Czech Technical University in Prague
Faculty of Electrical Engineering

Diploma thesis

Object Volume Calculation from Large Noisy Point-Clouds

Bc. Zuzana Tůmová

Supervisor: Ing. Milan Rollo, Ph.D.

Study Programme: Cybernetics and Robotics

Branch of Study: Cybernetics and Robotics

May 2018

Declaration

Acknowledgement

I would first like to thank my thesis advisor Ing. Milan Rollo, Ph.D., who supported my work and helped me to get the results of better quality. I also must express my very profound gratitude to my parents and family for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

Abstract

This diploma thesis is focused on methods of object volume calculation from large noisy point clouds. The work deals with filtering, creating triangular meshes, detecting ground plane and finally with volume calculation. The RMLS, RANSAC, Delaunay triangulation and signed volume of tetrahedrons calculation algorithms are described in detail. All these algorithms were implemented in Matlab environment and tested on simulated and real datasets. At the end of this thesis, the features of all algorithms and the accuracy of used volume calculation algorithm is shown.

Abstrakt

Tato diplomová práce se zabývá metodami výpočtu objemu objektů z rozsáhlých zašuměných mračen bodů. Práce je zaměřena na filtrování, tvorbu triangulačních sítí, detekci povrchu země a v závěru se zabývá výpočtem objemů. Jsou zde detailně popsány a využity algoritmy RMLS, RANSAC, Delaunay triangulace a výpočet orientovaných objemů tetrahedronů. Všechny tyto algoritmy byly implementovány v prostředí Matlab a otestovány na simulovaných i reálných datech. V závěru práce jsou ukázány vlastnosti všech algoritmů a přesnost algoritmu pro výpočet objemu.

vložit originální zadání !!!!!

x

Table of Contents

List of Figures

1 Introduction

In recent years, 3D modeling from point-cloud data has been in the sphere of interest. An increasing number of applications require the use of accurate 3D models. The example of popular applications includes virtual reality, autonomous mobile mapping, scanning of historical artifacts, gaming or 3D printing, as well as others.

A primary data source for various mapping applications have been point clouds from light detection and ranging (LiDAR) sensors. Processing large-scale geospatial data, especially point clouds, can be very computational and time demanding. They usually contain millions of points. Therefore it is crucial to implement fast and accurate processing algorithms. If applications handle unorganized point clouds, the complexity of all algorithms grows due to the scanning from multiple viewpoints and subsequent merging. The difficulty also increases with the amount of noise, outliers, and other inaccuracies.

Our application is composed of LiDAR scanning system mounted on terrestrial rover of small UAV (Unmanned Aerial Vehicle). Additional sensors include GNSS (Global Navigation Satellite System) receiver and IMU (Inertial Measurement Unit) with integrated accelerometers, gyroscopes, and magnetometers. The main purpose of this project is the mapping of agricultural areas such as forests or fields.

In this thesis, we come up with several algorithms for point-cloud data post-processing. As a programming tool, we chose MATLAB environment. The focus is on four main topics: filtering, triangulation, ground plane detection and most importantly object volume calculation. The filtering is dealing with noise and outliers removal, polynomial regression, and points quantity reduction. The triangulation part leads to creation of a 2.5D triangular mesh, which represents the model surface. These two sections together with ground plane detection are prerequisites for final volume calculation. The goal of this algorithm is to calculate the volume of any chosen 3D object defined by the closed triangular surface.

2 Point clouds

Point clouds are large data sets defined by points in a given coordinate system. In a 3D Cartesian coordinate system, a point is identified by three coordinates that, taken together, correlate to a precise point in space relative to the point of origin. The point cloud is a vector-based structure - each point has XYZ coordinates and other optional attributes that can represent time, intensity, reflectivity, color, flight line, etc. It is an accurate digital record that covers external surfaces of a sensed object [16].

Most common applications are using LiDAR laser scanners to collect geographic point cloud data. These data are mostly available in LAS (LiDAR Aerial Survey) or ASCII (XYZ) format. LAS is an industry standard file format defined by the American Society of Photogrammetry and Remote Sensing that includes a system of point classification. A processed LAS file may have points classified as bare earth, high or low vegetation, building, etc [2].

The critical factor in acquiring point cloud data is the visibility of scanned objects (regardless of the method of acquisition - LiDAR scans or photos). It is not possible to obtain points on the surfaces that are not visible directly from the measuring position. This means that to cover all objects we have to combine scans from multiple scanning positions. This process is known as point set registration or point matching, and its task is to find a spatial transformation that merges multiple data sets into a globally consistent model.

While point clouds can be directly rendered and inspected, they are often converted to polygon mesh or triangle mesh models, non-uniform rational basis spline (NURBS) surface models, or computer-aided design (CAD) models through a process commonly referred to as surface reconstruction. There are many techniques for converting a point cloud to a 3D surface like Delaunay triangulation, alpha shapes, ball pivoting, etc [7].

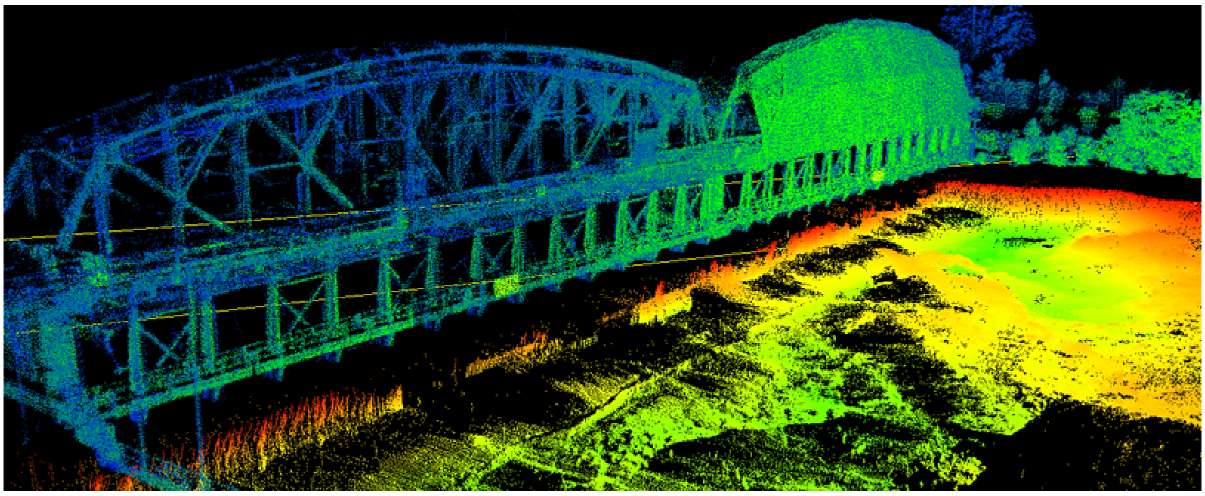

Figure 2.1: Example of point cloud from LiDAR data
Source: `www.h2hassociates.com`

3 Sensors

There are two major approaches for capturing of point clouds - either using LiDAR or photogrammetry. They are both valid methods that produce point clouds, but the resulting point cloud and the method of capturing have different characteristics. The LiDAR output takes far less time to capture, and it provides a clean and sharp point cloud that is easy for work. A LiDAR scanner is an active sensor, and therefore it is insensitive to the uniform surfaces (e.g., water, grassland, desert, etc.) and able to measure surfaces without significant structures. On the other hand, it misses the information about RGB color. The photogrammetric data collection and processing take slightly longer, and the point cloud requires extensive cleaning. Photogrammetric processing relies on identifiable features to be matched across the sequences of images. Because the uniform surfaces are missing the identifiable features, photogrammetric processing will fail in these areas, resulting in gaps or false results. Removing these misplaced points can be difficult and time-consuming [18].

The LiDAR sensor needs to be merged with other sensors to obtain desired point clouds. Mostly used additional sensors are inertial and positional sensors that provide required information about rotation and translation to align all scans into the coherent model.

3.1 LiDAR

Light detection and ranging (LiDAR) is an active optical sensor that transmits laser beams toward a target. It can dynamicly move through the specific survey routes or make the static scanns from different view points. The reflection of the laser beams from the target is detected and analyzed by receivers in the LiDAR sensor. These receivers record the precise time from when the laser pulse left the system to when it is returned to calculate the range distance between the sensor and the target [8].

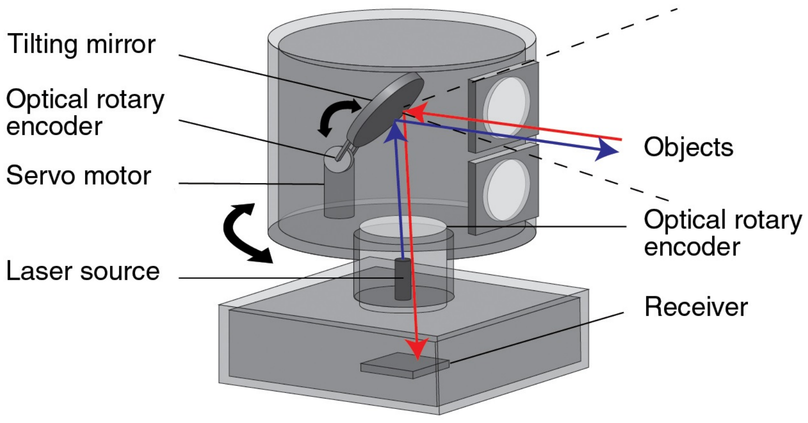

Figure 3.1: Diagram of the LiDAR optics and encoders
Source: `www.renishaw.com`

The LiDAR scanning system usually emits a single laser ray and uses an interior nodding mirror to distribute the laser ray covering a vertical field of view (different elevations). It uses a rotary encoder for closed-loop feedback control of the scan pattern. In most legacy designs, the rotary encoder is mounted on the mirror shaft (gimbal) alongside an electromagnetic motor capable of tilting the mirror clockwise and counter-clockwise about the tilt axis. The horizontal data acquisition (different azimuths) is the result of the rotation of the whole system around its base [6] (see Figure 3.1).

One emitted laser pulse can return as more reflected beams (see Figure 3.2). It is caused by different surfaces (e.g., trees, buildings, etc.). The first returned laser pulse is the most significant and intensive return, and it is associated with the highest feature in the landscape like a treetop or the top of a building. First returns represent about 70 % of all returned beams. The intermediate returns, in general, are used for vegetation structure, and the last return represents the terrain models. The last return is not necessarily the return reflected from the ground. For example, the laser beam can hit a thick branch of the tree and therefore doesn't reach down to the bottom to the ground [8].

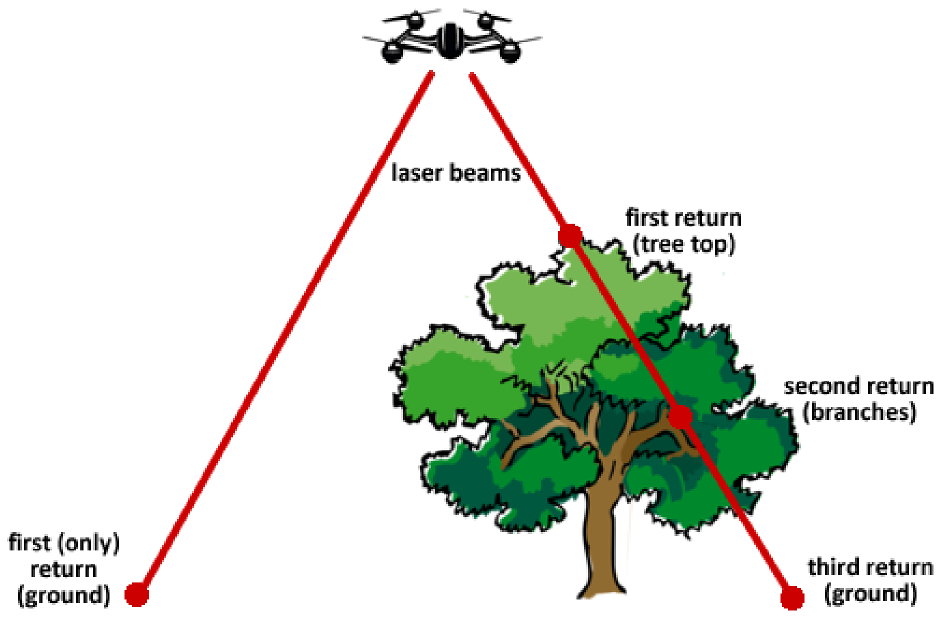

Figure 3.2: Example of a laser beam returns

Along with positional data (XYZ coordinates), the LiDAR point cloud can also contain additional attributes. Every point can store additional information such as intensity, return number, number of returns, point classification values, points that are at the edge of the flight line, RGB values, GPS time, scan angle or scan direction [8]. The extended information helps to categorize points and create more accurate dataset (e.g., we can consider only points with high intensity as valid).

3.2 IMU

An inertial measurement unit (IMU) is a device that measures Euler angles (roll, pitch, yaw - see Figure 3.3), angular velocity, acceleration and magnetic forces. To obtain these measurements, it uses accelerometers, gyroscopes and often also magnetometers. A typical configuration contains one accelerometer, gyroscope and magnetometer for each of three axes.

IMUs are sometimes also used for navigation purposes. The system continually integrates acceleration with respect to time to calculate velocity and position. But this approach has a significant disadvantage as it suffers from accumulated integration error. With every step the difference between the calculated and the actual position increases. Therefore the IMU is usually combined with GNSS system, and then the Kalman filter is used to process measurements from both sensors and provide corrected results.

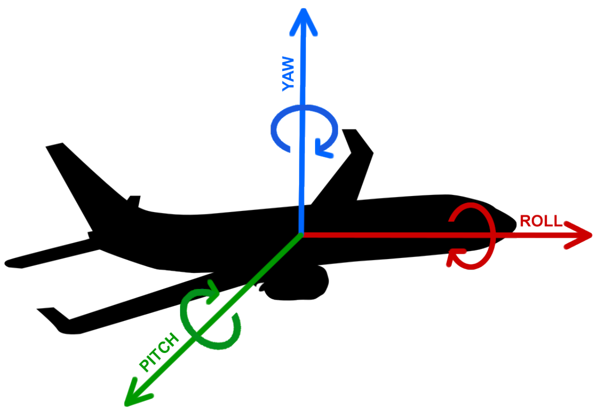

Figure 3.3: Euler angles - roll, pitch and yaw

3.3 GNSS

Global navigation satellite system (GNSS) is a satellite system that provides autonomous position measurements. Using small electronic GNSS receivers, that can decode satellite signals, we are able to determine receiver's location (longitude, latitude, and height). The accuracy is ranging from several meters to few millimeters depending on receiver's specifications and used positioning method.

The accuracy is influenced by many factors such as the type of the receiver (code delay or carrier phase receiver, single-frequency or multi-frequency receiver, number of channels, antenna type, etc.), current satellites visibility, ionosphere conditions, surroundings (high trees or buildings) or measurement method (single receiver, differential GNSS, real-time kinematics, etc.).

Even when using low-cost receivers, the accuracy can be significantly increased using differential measurements (DGNNS) or real-time kinematic (RTK) measurements. These two methods work with one (or more) additional receiver. The first receiver is called 'rover', and it moves through the environment and measures its path. The second one is called 'base', and it is placed on a stable place (tripod, the roof of a building, etc.) with the good satellite's visibility. This base receiver measures its precise fixed coordinates and saves it as a reference position. Based on this reference position it can compute differences (errors) and send this information to the rover and correct its current data (see Figure 3.4). Using RTK or GNSS method we can obtain absolute or relative positional data. It depends on whether the location of the base station is known (e.g., we can place the base receiver on the triangulation station or measure the accurate position in advance).

