Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Balázs Nagy

Computer Engineer MSc

3D scene understanding based on mobile laser scanning systems

Supervisor:

Csaba Benedek PhD

Budapest, 2016

Alulírott Nagy Balázs, a Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Karának hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, és a diploma munkámban csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem. Ezt a Diplomatervet más szakon még nem nyújtottam be.

....

Nagy Balázs

Table of Contents

Kivonat

Napjainkban egyre nagyobb automatizálási kényszer figyelhető meg az élet számos területén. A hatalmas adattömeg kezelése és feldolgozása, melyet a különböző szenzorok állítanak elő egy olyan kihívás, amit emberi erőforrással nem lehet hatékonyan megoldani. Ez a tendencia a vizuális szenzorok területén is megmutatkozik. Az elmúlt pár évben a különböző lézerszenzorok és a háromdimenziós térképező rendszerek óriási fejlődésen mentek keresztül. Várhatóan a széleskörű felhasználhatóságuk és a nagy precizitásuk miatt a sorozatgyártás után az áruk is drasztikusan csökkenni fog, ahogy ez más szenzorok esetében is megfigyelhető volt, például az optikai kamerák esetében.

A Velodyne Lidar szenzorjait önjáró járművek navigálásához tervezték. A szenzor egy 360°-os 2,5 dimenziós pontfelhőt állít elő a környezetéről, akár 15-20 időkeret/másodperc sebességgel. A Google önjáró járműjei már több mint öt éve használják a Velodyne lézerszkennerjeit, de a különböző nagyértékű munkagépek például a Dubaji olajkitermeléseknél használt dömperek is fel vannak szerelve a szenzorral. Önjáró járművek és különböző megfigyelési feladatok (térfigyelő rendszerek) esetében a valós idejű feldolgozás az egyik legnagyobb kihívás.

A különböző városrekonstrukciós, mérnöki (hídtervezés) és úthálózat karbantartási feladatok esetében a valós idejű feldolgozás nem játszik kulcsszerepet, de a feldolgozási folyamatok automatikus gyorsítására elengedtethetetlenül szükség van. Ezen típusú térképező rendszerek nagyon pontos és nagy sűrűségű színes pontfelhő részleteket állítanak elő, melyeket egy közös geo-referált koordinátarendszerbe transzformálnak. Az így kapott több 100 millió vagy akár milliárd pontot tartalmazó pontfelhőkben az egyes objektumok automatikus detektálása, vagy az adathalmaz zajmentesítése hatékony automatikusa algoritmusok létrehozását igényli.

Dolgozatomban bemutatom az iparban is használt piacvezető lézerszenzorokat és a mobil térképező rendszerek esetén bemutatom, hogyan lehet a kamerakalibrációs információkat felhasználva a kamerák 2D képei és a 3D pontfelhők között egyértelmű megfeleltetést találni. A további fejezetekben a pontfelhő feldolgozás alaplépéseit mutatom be, úgymint pontfelhő tisztítás, szegmentáció, objektum detekció és objektum felismerés, továbbá áttekintést adok a szakirodalom aktuális eredményeiről. A fejezetek között egy jelzőtábla detekciós algoritmus bemutatására is sor kerül, mely egy ipari rendszerbe is beépítésre került.

Munkám során a RIEGL és a Velodyne cég szenzorjai által készített adatokon dolgoztam, melyet az MTA SZTAKI EEE laborja és a Budapest Közút Zrt. biztosított.

Abstract

Security and surveillance issues in dynamic urban environments receive a major attention in various application fields, such as traffic management and control environment protection, accident and crime prevention. To ensure an optimal exploitation of environmental information, besides traditional image sensors, real-time 3D data sources are frequently used nowadays. A Lidar laser scanner mounted on the top of a moving vehicle provides a measurement sequence, where each frame is a three-dimensional point cloud, and the coordinate system of the consecutive point cloud frames is continuously updated, following the vehicle's motion.

Urban environment planning is another interesting applications of the Lidar laser scanners. To ensure the best coverage, fixed and mobile mapping systems are often used in parallel. However, the sensors utilized in such tasks, differ significantly from the above mentioned real time scanners: they are designed for collecting homogeneous, high precision and dense point clouds, which can be exploited through offline processing. Dealing with urban planning tasks includes 3D city reconstruction and road survey and registering street furniture components (benches, dustbins) and traffic controllers (traffic lamps and sings) into databases.

Thus, using Lidar scanners we can directly obtain accurate 4D (space + time) geometric information from the scene, however, the analysis of the point clouds presents several challenges for the automatic pattern recognition and reconstruction algorithms.

I will mainly deal with the measurement of two specific state of the art Lidar systems (namely (RIEGL VMX-450 and RIEGL VZ400), which can capture homogeneous and high density point clouds, moreover by recording optical photos in parallel with laser scanning, they can assign RGB information to the captured point clouds. To achieve this goal, I implemented a software tool for projecting the 3D points on to the corresponding images using the camera calibration parameters, and I also developed a frustum based back projection algorithm which enables the accurate 3D localization of objects recognized in the image domain. The second part of the work gives insight of various point cloud processing techniques, in part using the above mentioned RIEGL scanners, and in part with utilizing Velodyne's real-time Lidar sensors. First I present point cloud filtering and segmentation methods, then I propose several object detection techniques in point clouds, and in the last section of my work I introduce a feature extraction method and a convolutional deep neural network for identifying various objects such as car, pedestrian and street furniture in point clouds. At the end of each section, an application example is

given, which is part of an algorithm pipeline of road sign filtering, segmentation, detection and classification as a realization of a complex urban planning application.

As mentioned above, in the course of my work, I used point clouds obtained by the RIEGL VMX-450, RIEGL VZ400 and Velodyne HDL64-E Lidar devices. The point cloud data was recorded on the streets of Budapest and it was provided by Budapest Közút Zrt. (RIEGL scans) and MTA SZTAKI (Velodyne data).

My aim has been to develop and implement such methods and algorithms, which are able the use 2D image and 3D space information simultaneously. I have shown that the combination of different type of information allows to create more accurate models and we can solve more complex problems.

1. Introduction

In the last decade the capacity of the hard drives, and the performance of the CPU and GPU units have significantly increased thus managing a huge amount of data can be achievable. Nowadays, there are a lot of applications where observing large city scenes or areas is a crucial need.

Due to the above developments, in the recent years Lidar devices have evolved very quickly. Their performance became higher, while their size and weight declined steadily. Today, a lightweight Lidar sensor can be mounted even on a drone. Therefore, we can scan out-of-the-way and dangerous areas, furthermore large progress has been made in the field of sensor fusion, sensor calibration and data registration. As part of a calibrated mobile or terrestrial mapping system certain Lidar sensors such as the RIEGL VMX 450 and RIEGL VZ400 produce homogenous, high density point clouds with RGB color information assigned to each point. Then the colorized point clouds are transformed into a global world coordinate system so the result is a registered colorized high density and accurate point cloud.

In this thesis I propose novel algorithms dealing with various problems of point cloud processing and 3D scene understanding. During the work I dealt with point clouds produced by different types of sensors such as the RIEGL VMX-450 multi-sensorial mobile mapping system and the Velodyne HDL 64-E real time laser scanner.

In Section 1 and 2, I give an overview on the bases of laser scanning and I introduce the sensors I used in this work. Section 3 introduces the theoretical bases of sensor and data fusion and I present a method which I implemented to project the 3D points onto image domain and back. Section 4 deals with the bases of point cloud filtering and it proposes an efficient traffic sign filtering algorithm. In Sec. 5 I propose a novel hierarchical data structure and several algorithms for point cloud scene segmentation and object separation. Section 6 deals with object recognition in point clouds. Here I propose an efficient feature extraction method and a deep learning architecture for classifying various urban objects such as vehicles, pedestrians and street furniture.

Furthermore, through the sections I describe a traffic sign detection pipeline - integrated already into Budapest Közút's industrial system – as a realization of the mentioned algorithms.

1.1 Applications

Point cloud processing involves lots of opportunities and industrial trends show that it can be used by different research areas. The high geometric accuracy of point clouds obtained by laser scanning is already used in many engineering applications, such as designing bridge and building structures, detecting material errors or monitoring structural changes. The technology allows the detection accuracy of centimeters or even millimeters [1].

Another interesting application area is urban planning. It can be divided into two main parts. The first one is engineering design (as mentioned above) and the second part is street furniture annotation and registration into databases. These databases help the traffic management authorities to create accurate statistics, which contributes to the improvement of transport and the maintenance of the road network.

A few years ago the entertainment industry also started to utilize laser laser scanning technologies. One direction is the cinema and advertising industries, and some new generation computer games also use point cloud based engines for environment modeling. The bottleneck of the technology is the real time point cloud rendering, but today there are particular industrial solutions, such as Euclideon's Unlimited Detail engine [2] which is able to deal with the problem.

The colorized point clouds mentioned above may have very high accuracy and high density to achieve the best visual experience. On the other hand, applications such as surveillance, autonomous driving and robotics demand real-time processing and decision making, whose Lidar sensors produce lower density point cloud sequences with high update frequency. The flagship company is today the Velodyne Inc. in this segment. Their sensors are used by Google driverless cars and by several winners of the DARPA grand challenges.

1.2 Sensors in 3D technology

Beside Lidar technology, several sensors play role in the field of 3D data capture. One of the most well-known 3D sensors is the Kinect manufactured by Microsoft. It is a depth sensor and in the first place it is designed for Xbox for the game industry. Although, it is the cheapest sensor we mention, but it produces relatively dense and accurate point cloud from range 0.5 to 4.0 meter, so several research projects use its data. Time-of-flight (ToF) cameras such as Mesa are another sort of data providers in 3D vision. It is a range imaging camera system from 5 to 10m maximum scanning distance but its price is about ten times more expensive, than Kinect which only costs $100. The stereo vision concept uses multi camera system to extract the 3D information from the scene. Each optical camera observes

the same scene from a slightly different viewpoint, and using the calibration information the platform matches the relative object positions in the different images and calculates the 3D coordinates.
Generally, Lidar is a more expensive technology, than the others mentioned above but it is more accurate and it works well in natural lighting conditions. Furthermore, its detection range can be much larger than the others, for example the Velodyne HDL-64E Lidar sensor may acquire data from 150 meters. Using Kinect and ToF cameras in outdoor environments is practically impossible because they operate in near infrared range and the light of the Sun blind them, moreover their scan range is smaller than 10 meters. Though stereo systems work well in outdoor and see farther than 10 meters but their installation and calibration process is very complex and time-consuming.

1.3 Lidar

The term Lidar was created as “light” and “radar”. So Lidar is a radar that measures distance by emit a laser beam and analyzing the echo time of the laser. In practice, laser scanning can be divided into four main categories: aerial, hydrographic, terrestrial and mobile laser scanning. The airborne laser scanning enables the mapping of large areas therefor it is often used in urban planning and vegetation survey. By using terrestrial Lidar technology (due to the high point density) more accurate and largely detailed 3D models can be created, which properties are crucial requirements in architectural and engineering applications. Finally, mobile laser mapping allows quick surveys of the road network and environment, furthermore it can contribute to the management of mobile robots and driverless vehicles [1].
As for the technology background, the Lidar calculates the distance between the sensor and the target objects from the echo time of the emitted and the detected laser beam where the beam spreads with the speed of light. The result of the measurement is a highly accurate 3D point cloud where the coordinates of the points are given in a local or global coordinate system depending on the type of the Lidar system and the application area.

In the course of the thesis I mostly focus on mobile laser scanning (MLS) technologies, so the proposed and implemented algorithms are designed for MLS data type. Only for demonstration purpose I give some insight into terrestrial (TLS) and air borne point cloud processing.

2. Laser Scanning

All of ALS, TLS and MLS are laser scanning technologies thus these sensors have the same theoretical backgrounds, but they differ in the types of the Lidar scanner and the supplementary sensors. ALS has several main applications such as large area mapping (cities and fields), vegetation and coastline detection and 3D city modelling. In this case the Lidar's scanning range is very large up to several km, but the resolution of it is relatively low, so the maximum point density is 50 points/m^2. TLS is the opposite of ALS in the meaning of point density and the size of the scanning area. It can gather 500 000 points/m^2, however due to the difficult preprocessing work (managing huge amount of data, noise filtering and registration), it can be used only for scanning objects, buildings or small areas. Best quality MLS sensors produce up to several thousands points/m^2 by driving at normal urban speed limit (30-60 km/h) and their scanning range is also between the range of ALS and TLS.

2.1 Mobile Laser Scanning

Beside one or several Lidars, a navigation system and a calibrated camera system are also included in MLS platforms. The navigation system consists of a global navigation satellite system (GNSS) for positioning and an internal measurement unit (IMU) to determine the orientation. The best MLS systems produce high quality geo-referenced point clouds in a global coordinate system and 2-3 cm accuracy is achievable in the point cloud. In urban environment mapping MLS collect several hundred million or billion points, so data storing, managing and processing tasks are very challenging.

2.1.1 RIEGL VMX-450 Lidar sensor

VMX-450 is a high speed mobile laser scanning system for data acquisition [3]. It is mounted onto the top of a moving car and it offers extremely dense, accurate and feature-rich data even at high driving speed. The system integrates two RIEGL VQ-450 laser scanners, IMU, GNSS and a well-design camera platform ensures mounting up to six digital cameras. Each of the two VQ-450 laser

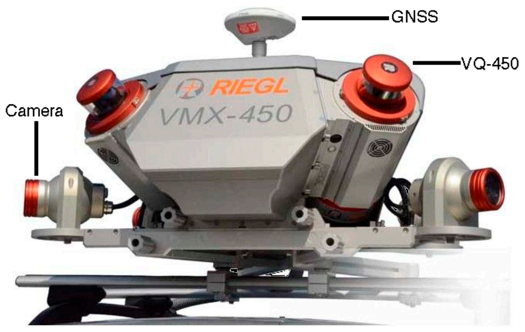

Figure 1 VMX-450 MLS system

scanners provides low-noise, gapless 360° frames at a measurement rate of 550 thousand points/sec. The camera system completes the scan data with precisely time-stamped images. Typical applications include mapping of transportation infrastructure, city modeling, network planning and surveying of mining.

