POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

# Visually aided changes detection in 3D lidar based reconstruction

Relatore: Prof. Matteo Matteucci
Correlatore: Ing. Andrea Romanoni

Tesi di Laurea di:
Gheorghii Postica, matricola 814370

Anno Accademico 2014-2015

*Alla mia famiglia*

# Abstract

Detecting changes in visual data has been of utmost importance in many disciplines, including computer vision, robotics, remote sensing, civil infrastructure, surveillance and medical diagnosis. In many computer vision systems, the detection of changes, in most of the cases, serve as a preparation step for other phases of the system, and is usually carried out by background subtraction algorithms, which assume the scene was acquired by a static camera. The problem of detecting changes in a scene captured by a moving sensor or camera is still an open problem both in computer vision and remote sensing. The aim of this thesis is to present a system capable of detecting and removing dynamic objects both in 3D scene acquired by a moving lidar device and in 2D images from a moving camera. We used the Demspter-Shafer Theory to represent the space occupancy and to aggregate the evidences of such occupancies from multi-temporal scene measurements. To avoid useless computation, we used a method to remove ground points from change detection test. To speed up the execution, we used octree data structure to partition the point cloud in voxels, and perform all the necessary change detection computation on a voxel level. The detected changes from the 3D scene are then propagated into the images where a subsequent step of validating these changes is performed. The validation is done on both color images and depth images rendered from the 3D point cloud. We are able to obtain, with a very high speed and a good precision, static 3D scenes without any dynamic objects and a set of binary masks representing changes from images. Moreover, the static scenes are merged into a global scene. The global scene can be decimated ready to be reconstructed. The presented system is highly modular and very scalable for future extensions such as 3D reconstruction.

# Sommario

Individuare i cambiamenti in una scena ripresa da una camera o percepita da laser è di grandissima importanza in tante discipline, come visione artificiale, telerilevamento, video sorveglianza e diagnosi medica. Nell'ambito della visione artificiale questi cambiamenti vengono solitamente riconosciuti con tecniche di sottrazione dello sfondo; queste tecniche però assumono che la camera sia fissa. Il problema del rilevamento dei cambiamenti nelle scene acquisite da sensori mobili è ancora un problema aperto nel campo della vision artificiale e telerilevamento. Lo scopo di questa tesi e´di presentare un sistema capace di rilevare congiuntamante gli oggetti in movimento nelle scene 3D acquisite dai dispositivi lidar in movimento e nelle immagini scattate da camere in movimento. In questo lavoro abbiamo usato la teoria di Dempster-Shafer per modellare l'occupazione dello spazio e per aggregare i dati temporali per riconoscere i punti appartenenti ad oggetti in movimento nella scena. Abbiamo inoltre partizionato la scena in una struttura dati octree per aumetare la velocita´di calcolo. I cambiamenti rilevati nella scena 3D vengono poi propagati nelle immagini dove viene eseguito un test di validazione di questi cambiamenti. Il test di validazione si basa su immagini a colore sincronizzate con il lidar, e le depth map estratte dalle misure di quest'ultimo. Con questo sistema, siamo riusciti ad ottenere scene statiche 3D senza oggetti in movimento e delle maschere binarie rappresentanti le regioni relative agli oggetti in movimento, con un'elevata velocità di esecuzione ed una buona precisione. La nuvola di punti 3D senza oggetti può essere così utilizzata per ricostruire la scena 3D in un passo successivo.

# Ringraziamenti

Desidero ringraziare il professor Matteo Matteucci per la sua disponibilità, la sua umanità e anche per le sue ottime idee, nonch'e per il tempo speso ad aiutarmi.

Ringrazio inoltre ing. Andrea Romanoni, che mi é stato sempre d'aiuto, mi ha dato sempre degli ottimi consigli e mi ha seguito fino all'ultimo minuto. Non sarei riuscito a finire questo lavoro senza di lui.

Un ringraziamento speciale va alla mia famiglia, a cui devo questi anni di soddisfazioni e fatiche, che mi hanno aiutato nei momenti più difficili e che mi hanno reso la persona che sono. Grazie per essermi stati vicini nei giorni peggiori cosi come anche nei giorni migliori e per avermi incorragiato tutti questi anni.

Desidero ringraziare anche la Corina per la sua immensa pazienza, per il suo sostegno e per l'incorragiamento in questi ultimi anni.

Ringrazio anche tutti i miei amici sia dell'università che fuori, con cui ho condiviso gioie e ansie.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The problem of reconstructing an environment from real world is widely studied. Many computer science branches, such as computer vision and robotics, at some point need a map of the environment, be it in symbolic, numerical or spatial form of information. The map is used as a model of the environment to solve other challenges, such as 2D/3D reconstruction and navigation of an agent in the environment. However, when mapping occurs, many problems arise, such as questionable quality of the measurements and the presence of temporary objects. Since these objects are present only at the time of the mapping, they should not be in the final map, otherwise the model is erroneous and may invalidate the results of the processes relying on that model. Therefore we propose a method to detect and eliminate those objects with a good precision and in a reasonable amount of time.

## 1.1 Detection of changes in high-dimensional data

Detecting dynamic objects in complex scenes can be applied to many processes, such as mapping, navigation, collision avoidance and 3D urban reconstruction. For localization and mapping a static environment is preferred, as the presence of dynamic object will deteriorate localization and mapping efficiency, thus a detection of these dynamic objects is very beneficial in this case. The obstacle avoidance also can benefit from the change detection step, as the detected dynamic objects can be tracked and avoided. Moreover, the dynamic objects detection and removal is also very important for urban 3d reconstruction methods. The 3D urban reconstruction methods build 3D models of the environment using high-dimensional data acquired by a multitude of sensors. These are used for many applications such as autonomous navigation. Urban area scenes are usually filled with many pedestrians, cyclists, cars and other public transportation. Without any of these moving objects, the 3D reconstruction methods become more precise with more pleasant results.

The problem of detecting changes is addressed in many disciplines with different techniques. In the computer vision field, where images are processed, analyzed and understood in order to obtain numerical information for various purposes, the changes most often are detected with background subtraction techniques. These techniques very often are based on the static camera assumption or, in some cases, on a slowly moving camera assumption. However these methods fail when the images are taken from a fast moving camera, as the distinction between static and dynamic objects fades out. In fact, the problem of detecting dynamic objects by a moving sensor is still an open one. In robotics many change detection techniques are based on indexing the data acquired from range sensors into various efficient data structures and perform a background subtraction-like technique with various instances of this data. The range sensors are also called active sensors, since their working principle is to send signal probes, be it light or ultra sound, and later, once they hit hard surfaces and reflect, to capture them. These sensors, unlike cameras which measure the world state in 2D in a given instant, can measure and represent the world in 3D. This gives the opportunity to analyze the model from different perspectives and understand the sensor motion in the world. This information can help in solving the problem of detecting moving objects in a cluttered environment by a moving sensor

To cope with this problem, we propose a detection method which uses a combination of 3D information of the world, as point cloud data, and 2D information, as images. This method uses the Dempster-Shafer Theory to represent the space occupancy and to aggregate the evidences of such occupancies from multi-temporal scene measurements acquired by a lidar device. The aggregation helps to classify the points between dynamic and static. The dynamic points validate using a set of color images, synchronized with the lidar, and depth maps extracted from laser scanning measurements. This validation step removes the incorrectly classified static points as dynamic and outputs a set of change mask which can be used for further uses.

We propose some enhancements to the method described above, namely an octree data indexing optimization and ground plane removal. The former allows the detection algorithm to work on a subset of points at a voxel level, translating in a substantial speed up. The ground plane removal helps to speed up the execution by avoiding useless computation, but it also heps lowering the mislassification rate of dynamic points. We also perform a final decimation step on the global point cloud to prepare it for future uses, such as 3D reconstruction.

The experimental results show that our algorithm outperform the state-of-the art algorithm both in accuracy and computational speed and that the proposed validation step further improves the change detection performed on lidar data.

## 1.2 Thesis outline

This thesis is structured as follows. In Chapter 2 an overview of existing change detection methods in literature is presented. Chapter 3 describes shortly all the material used to design and perform the algorithm presented in this thesis. In Chapter 4, the system architecture is described along with its core part of 3D change detection method. The Chapter 5 describes the detected changes validation method. The experimental results of the proposed method are shown and discussed in Chapter 6. Finally the conclusions and future work are presented in the Chapter 7.

# Chapter 2

# State of the art

The problem of change detection have been actively studied since 1970s with results that have been applied in many fields. In the next sections a brief analysis of change detection methods in photogrammetry, robotics and computer vision is given.

## 2.1 Image-Based Change Detection

Image-based change detection has been of widespread interest due to a vast number of applications in various disciplines (video surveillance, remote sensing, civil infrastructures, driver assistance systems, medical diagnosis and treatment, etc). Change detection is a crucial step in many computer vision applications, especially for video surveillance [61, 15], since people, vehicles, animals, etc. have to be detected before operating more complex processes for intrusion detection, tracking, people counting, etc.

The image-based change detection goal is to identify the set of pixels that are “significantly different” between the last image of the sequence and the previous images, the result being a foreground/background (foreground being the image points of non-stationary objects) segmentation also known as change mask (or foreground mask). Factors contributing to change mask are: appearance and disappearence of objects, motion of objects relative to the camera, shape changes of objects and changes in brightness and color of stationary objects. There are however unwanted factors contributing to change mask including sensor noise, illumination variation, shadows and non-uniform attenuation.

Change detection is usually performed in computer vision field by Background Subtraction (BGS) methods, which have been actively researched for

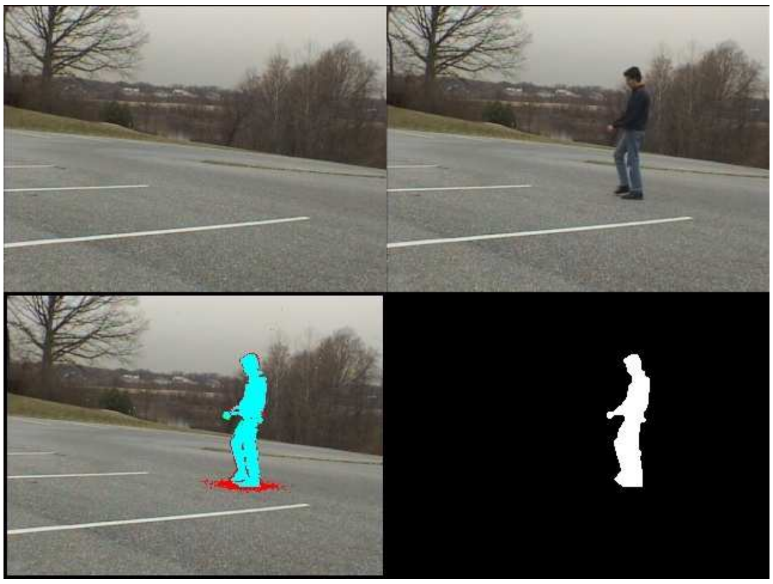


*Figure 2.1: An example of BGS. From top left in clockwise direction: background image, background with foreground, foreground detection and finally the change mask.*

at least three decades. An extensive study and evaluation of existing BGS algorithms has been performed in [41] and later in [58], and a general survey on various change detection techniques and problems, although heavily focused in remote sensing context, is given in [56] and [44], while an evaluation on the best performing algorithms considering the post-processing step is shown in [40].

A typical BGS algorithm generally performs three main steps: *background initialization*, *foreground detection* and *background maintenance*. In the background initialization step a background model is built, the foreground detection step performs a comparison (usually frame subtraction) and separates the foreground (change mask) while the background maintenance step adjusts the background model during the detection process by a learning rate (e.g. objects which stopped moving are slowly integrated into the background). Other steps may include *image pre-processing*, *foreground mask post-processing* and *motion compensation* (usually for moving cameras methods).

### 2.1.1 Pre-processing and Post-processing

Sometimes a pre-processing step is required to improve the final outcome of BGS. This step may consist in geometric adjustments techniques, where typically an image registration is perfomed, and various radiometric/intensity adjustments such as intensity normalization, homomorphic filtering and illumination modeling. In computer vision field pre-processing usually involves noise filtering, image manipulation (such as sharpening and smoothing) and edge detection.

In many change detection methods, the output foreground mask may have holes and rough edges, to fix these imperfections, usually the mask is improved by a post-processing step. Parks and Fels [40] have surveyed and evaluated various well-performing algorithms with an added post-processing step. They listed various post-processing techniques including noise removal, blob processing (morphological closing and area thresholding), saliency test, optical flow test and object-level feedback. The authors have shown a qualitative improvement for all considered BGS methods just by applying the post-processing step.

### 2.1.2 Basic Methods

The most basic and early BGS method, called *Simple Differencing*[48], is modeling the background $B$ with a static image without any moving objects and then testing the absolute difference between frames against a given threshold $\tau$:

$$F(\mathbf{x}) = \begin{cases} 1 & \text{if } |I(\mathbf{x}) - B(\mathbf{x})| > \tau \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $F$ is the foreground mask. These methods however require a static background (no moving objects) and fail with illumination changes(even with very small changes) and camera noise. To overcome the small illumination changes issue, a slightly better method was introduced, where the background maintenance step is added. In [32] the background is modeled and maintained by the arithmetic mean of the pixels between successive images:

$$B = \frac{1}{l}\sum_{t=1}^{l} Z_t \tag{2.2}$$

where $Z_t$ is the image at time $t$ and $l$ is the number of considered frames. The maintenance is performed by a recursive function:

$$B_t = (1 - \alpha)B_{t-1} + \alpha Z_t \tag{2.3}$$

where $\alpha \in [0, 1] \subset \mathbb{R}$ is the learning rate and $B_t$ is the background model at time $t \in \{1, l\} \subset \mathbb{Z}$. This method however has an issue as stated by [55], some foreground pixels are integrated into the background model during update process due to a crisp threshold. It happens most often in scenes with moving trees, water rippling and other dynamic background phenomena. In [55] a solution to this problem was proposed, which consists in updating only the regions with no moving objects.

For foreground detection, aside from absolute difference, many techniques have been proposed, such as texture and color features fusion [75], or texture features, RGB color features and Sobel edge detection fusion [2]. These methods may seem diverse, but they all have the same core idea of frame differencing.

### 2.1.3 Statistical Methods

Since basic methods suffer from camera noise and dynamic background (such as waving leaves and branches of the trees, water rippling, water surface, etc.), a new type of BGS techniques emerged: *statistical methods*. The core idea of statistical methods is to use a parametric probabilistic background model and statistical variables to classify pixels as foreground or background. Bouwmans[9] does an excellent job of discussing the various statistical methods. The statistical models proved to be more robust and better performing than basic ones.

One of the first statistical methods is the Single Gaussian Model (SGM) introduced by Wren et al[69]. In this work the background is modeled independently at each pixel location by ideally fitting a Gaussian probability density function (pdf) on the last $n$ pixel's values. In order to avoid fitting the pdf from scratch at each new frame time $t+1$, the mean and the variance are updated by a running average. If $|\mu_t - \mathbf{z}_t| < T$ ($\mathbf{z}$ is the pixel value and $T$ a user-defined threshold), the pixel is classified as background otherwise the pixel is classified as foreground.

