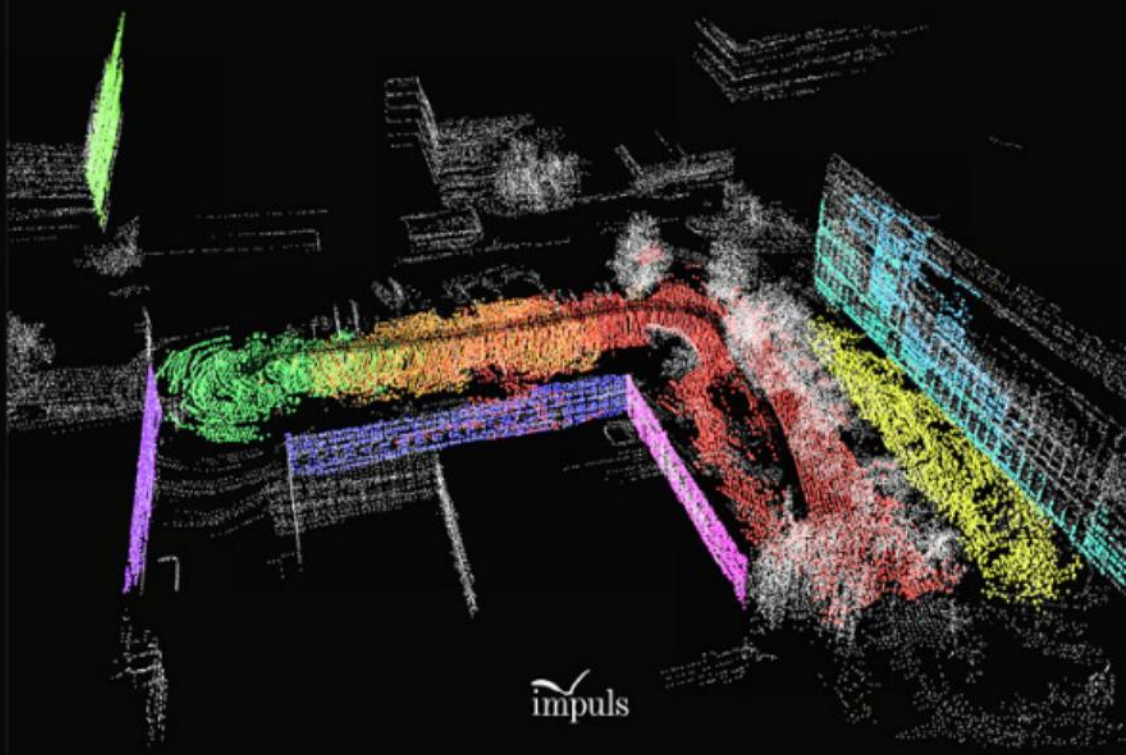
Krzysztof Ćwian

Feature-based laser simultaneous localization and mapping for automotive applications



impuls

Feature-based laser simultaneous localization and mapping for automotive applications

Krzysztof Ćwian

Kraków 2019

Contents

Preface

This technical report, published as a monographic book, deals with the problem of localization of a vehicle with a 3D laser scanner. Localization without GPS is currently a research topic with increased public interest due to the vast possible applications with a still elusive dream of an autonomous vehicle. Therefore, the book covers the process of developing a new featurebased laser simultaneous localization and mapping system and presents a complete evaluation in different conditions.

The core idea is to propose an efficient representation of planes and lines instead of the typical approach operating on points. The created map of planes and lines is used to obtain a more accurate representation of the environment and to obtain a greater localization accuracy. The book describes in detail the developed system and gives an overview of the feature processing pipeline. Conducted experiments made it possible to verify the efficiency of the created solution and to present the comparison of results obtained without and with high-level features. The experiments performed in both indoor and outdoor environment in road traffic conditions were analyzed to compare the accuracy and perform the time analysis of the processing steps of the algorithm.

The book is a natural extension of my master thesis titled “Feature-based laser simultaneous localization and mapping system“ and supervised by Michał Nowicki, PhD, as this topic was also my main research direction in the project “Advanced driver assistance system (ADAS) for precision maneuvers with single-body and articulated urban buses” supported by National Centre for Research and Development under grant POIR.04.01.02-00-0081/17. Due to the close collaboration with Solaris Bus & Coach S. A., it was possible to perform experiments on a real city bus in scenarios resembling regular operation of such a bus in a sub-urban (small town) environment. I’m grateful for this opportunity. Moreover, I would like to thank the SICK Polska Sp. z o.o. for their help with the SICK MRS6000 sensor and for providing its documentation.

I would also like to express my gratitude to Piotr Skrzypczyński for continuous discussions and always pushing me to achieve better results.

Chapter 1
Introduction

Mobile robotics is a field of science that combines different sub-fields, such as robotics, information engineering, mechatronics and electronics. The state-of-the-art in the field rapidly evolves with the constantly increasing number of possible usages and applications of mobile robots.

These robots are already widely used in automotive [6], transport [29], military [25] and by emergency services [22]. In the last few years, there was also an increase in the popularity of flying robots used for civil applications such as aerial photography [4].

Most of these purposes require some degree of autonomy for proper operation as the robots are required to perform more complex tasks. Therefore the system's reliability and robustness to different environmental conditions is sought from the proposed solutions. For that reason, it is important to verify implemented algorithms in different conditions, so that they could be used in a predictable and safe way.

1.1 PROBLEM STATEMENT

One of the key requirements for the autonomy of mobile robotics is self-localization as robots must be able to determine their position in order to perform appropriate actions. What is more, they should be capable of detecting surrounding objects, for example, to avoid collisions and plan their motion. Therefore, the Simultaneous Localization and Mapping (SLAM) problem has to be solved, which is possible with systems utilizing data from cameras [13] or laser scanners [14].

Systems based on these sensors do not require a Global Positioning System (GPS) signal to work properly, which might not be available or reliable for some applications. An example of such usage would be an Advanced Driver Assistance System (ADAS) applied in a vehicle moving in urban traffic. Its function would be to assist the driver while parking or maneuvering, especially in narrow, urban areas and in GPS-denied environments such as an underground car park. The system should be capable of self-localization in relation to other vehicles, buildings or landmarks, but only on relatively short distances. It is due to the fact that the localization error usually increases with time as laser mapping is an iterative process susceptible to drift. Whereas the above-defined requirements could be satisfied by a relatively simple laser-based (visual) odometry system that registers together with the consecutive scans in a source-to-target manner, we have decided to develop a system that builds a persistent map of the environment. The reasons for this decision are threefold:

- registration of the incoming scans to a map decreases the trajectory drift;
- the map can be further structured (into features) and used to handle more complicated situations, such as removal of spurious measurements or whole objects (e.g. dynamic);
- the spatial data collected in the map can be used to support motion planning algorithms and safety measures, which is however beyond the scope of this report.

Such a solution, which can be considered a SLAM system, is developed in the project performed by the Poznan University of Technology (PUT) and Solaris Bus & Coach (SBC). Its goal is to provide a system that will localize an electric bus relative to the charging station and help the driver during maneuvering. For this reason, the goals of this report coincide with the mentioned project requirements.

1.2 MOTIVATION AND GOALS OF THE REPORT

Laser scanners provide a large amount of data that has to be processed in real-time for the SLAM algorithm to provide accurate and up-to-date localization estimated. The purpose of this report is to propose a solution that will improve one of the state-of-the-art laser-based SLAM algorithms. The new approach is to detect and utilize higher-level features (planar surfaces) extracted from laser scans that will group the measured points. The main difference with respect to the original approach used in LOAM [32] it the fact, that the planar features are permanent in the map, i.e. they can be updated with new measurements, which allows them to grow much bigger than in LOAM. This, in turn, makes possible to represent meaningful geometric structures (e.g. walls) with individual features. This can potentially reduce the algorithm's computational complexity because this will make it possible to search for the scan points correspondences only in the closest features from the map. These features are also used to improve the robustness of the SLAM algorithm due to a more accurate feature-based map representation. What is more, the system based on planar features should be characterized by increased precision, because points that do not match any feature can be rejected. It is particularly relevant in urban areas with many buildings or in an indoor environment. Reduced map size will also contribute to a smaller number of calculations that are necessary for the optimization and thus accelerate this step. The solution with persistent geometric features solution gives an opportunity to perform optimization by finding correspondences between detected features instead of points. This is, however, a matter of our further research, and is not covered in this relatively short report.

1.3 REPORT ORGANIZATION

The report is divided into the following chapters:

- Chapter 2 presents the state of the art. It contains a description of the laser-based systems used as the base implementation for the proposed modifications. It also shows different approaches to localization problems.
- Chapter 3 describes the main components of the proposed system. It describes the process of creating, merging and deleting of planar features that are used to represent the registered environment map.
- Chapter 4 contains a description of the used sensory setup and the conducted tests. It also presents the obtained results and their analysis.
- Chapter 5 concludes all the work and suggests future work.

Chapter 2
State of the art

2.1 2D LASER SCANNING

Depending on the application, it may be sufficient to create only a 2D map of the surrounding environment. This problem is well-researched with several existing solutions. One of the golden standard solutions is Hector SLAM [16]. This system represents the existing map as an occupancy grid that is interpolated when needed to increase the accuracy. The incoming scans are then matched to the map utilizing the Gauss-Newton optimization to determine the best transformation that aligns the current laser scan with the map.

Another well-known 2D SLAM system is GMapping [11]. It is a grid-based SLAM, whose main feature is that it utilizes Rao-Blackwellized particle filter to solve localization and mapping problem. Each particle represents a single possible trajectory of a robot and carries its individual map. The biggest downside of this approach is the computational complexity and memory requirements that increase with the required number of used particles. For this reason, authors in [10] proposed a solution to improve the efficiency of the algorithm.

A more recent system is Google's Cartographer [12] that is composed of local and global subsystems. Local SLAM is responsible for building submaps constructed with sequential scans joined using the scan matching method. Global SLAM runs in the background and its task is to check for loop closure constraints in order to arrange submaps in such a way to obtain the consistent global map. Cartographer does not implement any filtration method such as particle filters but instead uses graph-based optimization. This makes it possible to achieve satisfying results with modest hardware.

All algorithms mentioned in this section are available as packages in the Robot Operating System (ROS) [21], an open-source operating system providing many useful libraries for robot applications.

2.2 ITERATIVE CLOSEST POINT (ICP)

An important factor in computing a reliable sensor motion estimate from a pair of associated scans is how many Degrees of Freedom (DOF) we have to handle. A free-floating sensor has 6 DOF, but some of these DOFs can be easily eliminated making commonsense assumptions, e.g. that the vehicle moves on a planar surface, or measuring the yaw and pith angles with respect to the gravity vector (using accelerometers). Solving the scan-matching problem with 3 DOF reduces computational complexity compared to 6 DOF solutions, but cannot be always applied to point cloud data in practical scenarios.

When point clouds from 3D laser scanners or RGB-D sensors are available, ICP [26] is usually used to match two point clouds. This method is based on distance calculation between points contained in both point clouds and its goal is to find best correspondences and iteratively calculate transformations between them. As a result, it creates a single model of the environment from the combined scan data. This is crucial in some applications such as terrain mapping and 3D reconstructions because of the limited range of the sensors that can not scan the whole scene during a single measurement. What is more, it gives the possibility to register some objects that were occluded initially but got revealed later. There also exist many alternative solutions for point cloud matching. A comparison of different point registration algorithms is contained in [2].

2.2.1 Point-to-point ICP

Point-to-point ICP algorithm determines the corresponding points in both clouds by searching for the nearest points in given sets based on Euclidean distance [20, 26]. For the *i*-th point in the original cloud, the distance d_i to the closest point in the second point cloud can be written as:

$$d_i = \min_j \sqrt{\mathbf{m_i}^2 - \mathbf{s_j}^2},\ i = 1, ..., N_M,\ j = 1, ..., N_S \tag{2.1}$$

where:

$\mathbf{m_i}$, $\mathbf{s_j}$ − points in given point clouds
N_S, N_M − number of points in point clouds

At this moment, points that do not have their correspondence closer than some threshold might be rejected in order to improve algorithm accuracy. Afterwards, both point clouds are centered. For this step it is necessary to compute the centroids $\mathbf{C_M}$ and $\mathbf{C_S}$ of the point clouds using the following equations:

$$\mathbf{C_M} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{m_i} \tag{2.2}$$

$$\mathbf{C_S} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{s_j} \tag{2.3}$$

where:

N − number of corresponding point pairs

Subsequently, points in given cloud are aligned with the center by subtracting calculated centroid from points coordinates:

$$\mathbf{M'} = \left\{\mathbf{m_i'} = \mathbf{m_i} - \mathbf{C_M}\right\}_{1...N} \quad \mathbf{S'} = \left\{\mathbf{s_i'} = \mathbf{s_i} - \mathbf{C_S}\right\}_{1...N} \tag{2.4}$$

The next step is to compute the transformation matrix which minimizes the distance between two clouds. It consists of matrix of rotation and translation and can be obtained using Singular Value Decomposition (SVD) of covariance matrix $\mathbf{H} = \mathbf{M'S'}$ [30]:

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{2.5}$$

where:

U, **V** − orthogonal matrices
Σ − diagonal matrix

At this stage, rotation matrix **R** and translation vector **t** can be calculated using following formulas:

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T, \quad \mathbf{t} = \mathbf{C_S} - \mathbf{R}\mathbf{C_M} \tag{2.6}$$

Eventually, the algorithm calculates error using only points with proper correspondence in the second point cloud. For this purpose, all points pairs are assigned weights $w_{ij} = 1$ if they are corresponding ones or $w_{ij} = 0$ otherwise. This way assigned weights indicate whether given pair of points should be included in equation below:

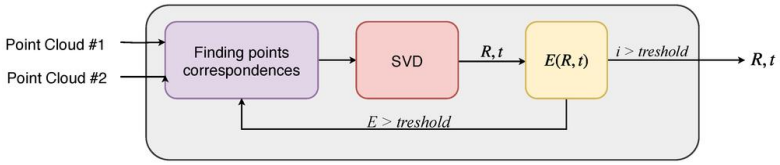
$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_M} \sum_{j=1}^{N_S} w_{ij} ||\mathbf{m_i} - (\mathbf{R}\mathbf{s_j} + \mathbf{t})||^2 \tag{2.7}$$

where:

$\mathbf{m_i}$ − point in the first point cloud
$\mathbf{s_j}$ − point in the second point cloud
R − rotation matrix
t − translation vector
w_{ij} − equals 1 for corresponding points pairs and 0 otherwise

Based on the result, algorithm continues or stops iterating until it obtains required accuracy or exceeds maximum number of iterations *i*. The block diagram of standard ICP algorithm is shown in Fig. 2.1.