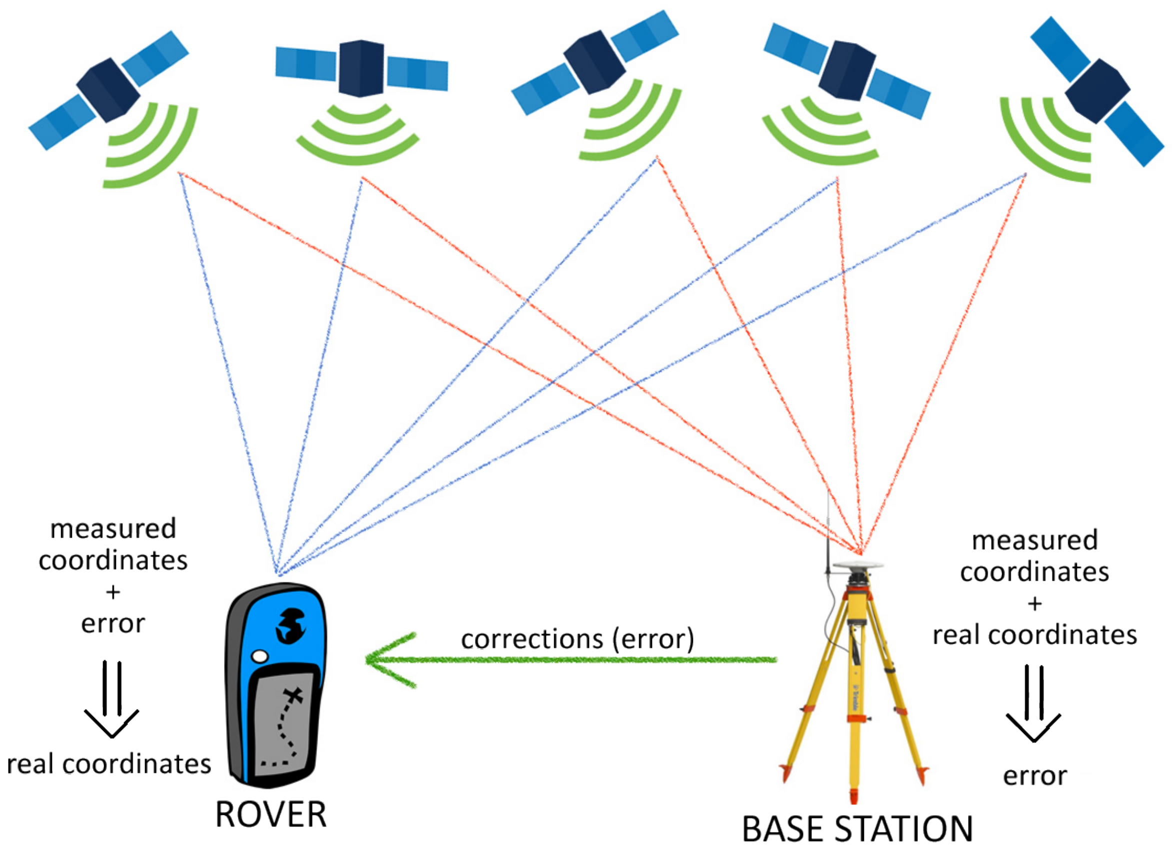

Figure 3.4: Illustration of RTK GNSS measurements

When we don't have two RTK receivers available, we can use permanent reference station instead of a base receiver. The permanent GNSS networks consist of static receivers that are located at several thousand locations all around the world. They permanently receive signals from all available navigation satellites and provide correction data. The CZEPOS is a reference network in Czech Republic that provides correction data neither for real-time or offline measurements [3, 20].

4 Filtering

The system (LiDAR with GNSS and IMU) can produce extensive and dense raw point sets. To be able to create a triangular mesh in subsequent steps we need to distinguish and remove outliers and maximally reduce noise and size of the data set.

There are many methods for point cloud filtering. The primary filtering approaches for 3D point cloud can be categorized into the following groups: statistical-based, neighbourhood-based, projection-based, signal processing-based, partial differential equations (PDE)-based methods and other hybrid filtering techniques. All of them were adequately described, tested and compared in [14].

Depending on computational and time complexity and errors, we chose to implement a robust moving least-squares (RMLS) algorithm for reconstructing a piecewise smooth surface from a potentially noisy point cloud. We use techniques from robust statistics to guide the creation of the neighbourhoods used by the moving least squares (MLS) computation. It leads to a conceptually simple approach that provides a unified framework for not only dealing with noise but also for enabling the modeling of surfaces with sharp features [12].

This algorithm is already included in Matlab's Computer Vision System Toolbox that was designed for 3D point cloud processing. This extension is commercial product and therefore we decided to implement our own algorithm. We implemented this algorithm guided especially by [12].

4.1 Robust moving least square algorithm

Moving least square (MLS) is a method of reconstructing continuous functions from a set of unorganized point samples via the calculations of a weighted least squares measure. It was described for example in [9]. This algorithm has an excellent capability to handle noisy input based on local fitting. It is easy to compute, and it gives us a perfect approximation of a surface. However, this approach is not resistant to outliers and sharp features. In this case, outliers can have a significant influence on the fitted model.

A statistical method is considered to be robust if it has a large breakdown point. Loosely speaking, a breakdown point is the proportion of incorrect observations an estimator can handle before giving an incorrect result. In our case, the breakdown point is 50 %. It means that it can reliably fit a model to data that contains up to 50 % outliers.

The algorithm filters the point cloud locally. Firstly we select arbitrary point and find its neighbouring points. If the number of points in the neighbourhood is sufficient for fitting the n-th order polynomial, we find the best fitting polynomial function that approximates current set of points. Then we project these points into the function (excluding outliers, i.e., points with larger distance than a given threshold). We store all the projected points and save their indices to make sure that every point will be projected only once. Then we select another arbitrary point from the point cloud that hasn't been projected yet, find his neighbourhood, find best fitting polynomial function and project neighbouring points. This process is repeated until there are some points from the point cloud left.

The point cloud usually consists of millions of points. It would be computationally demanding to work with such an extensive dataset. Therefore we need to reduce the size of the point cloud before the filtering and meshing process (after the filtering). For this purpose we use rounding (decreasing of the decimal precision) and cluster analysis.

4.1.1 Neighbourhood search

The model is being fitted locally. Therefore it is needed to use neighbourhood search algorithm. We can use k-nearest neighbours (k-nn) search or range search. The k-nn search works with a stable number of neighbouring points, but it can't handle unevenly dense point clouds. On the other hand, the range search can cover the point cloud evenly, but the number of neighbouring points can vary a lot. Because our input LiDAR datasets are unevenly distributed, we chose to use range search algorithm together with KD-tree structure (to speed up the searching process).

During the filtering of a point cloud the range parameter r remains constant. We need to estimate the range r with respect to the level of detail we want to achieve and the density of the point cloud. To obtain more detailed filtering we choose smaller r. But then the point cloud needs to be very dense (to fit the plane we need at least four points in each neighbourhood). For example if we want to filter a roadway with a curb we need to choose the r no larger than the size of the curb edge (height). Otherwise the curb will be smoothed.

4.1.2 Polynomial fitting

For every point in the point cloud, the points in range r are found. These points are then fitted with a polynomial function of n-th order.

The polynomial regression includes following steps:
1. from N points (all points in current range) select randomly k points
2. find a polynomial function that approximates these random points
3. compute the median of the residuals of the remaining N-k points
4. repeat this process T times to obtain different solutions
5. the best polynomial model has the minimal median

This algorithm can be expressed as

$$\underset{\beta}{\operatorname{argmin}} \operatorname*{median}_{i} |f_\beta(x_i) - y_i|,$$

where β represents the parameters of the polynomial function.

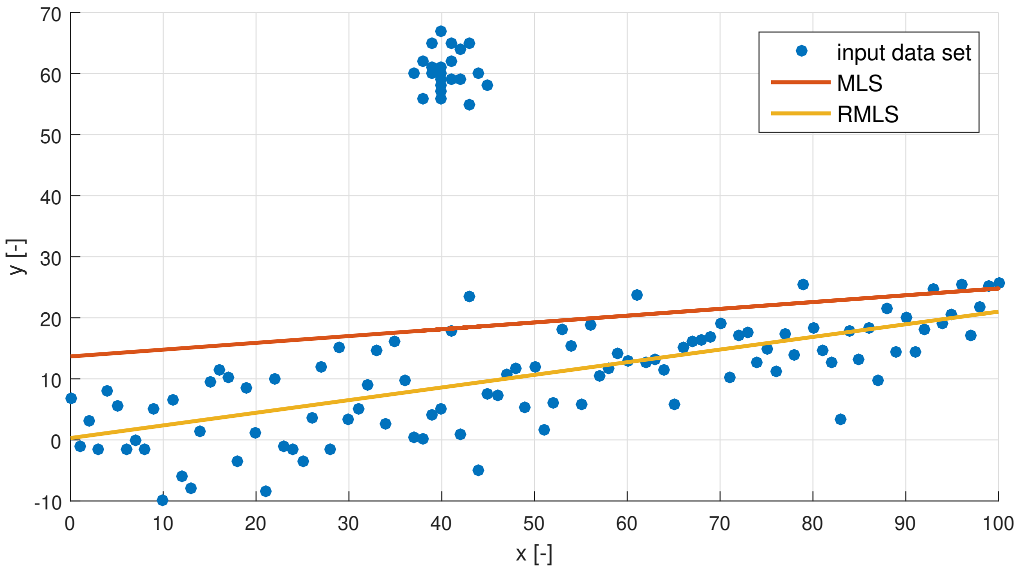

Figure 4.1: Comparison of MLS and RMLS approaches

We search for parameter β that minimizes the median of the absolute residuals. It makes the significant difference between MLS and RMLS approach. While MLS uses mean to estimate the function parameters, RMLS uses the median, which makes this algorithm robust and resistant up to 50 % outliers (see Figure 4.1).

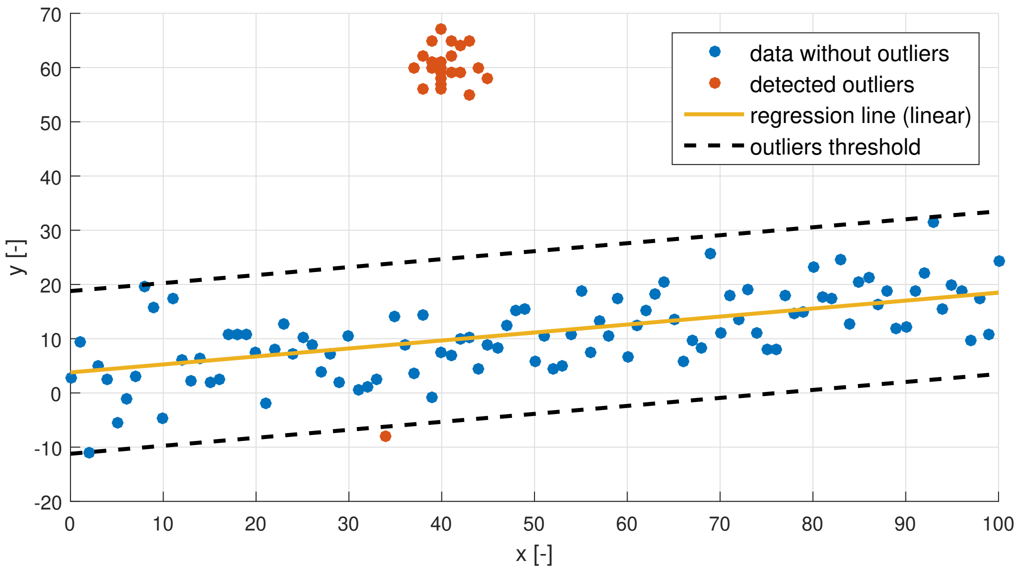

Figure 4.2: Example of outliers detection

Then we choose a threshold for outliers detection (see Figure 4.2). This threshold needs to be determined about the level of noise in the input point cloud to distinguish noise from outliers (observations with large errors that do not belong to the general model). In this work we set this threshold empirically. We examine the raw noisy point cloud in a sectional view to estimate the range of the noise. The threshold is the set equal to this noise range. We will further investigate the autonomous estimation of the threshold in a future.

4.1.3 Points projection

Finally, when we obtained the best polynomial function (described in Subsection 4.1.2) that approximates current dataset, we call for projection of these points (excluding outliers) to this polynomial function (see Figure 4.3). This can be described as $y_i = f_\beta(x_i)$. If there is some interval without input points, then there are no projection points generated. It could be solved using interpolation, but it could lead to misinterpretation of the input data.

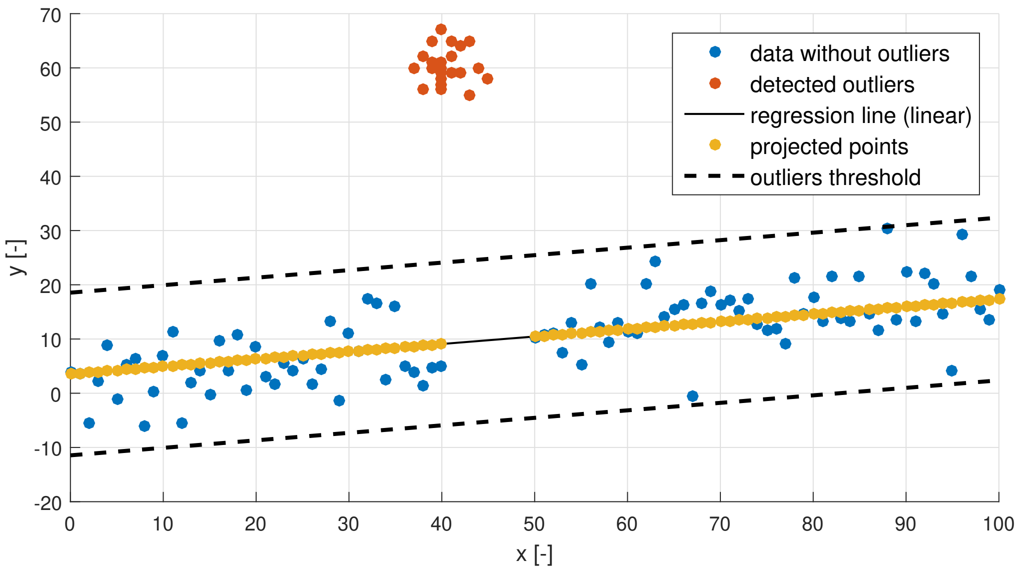

Figure 4.3: Projection of noisy points to polynomial function

4.2 Sharp features

When the surface contains sharp features, the requirement of being resistant to noise is especially challenging, since the noise and sharp features are ambiguous. Most techniques (e.g., fitting of higher order polynomial functions) tend to smooth important features or even amplify noisy samples. Moreover, sharp features consist of high frequencies of points which cannot be properly sampled by the finite resolution of the scanning device in the first place.

Points on sharp features are defined by multiple surfaces (e.g., edge or corner). Thus, dealing with sharp features requires fitting some surfaces locally. Using RMLS algorithm, we can define sharp features by treating the points across the discontinuities as outliers. If the ratio of detected outliers is higher than a specified threshold, then it is highly probable, that the surface contains sharp parts (see Figure 4.4).

