MTA SZTAKI
COMPUTER AND AUTOMATION RESEARCH INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMÍTÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓINTÉZET

Reconstruction of 3D Urban Scenes Using a Moving Lidar Sensor

Oszkár Józsa — Csaba Benedek

Technical Report

N° i4D-1

January 2013

Theme VISION

MTA SZTAKI
H-1111 Budapest, Kende u. 13-17, Hungary
H-1518 Budapest, P.O.B. 63, Hungary
Phone: (+36 1) 279 6000
Fax: (+36 1) 466 7503

MTA SZTAKI

Reconstruction of 3D Urban Scenes Using a Moving Lidar Sensor

Oszkár Józsa*, Csaba Benedek†

Theme VISION — Computer Vision
Divison: Distributed Events Analysis Research Laboratory

Research report — January 2013 — 41 pages

Abstract: In this report, we propose algorithms which interpret and display 3D environments. The input of this procedure is a LiDAR sensor mounted atop of a car. The sensor outputs a data stream covering more than 100 meters radius of space, collecting data at 15 Hz. The recording is done in real environment on the streets of Budapest in real time, while the processing is offline, implemented on CPU keeping in mind the future implementation on GPUs to reach real time data processing. The aim is to segment several region classes (such as roads, building walls, vegetation) and to identify specified objects (such as people, vehicles, traffic signs) in the point clouds through a presegmentation step. To achieve this classification, we need several features such as the color and geometrical properties of the specified objects and their possible geometrical and physical interactions. Also, we need to take into account the time domain features calculated based on the LiDAR data stream. After this presegmentation step we are able to reconstruct building facades in 3D and to track the detected objects in the 3D space. Also, in the future, this processed data set can be registered against 2D images provided by conventional cameras to reproduce realistic, colored 3D virtual spaces. The data is provided by a Velodyne HDL-64E high performance LiDAR device

Key-words: rotating multi-beam Lidar, pointcloud analysis, city reconstruction

This work is connected to the i4D project funded by the internal R&D grant of MTA SZTAKI.

* The author was supported by the Grant #83438 of the Hungarian Research Fund (OTKA)
† The author was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the Grant #101598 of the Hungarian Research Fund (OTKA)

Computer and Automation Research Institute
Hungarian Academy of Sciences
H-1111, Budapest Kende utca 13-17, Hungary
Telephone: +36 1 279 6000, Fax: +36 1 466 7503
http://www.sztaki.hu

3D városi környezet rekonstrukciója mozgó Lidar platform pontfelhő szekvenciáiból

Kivonat : Riportunkban bemutatunk általunk fejlesztett algoritmusokat dinamikus utcai környezetek 3D rekonstrukciójára és virtuális megjelenítésére. Az eljárás bemenetét egy autótetőre szerelhető LiDAR készülék szolgáltatja, aminek a segítségével nagy pontosságú 3D pontfelhő szekvencia nyerhető egy akár 100 méter feletti sugarú területről 15Hz rögzítési sebességgel. Az adatok rögzítése valós, Budapesti utcai környezetben történik, míg a feldolgozás offline, CPU-n megvalósított algoritmusokkal, szem előtt tartva a későbbi GPU-ra való optimalizálást a valós idejű futás elérésének érdekében. A cél az egyes területosztályok elkülönítése (pl.: úttest, házfal, növényzet), és meghatározott objektumok észlelése (emberek, járműveк, közlekedési táblák) a pontfelhőkben egy előszegmentációs lépés során. Az osztályozáshoz különböző jellemzők együttes felhasználása szükséges, magában foglalva az objektumok alakját és színét, lehetséges kölcsönhatásaik modelljeit valamint a LiDAR adatfolyam alapján számolt különféle időbeli jellemzőket. Ezt követően az előszegmentált pontfelhők alapján lehetőség nyílik az épülethomlokzatok 3D rekonstrukciójára és az észlelt objektumok követésére, valamint az így kapott feldolgozott, értelmezett ponthalmazt későbbiekben illeszteni lehet az optikai kamerák 2D képeihez is, és ezek alapján színezett, élethű háromdimenziós helyszínmodellek származtathatók. Az adatok rögzítése egy Velodyne HDL-64E nagy teljesítményu LiDAR eszközzel történik.

Kulcsszavak : Lidar, pontfelhő analízis, város rekonstrukció

Contents

1 Introduction

Analysis of 3D spaces comes from the need to understand the environment surrounding us and to being able to build more and more precise virtual representations of that space. Remote sensing is a widely researched field for many decades and so is scene interpretation. But in the last decade, as the three dimensional (3D) sensors begun to spread and the commercially available computing capacity has grown big enough to be sufficient for large scale 3D data processing, new methods and applications were born. In many real world problems, 3D sensing and scene interpretation started to take the place of the conventional 2D image processing based on two dimensional imagery data.

In recent years, more and more technologies started to appear that heavily rely on these new 3D methods. Robotic cars, ships and airplanes are successfully tested and in some cases are already used on everyday bases. As Luettel et. al.[1] state in their summatory article, automation of our vehicles are the "path to the future". A key element for these instruments is 3D interpretation, detection and recognition.

But not only the automation industry depends heavily on 3D mapping and scene interpretation. There are applications of this technology in industrial manufacturing, geology and mapping, architecture, disaster and crime prevention and control, space exploration, military intelligence and virtual gaming. All these fields have their own special tasks, where 3D data can help both the professionals and the users.

Three dimensional data can be produced in several ways. It can be generated via sound propagation (sonars), radio wave propagation (radars) and light propagation (CCD devices, LiDAR devices). Light Detection and Ranging (LiDAR) devices are common tools for three dimensional mapping as they provide accurate 3D data directly from the device (so there is no need of complex computation as via multiview cameras). These devices provide us the so called *point clouds*. A point cloud is a data set with small units of data, each representing a 3D point in the space. A point essentially has at least three information: its 3 coordinates, x, y and z (which of course can be represented in polar coordinate system or Euler angles or *lattitude*, *longitude*, *altitude* in geographic data sets, etc). Additionally color, intensity information could also be provided by some devices.

These instruments can quickly produce huge amount of data which makes data processing a hard task. Some of these devices output several million data points per second so our task is not only to interpret the data but to select the subset of points worth interpreting also. Efficient, fast methods are needed to filter the significant data out of these streams or high computing power is needed to post-process all this large amount of data. Once we are able to select the useful information, another interesting task is to merge all the data from the whole scan sequence to produce even larger and more useful 3D data sets. Matching subsequent scans to each other is called *point cloud registration* which can provide us even more useful data sets; for example registered urban point clouds can be processed and analyzed similarly as aerial LiDAR scans but at much higher resolution since the sensor is able to scan the scene from a much smaller distance.

The analysis of 3D dynamic urban scenes can be the first step for either real time scene interpretation tasks (essential for traffic monitoring and robotic cars), or for post processing three dimensional data (building realistic virtual models of real cities).

2 Light Detection and Ranging

In this section, we introduce the working principles of the Light Detection and Ranging (LiDAR) technology and show examples of devices and their usage. We also describe the data they provide us.

2.1 LiDAR technology

Light Detection and Ranging (LiDAR) is an optical remote sensing technology used for distance measurements. It is available in terrestrial, airborne and spaceborne devices. Wide range of instruments are available for variety of tasks - ranging from hobby devices about 100 US dollars up to military-grade high powered multi-beam ones for hundreds of thousands of US dollars.

The principals of LiDAR distance measurement is essentially the same as of a radar device but instead of radio waves, a LiDAR uses light to measure distances, thus the name *Light Detection and Ranging*. The distance of an object is calculated from the time it took the light beam to bounce and arrive back from and object. Since we know the speed of light, the total distance traveled is simply given by the time multiplied by the speed of light as shown in Equation 1.

$$object\ distance = \frac{\text{time of flight } * \text{speed of light}}{2} \tag{1}$$

Also there is a special type of LiDAR devices that is used for particle detection is gases, typically in the air. With a very high sensitivity sensor and/or using multiple laser frequencies several types of particles can be detected in gases. Such a device was in operation on the Phoenix spacecraft (Martian rover).[1]

A big advantage of this technology over conventional optical (CCD) imagery that it is not affected by lighting conditions. LiDAR is an active sensing technology, meaning that it illuminates its target so lack of external lightning (e.g. at night) does not corrupt the measurement. Also, special types of LiDARs are able to "see" through water, thus being able to scan underwater surfaces.[2]

In most cases, terrestrial LiDAR devices operate while surrounded by people. For safety reasons, these devices use eye safe (Class 1 laser) laser emitters, usually between 600 and

[1] For detailed description of this device, please refer to the Phoenix mission webpage: http://phoenix.lpl.arizona.edu/science_met.php

[2] So called hydrographic mapping are provided by companies such as Optech (http://optech.ca/) and Fugro (http://www.fugro-uae.com/)

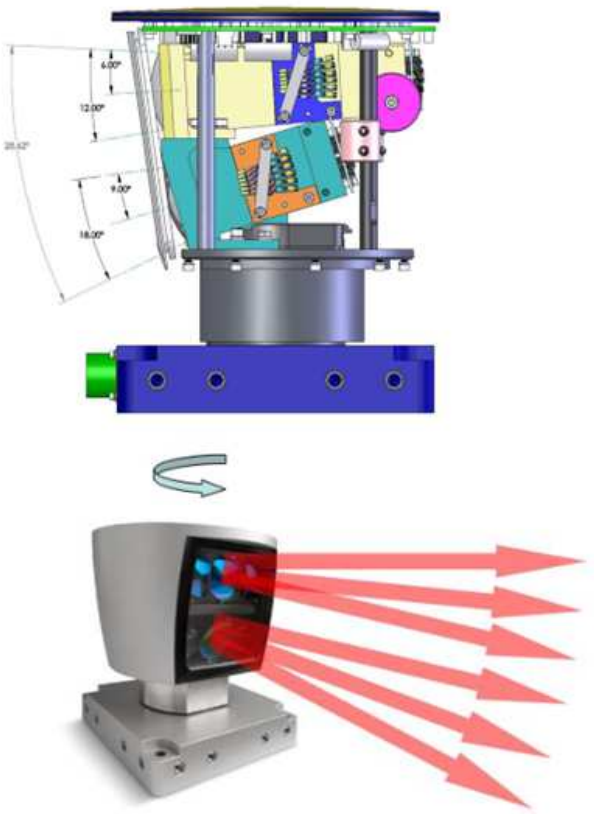


Figure 1: The internal structure and the working principle of the Velodyne HDL-64E sensor. It has a rotating head and 64 laser beams aligned along a single line. With this line, it scans the surrounding 3D space as the head rotates around in 360 °. The four shining plates on the face of the device are the laser emitters and sensors (two of each) (image source: Argocorp product description)

1000 nanometers of wavelength.[3] This light is in infrared range which makes the light beams invisible to human eye.

Different LiDARs have different operation principles. Some devices (such as the SICK LiDARs, or the industrial stationary laser scanners) have one laser beam that sweeps back and forth in one dimension, along a single line. Either the scanned object or the scanning platform has to move to for a complete 3D scan to be performed. Other types of LiDAR scanners, such as the Velodyne devices, have a revolving head which sweeps in a circle around the sensor, resulting in a full 360 ° scan. These sensors provide 3D images even at a stationary position, though in most cases these are used on a moving platform, also.

[3]Source: www.LiDAR-uk.com/how-LiDAR-works/

2.2 The Velodyne Sensor

The data processed in this report is provided by the *Velodyne HDL-64E* high performance LiDAR sensor. According to it's website, the HDL-64E LiDAR sensor is designed for obstacle detection and navigation of autonomous ground vehicles and marine vessels. Its durability, 360 ° field of view and very high data rate makes this sensor ideal for the most demanding perception applications as well as 3D mobile data collection and mapping applications."[4]

The typical usage of this sensor is to mount it on a moving platform, practically onto a car (Figure 2 shows the actual device while recording on the streets of Budapest). The LiDAR has a fast enough scanning speed to scan the whole 3D space at about every 30-60 centimeters of distance as the car travels with typical urban traffic speed. With each rotation, a full 3D scan is made, and overall the device outputs more than 1.3 million 3D points per second via an unshielded twisted pair (UTP) Ethernet cable.

Alternatively, it can be used as a stationary sensor at outdoor or relatively large indoor scenes (e.g.: warehouses, manufacturing halls). In this configuration, the sensor scans the same space with every rotation. Applications of this static configuration will not be discussed in this this report, instead, we will be focusing on the more challenging moving platform.