Figure 2.1: Block diagram of the point-to-point ICP

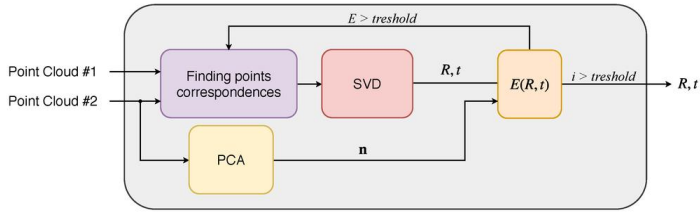


There exist modified types of the algorithm that were implemented to improve speed or accuracy of point clouds matching. One of them utilizes k-d tree [9, 19] structures for faster searching of closest neighbors. K-d trees are binary search trees that partition space to organize points. This approach significantly accelerates calculations, as searching for the closest neighbor is the most time-consuming operation required during scan matching.

2.2.2 Point-to-plane ICP

An alternative version of the ICP algorithm is the point-to-plane ICP [3, 26], that uses information about surface normals to improve accuracy. Instead of calculating error based on point-to-point distance, it calculates error based on the surface normal vector. For this task, it requires additional information, namely a surface normal associated with every point. The block diagram of point-to-plane ICP is shown in Fig. 2.2.

Figure 2.2: Block diagram of point-to-plane ICP



Normal vector can be calculated by creating a covariance matrix of nearby points and then using PCA (Principal Component Analysis) to calculate eigenvector associated with the smallest eigenvalue, as it corresponds with the surface normal. Then error can be calculated using following modification of equation 2.7 [26]:

$$E(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_M} \sum_{j=1}^{N_S} w_{ij} \left|\left| \mathbf{n} \cdot [\mathbf{m_i} - (\mathbf{R}\mathbf{s_j} + \mathbf{t})] \right|\right|^2 \tag{2.8}$$

where:

$\mathbf{m_i}$ − point in first point cloud
$\mathbf{s_i}$ − point in second point cloud
n − surface normal at point m_i
R − rotation matrix
t − translation vector
w_{ij} − equals 1 for corresponding points pairs and 0 otherwise

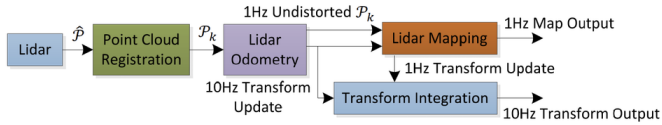
There also exist further modifications of point matching that can improve accuracy relative to point-to-plane ICP. Information about surfaces can be extracted from both point clouds. This method was implemented in Generalized-ICP [26] and named by authors plane-to-plane ICP. It relies on the assumption that all measured points are not random 3D points located in space, but instead, they are set of points lying on sampled planar surfaces.

2.3 3D LASER LOCALIZATION SYSTEMS

2.3.1 LOAM

LOAM (Lidar Odometry and Mapping) is a real-time self-localization method proposed by Ji Zhang and Sanjiv Singh [32, 33]. Initially, it was used in combination with a single line scanning lidar sensor but it was also adapted for multi-line 3D laser scanners. The processing diagram of this system is shown in Fig. 2.3.

Figure 2.3: Block diagram of LOAM system [32], consisting of point-to-point matching in *lidar odometry*, point-to-map matching in *lidar mapping* and final transformation integration



The processing flow of the LOAM system consists of a few steps, starting with the lidar odometry that performs matching of registered point clouds. It is followed by a less frequent point cloud to map matching in *lidar mapping*. The result of the processing is an environment map and a transformation that comes from joining the last *lidar mapping* pose estimate with the most recent *lidar odometry* increment.

The main feature of LOAM that distinguishes it from other methods, is a fact that it combines two processing pipelines that run simultaneously but asynchronously. The first of them is *odometry* that determines the sensor's pose based on laser measurements. It does not use any wheel encoders nor GPS receiver for this purpose but instead calculates movement increments based on matching subsequently registered scans. This causes the odometry to be prone to drift over time, but it gives a rough estimation of the sensor's location for further processing. It runs at high frequency (10 *Hz*) due to lower computational demands. The second algorithm (*mapping*) is responsible for creating a map of the environment by registering and matching subsequent scans to a map. It usually runs at ten times

smaller frequency (1 *Hz*), but it provides a more accurate pose estimate than the *odometry*.

Both *odometry* and *mapping* are based on matching features extracted from individual scans. These features can be assigned to two groups: sharp edges and planar surfaces. Points are arranged into these groups, based on their *smoothness* parameter, calculated upon the neighborhood of the point using equation [33]:

$$c = \frac{1}{|\mathbf{S}| \cdot ||\mathbf{X}^L_{(k,i)}||} || \sum_{j \epsilon \mathbf{S} \neq i} (\mathbf{X}^L_{(k,i)} - \mathbf{X}^L_{(k,j)}) || \tag{2.9}$$

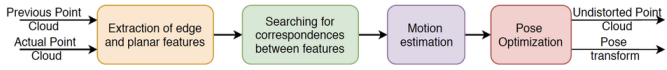
where:

i, j − point index
L − coordinate system
$\mathbf{S}$ − set of i points included in the same scan
$\mathbf{X}^L_{(k,i)}$ − coordinates of point i in the coordinate system L

Points with the highest value of *smoothness* parameter are considered edge features while points with the smallest value are named plane features. The number of selected points from a single scan is limited by threshold c. To ensure even distribution of features their number is also limited based on which region of the scan they were detected. For this purpose, each scan is divided into four subregions with a maximum number of 2 edges and 4 planar features per subregion. These numbers are partially dictated by the used laser scanner. What is more, planes that are parallel to the laser beam and occluded regions are rejected as they are considered to be unreliable.

Laser Odometry *Odometry* is used to estimate motion and pose of the sensor, based on correspondences found on subsequent scans. Block diagram presenting general *odometry* steps is shown in Fig. 2.4.

Figure 2.4: Block diagram of LOAM *odometry*, that is used to generate undistorted point cloud and pose transform based on previous and actual scan



At first, points from previous and actual point clouds are divided into the edge and planar features. The next step is to find correspondences between them. It is done by searching for the closest neighbors for all the points included in both clouds, using for this purpose k-d tree algorithm. In the described system, the linear feature is represented by two points, while planar one by three points. Because the used scanner provides only a single scan line and is constantly rotating during measurement acquisition, all the features must be found on two or three subsequent scans. After corresponding points were found, the algorithm calculates distances between them. For a given edge point i, and its corresponding edge line formed by points j and l, the point to line distance d_ε can be computed using the following equation:

$$d_\varepsilon = \frac{\left|(\tilde{\mathbf{X}}^L_{(k,i)} - \bar{\mathbf{X}}^L_{(k-1,j)}) \times (\tilde{\mathbf{X}}^L_{(k,i)} - \bar{\mathbf{X}}^L_{(k-1,l)})\right|}{\left|\bar{\mathbf{X}}^L_{(k-1,j)} - \bar{\mathbf{X}}^L_{(k-1,l)}\right|} \tag{2.10}$$

where:

$\tilde{\mathbf{X}}^L_{(k,i)}$, $\bar{\mathbf{X}}^L_{(k-1,j)}$, $\bar{\mathbf{X}}^L_{(k-1,l)}$ − coordinates of points i, j and l respectively

In an analogous way, for a given planar point i, and the corresponding planar patch formed by points j, l, m, the point to plane distance d_H is calculated as:

$$d_H = \frac{\left|\begin{array}{c}\tilde{\mathbf{X}}^L_{(k,i)} - \bar{\mathbf{X}}^L_{(k-1,j)} \\ ((\bar{\mathbf{X}}^L_{(k-1,j)} - \bar{\mathbf{X}}^L_{(k-1,l)}) \times (\bar{\mathbf{X}}^L_{(k-1,j)} - \bar{\mathbf{X}}^L_{(k-1,m)}))\end{array}\right|}{\left|((\bar{\mathbf{X}}^L_{(k-1,j)} - \bar{\mathbf{X}}^L_{(k-1,l)}) \times (\bar{\mathbf{X}}^L_{(k-1,j)} - \bar{\mathbf{X}}^L_{(k-1,m)}))\right|} \tag{2.11}$$

In order to calculate pose transform between starting time of the lidar sweep (t_k) and given point's i registration time stamp ($t_{(k,i)}$), the following equation is used:

$$\mathbf{T}^L_{(k,i)} = \frac{t_{(k,i)} - t_k}{t - t_k}\mathbf{T}^L_k(t) \tag{2.12}$$

where:

t − current timestamp
t_k − starting time of the lidar sweep k
$t_{(k,i)}$ − i point timestamp
$\mathbf{T}^L_k$ − the lidar pose transform between $[t_k, t]$

$\mathbf{T}^L_{(k,i)}$ − the lidar pose transform between [t_k, $t_{(k,i)}$]

Above equation is based on linear interpolation of previously calculated transform $\mathbf{T}^L_k = [\tau^L_k(t),\ \theta^L_k(t)]^T = [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z]^T$, consisting of translations and rotations in *x*, *y* and *z* axes. Given $\theta^L_k(t)$ and its skew symmetric matrix $\hat{\theta}^L_k(t)$, the corresponding rotation matrix is defined with the use of Rodrigues formula:

$$\mathbf{R}^L_k(t) = e^{\hat{\theta}^L_k(t)} = \mathbf{I} + \frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||}\sin||\theta^L_k(t)|| + \left(\frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||}\right)^2(1 - ||\cos\theta^L_k(t)||) \tag{2.13}$$

Afterwards, coordinates of a point *i* can be calculated as:

$$\tilde{\mathbf{X}}^L_{(k,i)} = \mathbf{R}^L_{(k,i)}\mathbf{X}^L_{(k,i)} + \tau^L_{(k,i)} \tag{2.14}$$

where:

$\tau^L_{(k,i)}$ − translation vector corresponding to $T^L_{(k,i)}$
$\mathbf{R}^L_{(k,i)}$ − rotation matrix corresponding to $T^L_{(k,i)}$
$\mathbf{X}^L_{(k,i)}$ − coordinates of a point *i*
$\tilde{\mathbf{X}}^L_{(k,i)}$ − coordinates of *i* point correspondence

By combining equation 2.14 with the distances calculated between both edge and planar point correspondences, using equations 2.10 and 2.11 accordingly, the following geometric relationships can be achieved:

$$f_\varepsilon(\mathbf{X}^L_{(k,i)}, \mathbf{T}^L_k(t)) = d_\varepsilon,\ i \in E_k \tag{2.15}$$
$$f_H(\mathbf{X}^L_{(k,i)}, \mathbf{T}^L_k(t)) = d_H,\ i \in H_k \tag{2.16}$$

where:

E_k − set of edge points
H_k − set of planar points

Eventually, by combining equations 2.15 and 2.16, the following nonlinear function is obtained:

$$\mathbf{f}(\mathbf{T}^L_k(t)) = \mathbf{d} \tag{2.17}$$

Next step is to compute the Jacobian matrix $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{T}^L_{k+1}$ and solve equation 2.17 in order to estimate motion and calculate the most accurate transform spe-

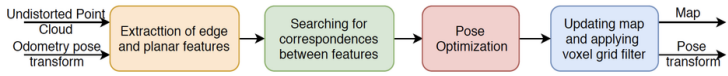
cifying traveled distance. It is accomplished using the Levenberg–Marquardt algorithm to iteratively minimize **d**:

$$\mathbf{T}_k^L \leftarrow \mathbf{T}_k^L - (\mathbf{J}^T\mathbf{J} + \lambda diag(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T\mathbf{d} \tag{2.18}$$

The algorithm stops when it converged or the the number of allowed iterations was exceeded. The result of the *odometry* is a point cloud without the distortion caused by scanner movement and also an actual pose estimate. Both of them are used as inputs for *mapping* algorithm.

Laser Mapping *Mapping* is based on the same principle as odometry, except that it uses 10 times more features from the current scan and matches them to the map of the environment instead of the previous laser scan. This way it provides a much more accurate estimation of lidar pose. Its block diagram is shown in Fig. 2.5.

Figure 2.5: Block diagram of LOAM *mapping* that uses *odometry* output to calculate more precise pose transform and create environment map



To ease the process of finding correspondences between each scan and the map, all the registered point clouds on the map are stored in cubes with a side of 50 *m*. This way only part of the map is used for matching given scan. What is more, while looking for the correspondences, the algorithm uses covariance matrix decomposition calculated with the use of points that surround the given feature. Based on the calculated eigenvalues and eigenvectors it is possible to determine whether surrounding points form an edge or planar feature and also detect its direction. After correspondences are found, the algorithm performs analogous pose optimization as in the case of *odometry*, which gives a pose transform estimate. The last step is to update the environment map and reduce its size by applying voxel grid filtering [17].