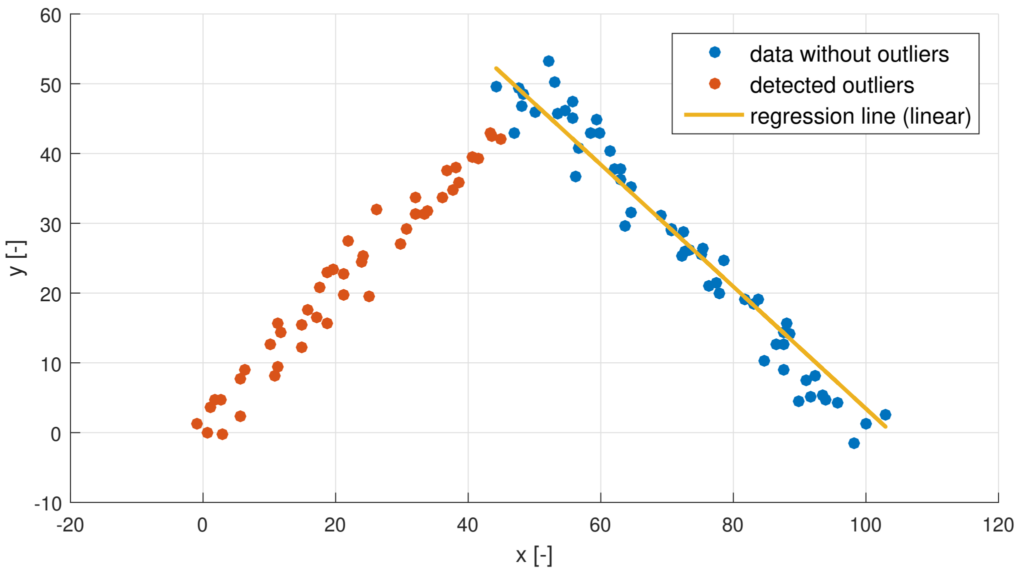

Figure 4.4: Detection of sharp feature

Secondly, we need to filter detected outliers again, and thus we obtain another polynomial function that approximates remaining points. Then we need to find an intersection point of both functions and decide which areas are going to create the new filtered dataset (see Figure 4.5).

When working in the 3D space, this problem becomes more difficult, because we are working with 3D surfaces (planes) and one (or even more) intersection lines. To define intersection line of two planes we can use basic linear algebra:

$$x_0 = A\backslash b,$$

$$x_{null} = null(A),$$

where A is a 2x3 matrix that represents three coefficients of both functions, b is a 3x1 vector that contains constants of both functions. Vector x_0 represents one particular found solution (one point that lies on the line) and x_{null} is a null vector (when we multiply Ax_{null}, we get a vector of zeros). This gives us the line equation in parametric form:

$$x = x_0 + tx_{null},$$

where t is an arbitrary parameter.

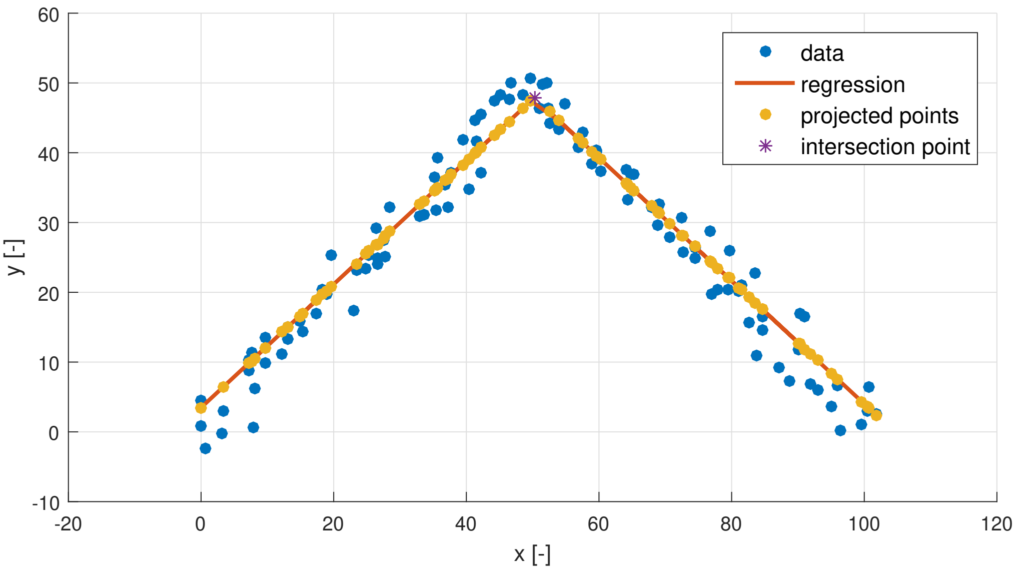

Figure 4.5: Dealing with sharp features

Then it is needed to decide which intersected surface area is going to be removed (see Figure 4.6). Therefore we compute vectors of point to line distances to know the octant where all points are placed:

$$P = P1 + \frac{(P2 - P1) \bullet v}{v \bullet v} v,$$

where $P1$ is an arbitrary point on the line, $P2$ is the point from which we want to have the closest point on the line, and P is the desired nearest point on the line. The octant where the point lies is then defined by the sign of two points difference $sign(P2 - P)$.

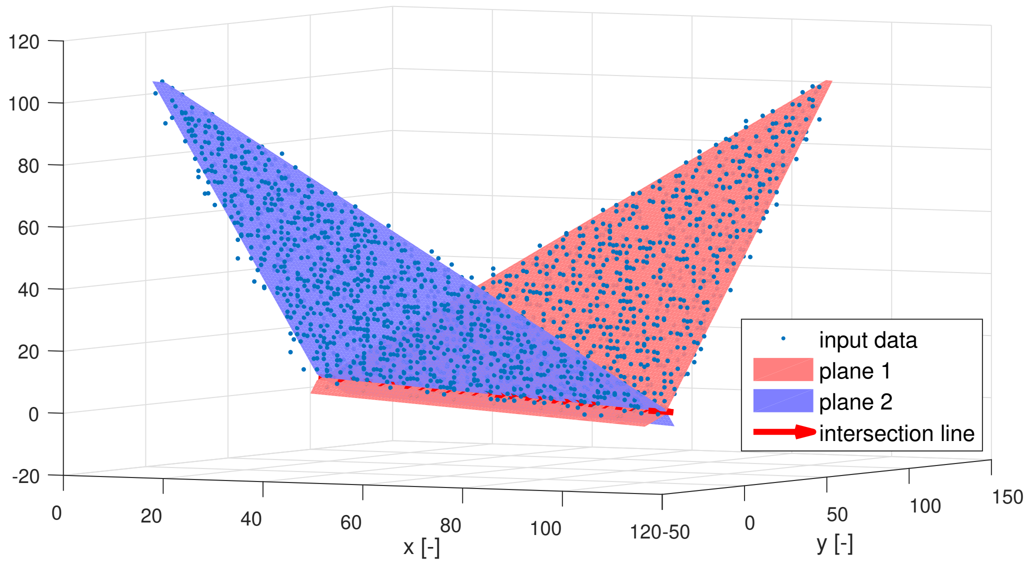

Figure 4.6: Dealing with sharp features in 3D space

These octants can be divided into six different half-planes depending on axis along which we want to split the data (e.g. if the first function lies in 'xy' plane and the second lies in 'xz' plane, then we want to differ the first data by 'y' coordinates and the second data by 'z' coordinates). After selecting the correct axis mode, we choose the half-plane that contains more points from the original dataset. Other points will be removed.

Corners are then treated similarly, but we are looking for three different intersection lines. Therefore we have to repeat the algorithm above for three different functions (planes) (see Figure 4.7).

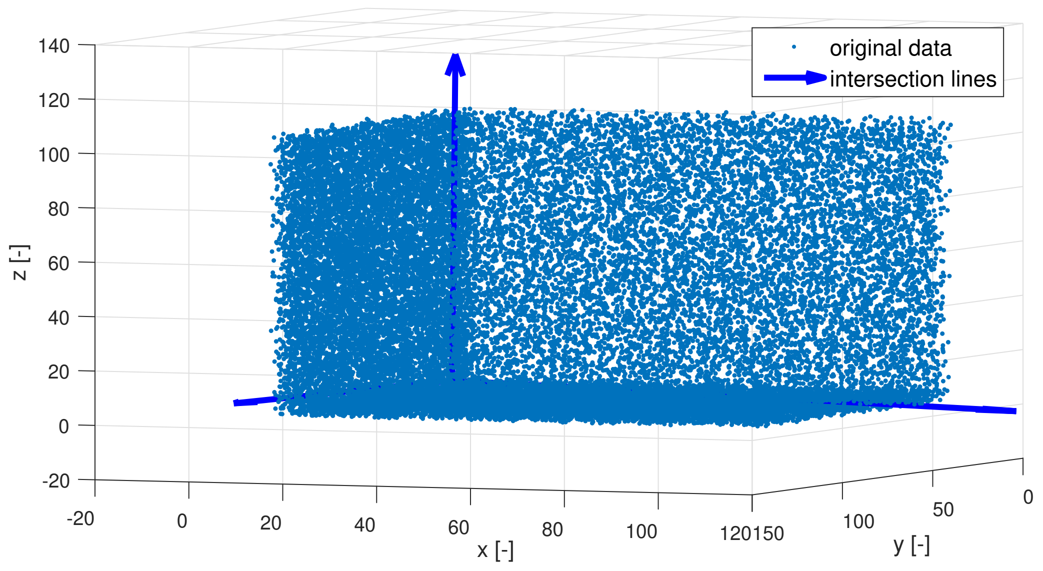

Figure 4.7: Filtering of corners

In this thesis we omit the intersection of four or more planes and intersection of generally non-linear surfaces, because then this problem becomes too complex and is beyond the scope of this thesis.

4.3 Implementation notes

This section provides some of implementation changes and improvements:

- To ensure the continuity and smoothness of the filtered point cloud it is helpful to sort all points in the point cloud before filtering by their position (e.g., primary sorting by x-axis or by the sum of x and y coordinates). It ensures that the next selected point is going to be in the neighbourhood of currently filtered area.
- To prevent significant differences of transitions between functions of neighbouring areas it is better to look for the $2r$ range of points and then filter current data depending on this larger surrounding area.

- In RMLS algorithm it is recommended to use median of residuals to obtain the best fitting function. As mentioned in [17] it is more reliable to use weighted random sample consensus (WRANSAC) approach, especially when dealing with outliers. In this case, we are looking for a function that suits (with small errors) to the biggest number of points.
- It is crucial to reduce the size of the point cloud. Therefore we rounded the precision of the input data set and removed redundant points.
- The number of randomly chosen points k (see Subsection 4.1.2) should be small enough (e.g., just four points from the dataset). Bigger k means that the probability of finding 'correct' function is lowered and it is needed to increase the number of repeating T. Because we need to prevent up to 50 % outliers, the k has to be always lower than the half of size of the data set ($k < 1/2N$).
- It is sufficient to approximate data sets only with first-order polynomials (planes) or second order polynomials at the most. Higher orders have no significant effect on precision and slower the computations a lot. When detecting sharp features, it is necessary to use first order polynomials, because finding intersection curves of general non-linear surfaces is not feasible.
- To further reduce the size of the resulting point cloud, we can remove points that are in clusters and therefore represent nearly the same points. We can compute the mean of these clusters and replace each cluster with only one point.

Algorithm 1: Implementation of filtering pseudo-algorithm

```
input      : point-cloud
output     : filtered point-cloud
parameters: radius, polynomial degree, T, k, minimum amount of points, outliers
            threshold, residuals threshold, edges threshold, sampling

down-sample the point-cloud by rounding and removing identical points;
sort all points by sum of their coordinates;
create KD-tree;
for all points do
    if point has already been filtered then
        continue;
    end
    currentData ← points in range r;
    extendedData ← points in range 2r;
    diffData ← currentData without already filtered points;
    if size of diffData $\geq$ minimum amount of points then
        [function,outliers] ← find regression function for extendedData (RMLS);
        project diffData (excluding detected outliers) into polynomial function;
        if ratio of outliers > edges threshold then              // edge detection
            [function,outliers] ← find regression function for outliers (RMLS);
            get intersection line;
            select correct points after intersection;
            if ratio of outliers > edges threshold then          // corner detection
                [function,outliers] ← find regression function for remaining outliers
                  (RMLS);
                get other two intersection lines;
                select correct points after intersection;
            end
        end
        store all filtered data;
    end
end
remove clusters;
```

5 Triangulation

A polygonal mesh consists of collection of vertices, edges and faces. It defines the shape of any object surface in 3D space. Polygonal meshes represent the environment in a compact but precise, continuous representation and can be used for example for localization, tracking, path planning or (as in our case) object volume calculations. The faces usually consist of triangles (triangle mesh), but they can generally consist of any simple convex polygons.

To make the carried geometric information accessible for object volume calculation, the input data has to be converted in a representation that can be handled by the chosen volume calculation algorithm. Our approach to achieving this goal is to compute polygonal (specifically triangular) representations of the scanned environments.

5.1 Delaunay triangulation

The algorithm was named after Boris Delaunay for his work, that was published in 1934 [10]. It is widely used in many applications, and it is one of the most popular methods used in problems related to generating of meshes.

The fundamental property is the Delaunay criterion. In the case of 2D triangulation, this is often called the empty circumcircle criterion. For a set of points in 2D, a Delaunay triangulation of these points ensures the circumcircle associated with each triangle contains no other point in its interior (see Figure 5.1) [4].

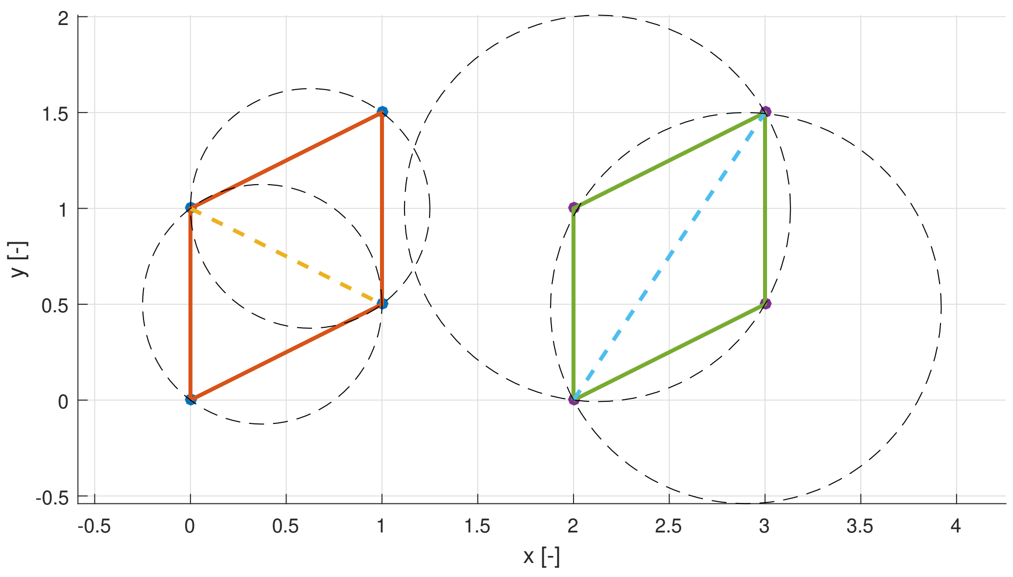

Figure 5.1: Delaunay (left) and non-Delaunay (right) triangulation

Many algorithms for computing Delaunay triangulations rely on fast operations for detecting when a point is within a triangle's circumcircle. In two dimensions, one way to detect if the point D lies in the circumcircle of the triangle ABC is to evaluate the determinant of matrix M [5,13]:

$$M = \begin{bmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{bmatrix} = \begin{bmatrix} A_x - D_x & A_y - D_y & (A_x - D_x)^2 + (A_y - D_y)^2 \\ B_x - D_x & B_y - D_y & (B_x - D_x)^2 + (B_y - D_y)^2 \\ C_x - D_x & C_y - D_y & (C_x - D_x)^2 + (C_y - D_y)^2 \end{bmatrix}$$

When points A, B and C are sorted in a counter-clockwise order, we can determine the position of the arbitrary chosen point D:

$$\begin{aligned} det(M) &> 0 \longrightarrow \text{D lies inside the circumcircle} \\ det(M) &= 0 \longrightarrow \text{D lies on the circumcircle} \\ det(M) &< 0 \longrightarrow \text{D lies outside the circumcircle} \end{aligned}$$

Properties of Delaunay Triangulation (DT) [5]:

- A circle circumscribing any Delaunay triangle does not contain any other input point in its interior.
- In the plane, the DT maximizes the minimum angle of all the angles of triangles in the mesh. Compared to any other triangulation of the points, the smallest angle in the DT is at least as large as the smallest angle in any other. However, the DT doesn't necessarily minimize the maximum angle. The DT also does not necessarily minimize the length of the edges.
- The union of all simplices in the triangulation is the convex hull of points.
- For a set of points on the same line, there is no DT (the notion of triangulation is degenerate for this case).
- For four or more points on the same circle (e.g., the vertices of a square, rectangle or isosceles trapezoid) the DT is not unique: each of the two possible triangulations that split the quadrangle into two triangles satisfies the Delaunay condition, i.e., the requirement that the circumcircles of all triangles have empty interiors (see Figure 5.2).

