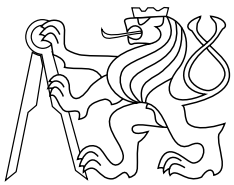
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY IN PRAGUE

MASTER THESIS

LIDAR Based Sequential Registration and Mapping for Autonomous Vehicles

Tomáš Sixta

sixta.tomas@gmail.com

May 26, 2017

Thesis Advisor: Ing. Martin Matoušek, Ph.D.

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Tomáš S i x t a

Study programme: Open Informatics

Specialisation: Computer Vision and Image Processing

Title of Diploma Thesis: LiDAR Based Sequential Registration and Mapping for Autonomous Vehicles

Guidelines:

The goal of this project is to develop and test a model that represents a possibly large amount of LiDAR (Light Detection and Ranging) data from a moving vehicle and uses the aggregated representation for a robust estimation of vehicle egomotion in the presence of normal street traffic via a 6D geometric registration between the aggregated model and a new data frame.

1. Propose a representation of 3D scene suitable for aggregation of LiDAR data - the map.
2. Develop an algorithm for robust registration of LiDAR measurements with the map.
3. Use the registration for ego-motion estimation of vehicle equipped with multiple LiDARs.
4. Develop an algorithm for updating the map with new (registered) measurements.
5. Test the methods on a real dataset from test drive.

Bibliography/Sources:

[1] Hartley, R. and Zisserman, A.: Multiple View Geometry. Cambridge University Press, 2nd ed., 2003.
[2] Thrun, S. and Bücken, A.: Integrating grid-based and topological maps for mobile robot navigation. Proceedings of the Thirteenth National Conference on Artificial Intelligence. 944-950, 1996.
[3] Aulinas, J.: The SLAM Problem: A Survey. Proceedings of the 2008 Conference on Artificial Intelligence Research and Development. 363-371, 2008.

Diploma Thesis Supervisor: Ing. Martin Matoušek, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 22, 2016

Author's declaration

I hereby declare that I have completed this thesis independently and that I have listed all used information sources in accordance with the Methodical guidelines on maintaining ethical principles during the preparation of university theses.

Prague, day ....
Signature

Acknowledgements

I would like to thank to Martin Matoušek for his guidance and mentoring, without which this thesis could not be completed, and to UP-Drive project for providing the datasets.

Abstract

In this thesis, we investigate the problem of registering LIDAR point clouds to a common representation of a scene. This is necessary for autonomous vehicle navigation. A representation of a 3D scene suitable for aggregation of LIDAR data is proposed, based on discrete probability density distribution of points in space. An online algorithm for robust registration of point clouds is introduced and used for improving the accuracy of egomotion. The method is tested on two LIDAR datasets from real test drives.

Abstrakt

V této práci řešíme problém správné registrace mraků bodů z LIDARu do společné reprezentace scény, což je důležité pro navigaci autonomních vozidel. Přicházíme s novou reprezentací 3D scény vhodnou pro agregování dat z LIDARu, která je založena na diskrétní distribuci pravděpodobnosti bodů v prostoru. Představujeme online algoritmus pro robustní registraci mraků bodů za účelem zpřesnění egomotion. Poté je tato metoda otestována na dvou datasetech z reálných jízd.

Contents

1. Introduction

In the recent years, LIDARs have become a common part of experimental self-driving cars, enabling them to create a 3D map of the environment and supporting their means of navigation. To perform these tasks, a huge amount of points that LIDARs record must be correctly registered during a drive. Accurate egomotion data is necessary for the correct registration, but for various reasons sensors providing the needed level of accuracy might not be available. It is then necessary to use a special algorithm that improves or estimates the egomotion, which in turn ensures the correct registration of new points.

Typically, a car is moving and new point clouds are being recorded. These point clouds are used for localizing the car and improving the egomotion. After that, they are registered into the scene representation. This process, known as the simultaneous localization and mapping (SLAM), is essential for 3D reconstruction of the environment and for autonomous cars to be able to navigate in it.

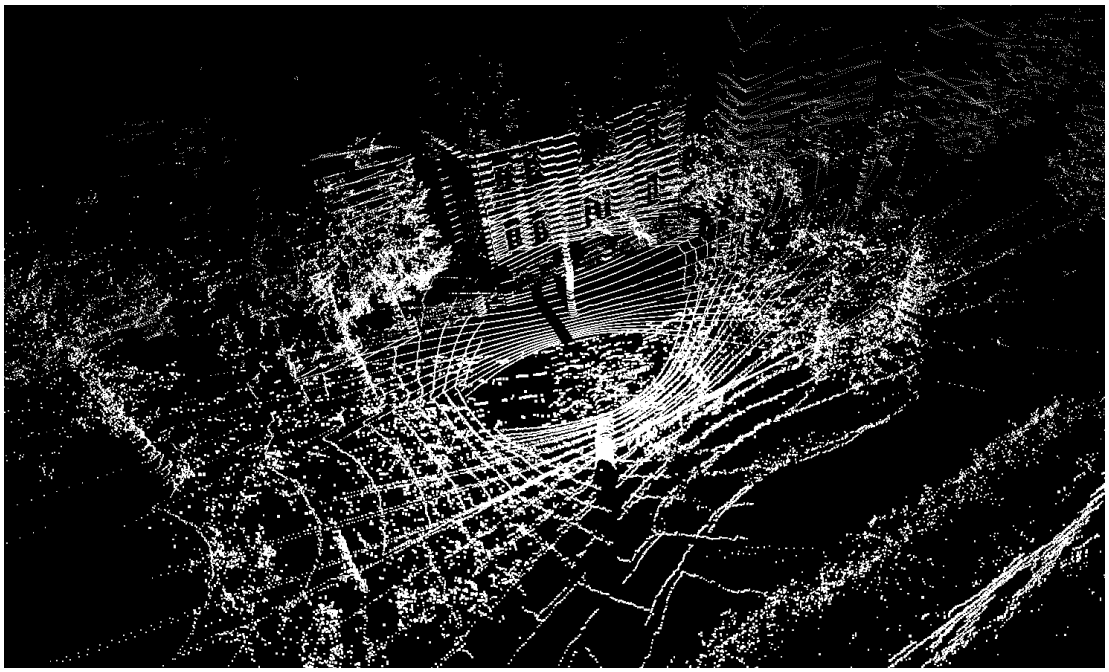


Figure 1.1. Example of a LIDAR point cloud (around 100 000 points) from a typical street

2. Related Work

2.1. Basic Approaches

One of the basic techniques for matching point clouds is ICP (Iterative Closest Point). This method tries to minimize the distance between two point clouds by estimating relative translation and rotation. Unfortunately, ICP-based methods encounter problems when the motion of the car is relatively high compared to the LIDAR scan rate. This problem is called motion distortion and means that points in a given point cloud do not necessarily correspond to the points in the consecutive point cloud. In that case it is meaningless to use ICP and try to minimize their distances.

Some ICP-based methods exist – they can be used when the LIDAR is moving slowly and motion distortion can be neglected, see for example [1]. Authors of [2, 3] try to compensate for motion distortion by various approaches and use modified ICP.

2.2. Visual and LIDAR Odometry

If the motion distortion occurs, more sophisticated approach must be taken. Visual odometry is one of the possibilities. It is based on using the input images to estimate the motion of the camera. Typically, the image is pre-processed and feature points are found, then these feature points are tracked across the images and motion flow is estimated. Some examples of visual odometry are [4, 5, 6].

With the spread of LIDAR technology, visual odometry has been enhanced by using 3D point clouds. Data from both cameras and LIDARs are used to estimate the egomotion, which improves the accuracy compared to purely visual odometry. Kalman filter is a popular technique used for example in [7, 8].

Authors of [9] use Kinect sensor instead of LIDAR, because it assigns depth to each image point. Feature points are detected in the image and 3D coordinates of those feature points are determined. They are then used for rotation and translation estimation between consecutive images.

However, relying on data from cameras might be problematic when the light conditions are worsened. That is why LIDAR-only approaches have been developed too.

In [12], authors rely on loop closure and matching of geometric structures to generate a map of an underground mine. In [13], authors use two-layer approach. One algorithm runs at high frequency, but with low fidelity and tries to estimate the velocity of the LIDAR. The other algorithm runs at lower frequency, but is used for finer registration of the point clouds. Feature points, such as those located on sharp edges or planar surfaces, are extracted and matched.

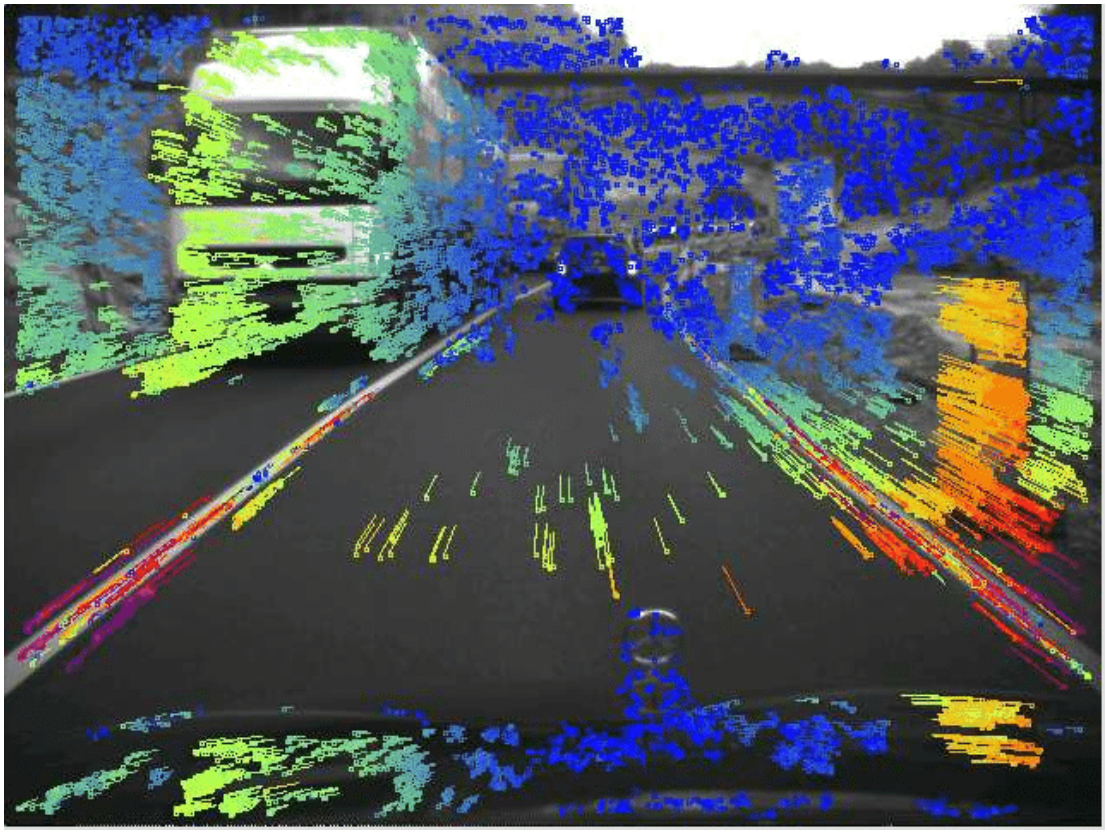


Figure 2.1. Optical flow example; courtesy of Fridtjof Stein, Computation of Optical Flow Using the Census Transform, 2004

Even though these state of the art methods have been successfully used, they come with some drawbacks.

- ICP-based methods are virtually unusable for fast-moving cars because of the motion distortion.
- Visual odometry is highly dependant on the light conditions – its performance significantly decreases for example in the dark, during heavy rain or fog.
- Approaches using LIDAR are the most robust, but also considerably complex. Unless a suitable scene representation is chosen, the memory requirements can be very high if every point is stored.

We propose a new algorithm based on efficient scene representation. Probability of each point cloud fitting into the scene representation is evaluated and this probability is maximized. This yields the improved egomotion, which is used for correct registration of the point cloud. Additionally, the scene representation naturally allows for easy detection of non-static objects in the environment, which further helps with the car navigation.

3. LIDAR Point Cloud Registration

3.1. Algorithm Overview

The proposed algorithm is based on the assumption that the point cloud that is to be registered comes from several LIDARs. The positions and rotations of these LIDARs with respect to the car coordinate system are known, the data they provide can be merged together into a single point cloud. This point cloud is then processed and registered.

