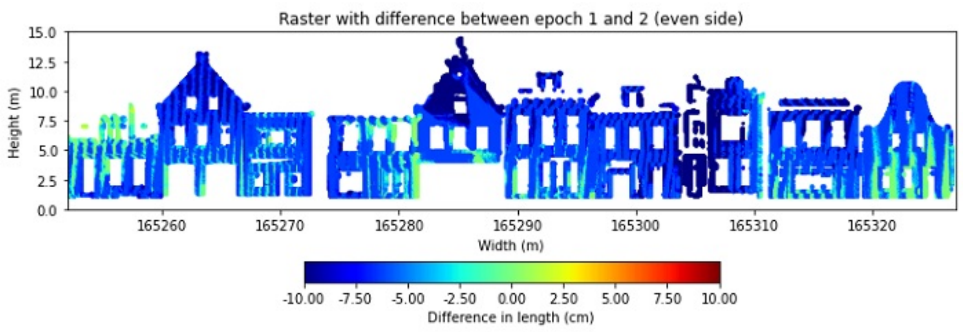
The rasters containing the difference between two epochs are used to determine the horizontal deformation of a building and the rasters of a single epoch are used to analyze local deformation patterns. Examples of things that can cause a difference in distance are things like cracks in the wall, a window, or a wall anchor.

The horizontal deformation of both the Voorstraat and Waagstraat are examined in Subsection 5.3.1. In Subsection 5.3.2, the buildings with the larger angles are visualized by creating rasters for a single epoch and by creating a raster showing the difference between the epochs.

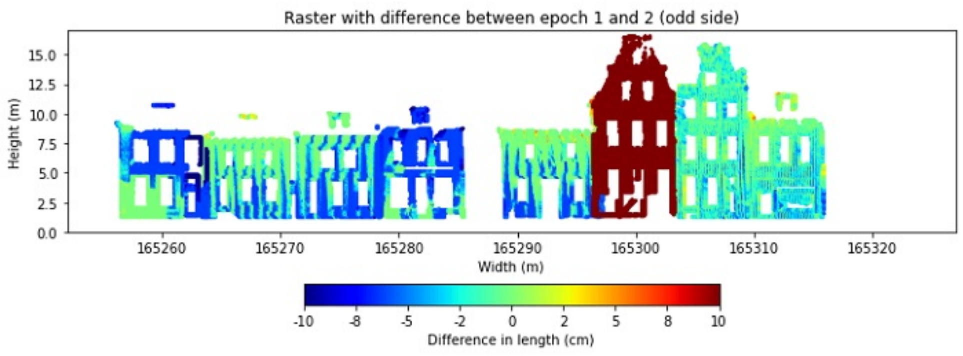
### 5.3.1. Raster of the whole street

A raster of the whole street is created to check if there is a difference in the horizontal distance. To create a raster for a whole street, the data is first sorted per house number. While sorting the data, the odd house numbers are on one side of the street, and the even numbers are on the other side.

The data per street is used to construct the rasters for epoch 1 and epoch 2. Using the rasters of the two epochs, the raster containing the difference between the epochs is constructed. The raster of the Voorstraat is shown in Figure 5.10 and the raster of the Waagstraat is shown in Figure 5.11.



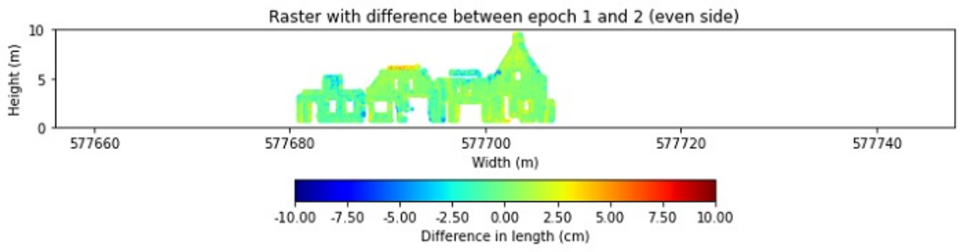
**(a)** *Even house numbers*



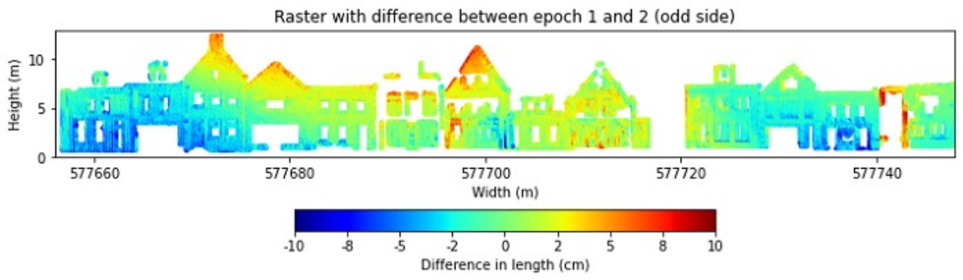
**(b)** *Odd house numbers*

**Figure 5.10.:** *Raster of the Voorstraat. The raster shows the difference in distance between the two epochs. The top raster contains the even house numbers and the bottom raster contains the odd house numbers.*

If the raster points are green, the difference in distance between the epochs is zero. However, when looking at the rasters of the Voorstraat, a lot of buildings are blue and there is even a red building. When looking at the even side of the Waagstraat, all the buildings are green. So, it is expected there is no deformation between the two epochs. On the odd side of the Waagstraat, there is a difference in color. Some parts are displaced forward (red) and some parts are displaced backward (blue).



**(a)** *Even house numbers*



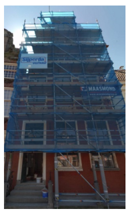
**(b)** *Odd house numbers*

**Figure 5.11.:** *Raster of the Waagstraat. The raster shows the difference in distance between the two epochs. The top raster contains the even house numbers and the bottom raster contains the odd house numbers.*

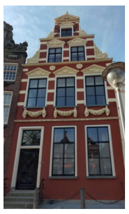
In the next sections, possible explanations are given for certain colors in the rasters of the Voorstraat. First, a possible explanation is given why one of the buildings on the odd side of the Voorstraat is red. Second, a possible explanation is given why the buildings at the Voorstraat are blue. Then, four of the buildings on the odd side of the Waagstraat are discussed in Subsection 5.3.2.

### Explanation of the red building at the Voorstraat

At the Voorstraat, almost all the buildings are either blue or green, but this specific building is red. In order to find out why this building looks different, there is looked at the colored point cloud of this side of the street for epoch 1 and epoch 2, which is shown in Figure 5.13. It can be seen that something blue is placed in front of the building in epoch 1 and this is removed in epoch 2.



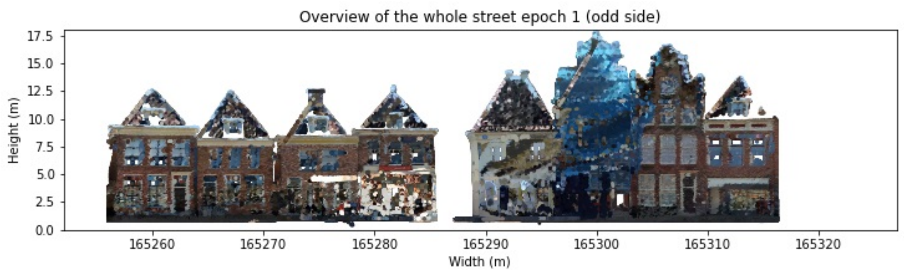
**(a)** *Epoch 1.*



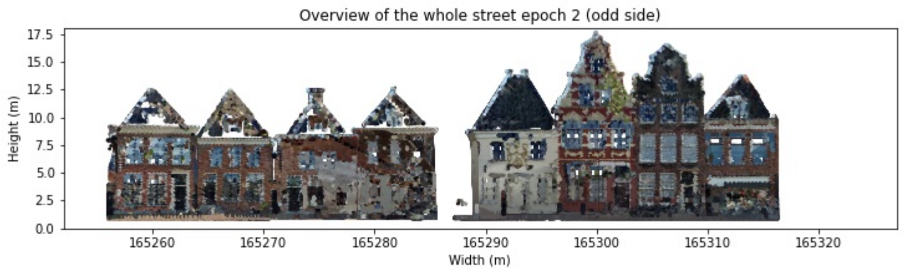
**(b)** *Epoch 2.*

**Figure 5.12.:** *Image of the two epochs of data for Voorstraat 51.*

To determine what this blue thing is, a Cyclorama image is used. The Cyclorama images of Voorstraat 51 are shown in Figure 5.12. From the image, it can be seen that this blue thing is a scaffolding with blue canvases which is placed in front of the building. It is assumed, some construction work is done on the facade of the building in epoch 1. In epoch 2, the facade is visible. So, in epoch 2 the construction work is finished.



**(a)** *Epoch 1*



**(b)** *Epoch 2*

**Figure 5.13.:** *Overview of the buildings at the odd side of the Voorstraat. The top image contains the buildings at epoch 1 and the bottom image contains the buildings at epoch 2.*

### Explanation why buildings in the Voorstraat are blue

In Chapter 4, the methodology of this research is discussed. In this section, the building at Voorstraat 58 was used to show some example images and calculations. From Figure 5.10, it can be seen that almost all the buildings on the even side of the Voorstraat are blue.

For the building at Voorstraat 58, two possible explanations were given for the difference in the length coordinate. The first option was that the facade of this building had deformed between the two epochs and the other explanation was a fault during processing resulting in a misalignment of the data. It is not very likely that the whole even side street has deformed because at some buildings there are green spots. It is more likely it has something to do with the processing or interpretation of the data.

To be sure the values inside the raster make sense, the raster values will be checked using Street Smart. For different locations in this street, the length coordinate of the facade will be determined and this location will be compared for the two epochs. Besides the two epochs shown in this research, there will also be looked at the two epochs before. So, there will be looked at data of 2022 (epoch 2), 2021 (epoch 1), 2020, and 2019. The position of the data points of these four years of data are displayed in Table 5.4 and the position of these points is visualized in Figure 5.14.

**Table 5.4.:** *The Y coordinate of three points at the even side of the Voorstraat.*

| | Left | Middle | Right |
|---|---|---|---|
| X coordinate [m] | 165257 | 165287 | 165324 |
| Z coordinate [m] | 2.5 | 7.5 | 8 |
| | Y coordinate [m] | Y coordinate [m] | Y coordinate[m] |
| 2022 (epoch 2) | 577792.95 | 577790.93 | 577793.48 |
| 2021 (epoch 1) | 577792.91 | 577790.85 | 577793.43 |
| 2020 | 577792.93 | 577790.93 | 577793.48 |
| 2019 | 577792.94 | 577790.93 | 577793.44 |
| Average difference between the years | 2 cm | 5 cm | 5 cm |

All three points seem to have a difference in distance of approximately 6 centimeters in the raster. It can be seen that the difference is 4 centimeters for the left point, 8 centimeters for the middle point, and 6 centimeters for the right point.

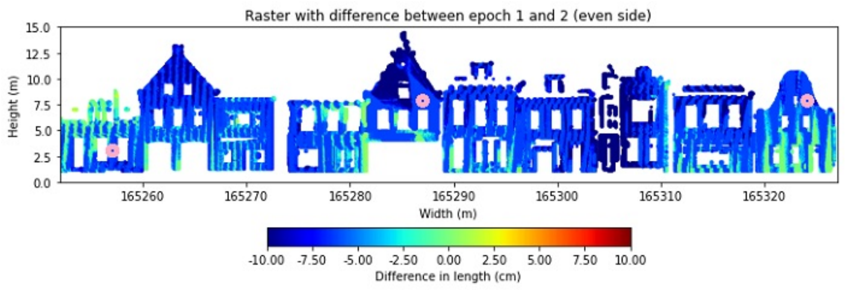


**Figure 5.14.:** *The location of the three points in Table 5.4 are visualized with pink dots.*

So, for some points, the value in the raster does not correspond to the value in the table. This difference could occur due to the way the values inside the raster are computed. It is reasonable that this can cause an offset of 2 centimeters because all three points are located on a brick facade. A value in a raster cell is the average value of the 10 closest points. Some of these 10 points might be located on the brick, but others might be located on the mortar between the bricks. Mostly, mortar is more concave than brick which could cause a difference of a centimeter.

However, this does not explain the difference in distance between epoch 1 and epoch 2. Looking at the data of the other years in Table 5.4, it can be seen that the measured $y$-coordinate of the middle pink point is only different for 2021. It is not likely that the building does not deform between 2019 and 2020, then deforms between 2020 and 2021, and finally returns to the same position as in 2019 and 2020. Therefore, a possible explanation of the difference in the position of the $y$-coordinate could be the accuracy and precision of the data.

In Table 5.5, the precision of the images in the Voorstraat is displayed for different years. The precision of the $x$-coordinate is different for the data of epoch 1. However, this is only a difference of 1 centimeter, not 6 centimeters.

**Table 5.5.:** *The recording date and image precision of the Voorstraat obtained from Street Smart.*

| Recording date | X precision [m] | Y precision [m] | Height precision [m] | Yaw precision [deg] |
|---|---|---|---|---|
| May 2022 (epoch 2) | 0 | 0.01 | 0.01 | 0.01 |
| June 2021 (epoch 1) | 0.01 | 0.01 | 0.01 | 0.01 |
| May 2020 | 0 | 0.01 | 0.01 | 0.01 |
| May 2019 | 0 | 0.01 | 0.01 | 0.01 |

Besides the length component of different years of data (see Table 5.4), the cloud-to-cloud distance between the buildings can be determined. The cloud-to-cloud distance between epoch 1 and epoch 2 of Voorstraat 46 is shown in Figure 5.15. Since all buildings on this side of the Voorstraat have a blue color, it is assumed the cloud-to-cloud distance of the buildings will be similar. Looking at the cloud-to-cloud distance, the difference in the location of the facade is also around 6 centimeters. So, there is indeed a difference in distance between the two epochs and this difference is not caused by how the raster is constructed, because computing the cloud-to-cloud distances gives the same result.

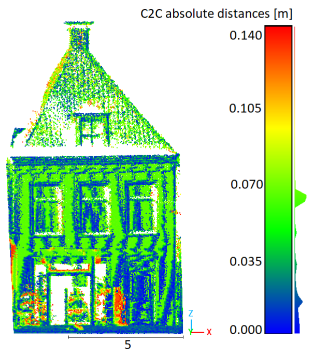


**Figure 5.15.:** *Cloud-to-cloud distance of Voorstraat 46.*

As mentioned in Subsection 3.1.1, the accuracy of the Cyclomedia data is 10 centimeters and the precision is 2 centimeters. In the case of comparing the location of a data point between two epochs, the accuracy will be relevant because the error depends on how the measured data is processed which is a systematic error. The differences between the points are all below the accuracy of 10 centimeters, so it is assumed the points are blue due to the accuracy of the data. If the accuracy of the points would be better, it is assumed more points will be green.

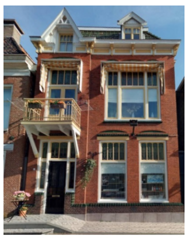
## 5.3.2. Raster of specific buildings

Besides a raster of the whole street, it is also possible to examine specific buildings that seem interesting. Buildings are considered to be interesting if deformation occurs. One possible type of deformation is either torsion or tilt of the facade. In Subsection 5.2, six buildings were found with a large angle. These buildings are Voorstraat 44, Voorstraat 59, Waagstraat 1, Waagstraat 9, Waagstraat 13, and Waagstraat 25. These buildings will be discussed in the following Sections.

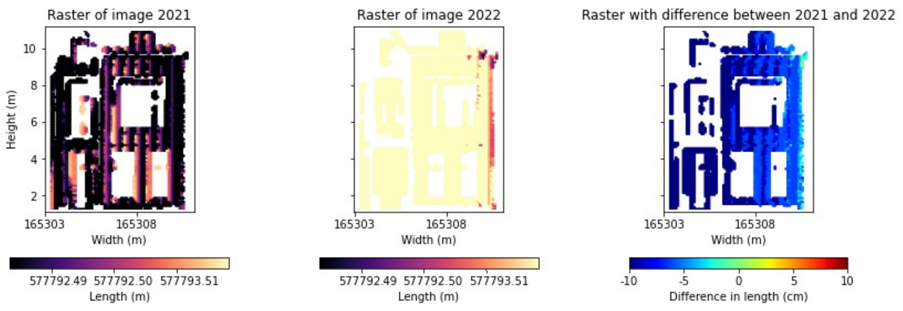
### Voorstraat 44

Voorstraat 44 is a building on the even side of the Voorstraat which is a special building due to the balcony above the front door. An image of this building is shown in Figure 5.16a. For this building, the measured torsion angle was above 0.6 degrees with a value of 1.4 degrees. The tilt angle of this building was very small (around 0 degrees).

The raster of this building is constructed and displayed in Figure 5.16b. Looking at the values in the raster, it can be seen that the left part of the building is more negative than the right part of the building. The difference could be due to the balcony on the left side of the building. From the raster, it is also visible on the right part the facade of the building is segmented, but it is more difficult to segment the data on the left part.



**(a)** *Image*



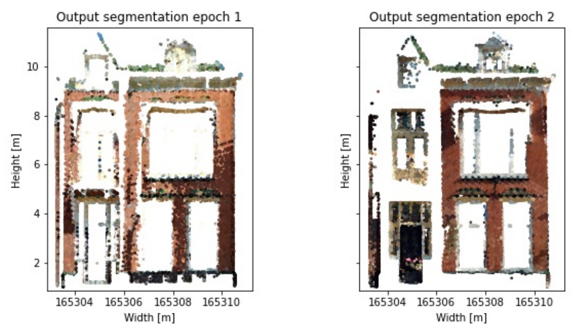
**(b)** *Raster*

**Figure 5.16.:** *Image and raster of Voorstraat 44.*

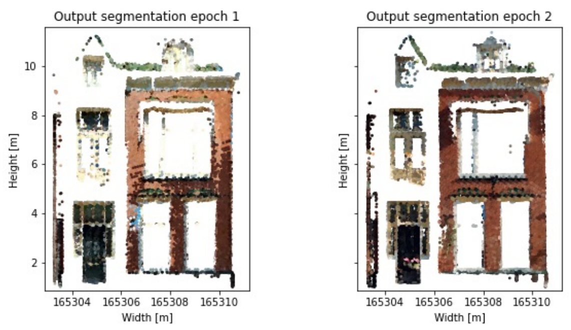
To check if the torsion angle occurs due to the way the data is segmented, the *segment plane* function is performed again. During the first segmentation, the torsion angle was 1.2 degrees. After rerunning the segmentation function, the torsion angle is only 0.3 degrees. So, the difference between the two times of segmentation is approximately 1 degree. In Figure 5.17, the two different outputs of the *segment plane* function are shown. In Figure 5.17a, it can be seen that in the left image, the door and the window of the balcony are removed and they are still there in the right image. So, the different epochs contain different parts of the data, leading to a larger torsion angle. In Figure 5.17b, it can be seen that the images of both epochs have similar points and this leads to a smaller torsion angle.

### Voorstraat 59

Voorstraat 59 is a building on the odd side of the Voorstraat. An image of this building is shown in Figure 5.18a. Looking at this image, there is a color difference visible in the bricks. Besides the color difference in the bricks, the building does not look special. For this building, the difference in torsion angle is 0.5 degrees and the difference in tilt angle is 0.6 degrees.



**(a)** *First time segmentation*

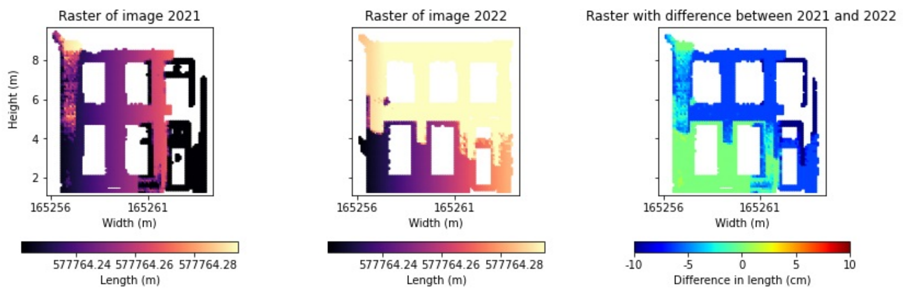


**(b)** *Second time segmentation*

**Figure 5.17.:** *Two results of the segmentation of Voorstraat 44. The left segmentation results in a difference in torsion angle of 1.4 degrees and the right segmentation results in a difference in torsion angle of 0.2 degrees.*

The raster of this building is displayed in Figure 5.18b. In reality, the lowest width coordinate is positioned at the left side of the building, so the raster is flipped horizontally. Looking at the raster, it seems like it is more difficult to segment the top part of the building (this part is yellow in epoch 2). Just like the building at Voorstraat 44, there is also trouble segmenting the part around the door and the window above the door.

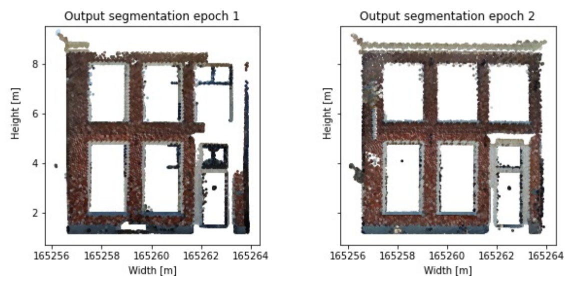
**(a)** *Image*



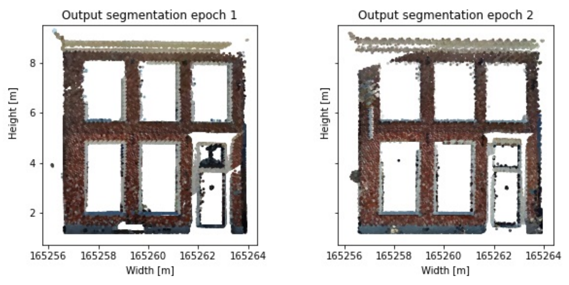
**(b)** *Raster*

**Figure 5.18.:** *Image and raster of Voorstraat 59.*

Just like Voorstraat 44, the data of Voorstraat 59 is also segmented for a second time to check whether the tilt angle occurs due to deformation or due to the way the data is segmented. Comparing the first with the second segmentation, it can be seen that something similar occurs for Voorstraat 44. In the case of a larger tilt angle (0.6 degrees), there are less points on the facade above the door for epoch 1, as shown in the left image in Figure 5.19a. For the second segmentation, the tilt angle is smaller (0.3 degrees). From the segmentation output, it can be seen that the data contains points in the same location. In both images, a gap is visible in front of the building at epoch 1. This gap is a bench with a pillow and a vase with flowers.



**(a)** *First time segmentation*



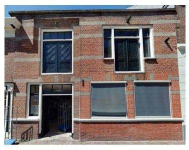
**(b)** *Second time segmentation*

**Figure 5.19.:** *Two results of the segmentation of Voorstraat 59. The left segmentation results in a difference in tilt angle of 0.6 degrees and the right segmentation results in a difference in torsion angle of 0.1 degrees.*

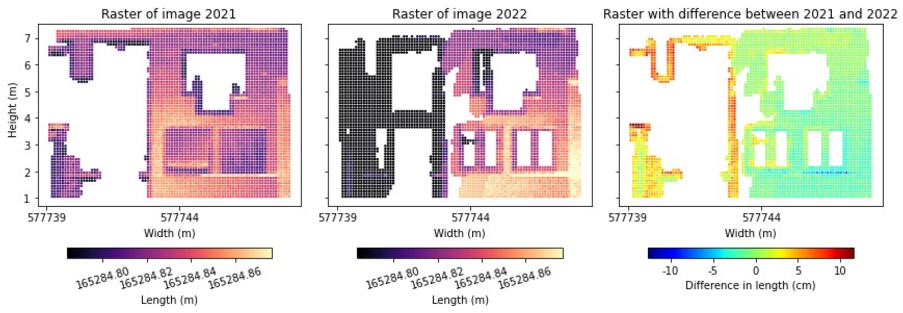
### Waagstraat 1

Waagstraat 1 is a building on the odd side of the Waagstraat. An image of this building is shown in Figure 5.20a. The bricks on the left side are a little bit backward compared to the bricks on the right side of the building. When measuring using Street Smart, a difference in distance of approximately 8 centimeters is measured. This difference in the location of the bricks might explain why the difference in torsion angle is 0.8 degrees. The difference in tilt angle is 0.3 degrees. The raster of the building is displayed in 5.20b. Looking at this raster, two special things are visible.

The first thing is the left part of the building which is a little more backward than the right part of the building. However, the backward part of the building is only visible for epoch 2. Besides this, it is also visible the curtains are closed in epoch 1 and the curtains are open in epoch 2.



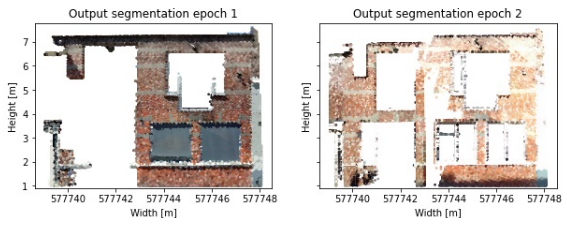
**(a)** *Image*



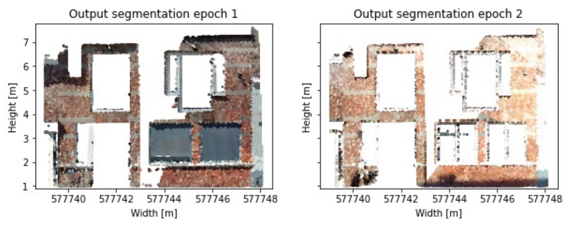
**(b)** *Raster*

**Figure 5.20.:** *Image and raster of Waagstraat 1.*

Similar to the buildings at the Voorstraat, the data of Waagstraat 1 is segmented again. This second segmentation gives a similar result just like the Voorstraat. During the second segmentation, similar parts of the building are returned as output for both epochs. This reduces the torsion angle from 0.8 degrees to 0.1 degrees.