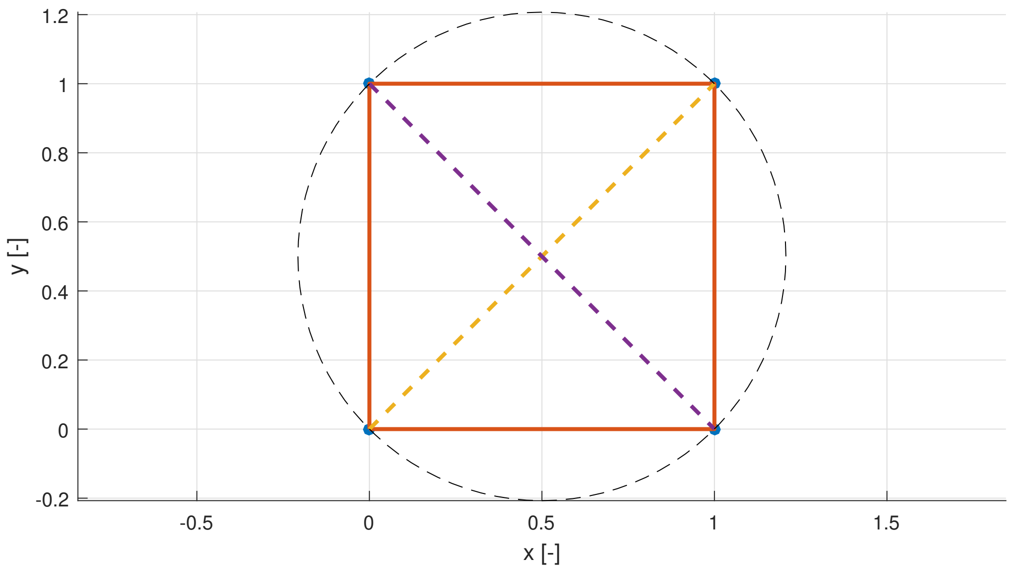

Figure 5.2: Special case of Delaunay criterion - points on the same circumcircle

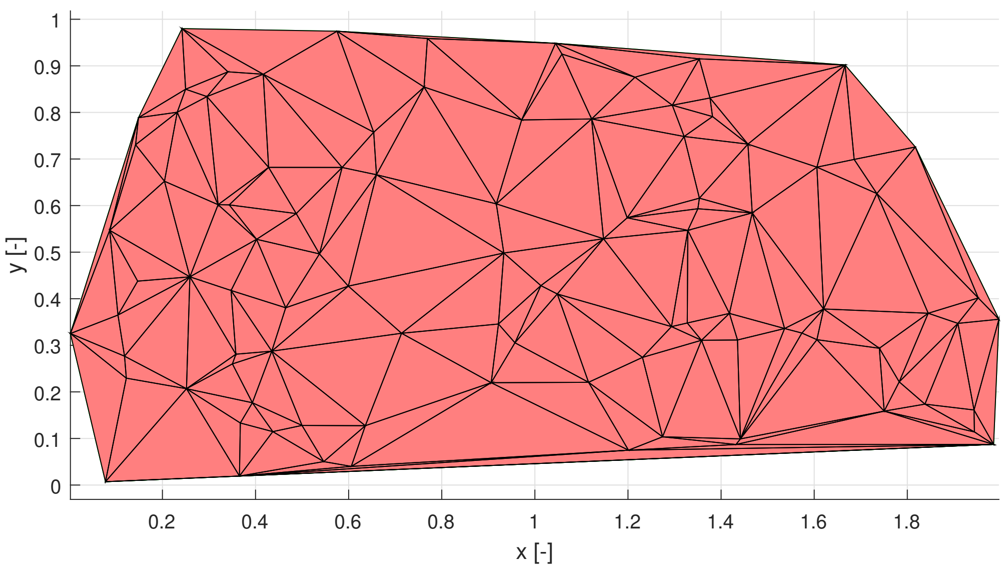

Figure 5.3: Example of Delaunay triangulation in 2D

5.2 Surface triangulation in 2.5D

The 2.5D Delaunay algorithm is based on the 2D Delaunay triangulation algorithm. The points are triangulated in the 2D space by projecting into xy-plane (considering just their x and y coordinates). The edges obtained between the 2D points are then applied to 3D space, leading to the so-called 2.5D triangulation (see Figure 5.4).

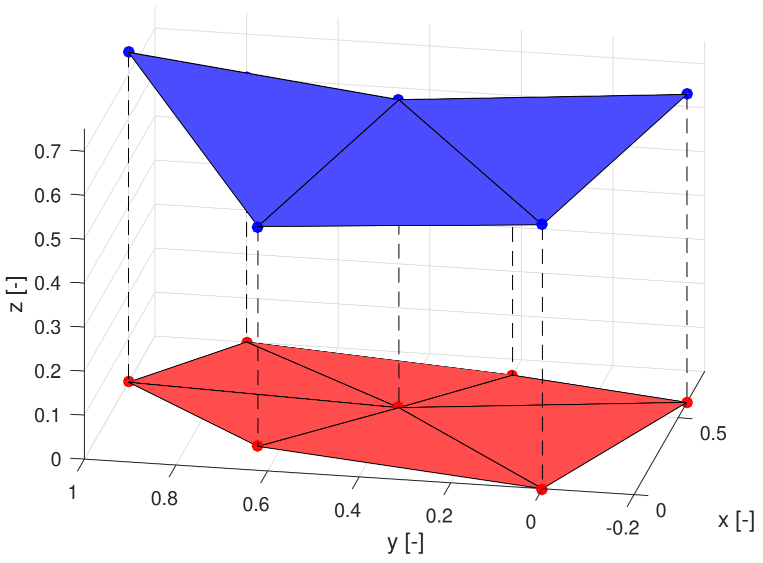

Figure 5.4: Delaunay triangulation in 2.5D

5.3 Implementation notes

- It is necessary to specify a maximum length of the triangle edges. It allows to remove large triangles (generally on the boundary) that are not necessarily meaningful.
- Because the dataset is not organized, it is required to calculate the triangulation piecewise. For every point in the dataset, we find neighbouring points in a range r and compute the DT.
- For the reason that points can be distributed in any plane, we compute normal vector for selected points (we approximate points with a plane and compute the plane's normal) in current iteration and transform (rotate) them into 'xy' plane to be able to do the projection described in Section 5.2 (e.g. if we have points in 'xz' plane, firstly we have to rotate them by $90°$ and project them into 'xy' plane).
- To speed up the algorithm it is efficient to store all points that can't be connected with any new triangles. These 'final' points are defined by all their neighbouring triangles. If the point (projected in 2D space) is surrounded by triangles and the sum of their angles that adjacent to this point is equal to $360°$, then this point can't be part of any other triangle in the mesh (is fully surrounded with triangles).
- The algorithm is very computationally demanding, especially when the point cloud is large and unevenly distributed. Therefore we resample all points by rounding (decreasing decimal precision) them and deleting all identical points.
- To avoid points that are lying on the same circumcircle (e.g., squares or rectangles) it is helpful to add some small noise to the point cloud. The resulting triangle mesh can be then applied on the original point cloud without the additional noise.

- Because the noise addition doesn't prevent not unique triangles robustly, it is needed to check for 'false' or 'empty' triangles in the final mesh (see Figure 5.5). The optimal mesh consists of triangles that have three other neighbouring triangles (with common edge). If some triangles have less or more neighbours, we need to check whether some triangle is redundant or missing.

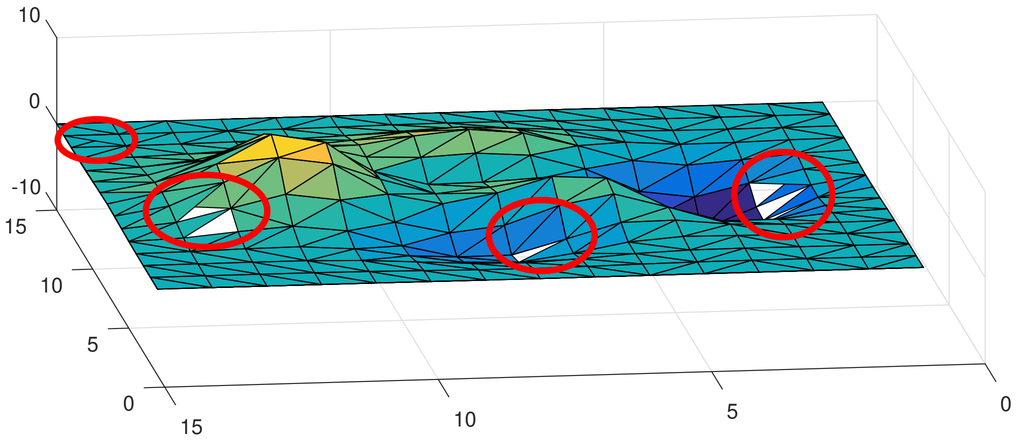

Figure 5.5: Example of 'false' and 'empty' triangles

The problem of 'false' and 'empty' triangles is caused by inaccuracies of determinant calculation (see Section 5.1). If a point is located on the circumcircle (or is very close to the circumcircle) of some triangle, the floating point precision can be insufficient. The algorithm can falsely decide which triangle is correct and which is not. It leads to overlapping or missing triangles (as shown in Figure 5.5).

We implemented an algorithm that checks the resulting triangular net for these 'false' or 'empty' triangles. Firstly we find all suspicious triangles. Those are triangles that don't have exactly three neighbouring triangles (with common edge). If they have less then three neighbours, they are likely to be located next to an 'empty' triangle. Otherwise, if they have more then three neighbouring triangles, they can be the falsely generated triangles.

In the second step all the suspicious triangles are further examined. Firstly we check all triangles that are likely to be 'false' (overlapping). We start with the triangle that has the most neighbours and we remove it from the mesh. Then we check the suspicious triangles for their neighbours again, because by removing one triangle other triangles can be no longer overlapping. We are removing triangles until only correct ones are left. Secondly we check all triangles that has only two neighbouring triangles. These can be located either at the edge of the point cloud or next to an 'empty' triangle. Therefore we are looking for three suspicious triangles that have three common vertices (they are gathered around the 'empty' triangle) - see Figure 5.6.

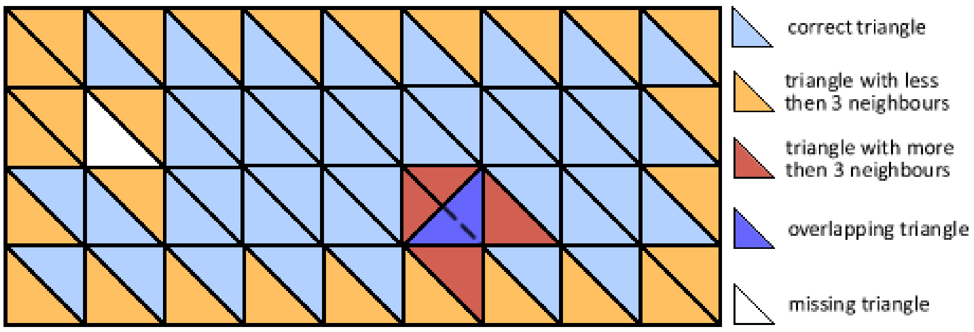
correct triangle
triangle with less then 3 neighbours
triangle with more then 3 neighbours
overlapping triangle
missing triangle

Figure 5.6: Illustration of 'false' and 'empty' triangles detection

This problem can be partly prevented by adding small noise to the filtered point cloud. It prevents especially right-angled triangles. Remaining 'false' and 'empty' triangles can be removed/added by algorithm described above. It can correct up to 80-90 % of falsely generated triangles. We need to further improve this algorithm to be able to detect and correct all overlapping or missing triangles in the mesh.

Algorithm 2: Implementation of Delaunay triangulation in 2.5D

```
input      : point-cloud
output     : list of trinagles
parameters: radius

add small random noise to prevent right-angled triangles;
create KD-tree;
for all points in point-cloud do
    check if current point is 'final';
    find all neighbouring points in range r;
    if number of range points > 2 then
        get normal vector and rotate points to xy plane;
        remove all final points;
        find all possible combinations of three points (triangles);
        remove all triangles with sides longer than threshold;
        for all triangles do
            check whether all three vertices are on the same line;
            if if any other neighbouring point isn't in circumcircle of this triangle then
                store this triangle;
            end
        end
        check current dataset for final points;
    end
end
check for false triangles;
check for empty triangles;
```

6 Ground plane detection

Plane detection is a prerequisite to various tasks. In computer vision, one of the most widely known methods for plane detection is the random sample consensus (RANSAC) algorithm [11]. It has been proved, that this algorithm successfully detects planes in 2D as well as in 3D space. It is reliable even in the presence of a high proportion of outliers.

6.1 RANSAC

The principle of RANSAC algorithm consists of searching for the best plane that approximates a set of 3D points. We randomly select number of points (note: when fitting a plane, it is sufficient to select just three random points) and it calculates the polynomial function of the corresponding surface (plane). Then it detects all points of the original cloud belonging to the obtained surface (plane), according to a given threshold. It repeats these procedures N times. Every time it compares the obtained result with the last saved one. If the new result is better (corresponds to the larger amount of points in point cloud), it replaces the saved result by the new one (see Figure 6.1) [21].

This principle assumes that the ground plane corresponds with the largest plane in the dataset. It can be problematic when we would be working with dense urban areas, where the ground doesn't necessarily cover the biggest space. But on the other hand, this algorithm can deal with discontinuities and general non-linear surfaces (using higher degree polynomials).

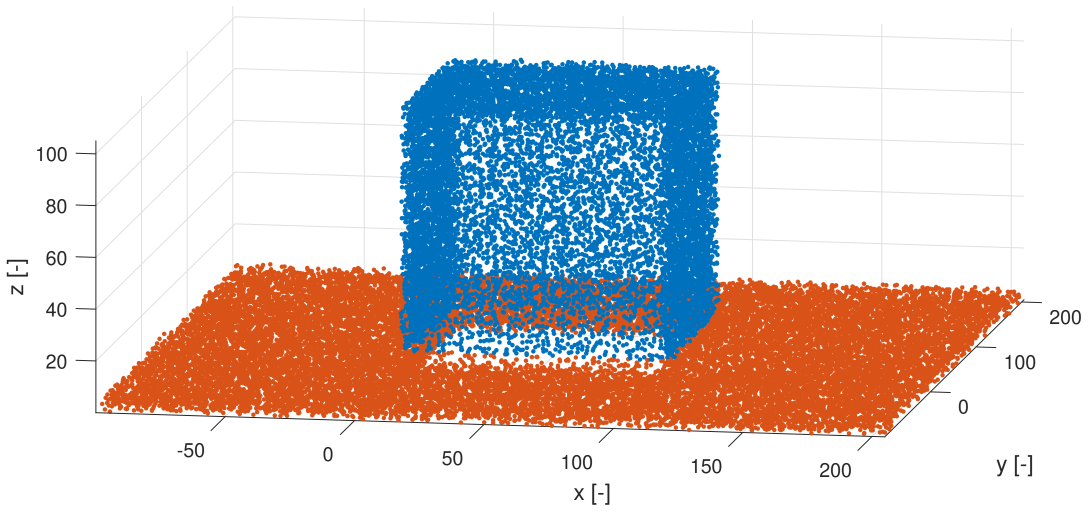

Figure 6.1: Ground plane detection

7 Volume calculation

There are two main approaches of volume calculation. One approach is to convert the model into a discrete 3D binary image (we fill the object with small cubes and sum their volumes). However, the transformation from a 3D mesh into a binary image is very time-consuming, and to improve the accuracy, the resolution of the 3D binary image needs to be very high (we have to use infinitesimally small cubes), which can further increase the computation load [22].