However the issue of dynamic background still persists with SGM methods and a better solution and in fact the most used statistical model is the *Gaussian Mixture Model* (GMM), introduced by Stauffer and Grimson[59] and further improved by Hayman and Eklundh[25]. In this model the distributions of each pixel color is represented by a sum of weighted Gaussian distributions defined in a given colorspace. These distributions are then updated using an online expectation-minimization algorithm. The multiple distributions per pixel allow dealing with multimodal backgrounds and gradual illumination changes. The likelihood that a pixel is a background pixel is given by:

$$P(\mathbf{z_t}|\mathbf{m_t}) = \sum_{n=1}^{N} w_n \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_{\boldsymbol{n}}|^{1/2}} \exp\{-\frac{1}{2}(\mathbf{z_t} - \boldsymbol{\mu_n})^T \mathbf{\Sigma}_{\boldsymbol{n}}^{\mathbf{-1}}(\mathbf{z_t} - \boldsymbol{\mu_n})\} \quad (2.4)$$

where $d$ is the dimension of color space of the pixel measure $\mathbf{z}_t$, $\mathbf{m}_t$ is the pixel model and each Gaussian $n$ is described by its mean $\boldsymbol{\mu}_n$ and coviariance matrix $\boldsymbol{\Sigma}_n = \boldsymbol{\sigma}_{i,n}^2 * \mathbf{I}$ ($i$ is color channel, each color channel is indipendent, $*$ is the element-wise multiplication and $\mathbf{I}$ is the identity matrix). The Gaussians are weighted by factors $w_n$ where $\sum_{n=1}^{N} w_n = 1$, and $|\cdot|$ is the matrix determinant. At each new frame the Gaussians are classified in foreground and background (usually using a threshold), then each pixel is tested to match a certain Gaussian (foreground or background) with subsequent creation of foreground mask. After the mask creation step, the parameters are updated in the following way: if the previous match succeded then the matched Gaussian is updated, otherwise the least probable Gaussian is replaced by a new one with default parameters.

While GMM has shown good performance it still suffers from sudden illumination variations, noise and shadows presence. Many studies have been done to overcome these problems. Kaewtrakulpong and Bowden [30] improved the update equations to better adapt the system to illumination changes. Chen et al. [13] propose a hierarchical and block-based BGS algorithm combining GMM and a contrast histogram.

Other more advanced and complex statistical methods are: Kernel Density Estimation (KDE) [20], Subspace Learning using Principal Component Analysis (SL-PCA) [39], Support Vector Models (SVM) [33] and others. The first one estimates background probabilities at each pixel from many recent samples over time using KDE (which is rather time consuming). The second applies SL-PCA on $N$ images to construct a background model, which is represented by the mean image and the projection matrix comprising the first $p$ significant eigenvectors of PCA. This leads to a foreground mask created from the difference between the input image and its reconstruction. The latter method models the background with probabilistic SVM (obtained from binary SVM[66] through a sigmoid model) which outputs a given probability (instead of binary classification) in foreground detection step, which is then confronted to a threshold for change mask creation.

### 2.1.4 Other Methods

In literature, there are many other models for change detection. In *Predictive Spatial Models* the intensity values of each pixel block are fitted to a polynomial function of pixel coordinate $(x, y)$:

$$\hat{I}_k(x, y) = \sum_{i=0}^{p} \sum_{j=0}^{p-i} \beta_{ij}^{k} x^i y^i \tag{2.5}$$

where $\hat{I}_k(x, y)$ is the pixel block centered on $(x, y)$ and $\beta^k$ is a polynomial coefficient. If the corresponding blocks in two images are best fit by the same polynomial coefficients $\beta^0_{ij}$, then the central pixel is from background, otherwise it is from foreground (different coefficients $\beta^1_{ij}$, $\beta^2_{ij}$). A detailed discussion of this method can be found in [28] and its further improvement in [57]. In *Predictive Temporal Models* pixel intensities are modeled over time as an autoregressive (AR) process. A good example of this model is the Wallflower algorithm by Toyama et al.[62], where a Weiner filter is used to predict pixel's current value from a linear combination of its $k$ previous values.

Some of the most recent models introduced the *fuzzy* concepts in change detection. A good result was obtained in [19] by using Choquet Integral [14] to measure the similarity of color and texture features (and even edge features in [2]) of input image and background model. Sigari et al.[55] proposed to extract the foreground and update the background model using a fuzzy function. They used a fuzzy running average to update the background model. Also for background subtraction, instead of using a crisp limiter (threshold), they used a linear saturation function with a further "binarization" by a low pass filter. This gives the advantage of having a noise reduction in foreground mask. An extensive overview of fuzzy based methods is given in [10].

Other recent techniques in BGS are based on neural networks, bayesian networks and neuro-fuzzy methods. Neural network is used to learn and classify each pixel in foreground or background [9]. Authors in [16] used a multilayered feed-forward neural network to learn the background model and a Bayesian classifier to identify foreground pixels.

One of the most recent works in BGS field, done by Bianco et al.[7], introduced a rather new concept to the field. The authors used Genetic Programming (GP) to combine inputs of detection algorithms with unary, binary and n-ary functions performing both foreground mask's combination and post-processing. Since there is no universal change detection algorithm which works well for all scenarios, Bianco et al. opted for GP to get the best out of various BGS algorithms. They claim their method performs better than more complex algorithms and it ranks first in ChangeDetection.net[29] average ranking.

### 2.1.5 Change Detection with Non-Stationary Camera

The methods introduced have a rather important flaw, they all have the assumption that the camera is stationary, which nowadays is too restrictive for industrial and commercial applications. A common and recurring theme in BGS with camera motion is to adapt already existing BGS methods for stationary cameras. This is usually done by incorporating an image registration step and a panoramic stitching of background models, also known as mosaicing [4]. A certain downfall to these methods is the increased computation time (due to a bigger background model) and stitching errors (especially on edges).

In order to avoid these pitfalls, Kim et al.[31] had the idea of using the same background image dimensions and overlap it on the last image in video sequence. Their method works by first pre-processing both the background and last frame, then the background is warped over the last frame using registration techniques. The foreground is extracted through a background subtraction based on pixel-wise spatio-temporal distribution of Gaussian, as this overcomes misclassified pixels due to small registration errors. To cope with illumination changes, the authors modeled the background with a SGM and added the aging factor into the updating formulae. Finally, as post-processing, they track moving objects through a discrete proportional-integral-derivative (PID) method, and then a probabilistic morphological refinement of the foreground mask is applied.

Sheikh et al.[54] proposed a rather different approach to deal with BGS with camera movements. Under the assumption that all 2D trajectories projected from stationary objects (3D scene background) must lie in the same subspace, they used a constrained model for the background called background trajectory space. A RANSAC estimator is run to estimate this space where outliers become foreground pixels and inliers are background pixels. A subsequent optimal pixel-wise foreground/background labeling is obtained by efficiently maximizing a posterior function. The shortcomings of this approach however are the use of affine camera instead a projective one, long initial delay and fast camera movements.

Authors in [72] noted that one of the arising issues when applying GMM to images taken by non-stationary cameras is due to a fixed learning rate. They proposed to model the background with a dual SGM using aging. Since their main focus was more on speed and less on quality, in their work the images are divided into grids, and pixels of the same cell are modeled with the same Gaussian. The novel idea was to keep and update a backup Gaussian model and swap the cells in the actual Gaussian model whether the backup cell have a bigger age than the actual one. This proved to avoid contaminating the background with the foreground and thus less misclassified pixels. The authors also proposed a motion compensation method which includes homography estimation and a further refinement specifically designed for grid structure. They claim to have their algorithm running in real-time on a mobile phone and with comparable results as the state of the art algorithms.

Romanoni et al.[47] proposed a new approach to deal with jittery camera change detection. In their work they suggested using a combination of temporal and spatio-temporal models to create the foreground mask. The authors opted for histogram representation for their models, as it best fits the discrete pixel intensity distribution, and the Bhattacharyya distance for histograms comparison. Moreover they proposed a simple, yet effective technique for dealing with sudden brightness changes usnig only the median operator.

### 2.1.6 Evaluation

Since evaluation of change detection algorithms is not an easy task, many image-based change detection benchmarks have emerged recently to help establish fair algorithms evaluation, most notable are ChangeDetection.net [29], SABS (Stuttgart Artificial Background Subtraction Dataset)[11], Multivision [22] and BMC (Background Models Challenge)[64]. These datasets contain challlenging videos (both synthetic and real), ground truth change masks, various algorithms ranking, and are fully available on the Web.

The ChangeDetection.net dataset provide a realistic, camera-captured (no CGI), diverse set of videos. These videos include the following challenges: dynamic background, camera jitter, intermittent object motion, shadows and thermal signatures.

The SABS dataset consists of video sequences for nine different challenges of background subtraction for video surveillance. The dataset contains ground-truth annotation provided as color-coded foreground masks and additional shadow annotation that represents for each pixel the absolute luminance distance between the frame with and without foreground objects

The Multivision dataset consists of two different sets of sequences: the first set has been recorded by using stereo cameras combined with three different disparity estimation algorithms; the second one has been recorded by using the Kinect sensor from Microsoft.

The BMC dataset provides two sets of real and synthetic videos. The dataset also provides a software designed to calculate the quality criteria for an efficient evaluation of a background subtraction algorithm.

## 2.2 Point Cloud Change Detection

Change detection based on remote sensing has been actively researched since the early 1980s. The main techniques were developed by photogrammetry community. However with the introduction of laser scanning technology, also known as LiDAR (Light Detection And Ranging), a lot of new research focused on change detection emerged, espetially in Geographical Information Systems (GIS) context. Laser scanners have the great advantage over photogrammetry of capturing a direct description of 3D geometry independent of lighting conditions and in a rapid way. Applications are found for monitoring and characterizing urban growth [12], for monitoring ecological conditions (including canopy cover [45] and landslide monitoring [27]), for assessing damages after natural disasters [67, 38], to assist in monitoring construction projects [24], or for infrastructure maintenance. Various laser scanners applications can be categorized, in terms of platforms, in airborne laser scanning (ALS), terrestrial laser scanning (TLS) and mobile laser scanning(MLS).

### 2.2.1 Airborne laser scanning

Airborne laser scanning (ALS) [5, 68] (introduced in 1988 by University of Stuttgart) offers a large coverage over a short time period and a high accuracy in three dimensions, and it can be performed from helicopters and fixed-wing aircrafts. Since ALS point clouds can be interpreted as a heightmap (2.5D) image, most change detection techniques for ALS data are pretty straightforward and usually consist in converting the ALS point clouds into Digital Surface Models (DSM) and from there photogrammetry change detection techniques are used (sometimes even a simple frame differencing is enough), as can be seen in [60].

### 2.2.2 Terrestrial laser scanning

Terrestrial laser scanning (TLS) is more precise and has a higher point density than ALS but instead has a smaller coverage. TLS change detection is usually applied in detection of landslides [27], construction sites [24], snow coverage [1] and other deformations measurements [37].

Girardeau et al. [24], in order to detect changes in construction site, stored TLS point clouds in an octree data structure (discussed later in more detail) and proposed three strategies to find differences in octree cells: average distance, best fitting plane orientation and Hausdorff distance. The idea is to use the same octree structure for both point clouds (previous and

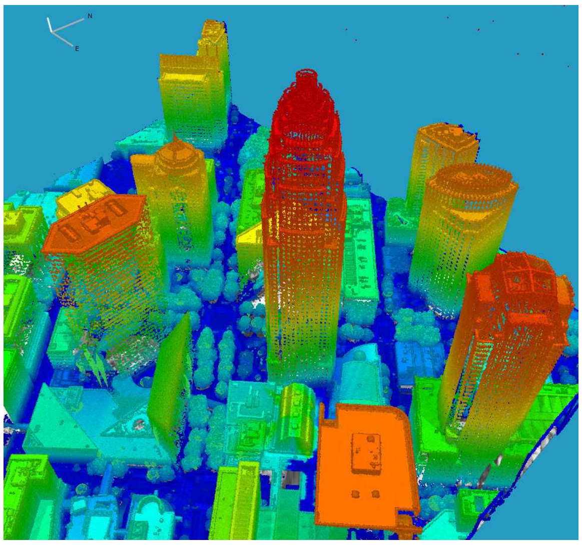


*Figure 2.2: An example of data acquired by ALS means*

current "frame") and then make comparisons of points in the same cell from both point clouds. Average distance strategy compares the average distance (same for all points in cell) of each point in one point cloud from its nearest neighbor from the other point cloud to a threshold. Best fitting plane orientation fits a plane in each cell from points of one point cloud and performs an outlier test with points from the other point cloud. Finally the best, but slowest result, was obtain with the Hausdorff distance (the maximum distance from a point in one set to the closest point in another set) strategy, which is computed for each point and if the distance is bigger than a threshold, the point is marked as changed.

Zeibak and Filin [74] had the idea to create depth maps from precisely registered TLS point clouds. They proposed to transform a point cloud in the frame of the other point cloud using transformation parameters (known from registration step), and then to convert the point clouds into depth maps. This way, after a pre-processing step, a simple frame differencing on depthmaps yields quite a good result of foreground mask (change points) which is then further refined in a post-processing step, where spurious change points are filtered. Their result looks promising, however it should be noted that they have used already registered (with very high accuracy) point clouds and their algorithm cannot resolve some conflicts with partial occlusions. The authors also give some insight on possible problems in change detection with TLS, including occlusions, sensor noise, non-reflective materials (e.g. windows) and non-uniform point density.

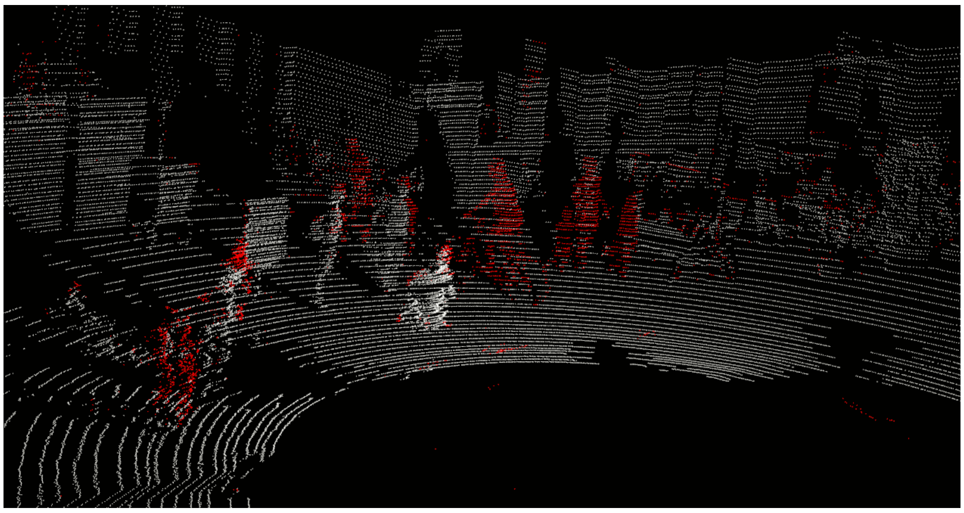


*Figure 2.3: An example of MLS change detection, red points correspond to moving objects*

### 2.2.3 Mobile laser scanning

Mobile laser scanning (MLS) has the advantages of both previously discussed laser scanner types and provides ease of mobilization and low costs when compared to airborne laser scanning and terrestrial laser scanning. It is lighter and less bulky thus easier to install on a ground vehicle. The spatial coverage is achieved by the movement of the vehicle and motion-tracking navigation devices such as Global Positioning System (GPS) and Inertial Measurement Unit (IMU). MLS has been used in many applications including detection and modeling of streets and their facilities [42], tracking of moving objects [3] and, in conjunction with ALS, a more detailed features extraction in a complex urban environment [76].