2.1.2 Velodyne HDL-64E Lidar sensor

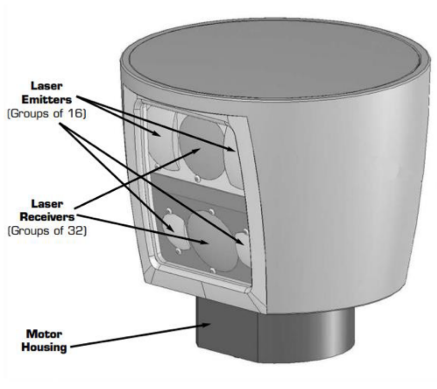

Figure 2 Velodyne HDL-64E Lidar sensor

Figure 2 illustrates the Velodyne HDL-64E sensor. It has been designed to navigate autonomous cars and ships. It has a 360-degree horizontal and a 26.8-degree vertical angle of view with 5-15 Hz operating speed. It has 64 laser beams and it can produce 1.3 million points/second. It provides a 2.5D point cloud within a 120m range. Beside the 3D relative positions, the points also contain an intensity value what gives the strength of the returned laser beam. This intensity value highly depends on the the angle of incidence and the surface properties of the objects. Using the mentioned Velodyne sensor, it is possible to monitor large complex dynamic scenes. Experiences and trends show that the 64 beam version will be used only for validation or in very large investments. But there are 32 and 16 beam versions and their price can decrease in series production to affordable category.

3. Sensor and data fusion

Nowadays data fusion is central task in visual surveillance and Lidar mapping applications. If we use more sensor modalities more accurate models can be achieved from the scene, thus it is important to merge and reconcile data from different sensors. In addition, with respect of data fusion we can solve complex problems.
However, sensor calibration and data fusion are major challenges both of the hardware and software sides. Very accurate sensors are required and one of the main tasks is to synchronize the sensor's timestamps to each other. Furthermore, data fusion is usually an offline process, because of the huge amount of data, and in most cases preprocessing is also needed for appropriate registration.

In the field of point cloud processing one of the most valuable additional information sources is the RGB color channel. Generally, the color information is provided by calibrated high resolution cameras. The images are used in a number of operations:

- they provide additional information about the visibility and types of objects in the scene
- they can improve detection accuracy through incorporating color information besides geometry into the modeling process
- they can be used for texturing the reconstructed geometric models

Three different cases can be distinguished by joint utilization of 3D point clouds and images:

- **independent photography:** there is no direct connection between the photo and the point cloud acquisition process.
- **applique photography:** the camera is connected to a laser scanner by some sort of adapter. Calibration is required to map the camera image onto the point clouds and vice versa.
- **built-in camera:** photography and scanning are executed in parallel by the same device.

The following sections give an overview of the image and point cloud registration. Two main cases can be distinguished. First, the point cloud points are projected onto the corresponding pixels of the image. Second, image pixels are back projected to the 3D points. Both cases were implemented and tested during my thesis work.

3.1 Point cloud projection onto image pixels

In this section, I introduce the mathematical background of point cloud projection onto an image which task can be solved in real-time. Accurate camera and Lidar scanner calibration and synchronization are essential parts of successful operation. By using the integrated RIEGL mapping system the calibration and synchronization process are solved by on hardware level but some post-processing of the measurement is needed by the operators.

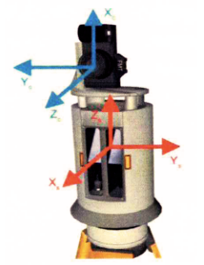

Figure 3 Applique laser and camera system

In the applique system illustrated in Figure 3, the laser scanner and the camera have two separate coordinate systems. These local coordinate systems can be transformed into a common, unified coordinate system using the corresponding transformation matrices.

The laser scanner's 3D point coordinates are given in a global coordinate system. To transform 3D points onto the 2D image pixels (called projection), we need to know the internal state of the camera (camera matrix), as well as the position and orientation of the camera (rotation matrix).

The transformation matrix (P) between the point cloud and the camera is the following:

$$\boldsymbol{P} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

The matrix P transforms coordinates measured in the scanner's 3D coordinate system to the image plane's coordinate system. The matrix contains three rotation angles and three offsets. The offset specifies the scanner's location in the coordinate system of the project and the rotation angles describe the scanner orientation with respect to the projection.

The camera matrix ($\boldsymbol{C}$) describes the parameters of the camera. In the camera matrix C, scalars f_x and f_y denote the focal length of the camera, while c_x and c_y are the coordinates of the principal point:

$$\boldsymbol{C} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

During the data transformation recorded by the camera the scales should also be considered. So the described camera model is the following:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = C * P * X = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In the model above, u and v represent the image coordinates. Each r_{ij} is the rotational component of perspective projection matrix P. Scalars t_x, t_y, t_z are the offset vector components and $X = [x, y, z]$ is the observed spatial point in the scanner's Euclidean coordinate system [1, 4].

3.1.1 Implementation of point cloud projection

The first step is to construct the corresponding $[r_{ij} \mid t]$ and camera matrix. Next one should transform the 3D points of the cloud onto the 2D image plane using the constructed matrices.

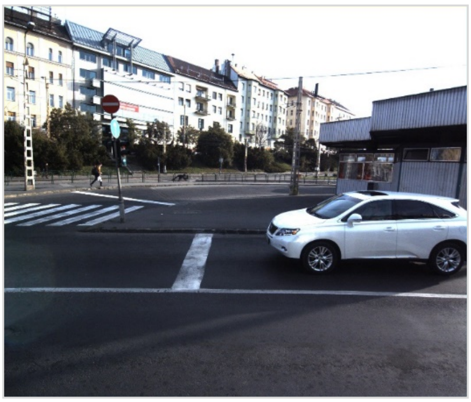
Figure 4 Image captured by optical camera. This is the image plane where the 3D points are projected

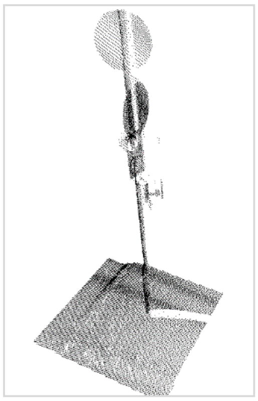
Figure 5 3D point cloud of a traffic sign

Constructing of the $[r_{ij} \mid t]$ rotation and offset matrix:

The 3X3 rotation matrix is composed of the three Euler angles, which are often referred by the yaw, pitch and roll terms. The yaw describes a rotation with α degree around axis z, the pitch is a β rotation around y axis and the last one rotation is defined by γ degree around axis x.

$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{pmatrix} cos\alpha cos\beta & cos\alpha sin\beta sin\gamma - sin\alpha cos\gamma & cos\alpha sin\beta cos\gamma + sin\alpha sin\gamma \\ sin\alpha cos\beta & sin\alpha sin\beta sin\gamma + cos\alpha cos\gamma & sin\alpha sin\beta cos\gamma - cos\alpha sin\gamma \\ -sin\beta & cos\beta sin\gamma & cos\beta cos\gamma \end{pmatrix}$$

It is important to pay attention to the order of the matrix multiplication, because this will affect the final transformation. This common rotational-translational matrix $[\mathbf{R} \mid \boldsymbol{t}]$ is usually called the matrix of the external parameters. The camera transform in the static point cloud can be describe using the [**R** | ***t***] matrix. That is, [**R** | ***t***] transforms the coordinates (X, Y, Z) into a fixed coordinate system with respect to the camera.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t, \quad x' = \frac{x}{z} \text{ és } y' = \frac{y}{z}$$

Finally, using the camera equation we can obtain the pixel coordinates:

$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y$$

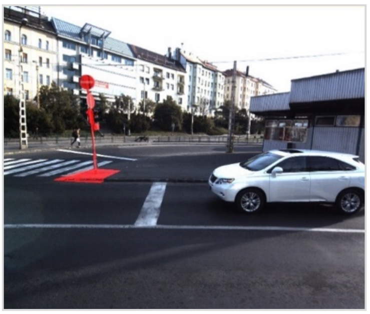
Figure 6 The image with the projected 3D

Figure 6 shows the result of the projection. It can be seen that there is a clear link between the points of the point cloud recorded by the Lidar scanner and the pixels of the image captured by optical cameras.

3.1.2 Point cloud coloring

Point clouds and camera images used in this tasks have been provided by Budapest Közút Zrt. The point clouds have been recorded by the RIEGL VMX-450 and RIEGL VZ-400 mobile and static scanning systems respectively, as already discussed in section 2.1. The 3D points are represented in the global EOV (projection system of Hungarian surveying maps) coordinate system format and the density of the point cloud varies between 0.5-3 cm [3].

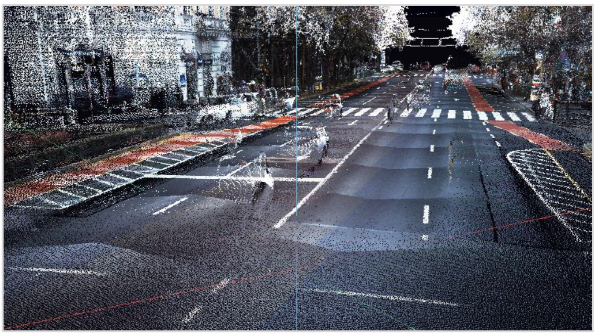
Figure 7 Scanned area (Oktogon, Budapest) by VMX-450. Figure shows the colored point cloud recorded by the Lidar scanner and colorized by optical cameras

In Figure 7, the colored point cloud demonstrates the high resolution of the sensor, and the nearly uniform distribution of the recorded data. Coloring process of the point cloud is based on six cameras viewing to different directions. In areas where the measuring car cannot pass, a static laser scanner (VZ-400) has been used.

Point cloud coloring is the same process as already described in Section 3.1. Once the point cloud is screened into image pixels, the pixel color information is assigned to the corresponding 3D points. Another interesting question is how to determine an object found at a given pixel position in the 3D space. For example, by traffic sign recognition, we may often localize a given object by its pattern in the 2D image, then we have to determine its accurate position and orientation in the 3D point cloud space. The next section presents a solution for this challenge.

3.2 Supporting 3D processing by using image information

This section presents the fundamentals of extracting 3D information from calibrated camera images and it proposes a frustum based back projection algorithm from image domain to the 3D space.

Stereo vision

Extraction of 3D information from digital images cab be achieved using stereo vision techniques. In most cases, two calibrated cameras are placed recording the same scene from slightly different view points. So the projection centers do not coincide with each other. The advantage of such a system is that the relative depth information can be obtained by comparing the images. 3D reconstruction can be performed by simple triangulation [5].

Although the MLS system discussed in this section is not a stereo camera system, but the cameras are calibrated and the proposed methods extract 3D information from the images using the stereo vision's doctrines. The main aim is to improve the effectiveness and the accuracy of the 3D processing by using 2D image information and to obtain more robust results. Next we describe the realization of the methods mentioned above.

Let us assume that a particular object or image region is detected in the image by a pattern recognition algorithm (e.g. template matching). To determine its position in the 3D cloud, the following steps are required:

- calculating the 2D bounding box
- back projection of the vertices of the bounding box to the 3D space
- starting lines from the 3D camera center through the 3D bounding box vertices
- determining the 3D points inside the frustum
- determining the common sections of several frustums

3.2.1 Back projection of image information in 3D space

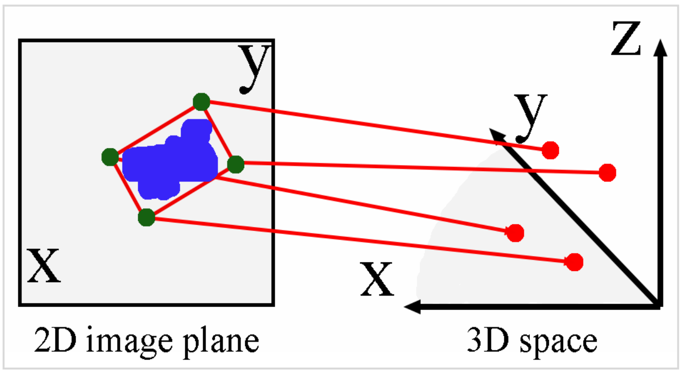

Figure 8 2D point projection back to 3D space

Several methods exist in the literature to calculate bounding boxes. Most of the algorithms are based on Principal Component Analysis (PCA) [6, 7] but there are other approaches as well [8]. This study does not deal with the bounding box detection problem the presented method starts with given corner points.

Back projection of a given 2D image point generates a line in the space:

$$\boldsymbol{X}(\alpha) = \alpha * P^{+}\boldsymbol{u} + \alpha * \boldsymbol{C}$$

- P^{+} is the pseudo-inverse of the P projection matrix presented in section 3.1. P^{+} can be calculated as follows: $P^{+} = P^{T}(PP^{T})^{-1}$ and $(PP^{+}) = I$, where I is the identity matrix, (note that the OpenCV library provides an efficient SVD decomposition method for calculating the P^{+} matrix).
- $\boldsymbol{C}$ is the camera origin [x position, y position, z position, 1] in the homogeneous coordinate form and $\boldsymbol{u}$ is the 2D pixel coordinate also in homogeneous form.
- α is a freely selectable parameter which determines the 3D point distance from the camera center, but it is just an optional point in the line.

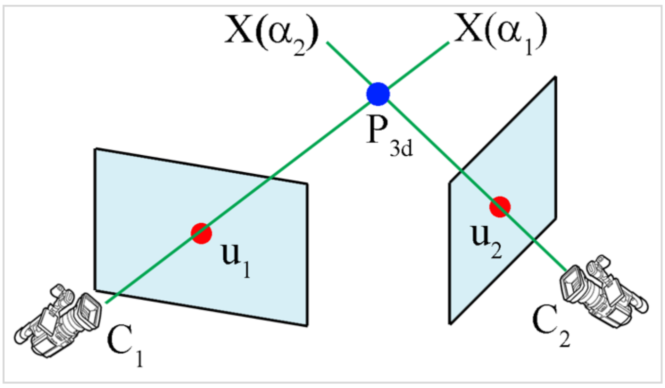

Figure 9 Triangulation

To obtain the actual 3D position of the 2D point we need at least two cameras and the point must be visible at least in two images as Figure 9 shows. In the intersection of the two lines we get the corresponding 3D point. This technique is based on stereo vision.