Another approach is to compute the volume from a polygonal mesh. This method is less computationally demanding, and we already obtained triangular mesh in previous steps.

The assumption for this algorithm is that the considered object is closed. If it is not, the closing process has to be performed first - see Section 7.3. If there are just some small gaps in the mesh or some overlapping triangles, they can be ignored, but it will lead to inaccuracies in the final result of volume calculation.

7.1 Organized triangles

Firstly we have to edit the triangular mesh to ensure the consistency of the direction of the normal vector of each triangle in the mesh. The direction of the normal vector of triangle is defined by order of its vertices. The normal vectors of two neighbouring triangles are consistent if their common edge has different directions. As shown in Figure 7.1, the two triangles ACB and ABD have common edge AB. In the triangle ACB the direction is from B to A and oppositely in the triangle ABD the direction is from A to B. Therefore the condition is met and the normal vectors N_{ACB} and N_{ABD} are consistent.

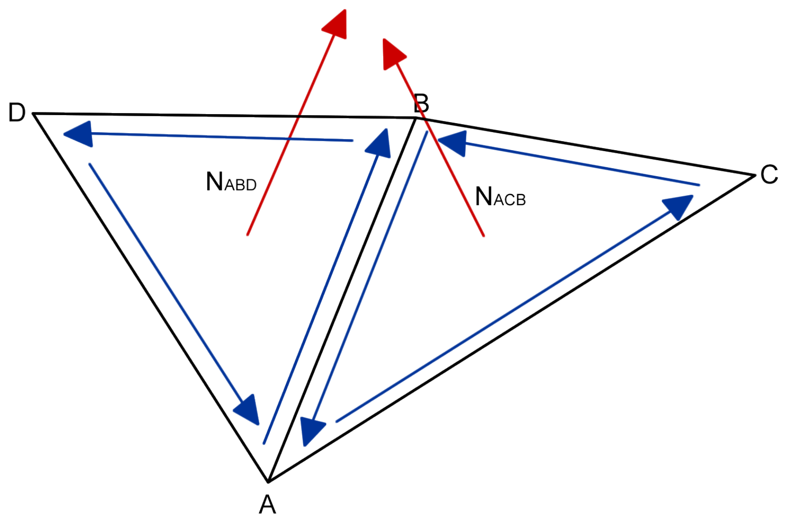

Figure 7.1: Direction of normal vectors and order of vertices

7.2 Signed volume of triangles/tetrahedrons

To explain the computing of the signed volume, we will start by explaining this algorithm in 2D space, where we want to compute an area of a polygon.

Firstly we need to obtain normals of each edge. These normals have to be pointing to the same side of the polygon (either inside or outside, as long as the direction is consistent). After getting the normals, we construct a set of triangles by connecting all the polygon vertices with the origin (or an arbitrarily chosen point). The area of this polygon is then defined as the sum of signed areas of all triangles, where the sign of the area is determined by the checking the position of the origin concerning the edge and the direction of the normal [22]. If the normal vector is pointing towards the origin, the sign of the area is negative and the other way around (see Figure 7.2). The area is defined as $S_{total} = \sum_i S_i = \sum_i \frac{1}{2}(-x_{i2}y_{i1} + x_{i1}y_{i2})$.

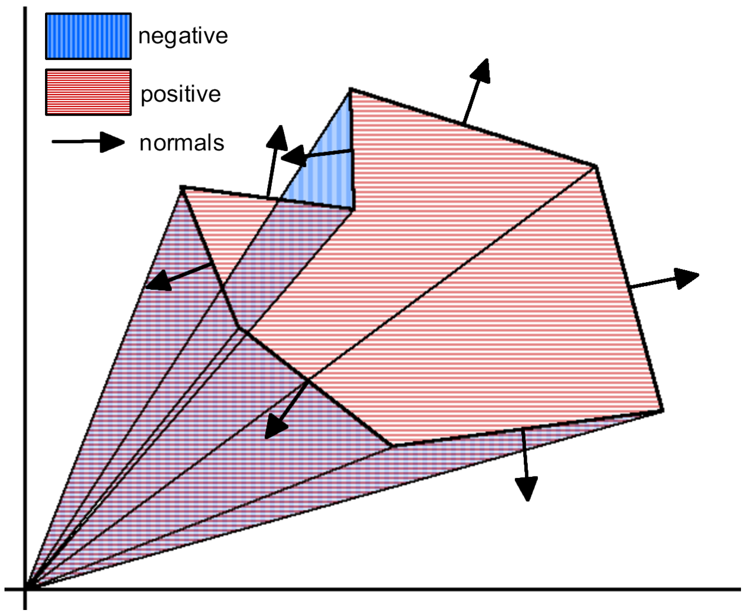

Figure 7.2: Area calculation using signed areas of triangles

Analogous approach can be applied in 3D space, where the elementary computational unit is tetrahedron. Again we have to sort all triangle vertices in such way that their normal vectors will be consistent as described in Section 7.1. Then we construct a set of tetrahedrons by connecting all triangles from mesh with an arbitrary chosen point (e.g., origin). The volume of the object is then defined as the sum of signed volumes of all tetrahedrons:

$$
\begin{aligned}
V_{total} &= \sum_i V_i \\
&= \sum_i \frac{1}{6}(-x_{i3}y_{i2}z_{i1} + x_{i2}y_{i3}z_{i1} + x_{i3}y_{i1}z_{i2} - x_{i1}y_{i3}z_{i2} - x_{i2}y_{i1}z_{i3} + x_{i1}y_{i2}z_{i3}) \\
&= \sum_i \frac{1}{6}(p_{i1} \bullet (p_{i2} \times p_{i3})),
\end{aligned}
$$

where $p_i = (x_i, y_i, z_i)$ are vertices of the triangle and i is index of each triangle in mesh.

The example of this algorithm is depicted in Figure 7.3. The object $ABCD$ consist of four faces (triangles). Each triangle is connected with the origin O that construct four tetrahedrons. The sign of the tetrahedrons volume is then defined by the direction of the normal vector (e.g., normal vector N_{ABC}).

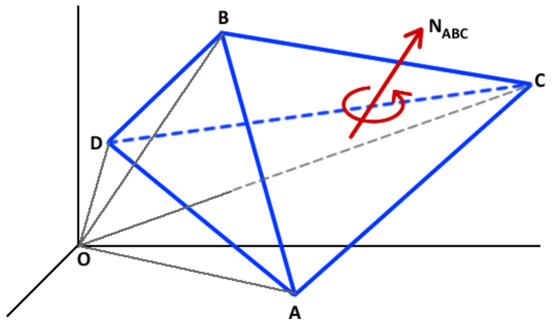

Figure 7.3: Volume calculation using signed volumes of tetrahedrons

The volume of an object is always positive. The negative volume can be obtained when using the opposite direction of the normal vectors. Therefore the final volume should be computed as the absolute value of the result.

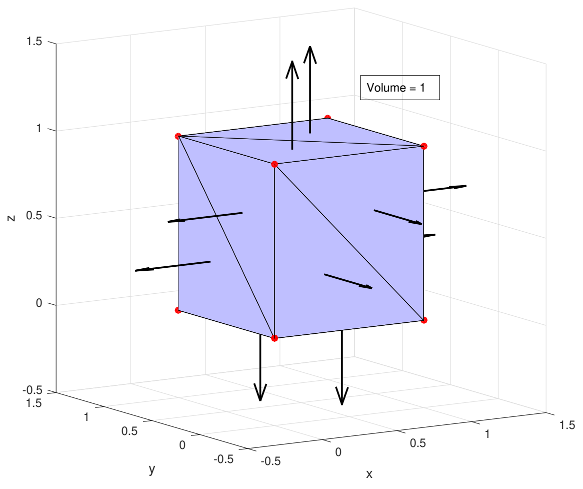

Figure 7.4: Example of volume calculation

7.3 Closed surfaces

A closed surface is a surface that is compact and without boundary. It has no outer edges and it has an inside and an outside.

The number of vertices, edges and triangle faces can be estimated using Euler's formula for triangle meshes:

$$V + F - E = 2$$

Since we are talking about a closed triangle mesh, there is a fixed relationship between the number of edges and the number of faces. To derive this it is helpful to think of the mesh as being made of half-edges. A half-edge is a pair of an edge and a face it borders. The total number of half-edges in the mesh is $2E$, since each edge has two halves. It is also $3F$, since each face touches three half-edges and this counts all the half-edges exactly once. Therefore $2E = 3F$. By substituting into the formula $V + F - E \approx 0$, we can easily derive the fact that $F \approx 2V$. Therefore we know that number of triangles in closed mesh will be approximately two times the number of vertices.

When scanning an object in real environment we usually can't obtain a closed object directly from the scan. Some parts of the object can be invisible during the scanning process (e.g., bottom side of the box placed on the ground). For this purpose we can use the ground plane detection algorithm (described in Chapter 6) and approximate the missing part of the object with points located on the ground plane.

To estimate the borders of the object it is possible to implement an autonomous object detection algorithm or we can select the border points manually. The autonomous object detection is very complex task and therefore we decided to use the manual selection of points in the point cloud. Then we can estimate which points from the plane are part of the object. When we estimated the missing side with points from the ground plane, we can also obtain the missing part of the triangular mesh.

8 Data source

This work uses sets obtained from previous projects that are described in detail in [15,19]. The system that provides us data for this application is composed of LiDAR, GNSS receiver, IMU, and auto-pilot unit (depicted in 8.1). Raw data from all sensors are being stored on-board via the Toradex SoC module and then processed offline.

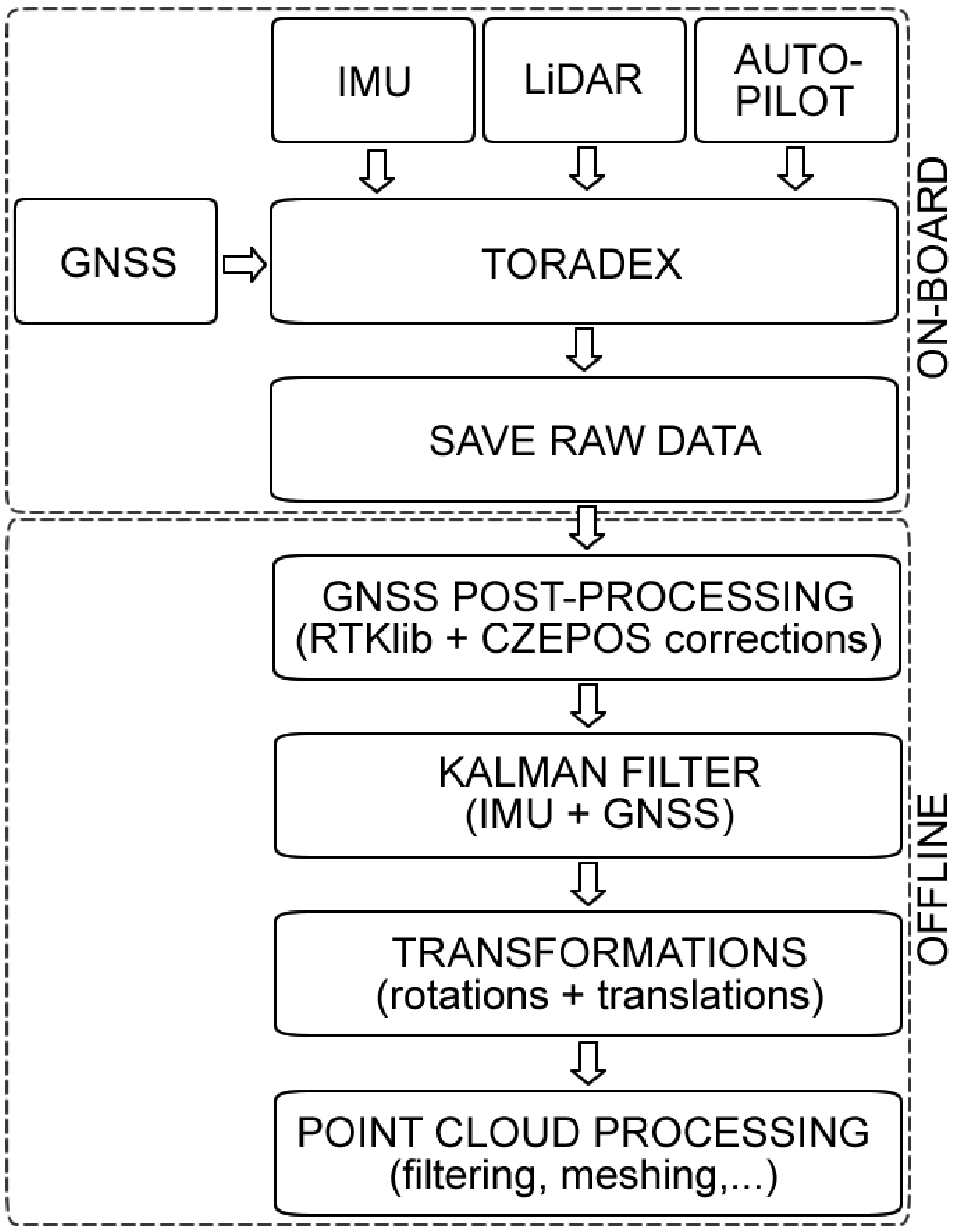
IMU
LiDAR
AUTO-PILOT
GNSS
TORADEX
SAVE RAW DATA
ON-BOARD
GNSS POST-PROCESSING (RTKlib + CZEPOS corrections)
KALMAN FILTER (IMU + GNSS)
TRANSFORMATIONS (rotations + translations)
POINT CLOUD PROCESSING (filtering, meshing,...)
OFFLINE

Figure 8.1: Rover/UAV measuring system

8.1 Velodyne VLP-16

Velodyne VLP-16 (Figure 8.2) puck is real-time 3D LiDAR sensor that weighs just 830 g. It is capable of measuring objects distant up to 100 m with the precision of 3 cm. It's 16 channels generates 300 000 data points per second. It has 360° horizontal field of view and ±15° vertical field of view.

Figure 8.2: LiDAR Velodyne VLP-16

8.2 Novatel OEM628

Novatel OEM628 (Figure 8.3) is real-time kinematic (RTK) based triple-frequency GNSS receiver. It can decode signals from constellations of GPS, GLONASS, Galileo, BeiDou and terrestrial SBAS. It can provide data with frequency up to 20 Hz. While the positional data are being post-processed, we also use RINEX data from the network of permanent GNSS stations in the Czech Republic (CZEPOS). It gives us corrections with which we can achieve accuracy up to 1 cm. This process is also known as differential GNSS (DGNSS or DGPS).

This receiver and reference network was described in detail and tested in [20].

Figure 8.3: Novatel OEM628 GNSS reciever

8.3 Microstrain 3DM-GX4-45

Inertial Measurement Unit (IMU) Microstrain 3DM-GX4-45 (Figure 8.4) provides a wide range of triaxial inertial measurements. It includes direct measurement of acceleration, angular velocity or atmospheric pressure. Sensor measurements are processed through an extended Kalman filter algorithm to produce high accuracy computed outputs. The output accuracy is 0.25° for roll and pitch angles and 0.8° for heading [1].

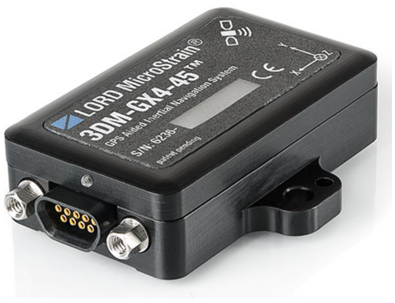

Figure 8.4: IMU Microstrain 3DM-GX4-45

8.4 Toradex Colibri T30 and Iris

Toradex Colibri T30 (Figure 8.5) is a computational unit that runs software for data acquisition. Collected data are then logged to its microSD card. This unit is also connected with Toradex Iris (Figure 8.5) carrier board that offers many interfaces such as Ethernet, UART RS-232 or USB which provide a connection with all sensors mentioned above.