Change detection with MLS is a relatively young topic, especially in computer vision. Azim and Aycard [3] noted that change detection is more problematic when using 3D LiDARs mainly because of the big amount of data to store and process (e.g. Velodyne HDL-64E outputs 1.3 million points per second which translates to roughly 17MB per second). They proposed using an octree occupancy grid for the point cloud and further usage of this grid to find inconsistencies between scans. Every voxel (octree cell) in the occupancy grid is classified as either free or occupied based on a raytracing technique. The detection is then very similar to BGS, although in 3D, where every voxel from one scan is compared to the corresponding voxel from the other scan. If a voxel from one scan is free and occupied in the other, then the voxel is called dynamic. The authors had the interesting

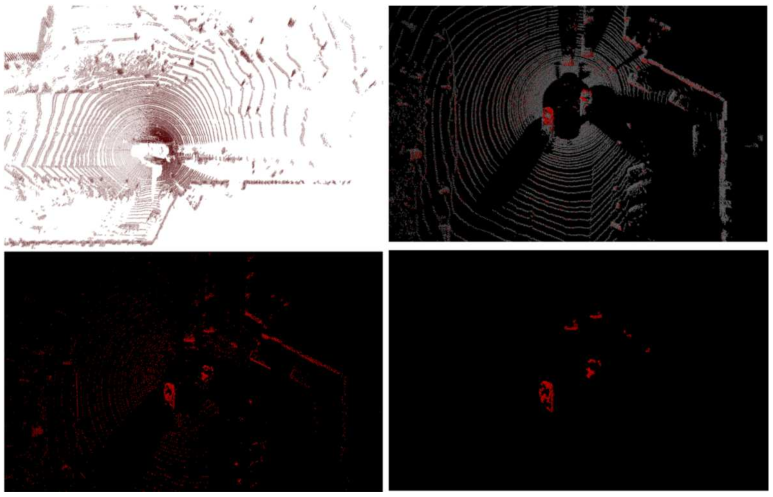


*Figure 2.4: The results of the method proposed by Azim et al. Red points represent change*

idea of keeping a history of dynamic voxels from the last scans, as these voxels do not deterministically identify moving objects. This history allows for a more robust inference of mobility points, i.e. if a voxel is occupied in previous scans and free in at least two next scans, then it is marked as having moving points. The authors also proposed a segmentation of identified moving points into moving objects for their subsequent tracking.

Xiao et al. [71] proposed using raw MLS data comparison (point to point comparison) for change detection in a complex street environment using a "sweep"-like laser scanner. The scan rays were approximately modeled using scanner parameters and the the Dempster-Shafer Theory (DST, explained later in more detail) was used for intra-scan ray fusion and an inter-scan consistency assessment to detect mobility points. This method however will be discussed with more detail in the next section.

## 2.3 Multi-sensor change detection

A certain category of problems where image-based change detection struggles badly is videos with low framerate taken from fast moving and freely moving cameras (6D movement). We could not find a single image-based method which treats this type of problems. In fact methods for dynamic background and moving camera exist (as seen previously), and they are usually fine for high framerate videos and/or slowly moving cameras (small baseline) but cannot handle scenarios such as change detection in an urban environment in a video taken from a moving vehicle. There is need for more complex data, such as RGB with depth information for example, to solve such problems. LiDARs provide a precise interpretation of the world geometry, thus letting us think that they also allow for a better change detection, which is in part true, but LiDARs lack the point density and color information of the cameras. Methods combining these two types of data acquisition technology started emerging lately thanks to reduced cost of the devices and very detailed online datasets such as KITTI [23]. However there is still limited research focusing on street-level change detection with hybrid data acquisition.

Worth mentioning are change detection methods [21, 70, 63] for augmented imagery which started appearing after the introduction of Kinect camera from Microsoft. This camera outputs color videos augmented with depth information, that is RGB-D (Red Green Blue and Depth) images. The depth information comes from an infrared camera. This camera however does not work outdoors and thus all the methods are applicable only indoors. These methods also represent the vast majority of multi-sensor change detection methods.

Qin and Gruen [43] proposed an approach to detect and update changes in urban areas using MLS and terrestrial stereo images, although at different time epochs. Their method consists in roughly four steps. The first step is recording the point clouds with MLS, and their subsequent cleaning and classification by semi-automatic means. The second step consists in semi-automatic registration of the terrestrial images taken at a later epoch, then the point cloud is reprojected onto the images using an ad-hoc z-buffering weighted window. The third step rectifies the images and reprojects them onto each other to check the geometrical consistency between point cloud and images. In the last step a graph cut optimization is run, consisdering the augmented information (color, depth and class information) of each object, to over-segment the data both in point clouds and images. This method however is based on highly accurate, and semi-automatic, registration of both point clouds and images and moreover it does not consider temporal dynamic objects such as passing cars, cyclists or pedestrians.

### 2.3.1 Related work

Vallet et al. [65] published recently a method for detecting changes in point clouds and images. This work is an extension of what Xiao et al. [71] proposed, although detecting changes only in point clouds. Since these two works are very similar to our work, we will summarize them with more

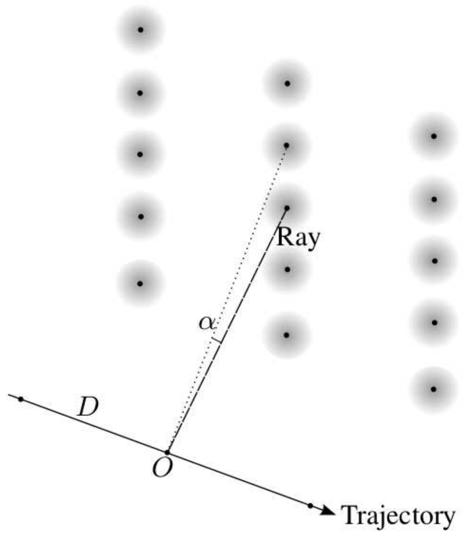


*Figure 2.5: The plane sweeper used by Xiao et al. [71] in their work. The scanning ray rotates on a vertical plane perpendicular to the trajectory*

detail.

Xiao et al. noted the importance of occupancy models in change detection methods. They proposed to model the occupancy space using Dempster-Shafer Theory (DST) and also a point cloud occupancy consistency check method. The space occupancy can be represented by the universal set $X = \{empty, occupied\}$, this way the space reached by a laser scanner ray can be either *empty* or *occupied*, or simultaneously both (*unknown* state) if the space has not been reached yet by the scan.

To use the DST, the following should hold:

$$m : 2^X \to [0,1], m(\emptyset) = 0, \sum_{A \in 2^X} m(A) = 1 \tag{2.6}$$

where $2^X = \{\emptyset, \{empty\}, \{occupied\}, \{empty, occupied\}\}$ is the power set of $X$ and $m$ is the mass of occupancy. The occupancy model is highly dependent on the data acquisition, which was performed by a RIEGL LMSQ120i lidar, which is a plane sweep scanner (see Figure 2.5. By denoting $m(\{empty\})$, $m(\{occupied\})$ and $m(\{unknown\})$ with $e$, $o$ and $u$ respectively, the overall occupancy of point Q at location P is given by:

$$m(P,Q) = \left\{ \begin{array}{c} e \\ o \\ u \end{array} \right\} = \left\{ \begin{array}{c} f_\theta \cdot f_t \cdot e_r \\ f_\theta \cdot f_t \cdot o_r \\ 1 - e - o \end{array} \right\} \tag{2.7}$$

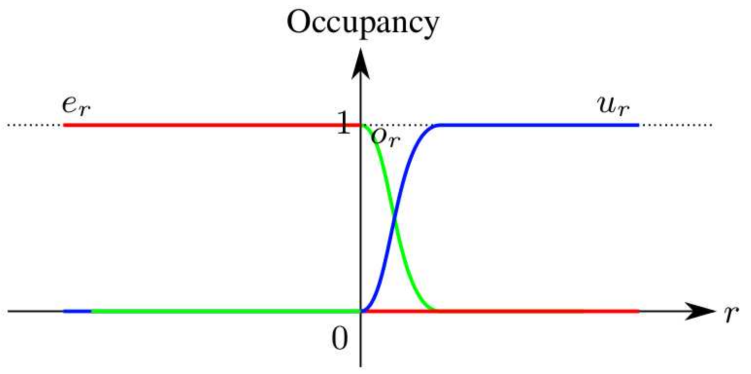


*Figure 2.6: Occupancy in ray direction*

where $f_\theta$ and $f_t$ are the occupancy in the rotation and trajectory directions respectively and are given by:

$$f_\theta = e^{-\frac{1}{2}(\frac{\theta}{\lambda_\theta})^2} \tag{2.8}$$

$$f_t = e^{-\frac{1}{2}(\frac{t}{\lambda_t})^2} \tag{2.9}$$

where $\lambda_\theta$ and $\lambda_t$ are parameters based on angular resolution and range accuracy of the sensor. The variables $t$ and $\theta$ are defined by:

$$\theta = arccos \frac{\overrightarrow{O_tQ} \cdot \overrightarrow{O_tP'}}{||\overrightarrow{O_tQ}|| \cdot ||\overrightarrow{O_tP'}||} \tag{2.10}$$

$$t = \overrightarrow{O_tP} \cdot norm_{traj} \tag{2.11}$$

where $O_t$ is the sensor position, $P'$ is the projection of $P$ on plane perpendicular to the trajectory and $norm_{traj}$ is the trajectory direction. The masses of occupancy $e_r$, $o_r$ and $u_r$ depend on $r = ||\overrightarrow{O_tP''}|| - ||\overrightarrow{O_tQ}||$ (where $P''$ is the projection of $P$ on $\overrightarrow{O_tQ}$) and are best described by Figure 2.6.

As noted before, the parameters and formulae are heavily based on the model of the sensor and its working principle. The occupancy functions of $m(P, Q)$ are then convolved with uncertainties modeled by normal distributions, which yields the following occupancy $m'(P, Q)$:

$$m'(P, Q) = \begin{Bmatrix} e' \\ o' \\ u' \end{Bmatrix} = \begin{Bmatrix} f'_\theta \cdot f'_t \cdot e'_r \\ f'_\theta \cdot f'_t \cdot o'_r \\ 1 - e' - o' \end{Bmatrix} \tag{2.12}$$

obtained from the following:

$$g(m) = \mathcal{N}(0, \sigma_m^2);\ \ g(r) = \mathcal{N}(0, \sigma_r^2) \tag{2.13}$$

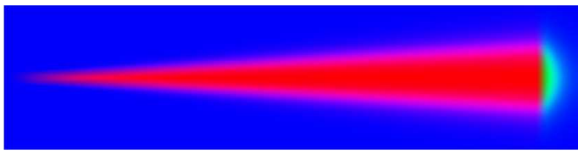


*Figure 2.7: The resultant ray. The red, green and blue represent empty, occupied and unknown respectively*

$$F = g(m) \otimes g(r); \tag{2.14}$$

$$e'_r = e_r \otimes F;\ o'_r = o_r \otimes F \tag{2.15}$$

$$f'_\theta = f_\theta \otimes \mathcal{N}(r, \sigma_\theta^2) \tag{2.16}$$

$$f'_t = f_t \otimes F \tag{2.17}$$

where $\sigma_m$, $\sigma_r$ and $sigma_\theta$ are measurement, registration and angle uncertainty parameters, respectively. The $\otimes$ represent the convolution operation. A visual example of the resultant modeled occupancy space for a scan ray can be seen in Figure 2.7.

To aggregate the occupancy from all scan rays in a scan, the DST is used, in the following way:

$$\begin{Bmatrix} e_1 \\ o_1 \\ u_1 \end{Bmatrix} \oplus \begin{Bmatrix} e_2 \\ o_2 \\ u_2 \end{Bmatrix} = \frac{1}{1-K} \begin{Bmatrix} e_1 \cdot e_2 + e_1 \cdot u_2 + u_1 \cdot e_2 \\ o_1 \cdot o_2 + o_1 \cdot u_2 + u_1 \cdot o_2 \\ u_1 \cdot u_2 \end{Bmatrix} \tag{2.18}$$

that is commutative and associative, $K = o_1 \cdot e_2 + e_1 \cdot o_2$ is the conflict and indicates incoherence in the aggregation of the above values, $\oplus$ is the fusion operator23. The overall occupancy at location $P$ with $I$ number of neighbouring rays $Q_i$ is then given by:

$$m(P) = \bigoplus_{i \in I} m(P, Q_i). \tag{2.19}$$

Xiao et al. then introduced a method to obtain changes in two datasets by comparing their occupancies. The idea is the following: given two datasets and their respective occupancy values ($E_1$, $O_1$, $U_1$) and ($E_2$, $O_2$,

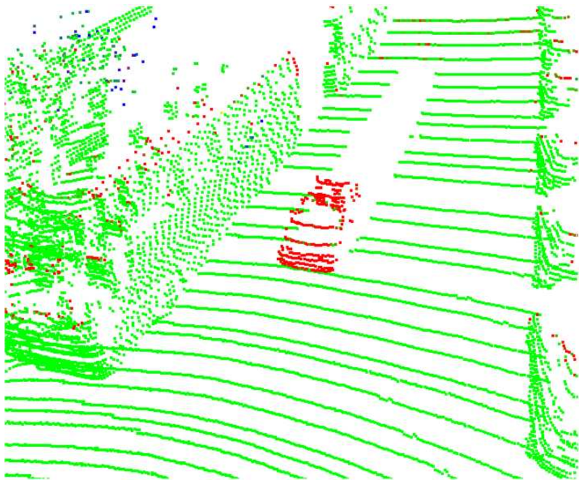


*Figure 2.8: The result of point cloud change detection. Red points are mobility points*

$U_2$), the change is defined where the datasets are not consistent, which is given by the following formulae:

$$\begin{aligned} Conf &= E_1 \cdot O_2 + O_1 \cdot E_2 \\ Cons &= E_1 \cdot E_2 + O_1 \cdot O_2 + U_1 \cdot U_2 \\ Unc &= U_1 \cdot (E_2 + O_2) + U_2 \cdot (E_1 + O_1) \end{aligned} \tag{2.20}$$

$Conf$ means conflicting, $Cons$ is consistent and $Unc$ uncertain. The change regions are those where $Conf > Cons$ and $Conf > Unc$. This method still outputs many false positives, which are stationary points marked as mobility ones, and in order to get as few as possible of those, Xiao et al. gave more weight to the consistent evidence of points.

Vallet et al. [65] took over and extended the Xiao's work to another LiDAR type and brought the mobility points to images. They used data captured from Velodyne HDL-64E, which captures a full 360° scan at 10Hz. They however proposed for consistency check to use a number of previous and next scans as one dataset, and consider the current scan as the other dataset, then testing the consistency between these two datasets. The result of point cloud change detection can be seen in Figure 2.8.