3.2.2 View of frustum

The four corner points of the bounding box define a frustum in the 3D space and the frustum includes those 3D points which belong to the part of the image within the bounding box. The view frustum is the key to localize an object detected on the image in the point cloud. For example, the algorithm may find a flat surface in the point cloud, which consists of several different types of regions. In this case, due to the cloud's it is limitations almost impossible or very hard to separate these areas. However, in the image domain the algorithm can use color and texture information supporting the separation and in the computer vision literature there exit a lot of pattern recognition techniques such as template matching, which can efficiently cope with the task [9, 10]. Once the regions are separated in the image using the frustum concept we get the distinct areas of the plain in the 3D cloud.

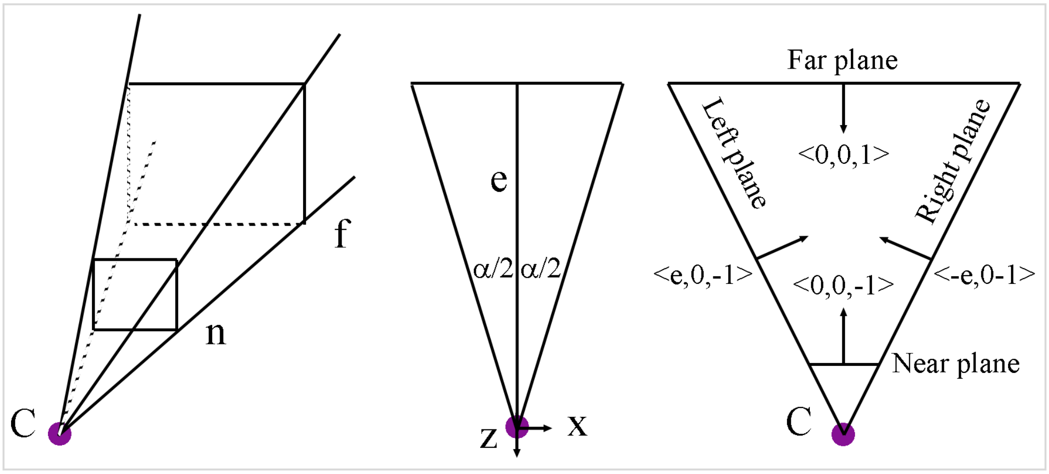

Figure 10 The view frustum encloses the space bounded by the near plane, the far plane and four side planes.

Figure 10 shows the view frustum, i.e. the volume of space which contains everything that is visible in the 3D scene. The frustum is bounded by six planes, where four of them correspond to the edge of the screen and the remaining two planes are the near (n) and far (f) planes, defined by the minimum and maximum distances visible from camera. The projection plane is perpendicular to the camera's viewing direction and lies at the distance

Plane	$< \boldsymbol{N}, D >$
Near (n)	$< 0,0,-1,-n >$
Far (f)	$< 0,0,1,f >$
Left	$< \frac{e}{\sqrt{e^2+1}}, 0, -\frac{1}{\sqrt{e^2+1}}, 0 >$
Right	$< -\frac{e}{\sqrt{e^2+1}}, 0, -\frac{1}{\sqrt{e^2+1}}, 0 >$
Bottom	$< 0, \frac{e}{\sqrt{e^2+a^2}}, -\frac{a}{\sqrt{e^2+a^2}}, 0 >$
Top	$< 0, -\frac{e}{\sqrt{e^2+a^2}}, -\frac{a}{\sqrt{e^2+a^2}}, 0 >$

Table 1 Plane vectors

from the camera. The angle α is called the horizontal field of view angle. The four side planes of the view frustum carve a rectangle out of the projection plane. The camera space normal directions for the six view frustum planes are also shown in Figure 10. The 4D plane vectors corresponding to the six sides of the view frustum are summarized in Table 1 where normal directions for the four side planes have been normalized to unit length [11].

3.2.3 Extracting corresponding point indices using the frustum concept

The used algorithms are based on the presented fundamentals from section 3. The first step in out proposed method extracts a geometric shape from the 3D point cloud, which is similar to the shape of the interested regions. In the next step the extracted shape is projected onto the image plane using the manner mentioned in section 3.1. The different types of regions are separated in the image plane using pattern recognition techniques. Finally, the algorithm uses the frustum concept and the extracted bounding boxes on the image to determine the individual parts of the 3D volume.

3.2.4 Calculate the position of a 3D point elative to the frustum

At this point the algorithm using the frustums is able to determine the individual sections of the space and now it just has to decide whether a given point of the cloud is inside one of the frustum.

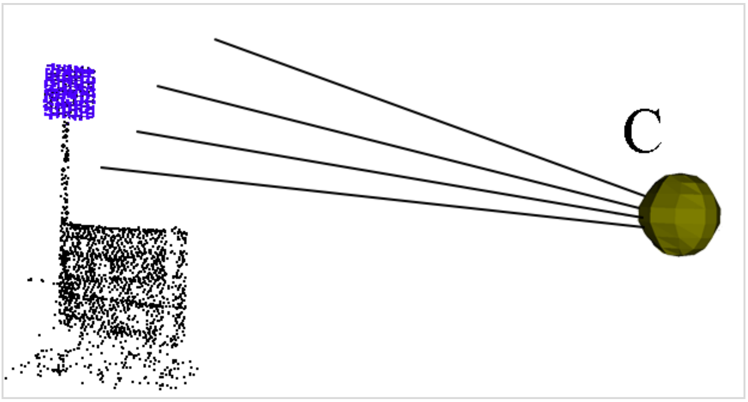

Figure 11 Result of the algorithm, blue dots represents the extracted sign head

Figure 11 demonstrates the frustum based point cloud segmentation where the blue points are inside the pyramid. To determine if a point is within the pyramid, the algorithm substitutes the point into the side plane vectors.

$$Ax + By + Cz + D = 0$$

So the form above is a commonly written plane equation where A, B and C are the x, y and z components of the normal vector $\boldsymbol{N}$ and $D = -\boldsymbol{N} * \boldsymbol{P}$. The algorithm normalizes the normal vector to unit length, because in that case the equation

$$d = \boldsymbol{N} * \boldsymbol{P} + D$$

gives the signed distance from the plane to an arbitrary $\boldsymbol{P}$ point. If $d = 0$, then the point $\boldsymbol{P}$ lies in the plain. If $d > 0$, we say that the point $\boldsymbol{P}$ lies on the positive side of the plane since P would be on the side in which the normal vector points. Otherwise, we say that the point lies on the negative side of the plane.

It follows that a 3D point is inside the pyramid if the results of the four substitution give the same solution with respect to the sign. So if all of the result are –d or all of them are +d regardless of the size of d, then the point is inside the pyramid. With this approach we obtain the specific points inside the pyramid, but the extracted points contain specific parts in front and behind the plate too. The solution came from the field of stereo vision. The method uses another image with a different camera view where a new pyramid is formed. Intersection of the two pyramid exactly specifies the points of the desired object, which are the points of the plate of a traffic sign in this example. Instead of calculating the intersection of the pyramid, the algorithm should only test those points which have been extracted from the previous pyramid. This simplification makes the algorithm more effective.

4. Point cloud filtering

Point clouds produced by laser scanners provide highly accurate 3D information. However, in many cases, the large number of reflections and the unexpected or ghost objects generate a lot of noise and redundant data. So generally the first step in point cloud processing is some kind of data filtering and cleaning. The noise filtering can greatly affect the operations of the further algorithms and also plays an important role in the quality assurance of the final results. In addition, it also greatly reduces the running time of various processing steps. So preliminary data filtering is essential for obtaining accurate results and performance.

Various kind of artifacts can be generated [38] during the scanning process:

1. Sensors have **physical limitations** which facts can lead to noise in the recorded data.
2. "Ghost" objects can be generated by **motion artifact**. MLS and TLS systems transform the points into global coordinate system, so moving objects (cars, pedestrians) appear as stretched, noisy phantom blobs.
3. Outliers are also generated by **multiple reflections.**
4. Due to **occlusion** and critical **reflectance properties**, holes and missing parts can be present in the surfaces of the models.

Several basic filtering methods exist in the literature. These ones use usually general point cloud properties such as distance between points, density of the local parts or point number. Outlier removals eliminate lonely points, which are mostly generated due to the inaccuracy of the sensor. If inside a search radius the minimum number of point neighbors are smaller than a predefined threshold, the algorithm removes the point. Statistical removals examine the normal distribution of a point neighborhood and if the point is far from the global mean, they remove the point.
Pass through filters simply remove the points which are out of a given spatial range.

Figure 12 demonstrates the motion artifact and the multiple reflection errors. In both cases, the upper part of the figure illustrates the original point clouds before filtering and the lower part of the image represents the cleaned point clouds. As one can see, the clouds contained huge amount of noise and phantom objects, which were efficiently removed.

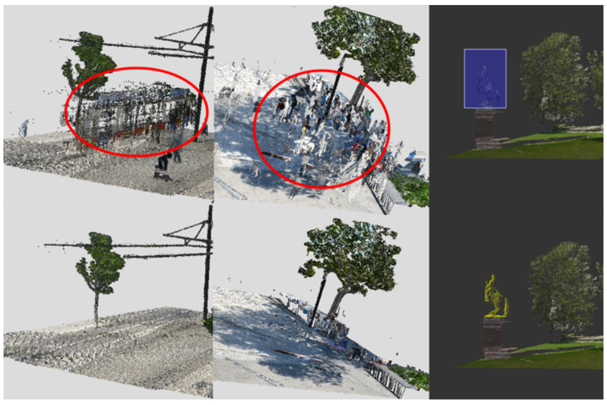

Figure 12 TLS point clouds before and after noise filtering and the annotation process on the right side

Manual point cloud cleaning is very exhausting and time consuming process however it is indeed used in the entertainment field such as movie productions or game industry, where visual impression is the most important factor. We developed an annotation tool for cleaning point clouds very precisely in a manual way. The right side of Figure 12 illustrates the using of the annotation tool where the first step is to mark off the removable points. Using the mouse and keyboard we can define clip planes to determine the undesired points. The method saves the index positions of the marked points in the point cloud vector and during a post-processing step, the points of the annotated objects were removed from the point cloud array based on their indices.

In applications where visual experience is not so important, automatic noise filtering is an applicable key step before further operations. In the following I propose an automatic, effective method for filtering point clouds of traffic signs. The main goal is to remove as much noise as possible, while we should preserve the points of the traffic sign. As input, we have obtained cylinder shaped point cloud segments from Budapest Közút, where each segment contained a complete traffic pole with one or multiple traffic signs and possible additional street objects such as tree parts. With other words, the extracted point clouds include not only the signs, but also all objects in the local environment of the traffic pole within a radius r.

General traffic sign characteristics in the point cloud

- The head of a sign is essentially a 2D shape in 3D space, because one direction of its extension is always negligible relatively to the other two directions.
- Points reflected from the surface of the sign show regularity. Due to the characteristic of the sensor the points are arranged in n rows and m columns and the distance between the points is consistent.
- In most cases the head of the sign has high reflectivity, in other words, it is retro-reflective. Consequently, the sensor assigns high intensity values to the corresponding points which can be efficiently separated from their surroundings in this way.

My proposed point cloud filtering algorithm consists of two main process:

1. The input shape file is an infinity cylinder respect to the direction of up (z) axis. To speed up the filtering process, in the first step the algorithm removes the ground and the points with extremely high elevation values using simple thresholds.
2. Based on the remaining point cloud, the algorithm builds a Kd-tree, then the points are filtered using point density and point normal features, with also considering the local intensity properties.

Conventional 3D processing usually uses sphere of radius r for neighborhood search. To ensure efficiency and accuracy I considered the properties of the sign and I proposed a cylinder-shaped neighborhood for the analysis.

Cylinder-based neighborhood search

The sign's head has a flat shape, and we have experienced hat during the neighborhood search, it is better to use a cylinder kernel instead of a sphere.

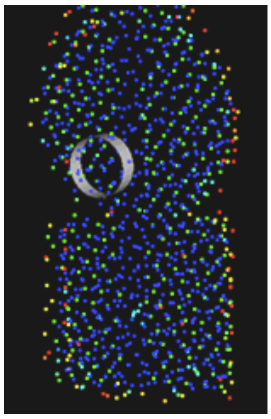
Figure 13 Cylinder-based search

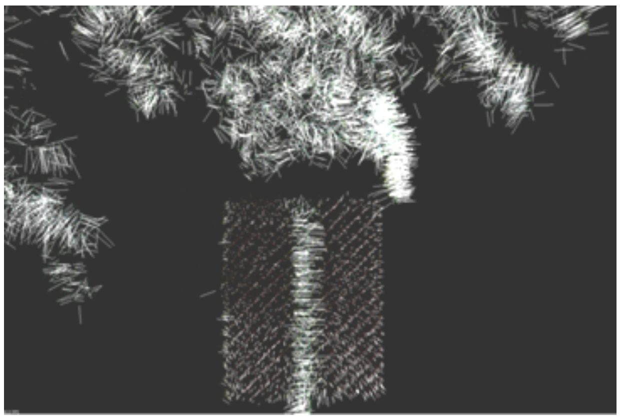
Figure 14 Normal-based filtering

Figure 13 illustrates the cylinder-based neighborhood search. Using this approach, density- and distribution-based filtering operations proved to be much more efficiently, because this sort of neighborhood takes into account the shape characteristic of the sign. In this way, I could apply powerful filtering criteria, e.g. when it was necessary I increased the number of required points within a given radius. Using statistical analysis of point neighborhoods, most parts of the vegetation can be filtered effectively, exploiting the lack of structural regularity. For comparison, we experience that using sphere-based search, much more part of the vegetation remained.