The algorithm is being continuously given new data and processes them. Registering a single point cloud is done by the following steps:

- Retrieve data from a single sensor
  - Transform the data from the sensor coordinate system to the car coordinate system
  - Perform motion compensation
  - Transform the data from the car coordinate system to the world coordinate system
- Merge data from four sensors together into a single point cloud
  - Find the ground plane
  - Compute height of each point from the ground
- Find the point cloud transformation so that it can be registered into the scene with the highest probability
  - Use a robust optimization function to find the rotation and translation that maximize the probability of the point cloud fitting into the scene
- Apply the transformation
- Add the transformed point cloud to the scene representation – the map

3.2. The Map

Storing every single point might appear as a straightforward solution, but it is impractical due to high memory and processing requirements, as several hundreds of thousands of points are stored and processed every second. It actually isn't necessary for the task of improving or estimating the egomotion. Therefore an alternative scene representation – the map – is used instead.

The XY plane is divided into a grid of square cells and each of these cells contains a histogram of points' heights, as illustrated in Figure 3.1. It is a discrete representation of probability density function of points in space. With a properly chosen cell size and histogram bin size, this leads to considerable memory savings while still providing valuable information for egomotion estimation.

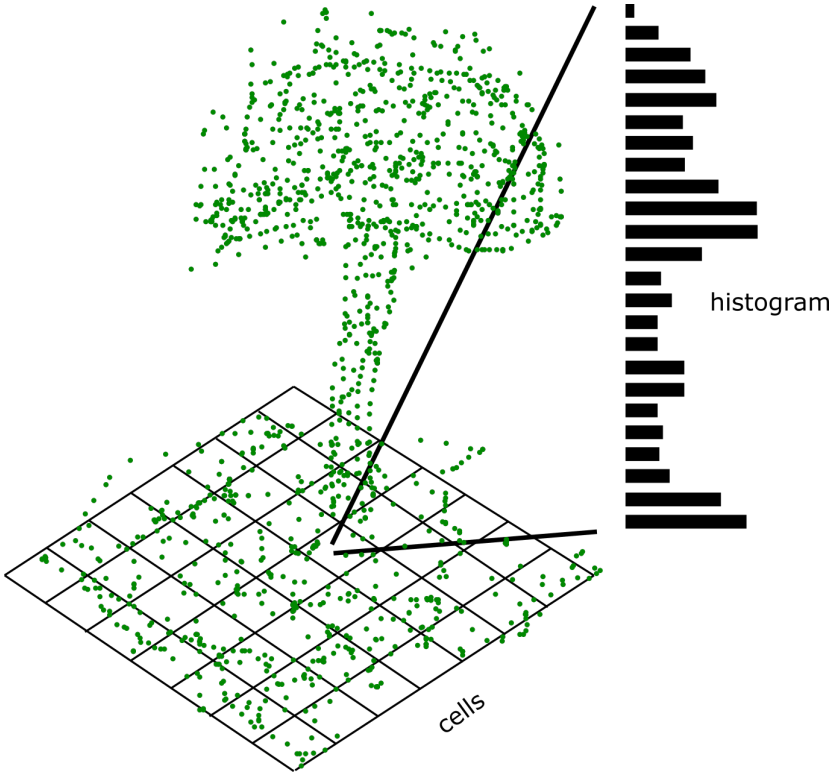


Figure 3.1. The map – example of representation of a scene with a tree

Preliminary experiments showed that if a point cloud is misplaced (registered with incorrect egomotion), it will have a low probability of fitting into the map. That's caused by the fact that points in that point cloud will be registered into the wrong cells and their respective heights into the wrong bins.

The probability of a point cloud fitting into the map can then be maximized in order to obtain the optimal transformation. This transformation is then used to update the egomotion and to transform the point cloud, then the point cloud is recorded into the map.

3.2.1. Non-static Scene Detection

Another use of this method is the detection of non-static objects in the scene. The idea behind the scene representation is based on whether each point belongs to its place or not. After the optimal transformation is found, all the points that still have a low probability can be marked as not belonging to the scene. This allows for simple detection of changes and new obstacles without needing to interpret the visual data.

3.3. Types of Point Displacement

There are two types of point displacement that can occur. Each needs to be dealt with by a different approach.

3.3.1. Within a Single Point Cloud

The LIDAR captures points much faster than the egomotion system provides the location data. While a unique timestamp is available for each point, the transformation from the car coordinates to the world coordinates is not. When a point needs to be transformed, it uses the last available transformation instead. Point displacement occurs (Figure 3.2), because as the car is moving, the point is recorded at a different position than the egomotion and thus is forced to use the incorrect transformation.

Because each point has a unique timestamp, the rotation matrix and the translation vector can be interpolated. This process is called motion compensation (addressed in Section 3.6) and it reduces this type of point displacement to a level where it is nearly unnoticeable.

3.3.2. Between Different Point Clouds

Points in different point clouds representing the same object can be registered incorrectly at different positions. This happens when the egomotion is inaccurate, as illustrated in Figure 3.3. For example, if the egomotion is generated from a wheel-based odometry, driving over a speed bump or breaking too fast can cause the car to tilt – this can't be detected by the odometry and can introduce a significant error to the point cloud registration. Moreover, the wheel-based odometry suffers from error accumulation.

This is more difficult to deal with and is the subject of this thesis.

3.4. Transforming Data between Different Coordinate Systems

In the context of this thesis, the point clouds after motion compensation are related by rigid motion and thus transformed by rotation R and translation t. Because all the point clouds are captured by sensors of the same type, there is no scale nor skew.

Transformation $f : A \to B$ of a point cloud from coordinate system A to coordinate system B can be expressed by

$$X_B = R_A \cdot X_A + t_A \ , \tag{3.1}$$

where X_A, X_B are points in their respective coordinate system, R_A is the rotation matrix and t_A is the translation vector of the transformation f.

3.5. Data Collection and Preprocessing

The experimental vehicle used in this thesis is equipped with four LIDARs in each of its roof corners. Data acquired from each of these sensors (point clouds X_s^i, $i = 1..4$) are stored in their respective coordinate system. The first thing that needs to be done is to transform the data into the car coordinate system.

Car specifications are known for the used datasets. These specify the location and rotation of each sensor, this information is given by a rotation matrix R_s^i and a translation vector t_s^i. Using R_s^i and t_s^i, the data are transformed into the car coordinate system by

$$X_c^i = R_s^i \cdot X_s^i + t_s^i, \qquad i = 1..4 \ . \tag{3.2}$$

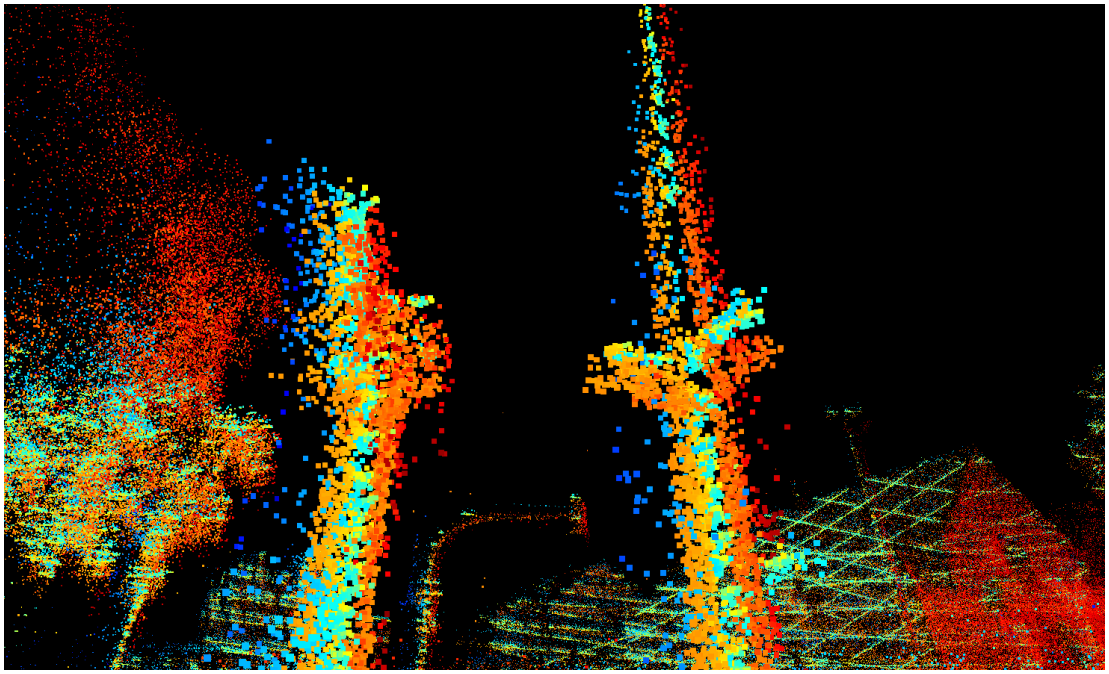


Figure 3.2. Point displacement within a single point cloud – points were transformed by the last available transformation that didn't match their timestamp, which caused the displacement

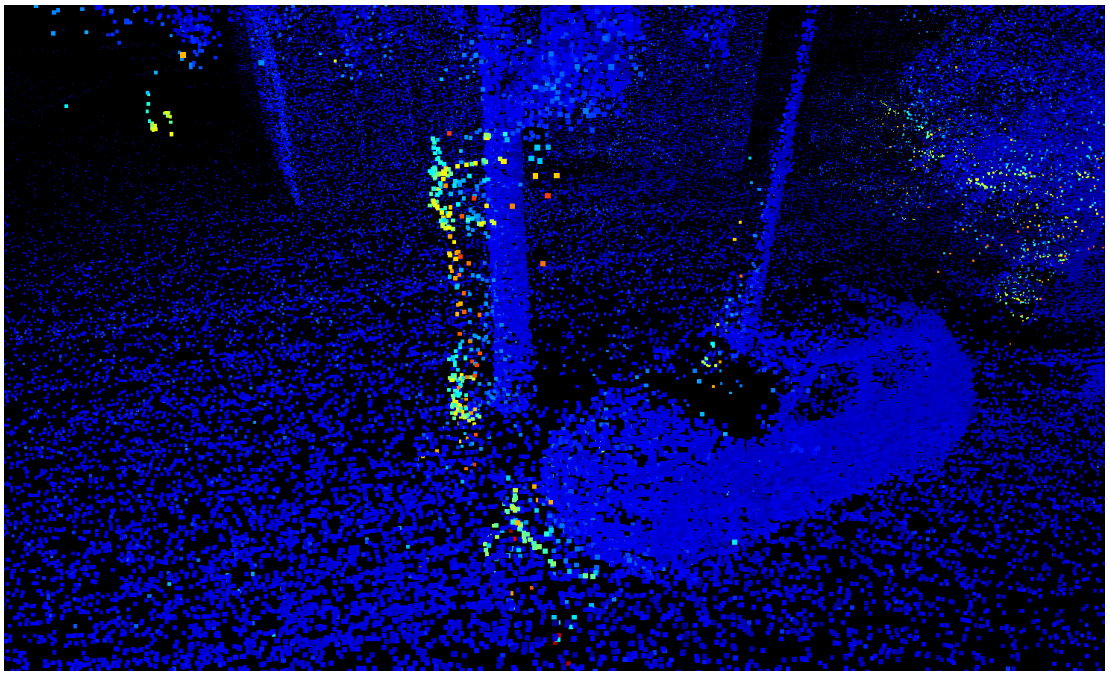


Figure 3.3. Point displacement between different point clouds – bright points belong to point clouds that were captured much later than the blue points; the accuracy of the egomotion had decreased in the meantime, which caused duplicate surfaces of the tree and the car

The preprocessing step consists of selecting points of special interest. This has to be done before the point cloud is transformed into the world coordinate system, because these points of interest have important location relative to the car. These points of interest are:

- Points on the car itself (these points can be removed, because they carry no useful information as they are always in the same position relative to the car)
- Ground plane candidates (these points are used for the ground detection)
- Points in a predefined neighbourhood (these are used for the histogram evaluation)

After the preprocessing is completed, the data are transformed into the world coordinate system and the motion compensation is performed.

3.6. Motion Compensation

The egomotion gives a rotation matrix R_c and a translation vector t_c. These provide position of the car in space and need to be used for transforming the points from the car coordinates to the world coordinates. The egomotion is available with much less frequency than the points are scanned, so it is interpolated to match the timestamp of those points.

Because interpolating R_c and t_c for every single point is very expensive, the points are divided into n groups and R_c and t_c are interpolated for these groups only. With a reasonable n, these groups will differ by only a fraction of millisecond while still drastically reducing the number of interpolations. See Algorithm 1 for details.