The contribution of their work stands in projecting the mobility points into the images and using graph cut optimization to delineate the moving objects in images. Unlike Velodyne's fixed capture rate, the images were taken every 5m. The points, considering the time awareness, are projected into the closest (temporally) image. First a weights image is created by points projection in the following way:

- (1, 0, 0) - pixels where mobility points are projected (red)

- (0, 1, 0) - pixels where stationary points are projected (green)
- (0, 0, 1) - pixels where there are no points projected (blue)

Then this image is used to create the labeling of corresponding pixels: $L : I \to \{0, 1\}$ where $I$ is the set of pixels of the image. A pixel is labeled 1 as mobile pixel if its corresponding weight pixel is (1, 0, 0) and it is labeled 0 if its corresponding weight pixel is (0, 1, 0). If however the corresponding weight pixel is (0, 0, 1) then the pixel label can be either 0 or 1 indifferently. After the initial labeling, the the graph-cut segmentation consists in minimizing the following energy function:

$$E(L) = \sum_{p \in I} sL(p) + m(1 - L(p)) + \sum_{(p,q) \in N_4} \frac{\lambda |L(p) - L(q)|}{d(I(p), I(q)) + \epsilon} \tag{2.21}$$

where $N_4$ is the set of adjacent pairs of pixels in image $I$ in 4-connectivity, $\lambda$ and $\epsilon$ are smoothness and similarity parameters, $s$ and $m$ are weights of edges between node/source and node/sink respectively, and $d$ is the normalized euclidean distance of image colors:

$$d(I(p), I(q)) = \frac{\sqrt{(r_q - r_p)^2 + (g_q - g_p)^2 + (b_q - b_p)^2}}{255} \tag{2.22}$$

where $r$,$g$ and $b$ are the red, green and blue components. The minimization problem is a graph cut problem where each pixel is a node and two types of edges:

1. Edges between source/sink and a node, with weights $s$ between the node and the source, and $m$ between the node and the sink.
2. Edges between nodes corresponding to adjacent pixels $p$ and $q$ with weights $(d(I(p), I(q)) + \epsilon)^{-1}$.

The result of graph-cut segmentation can be seen in Figure 2.9. The authors however do not provide any useful result metric, and their method present many false positives in the point cloud change detection step as discussed in the next chapters.

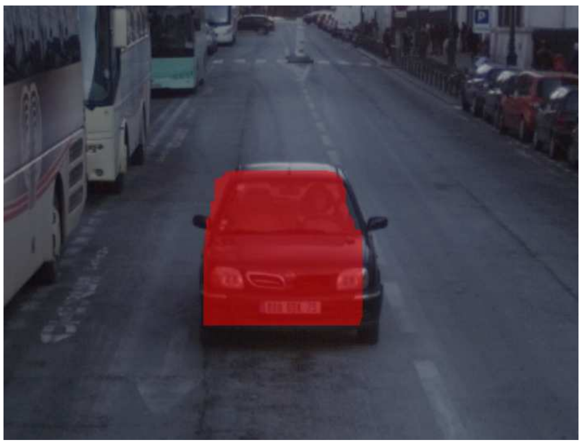


*Figure 2.9: The result of applying graph-cut segmentation*

# Chapter 3

# Background materials

## 3.1 Data acquisition

In this section a brief description and overview of the working principles of the sensors used is given. The data used in this work is taken from the Kitti dataset [23], more specifically the range measurements, from Velodyne device, and the color images from one of the four cameras is used. The Kitti sensors setup on vehicle can be seen in Figure 3.1.

### 3.1.1 LiDAR

LiDAR stands for Light Detection And Ranging (but also lidar + radar) and it originates from the early 1960s, shortly after the invention of lasers. The basic concept of most lidars is to first emit a light pulse, typically using a laser diode. The light travels until it hits a target, when a portion of the light energy is reflected back towards the emitter. A detector mounted near the emitter detects this return signal, and the time difference between the emitted and detected pulse determines the distance of the target.

When this pulsing distance measurement system is somehow actuated, a multitude of points (called a "point cloud") can be collected. If no targets are present, then the light would never return. When this collection of points is rendered, the point cloud begins to resemble a picture. The denser the point cloud the richer the picture becomes.

In the dataset used in this work, the range data was acquired by a Velodyne HDL-64E lidar (see Figure 3.3). The HDL-64E features 64 emitter-detector pairs (thus 64 lasers), each aligned to provide an equally spaced 26.5° elevation field of view, spanning from $+2°$ to $-24.5°$ with a vertical resolution of about 0.4°. The entire optical assembly rotates at 5-15Hz

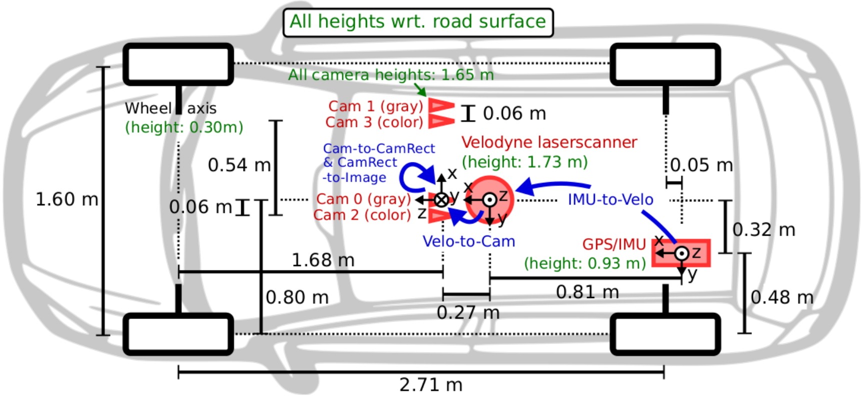


Figure 3.1: KITTI dataset: Sensors setup on vehicle.

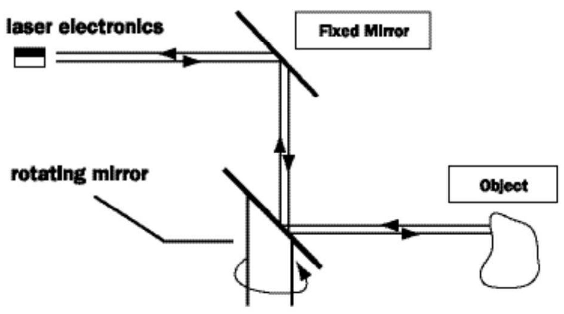


Figure 3.2: Single emitter/detector pair rotating mirror lidar design.

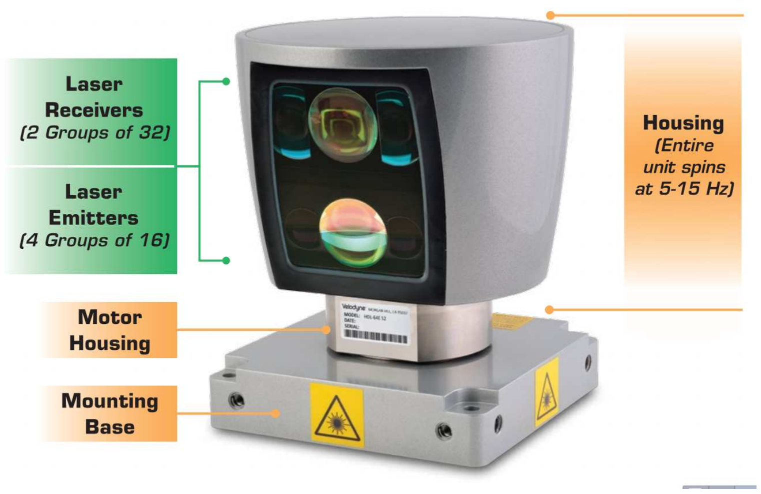


Figure 3.3: The Velodyne High Definition Lidar (HDL) 64E.

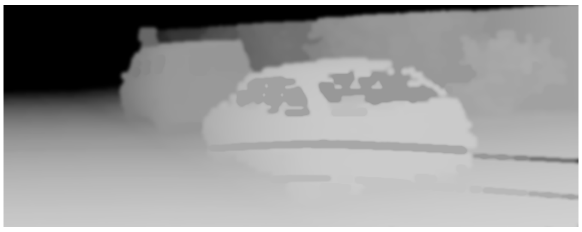


*Figure 3.4: Depth image rendered from HDL-64E data. Note the visible gap between the two laser banks.*

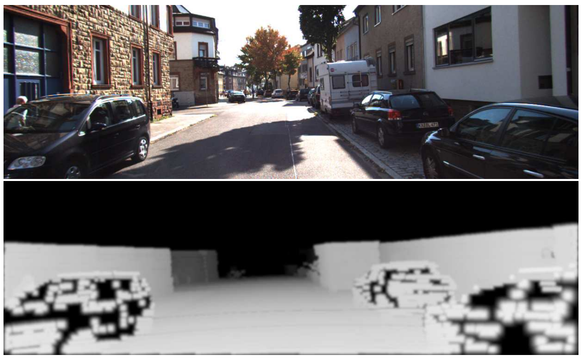


*Figure 3.5: Problem in acquiring range distances on dark materials. Note the corresponding missing depths on black cars.*

(10Hz by default) to provide a 360° azimuth field of view with a horizontal resolution of about 0.08°. The unit has a range of 120 m and typically has a distance error of less than 2.5 cm. Each laser is a Class 1 (eye safe) 905 nm Infra-Red (IR) laser diode that emits a short charged pulse for a duration of 5 ns and a power peak of 60 W that reflects off an object with less power or is absorbed by very dark materials. The reflected laser is first filtered by a UV sunlight filter to reduce noise and then it is digitized and processed by a 3GHz ADC and 3GHz DSP. Over 1.3 million data points each second are transmitted over the Ethernet interface.

The emitter-detector pairs are divided into two 32-laser banks with a visible gap between the two, as can be seen in Figure 3.4. The HDL-64E like most other laser range sensing devices struggles with low reflective materials (mostly very dark ones) and high reflectance transparent objects such as car windows as can be seen in Figure 3.5.

### 3.1.2 Camera

A camera is a mapping between the 3D world (object space) and a 2D image. The mapping is usually done by central projection, that is a ray from the point in 3D space is drawn through a fixed point in space, called *center of projection*, to intersect a specific plane in space called *image plane*.

This model is in accord with a simple model of a camera, in which a ray of light from a point in the world passes through the lens of a camera and impinges on a film or digital device, producing an image of the point. Ignoring such effects as focus and lens thickness, a reasonable approximation is that all the rays pass through a single point, the centre of the lens.

The mapping is expressed by the following formulation:

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = P_{3x4} \begin{pmatrix} X \\ Y \\ Z \\ T \end{pmatrix} \tag{3.1}$$

where $P_{3x4}$ is the camera matrix and will be discussed later. Note the homogeneous coordinates are used, this is because of the nature of projective geometry (which works with homogeneous coordinates) and, since all the incident rays on the same image pixel can have arbitrary $T$, in most cases it is better to set $T = 1$. It is straightforward to transform homogeneous coordinates to Cartesian ones.

In projective camera models the center of projection is called *camera center* (or *focal point*), the line from the camera center perpendicular to the image plane is called the *principal axis* (or *principal ray*) of the camera, and the point where the principal axis meets the image plane is called the *principal point*. To better explain the camera matrix, the *pinhole camera* model is described below, which is the most basic projective camera model. For a better understanding of the pinhole camera geometry see the Figure 3.6. Assuming the camera center to be the origin of Euclidean coordinate system, and the image plane $Z = f$ (also called *focal plane*), a point in space $\mathbf{X} = (X, Y, Z)^T$ is mapped to an image point $\mathbf{x} = (x, y)$ in the following way:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.2}$$

and the image point $(x, y)$ in non-homogeneous coordinates is given by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \end{pmatrix} \tag{3.3}$$

where $(p_x, p_y)$ are the coordinates of the principal points. The projection matrix used in (3.2) is called *camera calibration* (or *intrinsics*) matrix (usually denoted by K). The The concise form of the formula (3.2) is: $\mathbf{x} = \mathrm{K}[\mathrm{I}|0]\mathbf{X}_{cam}$ if the camera is located at the origin of Euclidean coordinate system (this is why $\mathbf{X}_{cam}$ is used). This system is called *camera coordinate system* (see Figure 3.7).

When the camera center is at the origin of the world coordinate system, the following formula applies:

$$\mathbf{x} = \mathrm{KR}[\mathrm{I}| - \mathbf{C}]\mathbf{X} \tag{3.4}$$

where R, $\mathbf{C}$ are the camera rotation and camera center position in the world coordinate system. Now the matrix $\mathrm{R}[\mathrm{I}| - \mathbf{C}]$ is called *camera extrinsics* matrix. The camera matrix $P$ is then the both intrisics and extrinsics camera matrices combined: $P = \mathrm{KR}[\mathrm{I}| - \mathbf{C}]$. There is another way of representing the camera matrix: $P = \mathrm{K}[\mathrm{R}|\mathbf{t}]$ where $\mathbf{t} = -\mathrm{R}\mathbf{C}$.

The pinhole camera model can be extended to take into account various other factors such as distortions and abberation. Considering the $\mathrm{R}, \mathbf{t}$ introduced above, the following formulation are designed to deal with distortions:

$$\begin{pmatrix} x_0 \\ y_0 \\ z \end{pmatrix} = \mathrm{R} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{t} \tag{3.5}$$

$$x_1 = x_0/z \tag{3.6}$$

$$y_1 = y_0/z \tag{3.7}$$

$$r^2 = x_1^2 + y_1^2 \tag{3.8}$$

$$x_2 = x_1 \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 x_1 y_1 + p2(r^2 + 2x_1^2) \tag{3.9}$$

$$y_2 = y_1 \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_2 x_1 y_1 + p1(r^2 + 2y_1^2) \tag{3.10}$$

$$x = f x_2 + p_x \tag{3.11}$$

$$y = f y_2 + p_y \tag{3.12}$$

where $k_1, k_2, k_3, k_4, k_5$ and $k_6$ are radial distortion coefficients, $p_1$ and $p_2$ are tangential coefficients.

In the dataset used in this work, four cameras were used (2 grayscale and 2 color) to capture the scene in front of the vehicle. The two by two pairing was designed to allow easier stereo vision methods testing with real world data. The camera used in this work is the color left one (Cam 2 in Figure 3.1). It captures 1.4MP ($1384 \times 1032$) resolution videos at variable framerate (15FPS by default) on a $1/2''$ ($4.65\mu$m pixel size) CCD sensor (Sony ICX267).