**(a)** *First time segmentation*



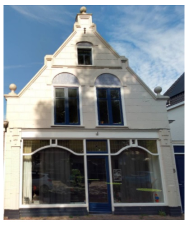
**(b)** *Second time segmentation*

**Figure 5.21.:** *Two results of the segmentation of Waagstraat 1. The left segmentation results in a difference in torsion angle of 0.8 degrees and the right segmentation results in a difference in torsion angle of 0.1 degrees.*

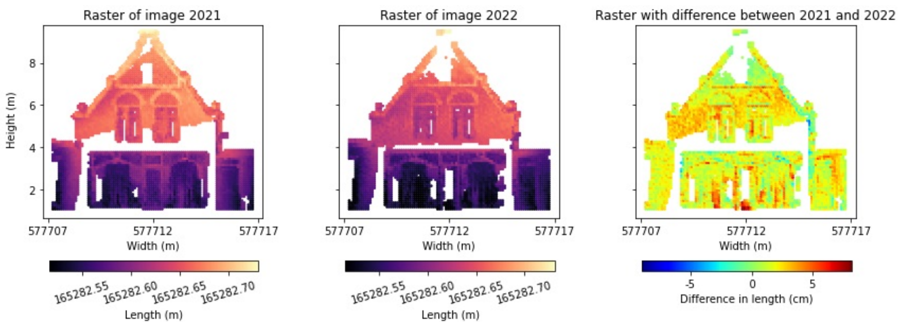
### Waagstraat 9

Waagstraat 9 is a building on the odd side of the Waagstraat. An image of this building is shown in Figure 5.22a. This building is a bit different than the other buildings. Instead of bricks, the building has larger tiles. For the building at Waagstraat 9, the difference in torsion and tilt angle are both 0.1 degrees. The tilt angle of epoch 1 is 1.4 degrees and the tilt angle of epoch 2 is 1.5 degrees. Looking at the raster in Figure 5.22b, the facade does not seem to be tilted. The data is segmented for a second time, but the tilt angle is still above 1 degree for both epochs. The *segment plane* function does not perform well for this specific building.

To check if the large angle is due to how the data is segmented, the data is segmented manually in CloudCompare. Looking at the raster in Figure 5.22b, it can be seen that the window at the bottom of the building is not removed during segmentation. This part is removed manually and the torsion and tilt angles are computed again.

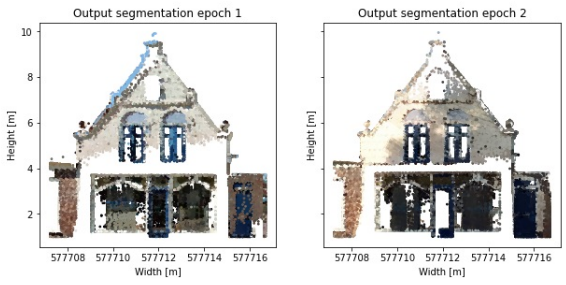


**(a)** *Image*

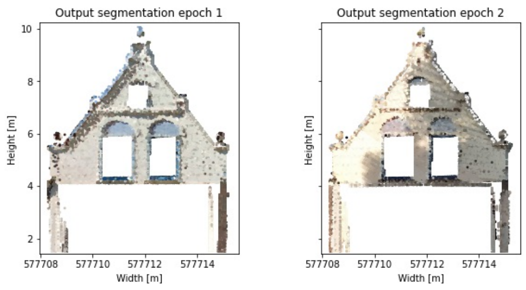


**(b)** *Raster*

**Figure 5.22.:** *Image and raster of Waagstraat 9.*



**(a)** *Automatic segmented data*



**(b)** *Partly manual segmented data*

**Figure 5.23.:** *Two results of the segmentation of Waagstraat 9. The left segmentation results in a difference in torsion and tilt angle of 0.8 degrees. The tilt angle of epoch 1 is 1.4 degrees and the tilt angle of epoch 2 is 1.5 degrees. The right segmentation results in a difference in torsion angle of 0 degrees, a difference in tilt angle of 0.1 degrees. The tilt angle of epoch 1 is 0.5 degrees and the tilt angle of epoch 2 is 0.6 degrees.*

Figure 5.23a shows the result of segmentation using the Python code. It can be seen that the code does not work very well for this specific building. A large part of the facade is deleted for the segmentation of epoch 1. The result of first manually segmenting the data in CloudCompare and then segmenting in Python is shown in Figure 5.23b. When the data is first manually segmented, the facade is not deleted from the data resulting in a difference in torsion angle of 0 degrees and a difference in tilt angle of 0.1 degree. The new tilt angle for epoch 1 is 0.5 degrees and the tilt angle for epoch 2 is 0.6 degrees. So, the tilt angle per epoch has decreased with a degree.

The raster of the partly manual segmented data is shown in Figure 5.24. In the raster, some red spots are visible. When looking at the images of the two epochs, there does not seem to be a difference between the two years. Maybe there is a difference due to the fact the value of a raster cell is the average of the 10 nearest neighbors. The red spots could also have something to do with the material of the facade.

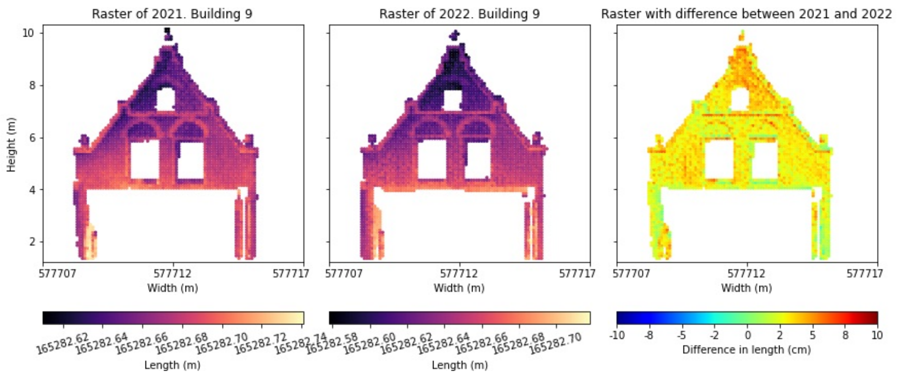
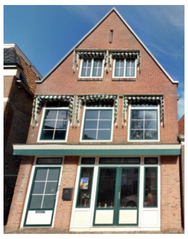
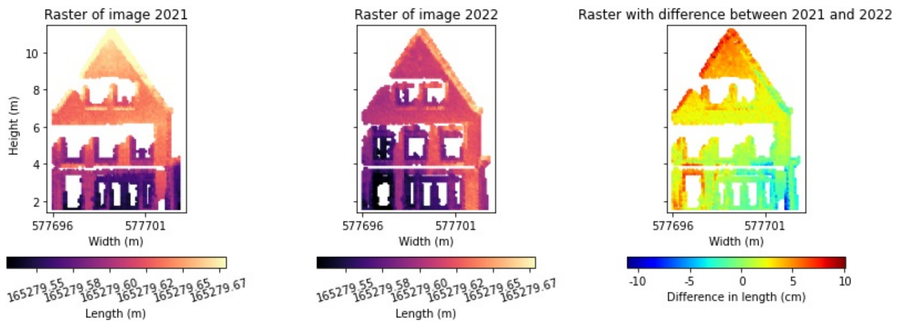


**Figure 5.24.:** *Raster of the partly manually segmented data for Waagstraat 9.*

### Waagstraat 13

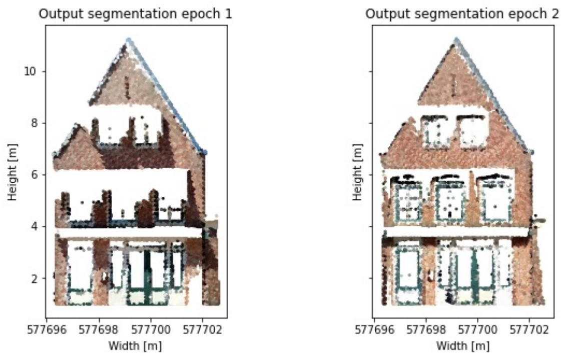
Waagstraat 13 is a building on the odd side of the Waagstraat. An image of this building is shown in 5.25a. For the building at Waagstraat 13, the difference in torsion angle is 0.7 degrees and the difference in tilt angle is 0.3 degrees. Looking at the raster in Figure 5.25b, a gap is visible in epoch 1. This gap is caused by the sunshades which are unfolded in epoch 1 and in epoch 2 they are folded in. The sunshades could be the reason for the difference in torsion angle. After a second segmentation, the difference in torsion angle is 0.2 degrees and the difference in tilt angle is 0.1 degrees. So, the difference in angle was most likely caused by the way the data was segmented and not due to the sunshades.


**(a)** *Image*

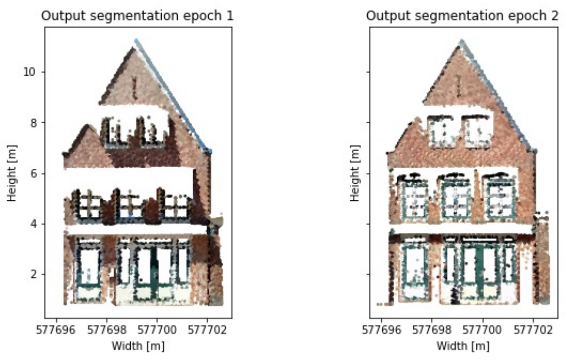


**(b)** *Raster*

**Figure 5.25.:** *Image and raster of Waagstraat 13.*



**(a)** *First time segmentation*



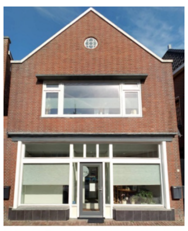
**(b)** *Second time segmentation*

**Figure 5.26.:** *Two results of the segmentation of the Waagstraat 13. The left segmentation results in a difference in torsion angle of 0.7 degrees and the right segmentation results in a difference in torsion angle of 0.2 degrees.*

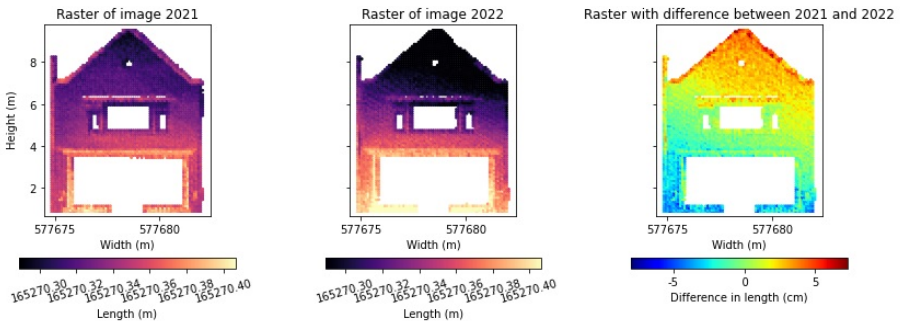
### Waagstraat 25

Waagstraat 25 is a building on the odd side of the Waagstraat. An image of this building is shown in 5.27a. For the building at Waagstraat 25, the difference in torsion angle is 0.2 degrees and the difference in tilt angle is 0.5 degrees. The tilt angle of epoch 1 is 0.5 degrees and the tilt angle of epoch 2 is 1.1 degrees. So, the tilt angle of epoch 2 is quite high and therefore this facade is examined. Looking at Figure 5.27b, it can be seen that the colors are darker at the top of the building in comparison with the bottom of the building. So, it makes sense there is a large tilt angle.

To check if the tilt angle is accurate, the data is segmented again (see Figure 5.28). This results in a tilt angle of 0.6 degrees for epoch 1, and a tilt angle of 1.2 degrees for epoch 2. These tilt angles are similar to the ones obtained during the first segmentation and the resulting segmented data seems to be points on the facade. Tilt is also visible when looking at the colors of the raster. For epoch 1 and epoch 2, the top of the building has a darker color in comparison to the rest of the facade. So, it is expected the building is really tilted. Standing in front of the building, the top part is leaning backward in epoch 1. In epoch 2, the top part of the building is leaning even more backward in comparison to epoch 1.

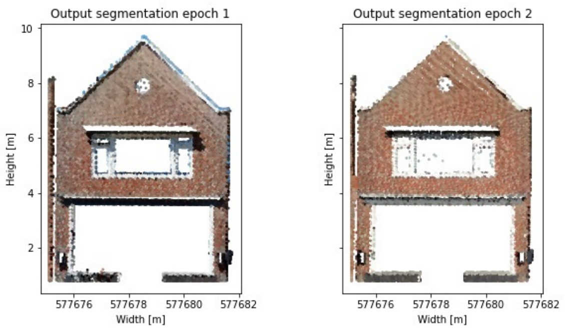


(a) *Image*

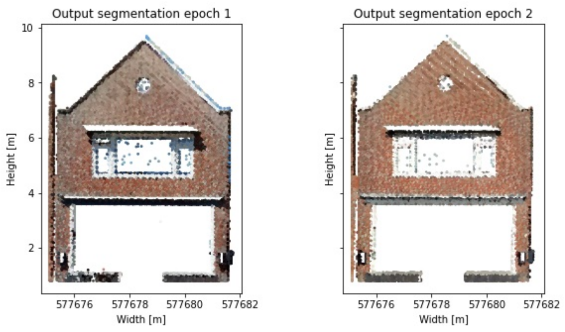


(b) *Raster*

**Figure 5.27.:** *Image and raster of Waagstraat 25.*



(a) *First time segmentation*



(b) *Second time segmentation*

**Figure 5.28.:** *Two results of the segmentation of Waagstraat 25. The left segmentation results in a tilt angle of 0.5 degrees at epoch 1 and a tilt angle of 1.1 degrees at epoch 2. The right segmentation results in a tilt angle of 0.6 degrees at epoch 1 and a tilt angle of 1.2 degrees at epoch 2.*

## 5.4. Summary of the results from Franeker

In Section 5.2 the torsion and tilt angles for Franeker are displayed and in Section 5.3 some rasters are visualized for different buildings in Franeker. The buildings that are visualized are buildings with either a high torsion or tilt angle (above 0.6 degrees) or buildings where the difference in tilt angle between the two epochs of data is high (above 1 degree).

The computed torsion and tilt angles are used to find interesting buildings for the raster method, but they can also be used to determine the relative deformation of a single building with respect to an ideal plane. The rasters can be used to examine the building further and determine if the torsion and tilt angles are caused by local deformation of the facade.

In summary, for most buildings, the obtained deformation is caused by how the data is segmented. For segmentation, RANSAC is used. Looking at the results, segmentation with RANSAC has a downside: it is not possible to control which points are removed from the facade because RANSAC randomly selects some initial points. So, it can be concluded it would be nice to have a more controllable segmentation method like machine learning. For some buildings, the windows and doors were removed using CloudCompare. Using machine learning, these parts could be removed automatically. Besides machine learning, RPCA could be an option instead of PCA because RPCA might also work without segmenting the data first. From the data, it could also be concluded that rasters with data points at the same locations resulted in a smaller difference in the computed angles. So, for future research, it would be nice to only look at the intersection of the two rasters to check if this indeed results in a smaller difference in angle between two epochs of data.

Besides the problems due to the segmentation method, deformation was also visible for some buildings. For example at Waagstraat 25, where the top of the building is tilted backward when looking at the raster and the computed tilt angles.

# 6. Results study area 2: Dancing houses

In this chapter, the results are shown for the dancing houses in Amsterdam. The dancing houses are located at Amstel 100 to 112, but Amstel 82 and 84 are also examined. These two buildings are added to compare buildings that are assumed to be under an angle with buildings that are assumed to have a facade with an angle close to zero.

For this study area, only one epoch of data is used. This epoch is captured on the 17th of January, 2023. There is chosen for one epoch of data because interest lies in the current tilt angle of the dancing houses and according to InSAR data the surface is not deforming (see Subsection 3.3.2).

First, the data is preprocessed in Section 6.1. In Section 6.2, a plane is fitted through the data that is used to determine the tilt angles of the buildings. After displaying the tilt angles, the buildings are examined further by creating a raster of the buildings containing the distance in the length direction in Section 6.3.

## 6.1. Preprocessing of the data

There are four different preprocessing steps. The buildings are fetched from the point cloud in Subsection 6.1.1. The ground points are removed from the data in Subsection 6.1.2. After removing the ground points, the data is segmented in Subsection 6.1.3. Finally, the data is transformed from the original coordinate system to the building-based coordinate system in Subsection 6.1.4.

### 6.1.1. Fetch a building from the point cloud

First, the buildings are cut out from the dataset. This results in a separate las-file for each building. These las-files are imported in Python in order to process the data. The cut-out point clouds of the Amstel are shown in Figure 6.1.

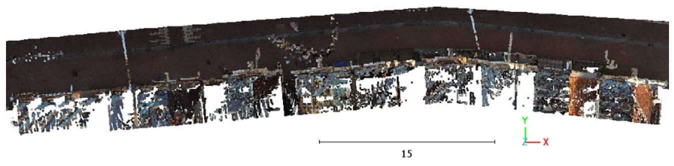


**Figure 6.1.:** *Point cloud data of the Amstel in Amsterdam.*

### 6.1.2. Removing the ground points from the data

The point clouds are imported in Python. Then the ground points are removed by using a histogram to determine the threshold below which points are considered to be ground points. Each building can have a different threshold value for the removal of ground points. If the ground points are removed, they can not influence the results of the segmentation method.

The threshold values for each building are shown in Table 6.1. Almost all the threshold values are 0.6 meters. The building at Amstel 112 has a threshold value of 0.8 meters. Using Street Smart, the height of the street can also be determined. Measuring the $z$-coordinate of the street using Street Smart, the level of the sidewalk is 0.6 meters next to the building and around 0.45 meters along the curbs. So, a threshold of 0.6 meters for ground removal seems reasonable.

For Amstel 112 the threshold value from the histogram is 0.8 meters. However, the building at Amstel 112 has a height of approximately 14 meters. A distance of 0.2 meters is only 1 to 2 percent of the height and therefore it is fine to keep the ground threshold at 0.8 meters.

**Table 6.1.:** *Result of the computed threshold for removing the ground points of the Amstel.*

| Building | Threshold [m] |
|---|---|
| Amstel 82 | 0.6 |
| Amstel 84 | 0.6 |
| Amstel 100 | 0.6 |
| Amstel 102 | 0.6 |
| Amstel 104 | 0.6 |
| Amstel 106 | 0.6 |
| Amstel 108 | 0.6 |
| Amstel 110 | 0.6 |
| Amstel 112 | 0.8 |

### 6.1.3. Segmenting the facade

A part of the data is removed during the segmentation of the data. The data is segmented using the RANSAC algorithm. The segmentation algorithm returns a plane model and the corresponding inliers. Again, the *segment plane* function can fit a plane through the side of the building instead of the facade when the side of the building is scanned. For example for Amstel 112, because next to this building the Bakkerstraat is located which is also scanned. Therefore, the side of this building is cut out from the point cloud before segmentation.

### 6.1.4. Transformation of the coordinate system

After segmentation, the data is transformed into the building-based coordinate system. The data can be transformed based on the plane model obtained during segmentation. The parameters of the plane model can be used to rotate the data around the $z$-axis (also known as the height axis). If the angle of rotation is above 90 degrees, 90 is subtracted from the rotation angle. The resulting rotation angles are displayed in Table 6.2.

The torsion angles seem reasonable when comparing the computed angles with Figure 6.1. The angle of 0.1 corresponds to the fourth building from the right, and this building seems to be in the same direction as the $x$-axis. The angles of the three buildings at the right are similar according to the table, and this is also visible in the point cloud data.

**Table 6.2.:** *Angle to transform the data from the original system* $(x, y, z)$ *to a building-based coordinate system* $(W, L, H)$.

| Building | Transform angle [degrees] |
|---|---|
| Amstel 82 | 4.0 |
| Amstel 84 | 5.4 |
| Amstel 100 | 5.2 |
| Amstel 102 | 3.8 |
| Amstel 104 | 4.6 |
| Amstel 106 | 0.1 |
| Amstel 108 | 6.4 |
| Amstel 110 | 6.0 |
| Amstel 112 | 6.3 |

## 6.2. Torsion and tilt angles of the buildings

The plane method is used to determine the tilt angle of the buildings. For the estimation of the plane parameters, the PCA algorithm is used. The estimated plane is a simplified model of the facade. The resulting tilt angles are shown in Figure 6.2. An overview of the angles is displayed in Table 5.3. It can be seen, Amstel 100 has the highest tilt angle. The buildings at Amstel 82 and 84 have the lowest tilt angle, which is expected because they are not a part of the dancing houses.

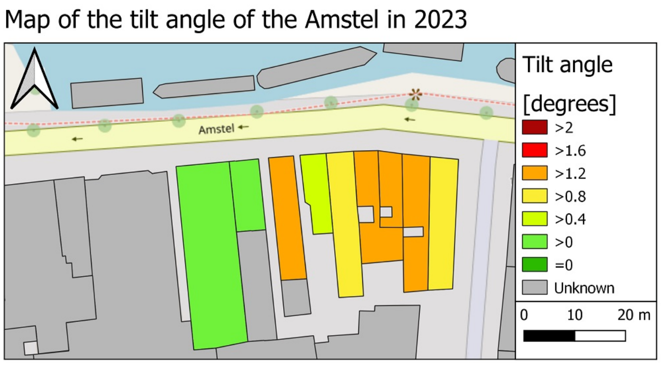


**Figure 6.2.:** *Tilt angles of the Amstel in 2023.*

**Table 6.3.:** *The computed tilt angles of the buildings at the Amstel in Amsterdam.*

| Building | Tilt angle [degrees] |
|---|---|
| Amstel 82 | 0.2 |
| Amstel 84 | 0.3 |
| Amstel 100 | 1.5 |
| Amstel 102 | 0.7 |
| Amstel 104 | 1.1 |
| Amstel 106 | 1.2 |
| Amstel 108 | 1.4 |
| Amstel 110 | 1.2 |
| Amstel 112 | 1.0 |

## 6.3. Raster containing the distances of the buildings

At the Amstel, 9 buildings are selected to examine. For all these buildings, a raster is constructed containing the vertical distances of the building. For the construction of the rasters, the segmented data is used. The distances of the raster are computed with respect to the mean length component. An overview of the result is shown in Figure 6.3. The mean is slightly different for each building, so there is no color scale with numbers.

In the next Sections, the rasters of the dancing houses (Amstel 100-112), Amstel 84, and Amstel 82 are shown. Besides the rasters, an image of the building and the output of segmentation is shown.

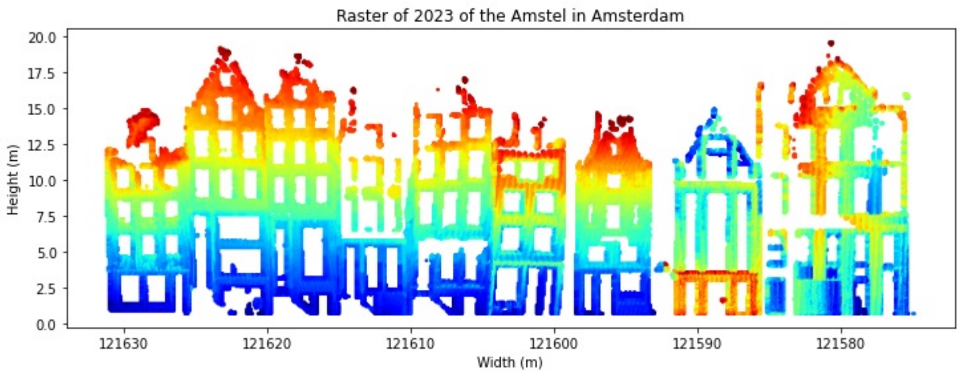
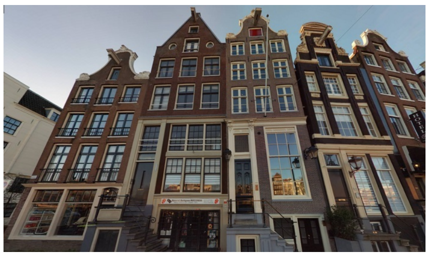


**Figure 6.3.:** *Overview of the raster values with respect to the mean of the length component.*

### Amstel 100-112

The dancing houses are located at Amstel 100 to 112 and an image is shown in Figure 6.4. Looking at Table 6.3, it can be seen almost all the buildings have a tilt angle above 1 degree, except for Amstel 107 which has a tilt angle of 0.7 degrees.