Normal-based filtering

Figure 14 illustrates the estimated surface normal vectors assigned to the points. The algorithm calculates the normal vectors based on an n neighborhood using the cylinder approach. It can be seen that the normal vectors of the sign surface point towards similar directions with only minor differences. On the other hand, the direction of the normal vectors belonging to the vegetation or the traffic pole have large variety. This property also takes the advantage of the previously mentioned ground filtering step. Since the ground has been removed during a pre-filtering, the estimation and analysis of the normals is significantly faster, and the extracted planar regions correspond indeed to the traffic sign heads. Besides applying the density and normal vector based filtering criteria, we include an additional intensity constraint to the points with high intensity are not removed in any cases, thus the algorithm keeps signs with noisy shape but high reflectivity.

The following figures demonstrate the output of the filtering method. It can be observed how noisy the raw input cloud is and beside the signs many other objects can be found in the scene.

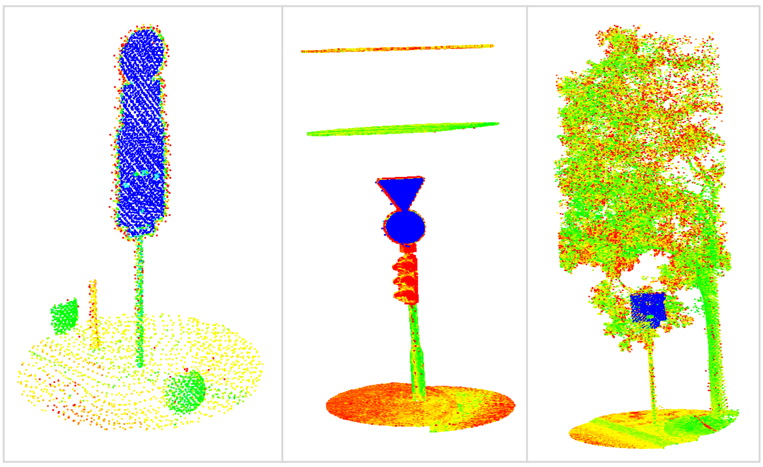
Figure 15 Unfiltered traffic sign point clouds

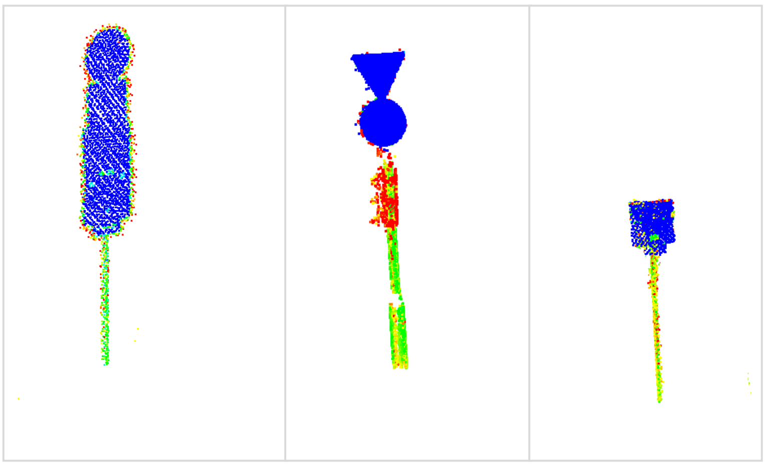

Figure 16 Filtered traffic sign point clouds

Figure 16 shows the output of the filtering method. It can be seen that the method also retains the columns beside the sign head. It will be important to further processing.

While the above introduced point cloud filtering (removing outliers, noise and undesired regions) is a preprocessing step, in order to clean the data for further processing such as scene segmentation and object detection, in the next section I propose an efficient data structure for efficient point cloud management and object detection.

5. Point cloud segmentation and object detection

Extracting various urban regions and objects from 3D point clouds is a critical task in many applications since the accuracy of perception and object recognition highly depends on the quality of objet detection and separation.

On one hand, processing huge amount of data, point density variation and noise are major challenges in point cloud based object detection. On the other hand, at object level several further problems occur such as overlapping between objects and the presence of incomplete visible object shapes caused by occlusion. In real world scenarios inter- and intra-class shape variations and orientation differences highly influence the result of object extraction.

In the following I give insight of the object detection literature, and at the end of this section I propose a data structure and various algorithms which I implemented for efficient objet detection.

In the literature of urban point cloud scene analysis, the main focus is to extract traffic signs, pole-like objects, roads, building facades, cars and pedestrians. The majority of existing object detection methods can be divided into two main groups. Methods from the first category are based on prior knowledge, while the second group of techniques uses global shape descriptors.

We can find several paper about object detection based on prior knowledge. In [19] road markings are extracted based on intensity and height information, and [41] detects road markings in point clouds and geo-referenced intensity images. Road edge detection using splines and peak extraction is introduced in [42]. Detecting facades and buildings are also very important and relevant in this field. [43] uses a marked point process algorithm to detect building from airborne point clouds. Special features are generated from point clouds to detect buildings in [44]. In [18] pole-like objects are detected based on a scan line structure, while [15] detects poles by considering their geometric properties. Detecting repeated structures [45] can be achieved using Fourier analysis, for example detecting a row of similar windows in a given building facade is an interesting related problem.

Object detection based on global and partial shape descriptors is the other highly referenced technique. While global techniques need very accurate pre-segmentation as precondition, partial shape based detection tolerates well the segmentation noise. [46] uses the super voxel idea to segment the point cloud then the segments are merged into objects. The surface growing method used in [47] segments the cloud into various clusters which are used as primitive shapes volumes. [25] detects small objects in urban environment. [48] is a typical part-based method for object detection using the Hough forest framework.

Because of occlusion, overlapping between objects and the above mentioned artifacts, accurate point cloud segmentation and object detection are still challenging problems in complex environments such as real urban scenarios.

5.1 Point cloud managing technique

In point cloud processing, generally the first step before object detection is that we assign predefined labels (ground, object, vegetation) to each points. It is called point cloud segmentation. We have to build an efficient data structure to reach the point neighborhood in order to make the segmentation and further processing steps achievable. After the segmentation step, it is necessary to determine criterions to group the points and their neighbors into objects or groups of objects. There are several well established approaches for building the point neighborhood model such as different space partitioning trees, like octal tree or kD-tree [6], furthermore regular voxel models and 2D grid based methods [49] are also often used.

5.2 Partitioning tree based methods

Let assume a division which always splits the space into two sides considering the point density, so that the size of the two halves are proportional with the local density. This method leads to a binary tree structure what is called the binary space partitioning tree (BSP-tree). If the plane which divides the space is always perpendicular one of the axes of the coordinate system, it is called a kD-tree structure.

If we split each node into exactly eight children without considering the point density, then we get an octal tree (octree). By comparing kD-trees to octrees, we can emphasize that a kD-tree adopts very precisely to the characteristic of the data. Using different type of clustering methods in the tree structure we can perform very robust and accurate object detection, but kD-tree based space partitioning also has some drawbacks. While reading the point neighborhood is very fast in kD-tree, but constructing the tree is often very slow. Moreover, if we work with streaming data then we have to rebuild the tree at each new frame which cannot be a real time processing. Octrees are more robust for streaming data so they are more commonly used in dynamic environments especially in game development.

After the space partitioning, one of the most commonly used clustering technique is the floodfill algorithm. Parameterization of the floodfill method can be determined efficiently for a given object type such as car or pedestrian, but in most cases there is no option to find a parameterization which performs well for the whole object set. Thus, octree and kD-tree based algorithms often over- or under-partition the 3D space.

5.3 Regular 2D grid and 3D voxel model

2D grid based methods define a 2D lattice in the first step what is fitted onto a horizontal plane, typically to the estimated ground plane [50]. In the case of regular grids, the cells have the same size. After the grid is created, the mapping algorithm assigns each points of the cloud to the corresponding grid cell, so that the assignment is actually a projection to the ground plane. After all, the grid cells contain some local parts of the cloud. The meaning of grid based segmentation that we assign a class label to each grid cell (and not to each point separately) based on the extracted statistical properties (density, height, scattering) of the local point cloud parts stored in the given cell. The size of the cells has a great effect to the computation time and the segmentation result. Using smaller cell size, in most cases the cells do not contain enough points to extract relevant information from them. Furthermore, decreasing the cell size results in longer processing time, because there are more cells in the same region. In case of larger cell size, the computation time is fast, but often too many points belong to one cell. Since the base unit of the method is the cell, if parts of different objects correspond the same cell, the algorithm will merge them into one object. Finding the optimal discretization size is very challenging, because of the high variance in the point density and the great variety between the object shapes in realistic point cloud scenes.
The basic principle of 3D voxel models is very similar to the mentioned 2D grid method. It is only expanded among the third dimension and the algorithm create a 3 dimensional multi-array instead of a 2 dimensional grid.

We have proposed an efficient two level grid structure for object segmentation in [8], however, it has a notable artifact. Let assume that an object lying on the ground and for example a crown of a tree or a wire is hanging over it. In this case some parts of them belong to the same cell and because there is no vertical partition the further algorithms merge them into one object. In this thesis, I propose an efficient data structure which can solve this problem.

5.4 Sparse Voxel World

In my thesis work, based on the previously mentioned techniques, I combined the best practices and I designed an efficient data structure for scene segmentation and object detection. In the following I present this data structure called Sparse Voxel World (SVW).

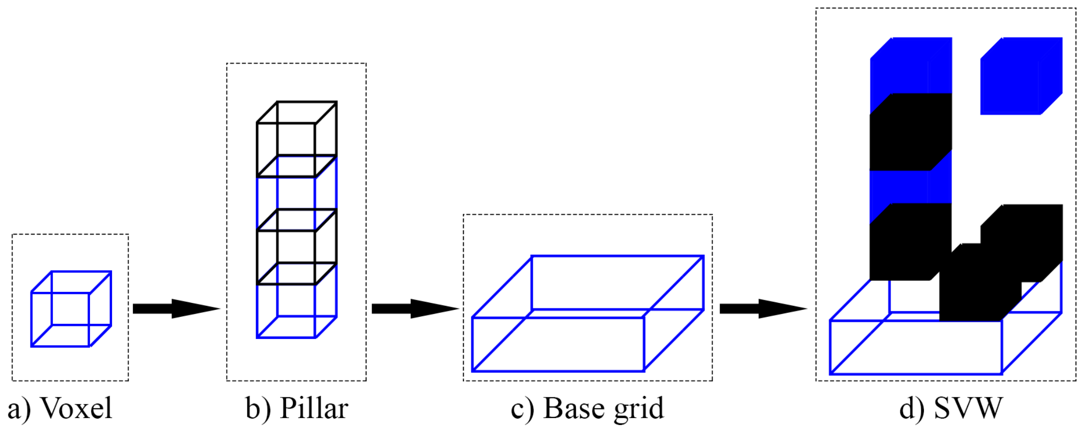

Figure 17 Sparse Voxel World

Figure 17 illustrates the implemented SVW. The base element of the SVW is the voxel, where voxel is a cube volume of the space. First of all, a voxel contains a local part of the point cloud, furthermore inside the voxels several properties and features are calculated. Figure 17/b illustrates a higher level part of the SVW, called the pillar. Basically, the pillar is a group of voxels which belong to the same base grid cell, where the base grid is formed from the lover level voxels. Finally, Figure 17/d represents the complete SVW structure. We can see that pillars are not fully filled, so voxels are only created if there is data on the given space region. Furthermore, it is important to note, that inside the pillars we may even find empty voxels space, let us consider as a typical example when a tree hangs over a car. Figure 18 shows a realization of the SVW.

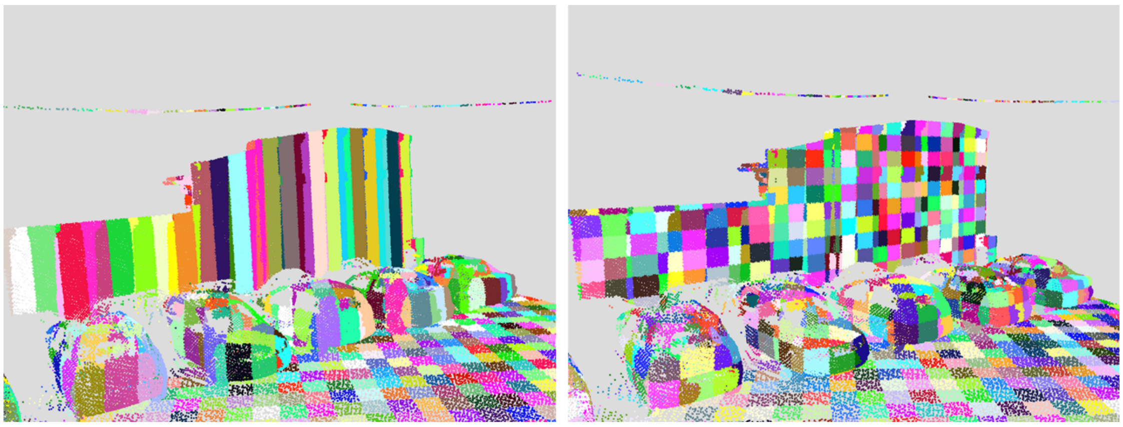
Figure 18 SVW realization. Left side illustrates the pillar structure and the right side shows the sparse voxel structure

In the above figure we can see the pillar and the voxel structure in a real urban scenario. This sample is created with a voxel size of 0.5 meters. If we fill the whole space with voxels then we get a structure with a total number of 15345 voxels, but using SVW, our structure contains only 1492 voxels. If I decreased the voxel size to 20 centimeters, then this rate was 230076 – 9803. (More than 20 times difference!) Using SVW we can reach much better performance in processing time and memory usage.