Algorithm 1 Motion Compensation

Require: X, ts, n – data in car coordinates, timestamp, number of groups
Ensure: Returns X – point cloud in world coordinates after the motion compensation

$idx \leftarrow X.point\_relative\_time$ ▷ interpolate timestamp for each group
$times \leftarrow$linspace(min(idx),max(idx), n)

$idx \leftarrow idx - \min(idx)$ ▷ assign points to the right group
$idx \leftarrow$ceil($\frac{idx}{\max(idx)} \cdot n$)

for i = 1:n **do** ▷ interpolate rotation and translation and update points
    $[R_i, t_i] =$interpolate($ts + times(i)$)
    $X(:, idx == i) = R_i \cdot X(:, idx == i) + t_i$

3.7. Finding Ground Plane

Once the point cloud is assembled, the ground plane is found. Heights of points from the ground plane are then registered into the scene representation.

3.7.1. Computing Heights of Points above Ground

Height of a point above the ground is its distance from the ground plane. Consider point x and ground plane $G = (n, g_0)$ given by its unit normal vector n and point g_0.

Let w be vector $w = g_0 - x$. Then projecting w onto n gives the distance $|d_G(x)|$ of point x from plane G [14],

$$|d_G(x)| = |\text{proj}_n(w)| \tag{3.3}$$

$$= \frac{|n \cdot w|}{|n|} \tag{3.4}$$

$$= |n \cdot w| \ , \tag{3.5}$$

where $|n|$ can be omitted from the denominator because it is a unit vector. Dropping the absolute value gives the signed distance

$$d_G(x) = n \cdot w \ . \tag{3.6}$$

The final formula for the signed distance $d_G(x)$ of x from G is then

$$d_G(x) = n \cdot (g_0 - x) \ . \tag{3.7}$$

To meaningfully measure how good the ground plane is, the objective function needs to be defined. This objective function is based on these two requirements:

- Angle between the ground plane's normal and the Z axis is less than a
- Each point under the ground plane is penalized by value p

The reasons for these requirements are simple. The angle requirement ensures that another dominant plane in the scene (e.g. a building) won't be considered ground. Because breaking / accelerating or driving over a speed bump tilts the car by only several degrees, this restriction discards most unwanted planes while keeping only those horizontal / nearly horizontal.

The other requirement is based on the assumption that no points under the ground can be detected (if they are, they are most likely outliers). This means that the found plane won't be e.g. a flat roof of a building.

The last thing necessary for the objective function is some threshold t. Points closer than the threshold are considered inliers, other points are considered outliers.

The objective function that computes plane's support s is based on the maximum likelihood estimate (MLE) [15] and is defined as

$$s = \begin{cases} -\infty, & \text{if } (n' \cdot [0;0;1] < a) \\ |I| - \frac{d_G(I)^2}{t^2} - |U| \cdot p \ , & \text{otherwise} \end{cases} \tag{3.8}$$

with point cloud X, ground plane $G = (n, g_0)$, distance function d_G as defined in Section 3.7.1, threshold t, angle restriction a, set of inliers $I = \{x \mid 0 \leq d(x) < t\}$, set

of underground points $U = \{x \mid d(x) < 0\}$ and penalty p for a point being under the ground.

First line of Equation 3.8 satisfies the angle restriction. On the other line, the number of inliers is computed, MLE criterion is applied and finally the penalty is applied for the number of underground points.

With the objective function defined, there are two possible scenarios.

3.7.2. Adjusting Plane from the Previous Point Cloud

If a plane in the previous point cloud is known, it can be used as a starting point for some optimization function. The support function as introduced in Equation 3.8 is maximized for the current point cloud. As long as the previous plane and the new plane have a reasonably similar support, this approach is considered sufficient and the new plane is used.

3.7.3. Finding New Plane

If a plane in the previous point cloud is not available or has too different support compared to the new plane, another approach needs to be taken. MLESAC [15] with the objective function as defined in Equation 3.8 is used. Only those points that were marked as relevant for the ground detection during the preprocessing step (Section 3.5) are used.

After the new plane is found, it is used as a starting point for some optimization function that maximizes the support (just like in Section 3.7.2). This adjusts the plane and provides slightly higher support than MLESAC could on its own. Only MLESAC inliers are used for this step.

With the new ground plane, height for each point is calculated.

3.8. Fitting Point Cloud into the Map

Consider a point x with height x_h. The relevant cell c for point x is determined based on position of x on the XY plane,

$$c_{coords} = \text{round}(\frac{[x_x, x_y] - M_{offset}}{M_{cell\_size}}) \ , \tag{3.9}$$

where M_{cell_size} is the size of the cell and M_{offset} is the offset of the map (data in the world coordinates are not placed near the origin [0; 0; 0], so M_{offset} compensates for that). This cell contains histogram H. Bin i in histogram H to which x_h belongs is determined as

$$i = \text{round}(\frac{x_h - H_{min}}{H_{bin\_size}}) \ , \tag{3.10}$$

where H_{min} is the minimal height that can be registered into the histogram and H_{bin_size} is the size of the bin (vertical step).

The probability $p(x_h)$ of height x_h belonging to H is

$$p(x_h) = \frac{H(i)}{\sum H} \ . \tag{3.11}$$

This probability also represents the probability that x under its current transformation belongs to the map, $p(x_h) = p(x)$.

It can happen that $p(x) = 0$, either due to H being empty or $H(i) = 0$. Having zero probability for some point is very likely to happen, considering one point cloud consists of tens of thousands of points. This would negatively affect the outcome of the algorithm, so a threshold value p_{min} is used to counter this. If probability $p(x)$ of some point is lower than this threshold, the probability is assigned the threshold value,

$$\text{if } (p(x) < p_{min}) \text{ then } p(x) \leftarrow p_{min} \ . \tag{3.12}$$

The probability $p(X)$ of the whole point cloud belonging to the map is the product of probabilities of individual points. For simplicity, the independence of points is assumed,

$$p(X) = \prod_{x \in X} p(x) \ . \tag{3.13}$$

For the sake of numerical stability, logarithm of the probability $\log(p(X))$ is used. The log-probabilities of the points are summed together, which gives the log-probability of the whole point cloud fitting into the map. This log-probability is what needs to be maximized to determine the optimal point cloud position and rotation with respect to the current map.

The final log-probability $p_{log}(X)$ of the point cloud fitting into the map is

$$p_{\log}(X) = \sum_{x \in X} \log(p(x)) \ . \tag{3.14}$$

3.9. Optimization and Point Cloud Update

Some robust optimization function is used to find the parameters that maximize the log-probability $p_{log}(X)$. These parameters are the rotation matrix R and the translation vector t that refine the egomotion and that need to be applied to the point cloud to register it with the highest possible probability.

At the beginning of the algorithm, R will be very similar to the identity matrix I and t to the zero vector $\mathbf{0}$. But as the car keeps moving, the error caused by inaccuracy in the egomotion accumulates and R diverges from I and t from $\mathbf{0}$ (that also happens if the egomotion is not available at all). Therefore some suitable initial parameters for the optimization function other than I and $\mathbf{0}$ need to be chosen.

The initial parameters used are the last known R and t. This is based on the assumption that if the last position reported by the egomotion system differs by t_{prev} from its correct value, the current egomotion will present a similar error. Same reasoning applies for R.

Once R and t are found, they are used to transform the whole point cloud,

$$X' = R \cdot X + t \ . \tag{3.15}$$

This updated point cloud then fits into the scene representation with the maximal possible probability and can be added.

3.10. Adding New Point Cloud to the Map

The final step is to add the updated point cloud to the map. For each point, the corresponding cell and bin is determined, just like in Section 3.8.

A simple solution would be to update the corresponding histogram H by $H(i) \leftarrow H(i)+1$. However, this would result in many sharp peaks in the histogram. Preliminary experiments showed that this could prevent the optimization function from converging to the correct value and get stuck in a local maxima instead. Therefore the histogram convolution is performed to make it smoother and to remove sharp peaks.

3.10.1. Histogram Convolution

Contributing to the histogram bin i is the same as adding vector $\underline{1}(i)$ to the histogram,

$$H(i) \leftarrow H(i) + 1 = H \leftarrow H + \underline{1}(i) \ . \tag{3.16}$$

The hard registration of x_h to a single bin is replaced by a softer alternative, where x_h contributes to its surrounding bins as well. This can be expressed as a convolution of vector $\underline{1}(i)$ with a Gaussian $\mathcal{N}(0, \sigma^2)$ before adding it to the histogram,

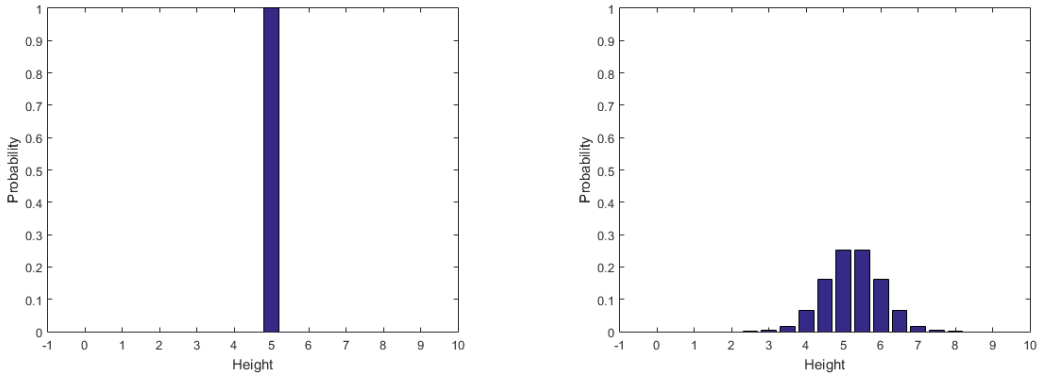
$$H \leftarrow H + (\underline{1}(i) \circledast \mathcal{N}(0, \sigma^2)) \ , \tag{3.17}$$

which is the same as

$$H \leftarrow H + \mathcal{N}(i, \sigma^2) \ , \tag{3.18}$$

where σ^2 is the variance of the Gaussian and i is used as the mean. Figure 3.4 shows the difference.

This will make the histogram and the objective function defined in Section 3.8 much smoother and easier to optimize.



(a) $\underline{1}(i)$ **(b)** $\mathcal{N}(i, \sigma^2) = \underline{1}(i) \circledast \mathcal{N}(0, \sigma^2)$

Figure 3.4. Two ways of updating the histogram H

3.10.2. Registering Points into the Surrounding Cells

Just like with the histogram, registering a point only into its cell makes it harder for the optimization function to converge. Therefore when a point x is about to be registered into cell c, Gaussian $\mathcal{N}(i, \sigma^2)$ is created as described in Section 3.10.1.

The Gaussian $\mathcal{N}(i, \sigma^2)$ is then added to histogram H in cell c as well as to histograms in other cells surrounding c. Before every addition, $\mathcal{N}(i, \sigma^2)$ is multiplied by some predefined coefficient that loosely models 2D Gaussian distribution around c.

Similarly to the histogram convolution, this improves the smoothness of the objective function and should make it easier for the optimization function to converge to the correct value.

When all points are recorded into the map, a new data is read and the new point cloud is processed in the same way.

4. Experiments

4.1. Hardware and Environment

Datasets used in this thesis were acquired by Volkswagen Golf E demonstration vehicle of H2020 EU project UP-Drive [16] with four LIDAR sensors, each in one of the car's roof corners. Table 4.1 provides basic specifications of the LIDAR sensors.

sensor type	Velodyne VLP-16
range	100 m
accuracy	3 cm
field of view – horizontal	360°
– vertical	± 15°
resolution – horizontal	0.1° - 0.4°
– vertical	2°
rotations per second	5 - 20
points per second	up to 300 000

Table 4.1. Basic specifications of the used LIDAR

In these datasets, 10 rotations per second is used, meaning a new point cloud is produced every 100 milliseconds. More details and the exact settings can be found in the attached configuration file.

The algorithm was implemented in MATLAB. Numerical optimizations were done using MATLAB's function *fminsearch()* that uses Nelder-Mead method [17]. For implementation pseudocode see Appendinx A.

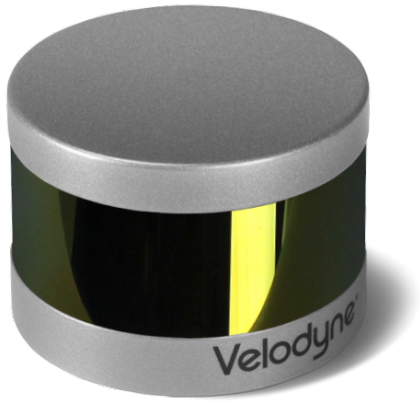


Figure 4.1. Velodyne VLP-16 LIDAR

4.2. Visualization and Datasets

Two different visualization modes are used in these experiments. In the first mode, the point's color codes the time when the point was captured – the older points are more blue, the newer points are more red. Unless specified otherwise, point clouds will be coloured using this time-based mode. The other mode is based on heights of the points – points that are low are more blue, points that are high are more red. See Figure 4.2 for illustration. To save memory, only 10 % randomly selected points are shown.