**(a)** *From left to right, Amstel 112, 110, 108 and 106.*

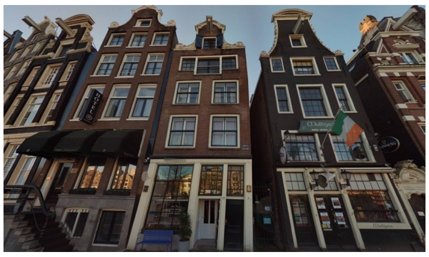
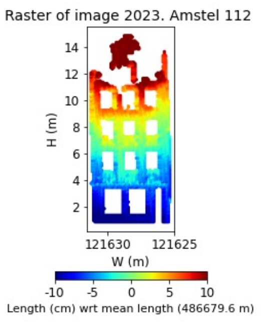
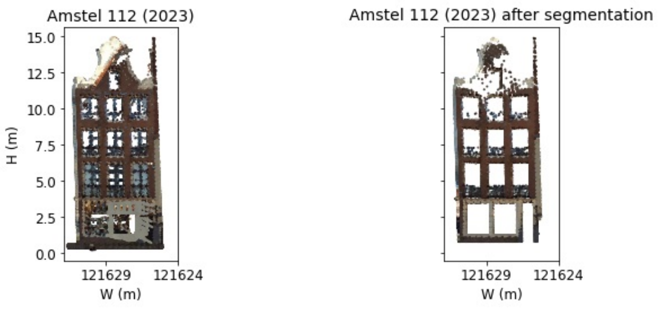

**(b)** *From left to right, Amstel 104, 102 and 100.*

**Figure 6.4.:** *Image of the dancing houses in at the Amstel in Amsterdam.*

The raster and the result of segmenting the data for Amstel 112 are shown in Figure 6.5. From the raster, it can be seen the building is leaning forward because the top part is closer and the bottom part is farther away. From the segmented data, it can be seen the segmentation function works quite well on the whole facade, except for the top part of the building. At the top part, some points on the facade are missing. This could be due to the fact these points are further away and therefore they might have a lower density. Besides tilt of the facade, it is also visible the facade of the building is leaning a bit to the left. This is visible in both segmented and unsegmented data and also in the raster.



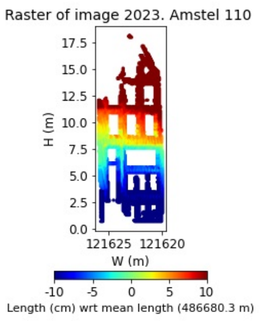
**(a)** *Raster.*



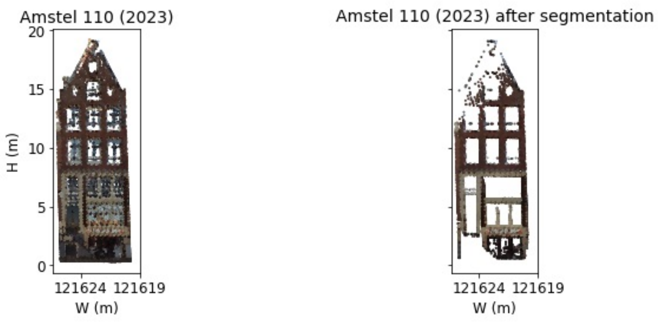
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.5.:** *Raster and segmentation result of Amstel 112.*

The raster and the result of segmenting the data for Amstel 110 are shown in Figure 6.6. From the raster, it can be seen the building is leaning forward because the top part is closer and the bottom part is farther away. The segmentation seems to be quite well, again with the exception of the left part at the top of the building. This building seems also to be leaning a bit to the left side.



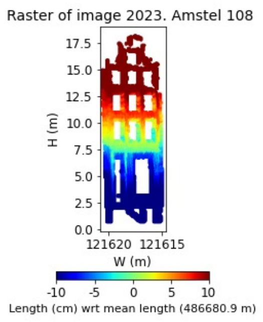
**(a)** *Raster.*



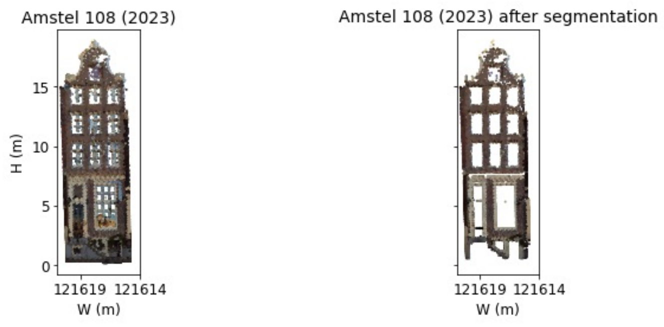
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.6.:** *Raster and segmentation result of Amstel 110.*

The raster and the result of segmenting the data for Amstel 108 are shown in Figure 6.7. This building is also leaning forward and the facade also seems to be leaning a bit to the left side. For this building, the output of the segmentation function seems to be better than Amstel 112 and 110 because fewer points are missing at the top of the building after segmentation.



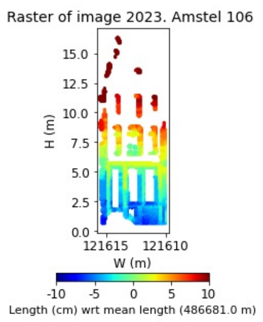
**(a)** *Raster.*



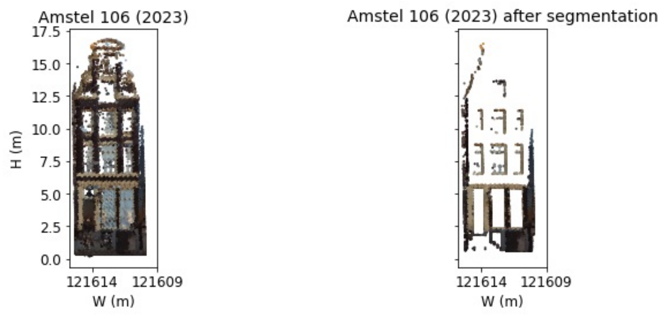
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.7.:** *Raster and segmentation result of Amstel 108.*

The raster and the result of segmenting the data for Amstel 106 are shown in Figure 6.8. For this building, the result after segmentation is assumed to be bad. A lot of points on the facade are removed above a height of 6 meters. In the remaining points, it is visible the building is leaning forward. In the unsegmented data, it can be seen the building is also learning a bit to the left. Even after rerunning the *segment plane* function multiple times, the result stays bad. Comparing the image of Amstel 108 with Amstel 106, the amount of data points on the facade of the unsegmented data seems similar. However, when looking more closely at the building at Amstel 106, this building has more decoration on the facade. For example, a decoration with the text "anno" and one with the number "1659" above the windows at a height of approximately 9 meters. This makes the building less smooth and therefore harder to segment.



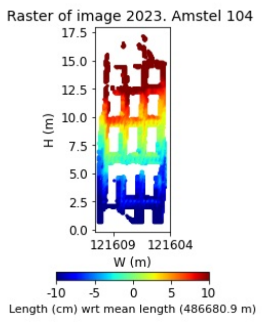
**(a)** *Raster.*



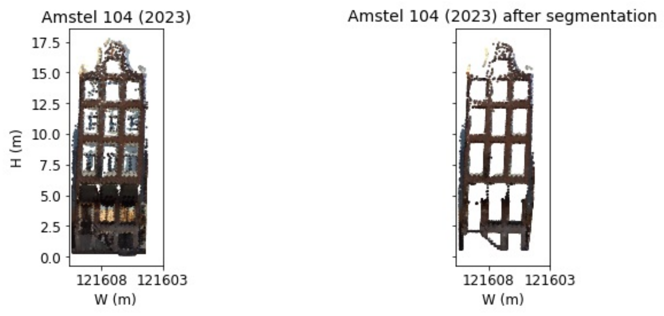
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.8.:** *Raster and segmentation result of Amstel 106.*

The raster and the result of segmenting the data for Amstel 104 are shown in Figure 6.9. This building is also tilted forward. However, the other buildings were leaning to the left side and this building is leaning to the right side. During segmentation, the windows and the sunshades are removed. So, for building 104 the segmentation works well.



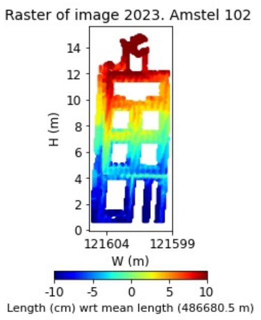
**(a)** *Raster.*



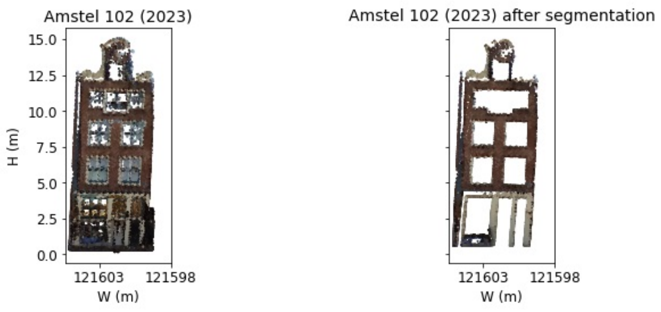
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.9.:** *Raster and segmentation result of Amstel 104.*

The raster and the result of segmenting the data for Amstel 102 are shown in Figure 6.10. This building is also tilted forward. The bottom part of the building is leaning a lot to the right side, but the top part is leaning less to the right side which is a bit strange. Maybe someone tried to decrease the leaning angle and adjusted the building. Another reason can be the construction materials. Maybe the bottom part is from another material, causing another degree of leaning to the right side. The segmentation seems good because the windows are removed and the points on the facade remain.



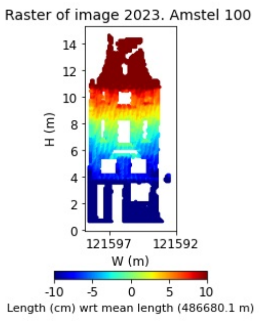
**(a)** *Raster.*



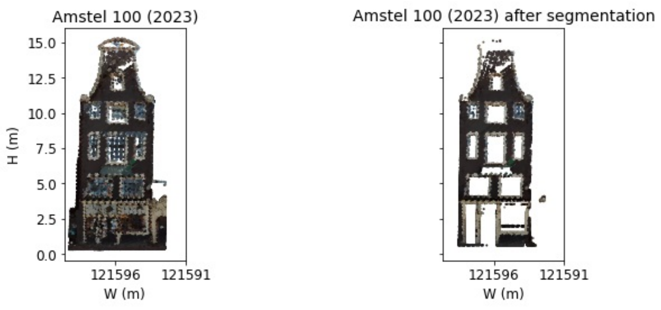
**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.10.:** *Raster and segmentation result of Amstel 102.*

The raster and the result of segmenting the data for Amstel 100 are shown in Figure 6.11. This building is tilted forward and leaning to the right. In this building, the opposite is visible in comparison to Amstel 102. For this building, the bottom part does not seem to be leaning to a side, but the top part of the building is leaning to the right side. Besides this, the output of the segmentation function looks good. On the facade of the building, a sign is placed with the text "Mulligans Irish music bar" and this sign is removed from the segmented data but the parts on the facade remain after segmentation, which is good.



**(a)** *Raster.*



**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.11.:** *Raster and segmentation result of Amstel 100.*

## Amstel 84

The image of Amstel 84 is shown in Figure 6.12. The raster and the result of segmenting the data for Amstel 84 are shown in Figure 6.13. This building has a tilt angle of 0.3 degrees. Looking at the colors in the raster, this seems reasonable. A large part of the building is colored green, meaning the building is standing straight. At the bottom, there are some red points which makes sense because this part is a bit in front of the bricks. At the top of the building, a part is blue. At this location, there is some decoration which might cause a difference in distance.

Looking at the segmentation result of the building, it can be seen only the bricks which are more forward remain. Using Street Smart, it is determined the distance between these bulging parts and the other bricks is 10 centimeters. So, it makes sense the other part of the facade is removed, because a threshold value of 4 centimeters is used for segmentation. A distance of 10 centimeters is above this threshold of 4 centimeters and therefore these points are removed.

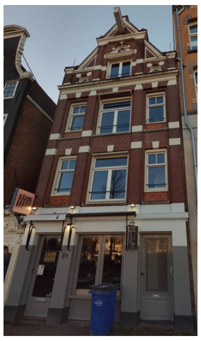
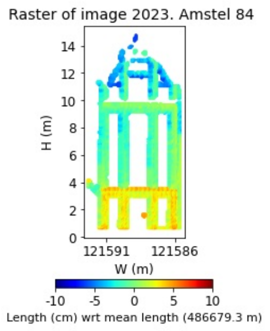
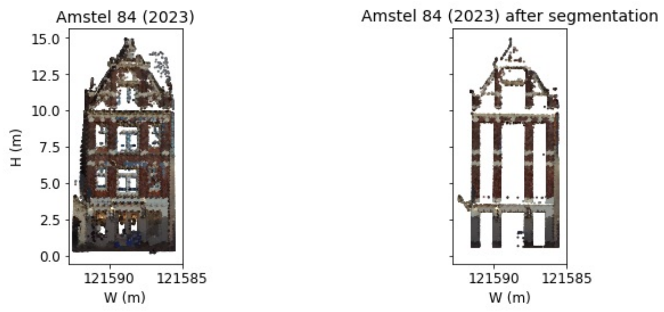


**Figure 6.12.:** *Image of Amstel 84 in Amsterdam.*



**(a)** *Raster.*



**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.13.:** *Raster and segmentation result of Amstel 84.*

### Amstel 82

The image of Amstel 82 is shown in Figure 6.14. The raster and the result of segmenting the data for Amstel 82 are shown in Figure 6.15. This building has a tilt angle of 0.2 degrees. The building is quite large with a width of approximately 10 meters. Looking at the raster, a large part of the building is green. This indicates the building is also standing straight. There are also some blue and red parts. However, looking at the image, there are no irregularities visible on the facade.

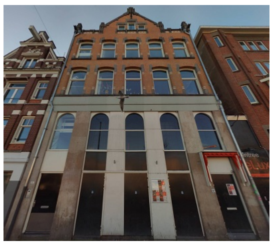
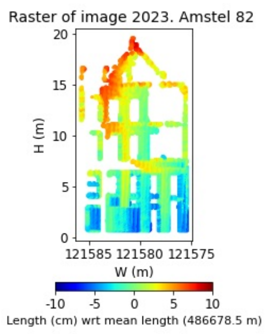


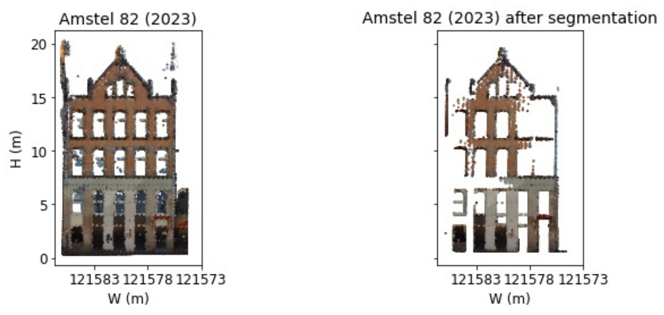
**Figure 6.14.:** *Image of Amstel 82 in Amsterdam.*

After segmenting the data, a large part of the facade is removed as is shown in Figure 6.15b. This could be due to the bottom part of the building. There are some doors and windows at the bottom part of the building that are not part of the facade. If the initial points are points on these doors or windows, this might influence the segmentation. To check if these points influence the segmentation, they are manually removed. This results in the data set and the corresponding segmentation results shown in Figure 6.16.

When the doors and the windows at the bottom part of the dataset are removed before the segmentation function, the result after segmentation is much better. So, it is expected the doors and windows at the bottom of the building have some influence on the segmentation function. The raster contains more green points after manually segmenting the data with CloudCompare. The tilt angle of the first manually segmented data is 0.1 degrees.

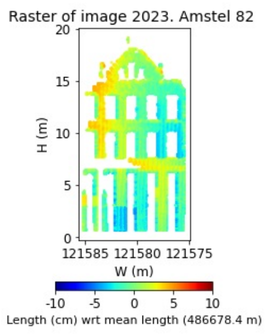


**(a)** *Raster.*

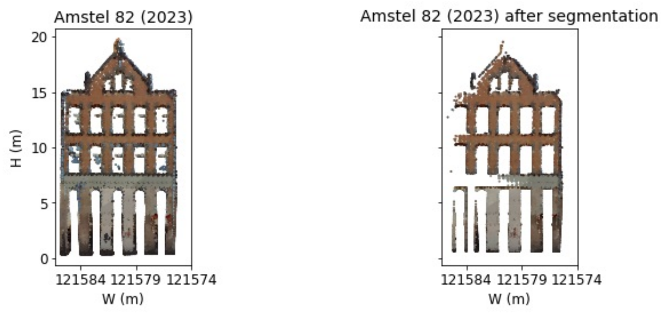


**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.15.:** *Raster and segmentation result of Amstel 82.*



**(a)** *Raster.*



**(b)** *Left, non segmented data. Right segmented data.*

**Figure 6.16.:** *Raster and segmentation result of Amstel 82 after manually segmenting the data first.*

## 6.4. Summary of the results from Amsterdam

According to the results in this chapter, it can again be concluded the segmentation method is not optimal and needs to be improved. Besides the segmentation method, it can be concluded the dancing houses are indeed under an angle. They are not only tilted around the width axis but also around the length axis. So, for future research, it would also be nice to compute how much a building is tilted towards the sides.

# 7. Discussion

First, some effects of assumptions are discussed in Section 7.1. Then the quality is discussed in Section 7.2. A conclusion is made about the usability of the current method in Section 7.3.

## 7.1. Effect of certain assumptions in the chosen method

The way the data is processed influences the results. For example, which part of the point cloud is cut out or how the data is segmented. In this section, the influence of certain assumptions or methods is discussed.

### 7.1.1. Influence of how the building is cut out in CloudCompare

How a building is cut out in CloudCompare affects the result. For example, the building mentioned in Subsection 5.1.3 (Voorstraat 51A in Franeker). For this building, three of the four walls are scanned as shown in Figure 7.1. The interest lies in the facade at the front of the building, which is marked in yellow. However, the yellow wall at the front can accidentally be removed from the data during segmentation. To prevent this, the points on the blue and green walls are removed during preprocessing in CloudCompare.

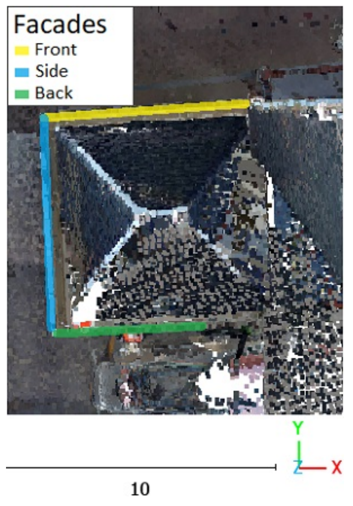


**Figure 7.1.:** *The three walls which are scanned for the building at Voorstraat 51A.*

For the code in this research, the buildings are manually cut out in CloudCompare with the 'Segment' tool. So, for each building, a las-file is created that is imported in Python. Another possibility might be to write a code that automatically extracts a building from the point cloud when the point cloud data is classified. The data could, for example, be classified with machine learning to detect the borders of the building. If each building has an unique class, it is possible to extract buildings from a single las-file in Python instead of importing multiple files. This is useful when you want to analyze many buildings because it is very time-consuming to manually cut out all the buildings. However, this is out of the scope of this research because in this research the amount of buildings is limited.

It might also be possible to cut out the buildings by using a shapefile from ONEGEO. This shapefile contains polygons of each building. The intersection of the polygon and the point cloud could be used to cut out the data points of a single building. This could be tried in future research.

### 7.1.2. Removing ground points from the data

For the removal of the ground points, different methods can be used. In this research, there is chosen to construct a histogram of the height of data points and remove the points below a threshold value. As mentioned before, the resulting threshold value depends on three factors: the bin width, the chosen height difference between the bins, and the cutoff value of the histogram.

First, the bin width is set to 0.2 meters. Second, the chosen height difference of the bin is that the left bin (compared to the bin to its right) needs to contain twice as many data points. Third, the cutoff value of the histogram (the maximum height value at which the ground points are assumed to be) is set to 3 meters. Besides the histogram method used in this research, other methods can be used to determine the ground points.

For example, using segmentation or manually determining the threshold of the ground points. Another option is to use the elevation values of a building. When downloading the three-dimensional building data from ONEGEO, the elevation of the building is included. However, it is difficult to obtain large areas of data from this service as it is not freely available. It is possible to test the data with a free coupon code of $50. The estimated threshold values using the histogram method and the elevation data from ONEGEO are shown in Table 7.1.

For Voorstraat 51, the estimated height from the histogram is 1.2 meters and the elevation from ONEGEO is 0.7 meters. However, looking at the elevation in Street Smart an elevation of 1.2 meters is found. For Waagstraat 29, the estimated height using the histogram is 1.2 meters and the elevation from ONEGEO is 0.7 meters. However, the elevation measured in Street Smart is 0.6 meters. So each method of computing the ground points or using already available data can occur some errors. The most reliable method for ground point removal is measuring the height in front of each building in Street Smart, but this is very time-consuming and not very efficient. It might be good to find another method to estimate the threshold of the ground points that is not time-consuming and does not contain errors.

**Table 7.1.:** *Height threshold from the histogram compared to the elevation from ONEGEO for Franeker.*

| | Threshold [m] | | | Threshold [m] | |
|---|---|---|---|---|---|
| Building | Histogram | ONEGEO | Building | Histogram | ONEGEO |
| Voorstraat 38 | 1.0 | 1.07 | Waagstraat 1 | 1.0 | 0.98 |
| Voorstraat 40 | 1.0 | 1.22 | Waagstraat 1AB | 0.8 | 0.92 |
| Voorstraat 44 | 1.0 | 1.32 | Waagstraat 3 | 1.0 | 0.96 |
| Voorstraat 46 | 1.0 | 1.44 | Waagstraat 4 | 0.6 | 0.63 |
| Voorstraat 47 | 1.2 | 1.43 | Waagstraat 5 | 1.0 | 1.13 |
| Voorstraat 48 | 1.0 | 1.44 | Waagstraat 6 | 0.6 | 0.53 |
| Voorstraat 49 | 1.2 | 1.32 | Waagstraat 8 | 0.6 | 0.54 |
| Voorstraat 50 | 1.0 | 1.44 | Waagstraat 9 | 1.0 | 0.92 |
| Voorstraat 51A | 1.2 | 1.14 | Waagstraat 10 | 0.6 | 0.56 |
| Voorstraat 51 | 1.2 | 0.67 | Waagstraat 11 | 0.8 | 0.93 |
| Voorstraat 52 | 1.0 | 0.98 | Waagstraat 13 | 1.0 | 0.92 |
| Voorstraat 53 | 1.2 | 1.08 | Waagstraat 15 | 0.8 | 0.85 |
| Voorstraat 54 | 1.2 | 1.14 | Waagstraat 19 | 0.8 | 0.81 |
| Voorstraat 55 | 1.2 | 1.35 | Waagstraat 25 | 0.8 | 0.93 |
| Voorstraat 58 | 1.2 | 1.19 | Waagstraat 27 | 0.6 | 0.73 |
| Voorstraat 57 | 1.2 | 1.27 | Waagstraat 29 | 1.2 | 0.72 |
| Voorstraat 59 | 1.2 | 1.31 | Waagstraat 31 | 0.6 | 0.72 |
| Voorstraat 60 | 1.0 | 1.27 | | | |

### 7.1.3. Segmenting the facade

In Chapter 5 and Chapter 6 containing the results, there is already mentioned the used *segment plane* function is not ideal. The function does not always segment the data how you want the data to be segmented. Sometimes, a part of the facade is cut out. It might be better to use another method to segment the data.

For example, a machine learning algorithm that detects objects like windows and doors and removes these objects from the data. Besides this, it can be seen removing windows or doors sometimes helps to get a better result from the *segment plane* function. So, it could be a nice addition to add code to do this automatically.

It might also be possible a segmentation method on something different than RANSAC performs better. However, no proof is given that another method might perform better because this is not tested in this research.

### 7.1.4. Transformation of the coordinate system

The data is transformed using the plane parameters obtained by the segmentation function as this function segments the data based on a plane model. However, it could be there is still a slight angle in the data even after transforming the data with a transformation angle. Further research can be done in another method to determine the transformation angle instead of using the torsion angle of the plane model obtained from the segmentation function.

If the determined angle for the transformation of the data is more accurate, it will be easier to see the deformation patterns in the raster. Without applying the transformation, the distances in the raster will be dependent on the angle of the building with respect to the original coordinate system.

## 7.2. Quality of the used methods

The quality of computed angles is discussed in Subsection 7.2.1 and the quality of the raster with the distances is discussed in Subsection 7.2.2.

### 7.2.1. Quality of the computed angles

The computed angles depend on the quality of the data. If the quality of the point cloud is better, the quality of the computed angles also increases. Besides the quality of the data points, the computed angles are affected by how the data is segmented. Each time the *segment plane* function is run, the points that are removed are different. Using a different set of points, the computed torsion and tilt angles can be different (as proofed in Subsection 4.3.1).

If the segmentation function is more stable, the computed angles are also expected to be more accurate. However, the angle will always depend on the parameters of the plane and the parameters of the plane will always depend on which data points remain after segmentation. So, it would be nice to construct a method that selects similar points for the two epochs of data. This could be possible using the raster method. Only the data points that are present in the rasters of both epochs are selected. So, there is investigated which raster cells are present at both epochs and these points are selected. If a raster cell is only present for one epoch, the data points in this raster cell are removed. It is expected this method results in datasets containing similar points. However, this method is not tested, so it is unsure if the computed angles will be more stable with this method.