Along the 3D (x-y-z) data, the sensor also gives a fourth value for each point, called *intensity*. This value shows how strong the reflection from that particular point was. In other words, how big fraction of the emitted light returned to the sensor. This intensity value mostly depends on the objects' surface quality: shiny, flat surfaces reflect light much better than matte, scattered surfaces (such as roads or vegetation). The angle of attack can also affect this intensity value i.e. if the beam hits the surface in a high angle, the light can scatter and thus the returning intensity will be lower. Although the algorithms implemented so far do not exploit the intensity information, in Section 7 we will present our research on where it might highly support some of the proposed detection tasks.

[4] For full product description, visit the official product website http://www.velodyneLiDAR.com/LiDAR/hdlproducts/hdl64e.aspx

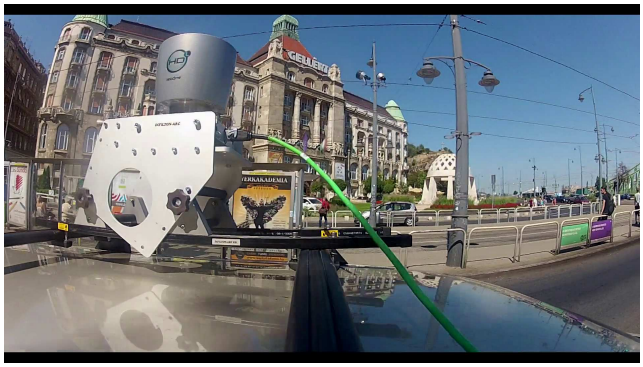


Figure 2: Velodyne HDL-64E sensor on the streets of Budapest (pictured in front of Gellért Bath)

Specifications	
Sensor:	• 64 lasers/detectors • 360 degree field of view (azimuth) • 0.09 degree angular resolution (azimuth) • 26.8 degree vertical field of view (elevation) - +2° up to -24.8° down with 64 equally spaced angular subdivisions (approximately 0.4°) • <2 cm distance accuracy (one sigma) • 5-15 Hz field of view update (user selectable) • 50 meter range for pavement (~0.10 reflectivity) • 120 meter range for cars and foliage (~0.80 reflectivity) • >1.333 M points per second • Operating temperature - 10° to 50° C • Storage temperature - 10° to 80° C
Laser:	• Class 1 - eye safe • 4 x 16 laser block assemblies • 905 nm wavelength • 5 nanosecond pulse • Adaptive power system for minimizing saturations and blinding
Mechanical:	• 15V ± 1.5V @ 4 amps • <29 lbs. • 10" tall cylinder of 8" OD diameter • 300 RPM - 900 RPM spin rate (user selectable) • Environmental Protection IP67
Output:	• 100 MBPS UDP Ethernet packets

Figure 3: Data sheet of the Velodyne HDL-64E LiDAR (image source: data sheet on the product website)

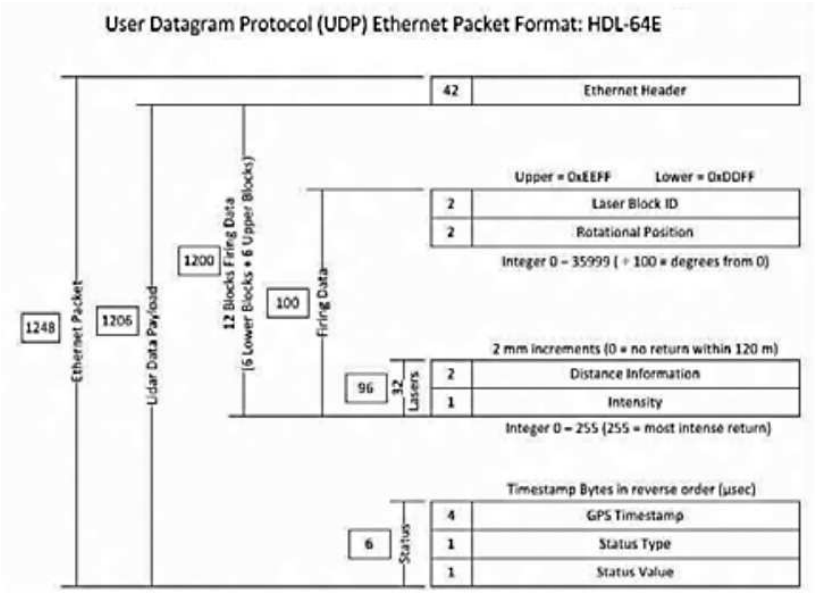


Figure 4: Data sheet of the Velodyne HDL-64E LiDAR (image source: product manual sheet)

2.3 Technical parameters

The sensor provides the data in an UDP stream through a standard Ethernet cable. The packet structure is detailed by the manual as follows:

2.3.1 Scanning configurations

The sensor has two main configurations when mounted atop of a vehicle. It can be either facing forward and thus, looking down in an angle of about 25 degrees or it can be tilted to face upwards. The former, forward facing configuration is more suitable for road mapping, traffic monitoring, object detection and tracking, etc. The latter configuration, because of its upward looking nature, is more suitable for scanning building facades. A typical angle in the tilted configuration is 45 ° which allows the sensor to see up several stories high from street level.

Though this report focuses on the forward facing configuration, the Velodyne HDL-64E LiDAR can be used in both configurations and some results of the different measurements and the possible uses of them will be briefly discussed later.

(a) Forward facing LiDAR (b) Tilted LiDAR

Figure 5: The two scanning configurations

2.4 Data

All the data had been recently recorded in urban areas, on the streets of Budapest. Several types of streets have been scanned to test the algorithms as well as bridges and large squares. The types of streets range from avenues to small hillside roads.

The data recording has been done with a software tool developed earlier at the Distributed Events Analysis Laboratory (DEVA). This tool saves the data into the standard *pcap* (packet capture) format which contains the raw UDP stream with timestamp and header information along with the 3D data (see Figure 4 for the packet structure). Also, this program can export the data into the simple *.pcd* file format provided by the Point Cloud Library (for more detail, see Section 8). In this format, the recording is a file sequence with every file containing one 360 ° scan.

The processing framework uses the PCD sequence format. A typical sequence contains a few hundred scans and covers a whole street or a segment from an avenue. A scan recorded with 10-15 Hz of rotation has a size about 2 to 6 megabytes so a sequence can reach multiple gigabytes in size. We call the acquired sequence 4D data since each frame is a 3D point cloud and the time domain represents the fourth dimension.

Different scans have been recorded with different rotational speeds, ranging from 5 to 15 Hz. The test showed that there are no differences in processing these data, though in case of a car moving around 50 kph, a faster recording speed is more suitable as there will be less movement during a single scan.

3 Problem formation

Though the LiDAR sensor provides high amount of data with each revolution, often this 3D information is not sufficient to understand a whole scene which sometimes spans over 200 meters. Typical challenging problems are occlusion (and thus, large “shadows” on the occluded background surface), sparse data and noise. Though the sensor has one sigma

accuracy, even a few centimeters of fluctuating noise can damage the surface reconstruction and make identification tasks challenging.

Also, the density of the provided point cloud is highly inhomogeneous: the further an object from the sensor is, the sparser its 3D scan will be. This fact hurts lots of existing processing methods developed for the analysis of homogeneous aerial LiDAR scans will not give the same results for the same object when it is close to the sensor and when it is far away.

As the sensor does not see directly down to the ground, each scan has a "hole" of about 3 meters radius in the center. Some registration algorithms consider this "edge" in the data as a significant feature. Once identifying these points as keypoints, they register two point clouds along these points which obviously results in a false registration since the "hole" in the data moves along with the car and should not be aligned in consecutive point clouds.

In addition to noise, moving points can cause hard challenges for the processing algorithms even more. In a realistic street scene, there are number of moving objects: people, vehicles, vegetation. All the points belonging to these objects change from frame to frame. A false registration can easily occur when the algorithm falsely detects moving points as significant points and tries to align consecutive point clouds along these moving points.

If we were able to register all the clouds in a sequence against each other and concatenate them into a large point cloud (within the same reference coordinate system), huge resolution could be achieved and most of the problems mentioned in this section could be overcome. Once we have the registered cloud, the resolution will be uniform and most of the occlusion will be gone due to the moving platform which helps the sensor to measure objects from different angles. Also, in this significantly larger resolution, effective noise reduction can be applied and detection algorithms can perform much better.

In summary: a data preprocessing method is needed that helps the point cloud registration process. In particular, a segmentation method, which removes (and optimally classifies) all the points that harden the registration task such as the road surface, moving objects and noise. Our aim is to select a set of points that are stable, static in each frame and can be confidently used as keypoints for point cloud registration.

The key contributions of this report are as follows. In Section 5, we present our presegmentation method which aims to help resolve the basic point cloud classification problem, i.e. separating ground, wall, object etc. regions in the point cloud. In Section 6, we introduce a new algorithm to transform the consecutive Velodyne frames into a joint coordinate system, in such difficult scenarios, when both the LiDAR sensor is moving and in and several moving objects are also present on the street. In addition, it should be highlighted that we do not use any special sensor information (IMU) about vehicle speed and acceleration for matching the frames.

In Section 7, we present our results on these registered data and some future plans for improving the current results. In Section 8, we present the framework we developed for the workflow (preprocessing, registration and visualization of the data). Finally, in Section 9, we will draw the conclusion and analyze the results.

4 Related work

Hereby we present a short overview of the state of the art results on this topic as the result of our literature research. Most researches are focusing on automation, robotic vehicles only fewer researches are made on the topic of virtual city reconstruction and building facade reconstruction.

In most cases, for point cloud registration in robotic environments the ICP [2] algorithm is used for matching which performs well in homogeneous 3D scans but often fails in noisy, fast-changing real world environments or has to be modified to be scene specific. In robotic applications, the main task is to detect and track moving objects[3, 4], while our current task is to understand a scene with both moving and static objects and build realistic virtual representation of that scene.

Douillard et. al. have developed a method to extract "segments" (interest regions) from the point cloud [5] [6]. This approach is similar to mine: with the preprocessing step, we also try to find sets of points that will later help the registration step. Makadia et. al. constructed an automatic registration method using large histograms calculated from the surface normal fields[7] but is computationally expensive and works well on homogeneous scans, not on LiDAR data.

Others have taken a different approach Behley et. al. [8] constructed a multiscale 3D histogram descriptor that can be efficiently used for point matching. Quadros et. al. presented a feature called *the line image* to support point matching that outperforms the widely used NARF descriptor but requires a computationally expensive principal component analysis (PCA) calculation[9].

After the registration is done, next step is virtual reconstruction. Since the scans are noisy by nature and possibly have several holes in them (e.g.: occlusion caused by trees and parking cars standing in front of buildings, etc.), reconstruction is not trivial. Recently, Shen et. al. showed an efficient method for virtual facade reconstruction which uses an iterative adaptive partitioning and is robust against the aforementioned issues [10]. Also, Wang et. al. of the NAVTEQ Corporation showed a method to upsample sparse LiDAR data and to clean the points corresponding to building interiors (since the laser beam travels through windows also, the LiDAR also scans the interiors of buildings)[11].

Also, for already registered, large data sets, there are effective detection, segmentation and recognition algorithms available. Velizhev et. al. developed a highly effective algorithm to interpret large outdoor urban scenes[12]. They method is proposed for mobile robot mapping; though, once we register LiDAR data with the method presented in this report, our data sets will be similar to theirs.

Though in our method, running time is not a bottleneck, in future development parallelisation might fasten up the algorithms greatly. Hu et. al. numerically proved that point cloud filtering and segmentation can be greatly faster with a graphics processing unit (GPU) implementation[13].

5 Class segmentation

To achieve the aforementioned goals, we developed a novel method to preprocess and process 3D LiDAR data. The key idea is to insert a simple and fast preprocessing step that will help the registration step, hence scene reconstruction and scene interpretation also.

We have defined four classes[5]

- Road surface (including sidewalks)
- Short street objects (which are moving objects in lots of cases, such as cars, people)
- Walls and tall static objects (lamps posts, traffic lights, etc.)
- Sparse data / noise

This classification is based on local point properties. Using some statistical descriptors, we segment the data into one of these semantic classes which later can be used together or separately for various tasks (eg. using only the object class for object detection).