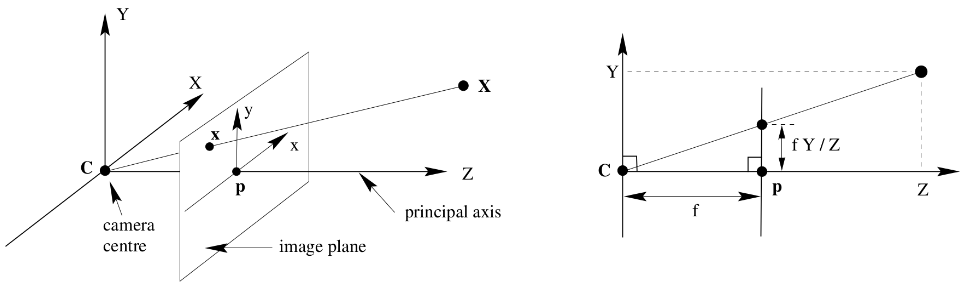


*Figure 3.6: Pinhole camera geometry.* **C** *is the camera center and* **p** *the principal point. The camera center is here placed at the coordinate origin. Note the image plane is placed in front of the camera center.*

The camera has been synchronized with the Velodyne lidar and the GPS/IMU (Inertial Measurements Unit) at 10Hz. The frames have been rectified (removed distortions) and have a final resolution of $1242 \times 375$.

## 3.2 Dempster-Shafer Theory

Dempster-Shafer Theory (DST) is a mathematical theory of evidence (or theory of belief functions) introduced by Shafer [53] based on the seminal work of Dempster [18]. In a finite discrete space, Dempster-Shafer theory can be interpreted as a generalization of probability theory where probabilities are assigned to sets as opposed to mutually exclusive singletons.

Often used as a method of sensor fusion, DST is based on two ideas: obtaining *degrees of belief* (also referred to as *masses*) for one question from subjective probabilities for a related question (like sensors measuring the same event), and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence. DST's framework allows for belief about such evidences to be represented as intervals, bounded by two values, belief (or support) and plausibility: $belief \leq plausibility$. Belief and plausibility can be directly computed from belief masses. DST assigns its masses to all of the non-empty subsets of the evidences that compose a system, thus the sum of all masses has to be 1.

The formulation of DST starts with the universal set $X$ of evidences and its power set $2^X$. A belief mass $m$ is assigned to each element of the power set:

$$m : 2^X \rightarrow [0, 1] \tag{3.13}$$

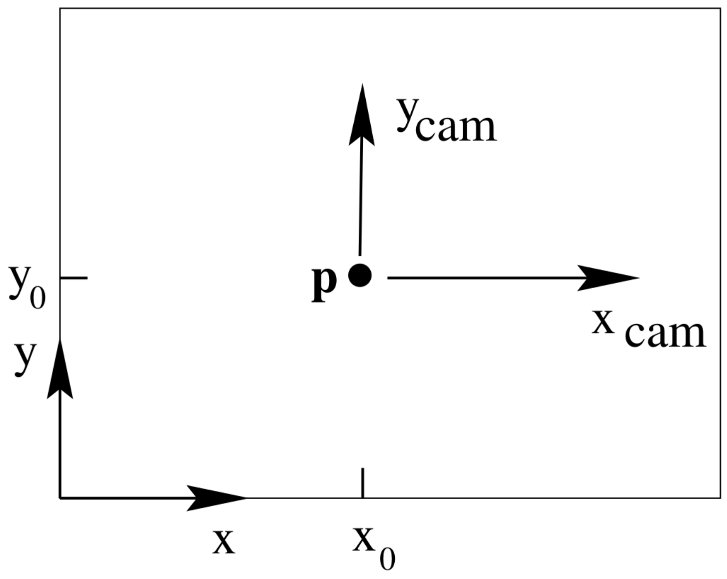


*Figure 3.7: Image $(x, y)$ and camera $(x_{cam}, y_{cam})$ coordinate systems.*

and the following should hold:

$$m(\emptyset) = 0, \qquad \sum_{A \subseteq X} m(A) = 1. \tag{3.14}$$

Belief $bel(A)$ and plausibility $pl(A)$ are lower and upper bounds of probabilty interval for an event, respectively, and are defined as follows:

$$bel(A) = \sum_{B|B \subseteq A} m(B) \tag{3.15}$$

$$pl(A) = \sum_{B|B \cap A = \emptyset} m(B) \tag{3.16}$$

and the relation between belief, plausibility and masses is:

$$pl(A) = 1 - bel(\bar{A}), \quad \bar{A} = X - A \tag{3.17}$$

$$m(A) = \sum_{B|B \subseteq A} (-1)^{|A-B|} bel(B). \tag{3.18}$$

To combine the masses from indipendent sets, the Dempser's rule of combination is applied. This rule is a fusion operator, which is commutative and associative. The combination (called also *joint mass*) is calculated from the two sets of masses $m_1$ and $m_2$ in the following manner:

$$m_{1,2}(A) = (m_1 \otimes m_2)(A) = \frac{1}{1-K} \sum_{B \cap C = A \neq \emptyset} m_1(B) m_2(C) \tag{3.19}$$

where $B \in 2^X$, $C \in 2^X$ and K is the conflict between two mass sets:

$$K = \sum_{B \cap C = \emptyset} m_1(B) m2(C) \tag{3.20}$$

As a final note, Zadeh [73] noted that the normalization factor $1 - K$ can ignore conflict and attribute any mass associated with conflict to the null set. For this reason this combination rule can produce counterintuitive results. Many other combination rules were proposed [51], all with their advantages and disadvantages.

## 3.3 Spatial indexing

In applications where spatial N-dimensional data is used, a fast search and store data structure is crucial. We will briefly describe the **octree** and **kd-tree** data structures as these two are the most popular and most used spatial data structures for 3D geometry.

The octree is a tree data structure in which each node has 8 children (see Figure 3.8). It was proposed by [36] for fast spatial representation and queries of 3D geometry (points and/or shapes). The octree root represents the center of the bounding box that encapsulates the 3D scene (point cloud). Every node can be of two types: *inner node* or *leaf node*. Leaf nodes contain points while inner nodes contain 8 children nodes which can be either leafs or inner nodes. If a leaf node contains a number of points superior to a predefined threshold, it is expanded and transformed into an inner node and all its points are redistributed to its children.

A node, called cell in the octree, is usually a cube. The cell, while still a leaf, contains points and, upon expansion, is equally split into 8 smaller cells, as shown in Figure 3.8, and its points are assigned to its children according to the space they occupy. There have been various node representations to minimize the memory consumption, and one of the most well-known ways is to encode the cell number among its siblings with a 3bit word (indices 0-7) and the children types of an inner node with an 8bit mask (1 for inner node and 0 for leaf). An inner node is cosider to contain all the points encapsulated by its children, recursively.

To perform a spatial search of a point's neighborhood, a tree depth-first search is enough. The maximum octree depth $D_{max}$, given the minimum distance $d_{min}$ between two points and the bounding box side length $s$, is limited by:

$$D_{max} \leq \log \frac{s}{d_{min}} + \log \sqrt{3} \tag{3.21}$$

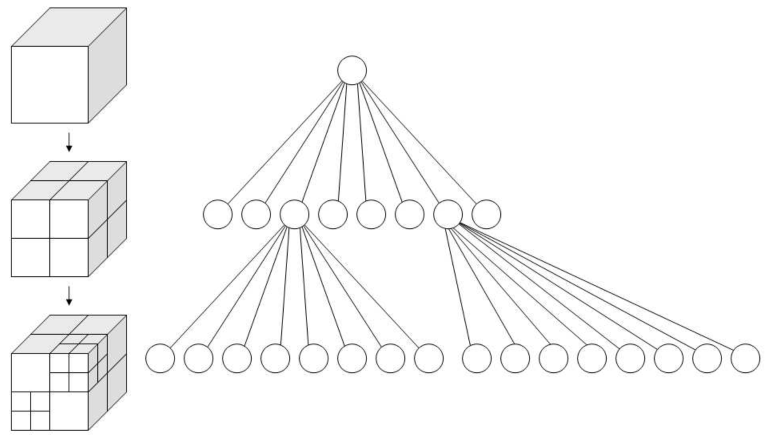


*Figure 3.8: Octree representation. Each node can split into 8 children nodes when minimum points thresholding is reached, and all its points are pushed to its children nodes.*

Another well-know data structure for spatial search is **kd-tree**, which is a binary tree designed for k-dimensional spatial partitioning and nearest neighbor search (for 2D example see Figure 3.9).

The principle of constructing the kd-tree is rather straightforward and can be summarized by the algorithm 1:

**Algorithm 1** KD-tree basic algorithm

```
1: procedure KDTREE($pointList$, $depth$)
2:     $axis := depth \mod k$
3:     $m := median$ from $pointList$ by $axis$
4:     $node :=$ new tree-node
5:     $node.location \leftarrow m$
6:     $node.leftChild \leftarrow$ kdtree($points$ in $pointList$ before $m$, $depth + 1$)
7:     $node.rightChild \leftarrow$ kdtree($points$ in $pointList$ after $m$, $depth + 1$)
8:     return $node$.
```

The above algorithm will lead to a balanced tree, which in general performs better than unbalanced ones (it depends on the application sometimes). A simple insertion or deletion renderes the kd-tree unbalanced. There is another way of constructing a balanced kd-tree: instead of performing the median extraction at every level, presort the points by all their axis using a sorting algorithm. The kdtree creation has a worst case complexity of $O(n \log^2 n)$ if the median extraction is performed by a $O(n \log n)$ algorithm, and $O(kn \log n)$ complexity if presorting is used. There are other

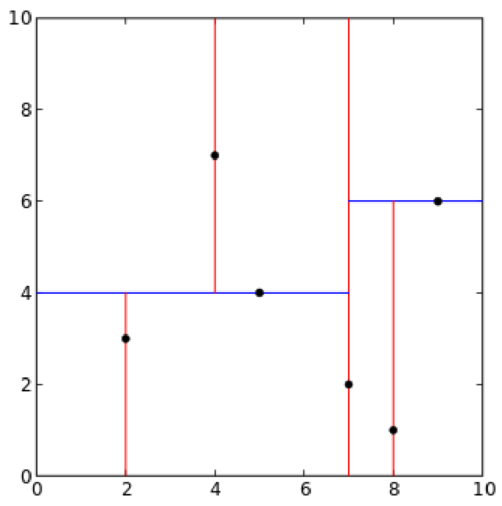


*Figure 3.9: KD-tree representation of a 2D space partitioning. Note the separation along the medians of point axes.*

ways to build the tree without resorting to expensive sorting or median extraction. These methods rely on extracting the mediam from a smaller subset of randomly chosen points, which in the end yields a quasi-balanced tree.

The search operation is performed by traversing the tree in a depth-first fashion until a leaf is reached, which is marked as closest point, and then backtracking and testing other points from close hyperplanes to test if they are closer than the closest point. The search operation has a $O(\log n)$ average time complexity for single neighbor search. This makes the kd-tree very practical for spatial search operation in k-dimentional space.

## 3.4 Point cloud registration

The outcome of the change detection in a point cloud partially depends on the quality of point cloud registration, also known as alignment. This step is however crucial not only to change detection, but to virtually all applications involving 3D geometry (point clouds, meshes, etc.), scanned or computed. The goal of the registration step is to estimate the best point correspondence between two point clouds. The most popular approach is the Iterative Closest Point (ICP) method introduced by [6], on which many other methods were built [34], [50], [52].

### 3.4.1 Iterative Closest Point

The core idea of ICP is to find and minimize the rotation and translation distance that maximizes the overlap between two point clouds in alternating steps. In other simple terms, given two point clouds $P = \{\mathbf{p}_1, ..., \mathbf{p}_r, ..., \mathbf{p}_n\}$ and $Q = \{\mathbf{q}_1, ..., \mathbf{q}_r, ..., \mathbf{q}_m\}$, the rotation $R$ and translation $\mathbf{t}$ are found by minimizing the following error function:

$$E(R, \mathbf{t}) = \frac{1}{r} \sum_{i=1}^{r} ||\mathbf{p}_i - R\mathbf{q}_i - \mathbf{t}||^2 \tag{3.22}$$

where $\mathbf{p}_i$ and $\mathbf{q}_i$ are corresponding points, and $r$ their cardinality. If the correct correspondences are known, then both rotation and translation can be calculated in closed form [26]. Considering the center of mass of both point clouds:

$$\bar{\mathbf{p}} = \frac{1}{r} \sum_{i=1}^{r} \mathbf{p}_i \qquad \bar{\mathbf{q}} = \frac{1}{r} \sum_{i=1}^{r} \mathbf{q}_i \tag{3.23}$$

the rotation $R$ is computed from:

$$W = \sum_{i=1}^{r} (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{q}_i - \bar{\mathbf{q}})^T = USV^T \tag{3.24}$$

and by using singular value decomposition (SVD) and the following:

**Theorem**: If rank$(W) = 3$, the optimal solution of $E(R, \mathbf{t})$ is unique and given by:

$$R = UV^T \qquad \mathbf{t} = \bar{\mathbf{p}} - R\bar{\mathbf{q}}. \tag{3.25}$$

However, the correspondences are almost never known, thus it is impossible to determine the optimal transformation in one step. To overcome this issue the ICP was designed, and its algorithm is as follows:

- Given an initial transformation guess, iterate until convergence:
  - Find corresponding points
  - Solve for $R$, $\mathbf{t}$:

$$(R, \mathbf{t}) = \underset{(R,\mathbf{t})}{\operatorname{argmin}} \sum_{i=1}^{r} ||\mathbf{p}_i - R\mathbf{q}_i - \mathbf{t}||^2. \tag{3.26}$$

The convergence criteria can be many, the most popular are: iterate until a certain transformation epsilon is reached, or until a certain number of iterations is reached. The selection of sampling points can be done exhaustively, by random sampling, by spatially uniform sub-sampling or by feature-based sampling (this one requires pre-processing). The point correspondence search can be done in the following ways:

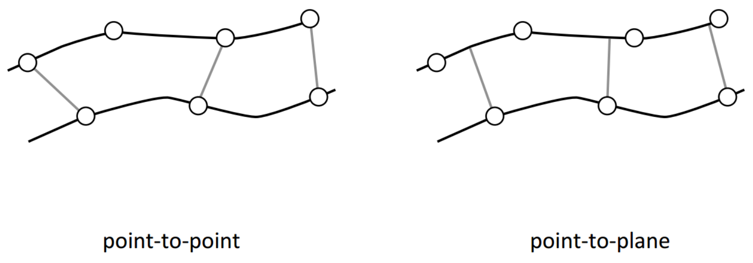


*Figure 3.10: The two most known error metrics of ICP*

- by searching the closest point in the other point cloud within a distance threshold, which usually yields good results but is slow.
- searching for the nearest neighbors in the other point cloud using a search data structure such as KD-tree or Octree, but it usually takes time to build those data structures.
- by using projection-based matching, which is orders of magnitude faster than closest point search but requires specific error metric such as point-to-plane error metric (discussed later).

There are also various error metrics to consider for ICP. The original error metric is point-to-point metric, and it is the one shown before, where the distance between corrisponding points is considered. An error metric worth noting is point-to-plane error metric.

This metric extension was introduced as a more robust variant of ICP. It considers the distance of the point to the closest plane of corresponding points in the other point cloud (see Figure 3.10). In other terms it lets flat regions to slide along each other. Its formulation is as follows:

$$E(R, \mathbf{t}) = \frac{1}{n} \sum_{i=1}^{n} ||\eta_i \cdot (\mathbf{p}_i - R\mathbf{q}_i - \mathbf{t})||^2 \tag{3.27}$$

where $\eta_i$ is the surface normal at $\mathbf{p}_i$. However to solve the minimization, a non-linear minimization technique is required.