For the determination of the quality of the angles, the data of a single building is used (SNS building at Voorstraat 58). Using the data of this building, the range, mean and standard deviation are determined by rerunning the code 500 times. Using the point cloud data of a different building, the range, mean and standard deviation could be different. However, in this research, it is assumed the quality will be approximately the same for each building, which might not be the case in reality.

### 7.2.2. Quality of the constructed raster

Besides the quality of the data points, the quality of the raster depends on the size of the raster cells. If the width and the height of the raster cells are larger, there will be more points inside a single raster cell. So, in the case of larger raster cells, more data is lost because only the 10 closest points to the raster cell are used to determine the value of the raster cell.

If the width and the height of the raster cells are too small, there will be fewer points inside a single raster cell. If there are less than 10 points inside a cell, the value of the raster cell is set to NaN. So, the size of the raster cell must not be too small because then there might not be enough points to determine the value of the raster cell. Besides fewer data points, the computation time is longer for a raster with smaller raster cells because there are more cells for which the value has to be determined.

For this research, there is chosen for a raster cell size of 10 by 10 centimeters. Using this size does not make the code very slow and the size is small enough so details are still visible. For example at Waagstraat 9, where the ornaments on the building are visible in the raster of a single epoch.

## 7.3. Usability of the method

A good thing about the current method is the availability of the data that is used because the LiDAR data is captured yearly. However, the current method is not ready to apply to a larger area. There is still a large variation in the computed angles. One time the angle has a certain value and the next time running the code the angle might have another value. The method that computes the angles might be more useful if the result of segmentation does not variate that much. So, for future research, it is a good idea to find a better way to segment the data.

Besides the computation of the torsion and tilt angles, rasters containing the distances are constructed. These rasters can be used to examine the local deformation patterns of the buildings. Using the rasters, tilt can be visualized. Besides tilt, it is also possible to visualize objects attached to the facade, like name plates, sunshades, or a flag. So, the raster method works quite well for relative deformations.

From the results of the Voorstraat in Franeker, it can be concluded that it is difficult to compare data of different epochs. At the Voorstraat, there was a difference in distance of approximately 10 centimeters between two epochs. This difference occur due to the fact that the data of different epochs is recorded at different locations. Using GPS, the measurement locations can be determined. However, the accuracy of the determination of the position is 10 centimeters and the accuracy could even be worse in areas with a lot of trees or high buildings. So, it is not recommended to compare the data of different epochs directly. It is better to compare tilt angles with respect to an ideal plane because the precision of the data of a single epoch is 2 centimeters. The same holds for the rasters. A raster can be constructed for a single epoch, and the patterns visible in the rasters of two epochs can be compared. It is not recommended to subtract the rasters of two epochs and look at these distances. So it can be concluded that the method is better suited for analyzing relative deformation than absolute deformation.

# 8. Conclusions and recommendations

First, the answers to the research questions are given in Section 8.1. Next, recommendations for future research are given in Section 8.2.

## 8.1. Conclusions

The main question of this research is

*How can street LiDAR (point clouds) and Cyclorama data (images) from the digital Cyclomedia recording system be used to analyze relative deformation of buildings?*

To analyze relative deformation, a suitable area must first be found. An area is considered to be suitable if deformation occurs (the surface is either subsiding or rising unequally) or if the after-effects of deformation are still visible (e.g. buildings with a tilt angle). If a suitable area is found, methods are needed to compare the point cloud data of different epochs. The data of different epochs can be compared by estimating a plane or constructing a raster using point cloud data to analyze deformation patterns. Besides the point cloud data, images can be used to analyze the deformation that is found. Finally, the data of the obtained results can be evaluated.

The following five subquestions give some support to the answer to the main question.

**1. How to find suitable areas where deformation occurs for which a method can be developed to analyze deformation?**

A suitable study area is an area where deformation occurs or an area where the after-effects of deformation are visible. An area where deformation occurs can be found using InSAR data. An example of InSAR data is the Dutch surface and object motion map showing the displacement in millimeters per year. In this map, red colors indicate the subsidence of the surface and objects, and blue colors indicate the rise of the surface and objects. So, these red and blue dots on the map indicate deformation.

Besides the Dutch surface motion map, news articles can be used to find areas. In news articles, people can complain about damage to their houses that could be caused by deformation. For example, the dancing houses in Amsterdam where the surface does not seem to move based on the Dutch surface motion map. However, the surface has subsided in the past, which resulted in deformation of the buildings and this deformation is still visible.

**2. How can relative deformation be estimated using data from multiple epochs?**

A facade can be modeled by a plane. A plane can be estimated with different algorithms, like Ordinary Least Squares (OLS), Total Least Squares (TLS), or Principal Component Analysis (PCA). Section 2.4 explains these plane fitting methods. For this research, there is chosen to use PCA because this method has a lower RMSE than OLS. Thereby, TLS is limited to a certain amount of data and PCA is not. How a plane can be created with PCA is discussed in Subsection 2.4.3. After creating the plane, the plane parameters $(A, B, C, D)$ are known. With these plane parameters, the torsion and the tilt angle of the plane are determined. Using these computed angles, the data of different epochs can be compared. The angle of an epoch can be determined with respect to the ideal plane, or the angle of an epoch can be determined with respect to another epoch. The angles are computed with:

$$\theta = \cos^{-1}\left(\frac{\mathbf{a} * \mathbf{b}}{|\mathbf{a}||\mathbf{b}|}\right)$$

For the torsion angle, the vector of an epoch is *[B,A]* and the vector of the ideal plane is *[1,0]*. For the tilt angle, the vector of an epoch is *[B,C]* and the vector of the ideal plane is *[1,0]*.

Besides the angles obtained using the estimated plane, the data of two epochs can be compared by creating rasters. How these rasters are created is discussed in Subsection 4.3.2. For a dataset with two epochs, three rasters are created: a raster with the length coordinate of epoch 1; a raster with the length coordinate of epoch 2; and a raster showing the difference in the length coordinate between the two epochs. For all these three rasters, the location of the raster cells is the same. This makes it possible to compare the rasters with each other.

Instead of a raster per epoch showing the length coordinate, it is also possible to subtract the mean of the length coordinate from the data. This could make it easier to interpret the data of a single epoch because the raster shows the distance with respect to the mean resulting in smaller values in comparison to using the length coordinate.

**3. What deformation patterns can be determined and with what confidence?**

In Section 2.1, different types of deformation are discussed which are: vertical deformation, tilt, torsion, cracks, bow, and bulge. From the results in Chapter 5 and 6, it can be concluded not all deformation patterns are visible in the data.

The used method is not really suitable to determine vertical deformation, because the building is displayed from the side and vertical deformation is easier to measure from the top view. So, vertical deformation is out of the scope of this research.

The used method is able to determine the tilt and torsion angle using the plane parameters. However, the determined angle depends on the result of the segmentation function. The segmentation function can give different results each time this function is run and therefore the torsion and tilt angles depend on the segmentation function. Overall, there is assumed the angles can be determined with a precision of $\pm$ 0.6 degrees and an accuracy of 0.10 degrees. So, the difference in angle must be in the order of a degree to be able to divide deformation with an angle caused by wrong segmentation.

In the used datasets, there were no buildings with cracks. So, cracks were not detected. Looking at the data, it did not seem like there were bowing facades. Therefore it is unknown if the method is suitable to determine cracks or bow.

Some buildings in the dataset contained some bulging (or protruding) parts. For example the building at Waagstraat 1, where the left part is more backward in comparison to the right part of the building. This is also visible in the data because the data of epoch 2 shows a large black part. To detect bulge on a facade, the data is first transformed into the building-based coordinate system. When the building is not transformed and the transformation angle is above 1 degree, it is very hard to detect local deformation patterns because the distance shows the length coordinate. The length coordinate depends on the transformation angle. If the transformation angle is high, the difference in the maximum and minimum length coordinate is much larger in comparison to a small transformation angle. Due to transformation, the range of the length coordinate decreases. The protruding parts can be removed from the data during segmentation. So, to determine the distance between a protruding part and a non-protruding part the unsegmented data is needed. From the data, it can be seen it is possible to detect a distance of 8 centimeters. So it is proven bulge can be detected when the difference in distance is at least 8 centimeters when using a segmentation threshold of 4 centimeters.

**4. How can the different types of Cyclomedia data be used to analyze deformation?**

In this research, two types of data from Cyclomedia have been used: the point clouds (LiDAR) and the Cycloramas (images). Using the point clouds, the torsion and tilt angles are determined. Besides the computation of these angles, the point clouds are used to construct a raster to determine vertical deformation. The Cycloramas are used to explain why certain results are occurring because images are easier to interpret than colored point clouds. For example, to find the reason why there are gaps in the data after segmentation. Looking at the Cycloramas, it can be seen there is a gap in the data at Voorstraat 59 due to a bench in front of the building, which is harder to see in a point cloud.

**5. How to evaluate the deformation measured using the planes or rasters?**

How the planes and rasters are created is discussed in subquestion 2. In this research question, there is discussed how the planes and rasters can be used to measure deformation and how to evaluate the measured deformation.

The angles from the plane method can be placed on a map to visualize them. Looking at this map it will be easy to see which buildings have a larger angle and which buildings have a smaller angle. A small angle will be marked with green, indicating the building is fine (no further investigation is needed). A large angle (above 0.8 degrees) will be marked with red, indicating something has happened with this building. These red buildings are recommended to examine further to find out if reinforcement might be needed.

The rasters are compared by looking at the colors of the raster cells. For a single epoch, the color represents the average length value of the 10 nearest neighbors of the raster cell. From this average value, the mean of the data can be subtracted to visualize the distance of certain raster cells with respect to the mean length coordinate of this raster. The distance with respect to the mean length can be used to make it easier to interpret the data. The colors of a raster can mean different things depending on the raster type: single epoch of data, or the difference between two epochs. For a single epoch, a green color means the building is standing straight, a red color indicates this part of the building is more forward with respect to the mean, and a blue color indicates this part of the building is more backward with respect to the mean. For the difference between epochs, a green color means the building is not deforming, a red color indicates the building is deforming backward with respect to the previous epoch, and a blue color indicates the building is deforming forward with respect to the previous epoch.

In conclusion, it can be said the two types of data from the Cyclomedia recording system, the point clouds and the images, can be used to analyze deformation.

The street LiDAR data can be used to analyze the deformation of buildings by using point clouds. Using point clouds, the facade of a building can be modeled with a plane. When a plane is estimated through the data, the torsion and tilt angles of this plane can be computed. Using these angles deformation is analyzed when there is a difference in torsion or tilt angle between two epochs of data, or when the tilt angle of a single epoch is above 1 degree. Besides analyzing deformation with a plane, deformation can be analyzed using a raster. The raster contains vertical distances of the building. The raster of a single building can be used to analyze if there are local deformation patterns and the difference in raster between two epochs can be used to analyze relative deformation.

Besides the point clouds, the Cyclorama data is used. The Cyclorama data are images of the building captured at different epochs. These images can give more insight into the reason why a building is deforming. For example, the building at Voorstraat 51 where a scaffolding is placed in front of the building in epoch 1 and the scaffolding is removed in epoch 2. Images are also useful to visualize the structure of a building. For example when there are ornaments on the facade, which is also the case at Voorstraat 51.

## 8.2. Recommendations

Examples of the aspects that can be further investigated are mentioned below.

**Find an alternative way to segment the data**

The results depend a lot on the output of the *segment plane* function. This is not necessarily bad, but the results of the segmentation are unstable and sometimes the output of the segmentation function does not even contain the points on the facade of the building, but it does contain the window frames. If a method is constructed to have more influence on the points which have to be removed, the result might be better. This was for example visible at Waagstraat 9. After manually segmenting a part of the data, some points that were previously removed are not removed anymore.

A possible alternative method to segment the data could be machine learning. Machine learning could be used to remove objects from the dataset, like windows, doors, plates or flags. Instead of removing object, it is also an option to detect points that correspond to the facade by using the properties of the facade. Examples of properties are the color of the materials, or the roughness.

Besides machine learning, it might also be possible to estimate a plane through the data with Robust Principal Component Analysis (RPCA) instead of PCA. In comparison to PCA, RPCA can handle outliers so maybe this algorithm can also deal with non-segmented data.

### Use depth maps instead of point clouds

In this research, point clouds are used to analyze deformation. However, point clouds of Cyclomedia are only available for 2 to 3 years. After this period, the point clouds are removed to save storage space. The point clouds are converted to depth maps, because depth maps have a smaller file size. For future research, it would be nice to compare the depth maps with the point clouds to check if these two datasets can analyze deformation with the same accuracy. Besides the comparison of the datasets, it is easier to compare multiple epochs of data using depth maps because there are depth maps available of multiple years.

### Other applications

Besides detecting the deformation of a facade, this method might also be useful for other applications. For example to find holes or other damage to a road using a raster with distances.

# Bibliography

Alter, O., Botstein, D., & Brown, P. O. (2000). Singular value decomposition for genome-wide expression data processing and modeling. *Proceedings of the National Academy of Sciences*, *97*(18), 10101–10106. https://doi.org/https://doi.org/10.1073/pnas.97.18.10101

Bakuła, K., Ostrowski, W., Szender, M., Plutecki, W., Salach, A., & Górski, K. (2016). Possibilities for Using LIDAR and Photogrammetric Data Obtained with an Unmanned Aerial Vehicle for Levee Monitoring. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, *41*. https://doi.org/10.5194/isprs-archives-XLI-B1-773-2016

Bauer, E., Milhomem, P. M., & Aidar, L. A. G. (2018). Evaluating the damage degree of cracking in facades using infrared thermography. *Journal of Civil Structural Health Monitoring*, *8*(3), 517–528. https://doi.org/https://doi.org/10.1007/s13349-018-0289-0

Cahyadi, I., Mokhamad Nur & Rwabudandi. (2019). Integration of GNSS-IMU for increasing the observation accuracy in Condensed Areas (Infrastructure and Forest Canopies). *E3S Web of Conferences*, *94*, 03015. https://doi.org/https://doi.org/10.1051/e3sconf/20199403015

Constant, N., Cay, G., Ravichandran, V., Diouf, R., Akbar, U., & Mankodiya, K. (2021). Data analytics for wearable IoT-based telemedicine. In *Wearable sensors* (pp. 357–378). Elsevier. https://doi.org/https://doi.org/10.1016/B978-0-12-819246-7.00013-9

Corso, J., Roca, J., & Buill, F. (2017). Geometric Analysis on Stone Facades with Terrestrial Laser Scanner Technology. *Geosciences*, *7*(4), 103. https://doi.org/https://doi.org/10.3390/geosciences7040103

CPM Educational Program. (2014). *CPM 3D Plotter*. Retrieved April 4, 2023, from https://technology.cpm.org/general/3dgraph/

Crosetto, M., Monserrat, O., Cuevas-González, M., Devanthéry, N., & Crippa, B. (2016). Persistent scatterer interferometry: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, *115*, 78–89. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2015.10.011

Day, J., Richard & Clarke. (2003). Plastic and thermal cracking. In *Advanced Concrete Technology: Concrete Properties* (pp. 3–17). Butterworth-Heinemann, Elsevier. https://doi.org/10.1016/B978-075065686-3/50249-4

De Gans, W. (2011). *De bodem onder Amsterdam: een geologische stadswandeling*. TNO Geologische Dienst Nederland. http://resolver.tudelft.nl/uuid:f63bef32-f957-4162-a88d-ba0fc7b29ebb

Derpanis, K. G. (2010). Overview of the RANSAC Algorithm. *Image Rochester NY*, *4*(1), 2–3.

Diaz, C., Cornadó, C., & Albareda, A. (2020). Damage in face-brick facades placed between concrete slabs. *Journal of Building Engineering*, *30*, 101312. https://doi.org/https://doi.org/10.1016/j.jobe.2020.101312

Eberly, D. (2000). Least squares fitting of data. *Chapel Hill, NC: Magic Software*, 1–10.

Eberly, D. (2001). Least Squares Fitting of Data. *Magic Software, Inc., Capel Hill, N.C.*, 1–10.

Esfahany, S. S., Hanssen, R. F., van Thienen-Visser, K., & Muntendam-Bos, A. (2010). On the effect of horizontal deformation on InSAR subsidence estimates. *ESA Special Publication*, *677*.

Feng, C., Zhang, D., Wang, H., & Zhang, X. (2021). Deformation response and safety evaluation of buildings affected by subway-station construction. *Advances in civil engineering*, *2021*, 1–18. https://doi.org/https://doi.org/10.1155/2021/1694563

Ferlisi, S., Nicodemo, G., & Peduto, D. (2019). Numerical analysis of the behaviour of masonry buildings undergoing differential settlements. *Geotechnical Engineering, Foundation of the Future: Conference Proceedings*, 1–8. https://doi.org/https://doi.org/10.32075/17ECSMGE-2019-0450

Geudtner, R., Dirk & Torres. (2012a). Sentinel-1 system overview and performance. *2012 IEEE International Geoscience and Remote Sensing Symposium*, 1719–1721. https://doi.org/10.1109/IGARSS.2012.6351191

Gokdemir, H., Ozbasaran, H., Dogan, M., Unluoglu, E., & Albayrak, U. (2013). Effects of torsional irregularity to structures during earthquakes. *Engineering failure analysis*, *35*, 713–717. https://doi.org/https://doi.org/10.1016/j.engfailanal.2013.06.028

Gordon, S. J., Lichti, D., Stewart, M., & Franke, J. (2005). *Structural deformation measurement using terrestrial laser scanners* (Doctoral dissertation). Curtin University of Technology.

Guéguen, A., P & Astorga. (2021). The torsional response of civil engineering structures during earthquake from an observational point of view. *Sensors*, *21*(2), 342.

Hein, G. W. (2020). Status, perspectives and trends of satellite navigation. *Satellite Navigation*, *1*(1), 22. https://doi.org/https://doi.org/10.1186/s43020-020-00023-x

Hestenes, M. R. (1961). Relative hermitian matrices. *Pacific Journal of Mathematics*, *11*(1), 225–245. https://doi.org/10.2140/pjm.1961.11.225

Hoes, O. (2021). *Grondwateronttrekking Delft-Noord Resultaten van de monitoring 2020* (tech. rep. AW_038_OH_180905). Acacia Water. Delft, The Netherlands.

Hofer, M., Strauß, G., Koulechov, K., & Dietz, A. (2005). Definition of accuracy and precision—evaluating cas-systems. *International congress series*, *1281*, 548–552. https://doi.org/https://doi.org/10.1016/j.ics.2005.03.290

Hoogland, T., van den Akker, J. J. H., & Brus, D. J. (2012). Modeling the subsidence of peat soils in the Dutch coastal area. *Geoderma*, *171-172*, 92–97. https://doi.org/https://doi.org/10.1016/j.geoderma.2011.02.013

Houtenbos, A. P. E. M. (2010). *Bodemdaling NW-Friesland 1976-2009*. https://doi.org/10.13140/RG.2.2.14028.41600

Isyumov, M., N & Poole. (1983). Wind induced torque on square and rectangular building shapes. *Journal of Wind Engineering and Industrial Aerodynamics*, *13*(1-3), 183–196. https://doi.org/https://doi.org/10.1016/0167-6105(83)90140-X

Jamwal, N., Jindal, N., & Singh, K. (2016). A survey on depth map estimation strategies. *International Conference on Signal Processing (ICSP 2016)*, 1–5. https://doi.org/10.1049/cp.2016.1453

Kuipers, G. M. (2021). Van tunnelvisie naar open blik: de schadevergoedingslessen van de Noord/Zuidlijn. *Overheid & Aansprakelijkheid*, *2021*(1), 2–13.

Kurgalin, S., Borzunov, S., Kurgalin, S., & Borzunov, S. (2021). Equation of a Plane in Space. In *Algebra and geometry with python* (pp. 307–320). Springer. https://doi.org/https://doi.org/10.1007/978-3-030-61541-3

Lu, J., Zhang, M., Zhang, X., Pei, W., & Bi, J. (2018). Experimental study on the freezing–thawing deformation of a silty clay. *Cold Regions Science and Technology*, *151*, 19–27. https://doi.org/https://doi.org/10.1016/j.coldregions.2018.01.007

Lyu, H., Sha, N., Qin, S., Yan, M., Xie, Y., & Wang, R. (2019). Manifold denoising by nonlinear robust principal component analysis. *Advances in neural information processing systems*, *32*.

Martins, A., Vasconcelos, G., & Costa, A. C. (2017). Brick masonry veneer walls: An overview. *Journal of Building Engineering*, *9*, 29–41. https://doi.org/https://doi.org/10.1016/j.jobe.2016.11.005

Mulder, P., Machiel & Perey. (2018). Gas production and earthquakes in Groningen; reflection on economic and social consequences. *Centre for Energy Economics Research (CEER), Policy Papers (No. 3)*.

Numan, I. (2018). *Reinforcements in Opwierde-Zuid, Appingedam: the social impact on its residents* (Doctoral dissertation). University of Groningen: Faculty of Spatial Sciences. Groningen, The Netherlands.

Nurunnabi, A., Belton, D., & West, G. (2012). Robust segmentation in laser scanning 3D point cloud data. *2012 International Conference on Digital Image Computing Techniques and Applications (DICTA)*, 1–8. https://doi.org/10.1109/DICTA.2012.6411672

Oosterhoff, J. (2008). *Kracht + vorm*. Bouwen met Staal, Zoetermeer.

Petty, S. E., PE, C., Slattery, P. W., et al. (2021). Forensic Inspection Assessments of Foundation Walls. In *Forensic engineering:damage assessments for residential and commercial structures* (pp. 557–586). CRC Press. https://doi.org/https://doi.org/10.1201/9781003189305

Pishro-Nik, H. (2016). *Introduction to probability, statistics, and random processes* [Available at https://www.probabilitycourse.com]. Kappa Research LLC.

Popescu, E. G. (2014). Multiple homogeneous coordinate transformations used in photogrammetry & remote sensing. *14th International Multidisciplinary Scientific GeoConference SGEM 2014*, 239–246.

Reed, S. (2019). Earthquakes are jolting the Netherlands. Gas drilling is to blame. *The New York Times, Page 1 of edition with the headline: Temblors Expose Flaws in Gas Extraction. Published on Oct 27, 2019.*

Ringnér, M. (2008). What is principal component analysis? *Nature biotechnology*, *26*(3), 303–304. https://doi.org/https://doi.org/10.1038/nbt0308-303

Shober, A. L., Denny, G. C., Reisinger, A. J., & Bean, E. Z. (2013). Soil compaction in the urban landscape. *Rep. SL*, *317*.

Sigaran-Loria, S., C & Slob. (2019). Seismic swarms in South Limburg (The Netherlands): Tectonic or induced as coal mining lagging effect? In *Earthquake Geotechnical Engineering for Protection and Development of Environment and Constructions* (pp. 5019–5026). CRC Press.

Soudarissanane, S. (2016). *The geometry of terrestrial laser scanning; identification of errors, modeling and mitigation of scanning geometry* (Doctoral dissertation). Delft, The Netherlands, Delft University of Technology Geoscience; Remote Sensing Department. https://doi.org/10.4233/UUID:B7AE0BD3-23B8-4A8A-9B7D-5E494EBB54E5

Staatstoezicht op de Mijnen (SodM). (2021). *Staat van de Sector Voormalige steenkoolwinning* (tech. rep. No. 5). Ministerie van Economische Zaken en Klimaat. The Hague, The Netherlands.

Stilla, U., Soergel, U., & Thoennessen, U. (2003). Potential and limits of InSAR data for building reconstruction in built-up areas. *ISPRS Journal of photogrammetry and remote sensing*, *58*(1-2), 113–123. https://doi.org/https://doi.org/10.1016/S0924-2716(03)00021-2

Strang, G. (2006). *Linear Algebra and Its Applications, 4th Edition*. Cengage Learning; 4th edition (January 1, 2006).

Tabatabaei, R. (2011). Torsional vibration of eccentric building systems. In *Recent Advances in Vibration Analysis* (pp. 169–193). InTech.

Teunissen, P. J. (2000). *Adjustment theory: an introduction* [Series on Mathematical Geodesy and Positioning]. Delft University Press, Delft.

Tiberius, C., van der Marel, H., Reudink, R., & van Leijen, F. (2021). *Surveying and Mapping*. TU Delft Open. https://doi.org/https://doi.org/10.5074/T.2021.007

Tofani, V., Raspini, F., Catani, F., & Casagli, N. (2013). Persistent Scatterer Interferometry (PSI) technique for landslide characterization and monitoring. *Remote Sensing*, *5*(3), 1045–1065. https://doi.org/https://doi.org/10.3390/rs5031045

Truesdell, W., C. & Noll. (2004). *The Non-Linear Field Theories of Mechanics* (S. S. Antman, Ed.). Springer. https://doi.org/https://doi.org/10.1007/978-3-662-10388-3

Vacca, G., Mistretta, F., Stochino, F., & Dessi, A. (2016). Terrestrial laser scanner for monitoring the deformations and the damages of buildings. *23rd International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences Congress, ISPRS 2016*, *41*, 453–460. https://doi.org/https://doi.org/10.5194/isprs-archives-XLI-B5-453-2016

Viles, H., Sternberg, T., & Cathersides, A. (2011). Is ivy good or bad for historic walls? *Journal of Architectural Conservation*, *17*(2), 25–41. https://doi.org/https://doi.org/10.1080/13556207.2011.10785087

Voncken, J. H. L. (2019). Introduction—History of Coal Mining in South Limburg/Aachen Area/Campine Coalfield. In *Geology of Coal Deposits of South Limburg, The Netherlands. Including Adjacent German and Belgian Areas*. Springer. https://doi.org/https://doi.org/10.1007/978-3-030-18286-1_1

Yao, Y., Ni, J., & Li, J. (2021b). Stress-dependent water retention of granite residual soil and its implications for ground settlement. *Computers and Geotechnics*, *129*, 103835. https://doi.org/https://doi.org/10.1016/j.compgeo.2020.103835

Young, A. P., Olsen, M., Driscoll, N., Flick, R., Gutierrez, R., Guza, R., Johnstone, E., & Kuester, F. (2010). Comparison of airborne and terrestrial lidar estimates of seacliff erosion in southern California. *Photogrammetric Engineering & Remote Sensing*, *76*(4), 421–427.