5.1 Method

As mentioned before, point cloud segmentation is done by a grid based approach. First, we fit a regular 2D grid S with W_S rectangle width onto the $P_{z=0}$ plane, where s denotes a single cell. We used a W_S value between 50cm and 80cm. Smaller grid size is not viable due to the resolution; smaller cells would not have enough points in them to calculate good enough statistical information. On the other hand, larger cell size can result in larger number of falsely classified points, since within a large cell, multiple objects can occur. Near-the-center grid cells within the above specified range have hundreds of points in them (grid cells further away obviously have less and less points).

Then, we assign to each $p \in \mathcal{P}$ point of the point cloud to the corresponding cell s_p, which contains the projection of p to $P_{z=0}$. Let us denote by $\mathcal{P}_s = \{p \in \mathcal{P} : s = s_p\}$ the point set projected to cell s. $y_{\max}(s)$, $y_{\min}(s)$ and $\hat{y}(s)$ are the maximum, minimum and average of the elevation values within $\mathcal{P}_s$.

The first step of the process is the segmentation of the cell-map, i.e. we assign to each cell (and to each point in that cell) $s \in S$ an ω_s class label from the finite label set: $\mathcal{L} = \{l_{\text{to}}, l_{\text{so}}, l_{\text{gr}}, l_{\text{sp}}\}$, corresponding to the classes (i) *tall structure object*, (ii) *short street object*, (iii) *ground* and (iv) *sparse region*.

5.1.1 Classes

At first, sparse regions (l_{sp}) are detected. These encapsulate only a few or not any points: we can obtain only very limited, and probably misleading information from these cells regarding the scene structure and objects, therefore we will neglect these regions in the later model

[5] in the source code, there is a fifth class called *NOCLASS* for initialization purposes

phases. The threshold of a cell s being considered as a sparse cell is typically 4-8 points - any cell containing less points than this threshold is considered sparse (the exact value of this threshold also depends on the sensors revolving speed, slower speeds allow to have larger threshold). Sparse data is classified first, so in later processing we will know not to check these grid cells, thus save some processing time.

A cell should belong to the l_{to} class if it contains a tall object of the street structure, such us building walls, lamp post or tree trunk. These object can be considered static in the scene, since these 3D points obviously belong to objects that are not moving and are large enough to be present in several consecutive scans. These points will be used later for point cloud registration of the consecutive time frames. The two criteria for a grid cell to belong in this class are:

$$y_{\max}(s) < 140 \;||\; y_{\max}(s) - y_{\min}(s) > 310 \tag{2}$$

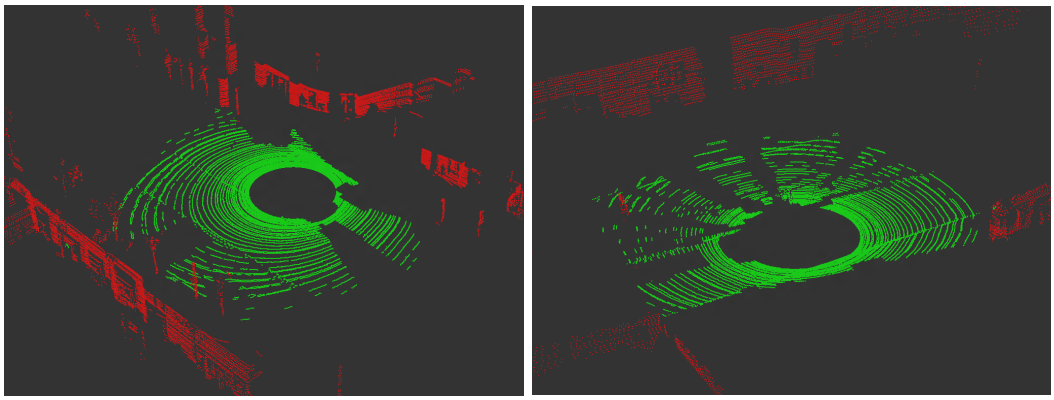
So either the maximal height is larger than 140 centimeters, either the height difference is larger than 310 centimeters. The two criteria are similar. The first one says that if there are points in the grid cell which are higher than the sensor height plus 120 centimeters, than the cell should be considered as a tall object (the sensor has a height about 2 meters). The second criteria says that if there is a height difference of more than 310 centimeters within the cell, it should be considered as a wall. The second criteria is needed for dealing objects standing on a lower point of the ground compared where the car travels (e.g.: elevation differences, walls seen from an overpass road, etc).

Next, the ground cells (l_{gr}) are identified. These cells contain only a surface which is not covered by any objects, but contains a considerable number of ground points. The criteria for the grid cell to belong into this class is:

$$y_{\max}(s) - y_{\min}(s) < 25 \;\&\&\; y_{\max}(s) < -50 \tag{3}$$

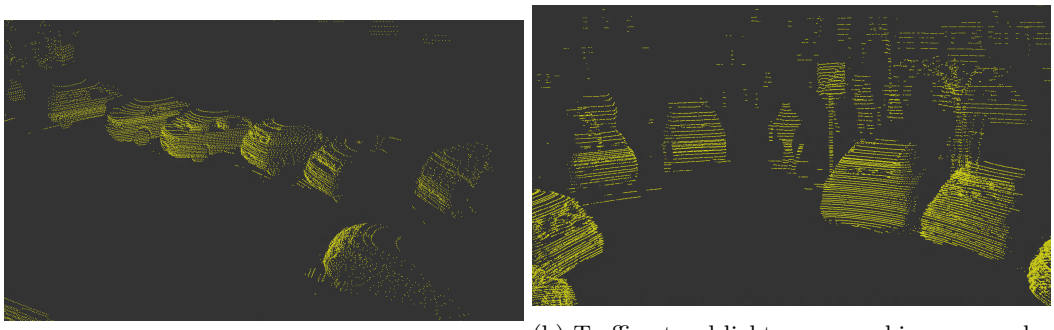
In words: if the points within a cell are "flat" enough, and the maximal height is below -100 centimeters, the grid cell is considered as road surface. The first criteria ensures the flatness or homogeneity of the points. Given a cell with 60 centimeters if width, this allows 22.6° of elevation within a cell; higher elevations are highly unrealistic in an urban scene. The second criteria ensures that this patch of flat surface is under the car; again, the sensor is at about 200 centimeters of height so the road surface directly beneath the car has a height of about -200 centimeters. Less than -50 criteria is used to deal with elevation differences that can occur within the field of view of the sensor.

The rest of the point cloud is assigned to class l_{so}. These points belong to short objects like vehicles, pedestrians, short road signs, line posts etc. These entities can be either dynamic or static, which attribute can only be determined later, after further, more complex investigation of the point cloud sequence.



(a) Bartók Béla street (b) Villányi street

Figure 6: Streets “cleaned” from noise and objects



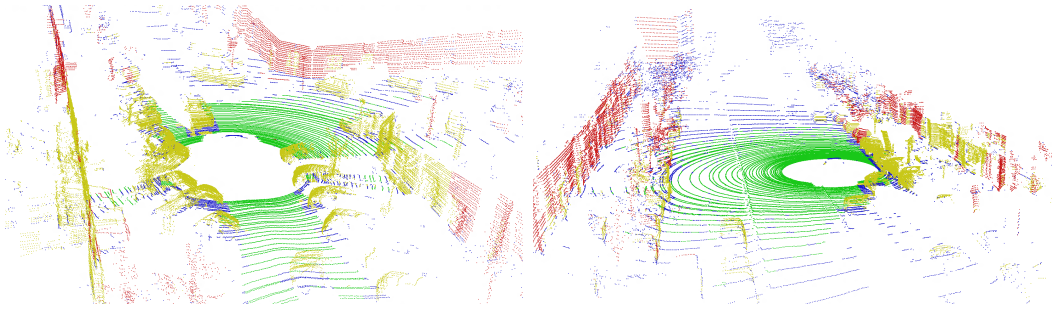
(a) Parking cars on the side of Villányi street (b) Traffic at red light, some parking cars and a person passing by on a small street

Figure 7: Streets with only the object clouds displayed

5.2 Results

Here we present some images that show the results of the segmentation. For detailed description about the processing framework, see Section 8.

Notice, that on Fig. 8 there are seemingly large amount of noise. Actually, since the majority of those points are in the sparse region, in most of the cases the amount of the noise/sparse data is below 10%.



(a) Intersecting one-way streets (b) A multi-lane street

Figure 8: Street segments with sparse and noise data included (blue points)

6 Point cloud registration

A single scan has many points but but since it covers a large area, the resolution is sufficiently good only within few meters of distance. Though the device has a sensing distance of more than 100 meters, the measurements at more than 15 meters of distance are too sparse for many detection or reconstruction algorithms. But since the sensor is on a moving platform, it is a good idea to register, or in other words, concatenate the consecutive scans in the sequence. Registration will help us to fill in the holes due to occlusion and we can achieve very high point density after registering several point clouds.

Although various established techniques do exist for point cloud registration, such as Iterative Closest Point (ICP) [2] and Normal Distribution Transform (NDT) [14], these methods fail, if we try to apply them for the raw Velodyne LIDAR point clouds for two reasons:

1. All moving points appear as outliers for the matching process, and since in a crowded street scene we expect several moving objects, many frames are erroneously aligned.

2. Due to the strongly inhomogeneous density of the LIDAR clouds, even the static ground points mislead the registration process. The above algorithms often match the concentric circles of the ground (See Fig. 6), which yields that the best match erroneously corresponds to a near zero displacement between two consecutive frames. However, we have also observed that the point density is quite uniform in local wall regions which are perpendicular to the ground.

Our key idea is to utilize the point classification result from the previous section to support the registration process. We only use as input of the registration algorithm the points segmented as high objects, since we expect that in majority, these points correspond to stationary objects (such as buildings), thus they provide stable features for registration. Details and test results are in the following sections.

6.1 Normal Distributions Transform

This method was introduced in Martin Magnusson's doctoral thesis, *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection* [14] which is a 3D implementation of the 2D Normal Distributions Transform developed by Biber and Straßer.[15]

The Normal Distributions Transform itself uses a grid based approach also. First, it divides the space into cubes. For each cube, calculates its so-called local probability density function (PDF) to describe that cube: "Each PDF can be seen as an approximation of the local surface, describing the position of the surface as well as its orientation and smoothness."[14]

For the registration step, it uses Newton's optimization method to find the rotation and translation between the two point clouds, searching for the best match between the PDFs of the two scans. This method is robust to outliers.

6.2 Related work

In many cases, this registration is done by the help of external sensors such as Inertial Measurement Units (IMU), Global Positioning System (GPS) or other sensors on the car (e.g. speedometers, wheel rotation sensors, radars).

Han et. al. presented a method to detect road boundaries and road segments based on LiDAR data and vehicle movement information [16]. Robotic cars such as Stanley and Junior built for the DARPA Grand Challenges[17] [18], the AnnieWAY self driving car [19] and Carnegie Mellon's Car, Boss[20] for example. use a huge amount of sensors to map their environment and interpret the scene surrounding them.

All of the results above incorporate additional sensors. The novelty of our method that it does not need any supplementary sensory information, also does not need any learning algorithms or huge sample database. Once the first correspondence is found, the calculated transformation matrix is used as a prediction for the next movement step, the NDT algorithm only has to "refine" this prediction according to changes in the movement of the car.

6.3 Registering raw data

Here we present the two typical types of false point cloud registration. The first one is shown in Fig. 9. In this case, the algorithm picked the hole in the middle of the scan (due to the sensor's field of view) as the key feature.

The latter one, shown in Fig. 10, displays a completely misaligned registration. Here, the algorithm aligned the flooring of the bridge on the last two scans to the stanchions of the bridge in the previous scan.