Modification The LOAM algorithm was chosen as a starting point for the system described in this report. This is due to the fact that it offers good performance and can work in realtime. What is more, it also achieved the best results

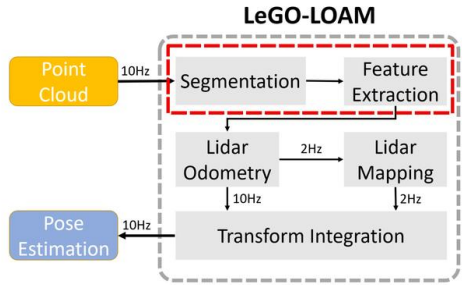
on the KITTI dataset, the most popular benchmark used for SLAM odometry systems. It is also worth to mention that the LOAM code, created by Ji Zhang, is available for public use at online repository[1].

1 https://github.com/laboshinl/loam_velodyne/

2.3.2 LeGO-LOAM

LeGO-LOAM (Lightweight and Ground-Optimized Lidar Odometry and Mapping) is a modification of the LOAM system implemented specifically for the usage on ground vehicles. It takes into account the fact that ground-moving vehicles have reduced degrees of freedom. The idea is to divide 6 DOF estimation into two separate optimizations. The planar features extracted from ground are used to estimate z-axis translation with roll and pitch angles ($[t_z, \theta_{roll}, \theta_{pitch}]$). Three remaining parameters ($[t_x, t_y, \theta_{yaw}]$) can be then obtained by finding correspondences between points located above the ground. The processing diagram of this system is presented in Fig. 2.6.

Figure 2.6: LeGO-LOAM system overview. Additional steps relative to LOAM are marked by a red rectangle



The main differences, in comparison to LOAM, are the additional *segmentation* and feature *extraction* steps (marked in red) that are performed on collected point clouds. *Segmentation* was implemented to provide good performance in variable terrain environment. It is not performed on ground points, as they are extracted and labeled at the beginning. This approach makes it possible to reject unreliable features causing mismatching and drift such as grass.

Segmentation is performed on a range image, created by projecting registered point clouds so that every point is assigned to a specific pixel in the image. From this point, standard image segmentation can be used to group characteristic points and extract features. As a result of this step, many small objects like leaves and grass are removed.

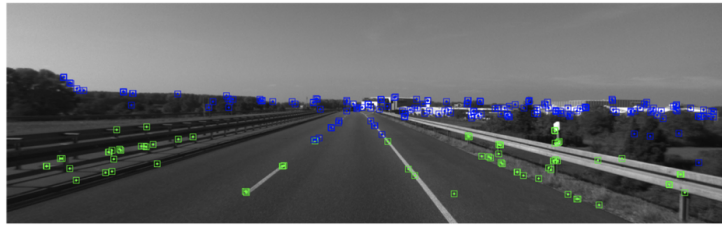
Next step is *feature extraction*. It is performed based on a previously created range image, instead of raw points. Same as in the case of original LOAM, features are divided into two groups, based on *smoothness* parameter calculated using equation 2.9. LeGO-LOAM implemented also modification to the *odometry* step. It utilizes labels that were assigned to points on range image for faster finding of correspondences.

In order to find the transformation between the two consecutive scans, it uses the Levenberg-Marquardt method, similarly as LOAM. The same method is utilized during the *mapping* stage, whose goal is to further improve accuracy. The map is stored as a set of features with the associated pose. This way only these points that are located nearby are included during calculations. What is more, LeGO-LOAM integrates the ability to detect loop closure for map refinement, which allows to significantly reduce system drift.

2.4 FEATURE-BASED VISION SLAM SYSTEMS

The LOAM and other systems utilizing ICP-like approaches estimate the transformation based on correspondences determined by the vicinity of points in the Euclidean space. On the other hand, the feature-based systems operating on images try to match detected features and then further optimize their location. These advanced systems can achieve comparable accuracy with cheaper sensors due to a more complex processing pipeline thus being an inspiration for the systems utilizing laser scans. One of the examples of these systems is the ORB-SLAM2 [18], a system using ORB [23] detector to detect features as presented in Fig. 2.7.

Figure 2.7: Features tracked in the ORB-SLAM2 [18]



The algorithm consists of three main components that run in separate threads. The first of them is tracking, whose goal is to estimate the camera's localization in a consecutive frame. The second thread is mapping and its task is to optimize the local map using bundle adjustment [7]. It stores all the registered points and also is responsible for deleting unnecessary ones. This way size of the map of the environment retains a reasonable level. The third thread is loop closure detection. This module makes it possible to eliminate accumulated drift and thus improves overall accuracy.

Another example of feature-based system might be Dense Planar SLAM [24]. It creates a map of the environment as a set of planes and was intended for indoor usage because indoor scenes consist mostly of planar surfaces. The system also uses depth images produced alternatively by RGB-D or stereo cameras. Planar features are formed by individual surfels that are extracted from images and

labeled. These surfels are divided into two groups. First of them are *planar region surfels* and second *non-planar regions surfels*. *Planar region surfels* that belong to the same common plane are assigned with the same parameters (normal vector and distance to a common plane) and thus form big flat areas. Created planar regions are refined and joined over time to provide efficient memory usage, but they are still stored as a set of individual surfels. It means that a single planar region is represented by many surface elements, which made it possible to represent complex planes that might even have holes. Authors presented also applications for this system, such as performing augmented reality. Using appropriate headsets allowed them to display user content like photos on planar surfaces such as walls and tables.

The system proposed in this report tries to take working parts of the vision-based systems and apply them in the laser-based SLAM. For example, the ORB-SLAM2 optimizes the location of the features based on measurements. A similar procedure could be used in the proposed system to optimize the plane equation based on the points assigned to that feature.

Chapter 3
Proposed feature-based 3D laser SLAM system

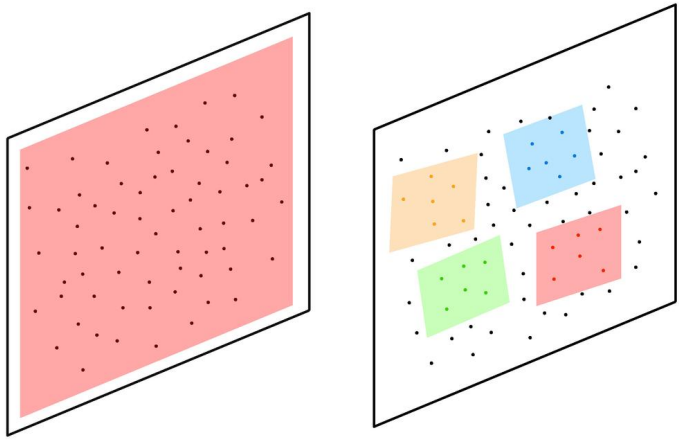
Although the ICP algorithm is one of the most common approaches during point cloud matching, it has several drawbacks. One of them is the fact, that it has relatively poor performance in terms of computational speed. What is more, it requires many iterations to achieve satisfying accuracy. For that reason, many variants and alternative solutions were developed. One approach, used in vision SLAM systems, is to use more elaborated features instead of raw points for matching. This approach was an inspiration for the system described in the report. Source code of implemented solution is available in the online repository[1].

1 https://github.com/kcwian/feature-based-LOAM

3.1 FEATURE REPRESENTATION IN THE SYSTEM

The original LOAM system stores the map as points marked either as edges or planes but these points do not form larger groups. The proposed solution makes it possible to represent a big group of points by a single, high-level feature. As a result, the feature-based system can represent the map of the environment with a smaller number of features when compared to the LOAM. This is the result of a different approach, which is especially visible in the case of large planes. In the original LOAM, all planes for optimization are formed from the five closest points, while in the proposed system their number can vary depending on the size of the plane. If we consider one big plane, such as a wall of the building, it can be stored as one big element instead of many small ones. This way it is easier to manage all the features, find correspondences between points or even detect some objects. The comparison between the proposed representation of a wall and the representation in the original LOAM is presented in Fig. 3.1.

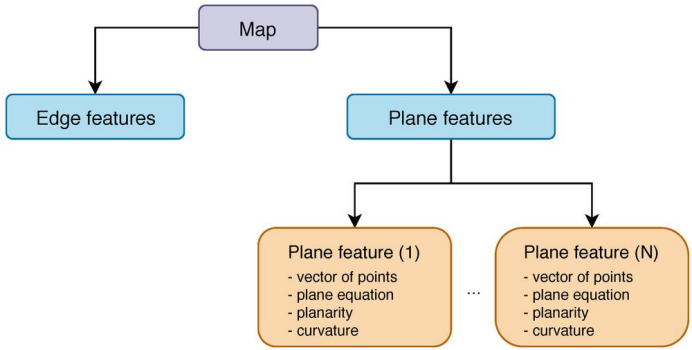
Figure 3.1: Comparison between the high-level planar feature in the modified LOAM system (left) and the cloud of planar points in the original LOAM system (right)



In the proposed system, the created map of the environment consists of both

edge and *plane* features, but only the latter ones were modified to aggregate a big number of points. This is due to the fact that the planar features are inherently larger and thus have a greater influence on the final estimate of the system. A diagram presenting the structure of the stored environment map is shown in Fig. 3.2.

Figure 3.2: Structure of stored environment map consists of edges stored as points belonging to edge features and planes forming high-level features



The single planar feature is represented by a plane equation calculated initially from several nearby points collected during a single laser scan. It has the following form:

$$Ax + By + Cz + D = 0, \tag{3.1}$$

where A, B, C, D are the coefficients of the plane. This equation is updated every time new points are added to that feature but only when the number of points forming the feature is smaller than the threshold set to 30 points. A data structure representing the planar feature contains also modified points that were downsampled by a voxel grid filter and their statistical parameters like mean value and covariance matrix. These parameters in combination with a plane equation are used when two features are considered for merging to determine whether they overlap, what was described in detail later in Section 3.1.3. All features have also coefficients describing their *planarity* and *curvature*. *Planarity* p is computed as a percentage of points with distance to the plane, which they create, smaller than certain value (0.2 m in the case of developed system):

$$p = \frac{m}{N} \cdot 100\% \tag{3.2}$$

where:

m − number of points within a distance of 0.2 m from plane
N − number of points belonging to the selected feature

Curvature is calculated using eigenvalues that are obtained by Principal Component Analysis (PCA) of a given feature's covariance matrix of points forming the feature. It is computed with the use of the following equation [24]:

$$c = \frac{\lambda_{min}}{\sum_{i=1}^{3} \lambda_i} \tag{3.3}$$

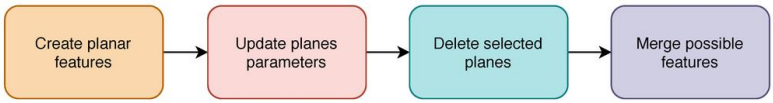
where:

c − *curvature* parameter
λ_i − eigenvalues of the covariance matrix of points forming the feature

Both of these coefficients provide similar ways to verify whether the given feature is planar and valid. The main difference between them is the fact that the *planarity* parameter might be improved by selecting points with a distance to the plane that exceeds the threshold and then deleting them. In contrast, there is not such a possibility while calculating *curvature*, as it is based only on eigenvalues, thus improving this parameter would require some additional calculations. Usability of the mentioned parameters was also discussed more broadly in Section 3.1.2.

All features have also some additional parameters, for example, the number of points, that are used mainly for statistical purposes. Management of all the created planes is accomplished in few steps consisting of creating, updating, deleting and merging of features. The general processing pipeline is shown in Fig. 3.3.

Figure 3.3: Processing pipeline implemented in the system, consisting of creating, updating, deleting and merging planar features

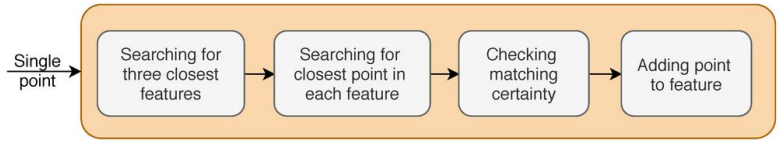


All the mentioned steps are described in detail in the following sections.

3.1.1 Creating planar features

The plane equation can be calculated using a minimal number of three points, but in order to reduce the probability of an error, the proposed method uses at least five points. This means that every plane feature is initially created using five nearby points. After that, an algorithm calculates plane equation which all newly added points must comply with. The process of creating new features and adding points to existing ones consists of several steps presented in Fig. 3.4.

Figure 3.4: Steps performed to match single point to the existing feature



In the first place, the implemented algorithm attempts to assign each new point to one of the already existing features. For this purpose it calculates the point-to-plane distance to the three nearest features using the following equation:

$$d = \frac{Ax_0 + By_0 + Cz_0 + D}{\sqrt{A^2 + B^2 + C^2}} \tag{3.4}$$

where:

(x_0, y_0, z_0) − point coordinates
A, B, C, D − parameters of plane normal equation

This step, in a relatively short period, reduces the number of potential matches as planes that are further away than some threshold, equal to 0.3 *m*, will be instantaneously rejected. Then it calculates distances to the nearest point included in each of the three features in order to determine which is the closest one. In this case, the distance must be smaller than 1 *m* for the feature to be further considered. If there exists more than one planar feature complying with both of these requirements, algorithm checks if one of them is considerably closer. If not, both of the features might be rejected as incorrect correspondences may significantly reduce the system's accuracy. This has been achieved by comparing distances between the given point and the closest points contained in two nearest features. If

the ratio of these distance is greater than 0.7:

$$\frac{d_{1min}}{d_{3min}} > 0.7 \tag{3.5}$$

where:

d_{1min} − distance between given point and closest points in first considered feature

d_{3min} − distance between given point and closest points in second considered feature

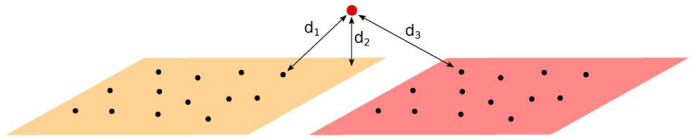
meaning that they are very similar, given point will not be assigned to any feature at all.