Another ICP variant is Generalized Iterative Closest Point (GICP) and it was introduced by Segal et al. [50] as an extension to ICP. Its idea is to assign individual covariance to each data point, computed from its neighborhood. This covariance is enforced to have a disk shape and to lie on the surface from where the point was sampled. The error metric then considers the Mahalanobis distance of these covariances (planar patches) instead of points.

This method is proven to be more robust than the point-to-plane variant of ICP, but it is computationally more expensive even though less iteration steps are required for convergence.

One of the most recent works was presented by Serafin and Grisetti [52] as yet another ICP variant named Normal Iterative Closest Point. The authors noted that an increase in the dimensionality of the points makes the whole system more observable and thus less correspondences are required for the optimization process. For this reason they proposed to augment the point data in the ICP formulation with normals for quasi-planar regions and/or tangents for regions of hig curvature. The authors claim their method outperformes the GICP method in robustness and speed.

We tried the algorithm given by Serafin (a big thank you to him) and while it was faster than GICP, it couldn't match the GICP robustness. The problem lies in the design of the algorithm, as for speed it operates on depth maps, but depth maps rendered from Velodyne point clouds are sparse and thus unsuitable for the algorithm.

### 3.4.2 Normal Distribution Transform

Normal Distribution Transform was proposed by Biber and Straßer [8] for 2D environment and later by Magnusson [35] for 3D environment. The approach is very similar to ICP, but instead of comparing directly the points, NDT method compares the point with its local statistics. The main drawback of ICP is the assumption that the points on surfaces in two point clouds are exactly the same. For this reason in the NDT method, the surface is approximated with a set of Gaussians capturing the local statistics of the surface in the neighborhood of a point. The correspondence is done using Mahalonobis distance. Its formulation is as follows:

$$(R, \mathbf{t}) = \operatorname*{argmin}_{(R,\mathbf{t})} \sum_{i=1}^{r} (\mu_i - R\mathbf{q}_i - \mathbf{t})^T \Sigma_i^{-1} (\mu_i - R\mathbf{q}_i - \mathbf{t}). \tag{3.28}$$

where $\mu_i$ and $\Sigma_i$ are the mean and covariance matrix of the local statistics of point $\mathbf{p}_i$ respectively.

# Chapter 4

# Fast Change Detection

In this chapter we propose an algorithm to detect changes in point clouds acquired by a lidar device. This algorithm tries to minimize typical 3D change detection issues found in [65] and [3], such as over detecting stationary objects or missing slow moving objects, and detect changes in a rapid way by exploiting fast spatial indexing data structures. We imposed three goals for this algorithm: speed, accuracy and robustness. In the following chapters we will show our system pipeline and describe the algorithm in detail.

## 4.1 System Pipeline

We have developed a system which takes images acquired by cameras and point cloud data acquired by a lidar device as input and outputs a resampled point cloud and a set of change masks. An overview of the system pipeline can be seen in Figure 4.1. We call every block a module, which is a unit responsible for one specific task and can be exchanged with any other module which performs the same task.

The system can be seen as two separate subsystems, one for point cloud and the other for change masks. The first subsystem is self-sustained, that is, it is indipendent from the other, while the latter is heavily dependent on the first one. To better explain these dependencies, a brief explanation follows. We will first start from the first subsystem. The input point cloud is aligned to the last input point cloud by a point cloud registration algorithm. The aligned point cloud is then pre-processed and pushed forward to the point cloud change detector and depth map extractor. The change detector is very complex, and will be described later in greater detail. Once the changes are detected and removed, the now static point cloud is pushed to the image-based change detector and to the mapping module, where a point cloud

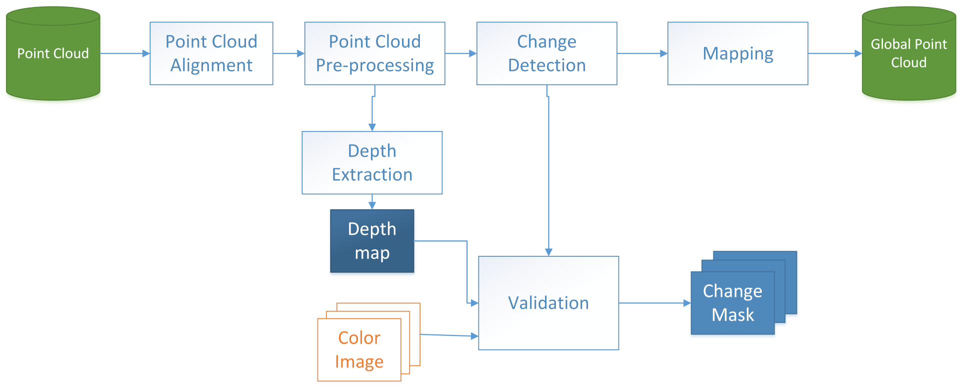


*Figure 4.1: System pipeline.*

is added to the global point cloud. The last module, once finished can be instructed to perform a point cloud downsampling.

The second subsystem relies on both images and point clouds. It first generates dense depth maps (depth images), although very approximate, from the point clouds. The depth maps, the color images from camera and the point clouds from first subsystem are then used in the image-based change detection module. This module generates change masks (foreground masks) by using these three types of information, although the depth maps information can be dropped in favor of performance.

A brief description of each module is given below:

**Point Cloud Alignment** This module has the task of aligning the input cloud with the previously aligned point cloud, using a registration algorithm, such as GICP, and GPS/IMU as additional information.

**Point Cloud Pre-processing** This module "shrinks" the point cloud, that is, only points with a distance from the cloud origin below a threshold are kept. This is done to speed up computation and has a minimal impact on the quality, since distant points are less dense and less precise, and thus provide less information than close points and can also bring misleading information (points under the street).

**Point Cloud Change Detection** This module is responsible for detecting changes in the world measured by the lidar. This is the core of our system, and since it is complex, it will be discussed in following section.

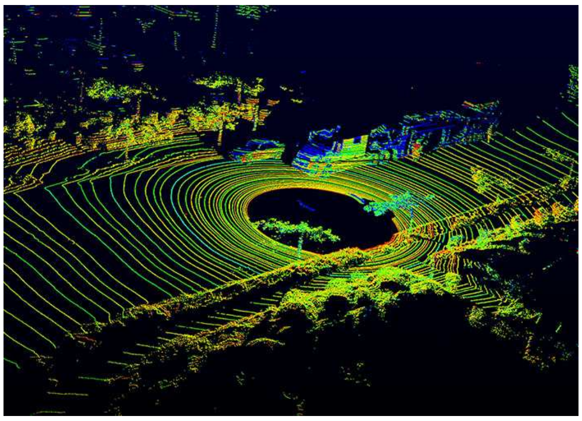


*Figure 4.2: Velodyne points. Note the points on the ground are in circles.*

**Point Cloud Mapping** Here the input point cloud without mobility points is added to the global point cloud using a fusion algorithm. This module can be triggered to perform a decimation of the global point cloud using a downsampling technique which considers the points' curvatures.

**Depth Map Extraction** This module creates dense depth maps from point clouds. These depth maps are very approximate and their usefulness will be explained in the next chapter.

**Image Based Detected Changes Validation** Here the change masks are created using a set of color images, depth maps and mobility points from point clouds. This module is also the core of our system and will be explained in greater detail in the next chapter.

### 4.1.1 Point cloud alignment

Since the point cloud alignment is the crucial part of the system, the selection of the registration algorithm is very important. We have tested a variety of algorithms including ICP, ICP with point-to-plane error metric, GICP, NDT and NICP. In the Figure 4.2 a velodyne point cloud is presented, where one can see the ground points are in circles. This point arrangement complicates a lot the search for corresponding pairs, since two point clouds have their ground points in circles with different centers.

This representation is not suited for ICP, where the correct correspondences are needed. In fact we have tested the ICP and it failed considerably. We have also tested the ICP without considering the ground points, but it also failed, althought with a smaller error. It failed mostly on aligning the point cloud on vertical axis, since the ground points represent the most part of the set of horizonzontal planes.

The NICP was also tested. It was fast but still imprecise. This is due to the nature of the algorithm, it works with depth maps for a faster execution, but the velodyne point clouds can generate a rather sparse depth map. The dense depth maps from the depth map extration is not precise, and cannot be considered for this algorithm.

The NDT proved to be very slow and somewhere imprecise. Because of its slow execution it was not a good candidate for our system.

The ICP with point-to-plane metric had the performance similar to that of NICP but it was more precise. In fact this algorithm can be used in the system whenever speed is the priority. We will later show that the registration step is the bottleneck of the system. Due to its design, where the distance between points and planes is considered, this algorithm fails when there are many non planar surfaces such as bushes, trees, corners, etc.

The GICP proved to be the most robust of all, although slower then ICP with point-to-plane metric but faster than NDT. It outperformed all other considered algorithms in precision, and for this reason it was selected as the registration algorithm for the system. It is to be noted however, that this algorithm can fail periodically whenever there is a significant rotation with many non-planar points.

### 4.1.2 Point cloud pre-processing and mapping

The pre-processing step is straightforward and consists in removing points which have a distance from point cloud center greater than a given threshold, along each axis. This method is called cropbox and, as it name suggests, it "crops" a box out of the point cloud, thus the output point cloud has a rectangular form.

The mapping step adds a point cloud to the global point cloud created as an aggregation of previous point clouds. A simple addition of points is not a good idea, since the majority of points from one point cloud is close to the points from the other point cloud. By just adding points of all point clouds leads to a very dense global point cloud, with many duplicate points. To avoid this issue the input point cloud is confronted with the last few aggregated point clouds and the input points are added only if there are no close points from those aggregated point clouds. This fills the gaps in point clouds and renders the global cloud free of duplicates. This idea is summarized in the algorithm 2.

BBox(...) procedure creates a bounding box given the center and side

**Algorithm 2** Point cloud mapping

```
1: procedure MapCloud($inputCloud$, $globalCloud$, $distThreshold$)
2:     $lastPoints := globalCloud.lastPoints$
3:     for all $point_i$ in $inputCloud$ do
4:         $closestPoint :=$ findClosest($point_i$, $lastPoints$)
5:         if distance($point_i$, $closestPoint$) $> distThreshold$ then
6:             $globalCloud.points \leftarrow globalCloud.points \bigcup point_i$
7:         end if
8:     end for
9:     $cloudDistance :=$ distance($inputCloud.origin$, $globalCloud.origin$)
10:    $sideL := inputCloud.boundingBox.sideLength$
11:    $boundingBox :=$ BBox($inputCloud.origin$, $sideL + cloudDistance$)
12:    $boxPoints :=$ cropBox($globalCloud.points$, $boundingBox$)
13:    $globalCloud.lastPoints \leftarrow$ kdtree($boxPoints$)
14:    return $node.$
```

length, cropBox(...) returns the points contained in the specified bounding box from the specified cloud, and kdtree(...) creates a kd-tree from the specified points.

## 4.2 Point cloud change detection

The change detector proposed in this section is the core part of the system and it is the main contribution of this work. It borrows some ideas from the work done by Xiao et al. [71] and Vallet et al. [65], such as the Dempster-Shafer Theory for occupancy space representation and Dempster combination rule for intra-scan evidence fusion from the first, and the idea of using previous and future scans for change detection from the latter. Aside from those similarities, the algorithm diverges quite a lot from the cited ones. Xiao was kind enough to send us their algorithm, for which we are very thankful to him. The algorithm is however designed for sweeping plane lidar, and Vallet et al. [65] extended it to Velodyne data by adding the previous/future scans in the algorithm, and they did not specify any additional modifications. For this reason we extended the Xiao algorithm to Velodyne data by ourselves considering the previous/future scans, just as Vallet et al. did.

The algorithm proposed by Vallet et al. [65] considers many points on the ground as false positives (stationary points marked as mobile), as seen in Figure 4.3. This occurs because of velodyne points distribution as concentric

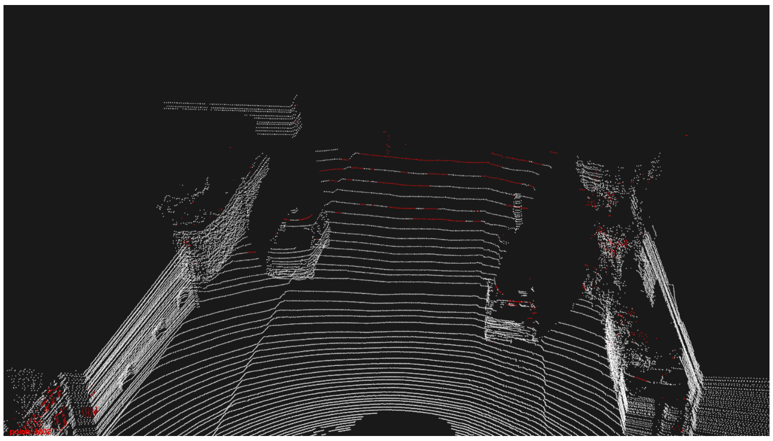


*Figure 4.3: Ground points marked as non-stationary points.*

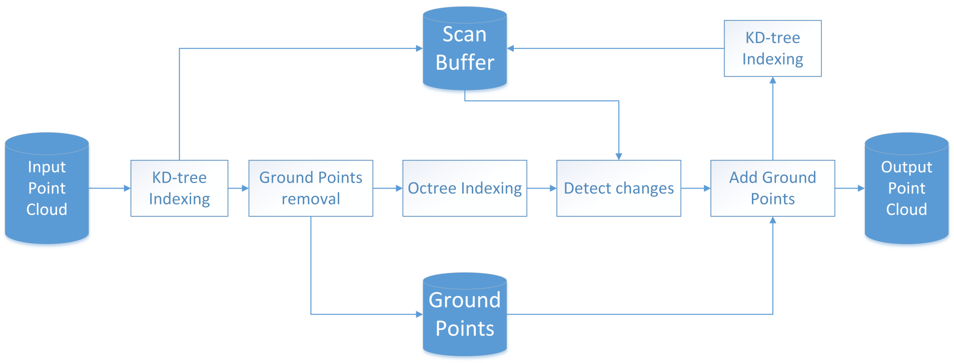


*Figure 4.4: Change detection algorithm scheme.*

circles with non-uniform growing radii. Since the algorithm works on point neighborhoods, the ever increasing radius leads to fewer neighbors per point, thus less evidence per point. For this reason our algorithm removes the ground points before doing any change detection checks. The cited algorithm uses a weighting mechanism to keep as few false positives as possible, but it struggles with far away points (for all scans) situated on planar surfaces such as walls perpendicular to the trajectory of the vehicle. We use a different, yet simpler and faster weighting which solves, in most part, the issue described before. Finally the speed of the cited algorithm is quite low, this is due to the very many checks and computation done per each point. We propose to avoid such costly computation by indexing the point cloud with an octree data structure and performing checks on a small set of points per leaf node. An overview of the algorithm is given in Figure 4.4.