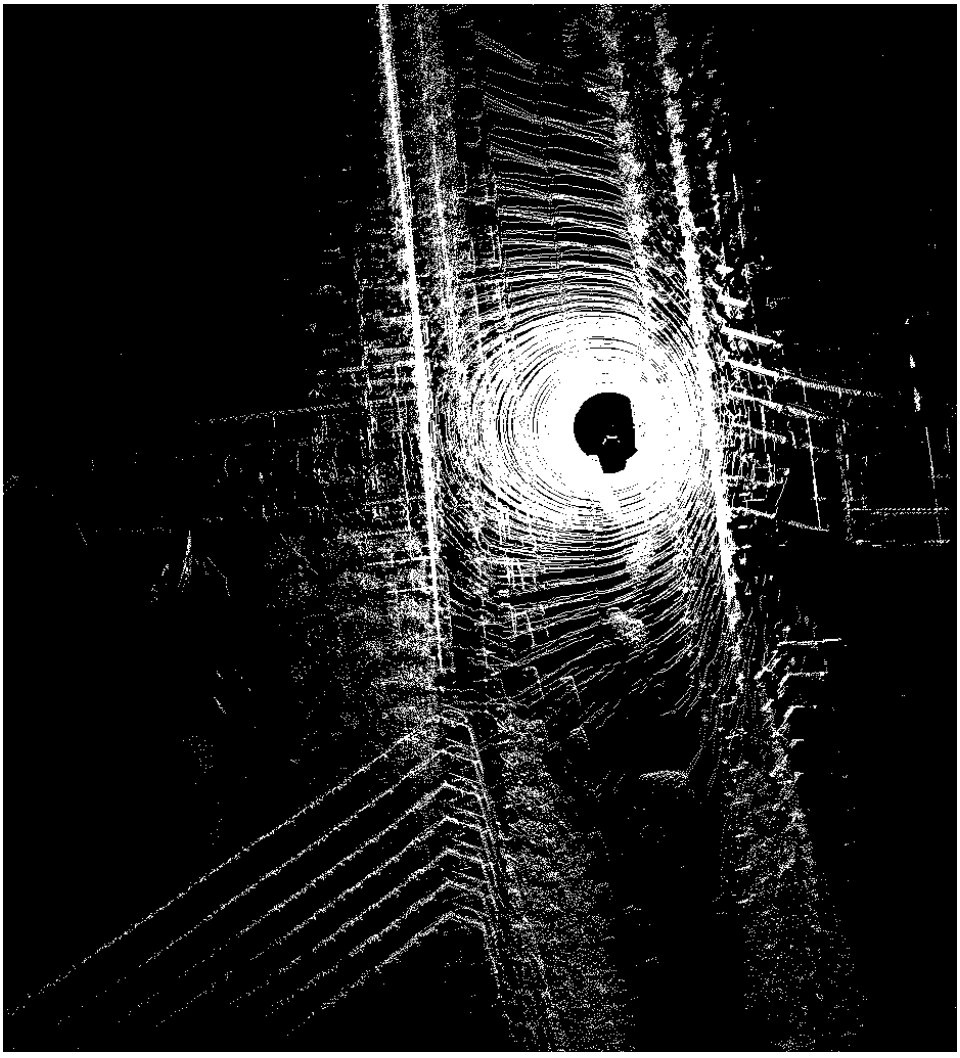


Figure 9: Misregistered street segment

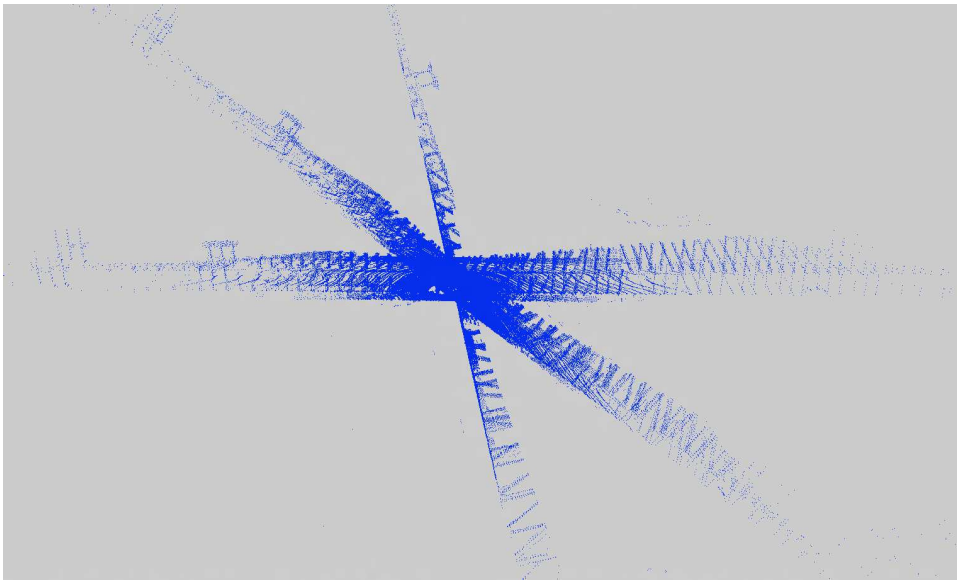


Figure 10: Misregistered Liberty Bridge (Szabadság híd). See Fig. 11 for proper registration

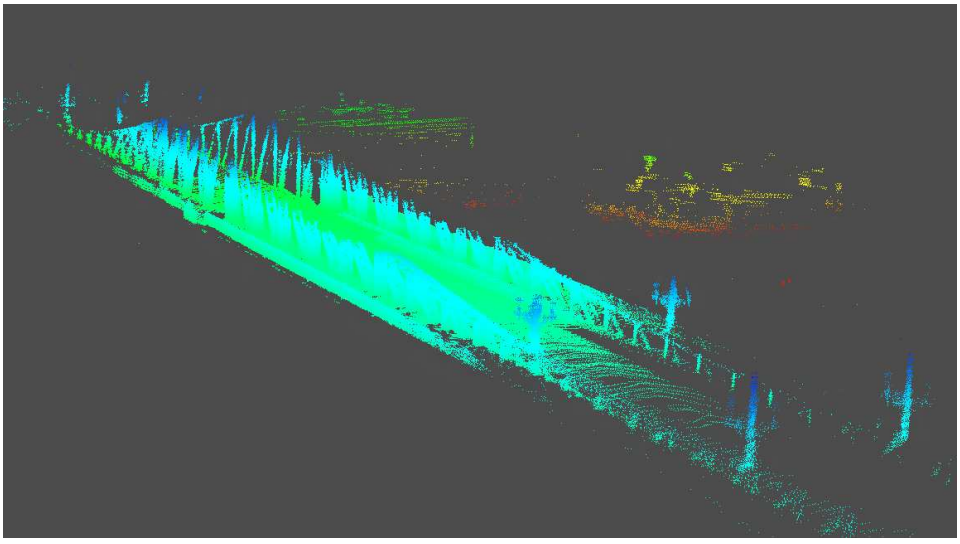
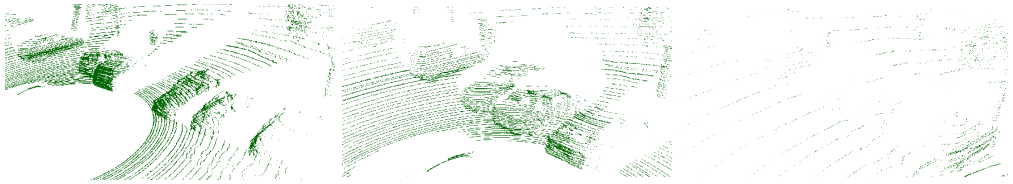
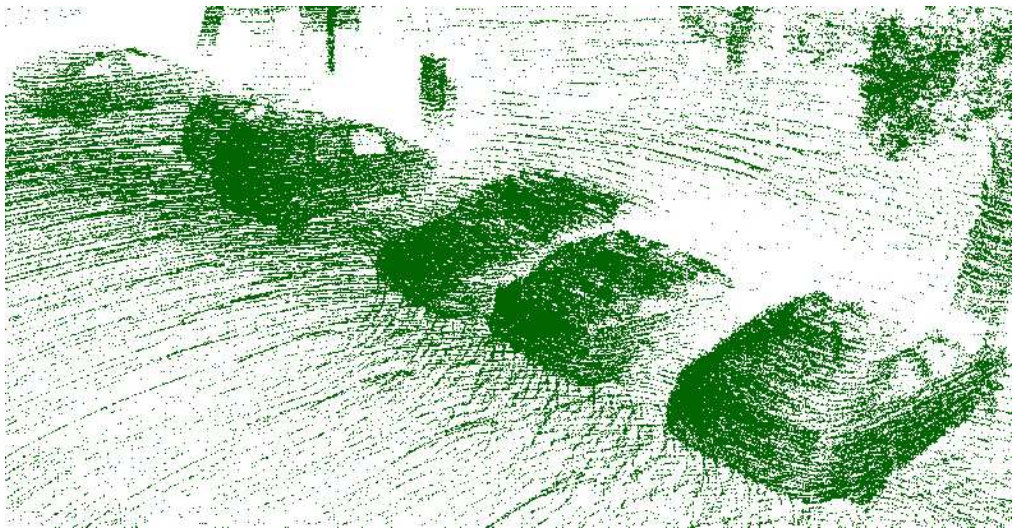


Figure 11: Liberty Bridge (Szabadság híd) with boats in the background

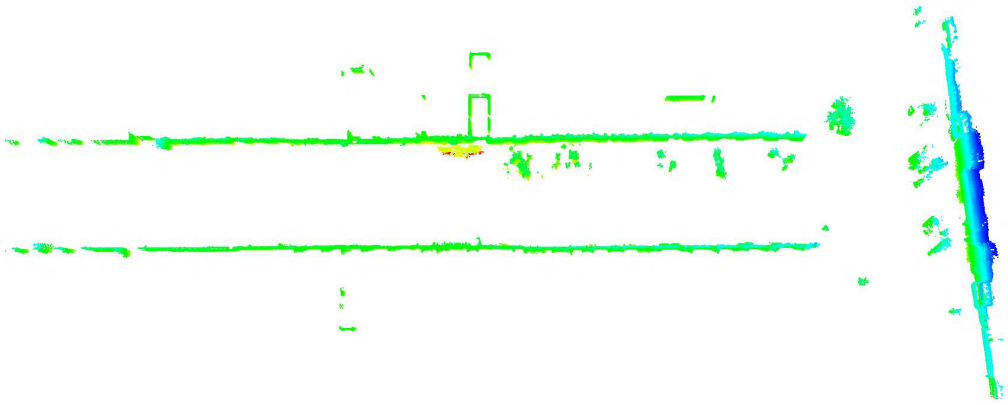


(a) First scan. (b) Second scan. (c) Third scan.

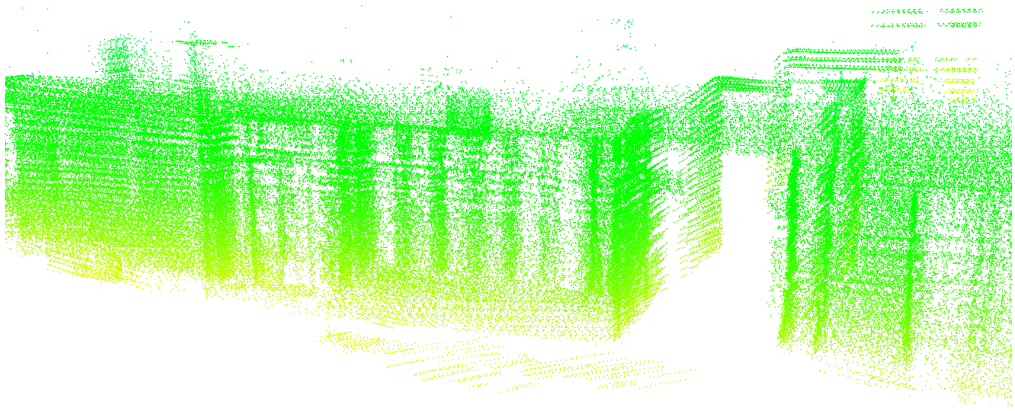


(d) The resulting registered scan (made out of 20 scans, not only the 3 above).

Figure 12: An image sequence showing the registration process.