The experiments were conducted on two datasets. The first dataset (Figure 4.3) is a series of point clouds recorded by a car that is moving forward on an almost straight street with a crossroad in the middle. It consists of approximately 400 point clouds collected over 40 seconds. Point displacement caused by inaccurate egomotion is rather low.

The other dataset (Figure 4.4) is a series of point clouds recorded by a car on a parking lot. In this dataset, the car makes several loops. It consists of over 1 000 point clouds collected over 100 seconds. There is a major point displacement caused by inaccurate egomotion, leading to numerous duplicate surfaces.

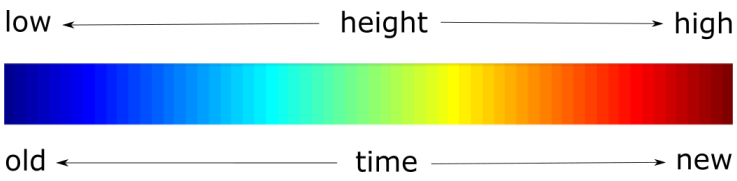


Figure 4.2. Visualization modes used for the experiments are based on height and time

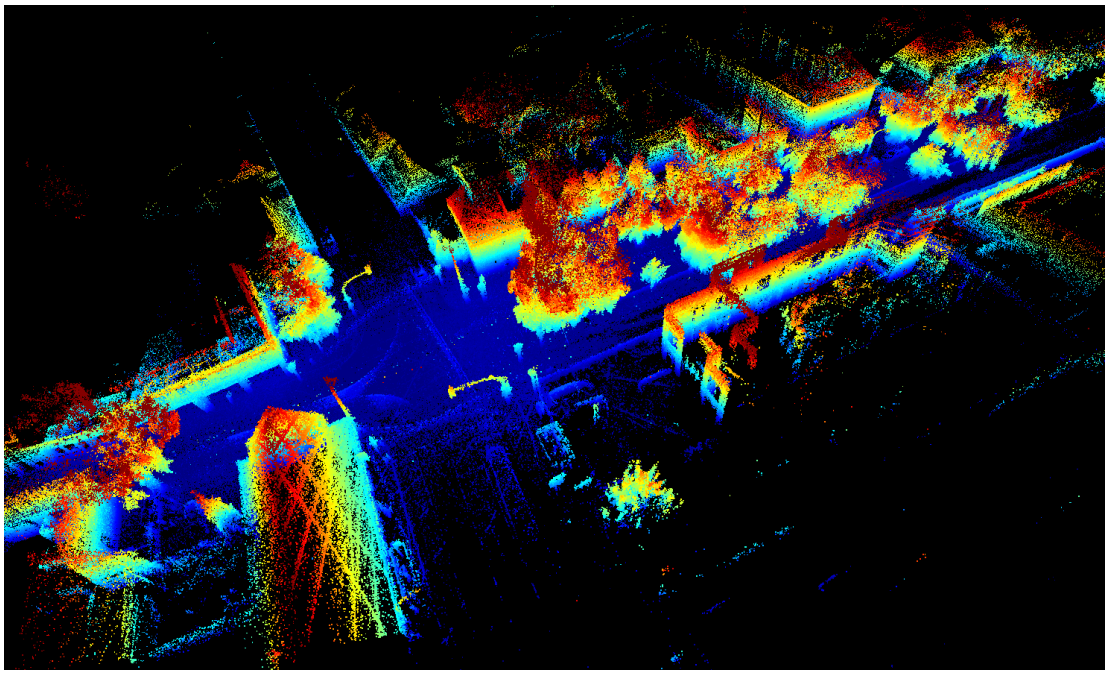


Figure 4.3. Dataset 1 – straight street with a crossroad; coloured based on points' height

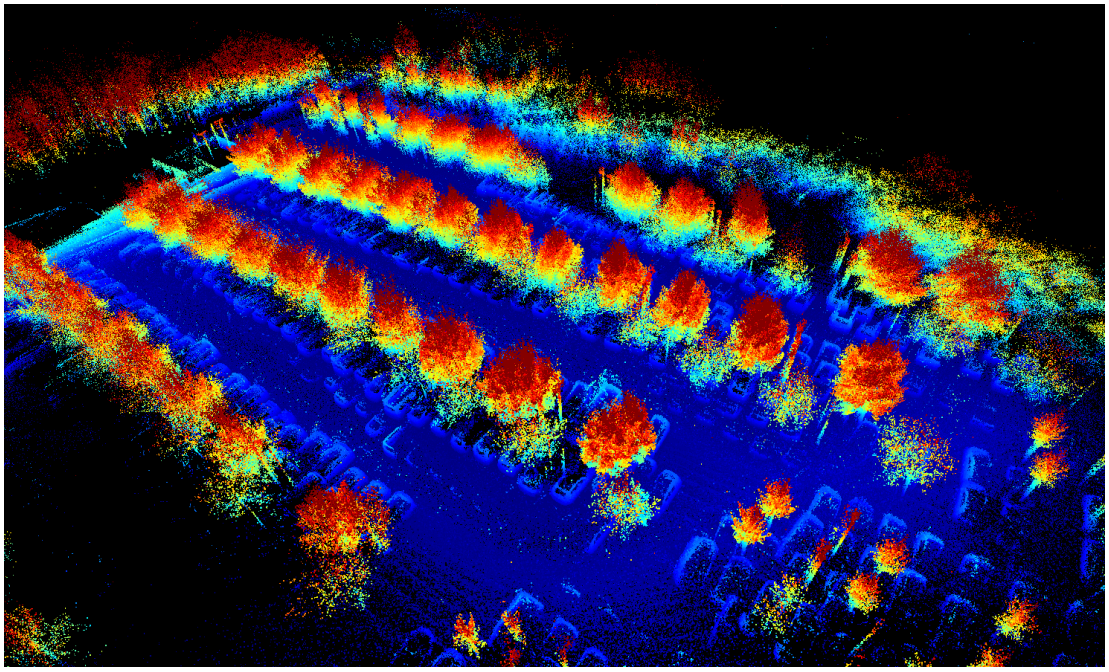


Figure 4.4. Dataset 2 – parking lot; major point cloud displacement is apparent (duplicate cars and trees); coloured based on points' height

4.3. Experimental Protocol

The experiments are divided into several sections, each investigating a specific part of the proposed algorithm:

- Setting of initial values
- Motion compensation
- Whether the ground plane can be consistently found across different point clouds and if it is useful to store heights of points in the scene representation
- If the proposed scene representation carries useful information for egomotion estimation and how well does the optimization function converge
- If the histogram convolution and registering points into the surrounding cells improves the convergence of the optimization function
- Overall results

Unfortunately, when evaluating the overall result, there is no clear way of determining its quality. The objective function that is optimized gives some probability, but this probability is quite complex to be interpreted in a meaningful way. Moreover, there are problems regarding local maxima, where the optimization function can get stuck. Therefore, evaluation of the overall results is done on an empirical basis just by looking at how well do the point clouds match.

4.4. Initial Values

The values used in the experiments (unless stated otherwise) are summarized in Table 4.2.

preprocessing	ground plane candidates	$[20, -20, 10, -10, 1, -1.5]$
	predefined neighbourhood	$[5, -30, 15, -15, 20, -5]$
motion compensation	n groups	100
the map	map cell size	50 cm
	histogram – bin size	10 cm
	– minimum height	-1 m
	– maximum height	10 m
	probability threshold p_{min}	10^{-6}

Table 4.2. Initial values used for the experiments

The vector of values for the ground point candidates and predefined neighbourhood marks those points that are inside the given range relative to the car ([front, back, left, right, top, bottom], in metres). Other values, like RANSAC settings or Gaussian parameters for the convolution are discussed in their respective sections.

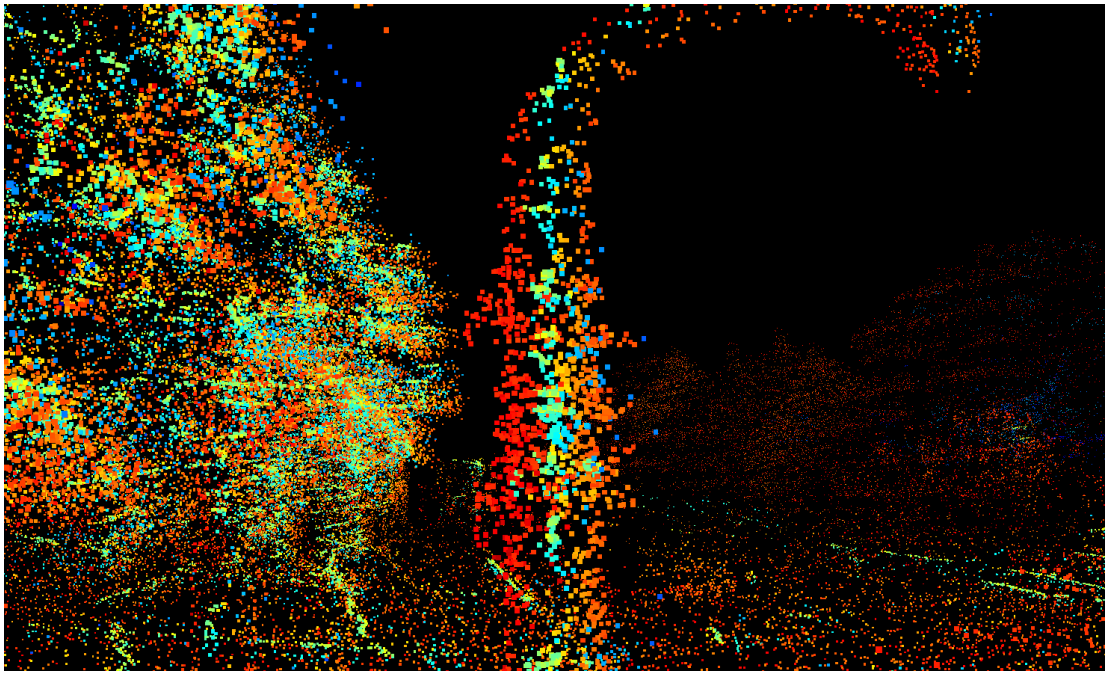
These values were selected empirically. Since the results are evaluated on the empirical basis, it would likely not be possible to refer to the best found values as optimal anyway. Some minor experiments were done to check some alternative values (for example cell size = 25 cm, bin size = 5 cm or $p_{min} = 10^{-3}$), but the differences were hardly noticeable.

4.5. Motion Compensation

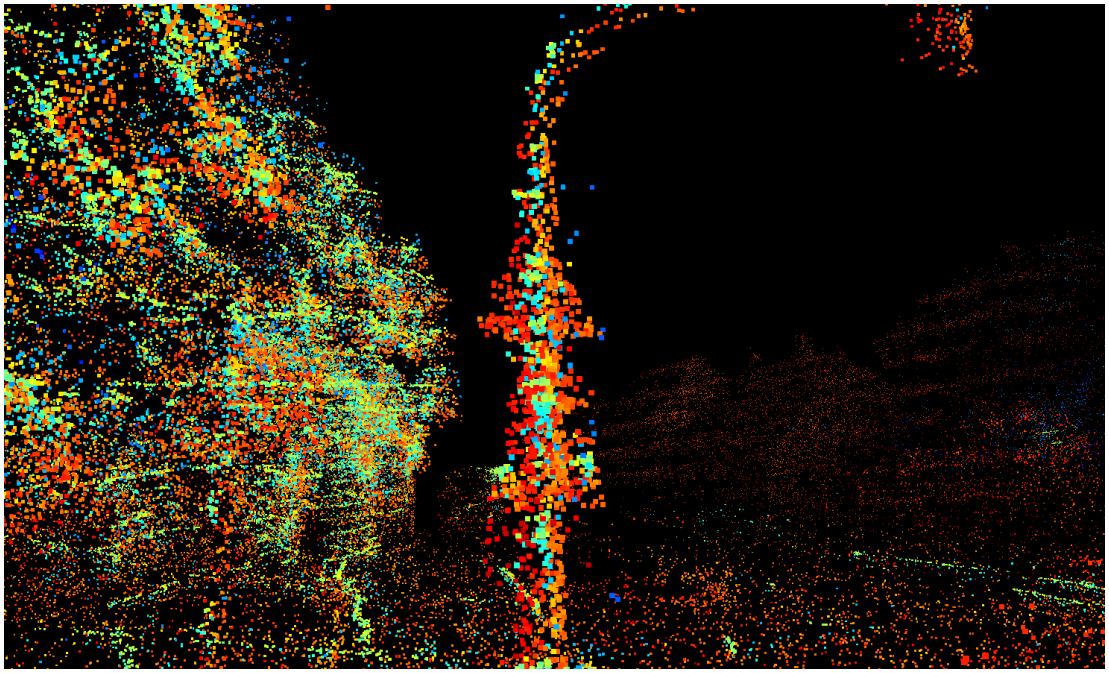
Motion compensation (as explained in Section 3.6) addresses the problem of point displacement within a single point cloud (Section 3.3.1). The process is quite straightforward and the only real parameter is the number n of groups used.