5.4.1 Create the SVW structure

Before the algorithm creates the SVW, it removes extremely high and low altitude points from the point cloud. In the first step the point cloud is sorted among the height dimension. The algorithm searches the ground region what contains lots of point and there are small height differences between the points. Here we filtered out those group of points that are close to each other but the group contains only few hundred points. Practically, these points are under the ground level and decrease the performance of the further processing. Though, the filter method loops through the cloud starting from the lowest point and clusters the regions until it finds the first large connected region (ground). If the ground region is founded, the points below the given height level are removed from the point cloud.

After filtering, the dimensions (X, Y, Z-height) of the cloud are recalculated:

$$dimension_x = \max_x - \min_x$$
$$dimension_y = \max_y - \min_y$$
$$dimension_z = \max_z - \min_z$$

Furthermore, the method calculates, how many voxels can be stored among each dimension with respect to the voxel size:

$$voxelnum_x = \frac{dimension_x}{voxel\ size}, voxelnum_y = \frac{dimension_y}{voxel\ size}, voxelnum_z = \frac{dimension_z}{voxel\ size}$$

Instead of creating a full 3D voxel grid containing $(voxelnum_x * voxelnum_y * voxelnum_z)$ voxels, the method creates a $(voxelnum_x * voxelnum_y)$ sized 2D voxel grid referred as base grid as shown in Figure 17/c. This base grid is responsible for storing the actual pillars. The projection method loops through the point cloud and assigns each point to a voxel as follows:

$$pos_x = \frac{point_x - \min_x}{voxelnum_x}, pos_y = \frac{point_y - \min_y}{voxelnum_y}, pos_z = \frac{point_z - \min_z}{voxelnum_z}$$

$pos_x, pos_y\ and\ pos_z$ determine the 3D position of the voxel. If the given voxel exists inside the corresponding pillar, the method simply adds the point to the voxel, otherwise it creates first the voxel and adds it to the right pillar, then it adds the actual point to the new voxel. In the implementation, pillars are represented as vectors and since the point cloud is sorted among the height dimension, in the consecutive pillar voxels are created in an increasing order. So the first element of the pillars is the lowermost voxel.

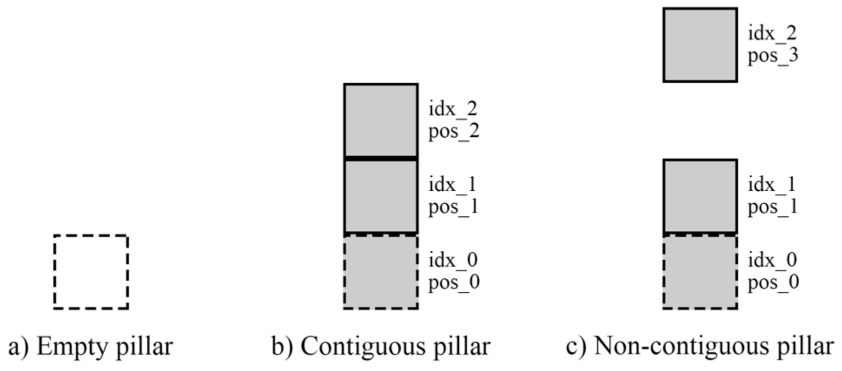

Figure 19 Pillar structure implementation

Figure 19 represents the potential cases of the pillar structure. The most basic case occurs when over the given base grid cell, there are no points, so the pillar is empty, as subfigure 19/a shows. In Figure 19/b the object is contiguous, because there is no empty voxel space in the pillar. Figure 19/c represents the mentioned case above when a tree hangs over an object. We can see that the indices are continuous in the vector but the real positions are also stored in the voxels, so that the length of the pillar vector is three, but we know from the position fields number of empty voxel spaces between the voxels.

When a new voxel is created, the algorithm links it with its neighboring voxels. A 3x3 kernel determines the potential pillars in the base grid, and if the difference between two voxel positions is smaller than two in absolute value, the algorithm connects the given voxels. This linked sparse voxel structure provides efficient neighborhood search using as much memory as needed.

5.5 Voxel computation unit

Voxel is the basic computation unit in SVW. Several local properties are calculated inside only one voxel but some features are computed using several neighboring voxels. This voxel based computation method can work very efficiently in a parallel way, furthermore the linked structure ensures efficient traversal between neighboring voxels.

5.5.1 Feature extraction

The next properties are computed continuously, during the projection process, whenever the points are assigned to the voxels:

- **centroid:** the mass of the local point cloud belonging to the given voxel
- **min and max values:** the extremes among each dimension
- **point number:** the number of points belonging to the local point cloud
- **voxel intensity:** the average intensity inside the voxel
- **voxel color:** the average color inside the voxel

In 3D the local neighborhood contains at most 26 voxels. To measure the statistical properties of it, a PCA is performed [51]. Let $\mu_1 > \mu_2 > \mu_3$ be the principal components and $\lambda_1 > \lambda_2 > \lambda_3$ the corresponding eigenvalues. These values are stored in the voxels and further features are computed from them. μ_1 is the normal of the fitted plane and μ_3 determines the main axis of this plane. When the points are projected to a principal component, the variance is equal to the corresponding eigenvalues, thus the total variance is equal the sum of the eigenvalues.
Examining the variance in each direction, the algorithm determines the linearity and flatness properties inside the neighborhood.

If λ_1 is large ($\lambda_1 > 0.7$) compared to the others, it means that the included points lie on a line, so the voxel neighborhood defines a linear structure such as poles or wires. Linearity is defined as follows:

$$L = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

If the total variance is defined by the two largest eigenvalues that means points lie on a plane:

$$F = 1 - \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}$$

If $F > 0.6$ the voxel is classified as flat voxel. Typically ground parts, wall segments and parts of larger objects are built from flat voxels.
Unstructured regions can also be detected using the third eigenvalues. If $\lambda_3 > 0.1$ that means point scattering is high in the region. Vegetation and noise can be detected efficiently using this property.

5.5.2 Segmentation

I implemented two different segmentation method. The first one is based on height evaluation comparing the voxel centroids, while the second one uses local variance properties based on PCA.

Figure 20 demonstrates the result of the height based segmentation.

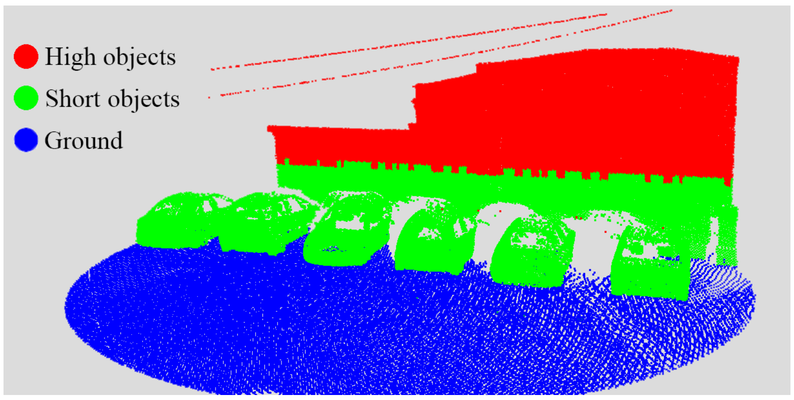

Figure 20 Height based segmentation

The algorithm loops through the voxels and it assigns a class label to each voxel, based on its centroid height (centroid_z) component:

- **ground:** centroid_z < 0.2m or the given voxel is part of the base grid
- **high object:** centroid_z > 2.5m
- **short object:** any other case

Height based segmentation is very fast, but it uses hard thresholds, furthermore it does not take into account the local properties. On the contrary, variance based segmentation is more time consuming, but it examines the local properties and ensures to obtain a more sophisticated segmentation. Figure 21 illustrates the variance based segmentation result for two scenarios. We can see that the pole-like objects show high variance in one direction, the ground regions are classified as flat voxels and unstructured, high scattered shapes such as vegetation are also observable. A notable property of the method is that edge regions are highlighted because they are neither linear not flat enough. In the following I propose methods that use one of the above segmentation models or use the combination of the two segmentation approaches.

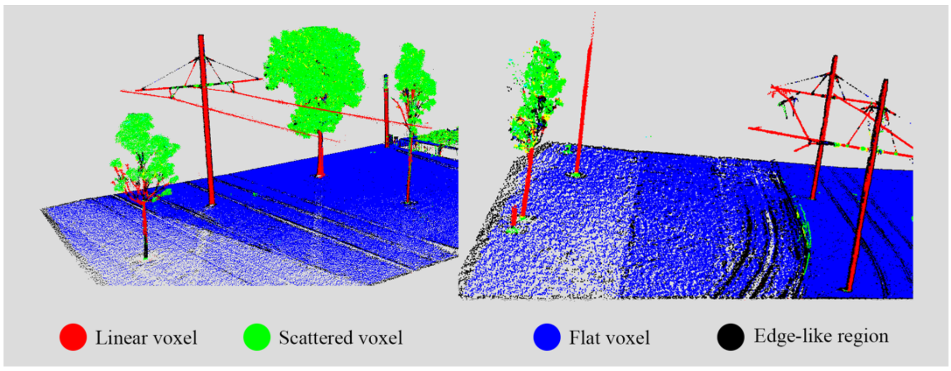

Figure 21 Variance based segmentation

5.6 Functionality of the base grid

In the first step the filtering method removes the noise under the ground level. The base grid, e.g. the lowermost voxels of the pillars contain the points of the ground regions. I implemented a very fast and efficient ground detection method that uses the base grid for extracting the ground. The method uses the result of the variance based segmentation mentioned in section 5.5.2.

The method dose not use global thresholds, because in most cases the ground has a slope yielding some elevation differences between the corresponding points. It examines the local neighborhood of the given voxel and if the 70% of its neighbors are flat voxels, and the height difference between the highest and lowest points inside the voxel is smaller than 0.1m then it is detected as ground.

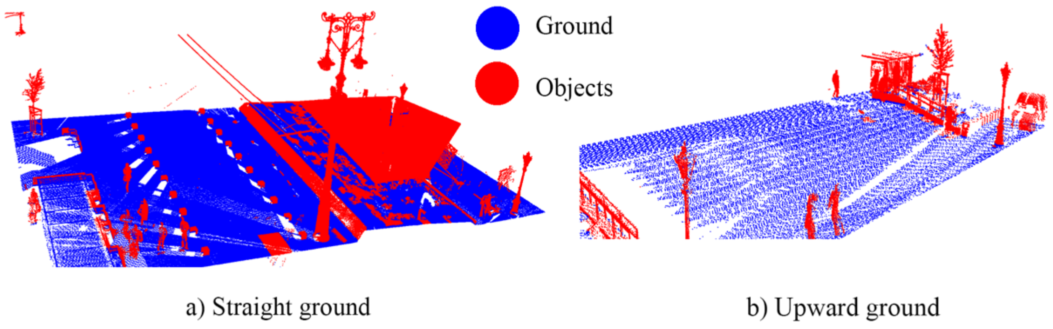

Figure 22 Result of ground detection

Figure 22 illustrates the result of the proposed ground detection algorithm. We can see on the right side that the ground has elevation differences (stairs), but the algorithm can manage the problem and detects the ground region efficiently. Needless to say, robust and accurate ground detection is very important for further processing, especially in object detection. The object extractor can easily merge the objects if their corresponding point clouds are connecting through erroneously misdetected ground points between the object.

5.7 Pillars as 2D grid

Essentially, pillars can work as a traditional 2D grid. The minimum height point of the lowermost voxel and the maximum one of the uppermost voxel within a pillar define the height of the local grid cell. Basically, it is a height map from top level view.

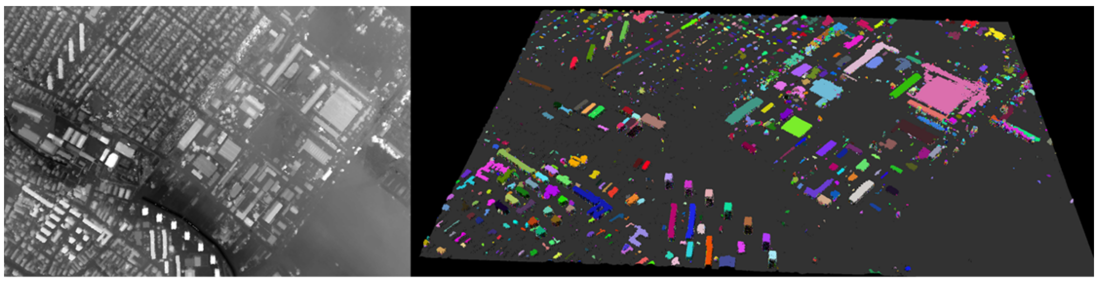

Figure 23 Object detection on areal imaginary data

Figure 23 demonstrates the object detection method based on [8] using the pillar structure. A connected component method merges grid cells to objects based on height evaluation. This aerial point cloud has very low point density (6-8 points/m^2), but the height based object detection works well on it. The algorithm processing time varies between 2-4 sec, while kD-tree based region growing methods run for 40-60 sec, furthermore finding optimal parameters in the conventional region growing approach is very challenging.