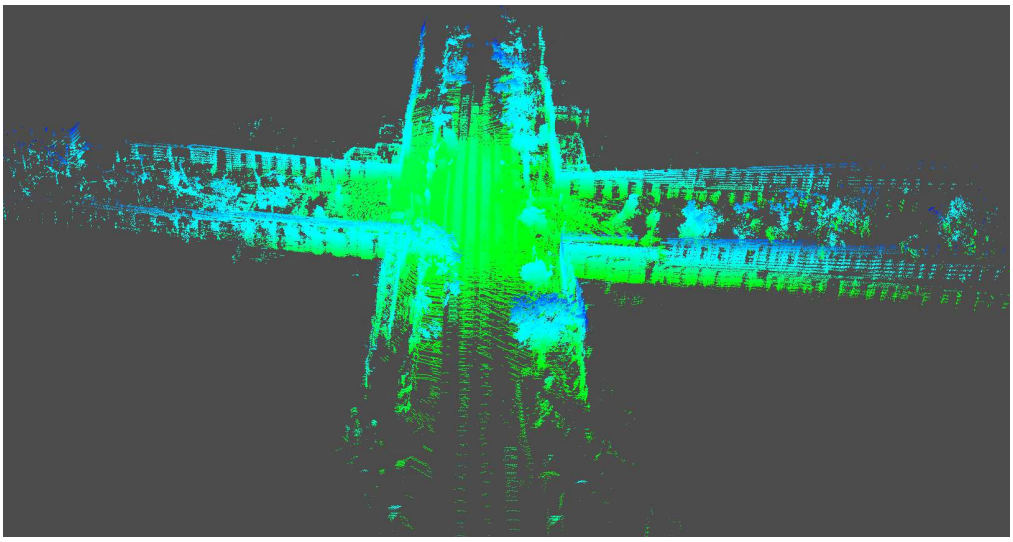


(a) Upper view of the street. Notice, how the interiors of the building gets visible as the scanner passes the door.

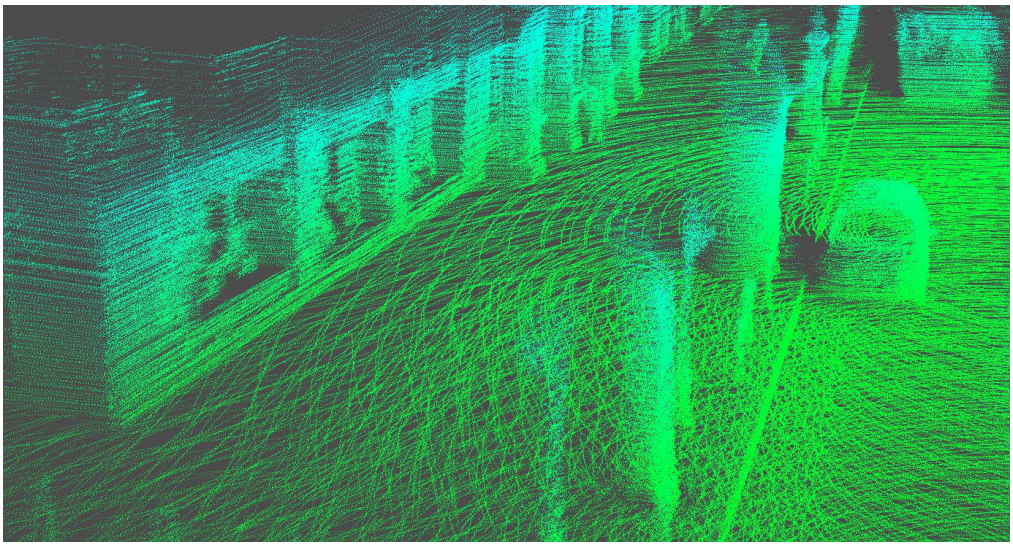


(b) Zoomed in picture.

Figure 13: A large registration of Kende street showing only the building class. Scan sequence starts at the middle of the street and runs to its end. The point cloud is made out of 230 point clouds, the full cloud contains more than 20 million points.



(a) The whole registered street segment



(b) Part of the street, zoomed in. Note the level of detail, even objects in the store window are visible.

Figure 14: Registered street segment. For registration, 30 point clouds were used, final cloud contains almost 10 million points.

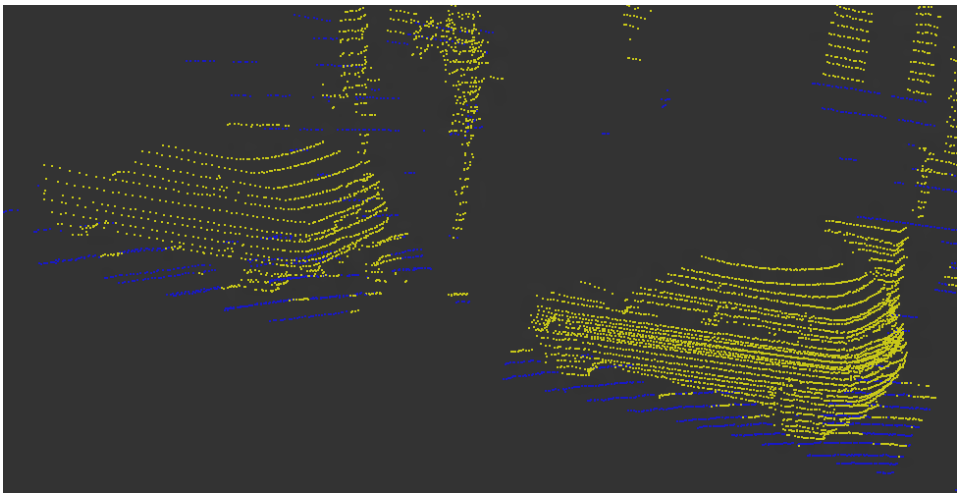


Figure 15: Cars with a layer of "carpet" under them. Blue color means it is classified as sparse/noise data

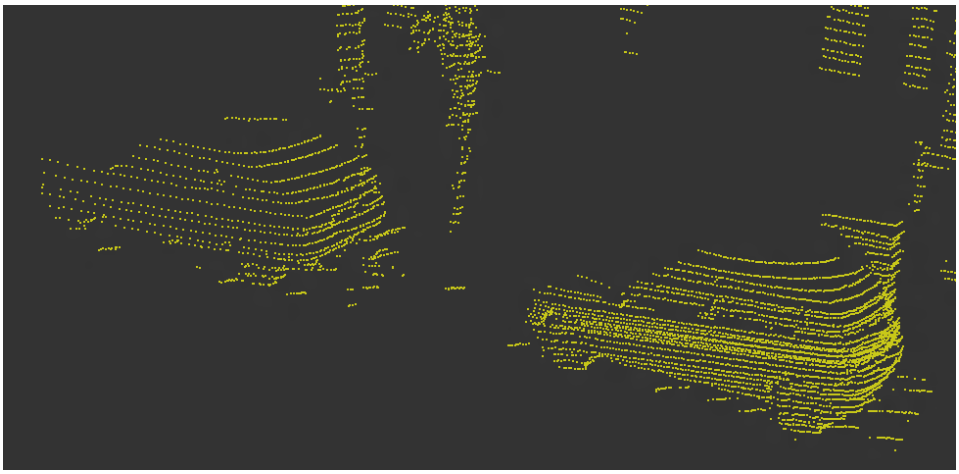


Figure 16: Cars without the detected road patches. It can be seen that this simple removal method is not sufficient to remove all the misclassified points

6.4 Results of our algorithm

In this section, we present some images showing the results of the point cloud registration done with the help of the pre-segmentation step we have developed.

Fig 11 shows an upper view of a large registered point cloud of a bridge (generated using the same data set that is shown in Fig. 10 where the registration failed without the presegmentation step). Fig 12 presents the process of the point cloud registration on a street segment, showing how radical improvement on the point cloud is possible. Lastly, Fig 13 and Fig 14 present two registration result, both recorded on the streets of Budapest.

7 Towards higher level scene interpretation

In this section we present various algorithms and test results of our work and some future plans, which point towards complex scene analysis and reconstruction. First, we introduce a point filtering algorithm, which is able to refine the grid based coarse point cloud classification step (Sec. 7.1). Then, we present our initial results on surface reconstruction and object detection in street scenes. Finally, models on higher level object and scene analysis are discussed.

7.1 More precise object clustering

As mentioned before, the grid is defined as a cuboid with a base of 50x50 to 80x80 centimeters. A classification is done for the whole cell meaning all points in the cell will be classified into the same class at all times. This means that if a wall is (correctly) classified as wall, some road points are (incorrectly) also classified as wall at the foot of the wall. These points are necessarily misclassified due to the nature of grid-based processing.

As Fig. 15 and 16 show, currently this small patch of misclassified road is simply removed from the data by cutting down a few centimeters of each cell that has object class. If possible, it should be further investigated and reclassified to road class if it belongs to the road surface.

7.2 Facade approximation with point clouds and triangle meshes

As mentioned before, the long-term aim is to build realistic, virtual 3D models of the streets of Budapest. For this task, the existing triangulation methods should be investigated and tested, as well as data smoothing methods. According to plans, later an RGB camera will be available - either a 360 ° field-of-view panoramic camera or a 360 ° looking multicamera system such as the PointGrey Ladybug camera[6]. Once such a camera is available, the point cloud can be colored and that additional color information could be used as a base of the 3D surface texturing methods.

After registering several point clouds against each other, the resolution can be high enough and uniform enough to create realistic building facade reconstruction. As the car

[6]Product description at http://www.ptgrey.com/products/spherical.asp

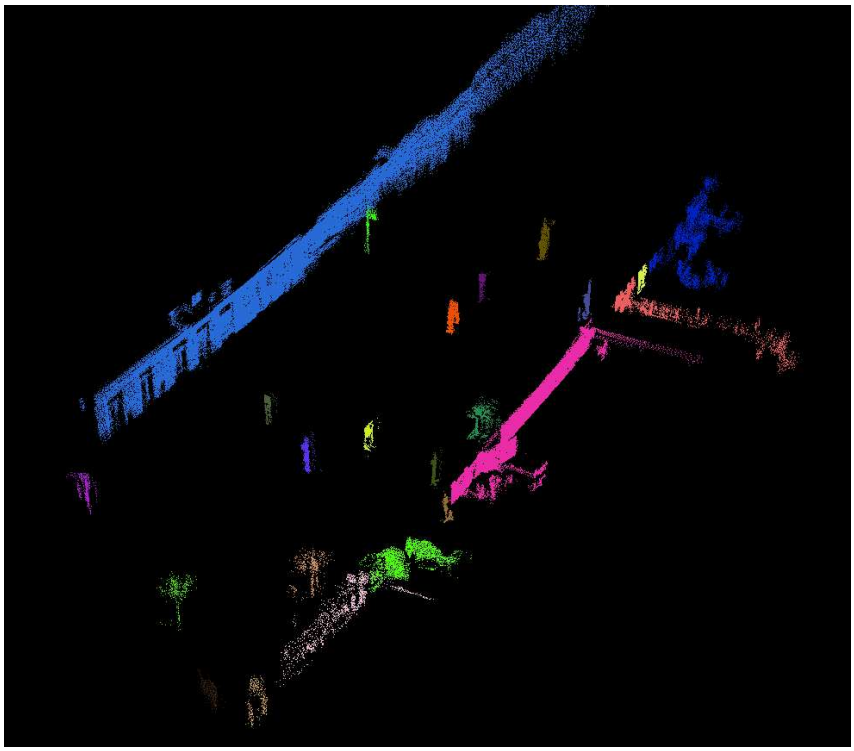


Figure 17: A clusterization result. Each color represents a separate cluster (colors do not have any particular meaning, they were assigned randomly).

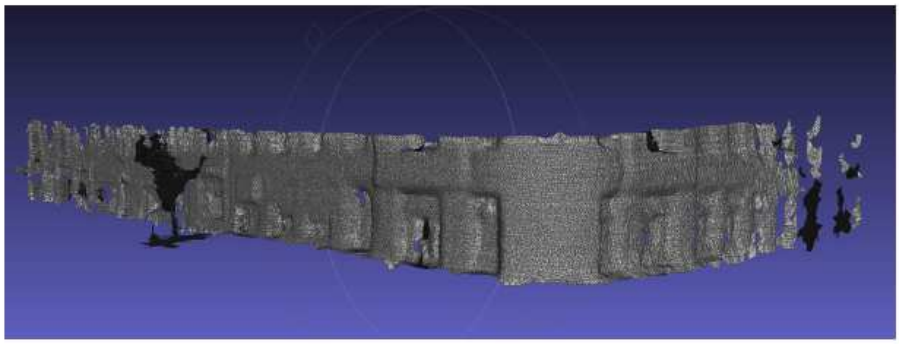


Figure 18: A triangulation result. It depicts a building on Móricz square.