Modifying this parameter has a very small impact on the result of the motion compensation. Because n follows logarithmic curve, setting $n = 20$ leads to correction of points by up to several centimetres compared to $n = 1$, but $n = 100$ only refines the points by additional couple of millimetres compared to $n = 20$. Thus, if n is too small, there are visible problems (displaced points). And setting n to higher values is possible, but the algorithm would require more processing time while providing questionable benefit.

Motion compensation examples (Figures 4.5 and 4.6) are only demonstrated on the dataset *street*, not on the *parking* dataset. In the latter case, there is too much point displacement caused by inaccurate egomotion that it hides any data relevant to motion compensation.

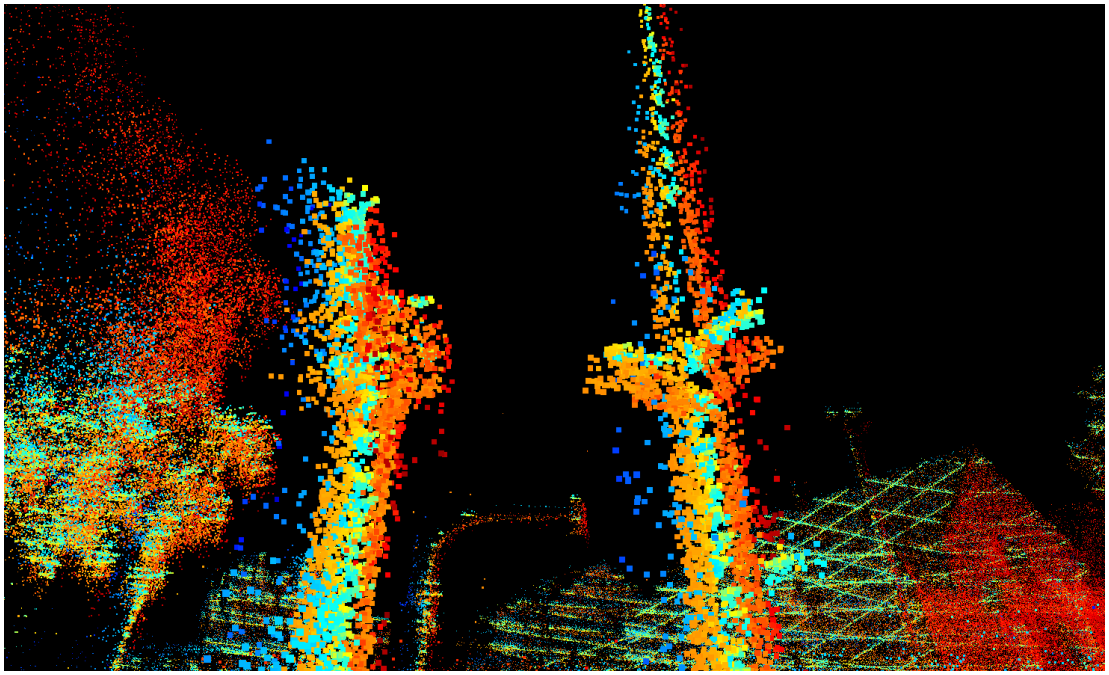


(a) Before

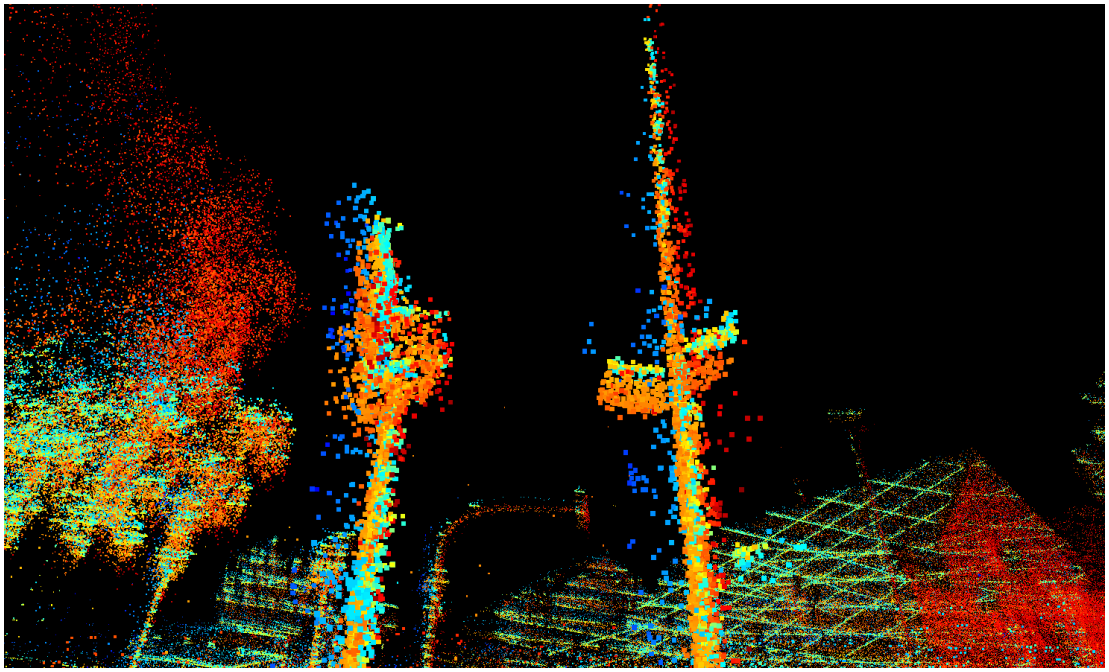


(b) After

Figure 4.5. Motion compensation on the dataset *street* – traffic light on the crossroad



(a) Before



(b) After

Figure 4.6. Motion compensation on the dataset *street* – a sign and a pole on the crossroad; notice that the blue points are still displaced – this is caused by inaccurate egomotion and cannot be fixed by motion compensation

4.6. Ground Plane and Heights of Points

Finding the ground plane consistently in different point clouds is essential for the proposed algorithm. If the plane is not consistent in two consecutive point clouds, the points representing the same object (that should be very close in the correctly reconstructed environment) would have different heights and would fall into different bins in the histogram. MLESAC with additional optimization, as described in Section 3.7, is used.

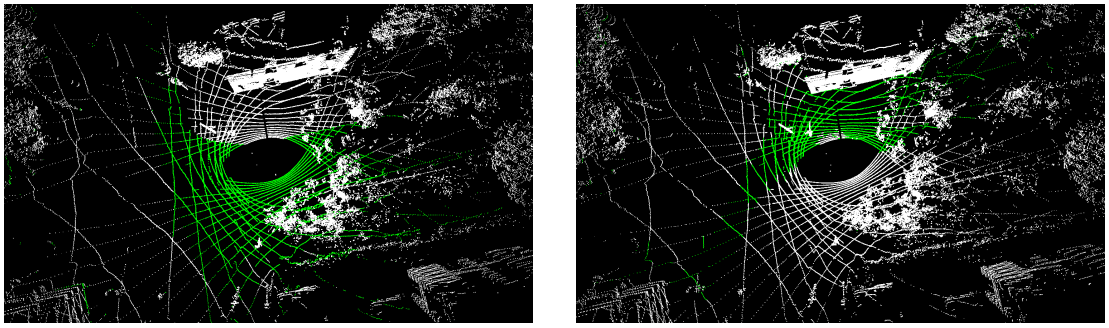
The maximal angular distance a of plane's normal n from the Z axis $[0; 0; 1]$ is set to $a = 5°$. This value is based on the assumption that breaking hard or driving over speed bump would cause the car to tilt, but no more than the chosen angle.

Threshold t was set to $t = 10$ cm. As it is usual with RANSAC parameters, this is based on observation and used datasets. But because MLE criterion is used, changing this value doesn't have a high impact on the result.

Penalty p is set to $p = 1000$ based on the datasets and the expected amount of noise (points incorrectly detected under the ground). Each underground point lowers the support by the same amount as p points directly on the plane would increase it.

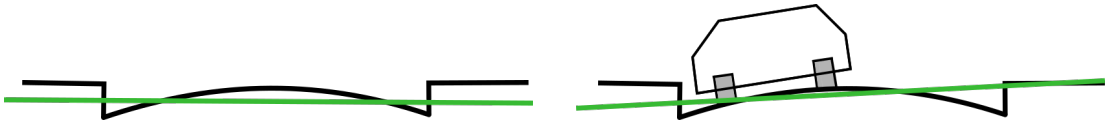
As it is apparent from Figure 4.7, locating the ground plane is problematic. It might be caused by the fact that the street is actually not a plane, but a slightly curved surface. This means that there are multiple positions where the ground plane could be located, all providing similarly good support.

On top of that, other objects close to the ground (like cars' wheels or sidewalks) directly affect the computed support and thus the plane's position. If there is e.g. a parked car, it would be registered in some point cloud and its wheels would affect locating the ground plane. But after a second or two, it would be out of range and the ground plane would be located differently, as outlined in Figure 4.8.



(a) Point cloud 1 **(b)** Point cloud 2

Figure 4.7. Inconsistent plane detection in two consecutive point clouds at the crossroad in the *street* dataset; green points are the plane inliers (within 10 cm from the plane)



(a) Empty street **(b)** Street with another car

Figure 4.8. Street cross-section; notice that the street is not flat, but rather arched; green line is the detected ground plane

4.6.1. Alternative Approach

Because locating the ground plane and computing point's heights proved to be problematic, a different approach is used in these experiments. Instead of recording heights of points, just the points' coordinates on the Z axis are registered. This completely removes the necessity for the ground plane detection while still following the original idea of the proposed algorithm.

This alternative comes with a slight disadvantage though. Registering heights of points would allow for easy parametrization of space. Thus, for example if a car was driving uphill, no special approach or modification would be necessary to the algorithm or the scene representation. This is no longer the case when the Z coordinates are registered. If a car drives uphill, the recorded Z axis coordinates might soon become too big to be registered into the histogram.

The histogram settings – the minimum and maximum values that can be registered – must be chosen with a specific dataset in mind. In real-life scenarios, any elevation of just a couple of tens of metres would be problematic regarding memory requirements. Therefore this alternative approach is only applicable in these experiments, where datasets are relatively small and flat. Robust ground plane localization and computation of points' heights is a topic of future work.

4.7. Convergence of the Optimization Function

It is essential for the optimization function to converge to the right value in order to register the points correctly. In this experiment, a short sequence of point clouds is merged together (there is no major point displacement, because that typically occurs when point clouds that are recorded far from each other are merged). Once they are combined, a random point cloud pc that has already been added is registered to the scene representation again, but slightly shifted (only x and y coordinates) from its original position. The correct location of pc in this experiment should be around point $[0; 0]$ – no relative shift from the used egomotion data. Registration of points in this experiment is done *without* the histogram convolution.

This experiment has two parts, as described in Algorithm 2. In the first part, pc is repeatedly shifted by $[i, j]$, where i and j range from -5 to 5 centimetres. For each shift, it is evaluated how well does pc fit to the map, the result is stored in a matrix P. At the end, the matrix P is visualized. If the scene representation carries useful information, there should be a strong peak in the middle, around point $[0; 0]$.

In the other part, the point cloud pc is shifted by a random amount r. The algorithm then tries to optimize the objective function as defined in Section 3.8, which yields translation t. If working correctly, this t should be reasonably similar to the negative of r, $t \simeq -r$, in order to counter the random shift and to place pc to its correct position around $[0; 0]$.