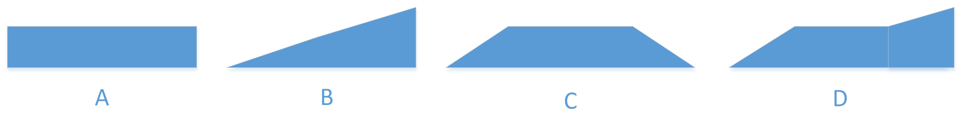


*Figure 4.5: Various types of slopes. The slope type A and type B is indistinguishable by the Velodyne. The ground on slopes of type C and D cannot be extracted by a plane segmentation*

### 4.2.1 Ground points removal

Ground points are always stationary, so they should not be checked for change detection. A naive way to eliminate the ground points is to remove all points which are at a certain negative height from the sensor. This will consider all points below that height as ground points, which is not always the case (e.g. cars on a negative slope). Another idea is to use plane segmentation to extract the ground points as a horizontal plane. This will work when all ground points lie on a plane, which is not the case where slopes are present, as seen in Figure 4.5. Therefore we propose a novel idea to eliminate the unnecessary ground points.

The idea is loosely based on Markov Random Fields and belief propagation. First we divide the point cloud in a 2D grid based on XY plane (Velodyne data coordinate system is: $X$: forward, $Y$: left and $Z$: up). Then starting from the tile at the origin of the point cloud, the ground height is propagated to all other cells in the following way:

- if a cell is marked as ground cell and its height is below the maximum received height from other cells, then use its height and propagate it to other cells.
- if a cell is not ground cell or is empty, then propagate the maximum ground height received from other cells.

The propagation is done from the cells of inner rings to the cells of outer rings, where a ring consists of cells having the same distance from the center. A propagation representation can be seen in Figure 4.6.

If we consider the slope $s$, every point height $P_{z,ij}$ of a cell $C_{ij}$, and the propagated ground height $H_G$, the maximum height $H_{ij}$ and minimum height $h_{ij}$ of a the cell given by:

$$H_G = \underset{H_k}{\text{argmax}}(H_k | k \in \{(i-1,j),(i,j-1),(i-1,j-1)\}) \tag{4.1}$$

$$H_{ij} = \text{argmax}\ (P_{z,ij} | P_{ij} \in C_{ij}) \tag{4.2}$$

$$h_{ij} = \text{argmin}\ (P_{z,ij} | P_{ij} \in C_{ij}) \tag{4.3}$$

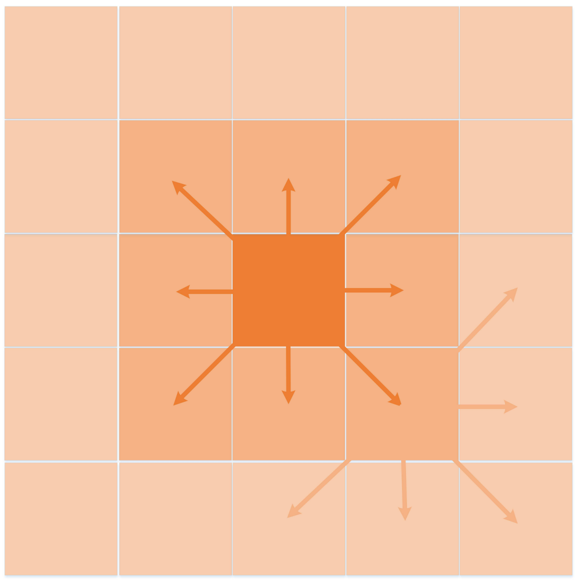


*Figure 4.6: The propagation of the ground height through cells.*

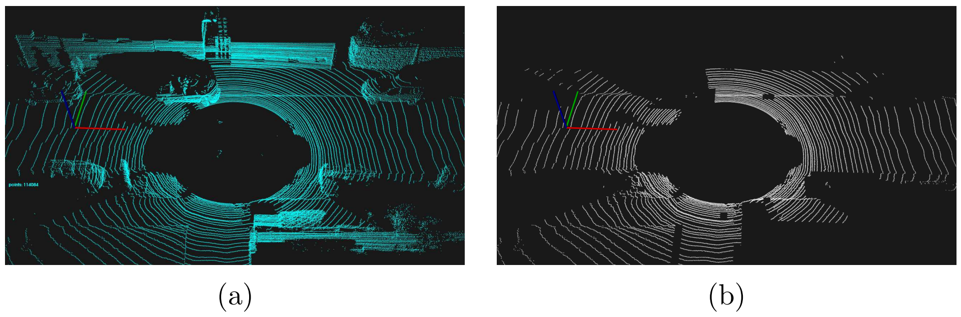


*Figure 4.7: The result of proposed ground points extraction. In (a) all points are showed, while in (b) only the ground ones.*

then the cell is classified as ground if the following holds:

$$H_{ij} - h_{ij} < s; \qquad H_{ij} < H_G + s \tag{4.4}$$

The maximum height $H_{ij}$ and minimum height $h_{ij}$ of a cell serve as lower bound and upper bound for a cell, which is similar to a horizontal plane segmentation, while the ground height $H_G$ and slope $s$ allow for a varying height of the cells in a connected way, thus selecting the ground cells even when a slope is present. A result of this method can be seen in Figure 4.7.

As of performance, the algorithm performs three passes over the all points in the point cloud. First pass is to get the bounding box of the cloud, the second is to divide the points into the 2D grid and the third pass is to compute the maximum and minimum heights of each cell. Then the last pass over all cells is to perform the ground check.

### 4.2.2 Intra-scan and inter-scan fusion

After ground points removal, we describe how we evaluate the occupancy of the location of a point $P$ belonging to scan $S_k$ due to another scan $S_i$. For this, we represent the occupancy space using Demspter-Shafer Theory (DST), similar to what Xiao et al. did. If we consider the space occupancy represented by a universal set $X = \{empty, occupied\}$ then the space reached by a laser scanner ray can be either *empty*, *occupied* or *unknown* (which is the space not yet reached by the ray). In order to apply the DST, the following should hold:

$$m : 2^X \to [0,1], m(\emptyset) = 0, \sum_{A \in 2^X} m(A) = 1 \tag{4.5}$$

where $2^X = \{\emptyset, \{empty\}, \{occupied\}, \{empty, occupied\}\}$ is the power set of $X$ and $m$ is the mass of occupancy. We denote $e = m(\{empty\})$, $o = m(\{occupied\})$ and $u = m(\{unknown\})$. Let then $OQ$ be a laser beam from the scan $S_i$, and $r = length(P'Q)$, where $P'$ is the projection of $P$ on $OQ$. The masses $e_r$ and $o_r$ parametrized over $r$ are defined as follows:

$$e_r = \begin{cases} 1 & \text{if } Q \text{ is behind } P \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

$$o_r = \begin{cases} e^{-\frac{r^2}{2}} & \text{if } P \text{ is behind } Q \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$\tag{4.8}$$

and the occupancy mass of point $P$ at $Q$ is given by:

$$m(P,Q) = \left\{ \begin{array}{c} e \\ o \\ u \end{array} \right\} = \left\{ \begin{array}{c} f_\theta \cdot e_r \\ o_r \\ 1{-}e - o \end{array} \right\} \tag{4.9}$$

$$f_\theta = e^{-\frac{\theta^2}{2\sigma_\theta^2}} \tag{4.10}$$

where $\theta$ is the angle between rays $OP$ and $OQ$ and $\sigma_\theta$ is the angular resolution of the sensor. If we apply DST combination rule with $e_r$ and $o_r$ values, many false positives will emerge. To avoid such misclassification, we weight the occupancy values more towards occupied state by adding the penalty $f_\theta$ to the empty values.

This model can already work, however it does not consider the uncertainties in data. We incorporate the uncertainties, modeled as Gaussians, into the model in the following way:

$$g(m) = \mathcal{N}(0, \sigma_m^2);\ g(r) = \mathcal{N}(0, \sigma_r^2) \tag{4.11}$$

$$F = g(m) \otimes g(r); \tag{4.12}$$

$$e'_r = e_r \otimes F;\ o'_r = o_r \otimes F \tag{4.13}$$

where $\sigma_m$ and $\sigma_r$ are measurement and registration uncertainty parameters, respectively. The occupancy then becomes:

$$m'(P,Q) = \left\{ \begin{array}{c} e' \\ o' \\ u' \end{array} \right\} = \left\{ \begin{array}{c} f_\theta \cdot e'_r \\ o'_r \\ 1{-}e' - o' \end{array} \right\} \tag{4.14}$$

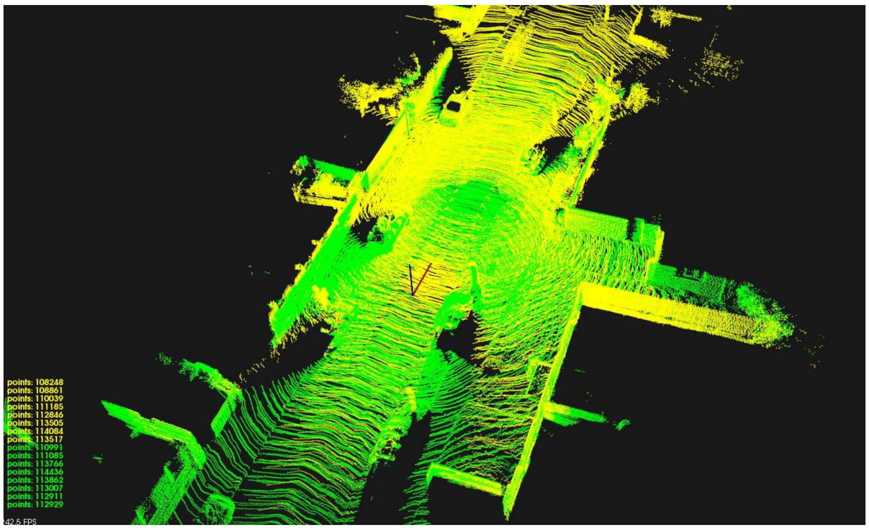
This occupancy is given only for one ray of a scan. To aggregate the occupancy from two rays, the Depster combination rule is applied:

$$\left\{ \begin{array}{c} e_1 \\ o_1 \\ u_1 \end{array} \right\} \oplus \left\{ \begin{array}{c} e_2 \\ o_2 \\ u_2 \end{array} \right\} = \frac{1}{1-K} \left\{ \begin{array}{l} e_1 \cdot e_2 + e_1 \cdot u_2 + u_1 \cdot e_2 \\ o_1 \cdot o_2 + o_1 \cdot u_2 + u_1 \cdot o_2 \\ u_1 \cdot u_2 \end{array} \right\} \tag{4.15}$$

that is commutative and associative, $K = o_1 \cdot e_2 + e_1 \cdot o_2$ is the conflict and indicates incoherence in the aggregation of the above values, $\oplus$ is the fusion operator. The overall occupancy at location $P$ with $I$ number of neighbouring rays $Q_i$ is then given by:

$$m(P) = \bigoplus_{i \in I} m(P, Q_i). \tag{4.16}$$

This formulation gives the occupancy state in a scan $S_k$ of the point $P$ belonging to scan $S_i$. Since we often need to search for the neighborhood of points, we used a kd-tree data structure to improve performance. The neighborhood search space has a cone like shape, which is easy to compute



```
points: 108248
points: 108861
points: 110039
points: 111185
points: 112846
points: 113505
points: 114084
points: 113517
points: 110991
points: 111085
points: 113766
points: 114436
points: 113862
points: 113007
points: 112911
points: 112929
42.5 FPS
```

*Figure 4.8: The previous and future frames considered for point classification. The green points correspond to the previous scans and the yellow ones to the future scans.*

on spherical coordinate system. We transform the points of each scan into spherical coordinate system ($\theta$, $\phi$, $\rho$) by using the following formula:

$$\rho = \sqrt{x^2 + y^2 + z^2} \tag{4.17}$$

$$\theta = arctg(\frac{y}{x}) \tag{4.18}$$

$$\phi = arctg(\frac{\sqrt{x^2 + y^2}}{z}) \tag{4.19}$$

where $(x, y, z)$ are the Cartesian coordinates of a point $P$. The newly computed coordinates $\theta$ and $\phi$ are used as indexing parameters for the kd-tree data structure. This allows a fast neighborhood search for points and thus significantly reduces the computational time. Moreover, to further improve the speed, we limit the neighborhood search to a few samples only, which are the closest to the search point.

In order to classify a point $P$ in a scan $S_k$ as being stationary or mobile, we combine its occupancy values from previous and future scans $\mathbb{S} = \{S_{k-K}, ..., S_{k-1}, S_{k+1}, ..., S_{k+K}\}$ ($K$ is the previous/future scans buffer half length, a visual example of $\mathbb{S}$ is presented in Figure 4.8) by using a novel discretized version of the original DST approach described earlier. Note that the system has a latency of $K$ frames due to the scan buffer. We first define the most distant point $B$ in the sample scan $S_i \in \mathbb{S}$, then we approximate the occupancy values of $P$ with respect to $S_i$ in the following way. We first define a constant

$$l = r_{sup} - \delta r \frac{||\overrightarrow{OP}||}{||\overrightarrow{OB}||} \tag{4.20}$$

where $r_{sup}$ and $r_{inf}$ are user defined upper bound and lower bound and $\delta r = r_{sup} - r_{inf}$. The meaning of constant $l$ is to define a belief stronger in the neighborhood of the laser sensor. Then from the occupancy $(e, o, u)$ computed as in the previous section we define the new occupancy of $P$ with respect to $S_i$:

$$e_{new} = \begin{cases} l & \text{if } e > o \wedge e > u \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

$$o_{new} = \begin{cases} l & \text{if } o > e \wedge o > u \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

$$u_{new} = 1 - e_{new} - o_{new} \tag{4.23}$$

This way the occupancy value of each point is approximated based on its distance from the sample scan origin. With these approximated values we perform again the Dempster combination rule among all sample scans in $\mathbb{S}$. The outcome of this combination defines the classification of the point, if its prevalent occupacy state is *empty* then the point is considered to be dynamic, otherwise it is a stationary point.

Moreover we impose the default occupancy state of a point to *unknown* if this point is outside the bounding box of the other scan. If however, the point is inside bounding box but it has no neighboring rays, it is more likely to be a non-stationary point, and for this reason it is given a predefined state, weigthed more towards *empty*. The ground points are removed only from the reference scan, but are kepts instead in the sample scans. This is done first to avoid testing the ground points (as they are assumed to be always stationary), and second it allows to have neighboring rays which hit the ground in sample scans, and thus identify potential non-stationary points. Once the non-stationary points are identified, they are removed from the reference scan and are kept in a data structure for future uses. The cleaned reference point cloud is then used as a sample scan for future reference scans. This allows to have a slight speed improvement and to better identify changes in future scans.

### 4.2.3 Octree speedup

Testing every point from reference scan with every neighboring ray in the sample scans is very expensive. We propose to avoid such expensive computations by indexing the point cloud with an octree data structure and perform the testing only on a small set of points in leaf nodes. Many non-stationary points in real world cannot be sparse, as they are part of a moving object, which means these points have neighboring non-stationary points. For this reason, if a small set of neighboring points are classified as non-stationary, their neighbors should also be considered as non-stationary.

The proposed method to speed up the change detection is the following:

- Index the point cloud with an octree with a given resolution
- Iterate for each leaf:
  - Extract the points from the leaf
  - Select a subset of points from these points either by using a heuristic or randomly (we opted for the latter)
  - Perform change detection test on this subset and if there are at least half of the points classified as non-stationary, then classify all the points in the leaf as non-stationary. Otherwise the leaf has stationary points.