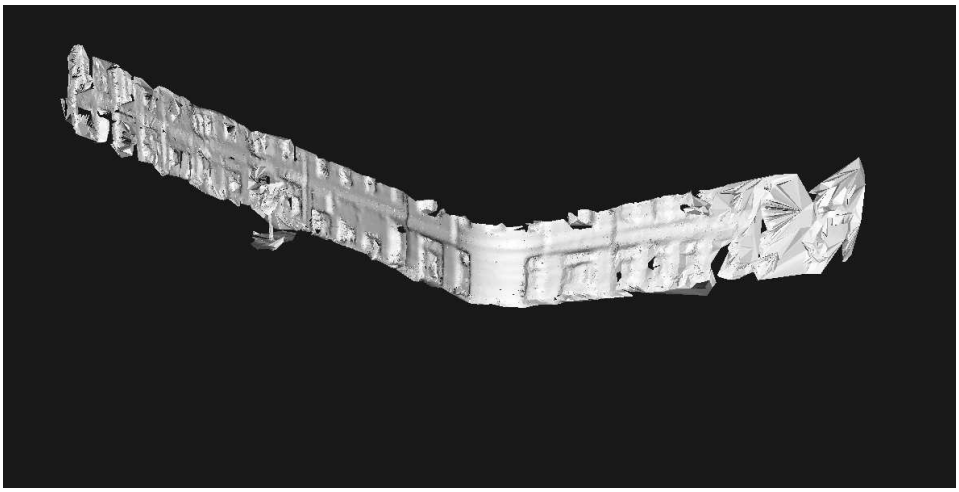


Figure 19: A second triangulation result for the Móricz square data sample

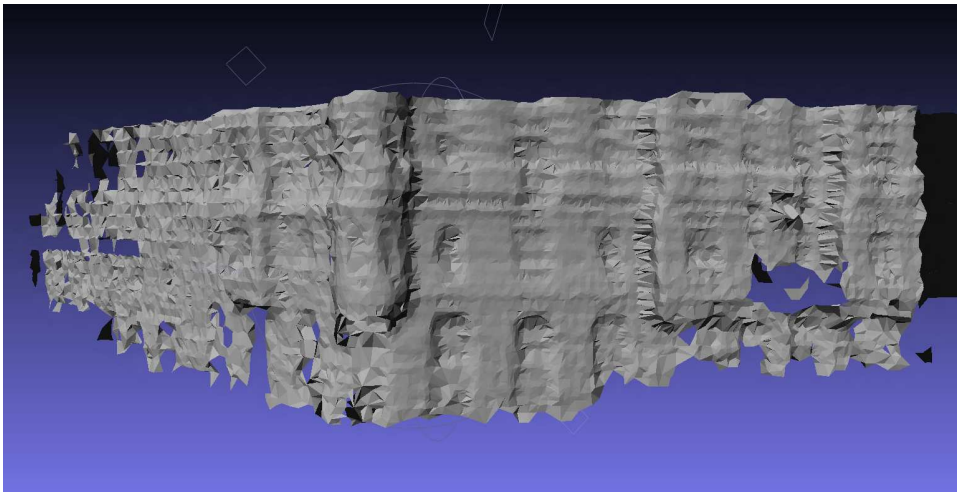


Figure 20: A reconstructed facade on Karinthy street. Though it looks noisy, the point cloud was highly sparse and once texturing is available, the visual experience will be much more realistic.

passes by a building, it collects data from several point of view so most of the wholes on the walls due to occlusion can be filled in. Also, after concatenating a few dozen scans, the resolution of the data will be significantly higher which results in a precise 3D reconstruction of wall surfaces. For noise reduction, we used the *moving least squares (MLS)* method. It is not edge preserving, but it is fast and highly helps the triangulation steps. After running MLS, the edges become a bit curvy, but after the texturing will be done, it will not have a great effect on the visual experience.

The current workflow we developed is as follows. After the transformation matrices are calculated via the NDT algorithm, they are applied only to the point clouds that contain the static tall points. After transforming these clouds into a mutual reference frame, we will get a large point cloud, which only contains walls, lamp posts, trees ans similar tall objects. On this larger, registered pointcloud a clusterization is done which separates the aforementioned objects into separate point clouds. With the help of a flood-fill algorithm (an Euclidean clustering in this case, see Fig. 17), the separate building components can be extracted from the data set.

Once we have the separate point clouds, each representing an object, we need to "clean" it and to smooth it. Cleaning is necessary because most likely there are outliers around the object and smoothing is required because of the noise. Since the resolution is high (millions of data points along a single facade), the noise can be eliminated and the reconstructed surface will be smooth enough.

After researching the available triangulation methods, the Ball-Pivoting algorithm [21] proved to be the best, though there is no available numerical measurement or analysis method for this but the human eye. Figures 18, 19 and 20 show three results of the triangulation step.

For this task, a tilted, upward looking LiDAR configuration is even more suitable than the forward facing configuration (see Fig 20 where the LiDAR scanned the building only up to about 3 meters of height).

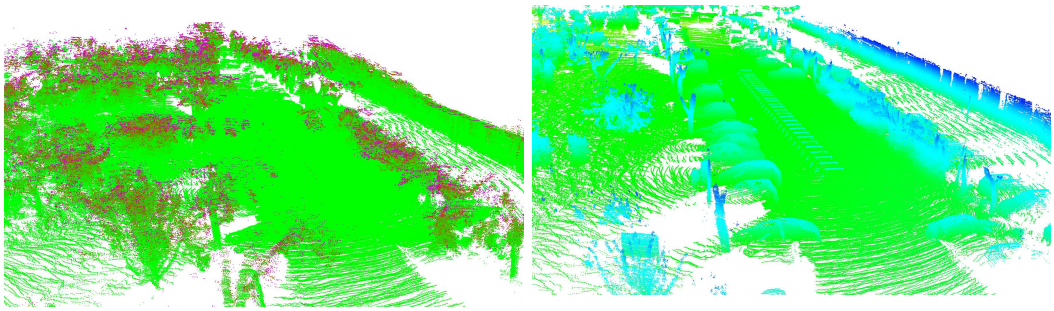
In the upward tilted case, the data is much more unfitting for traffic monitoring but more suitable for scanning buildings. As the sensor looks up it can scan the buildings up to their roofs. When registering this data, we can achieve incredibly high-detailed building facades.

We extended the framework for handle this different type of data and to be able to register these scans automatically, also. Results can be seen in Figures 21, 22 and 23.

Figure 21: An example of the tilted configuration. It depicts the Klotild Palace, near the Elizabeth bridge in Budapest

Figure 22: A block of building in the Fifth District of Budapest with amazing detail.

Figure 23: Example 3 - building of the Great Market Hall



(a) The detected leaf points are marked with red and purple.

(b) Point cloud after the leaf points are removed. The points are colored by their height for better visibility.

Figure 24: An example of vegetation removal

7.3 Vegetation detection

Another well defined task is vegetation (typically: trees and bushes) detection. The removal of the detected vegetation data from the point cloud can help detection algorithms, for example in the case of trees hanging over parking cars.

We have developed a multifeature descriptor for vegetation removal. It calculates a statistical feature for each point based on:

- the distance and irregularity of its neighbors (typically 20-40 neighbors)
- the intensity channel: vegetation reflects the laser beam with less intensity (though according to the tests, this feature is not as strong as expected)
- the height of the given point (as the main task is tree removal - bushes, grass, etc. is not as important to remove for object detection tasks)

Examples of the output can be seen on Figures 24, 25 and 26.

7.4 Traffic analysis

The next step of virtual city reconstruction is traffic interpretation. It is a highly researched area, and there are existing effective methods for aerial LiDAR data. After registering a whole street sequence, the data can be used similarly as of an aerial LiDAR data, with much higher resolution. For state of the art results on this topic please refer to the recent paper by Börcs et al.[22]. A longer overview and results in the topic of aerial LiDAR data can be found in [23].

For higher level traffic monitoring, efficient road interpretation and vehicle detection is necessary. Detailed information of the road surface is required for this task; such as dividing the road class into smaller semantic units e.g.: sidewalks, lanes, parking lots, road

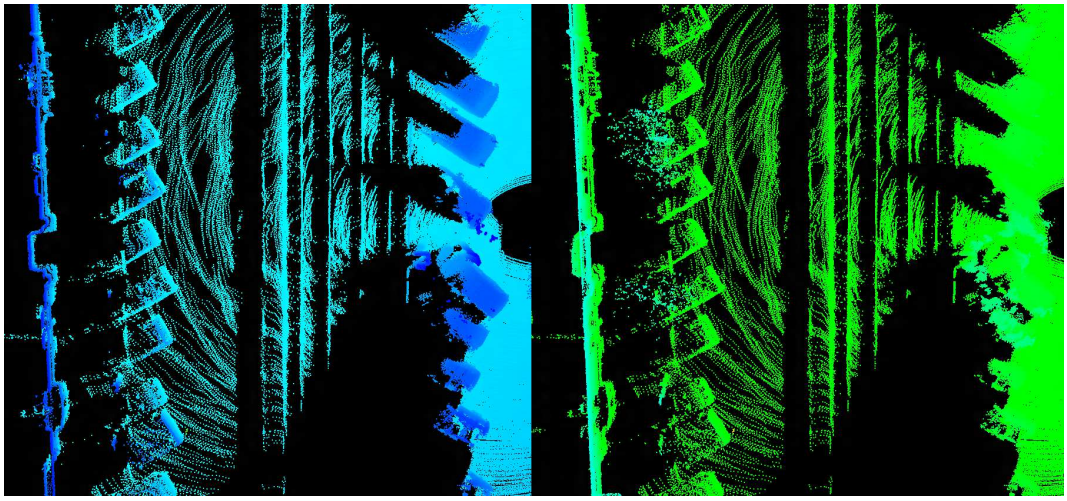


Figure 25: Second example of vegetation removal - original picture is on the right, "cleaned" is on the left. Again, majority of the leaf points are removed which should greatly help the car detection.

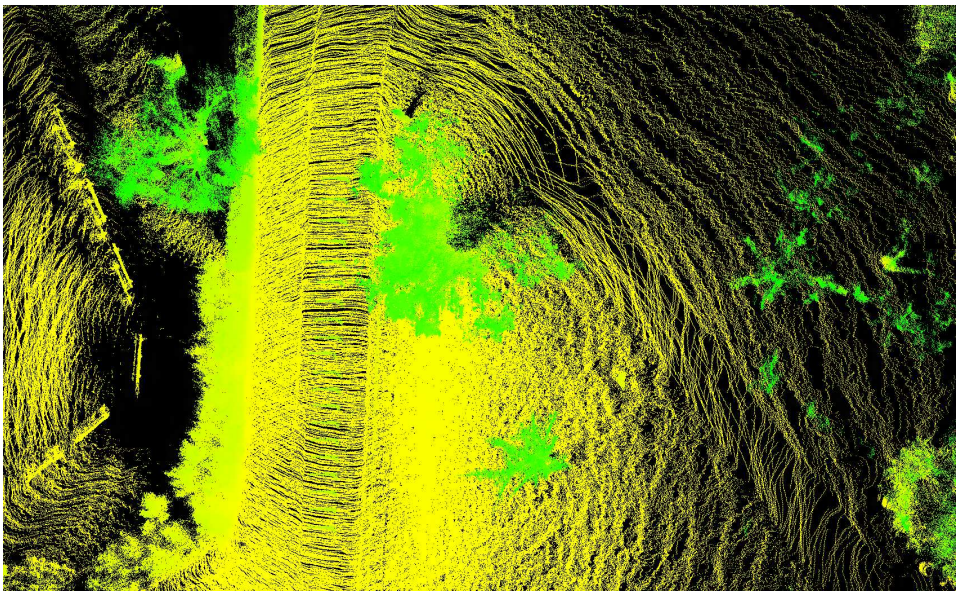
intersections. Lanes (and parking spots) can be identified by the lane markings. Lane markings are made out of bright painting which makes them fairly easily detectable. Also, later RGB information can be used to support this method.