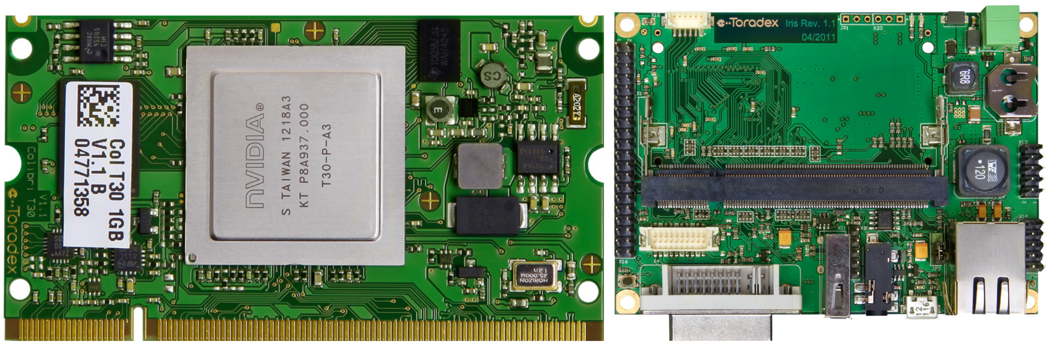

Figure 8.5: Toradex Calibri T30 (left) and Toradex Iris (right)

8.5 Rover/UAV

The whole measuring system is mounted on a rover or small UAV to obtain terrestrial or aerial point cloud data. For this purpose, all sensors and components were assembled into one rig (see Figure 8.7). This rig is compact, mobile and also covers valuable and fragile parts. The rover platform was created from a small model of RC car (Figure 8.6). Its roof cover was detached, and we built our platform on the chassis instead. This set up was used especially in the early phase of research and development for testing and debugging. For advanced aerial measurements, we mounted the rig on VTU BRUS drone (Figure 8.6), very stable high-performance coaxial hexa-copter that was developed for military purposes.

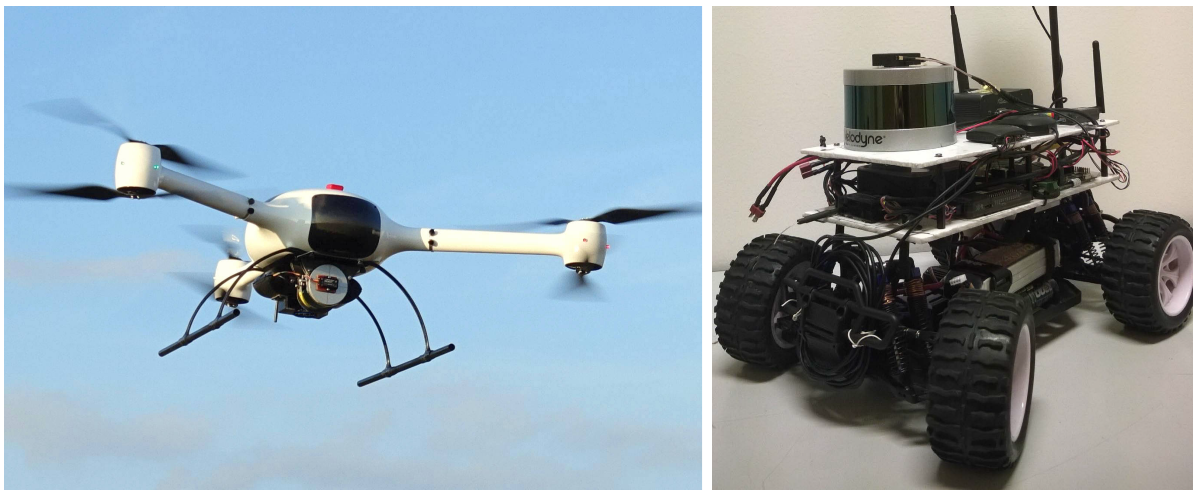

Figure 8.6: VTU BRUS drone (left) and RC rover (right) with our measuring system on-board

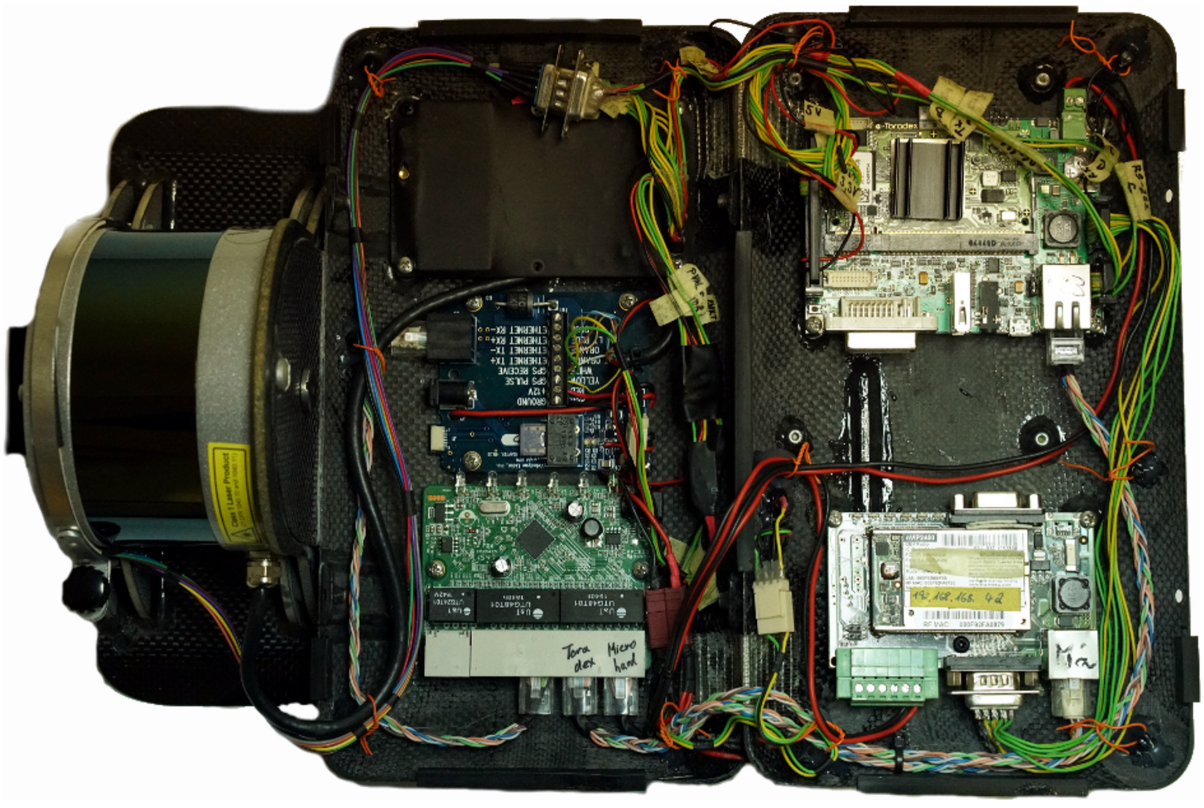

Figure 8.7: All sensors and components assembled into the rig

9 Experiments

This chapter is dedicated to the performed experiments of all algorithms described in previous chapters. It was tested mainly on three different datasets.

The first point cloud was captured in the real environment in Prague at Charles square (see Figure 9.1) with the measuring system of rover platform (described in Chapter 8). The point cloud features a bench and part of a fountain (see Figure 9.2). It contains 135100 points with average density 3680 pts/m^2. The level of noisiness is approximately ± 0.05 m. The path around the fountain is not filled with points, because the constellation of the rover and LiDAR doesn't allow to capture objects under the platform.

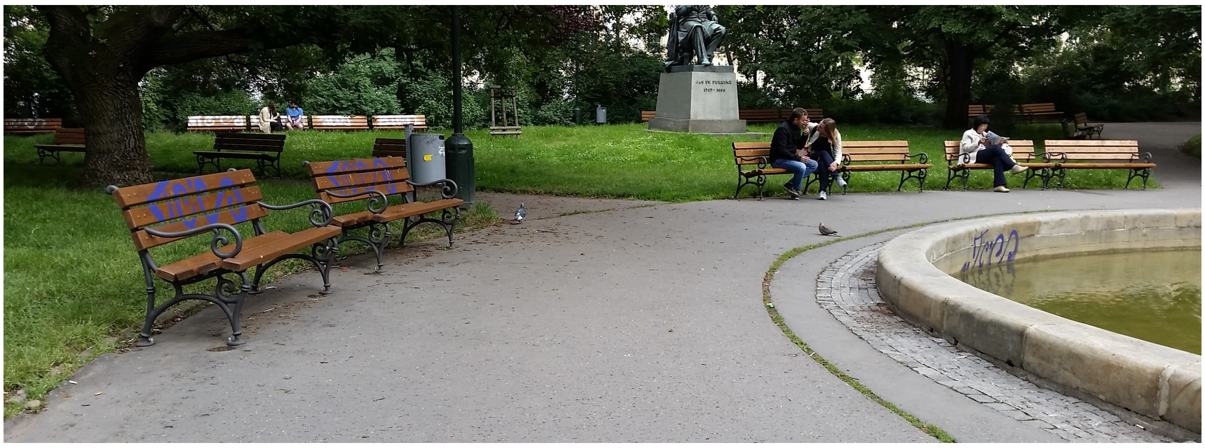

Figure 9.1: Photo of the scanned part of the Charles square

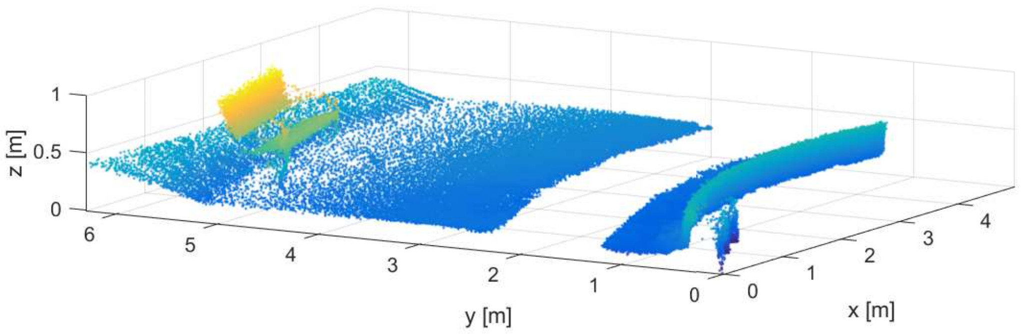

Figure 9.2: Raw point cloud captured at Charles square

Remaining two datasets were generated by simulations. One point cloud contains simulation of scanned cardboard box located on a plane (see Figure 9.3) and the other contains simulation of a sphere measurements (see Figure 9.4).

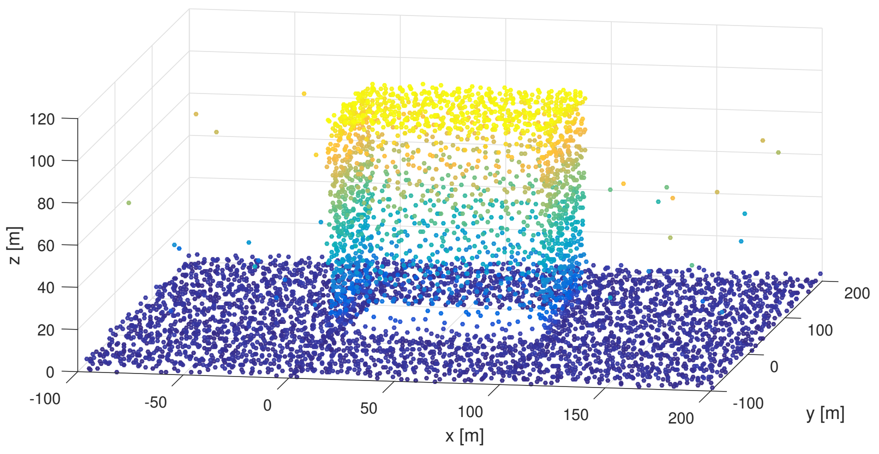

Figure 9.3: Simulation of captured cardboard box

The ratio of noisiness was kept the same. The level of noise in both point clouds is ± 5 cm. We also added 100 random points (in a range of each point cloud) into each dataset to form random outliers. The 'cardboard box' dataset consist of 5800 points with average density 650 pts/m^2 and the 'sphere' dataset contains 2200 points with average density 550 pts/m^2.

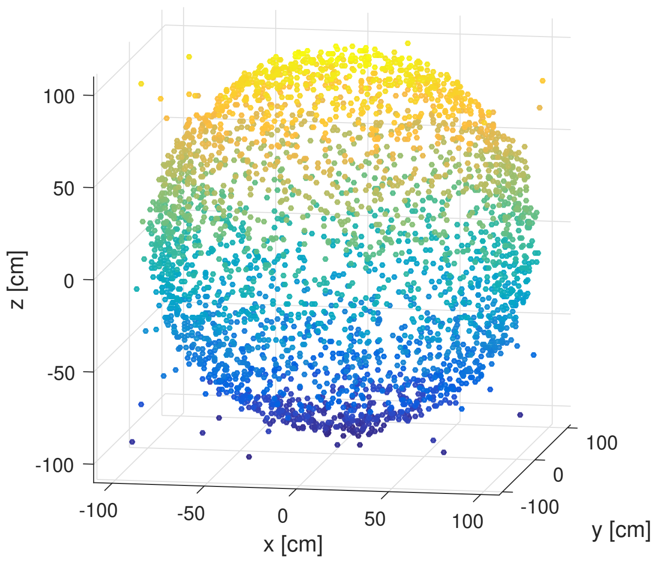

Figure 9.4: Simulation of captured sphere

These scanned objects are needed especially for the testing of object volume calculation. Unfortunately, the measuring system was inoperative and therefore we weren't able to capture object in a real environment. The LiDAR sensor had issues with fault timestamps and had to be sent to the manufacturer to be repaired.

9.1 Filtering

To filter the raw point cloud, we used the polynomial fitting method, namely RMLS algorithm with implementation changes and improvements (described in Chapter 4). This method was applied to all testing datasets.