Summarizing, in order to add single point to the existing feature, following conditions must be fulfilled:

- Distance to existing plane (d_2) must be smaller than 0.3 m
- Distance to nearest point included in that feature (d_1) must be smaller than 1 m
- Distance to the nearest point in the second closest feature (d_3) must be greater than $0.7 \cdot d_1$

New points are assigned to existing features only if all of these conditions are met. The first of them ensures that the given feature will not grow in the direction perpendicular to the plane, whereas the second one prevents adding the feature points that satisfy the plane's equation but are located too far away. The third one assures that matching was performed with high confidence. These required distances (d_1), (d_2) and (d_3) are also shown graphically in Fig. 3.5.

Figure 3.5: Distance to the closest point (d_1), distance to the closest plane (d_2) and the distance to the second closest plane (d_3) are all considered during the addition of a point to the existing planar feature



The aforementioned parameters are responsible for the number and the size of the created features in the system. Their values were obtained as a result of a series of simulations.

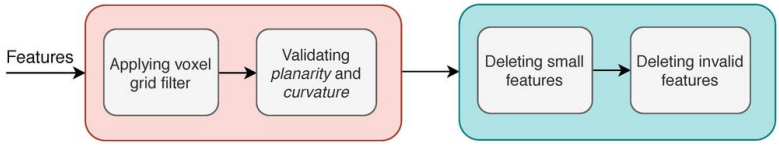
If a given point cannot be assigned to any existing feature with at least 5 points, the algorithm tries to find an existing group of points that could form one. Features smaller than 5 points have neither computed plane equation nor any other statistical parameters, so a new point can be added to them if it is just close enough. The distance between that point and the nearest point in that group must be smaller than 1 *m*.

If such a group of points was also not found, the new single-pointed feature is created. Although it consists of only one point, it might get enlarged by new ones during the processing of the rest of the scan. In other words, it creates an initial structure to which subsequent points can be added.

3.1.2 Updating and deleting features

Because of the large number of features that are created during each scan, which requires a significant amount of computation time, there is a need to reduce it. This has been accomplished by removing too small features and also by merging co-planar ones. What is more, also the number of points in each feature is reduced by applying a voxel grid filter. Subsequent steps are presented in Fig. 3.6.

Figure 3.6: Steps performed to update existing features and delete too small or invalid ones

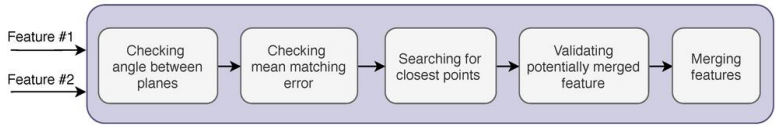


After each laser scan has been processed and all points were assigned to features, the algorithm performs all the necessary updates. In the first step, it looks for planes smaller than 5 points and marks them for deletion. Moreover, features that consist of less than 15 points after processing of 3 scans are also deleted. This prevents storing a big number of small features as they are not desirable. After that, it applies a voxel grid filter to reduce the number of points in all remaining features and also checks if they are still valid ones. For this purpose it uses parameters calculated using previously presented equations 3.2 (*planarity*) and 3.3 (*curvature*). Depending on the required accuracy their value might vary, but for the purpose of the report *planarity* threshold was set to 80% and *curvature* threshold to 0.00015, which was based on the value used in [24]. All features that do not comply with these requirements are selected to be deleted. The removal of the selected planes is done in the third step of the algorithm. Because all features are stored in a single vector, deleting one of them requires some additional operations, for example, relocation of all remaining elements. In order to reduce the time necessary during this operation, all previously selected features are deleted at once.

3.1.3 Merging features

The last stage of the algorithm has the task to perform the merge of co-planar and overlapping features. Merging algorithm also consists of a few steps (Fig. 3.7), with conditions that check if two planes are parallel to each other and if the distance between them is small enough.

Figure 3.7: Steps performed to merge two features, based on angle, matching error and distance between them



At first, angle α between two planes is calculated using following equation:

$$\alpha = \arccos\left(\frac{\mathbf{n_1} \cdot \mathbf{n_2}}{||\mathbf{n_1}|| \cdot ||\mathbf{n_2}||}\right) \tag{3.6}$$

where:

$\mathbf{n_i}$ − normal vector of the *i*-th plane

For the implemented system its threshold value is equal to 10°, which means that two planes for which this angle is bigger than 10° cannot be merged. If this condition is met, the algorithm computes mean error based on point-to-plane distances between all the points from the first feature and second plane to which they are to be merged. This procedure is also repeated in the other direction, meaning that also error between points from the second feature and the first plane is calculated:

$$\varepsilon_1 = \frac{1}{N}\sum |d_i^1|,\ i = 1...N_1 \tag{3.7}$$

$$\varepsilon_2 = \frac{1}{N}\sum |d_i^2|,\ i = 1...N_2 \tag{3.8}$$

where:

ε_1 − first calculated error

ε_2 − second calculated error

d_i^1 − distances between points from first feature and second feature's plane

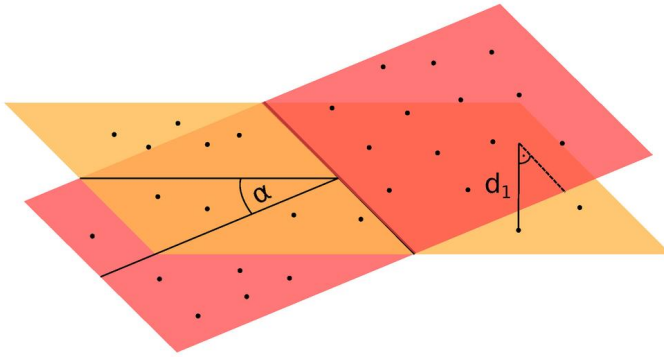
d_i^2 − distances between points from second feature and first feature's plane

N_1 − number of points in the first feature

N_2 − number of points in the second feature

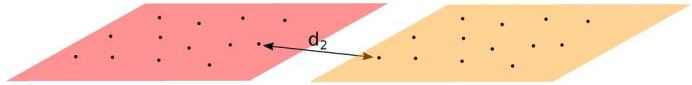
Both of these errors must be smaller than 0.1 *m* in order to continue the merging procedure. The considered angle α and exemplary distance d_1, required to calculate matching error are shown in Fig. 3.8.

Figure 3.8: Angle between two planes (α) and exemplary point-to-plane distance (d_1)



In the third step, the algorithm searches for the smallest distance between two points belonging to two different planes in order to determine if both features are located close to each other (Fig. 3.9). The found distance d_2 must be smaller than 1 *m*. Similarly, as in the case of creating features, this prevents the system from forming plane features that are inconsistent.

Figure 3.9: Distance d_2 between two points that belong to planes considered for merge

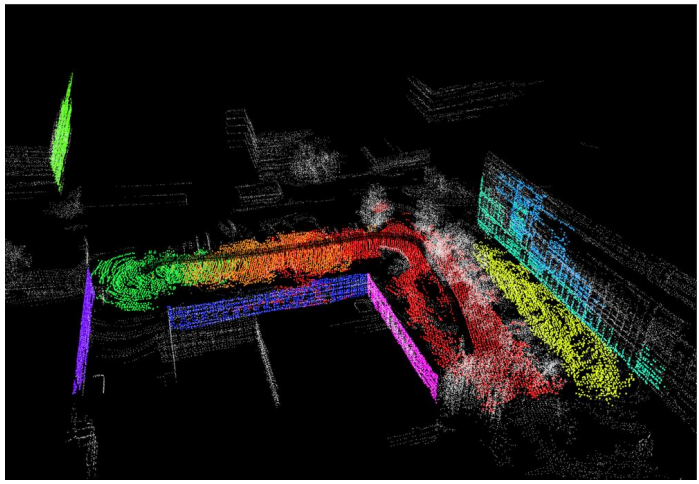


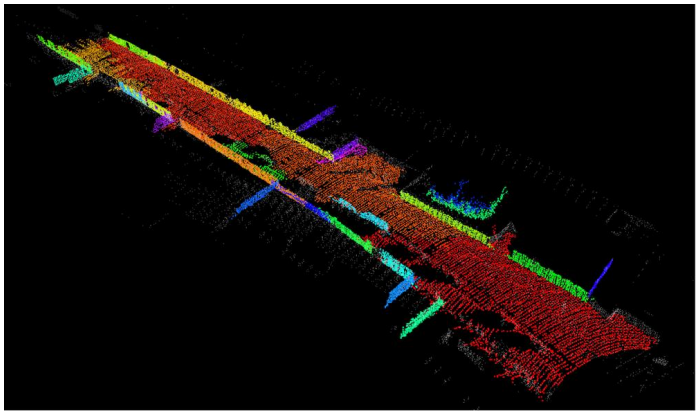
If steps above were completed successfully, two planes can potentially be merged, unless it would cause the new planar feature to be invalid. Because merging two planar features may cause the new feature to be incorrect, it is necessary to verify it. This is once again achieved by utilizing parameters like *planarity* and *curvature*. If the values of these parameters remain within an acceptable range after merging operations, then it can be performed. Otherwise, merging is abandoned, as it would cause these features to be deleted in the next iteration.

3.1.4 The example of high-level planar features

The amount and size of created features depend heavily on parameters that are used during the process of forming and merging. The most important ones, in this case, are distances that specify the maximal spacing between given points and also between points and planes. An example of biggest features created during tests (described in detail in Section 4.2) is shown in Fig. 3.10.

Figure 3.10: Example of planar features created during conducted experiments, showing that they form big planar surfaces





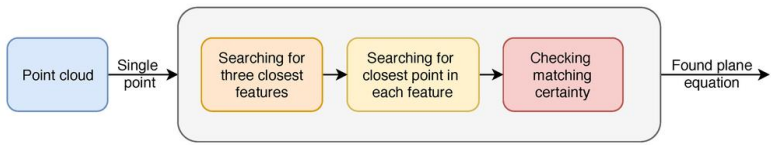
It can be seen, that biggest features are formed by objects, such as walls and road surface, even if they are not perfectly flat. Applied parameters values were chosen as a compromise between accuracy and the size of features. If these limits were too restrictive, there would be many small planes lying close to each other. This would reduce algorithm accuracy, as their equations could be incorrectly estimated. On the other hand, assigning them too big values would lead to forming spread point clouds instead of planes, also causing a reduction of accuracy. For this reason, the goal was to experimentally determine the values of parameters that will allow for covering objects, such as building walls, as single planes.

3.2 DETERMINING CORRESPONDENCES BETWEEN THE SCAN AND THE MAP

The main difference between the original and improved systems lies in the map representation.

In the case of the proposed solution, the map consists of much bigger plane features. The consequence of this is a different approach during searching for the correspondences between scan points and features from the map. These correspondences are used to determine the plane equation to which a given point will be adjusted during the optimization step. Searching for them was realized by a single function that takes at the input single point and looks through all existing features to find the best matching one. Block diagram presenting how correspondences between points and planar features are found is shown in Fig. 3.11.

Figure 3.11: Block diagram presenting steps required to find point's corresponding feature



As can be seen, correspondences, that are necessary for optimization, are calculated in a very similar way as during the process of creation of features. In order for the point to be taken into consideration during optimization, it has to comply with the same conditions, as described in Section 3.1.1. The difference lies in the values of the parameters specifying required distances, that were also selected based on conducted experiments and obtained the result:

- Distance to existing plane (d_1) must be smaller than 0.5 m
- Distance to nearest point included in feature (d_2) must be smaller than 0.6 m
- Distance to the nearest point in the second closest feature (d_3) must be greater than $0.7 \cdot d_2$

Found correspondences are used as constraints during the optimization step in order to iteratively improve the accuracy of the calculated laser scanner's pose. The

original system performs this step until the desired accuracy is reached or a maximal number of iterations is exceeded. For the original system these parameter values are:

- Maximal number of optimization iterations: 10
- Translation increment threshold: 0.05 *m*
- Orientation increment threshold: 0.05°

Because the optimization step was not changed for the purpose of the report, the mentioned values are also used in a modified system.

3.3 NOVELTY OF THE PROPOSED SYSTEM

Using features gives an advantage over points due to the fact that they create a logical structure and organize big groups of points. Thanks to that, it is possible to store only selected variables that describe the given feature. For example, big planar surfaces such as walls of buildings or roads can be represented as a single structure with highly reduced points number. Another advantage of feature-based representation of the environment is the fact, that it allows achieving better accuracy because points that can not be associated with any features or form too small groups will be rejected. To sum up, the main novel contributions of the proposed system are:

- Implementation of planar features allowing for higher-level representation of created environment map in the existing system
- Development of processing pipeline and management of features representing environment map
- Modified procedure of finding correspondences between scan points and map that makes use of the feature-based representation
- Improved pose estimation due to increased accuracy of the proposed solution

Chapter 4
Experimental evaluation

4.1 EXPERIMENTAL SETUP

The computer used in the experiments was equipped with the Ubuntu 18.04 and Robot Operating System (ROS) Melodic Morenia. The main advantage of the ROS system is the fact that it offers a large collection of libraries and packages that can be used free of charge. Thanks to that, there is no need to develop own device drivers as many of the sensors used in robotics have dedicated ROS packages. It also provides many tools for the visualization of all kinds of data. An example can be RViz that was used during experiments for displaying robots position, its odometry and data coming from sensors, such as Inertial Measurement Unit (IMU) orientation, camera image, GPS trajectory, and point clouds read from laser scanners.

Datasets used for the purpose of the report were collected using two different setups, described in the following sections. This made it possible to test the implemented system in different conditions and environments.