7.4.1 Analyzing moving and static objects on merged point clouds

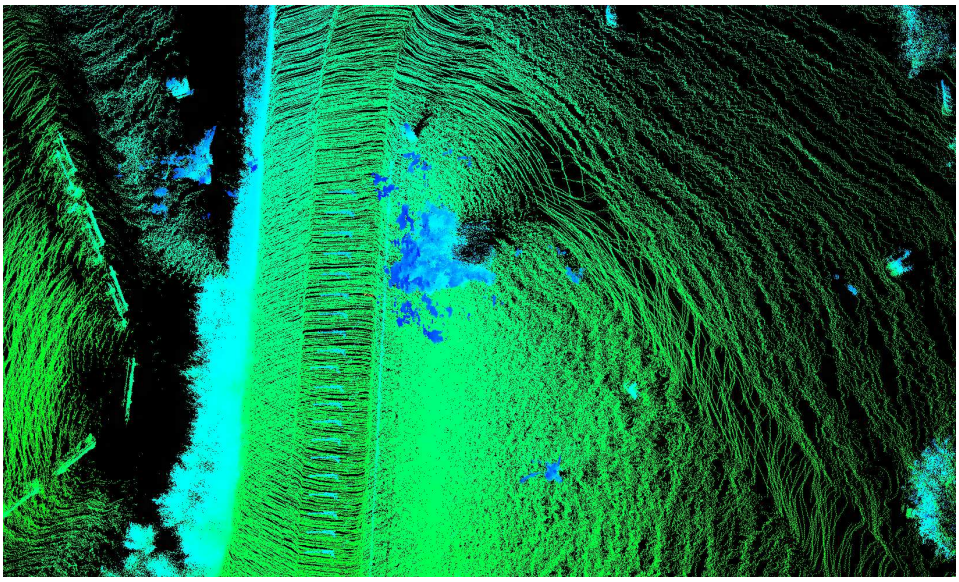
General 3D object reconstruction is a much harder task than reconstructing walls or road surface. While these have a well defined, and mostly flat surface, a general street object can have virtually any shape and due to small resolution and the fact that the LiDAR only scans one side of the object, the reconstruction is sometimes impossible.

Moving objects are even harder to reconstruct based solely on LiDAR data. As these objects (typically vehicles, people) are moving through the scene, they appear at different spots in each scan which make them appear like a long-drawn shadow in the registered point cloud.

Instead of reconstructing these objects from the LiDAR cloud, our plan is only to detect them in the cloud. Once they are detected, they can be replaced with realistic, full-scale 3D models in virtual space. Free 3D models of vehicles, vegetation, trash cans, etc are available to use for this purpose. Also, the Geometric Modeling and Computer Vision Laboratory of the Computer and Automation Research Institute has the capabilities to scan dynamic



(a) Before vegetation removal



(b) After removal - most of the overhead tree leaf points are removed, though not all of them

Figure 26: Third example of vegetation removal

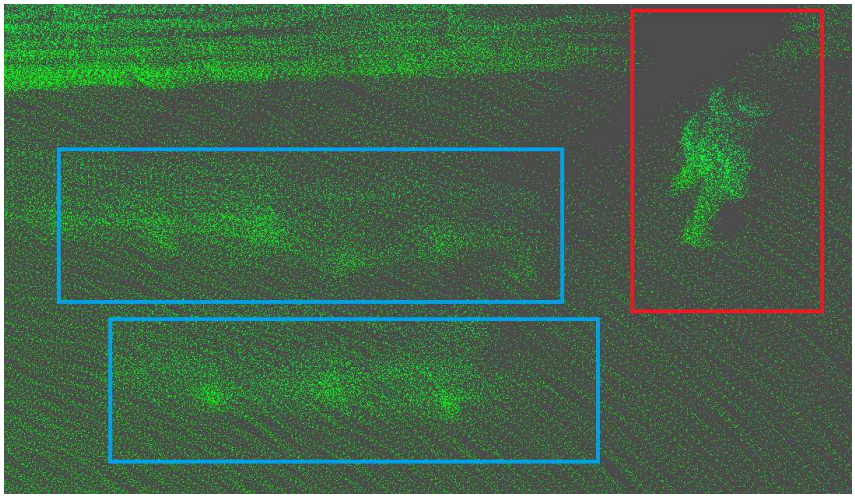


Figure 27: An example of how moving persons look like in the registered point cloud (the two blue boxes) versus a person who was standing still during the whole scanning process (red box)

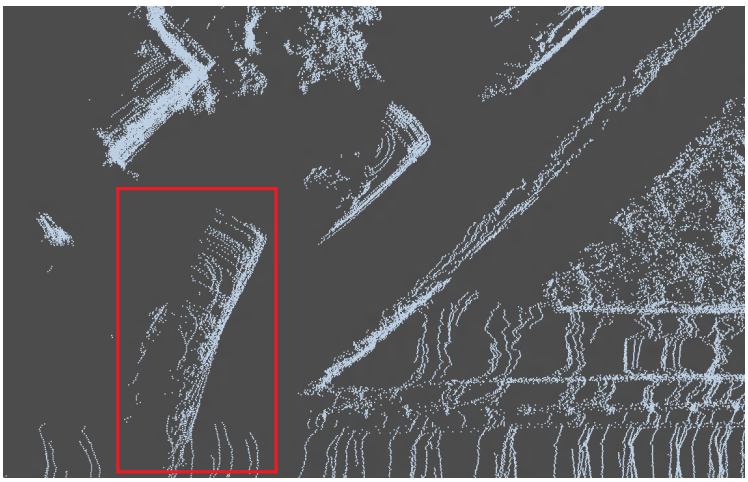


Figure 28: Another example of moving objects appearing as a smudge in the registered cloud. Here, a car turned onto the main street after stopping at the red light

Figure 29: Typical centroid points of a moving car in the registered cloud

(a) Centroid points inside a car

(b) Centroid points inside a car

Figure 30: Typical centroid points of standing cars in the registered cloud

models, such as walking people. These dynamic models can be inserted into our virtual space to achieve a realistic 3D world.[7]

The first step here is to modify the registering algorithm so it will support the distinction between standing and moving vehicles. For this we implemented a timestamping algorithm that extracts and identifies the original separate scans in a detected vehicle blob. After detection, we calculate the 3D centroids of these separate segments. First tests show that the distribution of these centroids using 10-30 scans is a very effective marker in moving and standing vehicle distinction (more tests and evaluation will be done later on).

Figures 29, 30 and 31 show the results of this development.

7.4.2 Traffic sign detection

A popular task on urban scenes is traffic sign detection. It is a crucial task in intelligent vehicles but also can be used by route authorities to inspect whether the necessary traffic signs are present where they should be.

The possibilities of traffic sign *detection* should be investigated in two aspects:

[7]For initial results on this topic, see the demo video at the website http://web.eee.sztaki.hu/home4/node/14

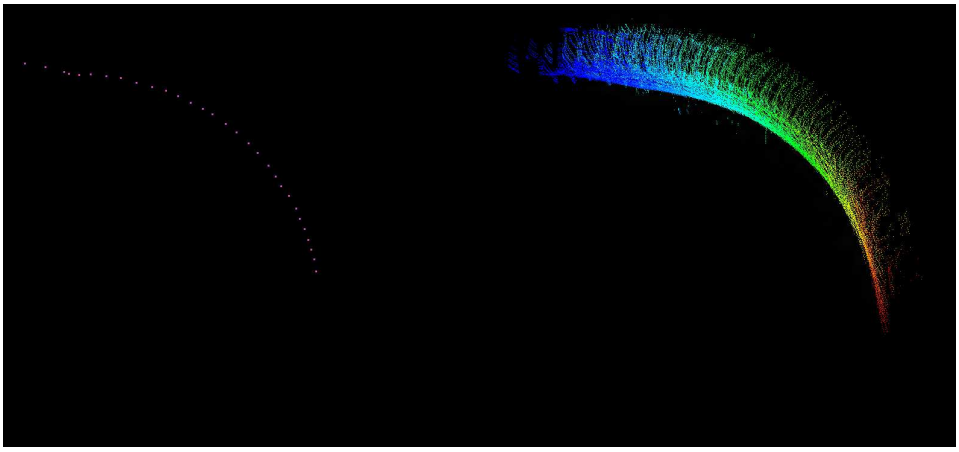


Figure 31: A turning car colored as a sequence in the registered, concatenated point cloud. Left side of the picture shows the centroid points (trajectory)

- **Intensity channel** - traffic signs have highly reflective paintings on them for better visibility. This can be a strong indicator of a traffic sign.
- **Physical constraints** - as the walls or the road surface, a traffic sing also has a well definable physical constraint. A model should be developed to represent these constraints. Also it should be investigated whether this model should be used in the existing grid-based processing or on full clouds without grid representation.

Even further development is traffic sign *identification*. For this task, a simple 3D data is not enough, color information is also needed to identify similarly shaped but different traffic signs. Once RGB referenced 3D data and the traffic sign detection method is available, the possibilities of pattern matching for identification should be investigated.

8 Framework for displaying and processing data

8.1 Development tools

The framework is written in the C++ programming language on the Microsoft Windows platform (Visual Studio 2010) but using only multiplatform technologies and software components.

8.1.1 Point Cloud Library

The Point Cloud Library (PCL)[8] is a free, open source C++ library for point cloud processing. It is available for all major platforms and the source code is licensed with the BSD license. It is developed by Willow Garage and many universities around the Globe and financially supported by companies such as Google, nVidia, Trimble, Velodyne. The library provides both high and low level 3D point cloud processing algorithms. For details on this software library, please refer to [24]

The library also defines a widely used file format for storing point clouds. It is a simple format containing a header and the 3D information, functions to read and write this format are available in the library. [9]. We store the Velodyne LiDAR data in this file format. It is flexible enough to contain supplementary information about each point such as labels for the classes, normal and curvature data, etc. The data is stored in binary format to speed up disk usage time. Reading and saving LiDAR data is done with the help of the PCL's input-output library. For the 3D transformation data, we constructed a simple file format which contains the calculated transformation matrices between the consecutive point clouds.

Functions we used from the PCL library:

- Input - output module (for reading *.pcd* files)
- Kd-trees for search within clouds
- Statistical outlier removal
- Visualization module
- Random Sample Consensus (RANSAC) model for comparison for road and wall detection
- Normal Distributions Transform to register point clouds (NDT is not yet available in the stable PCL library release at the time of writing but the code can be dowloaded from the)
- Euclidean Cluster Extraction for clustering the objects in a point cloud

We used PCL version 1.6.0, the latest version at the time of writing this report.

8.1.2 Boost library

The Boost library[10] is a well-known, industry-standard extension of the C++ language. It is installed by the PCL library by default. We used the Boost functions for smart pointers, parallelisation and for the signal-slot communication between the software threads.

We used Boost version 1.49.0.

[8] Webpage: http://pointclouds.org/
[9] The detailed description can be found on the webpage
[10] webpage: http://www.boost.org/

8.1.3 Doxygen

The source code is documented using the Doxygen documentation system[11]. The source and documentation is available on the attached DVD (see Appendix A for the descrtiption of the attachement).

8.2 Implemented framework

8.2.1 Descrtiption of the framework

The core of the framework is the cell-based segmenter class. It is responsible to build the grid and to calculate the statistical information about the point cloud and each cell. Also, it is responsible to segment the points (cells) into classes and color them accordingly for visualization.

The framework has a simple display module for visualizing data and the results of the algorithms. It is built atop of PCL's visualization library (which uses the Visualization Toolkit library's core functions). The display module also uses the Boost library's parallelization library; we implemented visualization and keyboard handling to run in separate threads.

The program offers command line input functionalities also. The display module can be turned off via starting the application with the *-novis* option. This way, the application will behave as a command line processing tool (additional command line options are: *-help* for displaying usage information *-verbose* for displaying statistical information about the processed data, *-o* for turning on a basic statistical outlier removal function on the point cloud and *-n* for turning on normal calculation.).

Available display modes:

- **Stream mode** to display data sequence without processing or coloring. This mode is useful for playback of 4D data
- **Simple Segment** class coloring. In this mode, the grid based segmentation will be performed and the point cloud will be colored according the pictures shown before.
- Linear coloring based on **average height** in a cell. In this mode, the processing function calculates the average height within a grid cell and colors the point cloud accordingly (similarly to a heatmap)
- Linear coloring based on the **point density** in a cell. In this mode one can easily check visually the point density distribution in a point cloud.
- Linear coloring based on **minimum-maximum height difference** in a cell. Similarly to the previous two, the processing module calculates a statistical information about each cell and colors the cloud accordingly. Useful for visually evaluating the height differences.

[11] webpage: http://www.stack.nl/ dimitri/doxygen/

The framework has some additional functionalities for evaluation and helping development. If the user clicks on a point, it displays information about the point and the grid cell the point belongs to. Also, the threshold parameters can be changed during 4D playback for faster evaluation. If the file save flag is set to true, the framework outputs the data into several files after processing the input:

- A colored point cloud. The colors correspond to the four classes.
- A point cloud containing only the walls and high objects.
- A point cloud containing only the street objects.

And finally, the registration workflow is as follows: first, the point clouds containing only the static wall points are loaded. Then going through this list of point clouds each pair is registered each against with the Normal Distribution Transform. Once this is done, we know all the transformation matrices that represent a single transformation between the coordinate systems of consecutive scans. Using these matrices, we can transform any point cloud (the full cloud, cloud with only the walls, etc) into one mutual coordinate system. After this, we only have to concatenate the point clouds and save the result onto the hard drive.