The settings used for the first 'bench and fountain' dataset filtering are shown in Table 9.1. These values were set by empirical evidence and testing. The result of this particular configuration can be seen in Figure 9.5. As seen in this figure, the point cloud doesn't contains any outliers and the level of noisiness is reduced to the minimum. The density of the point cloud was also reduced (the raw point cloud includes more than 135 thousand points, and the new filtered point cloud consist just of 18 thousand points). It took less than 300 seconds to compute the new filtered point cloud.

parameter	**value**	**note**
delta	0.2	radius for finding nearest neighbours
polDeg	1	maximal polynomial degree
T	200	number of RMLS iterations
k	20	number of randomly chosen points for polynomial fitting
minPts	polDeg*k	minimum points used for regression in one neighbourhood
sampling	0.03	minimal distance between two points
outliersThreshold	0.05	points further from the regression function than this threshold are treated as outliers
residualsThreshold	0.02	points up to this threshold from the regression function counts as 'best fitting' points - when using WRANSAC
edgesThreshold	1/10	ratio of outliers in current dataset for edge/corner detection

Table 9.1: Settings for filtering of the 'bench and fountain' dataset

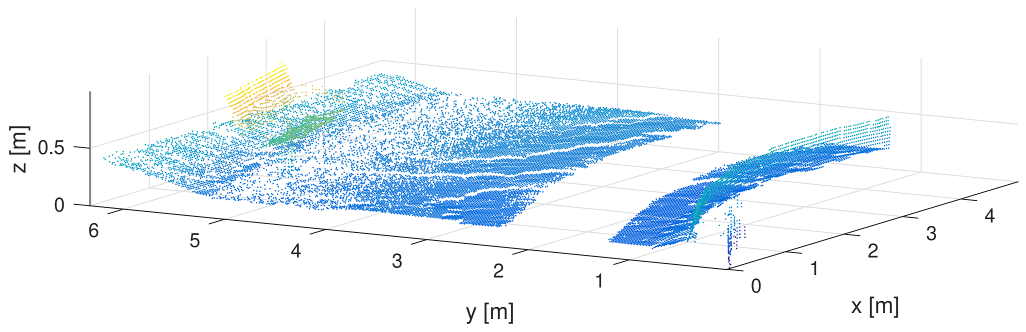

Figure 9.5: Filtered 'bench and fountain' dataset

The settings that were used for the 'cardboard box' dataset are listed in Table 9.2. We chose to use planes (linear polynomial functions) for regression. In this case, the detection and filtering of edges and corners were crucial. The resulting filtered point cloud consists of 5100 points and it took approximately 70 s to compute it.

parameter	**value**	**note**
delta	30	radius for finding nearest neighbours
polDeg	1	maximal polynomial degree
T	300	number of RMLS iterations
k	10	number of randomly chosen points for polynomial fitting
minPts	polDeg*k	minimum points used for regression in one neighbourhood
sampling	2	minimal distance between two points
outliersThreshold	5	points further from the regression function than this threshold are treated as outliers
residualsThreshold	3	points up to this threshold from the regression function counts as 'best fitting' points - when using WRANSAC
edgesThreshold	1/10	ratio of outliers in current dataset for edge/corner detection

Table 9.2: Settings for filtering of the 'cardboard box' dataset

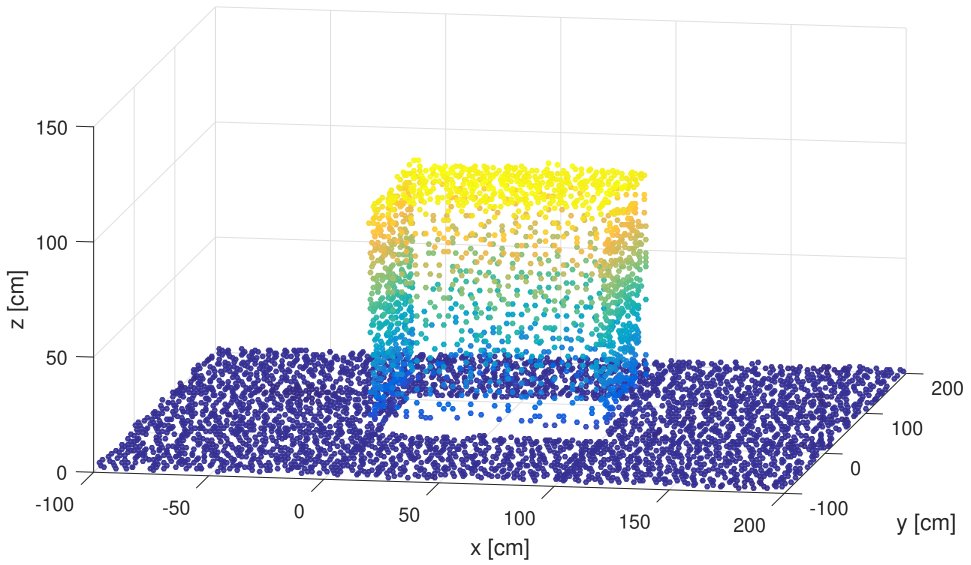

Figure 9.6: Filtered 'cardboard box' dataset

For the 'sphere' dataset we used the same settings as for the 'cardboard box' dataset with only difference - we chose to use the second degree polynomial functions to approximate the surface. The final filtered point cloud was computed in less then 130 s and it consists of 1800 points.

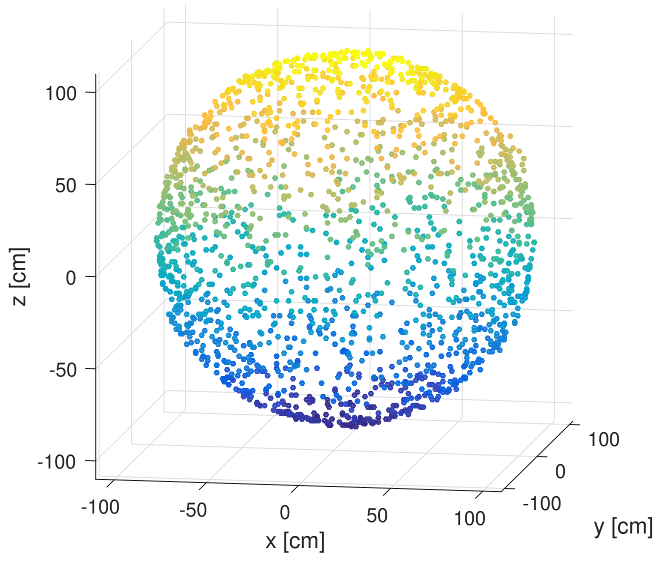

Figure 9.7: Filtered 'sphere' dataset

9.2 Triangulation

To create a triangular mesh, we used the algorithm described in Chapter 5. In this algorithm, there is only one used parameter, which is the r - radius of the range-search algorithm. In this case it was set to $r = 0.1$. The final triangular mesh of the 'bench and fountain' dataset is shown in Figure 9.8. As you can see, the part of the mesh under the bench is missing. It is caused by the insufficient density of points in this area. It can be solved by increasing the r parameter, but it could lead to false connections of other distant points. Therefore it is essential to find an optimal range parameter to balance it.

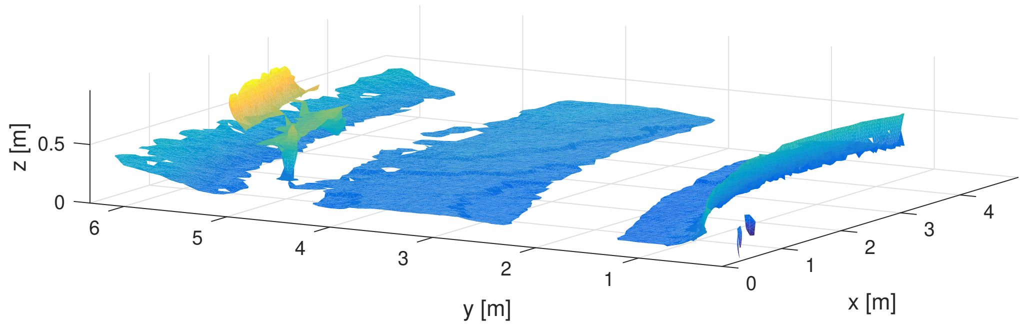

Figure 9.8: Triangular mesh of the 'bench and fountain' dataset

For the 'cardboard box' dataset the parameter was set to $r = 20$. The result of meshing can be seen in Figure 9.9. It took approximately 500 s and we generated 10650 triangles.
For the 'sphere' dataset the parameter was set to $r = 35$. The result of meshing can be seen in Figure 9.10. We generated 3724 triangles in approximately 250 s.

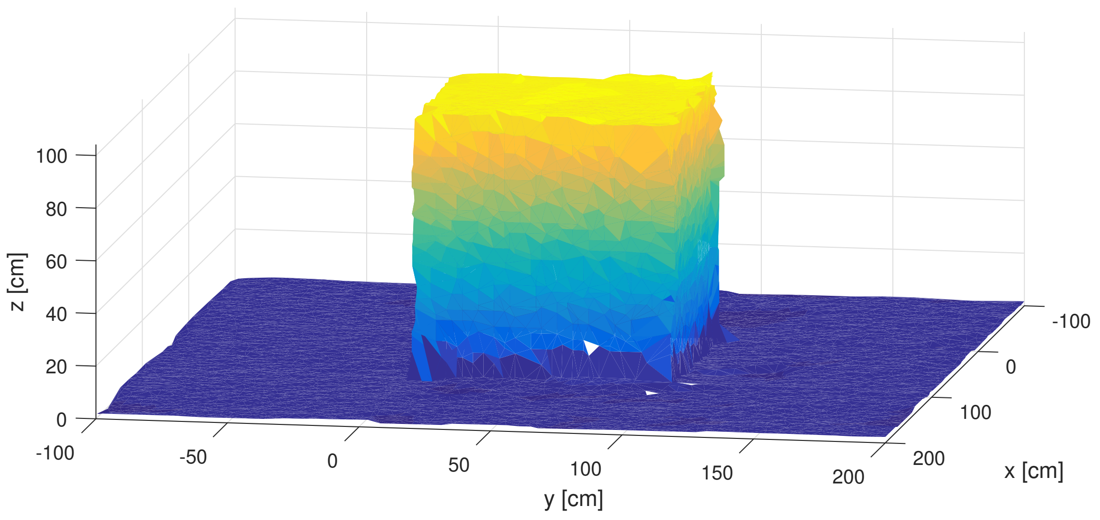

Figure 9.9: Triangular mesh of 'cardboard box' dataset

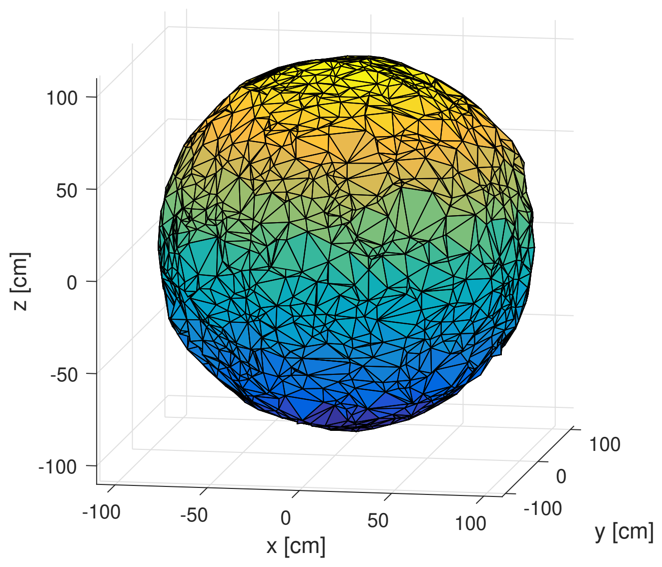

Figure 9.10: Triangular mesh of 'sphere' dataset

When looking at all these results in detail, there can be seen some missing or overlapping triangles. It is caused by computational inaccuracies that have been discussed in Section 5.3.

9.3 Ground plane detection

The algorithm for ground plane detection is described in Chapter 6. We fitted a first-degree polynomial (plane) into all points in the dataset. The used parameters are equal to filtering parameters. We only increased parameter T up to $T = 1000$, because the ground plane detection is not as computationally and time-demanding as the filtering and increasing parameter ensures us a better robustness. The results of ground plane detection in 'bench and fountain' and 'cardboard box' datasets are depicted in Figures 9.11 and 9.12. It takes just few seconds to detect the ground plane.

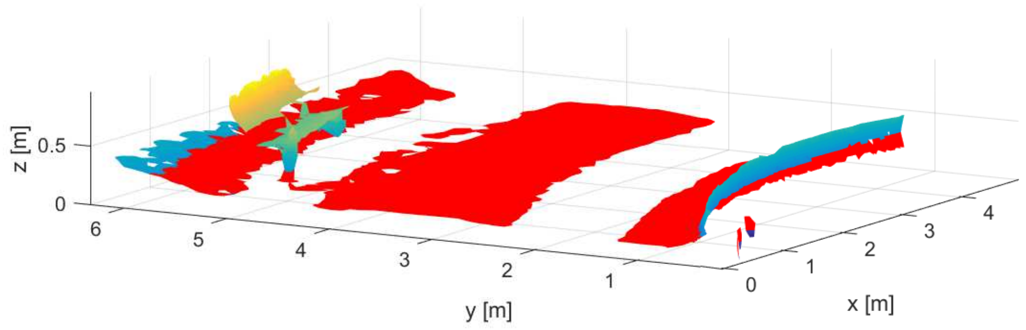

Figure 9.11: Ground detection of the 'bench and fountain' dataset

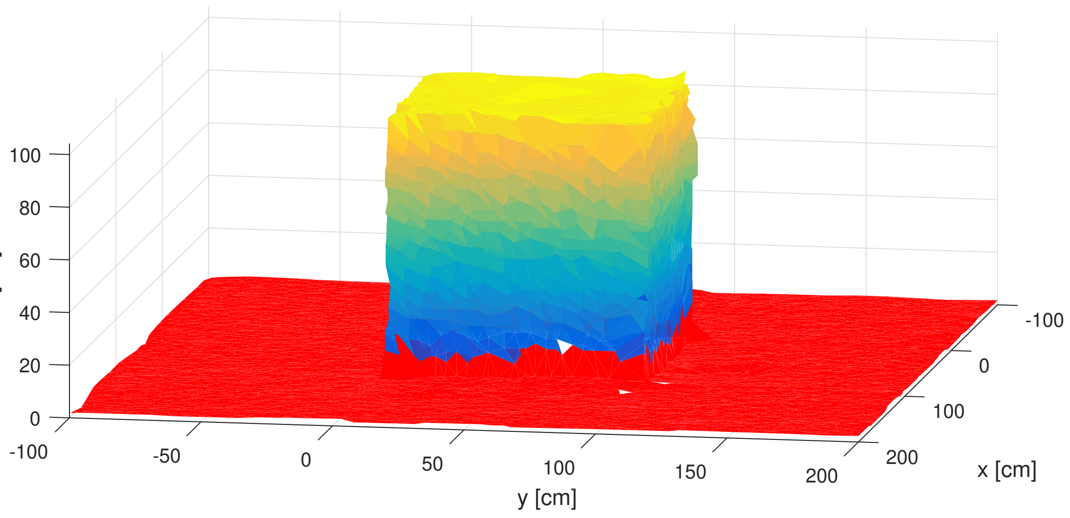

Figure 9.12: Ground detection of the 'cardboard box' dataset

9.4 Object volume calculation

The success of the volume calculation is influenced by previous three steps of the filtering, meshing and ground plane detection. The volume calculation algorithm was examined in Chapter 7.

As stated in Section 7.3, we need a closed object to be able to compute its volume. Currently available datasets of real measurements don't contain any significant object with the known volume. Therefore, we wanted to perform a real measurement with a cardboard box with known dimensions to test the algorithm on a real dataset. Unfortunately, a part of the hardware of our measuring rover system (described in Chapter 8) was inoperative. The LiDAR sensor had issues with fault timestamps and had to be send to the manufacturer to be repaired. Therefore the testing of volume calculation was performed on the simulated datasets.

This so-called 'cardboard box' dataset was filtered, meshed and the ground plane was detected (as seen in above). To obtain a closed object, it is necessary to compute the triangular mesh of the missing bottom part of the box which is not visible during the scanning process. This bottom part can be approximated using the detected ground plane (described in Section 7.3). However, this part wasn't implemented yet. For that reason, we used another simulation without the missing bottom part. It was filtered the same way as the previous one, and the triangular mesh was also created. Then we could calculate the volume of the box. The examples of meshing of the box and sphere and their computed volumes are depicted in Figures 9.13 and 9.14.

The problem of falsely generated triangles (discussed in Section 5.3) can cause errors in sorting of triangle vertices and differences in computed volumes. Therefore we tested each dataset in different configurations. Firstly we computed the volume of triangular mesh that hasn't been checked neither for 'false' or 'empty' triangles. Then we checked the mesh for 'false' and 'empty' triangles and finally we tried to remove all triangles that are suspicious for being false triangles (to ensure that all triangles are correct).

The results of testing of 'cardboard box' is shown in Table 9.3 and the testing of 'sphere' dataset is depicted in Table 9.4.