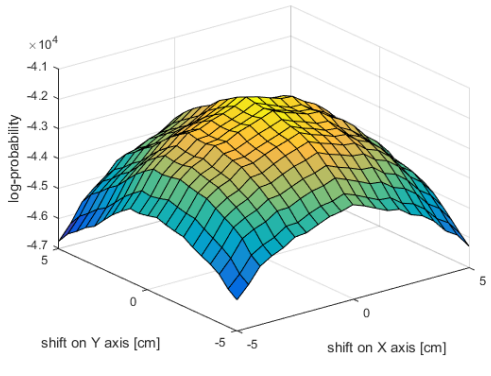
Note that the shift only affects x and y coordinates and shift of z coordinates and rotations are left out. This is purely for visualization purposes, as it would be impractical to try to visualize the results for more than 2 degrees of freedom.

Algorithm 2 Experiment pseudocode

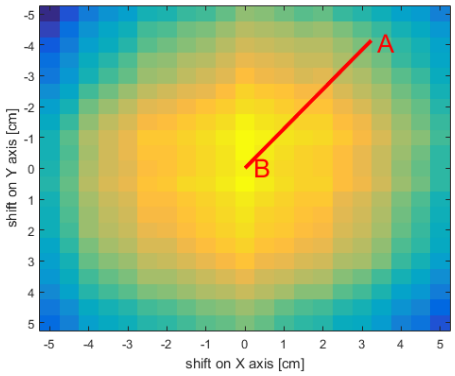
Require: Map, pc – map (scene representation), point cloud to be inserted

```
P = [ ]                                                    ▷ visualization part
for i = −5 : 5 do    // in cm
    for j = −5 : 5 do    // in cm
        P.record(Map.evaluate(shift(pc, [i; j])))
visualize P

r =random(1, 2)                                            ▷ convergence part
t =optimize(Map.evaluate(shift(pc, r)))
```

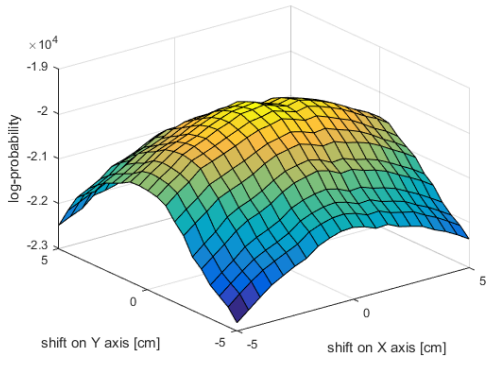


(a) Visualization part

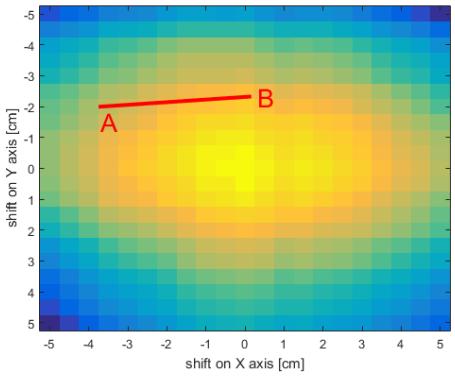


(b) Convergence part

Figure 4.9. The convergence experiment performed on a sequence of 20 point clouds from the dataset *street*; the point cloud *pc* is shifted to the location A and the optimization function correctly converges to B; note that the points are registered without the histogram convolution



(a) Visualization part



(b) Convergence part

Figure 4.10. The convergence experiment performed on a sequence of 20 point clouds from the dataset *parking*; the point cloud *pc* is shifted to the location A, but the optimization function fails to converge to the correct position and gets stuck in a local maximum B instead; note that the points are registered without the histogram convolution

Visualization part of Figures 4.9 and 4.10 shows a peak in the middle. On the dataset *street*, the function was able to converge to the correct value in the centre. However, in the sequence from the dataset *parking*, the function failed to converge correctly and got stuck in local maxima instead.

The functions are not smooth either, there are many local extrema present. As demonstrated, this could pose a problem for the convergence of the optimization function. Therefore the histogram convolution and registering points into the surrounding cells is used to smooth out the function and to try to remove the local extrema.

4.8. Histogram Convolution and Registering into the Surrounding Cells

Problem with the convergence of the optimization function in Section 4.7 is most likely caused by numerous local extrema. To remove them, or at least to limit their amount, the histogram is updated with a Gaussian rather than with a single value, as explained in Section 3.10.

The mean of the Gaussian $\mathcal{N}(i, \sigma^2)$ is i, the target bin of the histogram, and the standard deviation is set to be $\sigma = 1.5$. This Gaussian is then added to the histogram H.

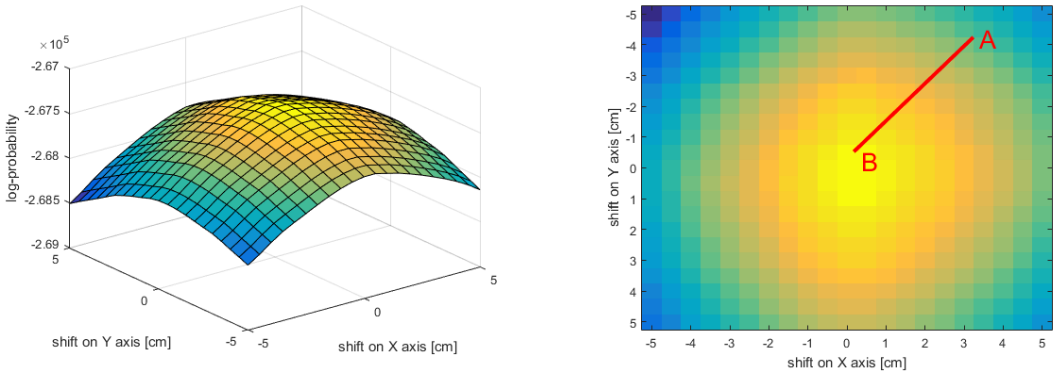
Because the Gaussian $\mathcal{N}(i, \sigma^2)$ is also registered to the surrounding cells, it is important to choose some suitable coefficients by which it is multiplied before it is added. These coefficients (Figure 4.11) try to loosely model 2D Gaussian distribution.

The experiment in Section 4.7 is repeated on the same data, this time *with* the convolution.

Results of these experiments (Figures 4.12 and 4.13) look similar to those in Section 4.7. But taking a closer look at the peaks, it is apparent that the convolution makes the objective function smoother, as shown in Figures 4.14 and 4.15. Unfortunately, the optimization function still doesn't converge correctly on the dataset *parking*. Figure 4.15 shows that there might be some local extrema remaining that cause the optimization function to fail. Choosing a different, more suitable optimization method is another topic of future work.

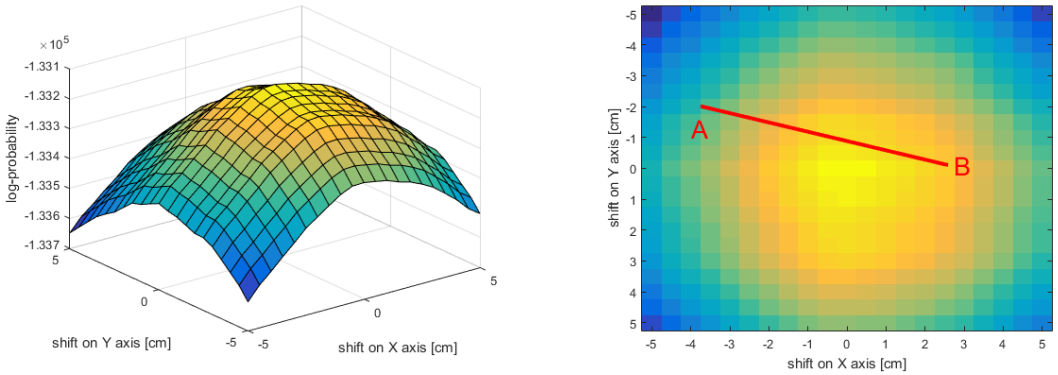
0.1	0.17	0.25	0.17	0.1
0.17	0.3	0.5	0.3	0.17
0.25	0.5	1	0.5	0.25
0.17	0.3	0.5	0.3	0.17
0.1	0.17	0.25	0.17	0.1

Figure 4.11. When Gaussian $\mathcal{N}(i, \sigma^2)$ is added to the central cell, it is also added to the surrounding cells after multiplying by these coefficients



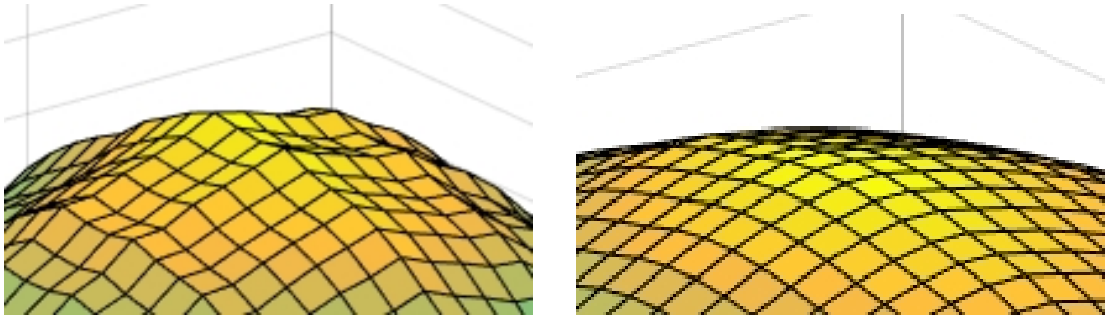
(a) Visualization part **(b)** Convergence part

Figure 4.12. The convergence experiment repeated with the same sequence from the dataset *street*, with the histogram convolution; the point cloud pc is shifted to the location A and the optimization function correctly converges to B; the objective function is significantly smoother



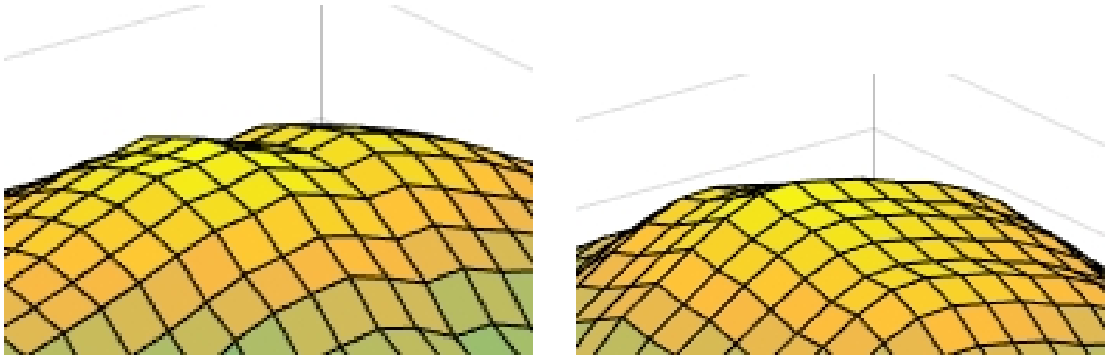
(a) Visualization part **(b)** Convergence part

Figure 4.13. The convergence experiment repeated with the same sequence from the dataset *parking*, with the histogram convolution; the point cloud pc is shifted to the location A, but the optimization function still fails to converge correctly, even though the objective function is somewhat smoother



(a) Before **(b)** After

Figure 4.14. Magnified peak from the dataset *street* used in the convergence experiment before and after the convolution; notice the local extrema present before the convolution are removed



(a) Before **(b)** After

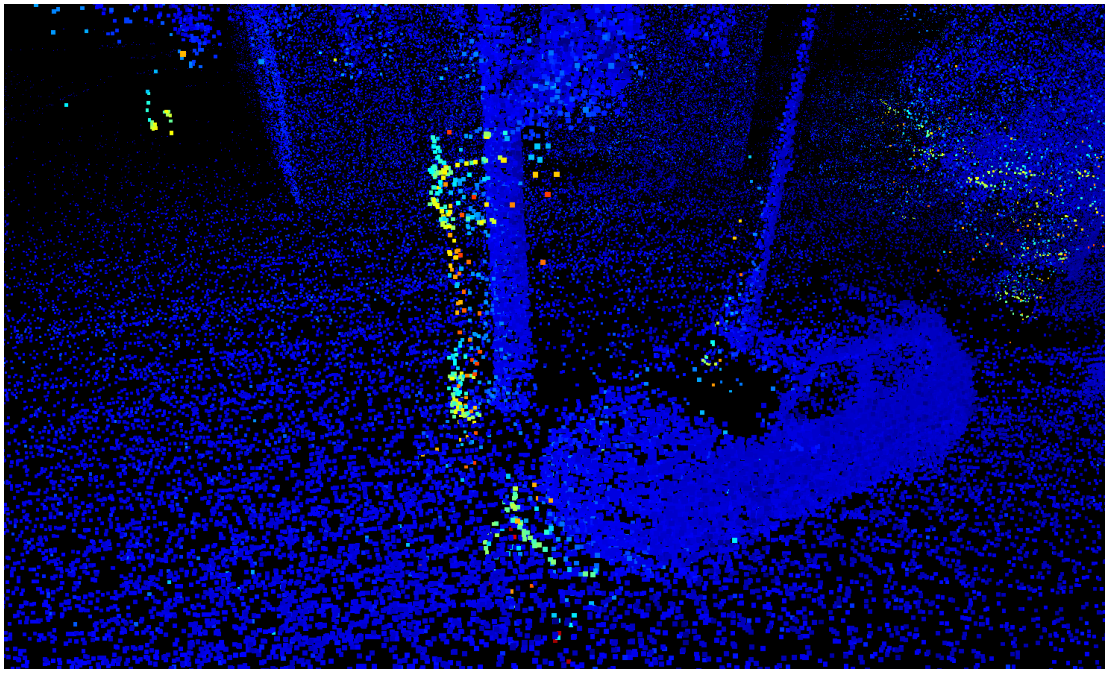
Figure 4.15. Magnified peak from the dataset *parking* used in the convergence experiment before and after the convolution; notice the number of local extrema present before the convolution is reduced