Zhou, Z., Ogot, M., & Schwartz, L. (2001). A finite element analysis of the effects of an increasing angle on the tower of Pisa. *Finite elements in analysis and design*, *37*(11), 901–911. https://doi.org/https://doi.org/10.1016/S0168-874X(01)00074-9

# Bibliography of Websites

Binnenlands Bestuur. (2008). Friese Bodem Blijft Dalen na staken Gaswinning. Retrieved February 2, 2023, from https://www.binnenlandsbestuur.nl/ruimte-en-milieu/friese-bodem-blijft-dalen-na-staken-gaswinning

Boom, B. (2019). *The Netherlands in Colorized Point Clouds*, Cyclomedia. Retrieved November 16, 2022, from https://geospatialworldforum.org/speaker/presentions2019/The-Netherlands-in-Colorized-Point-Clouds-Bas_Boom.pdf

CNE News. (2022). *Church tower in German Suurhusen no longer most leaning steeple* [Published on September 9, 2022]. Retrieved March 12, 2023, from https://cne.news/article/1708-church-tower-in-german-suurhusen-no-longer-most-leaning-steeple

College Westland. (2022). Stand van zaken integrale ophoging De Lier. *Westlanders.nu*. Retrieved February 2, 2023, from https://m.westlanders.nu/politiek/stand-van-zaken-integrale-ophoging-de-lier-39872

Darwinkel, G.-J. (2020). *Aardbevingsnieuws juli 2020: beste klappen in Loppersum, Hellum en Startenhuizen*. Retrieved February 2, 2023, from https://www.rtvnoord.nl/nieuws/729629/aardbevingsnieuws-juli-2020-beste-klappen-in-loppersum-hellum-en-startenhuizen

Delft op zondag. (2011). *De Bagijnetoren is verplaatst en daar wilden veel Delftenaren getuige van zijn* [Published on February 13, 2011]. Retrieved February 3, 2023, from https://www.delftopzondag.nl/nieuws/algemeen/71178/de-bagijnetoren-is-verplaatst-en-daar-wilden-veel-delftenaren-getuige-van-zijn

Foundations & Waterproofing. (2022). What causes Foundations Walls to bulge? Retrieved March 15, 2023, from https://58foundations.com/kb/what-causes-foundation-walls-to-bulge/

Gemeente Amsterdam. (2020a). *De dansende huizen*. Retrieved February 3, 2023, from https://www.amsterdam.nl/nieuws/achtergrond/dansende-huizen/

Helz, R. (2021). InSAR—Satellite-Based Technique Captures Overall Deformation "Picture". Retrieved February 3, 2023, from https://www.usgs.gov/programs/VHP/insar-satellite-based-technique-captures-overall-deformation-picture

Maas, M. (2021). *Ook Almere Worstelt met Bodemdaling*. Retrieved February 2, 2023, from https://www.binnenlandsbestuur.nl/ruimte-en-milieu/gemeente-zien-kosten-bodemdaling-oplopen

Mariga, L. (2021). *pyRANSAC-3D*. Retrieved March 2, 2023, from https://pypi.org/project/pyransac3d/

Meershoek, P. (2016). Verzakken doen de huisjes aan de Vijzelgracht nooit meer. *Het Parool*. Retrieved February 2, 2023, from https://www.parool.nl/nieuws/verzakken-doen-de-huisjes-aan-de-vijzelgracht-nooit-meer~b6d08319/

Nederlands Centrum voor Geodesie en Geo-informatica (NCG). (2023a). Kennis en uitleg. Retrieved February 3, 2023, from https://bodemdalingskaart.nl/en-us/kennis-datacentrum/

Nederlands Centrum voor Geodesie en Geo-informatica (NCG). (2023b). Veel gestelde vragen. Retrieved February 3, 2023, from https://bodemdalingskaart.nl/nl/veel-gestelde-vragen/

Poux, F. (2021). *How to automate 3D point cloud segmentation and clustering with python*. Retrieved March 2, 2023, from https://towardsdatascience.com/how-to-automate-3d-point-cloud-segmentation-and-clustering-with-python-343c9039e4f5

SkyGeo. (2022). *Bodemdalingskaart 2.0*. Retrieved March 22, 2023, from https://bodemdalingskaart.portal.skygeo.com/portal/bodemdalingskaart/u2/viewers/basic/

SmartBricks. (2018). *Top 12 reasons of cracks in buildings*. Retrieved March 15, 2023, from https://gosmartbricks.com/reasons-cracks-in-buildings/

Stichting Kennis Centrum Aanpak Funderingsproblematiek. (2022). *Aandachtsgebieden funderingsproblemen op de Kaart*. Retrieved February 2, 2023, from https://www.kcaf.nl/funderingsviewer/

Tromp, E. (2008). *Bouwen op slappe bodems* (tech. rep. No. 429980-400-0007 v01). Deltares. Delft, The Netherlands, Deltares. https://www.cob.nl/wp-content/uploads/2021/09/bouwen_op_slappe_bodems.pdf

Vos, S. (2020). Acht huishoudens in Kerkrade geëvacueerd na grondverzakking door oude mijnschacht. *De Limburger*. Retrieved February 2, 2023, from https://www.limburger.nl/cnt/dmf20200723_00169120

# Bibliography of Figures

Coating, A. W. (2019). *Cracked wall*. https://www.allweathercoating.co.uk/wp-content/uploads/2021/12/cracked-wall-768x513.jpeg

Deensel. (2017). *Torres kio*. https://commons.wikimedia.org/wiki/File:Torres_KIO_(37719354701).jpg

Denny, G. C. (n.d.). *Soil compaction occurs when the soil structure is compressed and the pore space between particles is decreased*. https://edis.ifas.ufl.edu/image/1775504/screen

Dotson, C. (n.d.). *Cracked Brick Foundation, house settling causing bricks crack*. https://www.shutterstock.com/nl/image-photo/cracked-brick-foundation-house-settling-causing-1740828668

Earthquake Engineering ANNEXES. (2007). *Mexico City building failure associated with the torsional-translation motion*. https://www.researchgate.net/profile/Ramin-Tabatabaei/publication/221915988/figure/fig1/AS:305167363788800@1449768977100/Mexico-City-building-failure-associated-with-the-torsional-translation-motion_W640.jpg

Gemeente Amsterdam. (2020b). *Toen de huizen nog iets rechter stonden op een foto uit 1943. De piepkleine ruimte tussen de 2 huizen rechts is de Balk in 't Oogsteeg*. https://www.amsterdam.nl/publish/pages/951371/010009003224-1_klein.jpg

Geudtner, R., Dirk & Torres. (2012b). Sentinel-1 system overview and performance. *2012 IEEE International Geoscience and Remote Sensing Symposium*, 1719–1721. https://doi.org/10.1109/IGARSS.2012.6351191

Griffiths, A. (2016). *Bulging building in shanghai*. https://www.dezeen.com/2016/11/01/bricklaying-robots-bulging-masonry-facade-china-shanghai-arts-centre-archi-union-architects/

Heymann, A. (2004). *Suurhusen Kirche*. https://commons.wikimedia.org/wiki/File:Suurhusen_kirche.jpg

Kater, K. (2019). This picture was taken by an Acculevel project manager during a routine free estimate appointment. https://acculevel.com/wp-content/uploads/2019/12/Bowed-Basement-wall-FINAL-1080x809.jpg

Kumar, S. (2021). *Cracks in buildings*. https://constrofacilitator.com/wp-content/uploads/2021/06/CRACK-11.jpg

Lintula, S. (2006). *The leaning towers of Pisa*. https://en.wikipedia.org/wiki/File:Leaning_tower_of_Pisa.jpg

Roessler, B. (2022). *Leaning tower in Gau-Weinheim*. https://www.nzz.ch/panorama/neigung-von-54277-grad-schiefster-turm-der-welt-steht-offiziell-nicht-in-pisa-sondern-in-gau-weinheim-ld.1702237?reduced=true

Tiktak, A. (2017). *Soil types in the Netherlands (based on Soil map of the Netherlands 1:50000)*. (tech. rep. No. 2816). Wageningen Environmental Research. https://www.researchgate.net/publication/325718593_An_improved_soil_organic_matter_map_for_GeoPEARL_NL

Werner, A. (2022). Bowing walls: 4 causes & 5 solutions. https://www.therealsealllc.com/blog/bowing-walls-4-causes-5-solutions/

Yao, Y., Ni, J., & Li, J. (2021a). *Ground heave and settlement during rainfall and evaporation: 1-day rainfall at an intensity of 381 mm/d*. https://ars.els-cdn.com/content/image/1-s2.0-S0266352X20303980-gr11.jpg

# Appendix: Python notebook to analyze deformation

On the next pages, the notebook is shown which is used to determine the torsion and tilt angles. Besides the determination of the angles, the code is also used to construct the rasters and visualize the whole street.

# Deformation notebook to compare multiple buildings at once

This notebook is created by Kirsten Bos for the master thesis: "Analyzing deformation of buildings using LiDAR point clouds obtained by a Mobile Laser Scanning System". The code is based on the data of 3 different streets:

- Voorstraat, Franeker
- Waagstraat, Franeker
- Amstel, Amsterdam

So, the code could also be applied on different areas with some small adjustments.

This notebook can be used to check if there is deformation of a building. The code consist of different steps and can be used to either look at a single building, or to compare different buildings. The difference between two epochs of data is approximately a year. The yellow cells are cells which are recommended to be edited because they contain parameters which must be set manually.

This notebook contains data about some houses at the Amstel in Amsterdam. These houses are also known as the dancing houses. So, currently, this notebook is suited when you want to look at a single building at one side of the road.

## Libraries

In this section, the used libraries are imported.

```
[ ]: # Import libraries
import laspy
import numpy as np
import matplotlib.pyplot as plt
import open3d as o3d
import math
import scipy
import pandas as pd
from IPython.display import HTML, display

# Code to give input cells a color
def set_background(color):
    script = (
        "var cell = this.closest('.code_cell');"
        "var editor = cell.querySelector('.input_area');"
        "editor.style.background='{}';"
        "this.parentNode.removeChild(this)"
    ).format(color)
    display(HTML('<img src onerror="{}">'.format(script)))
```

## Functions

In this section, the implemented Python functions are displayed. The application, input, and output of the functions are described in red.

```
[ ]: # Parameters for the functions
cellsize_ras = 1     # 0.5 = 0.5 m = 50 cm,     1 = 1 m
```

```
[ ]: ##### --- Functions --- #####

### -- Import data from LAZ or LAS file + get coordinates, colors -- ###
def import_las(filename):
    '''
    Function to import a las-file to get coordinates and colors of the points using the scale and offset␣
 ↪parameters.

    Input:
        filename:  location and name of the file which needs to be imported   [text]

    Output:
        XYZ:     coordinates in 3D             [x,y,z]
        col:     color of coordinate points    [R,G,B]
        scale:   scale factors                 [x,y,z]
        offset:  offset factors                [x,y,z]
    '''
```

```python
    data = laspy.read('lasdata/'+filename)

    # Offset and scale parameters
    scaleX  = data.header.scale[0]
    offsetX = data.header.offset[0]
    scaleY  = data.header.scale[1]
    offsetY = data.header.offset[1]
    scaleZ  = data.header.scale[2]
    offsetZ = data.header.offset[2]

    # Calculating coordinates, colors, intensity
    p_X = np.array((data['X'] * scaleX) + offsetX)
    p_Y = np.array((data['Y'] * scaleY) + offsetY)
    p_Z = np.array((data['Z'] * scaleZ) + offsetZ)
    XYZ = np.stack([(p_X),(p_Y),(p_Z)], axis=0).transpose((1,0))
    col = (np.stack([(data['red']),(data['green']),(data['blue'])], axis=0).transpose((1,0)) / 65280)   # RGBA
    return XYZ, col, scaleX, scaleY, scaleZ, offsetX, offsetY, offsetZ


### -- Ground point removal -- ###
# Determine height threshold
def height_threshold(height_coords):
    """
    Function to determine the threshold value of the ground points with a histogram.
    For the height threshold, the next bin (to the right) must be twice as small as the previous bin.
    The maximum possible height threshold is set to 3 meter (based on Voorstraat 50, Franeker).
    The minimum possible heigth threshold is set to 0 meter.

    Input:
        height_coords:  height coordinates of the building   [z]

    Output:
        height_thres:   height threshold   [a number]
    """
    min_range, max_range = 0, 3
    bin_n = round((max_range-min_range)*5)
    f1 = plt.hist(height_coords, range=(min_range, max_range), bins=bin_n, edgecolor='black')
    plt. close()
    height_thres = 0
    for i in range(len(f1[0])-1):
        if f1[0][i] > f1[0][i+1]*2:
            height_thres = (round(f1[1][1+i],2))
    return height_thres

# Remove the ground points using a height threshold
def remove_ground(coords_xyz, col, height_thres):
    '''
    Function which removes points below a certain height threshold.
    Removed points are considered to be ground points.

    Input:
        coords_xyz:    coordinates in 3D            [x,y,z]
        col:           color of coordinate points   [R,G,B]
        height_thres:  height threshold             [a number]

    Output:
        coords_xyz:  new 3D coordinates   [x,y,z]
        col:         new colors           [R,G,B]    (new = ground points removed)
    '''
    del_rows = []
    for i in range(len(coords_xyz)):
        if (coords_xyz[i, h] <= (height_thres)):
            del_rows.append(i)
    return np.delete(coords_xyz, del_rows, axis=0), np.delete(col, del_rows, axis=0)
```

```python
### -- Segmentation of the data using RANSAC -- ###
def in_segment(coords_xyz, filename_ply, thres):
    '''
    This function determines which points are on the facade using segmentation.
    Ransac_n is 3, because 3 sample points are needed to draw a plane.
    Num_iterations is 1000. So, the random plane is sampled and verified 1000 times.

    Input:
        coords_xyz:    coordinates in 3D              [x,y,z]
        filename_ply:  filename for ply file          [text]
        thres:         value for distance_threshold   [a number]

    Output:
        inlier_cloud:  points considered as inliers   [x,y,z]
        plane_model:   plane parameters               [A,B,C,D]
        plane_W:       width coordinates of plane     [array 100x100]
        plane_H:       height coordinates of plane    [array 100x100]
        plane_L:       length coordinates of plane    [array 100x100]
        inliers:       indices of the inlier points   [array with integers]
    ---
    This function is created using the following source:
        http://www.open3d.org/docs/latest/tutorial/Basic/pointcloud.html (© Copyright 2018-2020, www.open3d.org)
    '''
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(coords_xyz)
    o3d.io.write_point_cloud(filename_ply, pcd)

    plane_model, inliers = pcd.segment_plane(distance_threshold=thres, ransac_n=3, num_iterations=1000)
    inlier_cloud = pcd.select_by_index(inliers)
    outlier_cloud = pcd.select_by_index(inliers, invert=True)
    [A,B,C,D] = plane_model
    if w == 0:
        W = np.linspace(round(min(coords_xyz[:,w]))-1,round(max(coords_xyz[:,w]))+1,100)
        H = np.linspace(round(min(coords_xyz[:,h]))-1,round(max(coords_xyz[:,h]))+1,100)
        plane_W, plane_H = np.meshgrid(W,H)
        plane_L = (-A*plane_W - C*plane_H - D) / B
    if w == 1:
        W = np.linspace(round(min(coords_xyz[:,w]))-1,round(max(coords_xyz[:,w]))+1,100)
        H = np.linspace(round(min(coords_xyz[:,h]))-1,round(max(coords_xyz[:,h]))+1,100)
        plane_W, plane_H = np.meshgrid(W,H)
        plane_L = (-B*plane_W - C*plane_H - D) / A
    return np.asarray(inlier_cloud.points), plane_model, plane_W, plane_H, plane_L, inliers


### -- Converting the data from -- ###
# Building-based to XYZ coordinate system
def whl_to_xyz(data):
    '''
    Function to convert data in whl form to xyz

    Input:
        data_whl:  coordinates in 3D   [w,h,l]

    Output:
        data_xyz:  coordinates in 3D   [x,y,z]
    '''
    data_whl = np.copy(data)
    if w==0 and l==1 and h==2:
        data_xyz = np.stack([data_whl[:,0],data_whl[:,2],data_whl[:,1]], axis=0).transpose((1,0))
    if w==1 and l==0 and h==2:
        data_xyz = np.stack([data_whl[:,2],data_whl[:,0],data_whl[:,1]], axis=0).transpose((1,0))
    data_whl[:,:3] = data_xyz
    data_xyz = data_whl
    return data_xyz
```

```python
# XYZ to building-based coordinate system
def xyz_to_whl(data):
    '''
    Function to convert data in xyz form to whl

    Input:
        data_xyz:  coordinates in 3D   [x,y,z]

    Output:
        data_whl:  coordinates in 3D   [w,h,l]
    '''
    data_xyz = np.copy(data)
    data_whl = np.stack([data_xyz[:,w],data_xyz[:,h],data_xyz[:,l]], axis=0).transpose((1,0))
    data_xyz[:,:3] = data_whl
    data_whl = data_xyz
    return data_whl


### -- Computation of angles and transformation of the data (for torsion and tilt) -- ##
# Compute transformation angle (for transformation of original to building-based)
def transform_angle(ABCD_e1, ABCD_e2):
    '''
    Function to compute the torsion angle of the plane, also known as the transformation angle.
    The angle is computed using the plane parameters of the segmentation function (the plane model of RANSAC).
    If the computed angle is above 90 degrees, 90 degrees are subtracted to make the angle smaller.
    0 represents the ideal plane, 1 is epoch 1, and 2 is epoch 2.

    Input:
        ABCD_e1:  plane parameters of epoch 1   [A,B,C,D]
        ABCD_e2:  plane parameters of epoch 2   [A,B,C,D]

    Output:
        tors01:  torsion angle of epoch 1 with respect to ideal vector  [degrees]
        tors02:  torsion angle of epoch 2 with respect to ideal vector  [degrees]
        tors12:  torsion angle of epoch 2 with respect to epoch 2       [degrees]
    '''
    a_i1 = np.array([(ABCD_e1[1]), (ABCD_e1[0])])
    b_i1 = np.array([1, 0])
    idealVSe1 = np.arccos( np.dot(a_i1,b_i1) / (np.sqrt(sum(a_i1**2)) * np.sqrt(sum(b_i1**2))) )
    a_i2 = np.array([(ABCD_e2[1]), (ABCD_e2[0])])
    b_i2 = np.array([1, 0])
    idealVSe2 = np.arccos( np.dot(a_i2,b_i2) / (np.sqrt(sum(a_i2**2)) * np.sqrt(sum(b_i2**2))) )
    a_12 = np.array([(ABCD_e1[1]), (ABCD_e1[0])])
    b_12 = np.array([(ABCD_e2[1]), (ABCD_e2[0])])
    e1VSe2 = np.arccos( np.dot(a_12,b_12) / (np.sqrt(sum(a_12**2)) * np.sqrt(sum(b_12**2))) )

    tors01, tors02, tors12 = np.degrees(idealVSe1), np.degrees(idealVSe2), np.degrees(e1VSe2)
    if tors01>90:
        tors01 -= 90
    if tors02>90:
        tors02 -= 90
    if tors12>90:
        tors12 -= 90
    return tors01, tors02, tors12

# Transform the data (from original to building-based)
def tors_transform(data_xyz, thet_tors):
    """
    Function to rotate the data in torsion direction to make torsion angle of epoch 1 zero.
    (Useful to transform from original to building-based coordinate system.)

    Input:
        data_xyz:    3D coordinates of the original dataset (not rotated)   [x,y,z]
        theta_tors:  torsion angle of epoch 1 with respect to ideal vector  [degrees]

    Output:
        rotated_xyz:  3D coordinates of the rotated original dataset   [x,y,z]
        rot_matrix2:  transformation matrix to rotate the data points  [degrees]
    """
    # Rotate the data points with a certain torsion angle
    theta       = -thet_tors
    rot_matrix  = np.array([[np.cos(theta),-np.sin(theta),0], [-np.sin(theta),np.cos(theta),0], [0,0,1]])
    rotated_xyz = (data_xyz - (np.mean(data_xyz, axis=0))) @ rot_matrix + (np.mean(data_xyz, axis=0))
    if (np.max(data_xyz[:,1])-np.min(data_xyz[:,1])) < (np.max(rotated_xyz[:,1])-np.min(rotated_xyz[:,1])):
        theta = thet_tors
        rot_matrix  = np.array([[np.cos(theta),-np.sin(theta),0], [-np.sin(theta),np.cos(theta),0], [0,0,1]])
        rotated_xyz = (data_xyz - (np.mean(data_xyz, axis=0))) @ rot_matrix + (np.mean(data_xyz, axis=0))

    # Translate the data points to the original center
    tx = (np.mean((data_xyz @ rot_matrix)[:,0])-np.mean(data_xyz[:,0]))
    ty = (np.mean((data_xyz @ rot_matrix)[:,1])-np.mean(data_xyz[:,1]))

    # Transformation matrix to transform the data points
    rot_matrix2 = np.array([[np.cos(theta),-np.sin(theta),0,0],[-np.sin(theta),np.cos(theta),0,0],[0,0,1,0],
                            [-tx,-ty,0,1]])
    return rotated_xyz, rot_matrix2

# Compute torsion angle
def tors_angle(ABCD_e1, ABCD_e2):
    '''
    Function to compute the torsion angle of the plane.
    The plane parameters are obtained by using the chosen plane fitting algorithm (OLS, TLS, PCA).

    Input:
        ABCD_e1:  plane parameters of epoch 1   [A,B,C,D]
        ABCD_e2:  plane parameters of epoch 2   [A,B,C,D]

    Output:
        tors_idealVSe1:  torsion angle of epoch 1 with respect to ideal vector  [degrees]
        tors_idealVSe2:  torsion angle of epoch 2 with respect to ideal vector  [degrees]
        tors_e1VSe2:     torsion angle of epoch 2 with respect to epoch 2       [degrees]
    '''
    a_i1 = np.array([abs(ABCD_e1[1]), abs(ABCD_e1[0])])
    b_i1 = np.array([1, 0])
    tors_idealVSe1 = np.arccos( np.dot(a_i1,b_i1) / (np.sqrt(sum(a_i1**2)) * np.sqrt(sum(b_i1**2))) )
    a_i2 = np.array([abs(ABCD_e2[1]), abs(ABCD_e2[0])])
    b_i2 = np.array([1, 0])
    tors_idealVSe2 = np.arccos( np.dot(a_i2,b_i2) / (np.sqrt(sum(a_i2**2)) * np.sqrt(sum(b_i2**2))) )
    a_12 = np.array([abs(ABCD_e1[1]), abs(ABCD_e1[0])])
    b_12 = np.array([abs(ABCD_e2[1]), abs(ABCD_e2[0])])
    tors_e1VSe2 = np.arccos( np.dot(a_12,b_12) / (np.sqrt(sum(a_12**2)) * np.sqrt(sum(b_12**2))) )
    return np.degrees(tors_idealVSe1), np.degrees(tors_idealVSe2), np.degrees(tors_e1VSe2)

# Compute tilt angle
def tilt_angle(ABCD_e1, ABCD_e2):
    '''
    Function to compute the tilt angle of the plane.
    The plane parameters are obtained by using the chosen plane fitting algorithm (OLS, TLS, PCA).

    Input:
        ABCD_e1:  plane parameters of epoch 1   [A,B,C,D]
        ABCD_e2:  plane parameters of epoch 2   [A,B,C,D]

    Output:
        tilt_idealVSe1:  tilt angle of epoch 1 with respect to ideal vector  [degrees]
        tilt_idealVSe2:  tilt angle of epoch 2 with respect to ideal vector  [degrees]
        tilt_e1VSe2:     tilt angle of epoch 2 with respect to epoch 2       [degrees]
    '''
    a_i1 = np.array([abs(ABCD_e1[1]), abs(ABCD_e1[2])])
    b_i1 = np.array([1, 0])
    tilt_idealVSe1 = np.arccos( np.dot(a_i1,b_i1) / (np.sqrt(sum(a_i1**2)) * np.sqrt(sum(b_i1**2))) )
    a_i2 = np.array([abs(ABCD_e2[1]), abs(ABCD_e2[2])])
    b_i2 = np.array([1, 0])
    tilt_idealVSe2 = np.arccos( np.dot(a_i2,b_i2) / (np.sqrt(sum(a_i2**2)) * np.sqrt(sum(b_i2**2))) )
    a_12 = np.array([abs(ABCD_e1[1]), abs(ABCD_e1[2])])
    b_12 = np.array([abs(ABCD_e2[1]), abs(ABCD_e2[2])])
    tilt_e1VSe2 = np.arccos( np.dot(a_12,b_12) / (np.sqrt(sum(a_12**2)) * np.sqrt(sum(b_12**2))) )
    return np.degrees(tilt_idealVSe1), np.degrees(tilt_idealVSe2), np.degrees(tilt_e1VSe2)

# Transform the data from dataset with tilt to a dataset without tilt
def tilt_transform(data_xyz, thet_tilt):
    """
    Function to rotate the data in tilt direction to make tilt angle zero.

    Input:
        data_xyz:    3D coordinates of the dataset not rotated in tilt direction   [x,y,z]
        theta_tilt:  tilt angle of epoch 1 with respect to ideal vector            [degrees]

    Output:
        rotated_xyz:  3D coordinates of the rotated original dataset   [x,y,z]
        rot_matrix2:  transformation matrix to rotate the data points  [degrees]
    """
    # Rotate the data points with a certain tilt angle
    theta       = thet_tilt
    rot_matrix  = np.array([[1,0,0],
                            [0,np.cos(theta),np.sin(theta)],
                            [0,-np.sin(theta),np.cos(theta)]])
    rotated_xyz = (data_xyz - (np.mean(data_xyz, axis=0))) @ rot_matrix + (np.mean(data_xyz, axis=0))
    if (np.max(data_xyz[:,w])-np.min(data_xyz[:,w])) < (np.max(rotated_xyz[:,w])-np.min(rotated_xyz[:,w])):
        theta = thet_tilt
        rot_matrix  = np.array([[1,0,0],
                            [0,np.cos(theta),np.sin(theta)],
                            [0,-np.sin(theta),np.cos(theta)]])
        rotated_xyz = (data_xyz - (np.mean(data_xyz, axis=0))) @ rot_matrix + (np.mean(data_xyz, axis=0))

    # Translate the data points to the original center
    tx = (np.mean((data_xyz @ rot_matrix)[:,0])-np.mean(data_xyz[:,0]))
    tz = (np.mean((data_xyz @ rot_matrix)[:,2])-np.mean(data_xyz[:,2]))

    # Transformation matrix to transform the data points
    rot_matrix2 = np.array([[np.cos(theta),0,-np.sin(theta),0], [0,1,0,0], [np.sin(theta),0,np.cos(theta),0],
                            [-tx,0,-tz,1]])
    return rotated_xyz, rot_matrix2


# Compute plane parameters and coordinates
# Function to create a plane using ordinary least squares
def ols_to_plane(coords_whl):
    '''
    Function to compute ordinary least squares
    Error measured only in l-direction

    Input:
        coords_whl:  coordinates of points     [w,h,l]

    Output:
        plane_W:  width coordinates of plane    [array 100x100]
        plane_H:  height coordinates of plane   [array 100x100]
        plane_L:  length coordinates of plane   [array 100x100]
        ABC:      plane parameters              [A,B,C]
        e_hat:    minimized residuals, perpendicular distance to fitted model        [1D array]
        RMSE:     root mean squared error - precision of residuals based on A,B,C,D  [number]
        normal_vec:  normal vector of the plane                                      [w_c,l_c,h_c,A,B,C]
    '''
    # whl -> wlh
    wlh = np.stack([coords_whl[:,0],coords_whl[:,2],coords_whl[:,1]], axis=0).transpose((1,0))

    # Calculate mean
    mW, mL, mH = np.mean(wlh[:,0]), np.mean(wlh[:,1]), np.mean(wlh[:,2])

    # Assign/compute values
    dW, dL, dH = wlh[:,0]-mW, wlh[:,1]-mL, wlh[:,2]-mH
    WW, LL, HH = sum(dW**2), sum(dL**2), sum(dH**2)
    WL, WH, LH = sum(dW*dL), sum(dW*dH), sum(dL*dH);
    ONE = sum(np.ones(len(wlh[:,0])))

    # Put values in matrix/vector
    M_sum = np.array([[WW, sum(dW), WH],   [sum(dW), ONE, sum(dH)],   [WH, sum(dH), HH]])
    y_sum = np.array([WL, sum(dL), LH]).T

    # Solve M * [A,B,C] = y   ->   [A,B,C] = M^-1 @ y
    ABC = np.linalg.inv(M_sum) @ y_sum
    A, B, C = ABC

    # Errors
    A_matrix = np.stack([wlh[:,0]-mW, np.ones(len(wlh[:,1])), wlh[:,2]-mH], axis=0).transpose((1,0))
    L_hat = A_matrix @ ABC + mL
    e_hat = wlh[:,1] - L_hat
    RMSE  = np.sqrt((e_hat.T @ e_hat)/(len(wlh[:,0])-3))

    # Create coordinates of the plane
    w_c = np.linspace(round(w_min)-1, round(w_max)+1, 100)
    h_c = np.linspace(round(h_min)-1, round(h_max)+1, 100)
    [plane_W,plane_H] = np.meshgrid(w_c, h_c);
    plane_L = (A*(plane_W-mW) + B + C * (plane_H-mH)) + mL

    normal_vec = np.array([np.mean(plane_W), np.mean(plane_L), np.mean(plane_H), A, B, C])
    return plane_W, plane_H, plane_L, ABC, e_hat, RMSE, normal_vec

# Function to create a plane using least squares homogeneous equation + ABCD
def tls_to_plane(coords_whl):
    '''
    Function computing plane using homogeneous least squares

    Input:
        coords_whl:  coordinates of points     [w,h,l]

    Output:
        plane_W:  width coordinates of plane    [array 100x100]
        plane_H:  height coordinates of plane   [array 100x100]
        plane_L:  length coordinates of plane   [array 100x100]
        ABCD:     plane parameters              [A,B,C,D]
        e_hat:    minimized residuals, perpendicular distance to fitted model        [1D array]
        RMSE:     root mean squared error - precision of residuals based on A,B,C,D  [number]
        eigvec_svd:  eigenvectors                                                    [array 3x3]
        normal_vec:  normal vector of the plane                                      [w_c,l_c,h_c,A,B,C]
    '''
    # whl -> wlh
    A_matrix=np.stack([coords_whl[:,0],coords_whl[:,2],coords_whl[:,1],np.ones(len(coords_whl))],axis=0).
↪transpose((1,0))

    # Determine plane parameters
    ATA = A_matrix.T @ A_matrix
    eigvec_svd = np.linalg.eig(ATA)[1].T     # Eigenvectors
    eigval_svd = np.linalg.eig(ATA)[0]       # Eigenvalues
    ABCD = eigvec_svd[np.argmin(eigval_svd)]
    A, B, C, D = ABCD                        # Ax+By+Cy+D=0
    normal = ABCD[:3]
```

```python
    # Errors
    e_hat = A_matrix @ ABCD / (np.sqrt(A**2 + B**2 + C**2) ) # e_hat = A_matrix @ ABCD / sqrt(A^2 + B^2 + C^2)
    RMSE  = np.sqrt((e_hat.T @ e_hat)/(len(coords_whl[:,1])-4+1))  # RMSE = sqrt((e_hat.T @ e_hat)/ (n-m+1))
                                                                 # n:  datapoints,  m:   parameters (4)

    # Plane coordinates
    w = np.linspace(round(w_min)-1,round(w_max)+1,100)
    h = np.linspace(round(h_min)-1,round(h_max)+1,100)
    plane_W,plane_H = np.meshgrid(w,h)
    plane_L = (-A*(plane_W) - C*(plane_H) - D) / B
    normal_vec = np.array([np.mean(plane_W), np.mean(plane_L), np.mean(plane_H), A, B, C])
    return plane_W, plane_H, plane_L, ABCD, e_hat, RMSE, eigvec_svd, normal_vec

# Function to create a plane using PCA
def pca_to_plane(coords_whl):
    '''
    This function applies PCA and gives coordinates of the plane and the normal vector

    Input:
        coords_whl:  coordinates in 3D  [w,h,l]

    Output:
        plane_W:     width coordinates of plane    [array 100x100]
        plane_H:     height coordinates of plane   [array 100x100]
        plane_L:     length coordinates of plane   [array 100x100]
        ABCD:        plane parameters              [A,B,C,D]
        e_hat:       minimized residuals, perpendicular distance to fitted model        [1D array]
        RMSE:        root mean squared error - precision of residuals based on A,B,C,D  [number]
        vec:         eigenvectors sorted on eigenvalues (high to low)                   [array 3x3]
        normal_vec:  normal vector of the plane                                         [w_c,l_c,h_c,a,b,c]
    ---
    Source: https://stackoverflow.com/questions/49957601/how-can-i-draw-3d-plane-using-pca-in-python
    '''
    # whl -> wlh
    wlh = np.stack([coords_whl[:,0],coords_whl[:,2],coords_whl[:,1]], axis=0).transpose((1,0))

    # Rescale coordinates to the same scale, so there is less chance of a dominant parameter
    M = np.array([np.mean(wlh[:,0]), np.mean(wlh[:,1]), np.mean(wlh[:,2])])   # mean of the data [w0,l0,h0]
    W_c = (wlh-M)   # normalized dataset

    # Compute covariance matrix, eigenvalues and eigenvectors (coordinate left to right, vector top to bottom)
    C_x = W_c.T @ W_c                      # Covariance matrix
    e_values, eig_vec = np.linalg.eig(C_x)
    sum_values = np.sum(e_values)

    # Order eigenvectors (from highest to lowest eigenvalue)
    order = (sorted(e_values, reverse=True))
    o_idx = []
    for i in order:
        for j in range(3):
            if i == e_values[j]:
                o_idx.append(j)
    vec = np.stack([(eig_vec[:,o_idx[0]]),(eig_vec[:,o_idx[1]]),(eig_vec[:,o_idx[2]])],axis=0).transpose((1,0))

    # The eigenvectors with the lowest eigenvalues is selected as normal vector
    normal = vec[:,2]   # (a, b, c)
    A,B,C = normal

    # Calculate d, where D = A * w0 + B * l0 + C * h0
    D = -M @ (normal)
    ABCD = np.array([A,B,C,D])

    # Errors
    e_hat = W_c @ normal                               # e_hat = X_c @ normal
    RMSE  = np.sqrt((e_hat.T @ e_hat)/(len(wlh[:,1])-3))  # RMSE = sqrt((e_hat.T @ e_hat)/ (n-m))
                                                        # n:  datapoints,  m:  parameters (3)
    # Compute plane coordinates:  A*w + B*l + C*h + D = 0
    w = np.linspace(round(w_min)-1,round(w_max)+1,100)
    h = np.linspace(round(h_min)-1,round(h_max)+1,100)
    plane_W,plane_H = np.meshgrid(w,h)
    plane_L = (-A*(plane_W) - C*(plane_H) - D) / B
    normal_vec = np.array([np.mean(plane_W), np.mean(plane_L), np.mean(plane_H), A, B, C])
    return plane_W, plane_H, plane_L, ABCD, e_hat, RMSE, vec, normal_vec


### Create raster to analyze the data
# Construct a empty raster
def rasterpoints_WH(coords_whl, cellsize):
    """
    Function to determine the coordinates of the raster points

    Input:
        coords_whl:    coordinates in 3D         [w,h,l]
        cellsize:  cellsize of rastercells  [number with [m] as unit]

    Output:
        rasterpoints:  all middle points of the cells  [w,h,(l),(index),(R),(G),(B)]
        W_n_cells:     amount of columns               [integer]
        H_n_cells:     amount of rows                  [integer]
    """
    points  = coords_whl
    width_r = points[:,0]
    heigh_r = points[:,1]
    W_n_cells = math.ceil((w_max - w_min)/cellsize)
    H_n_cells = math.ceil((h_max - h_min)/cellsize)
    rasterpoints = []
    w_r = np.min(width_r)
    h_r = np.min(heigh_r)
    for i in range(W_n_cells):
        w_r = (w_min + 0.5*cellsize) + i * cellsize
        for j in range(H_n_cells):
            h_r = (h_min + 0.5*cellsize) + j * cellsize
            rasterpoints.append([w_r, h_r, np.nan, np.nan, np.nan, np.nan, np.nan])
    return np.array(rasterpoints), W_n_cells, H_n_cells

# Assign values to the raster
def assign_points(raster, colors, coords_xyz, k_number, cellsize):
    '''
    Function to assign the colors of the data and the L coordinate to the constructed raster.

    Input:
        raster:      X and Z position, other part empty   [W,H,nan,nan,nan,nan,nan]
        colors:      color of coordinate points           [R,G,B]
        coords_xyz:  coordinates in 3D                    [x,y,z]
        k_number:    number of neighbors                  [number, unitless]
        cellsize:    cellsize of rastercells              [number with [m] as unit]

    Output:
        raster:  raster is filled with data   [W,H,Red,Green,Blue,L,d]
    '''
    list_pts = np.stack([(coords_xyz[:,w]),(coords_xyz[:,2])], axis=0).transpose((1,0))
    kd = scipy.spatial.KDTree(list_pts)
    d, i = kd.query(raster[:, :2], k=k_number)
    for n in range(len(d)):
        if np.mean(d[n]) <= cellsize + (cellsize/2):
            raster[n,2] = np.max(colors[i[n],0])
            raster[n,3] = np.max(colors[i[n],1])
            raster[n,4] = np.max(colors[i[n],2])
            raster[n,5] = np.mean(coords_xyz[i[n],l])
            raster[n,6] = np.mean(d[n])
    return raster
```