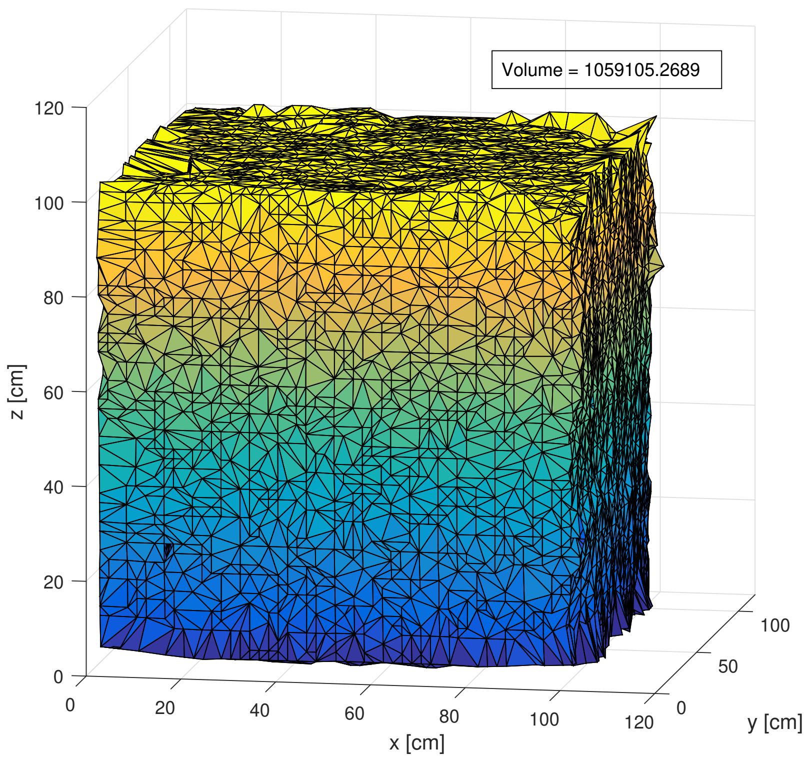

Figure 9.13: Object volume calculation using the 'cardboard box' dataset

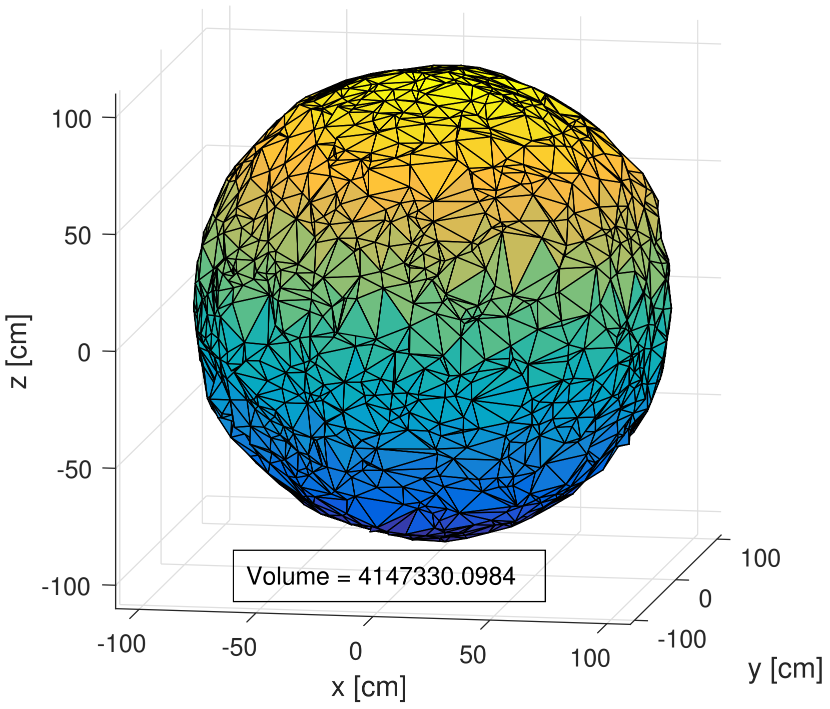

Figure 9.14: Object volume calculation using the 'sphere' dataset

We can estimate the true volumes and number of triangles of both objects. The true volume of 'cardboard box' dataset is $V = 10^6\ cm^3$ and the mesh should consist approximately of 10522 triangles (see Section 7.3). The volume of sphere is defined as $V = \frac{4}{3}\pi r^3 = \frac{4}{3}\pi 100^3 = 4188790$ cm^3 and the sphere surface should contain approximately 2662 triangles.

False △ check	**Number of △**	**Volume**	**Difference**	**%**
none	11127	1059105 cm^3	59105 cm^3	$5,9$ %
'empty'	11133	1060150 cm^3	60150 cm^3	$6,0$ %
'false'	10851	1030849 cm^3	30849 cm^3	$3,0$ %
'false' + 'empty'	10855	1031362 cm^3	31362 cm^3	$3,1$ %
remove all suspicious	10550	1001199 cm^3	1199 cm^3	$1,2$ %

Table 9.3: Testing of box volume calculation

False △ check	**Number of △**	**Volume**	**Difference**	**%**
none	2681	4154501 cm^3	34289 cm^3	$0,8$ %
'empty'	2690	4163056 cm^3	25731 cm^3	$0,6$ %
'false'	2668	4136695 cm^3	52095 cm^3	$1,2$ %
'false' + 'empty'	2677	4179051 cm^3	9739 cm^3	$0,2$ %
remove all suspicious	2626	4069415 cm^3	119375 cm^3	$2,8$ %

Table 9.4: Testing of sphere volume calculation

As we can see, the volume calculation of the 'sphere' dataset is more accurate than volume calculation of the 'cardboard box' dataset. We obtained the best result when we checked for 'false' and also for 'empty' triangles. The accuracy was $0,2$ %. This result is very close to the true volume of this sphere. Also the number of triangles in the mesh was similar to the estimated quantity.

The 'cardboard box' dataset contains a lot of overlapping triangles. The volume calculations differ from the true volume up to 6 %. This is caused especially by sharp edges and corners. Triangles near sharp features are often falsely generated and overlapping. Our algorithm for 'false' triangles checking wasn't able to fully cover all these overlapping triangles. Therefore the computed volume is bigger than the expected true volume.

9.5 Large point clouds

At the end we tested the filtering, meshing and ground plane detection algorithms on a large dataset captured at Charles square. This point cloud consist of $2,7$ millions of points. The filtering process took approximately 20 hours and we reduced the size of the point cloud to 300 thousands of points. The meshing algorithm took about 14 hours and it generated nearly 600 thousands of triangles. We also detected the ground plane that took just few seconds. The results can be seen in Figures 9.15 and 9.16.

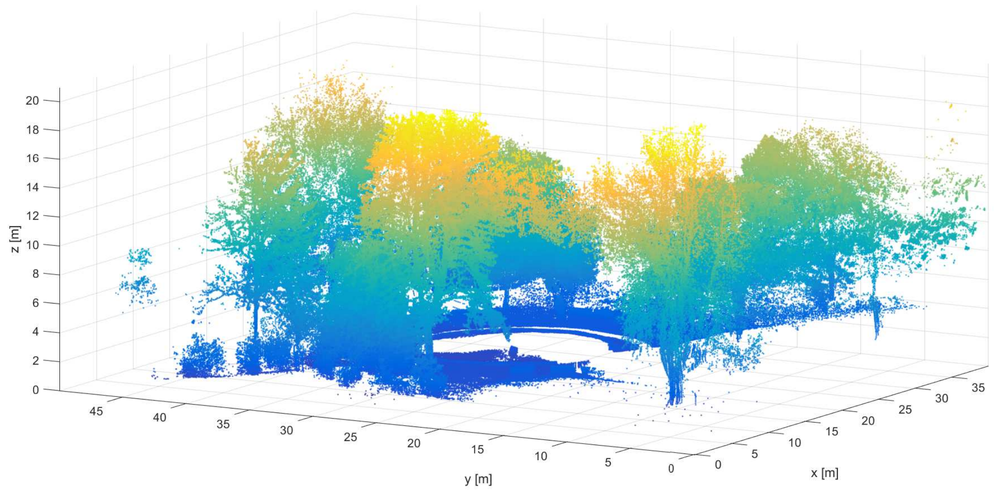

Figure 9.15: Raw large point cloud captured at Charles square

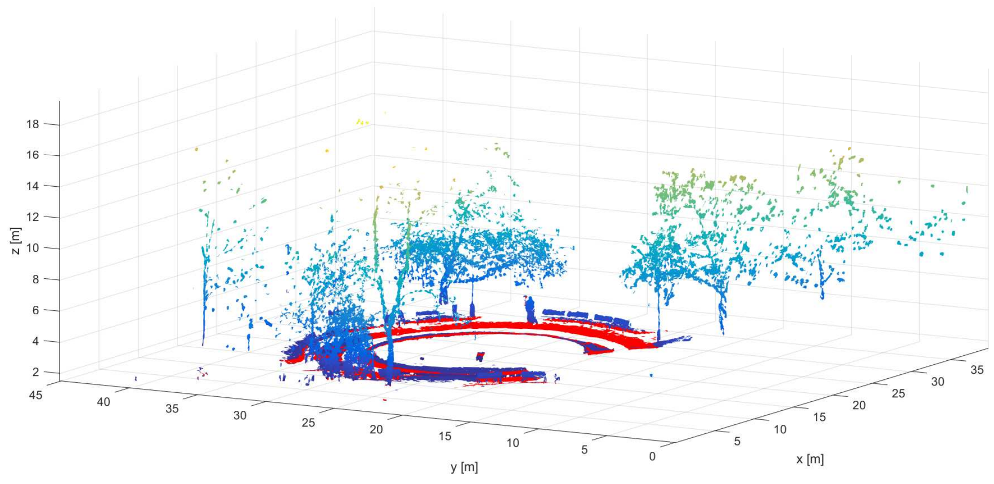

Figure 9.16: Triangular mesh of the large point cloud with ground plane detection

We can see that the size of the point cloud was significantly reduced. The filtering and meshing algorithm treated the tree crowns mostly as noise and outliers. The important features such as benches, fountain, lamps or people were filtered and meshed in detail.

The speed of the filtering and meshing algorithms should be further optimized. But it should be noted that that the point cloud was processed on a laptop with Intel Core i5-3210M processor and 8 GB RAM memory. Processing of point cloud on a high-performance computer would take less time.

10 Conclusion

In this work we focused on the theory of object volume calculation in large noisy point clouds. To achieve this goal we had to follow several steps.

In the first step it was necessary to implement filtering algorithm. We implemented the robust moving least square (RMLS) algorithm with some additional changes and improvements. It was crucial to solve the filtering of noise, outliers, sharp features (edges, corners) and reduce the size of the point cloud.

In the second step we studied and applied the theory of meshing and Delaunay triangulation. We had to design meshing algorithm for the special 2.5D case of non-organized point clouds. A special attention was paid to the problem of falsely generated triangles (missing or overlapping triangles), that are caused mostly by the computational inaccuracies.

The third step was dedicated to the ground plane detection. For this purpose we implemented the random sample consensus (RANSAC) algorithm. We extended this algorithm to be able to detect also generally non-linear ground surfaces (higher degree polynomial surfaces).

In the last step we achieved the main goal of object volume calculation. We applied the algorithm of calculation of signed volumes of tetrahedrons. This algorithm can be applied only on triangular meshes of closed objects and therefore it was necessary to implement all previous steps. To make this algorithm work correctly we had to implement an algorithm for sorting triangle's vertices to make normal vectors of all triangles facing the same way (neither inside or outside the object).

All implementation steps were tested in detail on simulated datasets and also on the real dataset, which was obtained using the system of LiDAR, GNSS and IMU sensors. The volume calculation algorithm was tested only on simulated datasets, because we didn't have any real environment measurements containing significant object with known volume.

We tested the object volume calculation algorithm using a sphere and cube object with predefined dimensions. The accuracy of calculated volume of sphere was $0,2$ % and the accuracy of calculated volume of cube was $1,2$ %. The accuracy of cube's volume was significantly worse, because the mesh contained a lot more overlapping triangles. This is an issue that needs to be solved in a future.

At the end of this thesis we tested the filtering, meshing and ground plane detection algorithms also on a large dataset obtained by capturing the real environment at Charles square in Prague. This point cloud contains about $2,7$ millions of points. The filtering reduced the size of the point cloud to 10 %. It treated the treetops as noise and outliers and therefore in the resulting point cloud remained only tree-trunks or thick branches. Other features (benches, fountain, lamps or even people) were kept in their original shapes. It took nearly 35 hours to process all algorithms. Therefore it is necessary to optimize all algorithms and also process them on a high-performance computer.

All the algorithms were successfully implemented with minor issues, which need to addressed in the future. We need to improve filtering algorithm to be able to detect and filter sharp features with multiple intersecting planes. In this thesis we implemented only the case of two and three planes. Then it is crucial to deal with falsely generated triangles in meshes. Some triangles in the triangular net are overlapping or missing (leading to small gaps in the mesh). In the implementation of volume calculation is still missing a very important part of the object closure. We can detect the ground plane, but to be able to calculate the volume, it is necessary to close the given object (add triangular mesh to the missing bottom side).

The advanced task for the future work is to update all used algorithms to the autonomous level. It would be interesting task to implement an algorithm for the estimation of all used parameters (e.g., range of one neighbourhood, outliers threshold, polynomial degree, etc.) or make these parameters even adaptive (dynamically changing during the algorithm process). The work also needs a lot more testing, especially on the real measured data sets. The real environments are more complex and difficult to process.

Bibliography

[1] *3DM-GX4-45 Inertial Navigation System User Manual*, 2014.

[2] About point clouds and lidar data. `http://help.autodesk.com/view/MAP/2016/ENU/?guid=GUID-7C7DD8A7-B561-45B0-A803-852E0A667F3C`, 2016. [Online; accessed 4-April-2018].

[3] Czepos. `http://czepos.cuzk.cz/`, 2018. [Online; accessed 11-May-2018].

[4] Delaunay triangulation - mathworks documentation. `https://www.mathworks.com/help/matlab/math/delaunay-triangulation`, 2018. [Online; accessed 17-April-2018].

[5] Delaunay triangulation - wikipedia. `https://en.wikipedia.org/wiki/Delaunay_triangulation`, 2018. [Online; accessed 17-April-2018].

[6] Optical encoders and lidar scanning. `http://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244`, 2018. [Online; accessed 9-May-2018].

[7] Point cloud - wikipedia. `https://en.wikipedia.org/wiki/Point_cloud`, 2018. [Online; accessed 4-April-2018].

[8] What is lidar data? `http://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm`, 2018. [Online; accessed 9-May-2018].

[9] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9:3–15, 2003.

[10] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, 6:793–800, 1934.

[11] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[12] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. Technical report, University of Utah, Tel-Aviv University, 2006.

[13] L. GUIBAS and J. STOLFI. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4:74–124, 1985.

[14] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao. A review of algorithms for filtering the 3d point cloud. *Signal Processing: Image Communication*, 10:103–112, 2017.

[15] J. Předota. Lidar based obstacle detection and collision avoidance in outdoor environment. Bachelor's thesis, CTU in Prague, 2016.

[16] M. Rouse. What is point cloud? `https://whatis.techtarget.com/definition/point-cloud`, 2016. [Online; accessed 4-April-2018].

[17] R. B. Rusu, N. Blodow, Z. Marton, A. Soos, and M. Beetz. Towards 3d object maps for autonomous household robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3191–3198, 2007.

[18] M. Schwind. Comparing lidar and photogrammetric point clouds. `https://www.gim-international.com/content/article/comparing-lidar-and-photogrammetric-point-clouds`, 2018. [Online; accessed 7-May-2018].

[19] T. Trafina. Construction of 3d point clouds using lidar technology. Bachelor's thesis, CTU in Prague, 2016.

[20] Z. Tůmová. Přesná lokalizace malých bezpilotních prostředků s využitím gnss. Bachelor's thesis, CTU in Prague, 2016.

[21] M. Y. Yang and W. Förstner. Plane detection in point cloud data. Technical report, Department of Photogrammetry, Institute of Geodesy and Geoinformation University of Bonn, 2010.

[22] C. Zhang and T. Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *Proceedings 2001 International Conference on Image Processing*, volume 3, pages 935–938, 2001.