4.1.1 3D laser scanners

Applications, where 3D SLAM is necessary, require sensors that provide 3D data points. Examples of these sensors are RGB-D and stereo cameras or LiDAR (Light Detection and Ranging). Both of these groups provide information about the external world and can be used in the mobile robotics or automotive industry to detect obstacles or measure the speed and direction of moving objects. Each group has its pros and cons that arise from its principle of operation. LiDARs are much less susceptible to light conditions as they emit their own light and measure the time necessary to cover the distance from the sensor to object back and forth. This means that they can operate at day and night without any significant difference. What is more, they are characterized by robustness to interference from sunlight or headlights of another vehicle in contrast to vision systems. On the other hand, most cameras, excluding monochrome ones, are able to detect colors, which can be very useful in some applications. Additional color information is useful to detect traffic or brake lights and can improve road signs recognition. Cameras are also much cheaper compared to LiDARs that can cost tens of thousands of dollars at present.

The report focuses only on LiDAR systems and sensors used for this purpose are Velodyne VLP-16 and Sick MRS-6124, presented in Fig. 4.1.

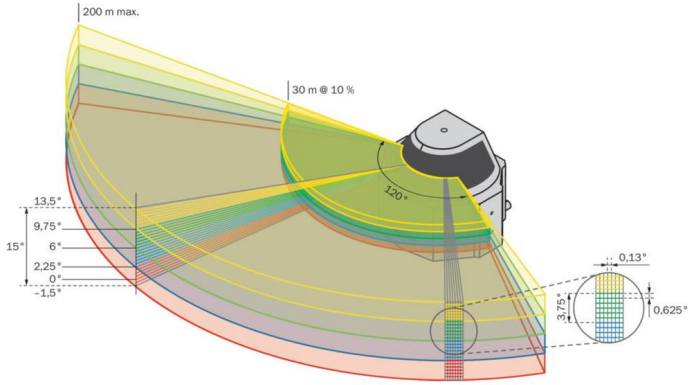
Figure 4.1: Sick MRS-6124 [27] (left) and Velodyne VLP-16 [31] (right)

Table 4.1: Comparison of Sick MRS-6124 and Velodyne VLP-16

Parameter	Sick MRS 6124	Velodyne VLP-16
Horizontal angle	120°	360°
Vertical angle	15°	30°
Horizontal resolution	0.13°	0.2°
Vertical resolution	0.625°	2°
Scanning lines	24	16
Scanning groups	4	1
Points per second	221 520	288 000
Maximal range	200 m	100 m

The main differences between scanners are the field of view and the density of measurements. As can be seen in Tab. 4.1, Velodyne VLP-16 has 360° horizontal and 30° vertical field of view, while in case of Sick MRS-6124, these values are limited to 120° and 15° accordingly. Velodyne's higher and wider field of view is achieved at the expense of smaller point density, which can be problematic in applications such as mobile robotics or autonomous cars. In order to reliably detect and recognize smaller or narrower objects, like for example lamppost and the human body, a maximum available resolution is desired. Another parameter that gives an advantage in favor of Sick MRS-6124 is its higher maximum range. The difference is also noticeable in the acquisition of the measurement of both scanners. Velodyne VLP-16 provides data from 16 scanning lines that are sent line by line in the top-bottom order. In comparison, data coming from the Sick MRS-6124 scanner is divided into 4 groups that are scanned one after the other. Each group consists of 6 scanning lines that are registered at the same time. Altogether it gives 24 scan layers, which translates to 0.625° vertical resolution, what is also shown in Fig. 4.2.

Figure 4.2: Scan layers and groups of Sick MRS-6124 [27]



Because of the fact, that scanner simultaneously registers points only from a single group, measurements coming from subsequent groups are shifted in time. This makes it difficult to eliminate the influence of scanner movement on measurements as points in different scanning groups will be registered in different moments. Although both scanners are suitable for SLAM purposes, Sick MRS-6124 was chosen for the developed system. It turned out to be better suited for automotive purposes than Velodyne VLP-16, despite the fact, that it has only 120° horizontal field of view. As mentioned earlier, the reason for this is the better resolution in both horizontal and vertical axes. This results in a higher density of points detected in front of the vehicle, which is the most important spot in terms of automotive applications. The comparison of the laser-based system using the data from both scanners is described in Section 4.3.1.

4.1.2 Mobile robot used in the experiments

The mobile robot used for collecting the first datasets is shown in Fig. 4.3. It has four independent motors coupled with every wheel and uses differential steering, meaning that it takes turns by changing the rotational speed of opposing wheels. The robot used in experiments was equipped with two laser scanners mounted at the top of aluminum construction. The role of the computer unit was temporary fulfilled by a laptop placed on the robot's base. Owing to the fact that scanners were mounted on a wheeled vehicle, they could be mounted at a fixed angle. Velodyne VLP-16 was mounted parallel to the ground, so that it scans all the surroundings horizontally. Sick MRS-6124 was mounted upside down at slightly upward angle, equal to 6°.

Figure 4.3: Velodyne VLP-16 and Sick MRS-6124 mounted on the mobile robot that was used in the experiments

Although this setup made it possible to verify and test if original LOAM software works correctly with both laser scanners, there was no GPS reference trajectory to compare. Another problem that occurred during experiments was associated with robot construction. Because scanners were mounted at the top of the robot its center of gravity was lifted and it turned out to be unstable during acceleration and turns.

4.1.3 Final sensor setup used in the experiments

The next step was to create a more advanced sensor system that will provide some additional data for the purpose of testing developed software. The most needed one was the reference trajectory that could be compared with odometry provided by the LOAM algorithm to draw conclusions about the accuracy of the system. To achieve this, the GPS receiver was used. Other sensors that were added in comparison to the initial setup are RGB camera and IMU. The created setup is shown in Fig. 4.4.

Figure 4.4: Final sensor setup consisting of Sick MRS-6124, Ublox LEA-6H GPS receiver, PointGrey Flea3 camera and Xsens MTi-30 AHRS

Apart from the base frame, that is equipped with three suction cups that allow mounting the whole structure on any flat surface, setup consists of the following elements:

- Sick MRS-6124

  The only scanner used in all the following experiments was Sick MRS-6124. Its mount has the ability to adjust the tilt angle if needed, depending on the requirements. A higher downward angle enables to detect more points on the road. On the other hand, if the goal is to detect also walls of the surrounding buildings, the tilt angle value should be closer to 0°. During conducted experiments, two values of downward tilt angles

were tested: 0° and 4.5°, but no significant difference between them was noticed.

- Ublox LEA-6H GPS receiver

  Due to the lack of Differential Global Positioning System (DGPS), which can provide few centimeters accuracy, only standard GPS receiver was used in the described setup. Although it offers much lower positioning precision, fluctuating around 4 *m*, it provides sufficient reference trajectory, especially for longer distances.

- PointGrey Flea3 FL3-U3-13E4C-C camera with Kowa LM3NCM 1/1.8" 3.5 mm F2.4 lens

  The PointGrey camera was used initially for debugging purposes, as it allows to peek what objects are within scanners range at a given moment. What is more, it also makes it possible to visually verify what happened at the selected moment without the need of generating GPS trajectory. Except that, it can be potentially used for much more complex purposes, for example in combination with laser scanner for object detection, data segmentation or localization. It provides additional data about the environment, such as objects colors and shapes, that can not be obtained using only scanners. What is more, it has a much wider field of view, which gives it an advantage in detecting very big objects, especially close ones.

- Xsens MTi-30-AHRS-2A8G4

  The described setup was also equipped with Xsens AHRS, but eventually, data from this sensor was not utilized. However, it gives the potential possibility to compensate for sudden movements caused by bumps and roughness of the road surface. It can also be used to provide data for fusion with other sensors and thus enhances the accuracy of odometry and mapping.

The main advantage of the built setup is its compactness and the fact, that all the sensors are mounted at a fixed position relative to each other. Thanks to that, there is no need to repeat the process of calibration every time it is used, but instead, it can be performed only once.

For the purpose of outdoors tests, a laser scanner, with all the other sensors, was mounted on the roof of the car, what is shown in Fig. 4.5a. This solution has an advantage over the mobile robot setup because of a much more steady attachment and variable tilt angle. It also gave the possibility to validate LOAM software with a higher velocity of the vehicle.

Figure 4.5: Sensors mounted on car's roof (left) and bus (right)

The experiments were also conducted in the road traffic and urban areas. As mentioned at the beginning of the report, a similar system is developed for the purpose of implementing ADAS on electric city buses, that are about to be produced in the near future. Because of that, it was possible to utilize datasets collected during a test with the city bus. Similarly to previous ones, all the sensors were mounted at the top of the vehicle, as shown in Fig. 4.5b.

4.2 RECORDED SEQUENCES

In order to test and verify both original and implemented system's accuracy, several datasets in different locations were recorded. This made it possible to compare the performance of the system in different conditions. Tests were held indoors and outdoors as the results were expected to vary depending on the properties of the surroundings. This section provides a general overview of the conducted experiments and describes how and where all data were collected, while the analysis of the obtained results is presented in Section 4.3. The summary of collected sequences used for the analysis of the system can be found in Tab. 4.2.

Table 4.2: Recorded sequences selected for the analysis

Name	Duration [s]	Used scanner	Environment	Setup
PUT-robot	210	Sick, Velodyne	outdoor	mobile robot
PUT-indoor	101	Sick	indoor	handheld
PUT-car	167	Sick	outdoor	car
MG-centre	100	Sick	outdoor	bus
MG-service	228	Sick	outdoor	bus
MG-suburbs	135	Sick	outdoor	bus

The first sequence was recorded in the outdoor environment at PUT (Poznan University of Technology) campus using a mobile robot setup. Its main purpose was to compare Sick MRS-6124 and Velodyne VLP-16 scanners to decide which of them is more suitable for automotive application on the city bus. From this point, all the following datasets were recorded using an updated setup presented in Fig. 4.4. It includes tests conducted in the indoor environment (PUTindoor), which were performed in the Centre of Mechatronics, Biomechanics and Nanoengineering building, located at PUT campus. During these experiments, all the sensors were hand-held due to the lack of a smaller and more maneuverable mobile robot. The next sequence (PUT-car) was recorded also at PUT campus, but this time all the sensors were mounted on the roof of a car. The surrounding was mainly a car park located in between buildings.

The last three sequences (MG-centre, MG-service, MG-suburbs) were recorded in Murowana Goślina, a small town located nearby Poznań, with the sensors placed on the roof of a bus. The MG-centre dataset was collected in the city cen-

ter including a town square and relative narrow streets with buildings on each side. In a similar, but more suburban conditions the MG-suburbs dataset was registered. The MG-service sequence was obtained at the SBC service place, which imposed that the bus was driving with significantly lower velocity compared to the previous ones.

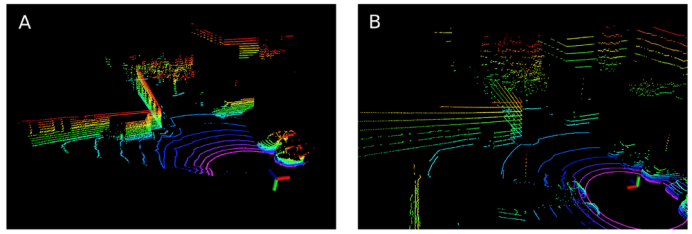
4.3 ACCURACY ANALYSIS

The accuracy analysis was performed on the systems with disabled optimization of edge features. This applies to both original and modified LOAM. Without edges, the systems utilized only planar features, which highlights the impact of the proposed modifications. This made it possible to compare both systems based on results obtained using only planar points, but it is worth to mention that it also reduced accuracy to a certain degree.

4.3.1 Outdoor tests with the mobile robot

The first dataset was recorded using a mobile robot at PUT campus. As mentioned earlier, its point was to initially compare two laser scanners and determine which would be a better solution for the developed system. Examples of single scans registered using both laser scanners are shown in Fig. 4.6.

Figure 4.6: Comparison between scans captured with Sick MRS-6124 (A) and VLP-16 (B)



It can be seen that Sick MRS-6124 provides much more dense measurement points in comparison to Velodyne VLP-16 as could be also deduced based on both scanner specifications contained in Section 4.1.1. Velodyne VLP-16 has an advantage of 360° field of view but it cannot be used to reliably detect small or narrow objects like tree trunks and lamppost on single scans. It also provides fewer points measured at the ground surface in front of the sensor, which may lead to the SLAM system drift in the vertical axis. On the other hand, it is independent of drive direction or robot orientation, as long as it is mounted flat. Thanks to that, Velodyne VLP-16 could be used during maneuvers that require driving in reverse and generally is much less susceptible to mount location.

Data from both of these sensors were recorded in the PUT-robot dataset, which made it possible to compare LOAM system results obtained with different scanners. The video presenting the working system during this sequence is available online[1]. The comparison of registered trajectories of the mobile robot and calculated error are shown in Fig. 4.7 and Tab. 4.3 accordingly.

1 https://youtu.be/peNifYDEM1Y

Figure 4.7: Comparison of trajectories obtained using Sick MRS-6124 and Velodyne VLP-16 for PUT-robot dataset (A) and robot's trajectory drawn on the Google Map (B)

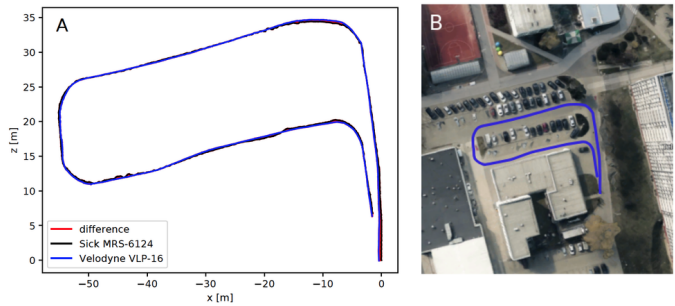


Table 4.3: Value of error between Sick MRS-6124 and Velodyne VLP-16 trajectories obtained using original LOAM for PUT-robot dataset

Error type	Value [m]
RMSE	1.06
Min	0.05
Max	2.03

For the purpose of plotting trajectories and calculating error between them, an existing SLAM evaluation script [28] was used. It is publicly available[2] and makes it possible to compare both system's accuracy. Firstly, it associates pose pairs based on recorded timestamps and afterward uses SVD to align both trajectories, so that the computed error takes the minimal value.