5.8 Object detection

In order to manage an object as a single unit we have to define some connections between voxels containing the parts of the object. The voxel merging criterion is a complex function what takes the computed properties and features of the voxels and returns a similarity

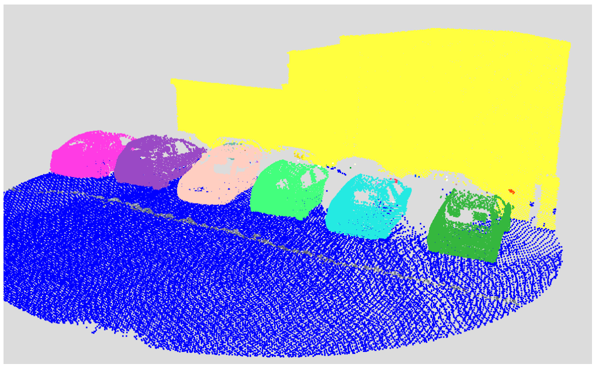
Figure 24 Object detection on MLS point cloud

measure what is the weighted linear combination of the parameters. If the similarity value is greater than a predefined threshold (0.75) then the voxels are merged into the same object candidate. The algorithm starts from a seed voxel and travers through the neighbor links while the similarity value is greater than the threshold. If all the neighbor links break, then the algorithm chooses randomly a new unvisited seed voxel and builds a new object. Basically, this is a 3D region growing algorithm in the voxel domain. Before the connected component search starts, the ground points are eliminated, thus the objects come apart. The merging criterion is very sensitive for the point cardinality and density. For MLS point clouds it is defined as follows:

- the Euclidean distance of the voxel centroids must be smaller, than the voxel size
- the Euclidean distance between the eigenvalues must be smaller, than 0.3
- if the voxels are in the same pillar it is a strongest weight

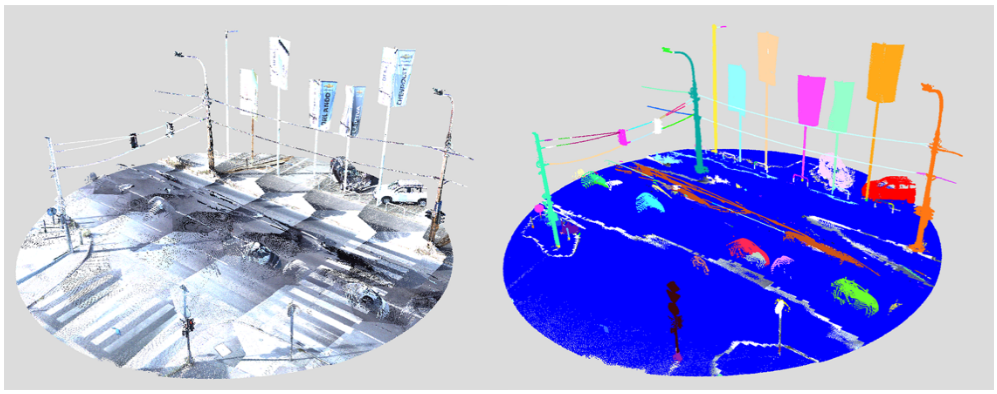
Figure 25 Original MLS point cloud and object detection result

In the Figure 25 above, we can see on the left side the original point cloud and the right side presents the result of the object detection method where each single object has a unique color.

5.9 Traffic sign’s head detection

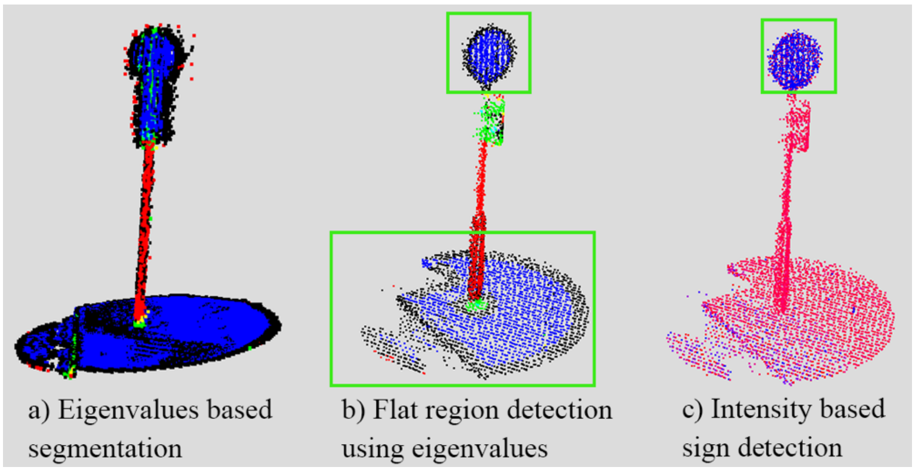
a) Eigenvalues based segmentation
b) Flat region detection using eigenvalues
c) Intensity based sign detection

Figure 26 Traffic sign detection: eigenvalues and intensity based coloring

We assume here that after filtering out the major parts of noise and vegetation, only planar and pole-like shapes remain back. Figure 16 is colorized by intensity and we can see that the head of the sign shows high intensity value because it is made from retro reflective material. Old signs are eroded by the weather, so they show lower intensity. In order to detect both new and old signs, the algorithm also uses the eigenvalues concept. Figure 26 illustrates that the planar regions like ground and sign plates are indeed segmented as flat regions while the pole contains linear voxels. The method gives larger weight to the intensity based detection, but the spatial covariance matrix also plays an important role.

6 Object classification

In this section I give an insight of object classification, machine and deep learning concepts and I propose algorithms for feature extraction to represent 3D objects in 2D form in order to make 2D convolution achievable on the extracted feature maps. Furthermore, I propose a deep neural network architecture which was implemented for urban objects classification such as vehicles, pedestrians and street furniture.

Classification is a general process to recognize and categorize objects to a class based on extracted features. Mathematically features are functions that map from the instance space to a given set of feature values. In other words, features are a compact representation of the object and through extracting the right features we can recognize the object itself. In 3D point cloud processing most studies in the literature use low level features such as object dimension, relative position to the ground and local point density. To extract these features, usually they use a bounding box fitted by applying a principal component analysis (PCA).

Several research studies dedicate many efforts to model or rule based object classification using MLS data. Building and facade detection and recognition have been studied in [12], [13], [14]. Observing vegetation is very important in environment protection, furthermore the crown of the trees uncover a lots of interesting regions and objects, so detecting them is crucial in MLS and ALS processing [15], [16]. There are many efforts in pole-like objects detection [17], [18], as well as on ground and road surface extraction [19], [20].
In the field of object based classification there are several widespread techniques, such as graph matching [21], 3D shape based or contextual processing [22] and using prior information and semantic rules [23], [24] are also very popular.
Objects from real world scenarios have various shape and size contrary to the synthetic objects, even entities from the same object type show high differences because of occlusion and noise. So in the recent years, trends show that the focus in object recognition shifts the direction of machine learning [25], [26], [27], [28], [29] mostly in Deep Learning.

6.1 Basics of machine learning

Machine learning (ML) is concerned with using the right features to build the right models that achieve the right tasks. Finding the best features is essential, because a model is only as good as its features. Depending on the desired output model, machine learning task can be clustering, regression and classification and from the side of the learning type we can distinguish supervised or unsupervised learning.

Unsupervised learning finds the structure in data without using predefined class labels.

- **Clustering** is a typical type of it where the goal is to group the input into groups based on similarity between the individuals.

Supervised learning works with pre-labeled training data to build the model.

- **Regression** produce continuous models rather than discrete ones.
- **Classification** algorithms based on training data learn the mapping from the feature space to the class labels in order to predict unseen examples.

The following parts of the chapter focus on the classification problem where objects are extracted from urban scenarios. In the literature machine learning experts suggest several geometric based features, such as the spin image [30], [25], point feature histograms [30], [26], hierarchical descriptors [30] and curvature and contour features [8].
The machine learning techniques mentioned above are based on handcrafted, predefined features, which are extracted in an automatic or semiautomatic way from the objects. In the recent years, end-to-end manner machine learning is in the focus where the learning method finds and extracts the best features from the input in an automatic way.

6.2 Deep Learning

Nowadays, deep learning is the leading machine learning techniques in computer vision. Deep Neural Networks (DNN) attempt to model the work of the human brain using many hidden layers embedded into an artificial neural network. Training huge amount of data in big networks needs lots of memory and high performance GPU units, so in the last few years NVIDIA and Intel made huge effort to develop such kind of chips [31] and now we have hardware to train millions of data record in huge nets.

Benefits of using DNN:

- It is more **robust** than traditional ML techniques because there is no need to design the features, they are automatically learned and optimized from input data.
- It is more **generalizable**, so we can use the same network for variant tasks and data types.
- We can increase the performance by adding more training data and the deep learning methods are massively parallelizable.

DNNs contain only fully connected (dense) layers, but in this thesis I focus on Convolutional Neural Networks (CNN) that contain some convolutional layers beside the dense ones. Basically, the CNN is a kind of neural networks that contain copies of the same neurons, and this trick allows that the network has lots of neurons meanwhile its actual parameter number remains the same. In other words, we have to learn a neuron only once and we can use it in many places.

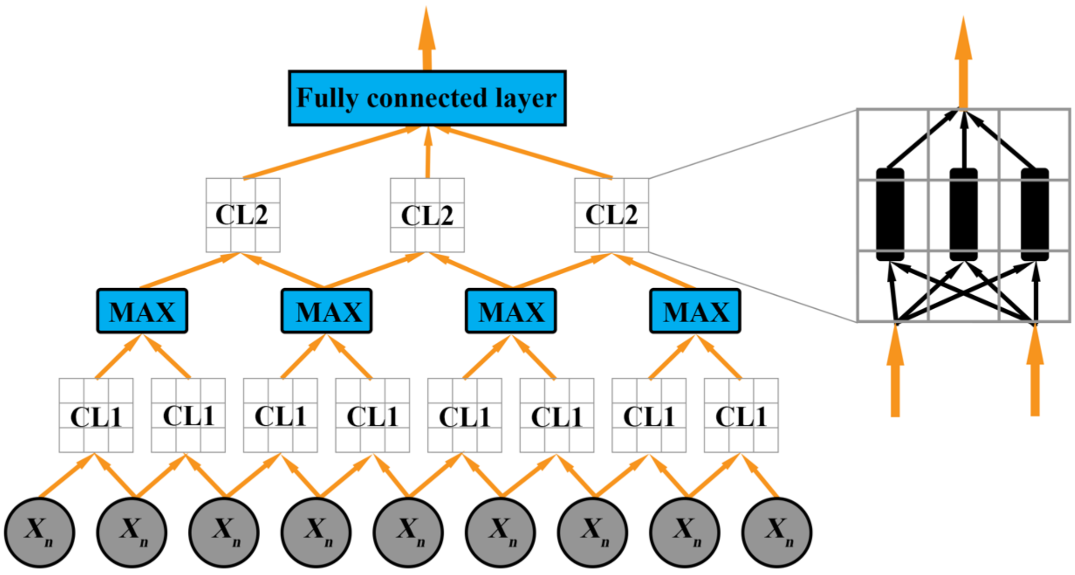

Figure 27 Typical structure of a CNN

Figure 27 demonstrates the typical structure of a convolutional neural network. If we eliminate the hidden layers and we connect all the inputs directly in a fully connected way with a dense layer we get a traditional neural network. Furthermore, if we use many dense layers, then we can create a deep neural network. But following the brain learning concept it is useful to know about the local properties of the data. In case of images we care about directions of edges, colors, textures and many other low-level features. For this reason, we use convolutional layers to examine small areas of the images. In the figure above every convolutional layer has only two inputs but most used networks the convolutional window could be much larger.

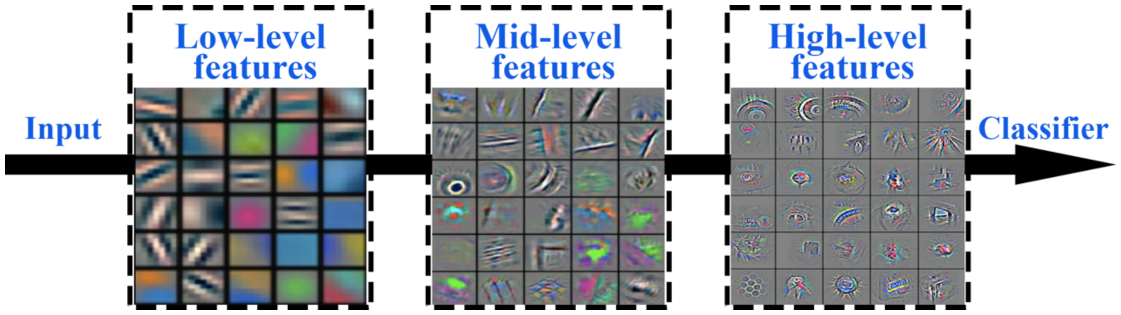

Figure 28 Connection between convolutional layers

The output of a convolutional layer can be fed into another convolutional layer. Figure 28 shows that on each layer the network can detect more abstract features. In the first level only edges and colors, then more complex patterns and shapes etc. Finally, the output of the last convolutional layer is linked into a dense layer. We can think about that the convolutional layers are responsible for the feature extraction and learning while in the dense layers the associations are formed. In image processing 2D convolution is used, but Figure 27 can only display a slice of it, because the neurons of the layers are aligned in a NxM structure. In Figure 27 we can also see a max-pooling layer. Max-pooling layer is a type of pooling layers where the maximum of the features are taken over small blocks of a previous layer. Usually, convolutional layers are connected through a pooling layer. Max pooling allows the next convolutional layers to work with larger segments so pooling is a kind of zooming out function.

Actually, a convolutional layer is a bunch of neurons, but one neuron might detect horizontal edges, another might detect the contrast between colors. The neurons are randomly initialized, so they are not handcrafted. These properties lead to an end-to-end learning scheme. The breakthrough of convolutional neural network was in 2012 when Alex Krizhevsky, Ilya Sutskever and Geoff Hinton published their result on ImageNet [32] competition, and since that time their network is known as ImageNet [33].

6.3 Creating 2D feature from 3D object

In the following I present a CNN I have developed to classify four different type of street objects: car, pedestrian, wall segments and street furniture (mostly traffic signs and trees) from point clouds collected in urban areas by a Velodyne HDL64 sensor. For building the network and applying deep learning algorithms I used the Theano framework [35]. In Theano there is no 3D convolution operator as in the other public library. So first, I had to produce a strong 2D representation from the 3D point cloud.