4.9. Overall Results

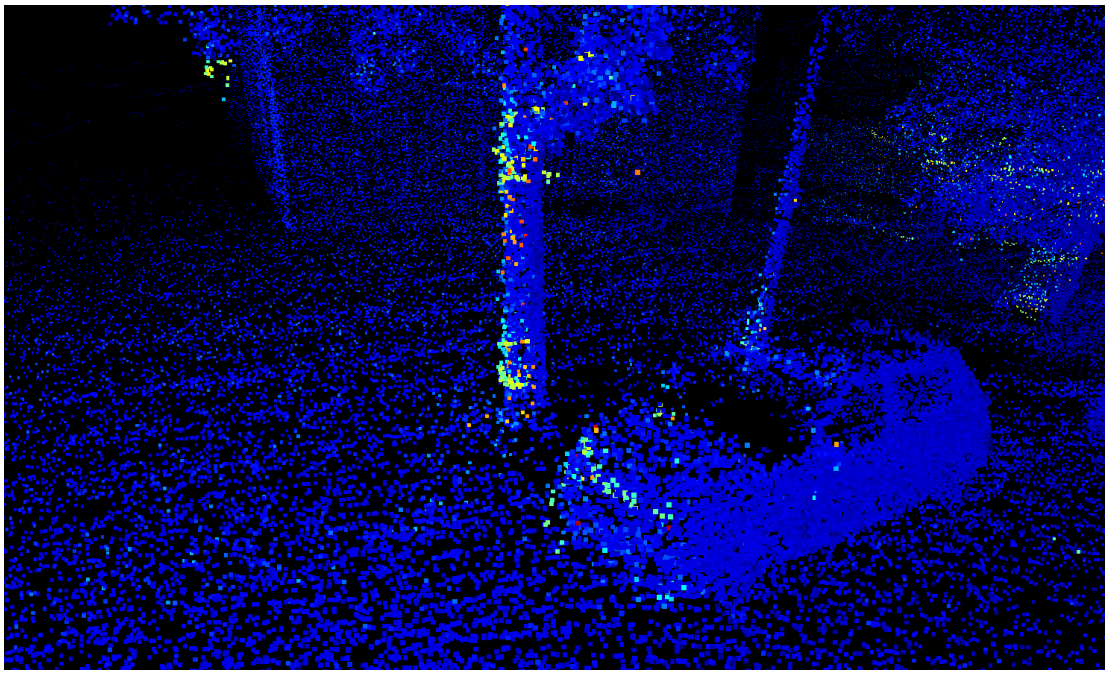
As illustrated by Figures 4.16 and 4.17, the algorithm performs very well on the dataset *street*. Misplaced points are detected and the point cloud is moved to the position with the highest probability, which improves the egomotion and fixes the problem of duplicate surfaces.

However, the dataset *parking* is far more challenging. Figure 4.18 shows that the algorithm fails to converge to the global maximum and all the major point displacement and duplicate surfaces remain.

The car makes several loops around the parking lot, which includes a couple of U-turns. While the yellow and orange points generated when the car is driving straight are matched correctly, the blue points that were recorded before the U-turn are displaced (Figure 4.19). It appears that the U-turn causes a rapid change of the rotation matrix and the optimization function fails to deal with that. Coupled with the non-smooth surface of the objective function shown in Figure 4.15, it gets stuck in a local maximum instead.

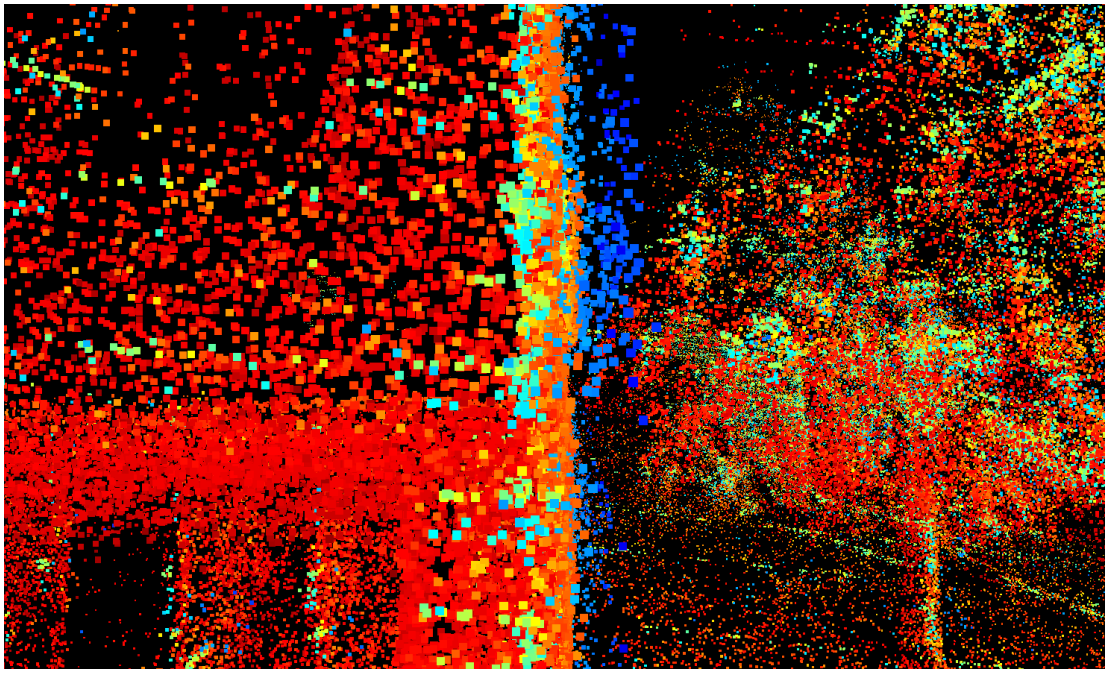


(a) Standard merging with inaccurate egomotion

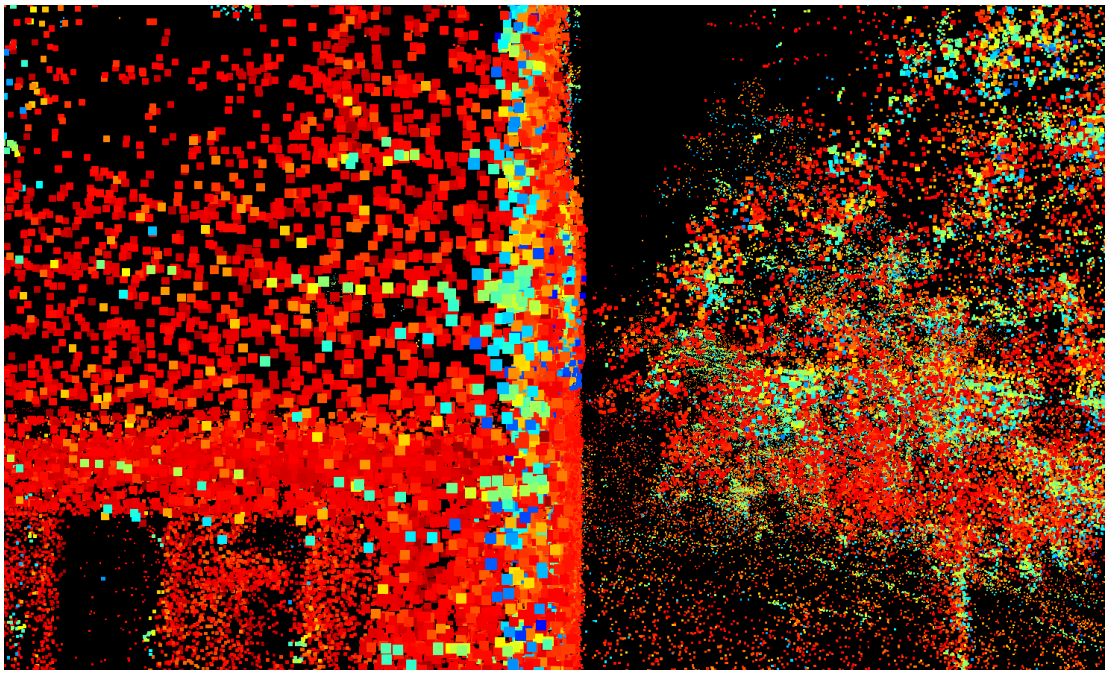


(b) Merging point clouds with improved egomotion

Figure 4.16. Results of the algorithm on the dataset *street*; the bright points were recorded later than the blue points and were given inaccurate egomotion data, leading to duplicate surface of the tree and the car; the algorithm finds that they could be inserted into the scene with a higher probability and transforms them accordingly