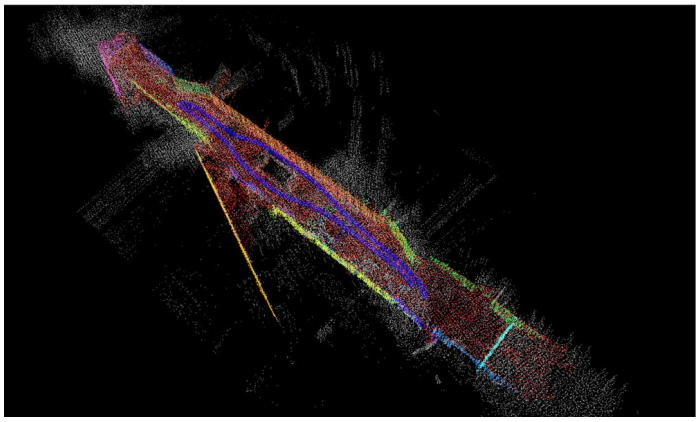
As can be seen, both scanners provide similar results. A Root Mean Square Error (RMSE) between two obtained trajectories for the PUT-robot sequence is equal to 1.06 *m*, which translates to 0.65% difference, as the sequence total distance was 162.58 *m*. This is also evidence of the versatility of the laser SLAM system, as it can be adapted to work with different scanners.

2 https://vision.in.tum.de/data/datasets/rgbd-dataset/tools

4.3.2 Indoor tests at PUT

For the purpose of testing the implemented algorithm in the indoor environment, one of the datasets was recorded inside the building. Due to the fact, that there are much more smooth plane surfaces than outdoors, the created system was expected to work more precisely. This is because implemented feature-based algorithm approximates all detected surfaces as planes, and subsequently uses these points for optimization. An example of plane features that were created during the indoor test is shown in Fig. 4.8 and also in the video attachment[3].

Figure 4.8: Features created during indoor environment tests (PUT-indoor dataset)



On the other hand, the used laser has a minimum range limited to approximately 0.5 *m*, so this can be a problem in very narrow corridors. Another difficulty might be caused by a fact, that all sensors were held in hands during these tests, as the mobile robot would be hard to operate indoors. Because of that, laser scanner and the rest of the sensors could not be mounted at a fixed tilt angle, but instead were vulnerable to movement caused by human steps. Unfortunately, there was also no reference signal to verify the results, as GPS could not be used. For this

3 https://youtu.be/pJKxojGmP_Q

reason, the analysis of the indoor sequences is reduced to a comparison of the original and modified LOAM system's trajectories presented in Fig. 4.9 and Tab. 4.4.

Figure 4.9: Error between original and modified system trajectories for PUT-indoor dataset

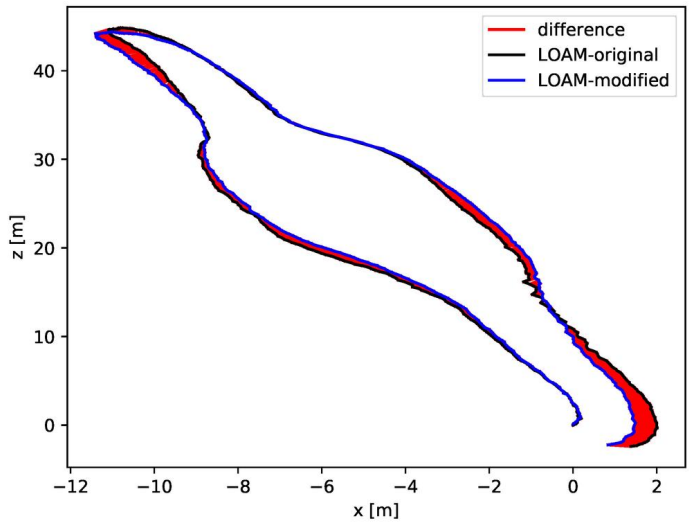


Table 4.4: Value of error between the original and modified system for PUT-indoor dataset

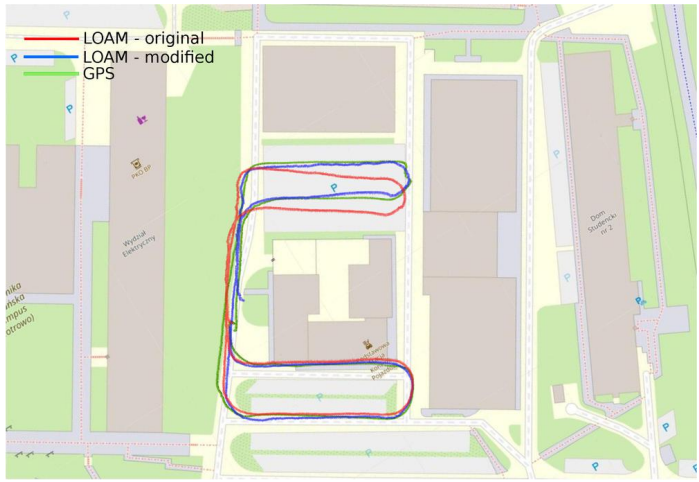
Error type	Value [m]
RMSE	0.41
Min	0.03
Max	1.44

The difference between original and modified LOAM system in an indoor environment is unremarkable, as output trajectories are very similar. The RMSE calculated between them at a distance of 107.08 *m* is lower than 0.5 *m* and it is hard to determine which systems give better results without any ground truth trajectory. Nevertheless, it can be assumed that both systems can operate to some degree in indoor conditions.

4.3.3 Outdoor tests with the car setup

Successive tests were conducted at PUT campus car park with the use of the sensors attached to the car's roof, as described in Section 4.1.3. The recorded trajectory is shown in Fig. 4.10. It also presents trajectories obtained by original LOAM and GPS, which was used as a reference. The video presenting a working system for this dataset is available in the attachment[4].

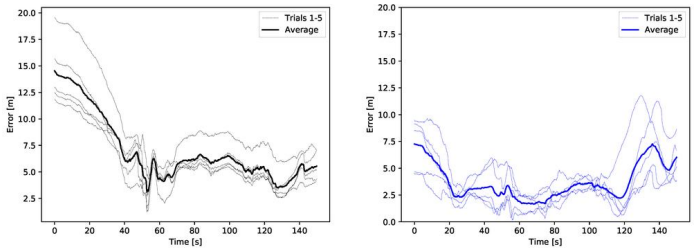
Figure 4.10: Sequences recorded during outdoor test (PUT-car dataset)



Obtained results from both modified and original system are shown in Fig. 4.11.

4 https://youtu.be/NGi4rnJSzlM

Figure 4.11: Translational error relative to GPS data for original (A) and modified (B) LOAM system for PUT-car dataset



(A) LOAM - original (B) LOAM - modified

Fig. 4.11a presents the error between trajectory from GPS, taken as the reference ground-truth signal, and the original LOAM algorithm. The graphs show errors calculated from five trials. As can be seen, results from different trials are not identical but are repeatable to a considerable extent. This non-deterministic behavior may be the result of multithreaded execution of *odometry* and *mapping*, which are synchronized in different moments. Another reason might be the high Central Processing Unit (CPU) load, which is caused especially by *mapping* process. It can potentially make it hard to perform all calculations on time, which leads to different results across many trials.

The continuous and bold line presents averaged results. Same data coming from implemented, feature-based system is shown in Fig. 4.11b. Third graph (Fig. 4.12) compares the values of errors on one plot and Tab. 4.5 presents three selected statistical parameters of the calculated error course.

Figure 4.12: Comparison of original and modified system's localization error relative to GPS data, calculated as average of 5 trials (PUT-car dataset)

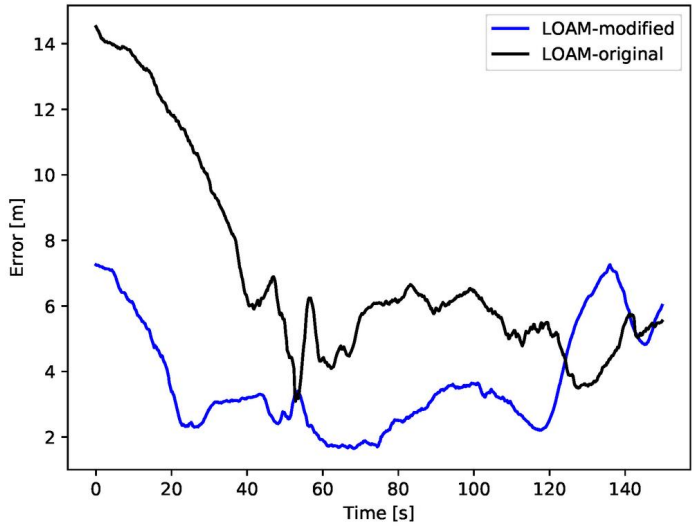


Table 4.5: Values of localization error for original (A) and modified LOAM (B)

(A) LOAM original

Error type	Value [m]
Mean	7.01
Min	3.09
Max	14.52

(B) LOAM modified

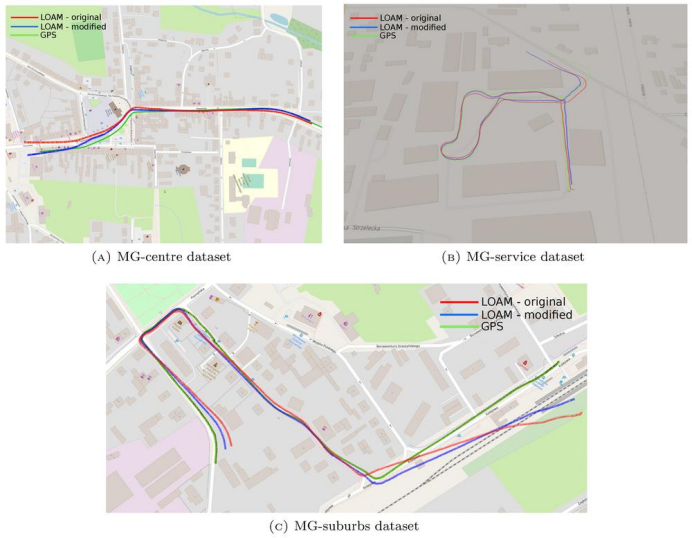
Error type	Value [m]
Mean	3.70
Min	1.65
Max	7.26

It can be observed that the original LOAM has significantly higher error. Its value is the biggest at the very beginning, despite the fact, that all generated trajectories start with position $x = 0$, $y = 0$, $z = 0$. This is the result of trajectory path alignment performed by the script mentioned earlier. The higher drift of the original system might be caused by inaccurate correspondences matching, especially on big, rough surfaces like asphalt or cobblestone road. One of the biggest, noticeable problems of both LOAM versions is its susceptibility to drift in the vertical axis, even if there are no hills or valleys at a given moment. It could be potentially fixed by utilizing IMU to compensate for sudden motions. Alternatively, degree of freedom in that axis could be reduced, but in that case, the system would work properly only on flat, lowlands terrains.

4.3.4 Outdoor tests with the bus setup

The final experiments were conducted in Murowana Goślina, a small town located nearby Poznań. They were conducted with the use of sensors that were mounted on the bus. Selected sequences that were recorded and used for analysis are shown in Fig. 4.13. Additionally, the MG-centre dataset is presented in the video attachment[5].

Figure 4.13: Comparison of the LOAM, modified LOAM and the GPS trajectories recorded during outdoor tests in Murowana Goślina town with the bus setup



(A) MG-centre dataset

(B) MG-service dataset

(C) MG-suburbs dataset

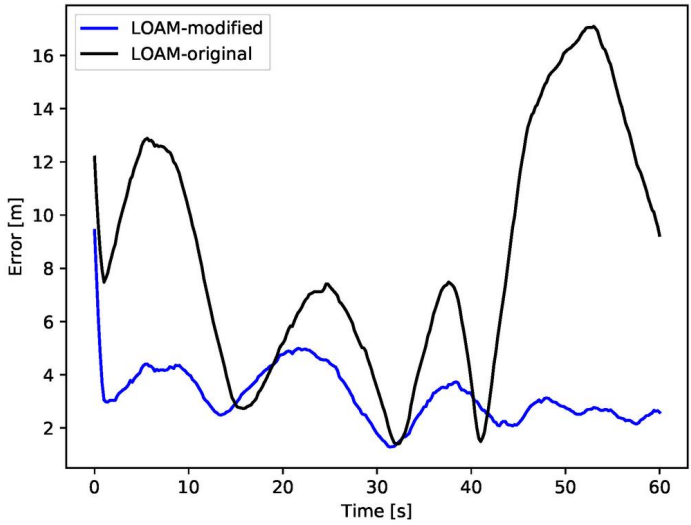
The trajectories marked on the images were recorded separately and overlaid on the Open-StreetMap with the use of *rviz_satellite*[6] ROS package. The discrepancy between starting points locations is again a result of trajectories alignment.