6.3.1 Extract 2D feature from 3D object

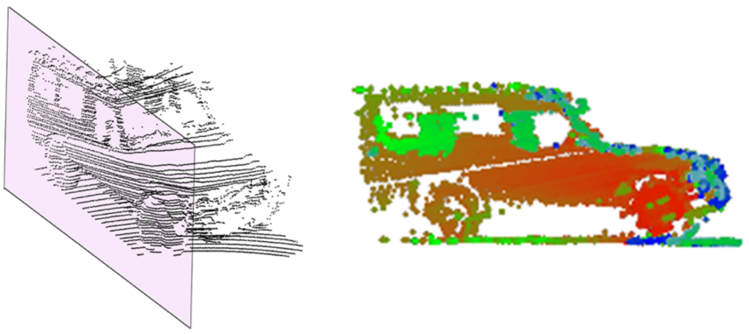
Figure 29 Projection plane and projected depth image

The input of the feature extraction method are 3D objects containing only 3D coordinates. In the first step the algorithm transforms the point clouds into metric form. It is necessary to use preprocessing steps, because there is a great variety in the units of the different datasets. After preprocessing, the algorithm applies a PCA transformation to calculate the main axes directions of the object. The eigenvectors corresponding to the first and the second maximal eigenvalues determine the main plane in the 3D space where the points are projected.

Figure 29 illustrates the projection plane and the result of the dimension reduction. After the method defines the projection plane, it assigns the up vector (0 1 0) to the corresponding main direction. It is important to retain the 2D orientation of an object. At a pole-like object (its height is greater than the other dimensions) such as traffic sign or a pedestrian the main direction is closer to the up vector, contrary to vehicles where the up vector is closer to the second main direction.

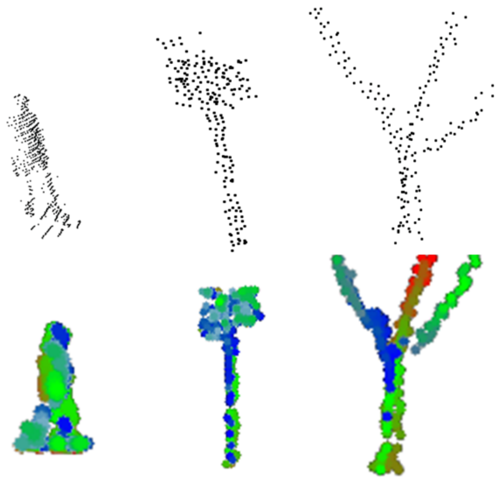
Figure 30 Depth images produced from point clouds

After the projection plane is calculated, the algorithm produces a depth image from the point cloud as Figure 30 shows. In the first step the points of the object are sorted among the smallest dimension, so the method calculates the distance of each points from the projection plane and sorts them. It ensures that the farthest points from the plane are projected last time, so we can keep the depth information of the object in 2D form. The image is colorized by rainbow colors only for demonstration, actually it is a grayscale image for practical purpose (see in the next session). The images have NxN shape, but the projected objects are scaled with their original aspect ratio, furthermore with a global number set to 7 meters. I applied the first scaling so that the objects keep their original shapes and the second scaling is applied since objects need to keep their size difference between themselves. I set the global scaling to 7 meter because the object detector method does not search larger objects than 7 meter. Figure 31 gives further depth image examples, but these are extracted from dense MLS point clouds.

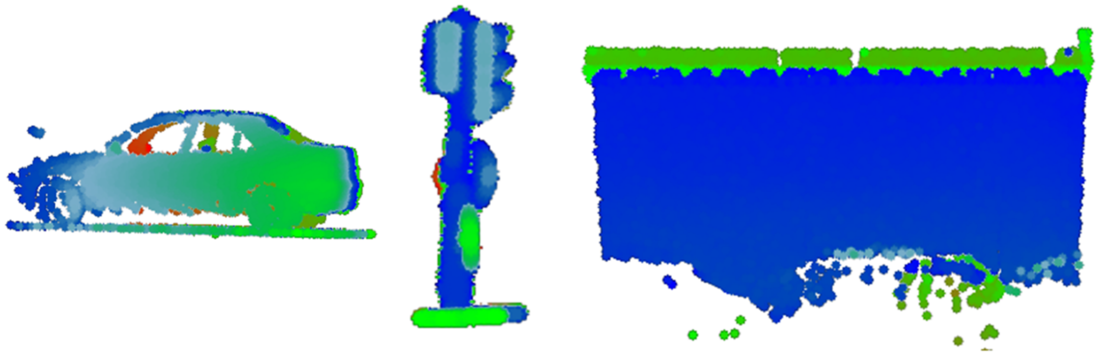

Figure 31 Depth image feature extracted from MLS point cloud

In the next session I give some insight of the machine learning libraries I used for solving classification problem, furthermore I present the structure of the trained network.

6.3.2 Trying different type of features

In the training session I used the feature mentioned above in section 6.3.1, but I also tried two other features. The first representation I tried contains four projections from four different directions. I projected the objects from front, back, top and bottom view and I create a unified image placing the slices next to each other. Experimental results show that about 70% precision is achievable using this feature but since the image is not coherent, it reduced the performance of the auto-feature extracting layers.

I have pursued to find a 2D object feature representation that is coherent and it preserves the objects from all sides. So I implemented a cylinder based projection. In the first step, the algorithm determines the center line of the object what is close to the up vector. Among the center line a predefined radius determines the surface of the cylinder. The method emits a perpendicular "beam" with the center line throughout a given point and the point is projected to where this beam hits the surface of the cylinder. This method gives slightly better result than the previous one, but because of occlusions and partial objects the 2D object prints show a high variance in a class so in practice finding a good model was impossible. I have tried this cylinder based method on objects extracted from terrestrial laser measurements where the projection gives a vey good result, but in this case the objects were very accurate.

6.4 Theano and Lasagne

I used the Lasagne and Theano frameworks [34], [35] to implement deep learning algorithms. Lasagne is a lightweight library to built and train neural networks. It is some kind of interface to make it easier the usage of the Theano engine. Both Theano and Lasagne are based on Python language but the core of the Theano engine has been written in standard C. The engine is highly optimized and it is capable to use Nvidia's GPU by CUDA supporting. Using GPUs during the training process can speed up the learning time even more than 1000 times, because of the massive parallelism. Theano is integrated with NumPy in order to work efficiently with multi-dimensional arrays.

6.5 Structure of the network

I was following the LeNet5-style [33], [36] to design my CNN for classifying street furniture. In addition to the *input* and the *output* layers I used four *convolutional* and two *dense* layers. Furthermore, I applied *max-pooling* and *dropout* layers between the convolutional layers. Figure 32 demonstrates the structure of the trained network.

```
#   name             size
--- ---------------  ---------------
0   input            1x96x96
1   conv2d1          32x90x90
2   maxpool1         32x45x45
3   dropout1         32x45x45
4   conv2d2          64x39x39
5   maxpool2         64x19x19
6   dropout2         64x19x19
7   conv2d3          128x15x15
8   maxpool3         128x7x7
9   dropout3         128x7x7
10  conv2d4          256x5x5
11  maxpool4         256x2x2
12  dropout4         256x2x2
13  dense1           512
14  dropout5         512
15  dense2           512
16  output           4
```

Figure 32 Network structure

Each convolutional layers are followed by a 2x2 max-pooling and a dropout layer. The first two convolutional layers are 7x7, the third is 5x5 and the last one has 3x3 shape. The first convolutional layer starts with 32 filters and it is doubled with every convolutional layers and the dense hidden layers have 512 units. The net was learning for 100 epochs and finishing an epoch session took about 30 minutes. The whole training process was two days. When the learning starts we are far from the optimum, so we want to learn quickly, but the closer we are to the optimum the smaller we want to step. Hence, I initialized the learning rate 0.9 and it decreased linearly with the number of epochs.

The final validation accuracy was 96.42% on the validation set what is a pretty good training result using about 3000 training and 500 validating examples. Increasing the number of training examples, we can get even better results and it will help to avoid over fitting. But I used only those object for training what the object detector algorithm extracted, so I had to manually annotate them and it is a slow process. In the near future I will plan to add objects from

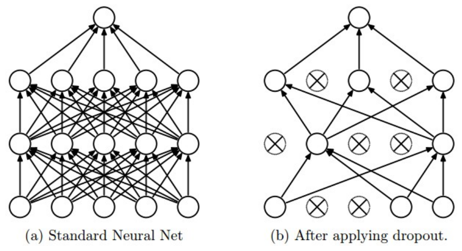

Figure 33 Effect of dropout

different databases to increase the size of the training set. Maintaining over fitting problem I used dropout [37] layers between convolutional and dense layers. Figure 33 demonstrates the effect of using dropout method. Every training iteration the dropout method randomly deletes a portion of the hidden layer, and in particularly we train different nets. We expect that every net over fit in a different way, and so, hopefully, the net effect of dropout will reduce the over fitting.

I separated the 10 percent of the examples for testing purpose and the final performance of the prediction was evaluated on this set.

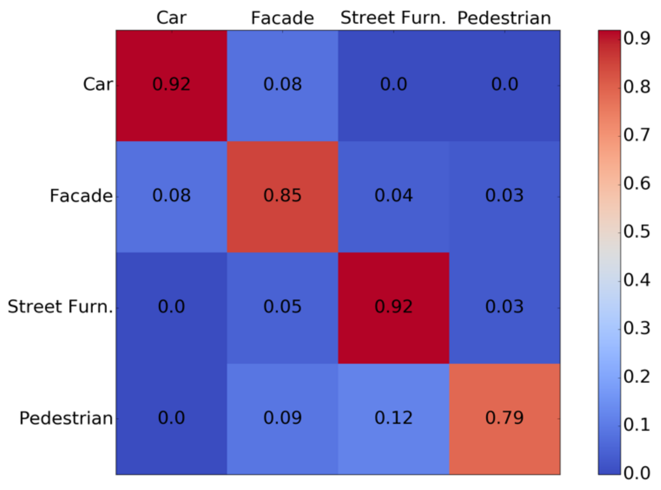

Figure 34 Evaluation of the network

Figure 34 is the confusion matrix of a prediction and we can see that the main diagonal is the most significant region of it. In most cases the net assigns each objects to the preferable class. The test above contained only objects extracted by object detector [8]. I completed the test set from public datasets, especially with pedestrians and facade segments. The tables below show the performance on the new dataset.

	Car	Facade	Furniture	Pedestrian
Car	**513**	39	3	0
Facade	27	**321**	16	1
Furniture	2	21	**376**	31
Pedestrian	0	3	19	**143**

Table 2 Dataset

	Precision	Recall	F-rate
Car	0,962	0,924	0,943
Facade	0,836	0,879	0,857
Furniture	0,874	0,908	0,891
Pedestrian	0,866	0,821	0,843
Sum	0,885	0,883	0,884

Table 3 Performance measurement on the completed dataset

Table 3 shows that the overall performance of the network is 88.4%. Furthermore, we can see the precision, the recall and the F-rate measure for each class separately. Figure 35 presents the result of the prediction process via a Velodyne point cloud frame. As we can see on the figure below, red color represents vehicle class, green is presents the pedestrians, while blue color is assigned to street furniture, such as traffic signs, pole-like objects. Dark gray is marks the facade class and the uncategorized point are colored with light gray color.

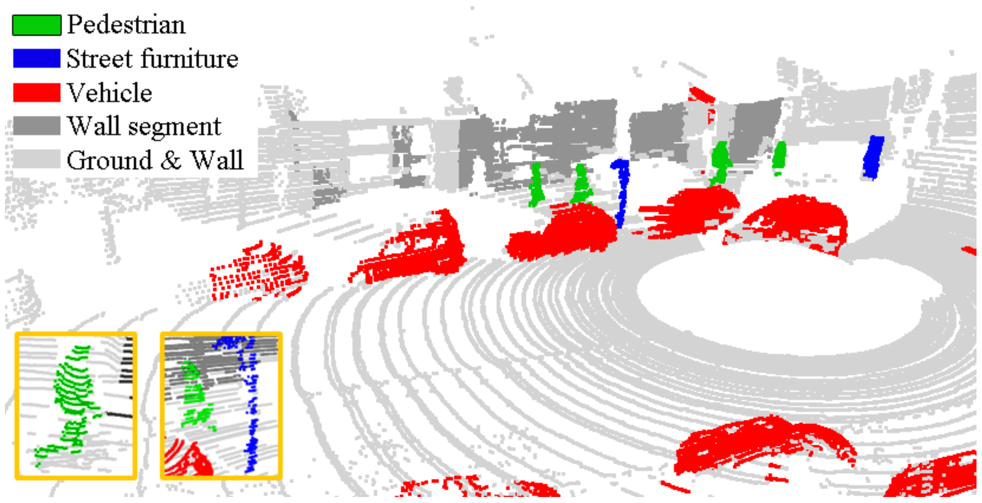

Figure 35 Real urban scenario with classified objects on the street of Budapest, Hungary

The classification algorithm is integrated into a real-time object detection framework [8]. In [8], the objects were classified by an SVM. It was only a binary classification between vehicle and many other objects. This work changes the SVM to the proposed deep learning concept and it performs a multi classification.

The cited framework has written in C/C++, so I had to write a C-Python wrapper. This wrapper ensures to use the Python based prediction from the C environment. To achieve the fastest working process, the feature extraction part (produce the 2D depth images from the 3D objects) is perform in the C side and only the prediction occurs in the Python side.

6.6 Datasets

The training set contains 3000 samples mostly from the outputs of the object detector [8], but the pedestrian and facade class were expanded from Sydney [39] and KITTI [40] datasets.

6.7 Traffic sign classification

I have tired the same network described above, for traffic sign classification. The training set contained 1000 samples from five classes. Figure 36 demonstrates the five classes.