8.2.2 Implementation details

As Fig. 32 shows, the program consists of 3 main modules.

The **IO module** is responsible for reading the *.pcd* PCL files, for displayig the 4D data as a 3D video stream (with keyboard and mouse functionalities such as turning features on and off) and for parsing the command line input and saving the input parameters.

The **Point Cloud Processing module** contains the implementations of the cell grid structure. These are responsible to build the grid representation, to calculate the statistical information and to segment the data. There is a separate class for the remaining point cloud task such as putting the grids back together into a whole point cloud.

Also there is a **statemachine class** that is used by the two previous modules containing all the important variables and constants (Fig. 33 is a screen from the documentation showing all these parameters).

A future task is to refactor the code and make the structure of the program more object-oriented.

8.3 Output quality

As it can be seen in the pictures above, successful point cloud registration can be done on the segmented, preprocessed data. The registration is accurate and can be easily performed on the static points.

The big advantage of the road surface detection method over a RANSAC-based plane matching that it can deal with curvatures, elevation changes in the scene. RANSAC can

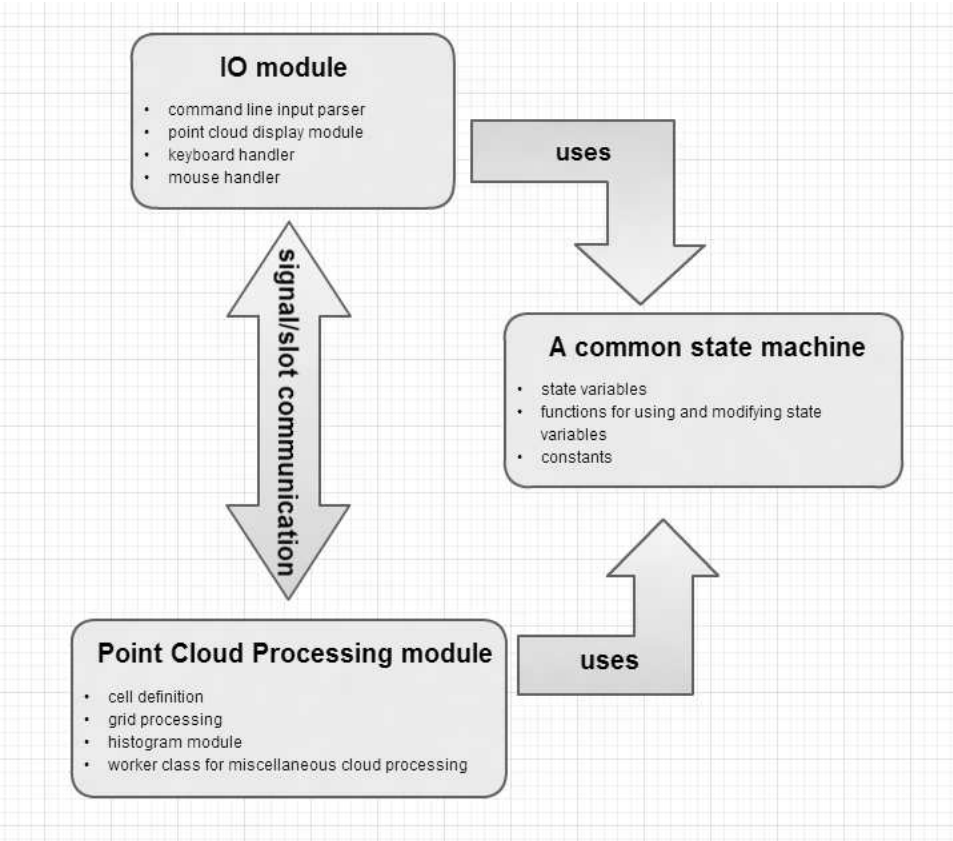


Figure 32: Outline of the program structure

Public Attributes

int	**fileNum**
	sequence number (where are we at at the processing)
int	**startFileNum**
	the end of the file sequence to open
int	**endFileNum**
	the end of the file sequence to open
bool	**running**
	is it running ot not
int	**mode**
	display mode
bool	**groundDisplay**
	display parameters
bool	**wallDisplay**
bool	**objectDisplay**
bool	**sparseDisplay**
bool	**save**
	save output files to disk
bool	**screensave**
	save player screenshots to disk
bool	**visualization**
	whether to open the viewer or not
bool	**verbose**
	verbose mode or not (command line output)
bool	**pause**
	is the running paused or not
bool	**tenmeter**
	whether to mark the points that are closer than 10 meters
bool	**fivemeter**
	whether to mark the points that are closer than 5 meters
int	**triangmode**
	triangulation mode, 0 if no triangulation
float	**voxelsize**
	voxel grid size
bool	**objectcluster**
	whether to cluster the segmented objects (flood fill)
boost::mutex	**updateMutex**
boost::mutex	**gridMutex**
bool	**outl**
int	**param**
bool	**wallreconstruct**
int	**cellsize**
int	**normal_neighbors**
int	**outlier_neighbors**

Figure 33: State variables of the program

Dataset name	Number of scans	Speed with presegmenatation (s)	Without presegmenatation (s)
Tas Vezér street	10	0.3209	5.3061
Tas Vezér street	10	0.2709	6.8782
Bartók Béla street	10	0.5888	6.3983
Bartók Béla street	15	0.5056	5.8474
Bartók Béla street	15	0.0868	4.0099*

Table 1: Speed comparison. * denotes failed registration

find the best fitting plane for the road surface but in real world, the road surfaces are not planars, especially in cities built on or next to a hill. Using the proposed method, such difficulties can be overcome and once the surface grid cells are detected, even an elevation map can be built.

One other benefit of the grid based segmentation that is also removes noise. Grids containing too few points are not useful, they make computation harder. The segmentation filters these points out also.

8.4 Processing speed

For the segmentation step, the main aim was to implement a fast algorithm that performs a presegmentation step. This aim has been met; once the data is in the memory (reading from the disk is the slowest step), the algorithm runs in real time on a modern CPU.[12]

Besides the segmentation being fast, it fastens up the registration step also. Once all the noise and interfering moving points are removed, the registration is easier, thus faster. As the registration algorithm tries to match up points, the overall error will be smaller thanks to the "clean" point cloud and the algorithm will converge faster towards the termination criteria. Registering point clouds containing only stable, stationary points takes only 1-4 seconds (this value depends on the scanning rotation speed and on the amount of points left in the cloud after segmentation). Registering a full cloud takes 5-10 second - if it converges, since in some cases the registration fails.

8.5 Robustness

The proposed workflow is robust enough to perform extremely well in real life environments (some experiments on this topic use only more sterile indoor or even more sterile artificial data sets).

The algorithm has been tested in lots of data sets as summarized in Table 2. The proposed workflow can concatenate these whole street segments - although the resulting point clouds can be hard to work with due to their size (hundreds of megabytes and tens of millions of data points). For typical use, 10-50 concatenated point clouds is reasonable.

[12]The test environment has an Intel Core i7 processor and 8 gigabytes of DDR3 RAM

Street name	Number of scans
Kende street	496
Bertalan Lajos street	154
Bartók Béla street	581
Villányi street	646
Tas Vezér street	345
Edömér street	24
Diószegi street	824
Dávid Ferenc street	972
Szüret street	770
Somlói street	945
Liberty Bridge	160
Andrássy street	125

Table 2: The data sets used to test the algorithm. Most of the data recordings had been done in the 11th District of Budapest

9 Conclusion

The results above show the success of the simple, yet useful presegmentation step that has a great positive effect on the point cloud registration. Dividing the initial point cloud into multiple semantic classes and using only the relevant points results in a faster, more stable registration.

Also, the proposed software framework has been successfully implemented atop of the PCL library for handling, processing and visualizing the Velodyne LiDAR data. The future development ideas have been presented and will be implemented in the near future.

References

[1] T. Luettel, M. Himmelsbach, and H.-J. Wuensche. Autonomous ground vehicles: Concepts and a path to the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1831 –1839, 13 2012.

[2] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[3] N. Wojke and M. Haselich. Moving vehicle detection and tracking in unstructured environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3082 –3087, may 2012.

[4] A. Azim and O. Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 802 –807, june 2012.

[5] B. Douillard, J.P. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the segmentation of 3D LIDAR point clouds. In *International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[6] B. Douillard, A. Quadros, P. Morton, J.P. Underwood, M. De Deuge, S. Hugosson, M. Hallstrom, and T. Bailey. Scan segments matching for pairwise 3d alignment. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3033 –3040, may 2012.

[7] A. Makadia, A.I. Patterson, and K. Daniilidis. Fully automatic registration of 3d point clouds. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1297 – 1304, june 2006.

[8] J. Behley, V. Steinhage, and A.B. Cremers. Performance of histogram descriptors for the classification of 3d laser range data in urban environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4391 –4398, may 2012.

[9] A. J. Quadros, J. P. Underwood, and B. Douillard. An occlusion-aware feature for range images. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4428–4435, St. Paul, MN, USA, 2012.

[10] C. H. Shen, S. S. Huang, H. Fu, and S. M. Hu. Adaptive partitioning of urban facades. In *SIGGRAPH Asia Conference*, pages 184:1–184:10, New York, NY, USA, 2011. ACM.

[11] R. Wang, J. Bach, J. Macfarlane, and F.P. Ferrie. A new upsampling method for mobile lidar data. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 17–24. IEEE, 2012.

[12] A. Velizhev, R. Shapovalov, and K. Schindler. An implicit shape model for object detection in 3D point clouds. In *ISPRS Congress*, Melbourne, Australia, 2012.

[13] X. Hu, X. Li, and Y. Zhang. Fast filtering of lidar point cloud in urban areas based on scan line segmentation and gpu acceleration. *IEEE Geoscience and Remote Sensing Letters*, 10(2):308 –312, March 2013.

[14] M. Magnusson. *The Three-Dimensional Normal-Distributions Transform — an Efficient Representation for Registration, Surface Analysis, and Loop Detection.* PhD thesis, Örebro University, December 2009. Örebro Studies in Technology.

[15] P. Biber and W. Strasser. The normal distributions transform: A new approach to laser scan matching. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2743–2748, Las Vegas, USA, October 2003.

[16] J. Han, D. Kim, M. Lee, and M. Sunwoo. Enhanced road boundary and obstacle detection using a downward-looking LIDAR sensor. *IEEE Transactions on Vehicular Technology*, 61(3):971 –985, march 2012.

[17] S. Thrun, M. Montemerlo, and A. Aron. Probabilistic terrain analysis for high-speed desert driving. *Proc. Robotics Science and Systems, Philadelphia, PA, USA*, 2006.

[18] A. Petrovskaya and S. Thrun. Model based vehicle tracking for autonomous driving in urban environments. *Proceedings of Robotics: Science and Systems IV, Zurich, Switzerland*, 34, 2008.

[19] C. Stiller and J. Ziegler. 3d perception and planning for self-driving and cooperative automobiles. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1–7. IEEE, 2012.

[20] W. Zhang. LIDAR-based road and road-edge detection. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 845 –848, June 2010.

[21] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 5(4):349–359, 1999.

[22] A. Börcs and C. Benedek. Urban traffic monitoring from aerial LIDAR data with a two-level marked point process model. In *International Conference on Pattern Recognition (ICPR)*, Tsukuba City, Japan, 2012.

[23] A. Börcs and Cs. Horváth. Városi környezet automatikus analízise és rekonstrukciója légi lidar mérések alapján, 2011. TDK Dolgozat.

[24] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

Departments of the institute
http://www.sztaki.hu/departments/

3D Internet-based Control and Communications Laboratory, Cellular Sensory and Optical Wave Computing Laboratory
Computer Integrated Manufacturing Laboratory, Department of Distributed Systems, ELearning Department
Distributed Events Analysis Research Laboratory, Geometric Modelling and Computer Vision Laboratory
Informatics Laboratory, Internet Technologies and Applications Department, Laboratory of Parallel and Distributed Systems
Network Security Department, Research Laboratory on Engineering & Management Intelligence, Systems and Control Lab

MTA SZTAKI Computer and Automation Research Instititute
Hungarian Academy of Sciences
Kende utca 13-17 H-1111 Budapest (Hungary)
http://www.sztaki.hu