5 https://youtu.be/Mj569vpXq9w

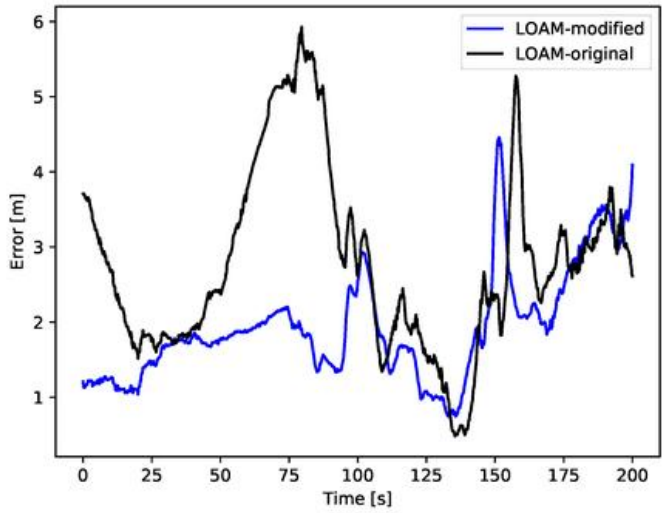
6 https://github.com/gareth-cross/rviz_satellite

More accurate analysis and error comparison are shown in Fig. 4.14.

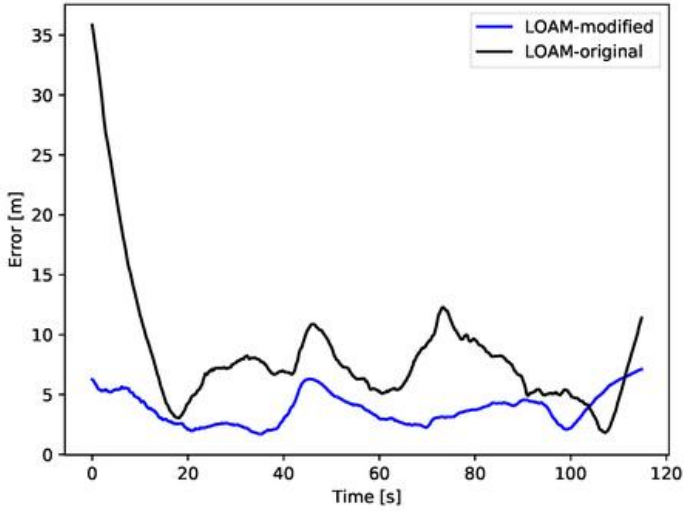
Figure 4.14: Comparison of original and modified system's localization error relative to GPS



(A) MG-centre dataset



(B) MG-service dataset



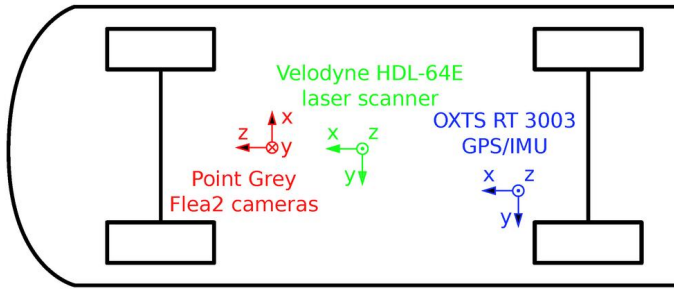
(C) MG-suburbs dataset

As can be seen, similar results were achieved while analyzing localization errors from other datasets. The smallest discrepancy between original and modified system is noticeable for MG-service sequence. It might be caused by the fact, that during this test bus was driving with the significantly smaller velocity, equal to around 6 km/h.

4.4 PUBLICLY AVAILABLE KITTI DATASET

The KITTI dataset [8] is one of the most popular validation and testing benchmark for camera-based or laser-based localization systems. The recorded dataset includes data recorded from two, grayscale and color, stereo PointGray cameras, the Velodyne HDL-64 laser scanner and OXTS RT3003 IMU/GPS navigation system with Real Time Kinematics corrections that is used as a ground truth reference. All sensors were mounted on the roof of the car, as shown in Fig. 4.15.

Figure 4.15: Sensory setup used for recording KITTI dataset, consisting of four video cameras, Velodyne HDL-64 3D laser scanner and a combined GPS/IMU inertial navigation system



The data was recorded in the city of Karlsruhe and its surroundings. Recorded sequences are divided into 3 main categories: *city*, *residential* and *road*. KITTI dataset is commonly used for machine learning purposes associated with automotive needs, such as semantic segmentation or object detecting and tracking. Due to the fact, that it contains both camera images and laser scans, it is also suitable for visual and laser SLAM system validation. The utilized laser scanner has the advantage of 64 scanning lines and higher resolution than sensors described in Sec. 4.1.1. The full specification of the Velodyne HDL-64E laser scanner is shown in Tab. 4.6.

Table 4.6: Specification of Velodyne HDL-64E LiDAR used in KITTI dataset

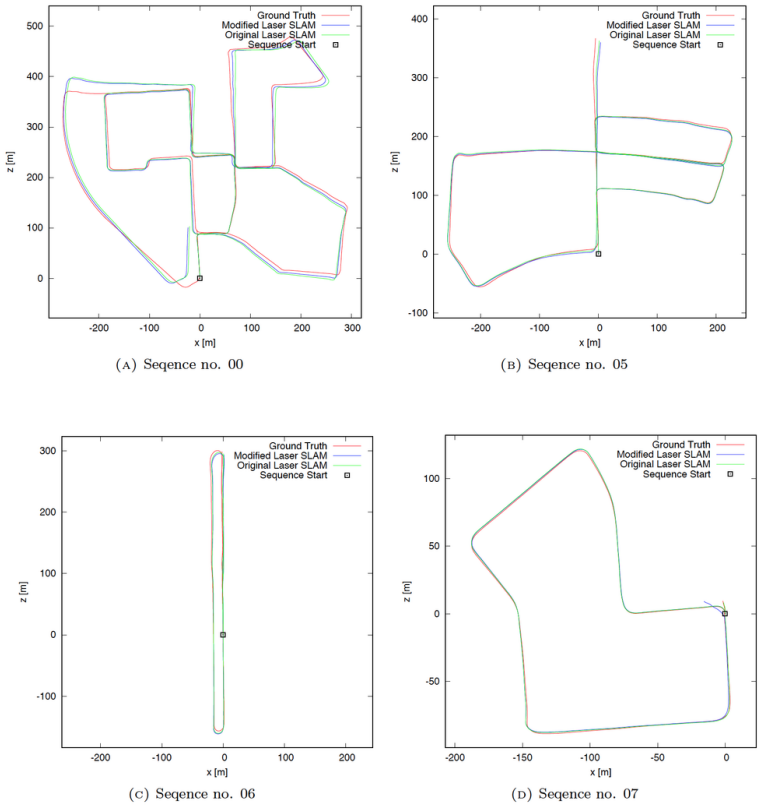
Parameter	Velodyne HDL-64E
Horizontal angle	360°
Vertical angle	26.9°
Horizontal resolution	0.09°
Vertical resolution	0.4°
Scanning lines	64
Scanning groups	1
Points per second	1.3 M
Maximal range	120 m

The KITTI benchmark consists of 22 sequences, 11 of which are provided with GPS reference trajectory for training possibilities. The remaining sequences are used for evaluation and for that reason they come without public GPS ground truth. For the purpose of validating the developed system, the sequences no. 00, 05, 06 and 07 were chosen. All of them are labeled as *residential* sequences and were recorded in the suburban environment. In order to make it possible to utilize the developed SLAM system in ROS, it was necessary to convert selected raw data sequences[7] into .bag files. For this purpose, the *kitti_to_rosbag*[8] package was used. The result evaluation was performed using the KITTI benchmark script, which made it possible to calculate both translation and rotation error and also plot ground truth and obtained trajectory. The result are presented in Fig. 4.16 and in Tab. 4.7.

7 http://www.cvlibs.net/datasets/kitti/raw_data.php

8 https://github.com/ethz-asl/kitti_to_rosbag

Figure 4.16: Ground truth, original and developed laser SLAM trajectories generated by KITTI evaluation script for selected sequences



(A) Seqence no. 00

(B) Seqence no. 05

(C) Seqence no. 06

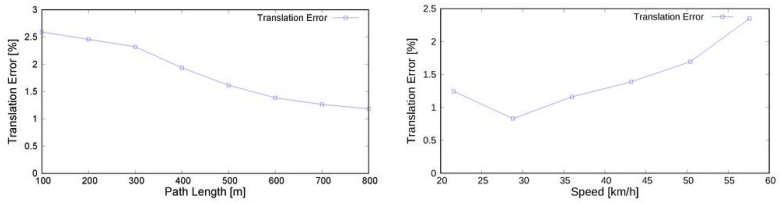
(D) Seqence no. 07

Table 4.7: Average translation and rotation error computed for selected sequences

	(Modified LOAM)		(Original LOAM)	
Sequence no.	Translation error	Rotation error	Translation error	Rotation error
00	1.97 %	0.0110 °/m	2.77 %	0.0119 °/m
05	1.91 %	0.0118 °/m	1.88 %	0.0112 °/m
06	1.33 %	0,0074 °/m	1.26 %	0.0072 °/m
07	1.49 %	0.0195 °/m	1.72 %	0.0186 °/m

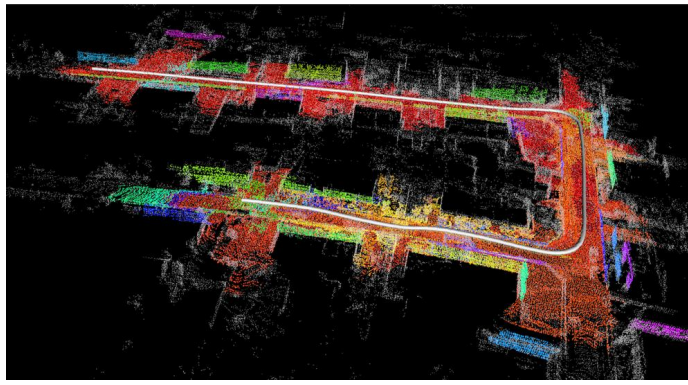
It can be seen that the average translation error in the case of a modified system for sequences no. 00 and 05 is smaller than 2% and under 1.5% for sequences no. 06 and 07. The trajectories obtained using original and modified LOAM are very similar. The biggest difference is noticeable in the case of sequence no. 00 in favor of the developed system. The results depend not only on the surrounding environment that varies across different sequences but also on the path length and vehicle speed. Influence of these factors is presented in Fig. 4.17.

Figure 4.17: Translation error plotted against path length (left) and vehicle speed (right) for exemplary sequence no. 5 (left) and no. 6 (right)



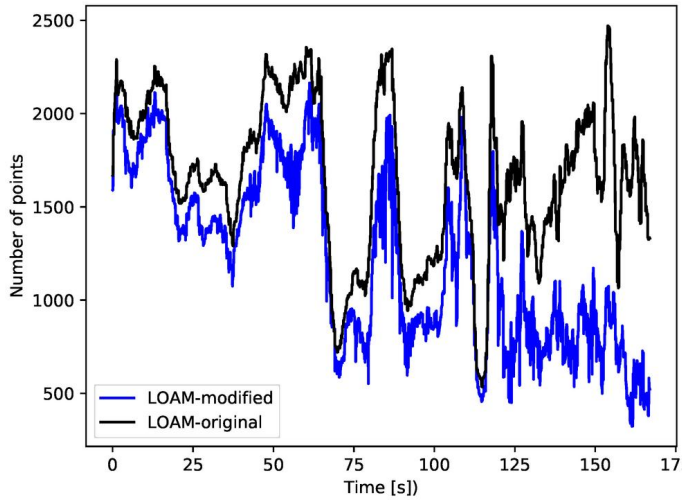
Based on the above results, it can be noted that the translation error decreases with the path length, which might be caused by the inaccurate pose estimation right after system initialization. It is also significantly affected by the velocity of the car, as the average localization error increases with increasing driving speed. It is worth to mention that the developed system is not intended to work in every environment as it requires many objects with preferably big planar surfaces such as buildings. These requirements were fulfilled by sequences that were selected for evaluation. An example of a created map using data from the KITTI dataset and consisting of the planar features can be seen in Fig. 4.18.

Figure 4.18: Example of planar features created using laser scans from KITTI dataset



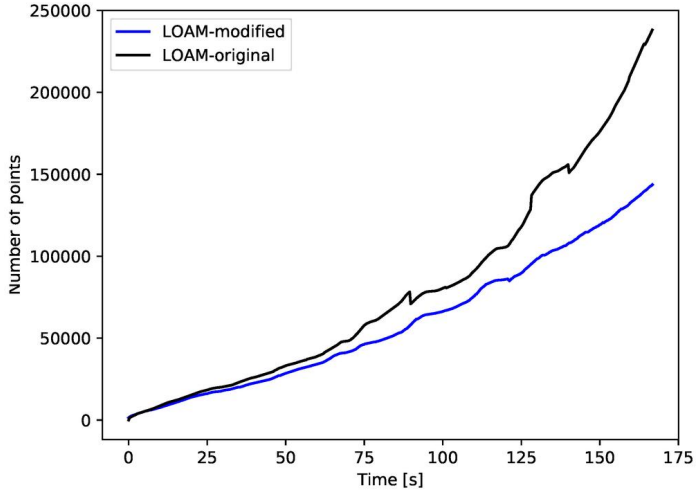
The feature-based system has also another advantage over the original LOAM system. It requires a smaller number of features to represent the same environment on the map. This results from the fact that calculated constraints, necessary for optimization steps, are more restrictive and thus more points are rejected. This also improves algorithms accuracy, as points that are not fitted into any bigger surface will not be taken into consideration. The number of planar points used for optimization during the execution of the algorithm for the selected dataset (MG-service) is shown in Fig. 4.19.

Figure 4.19: Comparison of the number of points used for the pose optimization



It can be noticed, that the number of points used in modified software is usually smaller by 200−500. On the other hand, this number cannot be too small, as having not enough points might lead to increased drift in any axis. Reduced number of stored points makes also the whole map smaller, what is presented in Fig. 4.20.