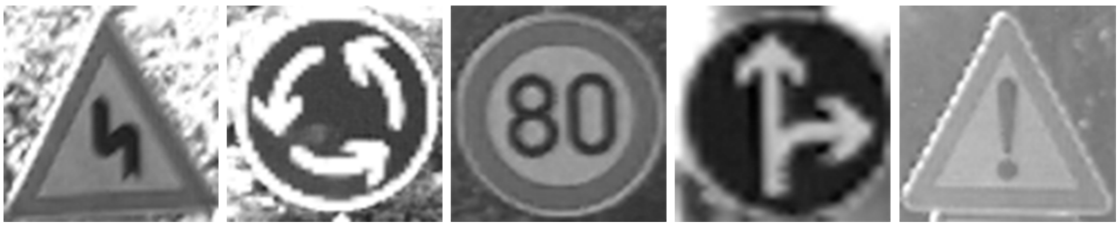

Figure 36 Traffic sign classes (Class0, Class1, Class2, Class3, Class4)

The cropped RGB images were converted into a grayscale form, furthermore the algorithm resized them into 96x96 size. End of the training process the validation accuracy was 88.74%. As a future plan to achieve better results I will train the network with lower learning rate and I will run the training over more hundred iterations. Furthermore, it is considerable to use the color information of the signs.
The trained network was evaluated in an independent dataset contains 500 images from the 5 classes. The prediction accuracy was 79.68% what is a quiet promising result for the first try, because the dataset is very challenging. It contains a lots of burnt-out, too small and occluded traffic signs.

Figure 37 illustrates the result of the traffic sign prediction. The order of the class numbers is the same as the order on Figure 36. You can see that the main diagonal is the strongest region, so in most cases the net predicts well. Furthermore, the most miss classifications are observable between Class0 and Class4 and between Class1 and Class3.

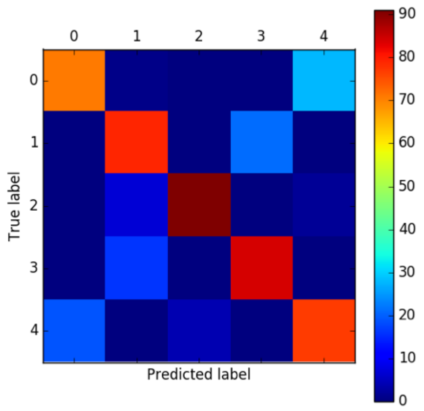

Figure 37 Traffic sign detection evaluation on the test set

7 Conclusion

In the beginning of my thesis I introduced some leading edge Lidar sensors and I presented the basics of laser scanning. In the next chapter I gave insight some data fusion practice between the image and 3D domain. I implemented a method that projects the 3D points onto image pixels, furthermore I propose a frustum based back projection algorithm.

In the course of my work I gave insight the main steps of point cloud processing such as filtering, segmentation and object detection and I also proposed a feature extracting method and a deep learning pipeline to classify objects. I also developed a sparse voxel structure to achieve more robust segmentation, object detection and feature extraction results considering the computation time.

In recent years, companies like Google, Amazon, Tesla or Toyota have been made a huge effort to develop autonomous robots and cars. Visual surveillance, urban planning, traffic control and several GIS applications also play increasing role in nowadays. Sensor companies such as Velodyne or Riegl are developing smaller, more efficient and accurate 3D sensors and mapping systems to satisfy the mentioned applications.

In my future work I will plan to design further algorithms and data structures to reach more accurate object detection. Furthermore, in autonomous driving real-time processing is a crucial need so I will speed up and optimize the algorithms. It is also interesting to use contextual based features and combine the deep learning output with model based machine learning.

Acknowledgement

This thesis work was conducted in the Distributed Events Analysis Research Laboratory (DEVA) of the Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI). Besides providing the infrastructure, MTA SZTAKI ensured the access to a Velodyne HDL-64 laser scanner, which was used for point cloud acquisition in the streets of Budapest. I also thank Budapest Közút Zrt. for providing us the RIEGL measurements and much professional advice.

I thank my colleagues from the DEVA group who gave me insight and expertise that greatly contributed to my research work. Special thanks to my supervisors, Dr. Csaba Benedek and Attila Börcs for assistance with methodology and for comments that greatly improved the manuscript.

List of figures and Abbreviations

Abbreviations

Lidar - Laser Illuminated Detection and Ranging

LASER - Light Amplification by Stimulated Emission of Radiation

CPU – Central Processing Unit

GPU – Graphics Processing Unit

IMU - Inertial Measurement Unit

GPS - Global Positioning System

GNSS – Global Navigation Satellite System

MLS – Mobile Laser Scanning

TLS – Terrestrial Laser Scanning

ALS – Airborne Laser Scanning

ML – Machine Learning

DNN – Deep Neural Network

CNN – Convolutional Neural Network

PCA – Principal Component Analysis

List of figures

Bibliography

[1] T. Dr. Lovas, A. Dr. Berényi and Á. Dr. Barsi, Lézerszkennelés, TERC könyvkiadó, 2012.

[2] Euclideon, "Euclideon unlimited," [Online]. Available: http://www.euclideon.com.

[3] RIEGL, "RIEGL Laser Measurement Systems," [Online]. Available: http://www.riegl.com.

[4] O. d. team, "Camera Calibration," [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html.

[5] Z. Dr. Kató, "Sztereó Kamera," [Online]. Available: http://www.inf.u-szeged.hu/~kato/teaching/IpariKepfeldolgozas/03-StereoCamera.pdf.

[6] M. Himmelsbach, A. Müller, A. Lüttel and H. J. Wünsche, "LIDAR based 3D Object Perception," Proceedings of 1st International Workshop on Cognition for Technical Systems, 2008.

[7] J. Lalonde, N. Vandapel, D. Huber and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," Journal of Field Robotics 23, 2006.

[8] A. Börcs, B. Nagy, M. Baticz, and C. Benedek, "A model-based approach for fast vehicle detection in continuously streamed urban LIDAR point clouds," in Proc. Workshop Scene Understanding Auton. Syst. ACCV, Singapore, 2014, pp. 413–425.

[9] O. d. team, "Template Matching," [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

[10] R. Brunelli, "Template Matching Techniques in Computer Vision," 2008. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/IAPR/researchers/D2PAGES/TUTORIALS/VisMac2008.pdf.

[11] E. Lengyel, Mathematics for 3D Game Programming and Computer Graphics, Third Edition, US: Course Technology PTR, 2011.

[12] C. Früh, S. Jain, and A. Zakhor, "Data processing algorithms for generating textured 3D building facade meshes from laser scans and camera images," Int. J. Comput. Vis., vol. 61, pp. 159–184, 2005.

[13] M. Rutzinger, B. Höfle, S. Oude Elberink, and G. Vosselman, "Feasibility of facade footprint extraction from mobile laser scanning data," Photogrammetrie—Fernerkundung—Geoinf., pp. 97–107, 2011.

[14] L. Zhu, J. Hyyppä, A. Kukko, H. Kaartinen, and R. Chen, "Photorealistic building reconstruction from mobile laser scanning data," Remote Sens., vol. 3, no. 7, pp. 1406–1426, 2011.

[15] S. Pu, M. Rutzinger, G. Vosselman, and S. Oude Elberink, "Recognizing basic structures from mobile laser scanning data for road inventory studies," ISPRS J. Photogramm. Remote Sens., vol. 66, pp. S28–S39, 2011.

[16] M. Rutzinger, A. K. Pratihast, S. Oude Elberink, and G. Vosselman, "Tree modelling from mobile laser scanning data-sets," Photogramm. Rec., vol. 26, pp. 361–372, 2011.

[17] C. Cabo, C. Ordoñez, S. García-Cortés, and J. Martínez, “An algorithm for automatic detection of pole-like street furniture objects from mobile laser scanner point clouds,” ISPRS J. Photogramm. Remote Sens., vol. 87, pp. 47–56, 2014.

[18] M. Lehtomäki, A. Jaakkola, J. Hyyppä, A. Kukko, and H. Kaartinen, “Detection of vertical pole-like objects in a road environment using vehiclebased laser scanning data,” Remote Sens., vol. 2, pp. 641–664, 2010.

[19] A. Jaakkola, J. Hyyppä, H. Hyyppä, and A. Kukko, “Retrieval algorithms for road surface modelling using laser-based mobile mapping,” Sensors, vol. 8, no. 9, pp. 5238–5249, 2008.

[20] Y. Lin and J. Hyyppä, “Geometry and intensity based culvert detection in mobile laser scanning point clouds,” J. Appl. Remote Sens., vol. 4, no. 1, pp. 1–15, Nov. 2010.

[21] S. Awan, M. Muhamad, K. Kusevic, P. Mrstik, and M. Greenspan, “Object class recognition in mobile urban lidar data using global shape descriptors,” in Proc. Int. Conf. 3D Vis., Seattle,WA, USA, 2013, pp. 350–357.

[22] Y. Yu, J. Li, H. Guan, C. Wang, and J. Yu, “Semiautomated extraction of street light poles from mobile LiDAR point-clouds,” IEEE Trans. Geosci. Remote Sens., vol. 53, no. 3, pp. 1374–1386, Mar. 2015.

[23] B. Yang and Z. Dong, “A shape-based segmentation method for mobile laser scanning point clouds,” ISPRS J. Photogramm. Remote Sens., vol. 81, pp. 19–30, 2013.

[24] B. Yang, Z. Dong, G. Zhao, and W. Dai, “Hierarchical extraction of urban objects from mobile laser scanning data,” ISPRS J. Photogramm. Remote Sens., vol. 99, pp. 45–57, 2015.

[25] A. Golovinskiy, V. G. Kim, and T. Funkhouser, “Shape-based recognition of 3D point clouds in urban environments,” in Proc. IEEE 12th Int. Conf. Comput. Vis., Kyoto, Japan, 2009, pp. 2154–2161.

[26] M. Himmelsbach, T. Luettel, and H. J. Wuensche, “Real-time object classification in 3D point clouds using point feature histograms,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., St. Louis, MO, USA, 2009, pp. 994–1000.

[27] E. Puttonen, A. Jaakkola, P. Litkey, and J. Hyyppä, “Tree classification with fused mobile laser scanning and hyperspectral data,” Sensors, vol. 11, no. 5, pp. 5158–5182, 2011.

[28] A. Serna and B. Marcotegui, “Detection, segmentation and classification of 3D urban objects using mathematical morphology and supervised learning,” ISPRS J. Photogramm. Remote Sens., vol. 93, pp. 243–255, 2014.

[29] X. Zhu, H. Zhao, Y. Liu, Y. Zhao, and H. Zha, “Segmentation and classification of range image from an intelligent vehicle in urban environment,” in Proc. IEEE Int. Conf. Intell. Robots Syst., Taipei, Taiwan, 2010, pp. 1457–1462.

[30] T. Chen, B. Dai, D. Liu, and J. Song, “Performance of global descriptors for Velodyne-based urban object recognition,” in Proc. IEEE Intell. Veh. Symp., Dearborn, MI, USA, 2014, pp. 667–673.

[31] Nvidia, “NVIDIA DGX-1 Deep Learning System”, [Online]. Available: http://www.nvidia.com/object/deep-learning-system.html

[32] ImageNet, [Online]. Available: http://www.imagenet.org/

[33] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks”, Neural Information Processing Systems, 2012

[34] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley and Y. Bengio. “Theano: new features and speed improvements”. NIPS 2012 deep learning workshop.

[35] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. “Theano: A CPU and GPU Math Expression Compiler”. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX.

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.Proceedings of the IEEE, november 1998.

[37] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, Computer Vision and Pattern Recognition, 2012 Jul. 3.

[38] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella and M. Gross. Post-processing of Scanned 3D Surface Data, Eurographics Symposium on Point-based Graphics, 2004.

[39] Australian Centre for Field Robotics, “Sydney urban objects dataset”, [Online]. Available: http://www.acfr.usyd.edu.au/papers/SydneyUrbanObjectsDataset.shtml

[40] Andreas Geiger and Philip Lenz and Christoph Stiller and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset, International Journal of Robotics Research (IJRR), 2013.

[41] H. Guan et al., “Using mobile laser scanning data for automated extraction of road markings,” ISPRS J. Photogramm. Remote Sens., vol. 87, pp. 93–107, 2014.

[42] C. Mc Elhinney, P. Kumar, C. Cahalane, and T. McCarthy, “Initial results from European Road Safety Inspection (EURSI) mobile mapping project,” in Proc. ISPRS Commission V Tech. Symp., 2010, pp. 440–445.

[43] Y. Yu, J. Li, H. Guan, C. Wang, and J. Yu, “Automated detection of road manhole and sewer well covers from mobile LiDAR point clouds,” IEEE Geosci. Remote Sens. Lett., vol. 11, no. 9, pp. 1549–1553, Sep. 2014.

[44] B. Yang, Z. Wei, Q. Li, and J. Li, “Automated extraction of street-scene objects from mobile lidar point clouds,” Int. J. Remote Sens., vol. 33, pp. 5839–5861, 2012.

[45] S. Friedman and I. Stamos, “Online detection of repeated structures in point clouds of urban scenes for compression and registration,” Int. J. Comput. Vision, vol. 102, pp. 112–128, 2013.

[46] A. K. Aijazi, P. Checchin, and L. Trassoudaine, "Segmentation based classification of 3D urban point clouds: A super-voxel based approach with evaluation," Remote Sens., vol. 5, pp. 1624–1650, 2013.

[47] X. Ning, Y. Wang, and X. Zhang, "Object shape classification and scene shape representation for three-dimensional laser scanned outdoor data," Opt. Eng., vol. 52, p. 024301, 2013.

[48] H.Wang et al., "Object detection in terrestrial laser scanning point clouds based on Hough forest," IEEE Geosci. Remote Sens. Lett., vol. 11, no. 10, pp. 1807–1811, Oct. 2014.

[49] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid," In: Proc. IEEE Int. Conf. on Robotics and Automation, 2007, Rome, Italy, pp. 1610–1616.

[50] O. Józsa and Cs. Benedek, ''Reconstruction of 3D Urban Scenes Using a Moving Lidar Sensor," MTA SZTAKI, 2013, Budapest, Hungary, Tech. Rep. N ° i4D-1, pp. 10-13.

[51] Matti Lehtomäki, Anttoni Jaakkola, Juha Hyyppä, Jouko Lampinen, Harri Kaartinen, Antero Kukko, Eetu Puttonen, and Hannu Hyyppä, "Object Classification and Recognition From Mobile Laser Scanning Point Clouds in a Road Environment", IEEE Transactions On Geoscience and Remote Sensing, Vol. 54, No. 2, Feb. 2016.