## 1. Preprocessing

Insert the filenames in the cell below

```
[ ]: # Import data: insert filename and title for the plots -> Amstel, Amsterdam
filename_1 = ['amstel_amsterdam/amstel82_2023_manual.las',  'amstel_amsterdam/amstel84_2023.las',
              'amstel_amsterdam/amstel100_2023.las',        'amstel_amsterdam/amstel102_2023.las',
              'amstel_amsterdam/amstel104_2023.las',        'amstel_amsterdam/amstel106_2023.las',
              'amstel_amsterdam/amstel108_2023.las',        'amstel_amsterdam/amstel110_2023.las',
              'amstel_amsterdam/amstel112_2023.las']
filename_2 = None

# Set the title names of the plots
epoch_1 = np.array(['Amstel 82 (2023). ', 'Amstel 84 (2023). ', 'Amstel 100 (2023).', 'Amstel 102 (2023).',␣
 ↪'Amstel 104 (2023).', 'Amstel 106 (2023).', 'Amstel 108 (2023).', 'Amstel 110 (2023).', 'Amstel 112 (2023).'])
epoch_2 = np.array(['Amstel 82 (2023). ', 'Amstel 84 (2023). ', 'Amstel 100 (2023).', 'Amstel 102 (2023).',␣
 ↪'Amstel 104 (2023).', 'Amstel 106 (2023).', 'Amstel 108 (2023).', 'Amstel 110 (2023).', 'Amstel 112 (2023).'])
name_dataset = np.array(['Amstel 82  ',  'Amstel 84  ',  'Amstel 100 ',  'Amstel 102 ',   'Amstel 104 ', ␣
 ↪'Amstel 106 ',  'Amstel 108 ',  'Amstel 110 ',  'Amstel 112 '])
name_street = name_dataset
home_numbers = np.array([82, 84, 100, 102, 104, 106, 108, 110, 112])
epoch_data = [1898, 1744, 1742, 1710, 1710, 1905, 1725, 1706, 1727]
epoch1_year = 2023
epoch2_year = None

set_background('wheat')
```

```
[ ]: # Show plots    True if you want to see plots, False if you don't want to see plots
plot=True

if plot==True:
    plot_front=True
    plot_groundremoval=True
    plot_segment=True
    plot_plane=True
    plot_torsion=True
    plot_tilt=True
else:
    plot_front=False
    plot_groundremoval=False
    plot_segment=False
    plot_plane=False
    plot_torsion=False
    plot_tilt=False

set_background('wheat')
```

```
[ ]: # Import data + calculate coordinates, colors + show imported data (top view)
XYZ_house_1, XYZ_house_2, col_house_1, col_house_2 = [], [], [], []

for nmbr in range(len(filename_1)):
    XYZ_house_1.append(import_las(filename_1[nmbr])[0])
    col_house_1.append(import_las(filename_1[nmbr])[1])
    xyz_plot_1 = np.array(XYZ_house_1, dtype=object)[nmbr]
    col_plot_1 = np.array(col_house_1, dtype=object)[nmbr]

    if filename_2 != None:
        XYZ_house_2.append(import_las(filename_2[nmbr])[0])
        col_house_2.append(import_las(filename_2[nmbr])[1])
        xyz_plot_2 = np.array(XYZ_house_2, dtype=object)[nmbr]
        col_plot_2 = np.array(col_house_2, dtype=object)[nmbr]

    if plot == True:
        if filename_2 != None:
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)
            fig.suptitle("Top view of the data of "+str(name_dataset[nmbr]), fontsize="x-large", y=1.05)
            ax1.scatter(xyz_plot_1[:,0], xyz_plot_1[:,1], c=col_plot_1, s=1)
            ax1.grid()
            ax1.set_xticks(np.arange(round(min(xyz_plot_1[:,0])), round(max(xyz_plot_1[:,0]))+2, 5))
            ax1.set_yticks(np.arange(round(min(xyz_plot_1[:,1])), round(max(xyz_plot_1[:,1]))+2, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel('X (m)')
            ax1.set_ylabel('Y (m)')
            ax1.set_title('Image '+str(epoch_1[nmbr]))

            ax2.scatter(xyz_plot_2[:,0], xyz_plot_2[:,1], c=col_plot_2, s=1)
            ax2.grid()
            ax2.set_xticks(np.arange(round(min(xyz_plot_1[:,0])), round(max(xyz_plot_1[:,0]))+2, 5))
            ax2.set_yticks(np.arange(round(min(xyz_plot_1[:,1])), round(max(xyz_plot_1[:,1]))+2, 5))
            ax2.ticklabel_format(useOffset=False)
            ax2.set_aspect('equal')
            ax2.set_xlabel('X (m)')
            ax2.set_title('Image '+str(epoch_2[nmbr]))
            plt.show()
        else:
            fig, (ax1) = plt.subplots(1, 1, figsize=(4, 4), sharex=True, sharey=True)
            fig.suptitle("Top view of the data of "+str(name_dataset[nmbr]), fontsize="x-large", y=1.05)
            ax1.scatter(xyz_plot_1[:,0], xyz_plot_1[:,1], c=col_plot_1, s=1)
            ax1.grid()
            ax1.set_xticks(np.arange(round(min(xyz_plot_1[:,0])), round(max(xyz_plot_1[:,0]))+2, 5))
            ax1.set_yticks(np.arange(round(min(xyz_plot_1[:,1])), round(max(xyz_plot_1[:,1]))+2, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel('X (m)')
            ax1.set_ylabel('Y (m)')
            ax1.set_title('Image '+str(epoch_1[nmbr]))
            plt.show()
```

Insert the location of the building (left or right side of the road) in the cell below

```
[ ]: # Direction -> waagstraat
         # True = coordinates at the left part of the facade are higher than the coordinates of the right part
         # False = coordinates at the left part of the facade are lower than the coordinates of the right part
     Dir = [True, True, True, True, True, True, True, True, True]

     # Set which coordinate represents the width (w) and the depth/length (l) of the facade
     wlh = []
     for i in range(len(Dir)):
         wlh.append([0,1,2])
     wlh = np.array(wlh)

     set_background('wheat')
```

```
[ ]: # Plot image of the facade of the building
     facade_e1, facade_e2 = [], []

     for nmbr in range(len(filename_1)):
         w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
         if (w==0 and l==1 and h==2):
             label_x = 'W' #'X'
             label_y = 'L' #'Y'
             label_z = 'H' #'Z'
         if (w==1 and l==0 and h==2):
             label_x = 'L'#'Y'
             label_y = 'W'#'X'
             label_z = 'H'#'Z'

         xyz_plot_1 = np.array(XYZ_house_1, dtype=object)[nmbr]
         col_plot_1 = np.array(col_house_1, dtype=object)[nmbr]
```

```
    if filename_2 != None:
        xyz_plot_2 = np.array(XYZ_house_2, dtype=object)[nmbr]
        col_plot_2 = np.array(col_house_2, dtype=object)[nmbr]

    # Determine min width, max width, min height, max height
    if filename_2 != None:
        cells_input = [min([min(xyz_plot_1[:,w]), min(xyz_plot_2[:,w])]),
                       max([max(xyz_plot_1[:,w]), max(xyz_plot_2[:,w])]),
                       min([min(xyz_plot_1[:,h]), min(xyz_plot_2[:,h])]),
                       max([max(xyz_plot_1[:,h]), max(xyz_plot_2[:,h])]),
                       min([min(xyz_plot_1[:,l]), min(xyz_plot_2[:,l])]),
                       max([max(xyz_plot_1[:,l]), max(xyz_plot_2[:,l])])]
    else:
        cells_input = [min([min(xyz_plot_1[:,w])]),
                       max([max(xyz_plot_1[:,w])]),
                       min([min(xyz_plot_1[:,h])]),
                       max([max(xyz_plot_1[:,h])]),
                       min([min(xyz_plot_1[:,l])]),
                       max([max(xyz_plot_1[:,l])])]
    w_min, w_max, h_min, h_max, l_min, l_max = cells_input

    facade_e1.append([filename_1[nmbr], home_numbers[nmbr], xyz_plot_1, col_plot_1])

    if filename_2 != None:
        facade_e2.append([filename_1[nmbr], home_numbers[nmbr], xyz_plot_2, col_plot_2])
    if plot_front == True:
        if filename_2 != None:
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

            ax1.scatter(xyz_plot_1[:,w], xyz_plot_1[:,h], c=col_plot_1, s=1)
            ax1.grid()
            ax1.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel(str(label_x)+' (m)')
            ax1.set_ylabel(str(label_z)+' (m)')
            ax1.set_title('Input image '+str(epoch_1[nmbr]))
            if Dir[nmbr] == True:
                ax1.invert_xaxis()

            ax2.scatter(xyz_plot_2[:,w], xyz_plot_2[:,h], c=col_plot_2, s=1)
            ax2.grid()
            ax2.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.ticklabel_format(useOffset=False)
            ax2.set_aspect('equal')
            ax2.set_xlabel(str(label_x)+' (m)')
            ax2.set_title('Input image '+str(epoch_2[nmbr]))
            plt.show()
        else:
            fig, (ax1) = plt.subplots(1, 1, figsize=(4, 4), sharex=True, sharey=True)

            ax1.scatter(xyz_plot_1[:,w], xyz_plot_1[:,h], c=col_plot_1, s=1)
            ax1.grid()
            ax1.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel(str(label_x)+' (m)')
            ax1.set_ylabel(str(label_z)+' (m)')
            ax1.set_title('Input image '+str(epoch_1[nmbr]))
            if Dir[nmbr] == True:
                ax1.invert_xaxis()
            plt.show()
```

## 1.1. Ground removal

Function automatically determines which points are considered groundpoints and these points are removed from the data.