Figure 4.20: Comparison of the sum of planar points that create environment map in original and modified system



The shown graph presents a situation, where none of the past fragments of the map are deleted from memory. Although it is easily possible to deploy such a feature in both algorithms, this would prevent to implement loop closure and could decrease accuracy in some cases. To sum up, a smaller number of points in the modified system allows not only to reduce memory usage but also contributes to faster searching of correspondences and speeds up optimization steps.

4.6 TIME ANALYSIS OF THE 3D LASER SLAM SYSTEMS

One of the most important aspects of considered SLAM system, alongside with its accuracy, is the performance in terms of computational speed. It should be capable of real-time operation as most of the applications require all data to be calculated online. This is why this section compares the time necessary for subsequent steps of the algorithm. The most time-consuming part of both original and modified LOAM is calculating constraints for optimization. It includes searching for the nearest neighbors for all points in the scan. In the original system, it is accomplished by utilizing a k-d tree search, while modified one uses standard search performed in the loop. This turned out to be the bottleneck of the modified system, despite the fact that a large part of points can be rejected from the search, based on a point-to-plane distance between points and features, as described in Section 3.2. Point-based optimization, implemented in the original LOAM, requires to find correspondences using raw points. The computational complexity of this problem, including the process of construction and searching through the k-d tree, is expressed by the following equation:

$$O(n \cdot \log n + k \cdot m \cdot \log n) \tag{4.1}$$

where:

k − number of optimization iterations
m − number of points in scan
n − number of points in map

Using features in the modified system makes it possible to reduce the number of points that can be potentially matched, but it requires some additional calculations. In this case, complexity can be expressed by the formula:

$$O(k \cdot m \cdot l + k \cdot m \cdot p) \tag{4.2}$$

where:

k − number of optimization iterations
l − number of planar features
m − number of points in scan
p − average number of points in a single feature

Thanks to the feature-based representation of the map, correspondences are

found in two steps. At first, only the closest features are found, which makes it possible to reject most of the points from further calculations. This step is accomplished based on point-to-plane distance and its time is directly proportional to both the number of points in scan and features. The second step requires calculating distances to each point in a given feature. Because of the fact, that the average number of points in features is much smaller than the number of all points in the map, it can be potentially accomplished relatively fast. Unfortunately, as mentioned earlier, the implemented solution is based on a simple search loop, as it was not optimized in terms of computational speed, which extends calculations time in comparison to the k-d tree algorithm.

To compare both systems experimentally, the time necessary for each step was measured. The time analysis of the original system's *odometry* and *mapping* node are shown in Fig. 4.21 and Fig. 4.22 accordingly.

Figure 4.21: Time analysis of individual steps performed in the LOAM *odometry* thread

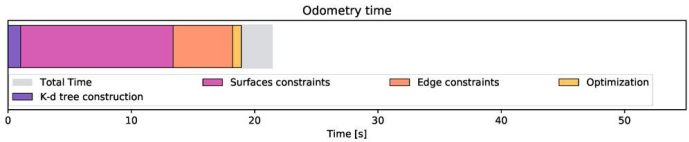
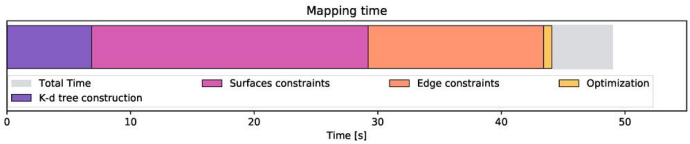


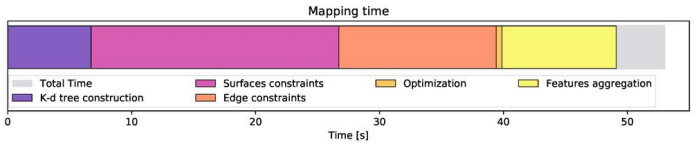
Figure 4.22: Time analysis of the individual steps performed in the LOAM *mapping* thread



Presented values are total times, calculated as the sum of time spent on a given step during the whole run. Even though *odometry* is executed with ten times greater frequency (10 *Hz*) than *mapping* (1 *Hz*), it consumes nearly two times less CPU cycles. This is due to much more accurate calculations that are performed in *mapping*. It is worth noting, that optimization is the least time-demanding step from all presented.

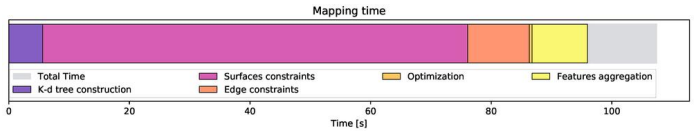
As far as a modified version of LOAM is concerned, the first goal was to check how much processing time will be lengthened after adding and storing planar features. Additional time is marked with yellow color in Fig. 4.23.

Figure 4.23: *Mapping* process time with additional *features aggregation* step performed for the needs of the modified system



It can be seen that this modification extends total mapping time by around 20%. Unfortunately, using created features for optimization takes a much longer time. This is because of search time that is necessary to find points correspondences using simple loops for searching through all points in a given feature. What is more, this time is also multiplied by the number of iterations used during the optimization step, so finally, whole mapping time turned to be around two times longer than originally, what is presented in Fig. 4.24.

Figure 4.24: Time analysis of the individual steps performed in the modified LOAM *mapping* thread



Although this has not been accomplished in the system described in the report, the calculated time might be potentially reduced. It could be achieved by using the same k-d tree method for finding the point's closest neighbors in a given feature.

Chapter 5
Conclusions

5.1 SUMMARY

The report describes in detail implemented feature-based modifications and presents a developed solution, which improves the existing laser SLAM system. It gives an overview of the whole process of creation and management of features that replaced the original point-based representation of the environment map.

It also presents experiments that were conducted in order to verify if the created system works in line with expectations. Experiments were performed using a custom sensor setup that made it possible to record test sequences in different conditions, including both indoor and outdoor surroundings. System evaluation was also performed using publicly available KITTI benchmark.

The report contains also an analysis of the results and a comparison of the modified and original systems. The comparison was performed taking into consideration obtained accuracy and also general characteristic of both systems, including the number of points stored in the map and used during the optimization step. What is more, it contains time analysis of subsequent steps, as this is one of the most important aspects of such a system aside from accuracy.

5.2 CONCLUSIONS

The feature-based system has the potential of improving accuracy as it reduces errors in scanners measurements. Features are created using a greater number of points, which translates into better precision of calculated planes equations. The downside of the developed system is increased processing time that is the result of non-optimized points operations. It could be potentially reduced and improved by using efficient 3D space decomposition such as k-d tree, implemented in the original system.

Conducted experiments showed that as far as accuracy is considered, the modified system achieved better results in all recorded sequences. The proposed system is also characterized by a lesser number of points that are used for pose optimization. This is due to the fact that constraints are calculated in a more restrictive manner. Thanks to that, correspondences are computed more precisely, thus provide better accuracy and results. The tests were performed not only at PUT campus, but also on a public road to verify its efficiency in traffic conditions. Although comparison was based on the ground truth trajectory obtained using a standard GPS module, it can be concluded that it was a sufficient solution for the recorded sequences, especially long-distance ones. Tests confirmed also that it is possible to use laser scanners for localization purposes in the indoor environment.

It is worth to mention that developed system performance substantially depends on selected parameters values, as they have a significant influence on size and number of created features. Therefore, inappropriately selected ones might cause the system to work unreliably. For this reason, a significant amount of time was spent to select universal parameters values that were finally utilized and are suitable for different kinds of environments.

5.3 FUTURE WORK

The developed system gives a more accurate localization estimate than the original one, but there is room for many improvements in terms of the system's performance. First of all, it can be improved in terms of the processing pipeline. During conducted experiments, it turned out that localization drift is caused by points measured on moving objects like grass or leaves. One potential solution for this could be integrating laser scans points with the camera image. This would enable the system to perform scans segmentation based on additional information such as RGB values of measured points. Thanks to that, it would be possible to reject unreliable measurement that causes localization error.

The second area for improvement would be the accuracy of odometry. It is prone to drift, which is caused for example by a sudden movement of the laser scanner on the road bumps. One of the exemplary approach to eliminate it would be utilizing some additional sensors like IMU to detect unexpected motions. Moreover, it could also use the GPS receiver to compensate for the error in the long term sequences. The most common way to fuse data coming from those sensors would be to implement Extended Kalman Filter [15], as it is successfully used in many mobile robotics [5] and SLAM [1] applications.

What is more, the report focuses only on planar features that are used to represent a map of the environment and to find optimization constraints. However, with relatively small effort, this idea could also be transferred to implement other types of features such as edges.

The last proposed modification would be to use differential GPS receiver instead of standard one as a source of reference ground truth trajectory. Despite the fact, that there was available such a receiver that could be lent for the needs of the report, it was impossible to obtain high accuracy localization mode due to problems with communication between base and rover stations. For that reason, only the standard GPS could be used. Although it was possible to verify the efficiency of the developed system and compare it to the original one, the results would be much more trustworthy in the case of DGPS.

Bibliography

[1] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot. Consistency of the ekf-slam algorithm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3562–3568, Oct 2006.

[2] J. Będkowski, T. Röhling, F. Hoeller, D. Schulz, and F. Schneider. Benchmark of 6d slam (6d simultaneous localisation and mapping) algorithms with robotic mobile mapping systems. *Foundations of Computing and Decision Sciences*, 42, 09 2017.

[3] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729 vol. 3, April 1991.

[4] I. Colomina and P. Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 92:79–97, 2014. ISSN 0924-2716.

[5] T. D. Larsen, K. Lentfer, N. A. Andersen, and O. Ravn. Design of kalman filters for mobile robots; evaluation of the kinematic and odometric approach. volume 2, pages 1021–1026 vol. 2, 02 1999. ISBN 0-7803-5446-X.

[6] F. Duchoň, P. Hubinský, J. Hanzel, A. Babinec, and M. Tölgyessy. Intelligent vehicles as the robotic applications. *Procedia Engineering*, 48:105–114, 2012. ISSN 1877-7058. Modelling of Mechanical and Mechatronics Systems.

[7] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *Photogrammetric Computer Vision*, 2006.

[8] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, Jun. 2012.

[9] M. Greenspan and M. Yurick. Approximate k-d tree search for efficient icp. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings*, pages 442–448, Oct 2003.

[10] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA)*, 2005.

[11] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb.2007.

[12] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.
[13] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. volume 2, pages 1403–1410, 01 2003.
[14] F. Kallasi, D. L. Rizzini, and S. Caselli. Fast keypoint features from laser scanner for robot localization and mapping. *IEEE Robotics and Automation Letters*, 1:1–1, 01 2016.
[15] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.
[16] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, Nov. 2011.
[17] M. Miknis, R. Davies, P. Plassmann, and A. Ware. Near real-time point cloud processing using the pcl. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 153–156, Sep. 2015.
[18] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.
[19] A. Nuchter, K. Lingemann, and J. Hertzberg. Cached k-d tree search for icp algorithms. pages 419–426, 08 2007.
[20] J. Procházková and D. Martišek. Notes on iterative closest point algorithm. 04 2018.
[21] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. On Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.
[22] A. Restas. drone applications for supporting disaster management. pages 316–321, 01 2015.
[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, 2011.
[24] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison. Dense planar slam. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 157–164, Sep. 2014.
[25] M. Sathiyanarayanan, S. Azharuddin, S. Kumar, and G. Khan. Self controlled robot for military purpose. *International Journal for Technological Research in Engineering*, 1:2347–4718, 06 2014.
[26] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Robotics: Science and Systems*, Seattle, USA, Jun. 2009.
[27] *Sick MRS6000 Operating Instructions*. SICK AG, 11 2017.
[28] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, Oct. 2012.

[29] M. Takahashi, T. Suzuki, H. Shitamoto, T. Moriguchi, and K. Yoshida. Developing a mobile robot for transport applications in the hospital domain. *Robotics and Autonomous Systems*, 58(7):889–899, 2010. ISSN 0921-8890. Advances in Autonomous Robots for Service and Entertainment.

[30] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, April 1991. ISSN 0162-8828.

[31] *Velodyne LiDAR PUCK Datasheet*. Velodyne LiDAR, 2015.

[32] J. Zhang and S. Singh. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference*, Pittsburgh, PA, Jul. 2014.

[33] Ji Zhang and Sanjiv Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41:401–416, 02 2017. doi: 10.1007/s10514-016-9548-2.

Abbreviations

ADAS **A**dvanced **D**river **A**ssistance **S**ystem
AHRS **A**ttitude and **H**eading **R**eference **S**ystem
CPU **C**central **P**rocessing **U**nit
DGPS **D**ifferential **G**lobal **P**ositioning **S**ystem
DOF **D**egrees of **F**reedom
GPS **G**lobal **P**ositioning **S**ystem
ICP **I**terative **C**losest **P**oint
IMU **I**nertial **M**easurement **U**nit
LiDAR **L**ight **D**etection and **R**anging
PCA **P**rincipal **C**omponent **A**nalysis
PUT **P**oznan **U**niversity of **T**echnology
RMSE **R**oot **M**ean **S**quared **E**rror
ROS **R**obot **O**perating **S**ystem
SBC **S**olaris **B**us & **C**oach
SLAM **S**imultaneous **L**ocalization **A**nd **M**apping
SVD **S**ingular **V**alue **D**ecomposition

List of Figures

List of Tables

Reviewer:
dr hab. inż. Janusz Będkowski

Desktop publisher:
Andrzej Augustyński

Cover design:
Anna M. Damasiewicz

Cover graphic:
Krzysztof Ćwian

ISBN e-book 978-83-8095-730-5

impuls

Oficyna Wydawnicza „Impuls”
30-619 Kraków, ul. Turniejowa 59/5
phone/fax: (12) 422 41 80, 422 59 47, 506 624 220
www.impulsoficyna.com.pl, e-mail: impuls@impulsoficyna.com.pl
Edition I, Kraków 2019