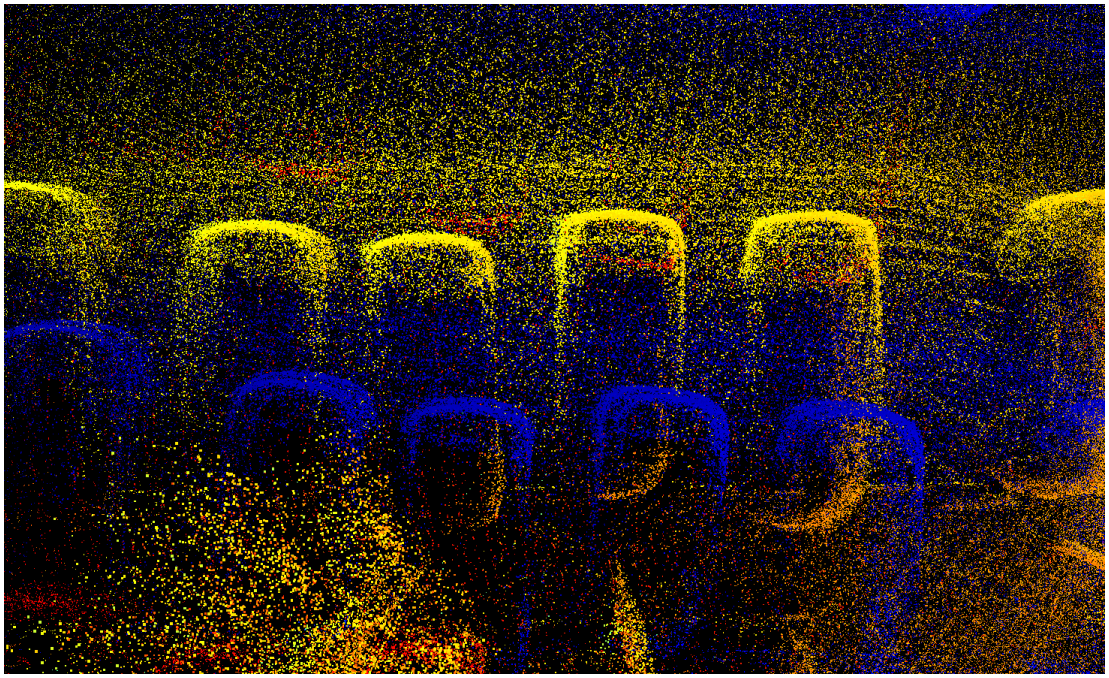


(a) Standard merging with inaccurate egomotion

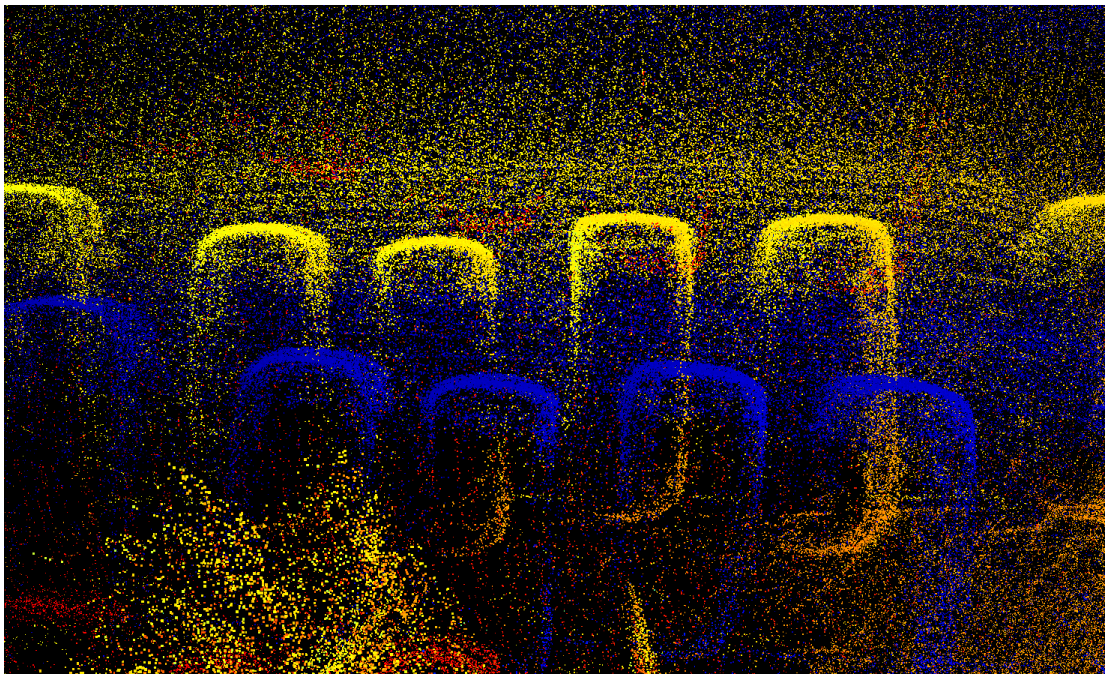


(b) Merging point clouds with improved egomotion

Figure 4.17. Results of the algorithm on the dataset *street*; the blue points were recorded much sooner than the red points and the accuracy of the egomotion had decreased in the meantime, leading to duplicate surface of the building; the algorithm detects that they could be inserted into the scene with a higher probability and transforms them accordingly

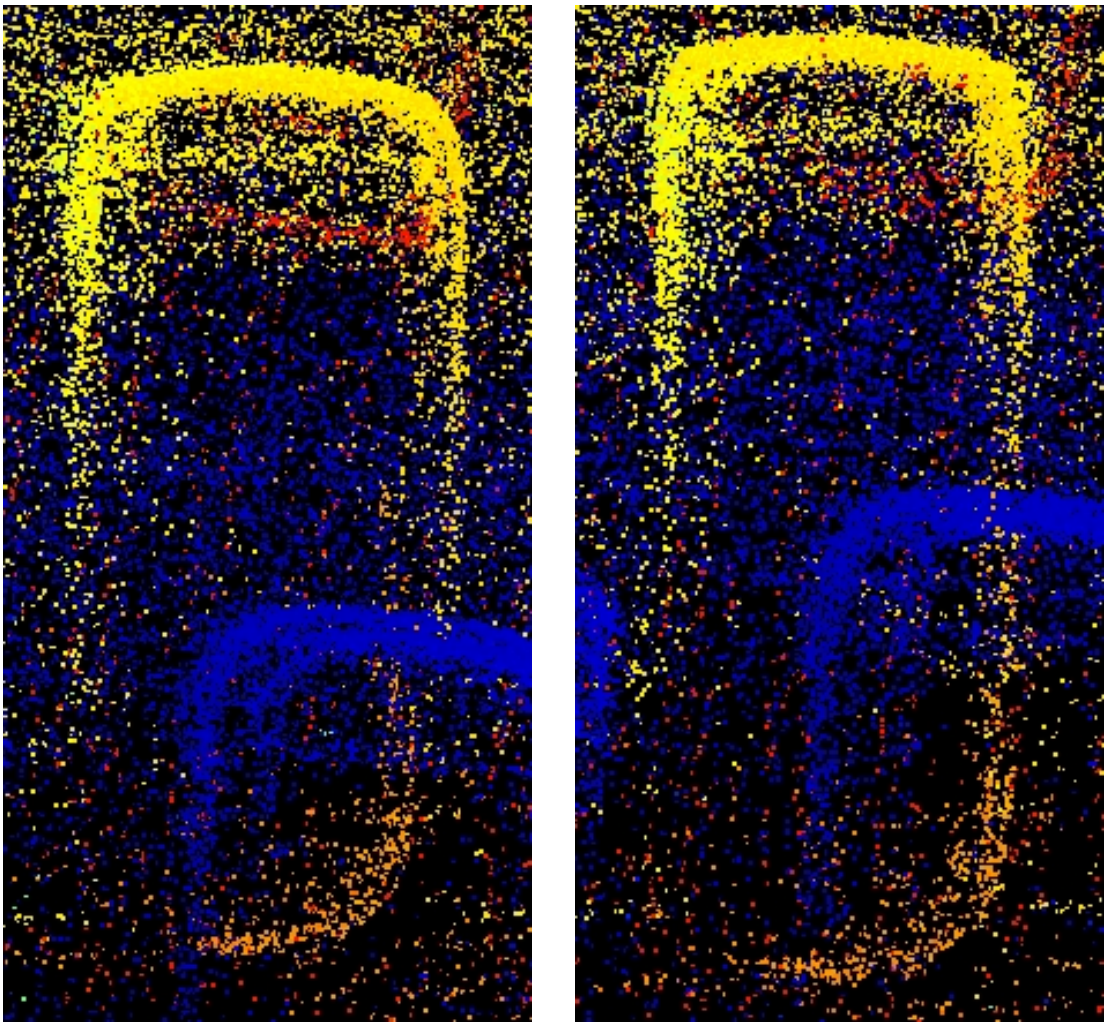


(a) Standard merging with inaccurate egomotion



(b) Merging point clouds with improved egomotion

Figure 4.18. Results of the algorithm on the dataset *parking*; the car was driving in circles around the parking lot – yellow points were recorded 1 lap after the blue points, orange points 1.5 laps after the blue points; the algorithm fails to converge to the global optimum and match the blue and yellow points together; however, some improvement is observable as the yellow and orange points are matched correctly



(a) Standard merging with inaccurate egomotion

(b) Merging point clouds with improved egomotion

Figure 4.19. Zoomed-in result from the dataset *parking* – yellow and blue points represent the same car and are not matched properly; however, yellow and orange points, also representing the same car, are matched correctly using the improved egomotion

5. Summary

In this thesis, the problem of correct registration of LIDAR point clouds and egomotion estimation was investigated. A new scene representation was proposed, where the XY plane is divided into cells, each of the cells contains a histogram of points' heights. It is a discrete representation of probability density function of points in space.

An online algorithm for registering new point clouds to this scene representation was introduced and used for improving the egomotion. Registering a new point cloud could be summarized as follows:

- Point cloud is assembled
  - Ground plane is found
  - Heights of points are computed
- Optimization is performed so that the point cloud fits into the scene representation with the highest probability
- Point cloud is transformed according to the optimization result
- Point cloud is added to the scene representation
  - Histogram convolution is performed to limit the adverse effects of local extrema on the optimization function

The experiments conducted on datasets from real drives showed that the proposed scene representation carries useful information for improving the egomotion. However, computing the heights of points proved to be quite problematic. Therefore an alternative approach was tested, where only the Z coordinate of each point was recorded into the histogram.

The proposed scene representation and algorithm were successfully used for improving the egomotion and reconstructing the 3D environment. Even though the histogram convolution was performed, in some cases the remaining local extrema prevented the optimization function from converging to the correct value.

5.1. Future Work

More robust way of determining the heights of points should be studied. Currently, locating the ground plane is affected by other objects like cars' wheels, which leads to inconsistent results. The same points are assigned different heights and cannot be matched.

Also, different means of optimization should be explored that would avoid the local extrema and converge to the correct value.

Bibliography

[1] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing icp variants on real-world data sets. *Auton. Robots*, 34(3):133–148, April 2013. 2

[2] Frank Moosmann and Christoph Stiller. Velodyne slam. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 393–398. IEEE, 2011. 2

[3] Seungpyo Hong, Heedong Ko, and Jinwook Kim. Vicp: Velocity updating iterative closest point algorithm. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1893–1898. IEEE, 2010. 2

[4] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1):3–20, 2006. 2

[5] Bernd Kitt, Andreas Geiger, and Henning Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 486–492. IEEE, 2010. 2

[6] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007. 2

[7] Yashar Balazadegan Sarvrood, Siavash Hosseinyalamdary, and Yang Gao. Visual-lidar odometry aided by reduced imu. *ISPRS International Journal of Geo-Information*, 5(1):3, 2016. 2

[8] Sebastian Schneider, Thorsten Luettel, and Hans-Joachim Wuensche. Odometry-based online extrinsic sensor calibration. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1287–1292. IEEE, 2013. 2

[9] Mark Fiala and Alex Ufkes. Visual odometry using 3-dimensional video input. In *Computer and Robot Vision (CRV), 2011 Canadian Conference on*, pages 86–93. IEEE, 2011. 2

[10] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2174–2181. IEEE, 2015.

[11] Julien Moras, Véronique Cherfaoui, and Phillipe Bonnifait. A lidar perception scheme for intelligent vehicle navigation. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 1809–1814. IEEE, 2010.

[12] Robert Zlot and Michael Bosse. Efficient large-scale 3d mobile mapping and surface reconstruction of an underground mine. In *Field and service robotics*, pages 479–493. Springer, 2014. 2

[13] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2. Citeseer, 2014. 2

[14] Eric W Weisstein. Point-plane distance. MathWorld - A Wolfram Web Resource, http://mathworld.wolfram.com/Point-PlaneDistance.html, 2002. 9

[15] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000. 9, 10

[16] Up-drive - h2020 european union program. http://up-drive.eu/, 2016. 14

[17] Donald M Olsson and Lloyd S Nelson. The nelder-mead simplex procedure for function minimization. *Technometrics*, 17(1):45–51, 1975. 14

[18] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[19] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951, 1996.

[20] Josep Aulinas, Yvan R Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *CCIA*, pages 363–371. Citeseer, 2008.

A. Implementation Pseudocode

The algorithm is divided into several parts. Here is a brief overview of each of them and how they interact with each other.

The main loop is located in function *reconstruct_trip()*. It is being given new data, processes them and corrects the egomotion and the point cloud. Map is the scene representation as described in (3.2).

Algorithm 3 function *reconstruct_trip()*

Require: Nothing
Ensure: Exports correctly reconstructed environment X

```
X ← [ ]
Map ←new Map()
while d = available_data do
    X_temp ←reconstruct_pointcloud(d)
    [R, t] ←optimize(Map.evaluate(X_temp))
    X_corr ← R · X_temp + t
    Map.add(X_corr)
    X.append(X_corr)
export X    // correctly reconstructed environment
```

Function *reconstruct_pointcloud()* is given a chunk of data. These data belong to four different sensors and must be processed separately before being combined into the final point cloud. Once the point cloud is complete, the ground plane is found and height of each point is computed.

Algorithm 4 function *reconstruct_pointcloud(data)*

Require: data – raw data from Velodyne
Ensure: Returns X, h – single point cloud reconstructed from 4 LIDAR sensors and heights of respective points

```
X ← [ ]
for sensor = 1:4 do
    X_temp ←reconstruct_sensor(data(sensor))
    X.append(X_temp)
// ground =find_ground(X) - removed from the experiments
// h =compute_heights(X, ground) - removed from the experiments
X =pc_color(X)    // optional, add colors to the point cloud for easier interpretation
```

Function *reconstruct_sensor()* takes the raw data and transforms them from the sensor coordinate system into the car coordinate system. Then some preprocessing and motion compensation is performed. After that, the data are transformed into the world coordinate system and returned.

Algorithm 5 function *reconstruct_sensor(data)*

Require: data – raw data from Velodyne
Ensure: Returns X_{part} – part of the point cloud in world coordinates
X_{part} ←sensor2car($data$) //transform data from sensor coords to car coords
preprocessing of X_{part}
motion compensation of X_{part}
X_{part} =car2world(X) // transform data from car coords to world coords