[ ]:
```python
# Remove ground points with histogram and plot result
XYZ_house_1_ng, XYZ_house_2_ng, col_house_1_ng, col_house_2_ng, data_ground = [], [], [], [], []

for nmbr in range(len(filename_1)):
    xyz_plot_1 = np.array(XYZ_house_1, dtype=object)[nmbr]
    col_plot_1 = np.array(col_house_1, dtype=object)[nmbr]
    if filename_2 != None:
        xyz_plot_2 = np.array(XYZ_house_2, dtype=object)[nmbr]
        col_plot_2 = np.array(col_house_2, dtype=object)[nmbr]

    w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
    if (w==0 and l==1 and h==2):
        label_x = 'w' #'X'
        label_y = 'l' #'Y'
        label_z = 'h' #'Z'
    if (w==1 and l==0 and h==2):
        label_x = 'l'#'Y'
        label_y = 'w'#'X'
        label_z = 'h'#'Z'
    # Determine min width, max width, min height, max height
    if filename_2 != None:
        cells_input = [min([min(xyz_plot_1[:,w]), min(xyz_plot_2[:,w])]),
                       max([max(xyz_plot_1[:,w]), max(xyz_plot_2[:,w])]),
                       min([min(xyz_plot_1[:,h]), min(xyz_plot_2[:,h])]),
                       max([max(xyz_plot_1[:,h]), max(xyz_plot_2[:,h])]),
                       min([min(xyz_plot_1[:,l]), min(xyz_plot_2[:,l])]),
                       max([max(xyz_plot_1[:,l]), max(xyz_plot_2[:,l])])]
    else:
        cells_input = [min([min(xyz_plot_1[:,w])]),
                       max([max(xyz_plot_1[:,w])]),
                       min([min(xyz_plot_1[:,h])]),
                       max([max(xyz_plot_1[:,h])]),
                       min([min(xyz_plot_1[:,l])]),
                       max([max(xyz_plot_1[:,l])])]
    w_min, w_max, h_min, h_max, l_min, l_max = cells_input

    if filename_2 != None:
        height_par = np.min([height_threshold(xyz_plot_1[:,h]), height_threshold(xyz_plot_2[:,h])])
        XYZ_house_1_ng.append(remove_ground(xyz_plot_1, col_plot_1, height_par)[0])
        col_house_1_ng.append(remove_ground(xyz_plot_1, col_plot_1, height_par)[1])
        XYZ_house_2_ng.append(remove_ground(xyz_plot_2, col_plot_2, height_par)[0])
        col_house_2_ng.append(remove_ground(xyz_plot_2, col_plot_2, height_par)[1])
        xyz_plot_1_ng = np.array(XYZ_house_1_ng, dtype=object)[nmbr]
        xyz_plot_2_ng = np.array(XYZ_house_2_ng, dtype=object)[nmbr]
        col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
        col_plot_2_ng = np.array(col_house_2_ng, dtype=object)[nmbr]
        per1 = round((len(xyz_plot_1)-len(xyz_plot_1_ng))/len(xyz_plot_1)*100,1)
        per2 = round((len(xyz_plot_2)-len(xyz_plot_2_ng))/len(xyz_plot_2)*100,1)
        data_ground.append([len(xyz_plot_1), len(xyz_plot_2), height_par, per1, per2])
    else:
        height_par = np.min([height_threshold(xyz_plot_1[:,h])])
        XYZ_house_1_ng.append(remove_ground(xyz_plot_1, col_plot_1, height_par)[0])
        col_house_1_ng.append(remove_ground(xyz_plot_1, col_plot_1, height_par)[1])
        xyz_plot_1_ng = np.array(XYZ_house_1_ng, dtype=object)[nmbr]
        col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
        per1 = round((len(xyz_plot_1)-len(xyz_plot_1_ng))/len(xyz_plot_1)*100,1)
        data_ground.append([len(xyz_plot_1), height_par, per1])

    if plot_groundremoval == True:
        if filename_2 != None:
            fig, ((ax1, ax2, ax3, ax4)) = plt.subplots(1, 4, figsize=(16, 5), sharex=True, sharey=True)
            fig.tight_layout(pad=5)
            fmt = '%1.0f'

            fig.suptitle("Removing groundpoints using a threshold value for Z", fontsize="x-large", y=1.01)
            im1=ax1.scatter(xyz_plot_1[:,w], xyz_plot_1[:,l], c=xyz_plot_1[:,h], s=1)
            clb1=fig.colorbar(im1, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb1.set_label(str(label_z)+' (m)')
            ax1.grid()
            ax1.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_title('Top view '+str(epoch_1[nmbr]))
            ax1.set_xlabel(str(label_x)+' (m)')
            ax1.set_ylabel(str(label_y)+' (m)')
            if Dir[nmbr] == True:
                ax1.invert_xaxis()

            im2=ax2.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c=xyz_plot_1_ng[:,h], s=1)
            clb2=fig.colorbar(im2, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb2.set_label(str(label_z)+' (m)')
            ax2.grid()
            ax2.set_yticks(np.arange(round(l_min), round(l_max)+1, 2))
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.ticklabel_format(useOffset=False)
            ax2.set_aspect('equal')
            ax2.set_title(str(epoch_1[nmbr])+' Ground removed')
            ax2.set_xlabel(str(label_x)+' (m)')

            im3=ax3.scatter(xyz_plot_2[:,w], xyz_plot_2[:,l], c=xyz_plot_2[:,h], s=1)
            clb3=fig.colorbar(im3, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb3.set_label(str(label_z)+' (m)')
            ax3.grid()
            ax3.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax3.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax3.ticklabel_format(useOffset=False)
            ax3.set_aspect('equal')
            ax3.set_title('Top view '+str(epoch_2[nmbr]))
            ax3.set_xlabel(str(label_x)+' (m)')

            im4=ax4.scatter(xyz_plot_2_ng[:,w], xyz_plot_2_ng[:,l], c=xyz_plot_2_ng[:,h], s=1)
            clb4=fig.colorbar(im4, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb4.set_label(str(label_z)+' (m)')
            ax4.grid()
            ax4.set_yticks(np.arange(round(l_min), round(l_max)+1, 2))
            ax4.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax4.ticklabel_format(useOffset=False)
            ax4.set_aspect('equal')
            ax4.set_title(str(epoch_2[nmbr])+' Ground removed')
            ax4.set_xlabel(str(label_x)+' (m)')
            plt.show()
        else:
            fig, ((ax1, ax2)) = plt.subplots(1, 2, figsize=(8, 5), sharex=True, sharey=True)
            fig.tight_layout(pad=5)
            fmt = '%1.0f'

            fig.suptitle("Removing groundpoints using a threshold value for Z", fontsize="x-large", y=1.01)
            im1=ax1.scatter(xyz_plot_1[:,w], xyz_plot_1[:,l], c=xyz_plot_1[:,h], s=1)
            clb1=fig.colorbar(im1, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb1.set_label(str(label_z)+' (m)')
            ax1.grid()
            ax1.set_yticks(np.arange(round(h_min), round(h_max)+1, 1))
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
```

```
            ax1.set_title('Top view '+str(epoch_1[nmbr]))
            ax1.set_xlabel(str(label_x)+' (m)')
            ax1.set_ylabel(str(label_y)+' (m)')
            if Dir[nmbr] == True:
                ax1.invert_xaxis()

            im2=ax2.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c=xyz_plot_1_ng[:,h], s=1)
            clb2=fig.colorbar(im2, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb2.set_label(str(label_z)+' (m)')
            ax2.grid()
            ax2.set_yticks(np.arange(round(l_min), round(l_max)+1, 2))
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.ticklabel_format(useOffset=False)
            ax2.set_aspect('equal')
            ax2.set_title(str(epoch_1[nmbr])+' Ground removed')
            ax2.set_xlabel(str(label_x)+' (m)')
            plt.show()
```

```
[ ]: # Table with amount of points removed after applying threshold value for groundpoints
data_rounded = np.append(np.array(epoch_data,dtype=int).reshape(len(epoch_data),1),
                         np.array(data_ground), axis=1)

if filename_2 != None:
    #define header names
    col_names=pd.DataFrame([["","Year"], ["Datapoints", "Epoch 1"], ["Datapoints", "Epoch 2"], ["","Threshold␣
 ↪[m]"], ["Points removed", "Epoch 1 [%]"], ["Points removed", "Epoch 2 [%]"]],   columns=["","Building"])
    columns = pd.MultiIndex.from_frame(col_names)
    index = name_street

    #display table
    df = (pd.DataFrame(data_rounded, columns=columns, index=index))
    df["","Year"] = df["","Year"].astype('int')
    df["Datapoints", "Epoch 1"] = df["Datapoints", "Epoch 1"].astype('int')
    df["Datapoints", "Epoch 2"] = df["Datapoints", "Epoch 2"].astype('int')
    display(df)
else:
    #define header names
    col_names=pd.DataFrame([["","Year"], ["Datapoints", "Epoch 1"], ["","Threshold [m]"], ["Points removed",␣
 ↪"Epoch 1 [%]"]],   columns=["","Building"])
    columns = pd.MultiIndex.from_frame(col_names)
    index = name_street

    #display table
    df = (pd.DataFrame(data_rounded, columns=columns, index=index))
    df["","Year"] = df["","Year"].astype('int')
    df["Datapoints", "Epoch 1"] = df["Datapoints", "Epoch 1"].astype('int')
    display(df)
```

## 1.3. Segmentation

Segmentation is performed to remove points from the data. The removed points are expected to be a certain distance away from the plane. The resulting dataset is a set of points which are considered to lie on the facade of the building.

Insert the threshold value for segmentation in the cell below

```
[ ]: t = 0.04

set_background('wheat')
```

```
[ ]: # Segment the data and plot result
# ply files to store the data
filename_1_noground = 'epoch1.ply'
filename_2_noground = None
if filename_2 != None:
    filename_2_noground = 'epoch2.ply'
```

```
segment_output_1, segment_output_2, segment_result = [], [], []

for nmbr in range(len(filename_1)):
    xyz_plot_1_ng = np.array(XYZ_house_1_ng, dtype=object)[nmbr]
    col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
    if filename_2 != None:
        xyz_plot_2_ng = np.array(XYZ_house_2_ng, dtype=object)[nmbr]
        col_plot_2_ng = np.array(col_house_2_ng, dtype=object)[nmbr]

    w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
    if (w==0 and l==1 and h==2):
        label_x = 'w' #'X'
        label_y = 'l' #'Y'
        label_z = 'h' #'Z'
    if (w==1 and l==0 and h==2):
        label_x = 'l'#'Y'
        label_y = 'w'#'X'
        label_z = 'h'#'Z'
    # Determine min width, max width, min height, max height
    if filename_2 != None:
        cells_input = [min([min(xyz_plot_1_ng[:,w]), min(xyz_plot_2_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w]), max(xyz_plot_2_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h]), min(xyz_plot_2_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h]), max(xyz_plot_2_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l]), min(xyz_plot_2_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l]), max(xyz_plot_2_ng[:,l])])]
    else:
        cells_input = [min([min(xyz_plot_1_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l])])]
    w_min, w_max, h_min, h_max, l_min, l_max = cells_input

    if filename_2 != None:
        segment_output_1.append([in_segment(xyz_plot_1_ng, filename_1_noground, t)])
        segment_output_2.append([in_segment(xyz_plot_2_ng, filename_2_noground, t)])

        # Results segment function from open3d  (ground points removed before inserting points in the function)
        in_1_ng,plane_1_ng,pW_1_ng,pH_1_ng,pL_1_ng,idx_1_ng = np.array(segment_output_1, dtype=object)[nmbr][0]
        in_2_ng,plane_2_ng,pW_2_ng,pH_2_ng,pL_2_ng,idx_2_ng = np.array(segment_output_2, dtype=object)[nmbr][0]
        col_in_1_ng = col_plot_1_ng[idx_1_ng]
        col_in_2_ng = col_plot_2_ng[idx_2_ng]
        per1 = round((len(xyz_plot_1_ng)-len(in_1_ng))/len(xyz_plot_1_ng)*100,1)
        per2 = round((len(xyz_plot_2_ng)-len(in_2_ng))/len(xyz_plot_2_ng)*100,1)
        segment_result.append([len(xyz_plot_1_ng), len(xyz_plot_2_ng), t, per1, per2])
    else:
        segment_output_1.append([in_segment(xyz_plot_1_ng, filename_1_noground, t)])
        in_1_ng, plane_1_ng, pW_1_ng, pH_1_ng, pL_1_ng, idx_1_ng = np.array(segment_output_1,␣
↪dtype=object)[nmbr][0]
        col_in_1_ng = col_plot_1_ng[idx_1_ng]
        per1 = round((len(xyz_plot_1_ng)-len(in_1_ng))/len(xyz_plot_1_ng)*100,1)
        segment_result.append([len(xyz_plot_1_ng), t, per1])

    if plot_segment==True:
        if filename_2 != None:
            fig,(((ax1,ax2,ax3,ax4),(ax5,ax6,ax7,ax8)))=plt.subplots(2,4, figsize=(16,8),sharex='all',␣
↪sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'

            fig.suptitle("Groundpoints are removed before segmentation. Threshold = " + str(t)+" m",␣
↪fontsize="x-large", y=1.05)
```

```
            im1=ax1.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,h], c=col_plot_1_ng, s=1)
            ax1.ticklabel_format(useOffset=False)
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.set_aspect('equal')
            ax1.set_title(str(epoch_1[nmbr]))
            ax1.set_xlabel('W (m)')
            ax1.set_ylabel('H (m)')
            if Dir[nmbr] == True:
                ax1.invert_xaxis()

            im2=ax2.scatter(in_1_ng[:,w], in_1_ng[:,h], s=1, c=col_in_1_ng)
            ax2.ticklabel_format(useOffset=False)
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.set_aspect('equal')
            ax2.set_title(str(epoch_1[nmbr])+' after segmentation')
            ax2.set_xlabel('W (m)')

            im3=ax3.scatter(xyz_plot_2_ng[:,w], xyz_plot_2_ng[:,h], c=col_plot_2_ng, s=1)
            ax3.ticklabel_format(useOffset=False)
            ax3.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax3.set_aspect('equal')
            ax3.set_title(str(epoch_2[nmbr]))
            ax3.set_xlabel('W (m)')

            im4=ax4.scatter(in_2_ng[:,w], in_2_ng[:,h], s=1, c=col_in_2_ng)
            ax4.ticklabel_format(useOffset=False)
            ax4.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax4.set_aspect('equal')
            ax4.set_title(str(epoch_2[nmbr])+' after segmentation')
            ax4.set_xlabel('W (m)')

            im5=ax5.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c=col_plot_1_ng, s=1)
            ax5.ticklabel_format(useOffset=False)
            ax5.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax5.set_yticks(np.arange(round(l_min), round(l_max)+1, 5))
            ax5.set_aspect('equal')
            ax5.set_title(str(epoch_1[nmbr]))
            ax5.set_xlabel('W (m)')
            ax5.set_ylabel('L (m)')

            im6=ax6.scatter(in_1_ng[:,w], in_1_ng[:,l], s=1, c=col_in_1_ng)
            ax6.ticklabel_format(useOffset=False)
            ax6.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax6.set_aspect('equal')
            ax6.set_title(str(epoch_1[nmbr])+' after segmentation')
            ax6.set_xlabel('W (m)')

            im7=ax7.scatter(xyz_plot_2_ng[:,w], xyz_plot_2_ng[:,l], c=col_plot_2_ng, s=1)
            ax7.ticklabel_format(useOffset=False)
            ax7.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax7.set_aspect('equal')
            ax7.set_title(str(epoch_2[nmbr]))
            ax7.set_xlabel('W (m)')

            im8=ax8.scatter(in_2_ng[:,w], in_2_ng[:,l], s=1, c=col_in_2_ng)
            ax8.ticklabel_format(useOffset=False)
            ax8.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax8.set_aspect('equal')
            ax8.set_title(str(epoch_2[nmbr])+' after segmentation')
            ax8.set_xlabel('W (m)')
            plt.show()
        else:
            fig, (((ax1, ax2),(ax5, ax6))) = plt.subplots(2,2, figsize=(8,8),sharex='all',sharey='row')
            fig.tight_layout(pad=3)
            fmt = '%1.0f'
            fig.suptitle("Groundpoints are removed before segmentation. Threshold = " + str(t)+" m",␣
↪fontsize="x-large", y=1.05)
            im1=ax1.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,h], c=col_plot_1_ng, s=1)
            ax1.ticklabel_format(useOffset=False)
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.set_aspect('equal')
            ax1.set_title(str(epoch_2[nmbr]))
            ax1.set_xlabel('W (m)')
            ax1.set_ylabel('H (m)')
            if Dir[nmbr] == True:
                ax1.invert_xaxis()

            im2=ax2.scatter(in_1_ng[:,w], in_1_ng[:,h], s=1, c=col_in_1_ng)
            ax2.ticklabel_format(useOffset=False)
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.set_aspect('equal')
            ax2.set_title(str(epoch_2[nmbr])+' after segmentation')
            ax2.set_xlabel('W (m)')

            im5=ax5.scatter(xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c=col_plot_1_ng, s=1)
            ax5.ticklabel_format(useOffset=False)
            ax5.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax5.set_yticks(np.arange(round(l_min), round(l_max)+1, 5))
            ax5.set_aspect('equal')
            ax5.set_title(str(epoch_2[nmbr]))
            ax5.set_xlabel('W (m)')
            ax5.set_ylabel('L (m)')

            im6=ax6.scatter(in_1_ng[:,w], in_1_ng[:,l], s=1, c=col_in_1_ng)
            ax6.ticklabel_format(useOffset=False)
            ax6.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax6.set_aspect('equal')
            ax6.set_title(str(epoch_2[nmbr])+' after segmentation')
            ax6.set_xlabel('W (m)')
            plt.show()
```

[ ]:
```
# Table with amount of points removed due to segmentation
data_rounded = np.append(np.array(epoch_data,dtype=int).reshape(len(epoch_data),1),
                         np.array(segment_result), axis=1)
if filename_2 != None:
    #define header names
    col_names=pd.DataFrame([["","Year"], ["Datapoints", "Epoch 1"], ["Datapoints", "Epoch 2"], ["","Threshold␣
↪[m]"], ["Points removed", "Epoch 1 [%]"], ["Points removed", "Epoch 2 [%]"]],   columns=["","Building"])
    columns = pd.MultiIndex.from_frame(col_names)
    index = name_street

    #display table
    df = (pd.DataFrame(data_rounded, columns=columns, index=index))
    df["","Year"] = df["","Year"].astype('int')
    df["Datapoints", "Epoch 1"] = df["Datapoints", "Epoch 1"].astype('int')
    df["Datapoints", "Epoch 2"] = df["Datapoints", "Epoch 2"].astype('int')
    display(df)
else:
    #define header names
    col_names=pd.DataFrame([["","Year"], ["Datapoints", "Epoch 1"], ["","Threshold [m]"], ["Points removed",␣
↪"Epoch 1 [%]"]],   columns=["","Building"])
    columns = pd.MultiIndex.from_frame(col_names)
    index = name_street

    #display table
    df = (pd.DataFrame(data_rounded, columns=columns, index=index))
    df["","Year"] = df["","Year"].astype('int')
    df["Datapoints", "Epoch 1"] = df["Datapoints", "Epoch 1"].astype('int')
    display(df)
```

## Rotate datapoints

```
[ ]: # Fit plane and compute the torsion and tilt angles
data_notors_wrt1_1, data_notors_wrt1_2 = [], []
trans_angle = []

for nmbr in range(len(filename_1)):
    col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
    in_1_ng, plane_1_ng, pW_1_ng, pH_1_ng, pL_1_ng, idx_1_ng = np.array(segment_output_1, dtype=object)[nmbr][0]
    xyz_plot_1_ng = np.array(in_1_ng, dtype=object)

    if filename_2 != None:
        col_plot_2_ng = np.array(col_house_2_ng, dtype=object)[nmbr]
        in_2_ng,plane_2_ng,pW_2_ng,pH_2_ng,pL_2_ng,idx_2_ng = np.array(segment_output_2, dtype=object)[nmbr][0]
        xyz_plot_2_ng = np.array(in_2_ng, dtype=object)
    w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
    if (w==0 and l==1 and h==2):
        label_x = 'w' #'X'
        label_y = 'l' #'Y'
        label_z = 'h' #'Z'
    if (w==1 and l==0 and h==2):
        label_x = 'l'#'Y'
        label_y = 'w'#'X'
        label_z = 'h'#'Z'
    if filename_2 != None:
        cells_input = [min([min(xyz_plot_1_ng[:,w]), min(xyz_plot_2_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w]), max(xyz_plot_2_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h]), min(xyz_plot_2_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h]), max(xyz_plot_2_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l]), min(xyz_plot_2_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l]), max(xyz_plot_2_ng[:,l])])]
    else:
        cells_input = [min([min(xyz_plot_1_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l])])]
    w_min, w_max, h_min, h_max, l_min, l_max = cells_input

    if filename_2 != None:
        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        whl_2 = np.array(xyz_to_whl(xyz_plot_2_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        r_col_2 = col_plot_2_ng[idx_2_ng]
        thet_trans1 = transform_angle(plane_1_ng, plane_2_ng)[0]
        no_trans_1, trans_matrix_1 = tors_transform(xyz_plot_1_ng, np.radians(thet_trans1))
        transformed_2 = np.array([xyz_plot_2_ng[:,0], xyz_plot_2_ng[:,1],
                                  xyz_plot_2_ng[:,2], np.ones(len(xyz_plot_2_ng))]).T @ trans_matrix_1
        data_notors_wrt1_1.append(no_trans_1)
        data_notors_wrt1_2.append(transformed_2)
        trans_angle.append(round(thet_trans1,1))
    else:
        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        thet_trans1 = transform_angle(plane_1_ng, plane_1_ng)[0]
        no_trans_1, trans_matrix_1 = tors_transform(xyz_plot_1_ng, np.radians(thet_trans1))
        data_notors_wrt1_1.append(no_trans_1)
        trans_angle.append(round(thet_trans1,1))

    if plot_torsion==True:
        if filename_2 != None:
            fig, (((ax3, ax4))) = plt.subplots(1,2, figsize=(8,4), sharey='row')#,sharex='all')#,sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(      xyz_plot_1_ng[:,0],   xyz_plot_1_ng[:,1], c='r', label='Original')
            im3=ax3.scatter(no_trans_1[:,0], no_trans_1[:,1], c='g', label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_trans1,1))+' degrees')
            ax3.set_xlabel('x (m)')
            ax3.set_ylabel('y (m)')
            ax3.legend(loc=0)

            ax4.scatter(      xyz_plot_2_ng[:,0],   xyz_plot_2_ng[:,1], c='r', label='Original')
            im4=ax4.scatter( transformed_2[:,0],  transformed_2[:,1], c='g', label='Rotated')
            ax4.ticklabel_format(useOffset=False)
            ax4.set_aspect('equal')
            ax4.set_title('Rotate torsion: '+str(epoch_2[nmbr])+'\n With '+str(round(thet_trans1,1))+' degrees')
            ax4.set_xlabel('x (m)')
            ax4.legend(loc=0)
            plt.show()
        else:
            fig, (((ax3))) = plt.subplots(1,1, figsize=(4,4), sharey='row')#,sharex='all')#,sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(      xyz_plot_1_ng[:,0],   xyz_plot_1_ng[:,1], c='r', label='Original')
            im3=ax3.scatter(no_trans_1[:,0], no_trans_1[:,1], c='g', label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_trans1,1))+' degrees')
            ax3.set_xlabel('x (m)')
            ax3.set_ylabel('y (m)')
            ax3.legend(loc=0)
            plt.show()
```

```
[ ]: # Table with angles of rotation to transform from original to building system
data_rounded = np.append(np.array(epoch_data,dtype=int).reshape(len(epoch_data),1),
                         np.array(trans_angle).reshape(len(trans_angle),1), axis=1)
#define header names
col_names=pd.DataFrame([["Year"], ["Angle [degrees]"]],   columns=["Building"])
columns = pd.MultiIndex.from_frame(col_names)
index = name_street

#display table
df = (pd.DataFrame(data_rounded, columns=columns, index=index))
df["Year"] = df["Year"].astype('int')
display(df)
```

## Fit plane and compute torsion + tilt angles

```
[ ]: # Data
cellsize_ras = 1    # 0.5 = 0.5 m = 50 cm,    1 = 1 m

# Method
cA = 'PCA'      # cA: the algortihm that is chosen
                # 'TLS' for homogeneous least squares,   'PCA' for pca

set_background('wheat')
```

```
[ ]: # Fit plane and compute the torsion and tilt angles
angle_tors_list, angle_tilt_list = [], []
data_notors_1, data_notors_2 = [], []
data_notilt_1, data_notilt_2 = [], []

for nmbr in range(len(filename_1)):
    col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
    in_1_ng, plane_1_ng, pW_1_ng, pH_1_ng, pL_1_ng, idx_1_ng = np.array(segment_output_1, dtype=object)[nmbr][0]
    xyz_plot_1_ng = data_notors_wrt1_1[nmbr]
    if filename_2 != None:
        col_plot_2_ng = np.array(col_house_2_ng, dtype=object)[nmbr]
        in_2_ng,plane_2_ng,pW_2_ng,pH_2_ng,pL_2_ng,idx_2_ng = np.array(segment_output_2, dtype=object)[nmbr][0]
        xyz_plot_2_ng = data_notors_wrt1_2[nmbr]

    w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
    if (w==0 and l==1 and h==2):
        label_x = 'w' #'X'
        label_y = 'l' #'Y'
        label_z = 'h' #'Z'
    if (w==1 and l==0 and h==2):
        label_x = 'l'#'Y'
        label_y = 'w'#'X'
        label_z = 'h'#'Z'
    # Determine min width, max width, min height, max height
    if filename_2 != None:
        cells_input = [min([min(xyz_plot_1_ng[:,w]), min(xyz_plot_2_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w]), max(xyz_plot_2_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h]), min(xyz_plot_2_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h]), max(xyz_plot_2_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l]), min(xyz_plot_2_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l]), max(xyz_plot_2_ng[:,l])])]
    else:
        cells_input = [min([min(xyz_plot_1_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l])])]
    w_min, w_max, h_min, h_max, l_min, l_max = cells_input

    if filename_2 != None:
        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        whl_2 = np.array(xyz_to_whl(xyz_plot_2_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        r_col_2 = col_plot_2_ng[idx_2_ng]

        if cA=='TLS':
            ABCD_alg_a_r_1 = tls_to_plane(whl_1)[3]
            eigvec_1 = tls_to_plane(whl_1)[6]
            ABCD_alg_a_r_2 = tls_to_plane(whl_2)[3]
            eigvec_2 = tls_to_plane(whl_2)[6]
            plane_W_1, plane_H_1, plane_L_1 = tls_to_plane(whl_1)[:3]
            plane_W_2, plane_H_2, plane_L_2 = tls_to_plane(whl_2)[:3]
        if cA=='PCA':
            ABCD_alg_a_r_1 = pca_to_plane(whl_1)[3]
            eigvec_1 = pca_to_plane(whl_1)[6]
            ABCD_alg_a_r_2 = pca_to_plane(whl_2)[3]
            eigvec_2 = pca_to_plane(whl_2)[6]
            plane_W_1, plane_H_1, plane_L_1 = pca_to_plane(whl_1)[:3]
            plane_W_2, plane_H_2, plane_L_2 = pca_to_plane(whl_2)[:3]
        thet_tors1, thet_tors2, thet_tors12 = tors_angle(ABCD_alg_a_r_1, ABCD_alg_a_r_2) #eigvec_1, eigvec_2)
        thet_tilt1, thet_tilt2, thet_tilt12 = tilt_angle(ABCD_alg_a_r_1, ABCD_alg_a_r_2) #eigvec_1, eigvec_2)
        angle_tors_list.append([nmbr, thet_tors1, thet_tors2, thet_tors12])
        angle_tilt_list.append([nmbr, thet_tilt1, thet_tilt2, thet_tilt12])
        no_tors_1, tors_matrix_1 = tors_transform(xyz_plot_1_ng, np.radians(thet_tors1))
        no_tors_2 = tors_transform(xyz_plot_2_ng[:,:3], np.radians(thet_tors2))[0]
        no_tilt_1 = tilt_transform(no_tors_1, np.radians(thet_tilt1))[0]
        no_tilt_2 = tilt_transform(no_tors_2, np.radians(thet_tilt2))[0]
        data_notors_1.append(no_tors_1)
        data_notors_2.append(no_tors_2)
        data_notilt_1.append(no_tilt_1)
        data_notilt_2.append(no_tilt_2)
        torsed_2 = np.array([xyz_plot_2_ng[:,0],xyz_plot_2_ng[:,1],
                             xyz_plot_2_ng[:,2],np.ones(len(xyz_plot_2_ng))]).T @ tors_matrix_1
    else:
        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        if cA=='TLS':
            ABCD_alg_a_r_1 = tls_to_plane(whl_1)[3]
            eigvec_1 = tls_to_plane(whl_1)[6]
            plane_W_1, plane_H_1, plane_L_1 = tls_to_plane(whl_1)[:3]
        if cA=='PCA':
            ABCD_alg_a_r_1 = pca_to_plane(whl_1)[3]
            eigvec_1 = pca_to_plane(whl_1)[6]
            plane_W_1, plane_H_1, plane_L_1 = pca_to_plane(whl_1)[:3]
        thet_tors1 = tors_angle(ABCD_alg_a_r_1, ABCD_alg_a_r_1)[0]
        thet_tilt1 = tilt_angle(ABCD_alg_a_r_1, ABCD_alg_a_r_1)[0]
        angle_tors_list.append([nmbr, thet_tors1])
        angle_tilt_list.append([nmbr, thet_tilt1])
        no_tors_1, tors_matrix_1 = tors_transform(xyz_plot_1_ng, np.radians(thet_tors1))
        no_tilt_1 = tilt_transform(no_tors_1, np.radians(thet_tilt1))[0]
        data_notors_1.append(no_tors_1)
        data_notilt_1.append(no_tilt_1)
    if plot_plane==True:
        if filename_2 != None:
            fig, (((ax3, ax4))) = plt.subplots(1,2, figsize=(8,4), sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter( xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c='r',      s=1, label='Datapoints')
            im3=ax3.scatter(plane_W_1,    plane_L_1, c='orange', s=1, label='Plane')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-2, np.max(in_1_ng[:,l])+2)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tors1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)

            ax4.scatter( xyz_plot_2_ng[:,w], xyz_plot_2_ng[:,l], c='r',      s=1, label='Datapoints')
            im3=ax4.scatter(plane_W_2,    plane_L_2, c='orange', s=1, label='Plane')
            ax4.ticklabel_format(useOffset=False)
            ax4.set_aspect('equal')
            ax4.set_title('Rotate torsion: '+str(epoch_2[nmbr])+'\n With '+str(round(thet_tors2,1))+' degrees')
            ax4.set_xlabel('w (m)')
            ax4.legend(loc=1)
            plt.show()
        else:
            fig, (((ax3))) = plt.subplots(1,1, figsize=(4,4), sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter( xyz_plot_1_ng[:,w], xyz_plot_1_ng[:,l], c='r',      s=1, label='Datapoints')
            im3=ax3.scatter(plane_W_1,    plane_L_1, c='orange', s=1, label='Plane')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-2, np.max(in_1_ng[:,l])+2)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tors1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)
            plt.show()
    if plot_torsion==True:
        if filename_2 != None:
            fig, (((ax3, ax4))) = plt.subplots(1,2, figsize=(8,4), sharey='row')#,sharex='all')#,sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(      xyz_plot_1_ng[:,w],   xyz_plot_1_ng[:,l], c='r', label='Original')
            im3=ax3.scatter(no_tors_1[:,w], no_tors_1[:,l], c='g', label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-2, np.max(in_1_ng[:,l])+2)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tors1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)

            ax4.scatter(      xyz_plot_2_ng[:,w],   xyz_plot_2_ng[:,l], c='r', label='Original')
            im4=ax4.scatter(no_tors_2[:,w], no_tors_2[:,l], c='g', label='Rotated')
            ax4.ticklabel_format(useOffset=False)
            ax4.set_aspect('equal')
            ax4.set_title('Rotate torsion: '+str(epoch_2[nmbr])+'\n With '+str(round(thet_tors2,1))+' degrees')
            ax4.set_xlabel('w (m)')
            ax4.legend(loc=1)
            plt.show()
        else:
            fig, (((ax3))) = plt.subplots(1,1, figsize=(4,4), sharey='row')#,sharex='all')#,sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(      xyz_plot_1_ng[:,w],   xyz_plot_1_ng[:,l], c='r', label='Original')
            im3=ax3.scatter(no_tors_1[:,w], no_tors_1[:,l], c='g', label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-2, np.max(in_1_ng[:,l])+2)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate torsion: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tors1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)
            plt.show()
    if plot_tilt==True:
        if filename_2 != None:
            fig, (((ax3, ax4))) = plt.subplots(1,2, figsize=(8,4), sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(no_tors_1[:,w], no_tors_1[:,l], c='m', s=1, label='Previous')
            ax3.scatter(no_tilt_1[:,w], no_tilt_1[:,l], c='b', s=1, label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-1, np.max(in_1_ng[:,l])+1)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate tilt: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tilt1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)

            ax4.scatter(no_tors_2[:,w], no_tors_2[:,l], c='m', s=1, label='Previous')
            ax4.scatter(no_tilt_2[:,w], no_tilt_2[:,l], c='b', s=1, label='Rotated')
            ax4.ticklabel_format(useOffset=False)
            ax4.set_aspect('equal')
            ax4.set_title('Rotate tilt: '+str(epoch_2[nmbr])+'\n With '+str(round(thet_tilt2,1))+' degrees')
            ax4.set_xlabel('w (m)')
            ax4.legend(loc=1)
            plt.show()
        else:
            fig, (((ax3))) = plt.subplots(1,1, figsize=(4,4), sharey='row')
            fig.tight_layout(pad=4)
            fmt = '%1.0f'
            ax3.scatter(no_tors_1[:,w], no_tors_1[:,l], c='m', s=1, label='Previous')
            ax3.scatter(no_tilt_1[:,w], no_tilt_1[:,l], c='b', s=1, label='Rotated')
            ax3.ticklabel_format(useOffset=False)
            ax3.set_ylim(np.min(in_1_ng[:,l])-1, np.max(in_1_ng[:,l])+1)
            ax3.set_aspect('equal')
            ax3.set_title('Rotate tilt: '+str(epoch_1[nmbr])+'\n With '+str(round(thet_tilt1,1))+' degrees')
            ax3.set_xlabel('w (m)')
            ax3.set_ylabel('l (m)')
            ax3.legend(loc=1)
            plt.show()
```

```
[ ]: # display torsion and tilt angles [1 decimals]
     if filename_2 != None:
         angle_tors_arr = np.array(angle_tors_list)
         angle_tilt_arr = np.array(angle_tilt_list)
         data_angles = np.around(np.array([angle_tors_arr[:,1], angle_tors_arr[:,2], angle_tors_arr[:,3],
                                 angle_tilt_arr[:,1], angle_tilt_arr[:,2], angle_tilt_arr[:,3]]).T,  decimals=1)
         data_all = np.append(np.array(epoch_data,dtype=int).reshape(len(epoch_data),1),
                              data_angles, axis=1)

         #define header names
         col_names=pd.DataFrame([["Year",""], ["Torsion angle [degrees]", "Epoch 1"], ["Torsion angle [degrees]",␣
     ↪"Epoch 2"], ["Torsion angle [degrees]", "Epoch 1 vs 2"], ["Tilt angle [degrees]", "Epoch 1"], ["Tilt angle␣
     ↪[degrees]", "Epoch 2"], ["Tilt angle [degrees]", "Epoch 1 vs 2"] ],   columns=["Building", ""])
         columns = pd.MultiIndex.from_frame(col_names)
         index = name_street

         #display table
         df = (pd.DataFrame(data_all, columns=columns, index=index))
         df['Year'] = df['Year'].astype('int')
         display(df)
     else:
         angle_tors_arr = np.array(angle_tors_list)
         angle_tilt_arr = np.array(angle_tilt_list)
         data_angles = np.around(np.array([angle_tors_arr[:,1],
                                 angle_tilt_arr[:,1]]).T,
                                 decimals=1)
         data_all = np.append(np.array(epoch_data,dtype=int).reshape(len(epoch_data),1),
                              data_angles, axis=1)

         #define header names
         col_names=pd.DataFrame([["Year",""], ["Torsion angle [degrees]", "Epoch 1"], ["Tilt angle [degrees]", "Epoch␣
     ↪1"],], columns=["Building", ""])   columns = pd.MultiIndex.from_frame(col_names)
         index = name_street

         #display table
         df = (pd.DataFrame(data_all, columns=columns, index=index))
         df['Year'] = df['Year'].astype('int')
         display(df)
```

## 3.2. Raster with distances

Insert the cellsize of the raster in the cell below

```
[ ]: cellsize = 0.1   # m
     p_s = 2

     set_background('wheat')
```

```
[ ]: raster_rot_wrt_tors1 = []

     # Rotated wrt torsion angle of epoch 1
     for nmbr in range(len(filename_1)):
         col_plot_1_ng = np.array(col_house_1_ng, dtype=object)[nmbr]
         in_1_ng, plane_1_ng, pW_1_ng, pH_1_ng, pL_1_ng, idx_1_ng = np.array(segment_output_1, dtype=object)[nmbr][0]
         in_1_ng = data_notors_wrt1_1[nmbr]
         xyz_plot_1_ng = np.array(in_1_ng, dtype=object)

         w, l, h = wlh[nmbr,0], wlh[nmbr,1], wlh[nmbr,2]
         if (w==0 and l==1 and h==2):
             label_x = 'w' #'X'
             label_y = 'l' #'Y'
             label_z = 'h' #'Z'
         if (w==1 and l==0 and h==2):
             label_x = 'l'#'Y'
```

```
        label_y = 'w'#'X'
        label_z = 'h'#'Z'

    if filename_2 != None:
        col_plot_2_ng = np.array(col_house_2_ng, dtype=object)[nmbr]
        in_2_ng,plane_2_ng,pW_2_ng,pH_2_ng,pL_2_ng,idx_2_ng = np.array(segment_output_2, dtype=object)[nmbr][0]
        in_2_ng = data_notors_wrt1_2[nmbr]
        xyz_plot_2_ng = np.array(in_2_ng, dtype=object)
        cells_input = [min([min(xyz_plot_1_ng[:,w]), min(xyz_plot_2_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w]), max(xyz_plot_2_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h]), min(xyz_plot_2_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h]), max(xyz_plot_2_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l]), min(xyz_plot_2_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l]), max(xyz_plot_2_ng[:,l])])]
        w_min, w_max, h_min, h_max, l_min, l_max = cells_input

        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        whl_2 = np.array(xyz_to_whl(xyz_plot_2_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        r_col_2 = col_plot_2_ng[idx_2_ng]

        raster_1 = rasterpoints_WH(whl_1, cellsize)[0]
        raster_2 = rasterpoints_WH(whl_2, cellsize)[0]
        raster_1 = (assign_points(raster_1, r_col_1, xyz_plot_1_ng, 10, cellsize))
        raster_2 = (assign_points(raster_2, r_col_2, xyz_plot_2_ng, 10, cellsize))

        y_delta_1 = 100*(raster_1[:,5]-raster_2[:,5])
        y_d_min = --np.max([abs(np.nanmin(y_delta_1)), abs(np.nanmax(y_delta_1))])
        y_d_max = -y_d_min
        y_min_1 = np.max([np.nanmin(raster_1[:,5]), np.nanmin(raster_2[:,5])])
        y_max_1 = np.min([np.nanmax(raster_1[:,5]), np.nanmax(raster_2[:,5])])

        raster_building =  np.hstack((raster_1[:,:2], y_delta_1.reshape(len(y_delta_1),1)))
        raster_rot_wrt_tors1.append([filename_1[nmbr], home_numbers[nmbr], raster_building])
    else:
        cells_input = [min([min(xyz_plot_1_ng[:,w])]),
                       max([max(xyz_plot_1_ng[:,w])]),
                       min([min(xyz_plot_1_ng[:,h])]),
                       max([max(xyz_plot_1_ng[:,h])]),
                       min([min(xyz_plot_1_ng[:,l])]),
                       max([max(xyz_plot_1_ng[:,l])])]
        w_min, w_max, h_min, h_max, l_min, l_max = cells_input

        whl_1 = np.array(xyz_to_whl(xyz_plot_1_ng), dtype='float64')
        r_col_1 = col_plot_1_ng[idx_1_ng]
        raster_1 = rasterpoints_WH(whl_1, cellsize)[0]
        raster_1 = (assign_points(raster_1, r_col_1, xyz_plot_1_ng, 10, cellsize))
        y_d_min = -np.max([abs(np.nanmin(raster_1[:,5])), abs(np.nanmax(raster_1[:,5]))])
        y_d_max = -y_d_min
        y_min_1 = -10 #np.max([np.nanmin(raster_1[:,5])])
        y_max_1 = 10  #np.min([np.nanmax(raster_1[:,5])])
        raster_building =  np.hstack((raster_1[:,:2], (raster_1[:,5]).reshape(len(raster_1),1)))
        raster_rot_wrt_tors1.append([filename_1[nmbr], home_numbers[nmbr], raster_building])

    plot = True
    if plot==True:
        if filename_2 != None:
            fig, ((ax1, ax2, ax3)) = plt.subplots(1, 3, figsize=(12, 4), sharex=True, sharey=True)
            fig.tight_layout(pad=1)
            fmt = '%1.0f'
            im1=ax1.scatter(raster_1[:,0], raster_1[:,1], c=raster_1[:,5], s=p_s, cmap='magma', vmin=y_min_1,␣
↪vmax=y_max_1)
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel('Width (m)')
            ax1.set_ylabel('Height (m)')
            ax1.set_title('Raster of image 2021')
            clb1 = fig.colorbar(im1, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb1.set_label('Length (m)')

            im2=ax2.scatter(raster_2[:,0], raster_2[:,1], c=raster_2[:,5], s=p_s, cmap='magma', vmin=y_min_1,␣
 ↪vmax=y_max_1)
            ax2.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax2.ticklabel_format(useOffset=False)
            ax2.set_aspect('equal')
            ax2.set_xlabel('Width (m)')
            ax2.set_title('Raster of image 2022')
            clb2 = fig.colorbar(im2, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb2.set_label('Length (m)')

            im3=ax3.scatter(raster_1[:,0], raster_1[:,1], c=y_delta_1, s=p_s, cmap='jet', vmin=y_d_min,␣
 ↪vmax=y_d_max)
            ax3.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax3.ticklabel_format(useOffset=False)
            ax3.set_aspect('equal')
            ax3.set_xlabel('Width (m)')
            ax3.set_title('Raster with difference between 2021 and 2022')
            clb3 = fig.colorbar(im3, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
            clb3.set_label('Difference in length (cm)');
        else:
            fig, ((ax1)) = plt.subplots(1, 1, figsize=(8, 4), sharex=True, sharey=True)
            fig.tight_layout(pad=1)
            fmt = '%1.0f'
            im1=ax1.scatter(raster_1[:,0], raster_1[:,1], c=(raster_1[:,5]-np.nanmean(raster_1[:,5]))*100,
                                                             s=p_s, cmap='jet', vmin=y_min_1, vmax=y_max_1)
            ax1.set_xticks(np.arange(round(w_min), round(w_max)+1, 5))
            ax1.ticklabel_format(useOffset=False)
            ax1.set_aspect('equal')
            ax1.set_xlabel('W (m)', fontsize=12)
            ax1.set_ylabel('H (m)', fontsize=12)
            ax1.tick_params(axis='x', labelsize=12)
            ax1.tick_params(axis='y', labelsize=12)
            ax1.set_title('Raster of image 2023. ' +str(name_dataset[nmbr]), fontsize=14)
            clb1 = fig.colorbar(im1, orientation='horizontal', format=fmt, fraction=0.03, pad=0.16)
            clb1.set_label('Length (cm) wrt mean length ('+str(round(np.nanmean(raster_1[:,5]),1))+' m)',
                           fontsize=11)
            clb1.ax.tick_params(labelsize=12)
            if Dir[nmbr] == True:
                ax1.invert_xaxis()
            plt.show();
```

## Overview image (raster of whole street)

[ ]:
```
facade_str1_e1, facade_str1_e2, facade_str2_e1, facade_str2_e2 =[], [], [], []
ras_str1, ras_str2 = [], []

for i in range(len(np.array(raster_rot_wrt_tors1, dtype=object)[:,1])):
    if ((np.array(raster_rot_wrt_tors1, dtype=object))[i,1]%2) == 0:
        ras_str1.append([np.array(raster_rot_wrt_tors1, dtype=object)[i,1], np.array(raster_rot_wrt_tors1,␣
 ↪dtype=object)[i,2]])
        facade_str1_e1.append([np.array(facade_e1, dtype=object)[i,1],
                               np.array(facade_e1, dtype=object)[i,2],
                               np.array(facade_e1, dtype=object)[i,3]])
        if filename_2 != None:
            facade_str1_e2.append([np.array(facade_e2, dtype=object)[i,1],
                                   np.array(facade_e2, dtype=object)[i,2],
                                   np.array(facade_e2, dtype=object)[i,3]])
    else:
        ras_str2.append([np.array(raster_rot_wrt_tors1, dtype=object)[i,1], np.array(raster_rot_wrt_tors1,␣
↪dtype=object)[i,2]])
        facade_str2_e1.append([np.array(facade_e1, dtype=object)[i,1],
                               np.array(facade_e1, dtype=object)[i,2],
                               np.array(facade_e1, dtype=object)[i,3]])
        if filename_2 != None:
            facade_str2_e2.append([np.array(facade_e2, dtype=object)[i,1],
                                   np.array(facade_e2, dtype=object)[i,2],
                                   np.array(facade_e2, dtype=object)[i,3]])
```

[ ]:
```
# Buildings in color per street
# Buildings epoch 1
if len(facade_str1_e1) != 0:
    fig, axs = plt.subplots(1,1, figsize=(10,6))
    fig.tight_layout(pad=1)
    fmt = '%1.0f'
    for i in range(len(np.array(facade_str1_e1,dtype=object))):
        axs.scatter((facade_str1_e1[i][1])[:,w],(facade_str1_e1[i][1])[:,h],c=facade_str1_e1[i][2],s=p_s)
        axs.grid()
        axs.ticklabel_format(useOffset=False)
        axs.set_aspect('equal')
        axs.set_xlabel('Width (m)')
        axs.set_ylabel('Height (m)')
        axs.set_title('Overview of the whole street '+str(epoch1_year)+' (even side)')
        axs.invert_xaxis()

# Buildings epoch 2
if len(facade_str2_e1) != 0:
    fig, axs = plt.subplots(1,1, figsize=(10,6))
    fig.tight_layout(pad=1)
    fmt = '%1.0f'
    for i in range(len(np.array(facade_str2_e1,dtype=object))):
        axs.scatter((facade_str2_e1[i][1])[:,w],(facade_str2_e1[i][1])[:,h],c=facade_str2_e1[i][2],s=p_s)
        axs.grid()
        axs.ticklabel_format(useOffset=False)
        axs.set_aspect('equal')
        axs.set_xlabel('Width (m)')
        axs.set_ylabel('Height (m)')
        axs.set_title('Overview of the whole street '+str(epoch1_year)+' (odd side)')
if filename_2 != None:
    if len(facade_str1_e2) != 0:
        # Building image 2022
        fig, axs = plt.subplots(1,1, figsize=(10,6))
        fig.tight_layout(pad=1)
        fmt = '%1.0f'
        for i in range(len(np.array(facade_str1_e2,dtype=object))):
                axs.scatter((facade_str1_e2[i][1])[:,w],(facade_str1_e2[i][1])[:,h],c=facade_str1_e2[i][2],s=p_s)
                axs.grid()
                axs.ticklabel_format(useOffset=False)
                axs.set_aspect('equal')
                axs.set_xlabel('Width (m)')
                axs.set_ylabel('Height (m)')
                axs.set_title('Overview of the whole street '+str(epoch2_year)+' (even side)')
                axs.invert_xaxis()
    if len(facade_str2_e2) != 0:
        fig, axs = plt.subplots(1,1, figsize=(10,6))
        fig.tight_layout(pad=1)
        fmt = '%1.0f'
        for i in range(len(np.array(facade_str2_e2,dtype=object))):
            axs.scatter((facade_str2_e2[i][1])[:,w],(facade_str2_e2[i][1])[:,h],c=facade_str2_e2[i][2],s=p_s)
            axs.grid()
            axs.ticklabel_format(useOffset=False)
            axs.set_aspect('equal')
            axs.set_xlabel('Width (m)')
            axs.set_ylabel('Height (m)')
            axs.set_title('Overview of the whole street '+str(epoch2_year)+' (odd side)')
```

[ ]:
```
# L in color
# Buildings epoch 1
if len(facade_str1_e1) != 0:
    fig, axs = plt.subplots(1,1, figsize=(10,6))
    fig.tight_layout(pad=1)
    fmt = '%1.0f'
    for i in range(len(np.array(facade_str1_e1,dtype=object))):
        im=axs.scatter((facade_str1_e1[i][1])[:,w],(facade_str1_e1[i][1])[:,h],c=(facade_str1_e1[i][1])[:
 ↪,l],s=p_s,cmap='jet', vmin=l_min-2, vmax=l_max+2)
        axs.grid()
        axs.ticklabel_format(useOffset=False)
        axs.set_aspect('equal')
        axs.set_xlabel('Width (m)')
        axs.set_ylabel('Height (m)')
        axs.set_title('Overview of the whole street '+str(epoch1_year)+' (even side)')
        axs.invert_xaxis()
    clb = fig.colorbar(im, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
    clb.set_label('Difference in length (cm)');

# Buildings epoch 2
if len(facade_str2_e1) != 0:
    fig, axs = plt.subplots(1,1, figsize=(10,6))
    fig.tight_layout(pad=1)
    fmt = '%1.0f'
    for i in range(len(np.array(facade_str2_e1,dtype=object))):
        im=axs.scatter((facade_str2_e1[i][1])[:,w],(facade_str2_e1[i][1])[:,h],c=(facade_str2_e1[i][1])[:
 ↪,l],s=p_s,cmap='jet', vmin=l_min-2, vmax=l_max+2)
        axs.grid()
        axs.ticklabel_format(useOffset=False)
        axs.set_aspect('equal')
        axs.set_xlabel('Width (m)')
        axs.set_ylabel('Height (m)')
        axs.set_title('Overview of the whole street '+str(epoch1_year)+' (odd side)')
    clb = fig.colorbar(im, orientation='horizontal', format=fmt, fraction=0.04, pad=0.2)
    clb.set_label('Difference in length (cm)');
```

[ ]:
```
# Raster images (difference between epochs OR distance wrt the mean length)
if filename_2 != None:
    if len(ras_str1) != 0:
        fig, axs = plt.subplots(1,1, figsize=(10,6))
        fig.tight_layout(pad=1)
        fmt = '%1.2f'
        for i in range(len(np.array(ras_str1,dtype=object))):
            im=axs.scatter((ras_str1[i][1])[:,0],(ras_str1[i][1])[:,1],c=(ras_str1[i][1])[:,2],s=p_s,cmap='jet',␣
 ↪vmin=-10,vmax=10)
            axs.set_aspect('equal')
            axs.ticklabel_format(useOffset=False)
            axs.set_ylabel('Height (m)')
            axs.set_xlabel('Width (m)')
            axs.set_title('Raster with difference between epoch 1 and 2 (even side)')
        clb = fig.colorbar(im, orientation='horizontal', format=fmt, fraction=0.04, pad=0.1)
        clb.set_label('Difference in length (cm)');
    if len(ras_str2) != 0:
        fig, axs = plt.subplots(1,1, figsize=(10,6))
        fig.tight_layout(pad=1)
        fmt = '%1.0f'
        for i in range(len(np.array(ras_str2,dtype=object))):
            im=axs.scatter((ras_str2[i][1])[:,0],(ras_str2[i][1])[:,1],c=(ras_str2[i][1])[:,2],s=p_s,cmap='jet',␣
 ↪vmin=-10,vmax=10)
            axs.set_aspect('equal')
            axs.ticklabel_format(useOffset=False)
            axs.set_ylabel('Height (m)')
            axs.set_xlabel('Width (m)')
            axs.set_title('Raster with difference between epoch 1 and 2 (odd side)')
        clb = fig.colorbar(im, orientation='horizontal', format=fmt, fraction=0.04, pad=0.1)
        clb.set_label('Difference in length (cm)');
else:
    fig, axs = plt.subplots(1,1, figsize=(10,6))
    fig.tight_layout(pad=1)
    fmt = '%1.2f'
    for i in range(len(np.array(ras_str1,dtype=object))):
        im=axs.scatter((ras_str1[i][1])[:,0],(ras_str1[i][1])[:,1],c=((ras_str1[i][1])[:,2]-np.
 ↪nanmean((ras_str1[i][1])[:,2]))*100, s=p_s,cmap='jet',vmin=-10, vmax=10)
        axs.set_aspect('equal')
        axs.ticklabel_format(useOffset=False)
        axs.set_ylabel('Height (m)')
        axs.set_xlabel('Width (m)')
        axs.set_title('Raster of 2023 of the Amstel in Amsterdam')
        axs.invert_xaxis()
    clb = fig.colorbar(im, orientation='horizontal', format=fmt, fraction=0.04, pad=0.1)
    clb.set_label('Length (cm) wrt a length of 486680 m');
```

**End**