If the number of points in a leaf however is small, these points are sparse and all these points are tested. This method not only improves performance but also reduces the amount of misclassified mobile points in stationary objects. A visual result of our approach can be seen in Figure 4.9.

### 4.2.4 Global Point Cloud decimation

As an optional step, we have designed a novel method to downsample the global point cloud. The global point cloud, even with the mapping algorithm previously described, still presents many points. To reduce the number of points without losing much quality, we compute the surface curvature of every point, and divide the points by their curvature in a predefined number number of sets (3 in our case). Then we apply a downsampling technique, namely Voxel Grid, on each set with varying voxel size. The Voxel Grid basically divides the point cloud in voxels with a given size and, to decimate, selects the centroid of every voxel. The result of this method can be seen in Figure 4.10. This results in a lot less points without losing much information, which suits very well 3D urban reconstruction methods.

In the following chapter we will discuss the second subsystem.

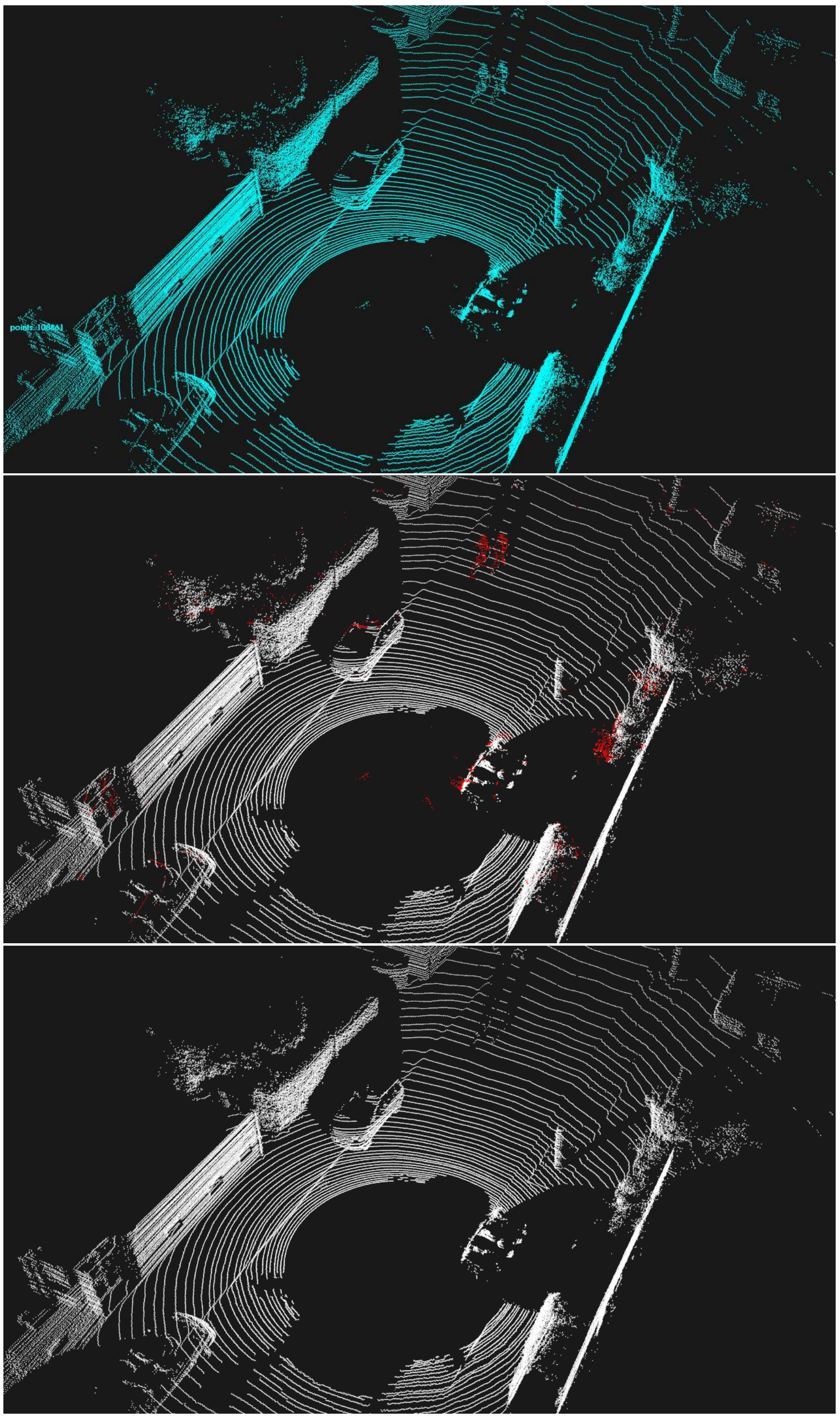


*Figure 4.9: The result of our approach. From top to bottom: the point cloud with all points, the stationary points (white) and detected mobile points (red), and the point cloud without mobile points*

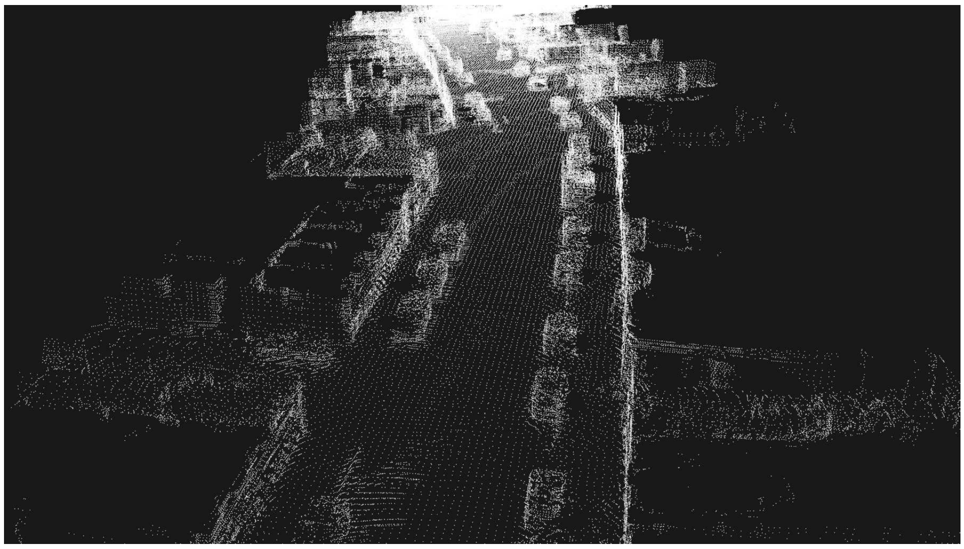


*Figure 4.10: The result of the point cloud decimation based on point curvature.*

# Chapter 5

# Detected changes validation

In the previous chapter we have described the process of detecting points of dynamic objects in a 3D scene acquired by a lidar device. The detection however, is not perfect, as it still manages to output many stationary points marked as mobile, that is, false positives. This is due not only to the imprecision and limitations of the lidar measurements, such as struggling to capture dark materials, but also due to small or very textured objects such as bushes and due to imprecision of the point cloud registration step. To improve the precision of detection, we propose a subsequent validation step of these changes by projecting them into the synchronized camera images and perform a patch similarity test. This process not only gives us less false positives, but also serve as input for a change mask extraction. In this chapter, we will give a detailed explanation of the depth map extraction, changes detection validation and change masks creation steps.

## 5.1 Depth map extraction

A depth map is an image whose intesity values represent depths measured by range sensors or computed by other means such as stereo vision depth estimation. The Velodyne device outputs range measurements of the environment at 360°, with a throughtput of about 1.3 million points. Even if this seems a lot, it is still not enough to extract a dense depth map from it. In fact the straightforward projection of Velodyne depth values into the camera generates a very sparse depth map as can be seen in Figure 5.1.

Our system however require dense depth maps for proper validation. The depth maps are used as a second validation test to cope with color flat surfaces since, as explained later, those create issues for the color patch similarity measurement, which is the first test. Considering the design and

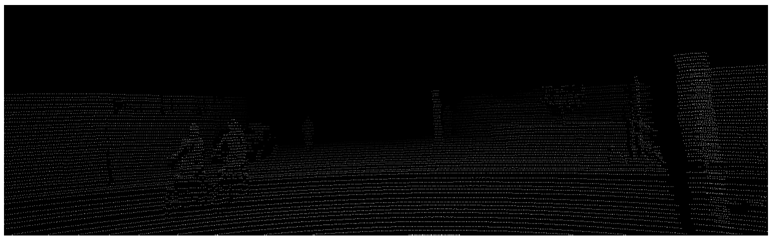


*Figure 5.1: Example of sparse depth map*

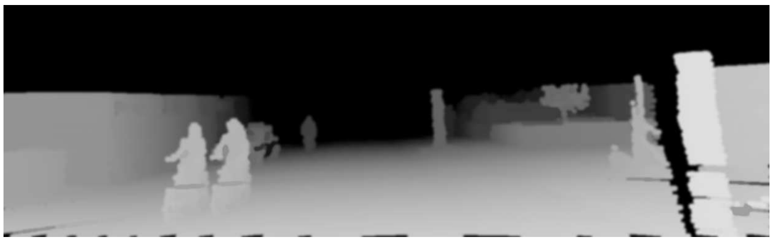


*Figure 5.2: Example of dense depth map*

imprecisions of the lidar device, it is hard to generate a good dense depth from its measurements. Projected points in the image are sparse and are non-uniformally distributed across the whole image.

Considering that our system uses the depth maps only to test flat surfaces, in most of the cases, these don't have abrupt changes in their depths. Therefore the system do not require precise dense depth maps, it is enough only for it to be dense. We do not intend with precise depth maps those which are precise in depth values, but those which are precise on edges, where the depths change abruptly, like edges on a car.

For this reason we used a simple method for extracting detph maps. First the lidar points are projected into the camera, by using the camera matrix, thus creating a grayscale image. Then a simple morphological filter of disk shaped dilation is applied to this image, which should close all the gaps between close points. Finally a gaussian filter is applied to the image to smooth the changes between close blocks resulted from the previous step. However there could still be gaps due to missing projected points from lidar point cloud (low reflective matter), and a hole filling method is required in this cases. The resulting image (see Figure 5.2) is a very approximate depth map of the lidar points, but it is computed fast and is good enough for our second validation step, as points on edges will be tested by the first step.

## 5.2 Validating change detection on images

To check whether a mobile point is a false positive, we have implemented a system which does this check in two stages: the first is on color images and the second is on depth maps. The reason behind these steps is simple. A false positive point, that is, a stationary point misclassified as mobile, projected in two images captured with a short delay between them, will have its neighborhood pixels with the same intensity values. This is true if we ignore factors such as camera noise, illumination changes and if the point is on highly reflective surfaces such as mirrors and car paint. Nevertheless we have implemented the method which compares the neighborhood pixels of the projected point in one image to the neighborhood of the same projected point in another image.

The first stage of the method is to do the comparison on projected point's neighborhood pixels in color images. The 3D point $i$ is projected into a color image $I_k$ using the camera matrix $P$. This projection corresponds to a pixel $p_{ik}$, for simplicity called reference pixel. The squared image patch $patch_{ik}$ around this pixel is selected with a side length $b_{ik}$, measured in number of pixels, given by the following formula:

$$b_{ik} = \frac{h}{d_{ik}} f_{xy} \tag{5.1}$$

where $h$ is a parameter for height in real world, $d_{ik}$ is the distance from the camera at $k$ of the projected point $i$ corresponding to the reference pixel and $f_{xy}$ is the focal length given by the camera intrinsic matrix $K$. For simplicity we considered the pixels to be square. If the pixel size $s$ and the focal distance $f$ are known, then $f_{xy} = f/s$.

Once the patch is extracted, the 3D point is projected into another camera image $I_l$ (where $l = k - \delta t$), taken closely to the first camera image both in time $\delta t$ and space. This projection results in s sample pixel $p_{il}$ on image $I_l$. The patch $patch_{il}$ is then extracted with the same $h$ but a different $d_{il}$ which results in a $b_{il}$ smaller than $b_{ik}$ if $l < k$ if we consider the camera movement direction is forward (camera Z-axis).

Since the comparison should be done on patches of the same size, one of the patches should be resized to reach the dimensions of the other one. The resizing is done with already existing methods, which also do the pixel intensity interpolation. We choose to set $k$ as the last camera image, which by definition makes the $l < k$ in time. This way it is more convenient to perform the resizing of the $patch_{ik}$ to the dimensions of $patch_{il}$, by a factor $\delta r = b_{il}/b_{ik}$, both from quality and performance points of view. It renders the following computation faster by having to compare smaller patches, and interpolating pixel intesities while minimizing patches yields a better result than while maximizing. We tried also warping one patch into another, this idea was then dropped, since the results were very similar to resizing, but the computation effort was higher.

Once both patches are prepared and resized, a similarity comparison is checked. For this comparison, we tried using both the Sum of Squared Differences (SSD) metric as well as Normalized Cross Correlation (NCC) metric to get the patch similarity coefficient. Both metrics were computed on each color channel of the patches indipendently. The SSD is computed in the following way:

$$E_k(patch_i, patch_j) = \frac{1}{b_i^2} \sum_{x=1}^{b_i} \sum_{y=1}^{b_i} (patch_i(x, y, k) - patch_j(x, y, k))^2 \tag{5.2}$$

where $k$ is the color channel and $patch_i(x, y, k) \rightarrow [0, 1]$ is the color intensity in channel $k$ at pixel $(x, y)$ of the patch i. Note that $E_k$ is already normalized. The patches are considered similar if $\forall k, E_k < \tau$, where $\tau$ is the threshold parameter. The NCC instead, is computed as follows:

$$C_k(u, v) = \frac{\sum p_i(x, y, k)(p_j(x - u, y - v, k) - \bar{p_j}(k))}{\sqrt{\sum (p_i(x, y, k) - \bar{p_i}(k))^2 \sum (p_j(x - u, y - v, k) - \bar{p_j}(k))^2}} \tag{5.3}$$

$$E_k(p_i, p_j) = 1 - \max_{u,v} C_k(u, v) \tag{5.4}$$

where $p_i$ and $p_j$ are $patch_i$ and $patch_j$ respectively, $\bar{p_i}(k)$ and $\bar{p_j}(k)$ are the mean of $patch_i$ and $patch_j$ in color channel $k$. To see if patches are similar the same idea applies as with SSD: if $\forall k, E_k < \tau$ then the two patches are similar. Note the anomaly with NCC, if one of the patches has one of its channels flat than the formula above fails, as it is a division by zero problem. This plays well for our sistem, since we have a second test which is specifically designed for flat color surfaces.

Using the SSD has however a limitation. Some patches may change intensity because of illumination changes. This occurs for example when the camera transitions from shadow area to a lit one. This causes the intensities to change in the whole image. SSD fail with these illumination changes, as it compares directly the pixel intensities. This is the reason why we opted for NCC measure, as it is not affected by such illumination changes, even thought it is computationally more expensive than SSD.

Before computing any similarities between patches, these are checked for the uniformmity of their intensities. If their intesity standard deviation is above a certain threshold, then the patches are computed for similarities as explained above, otherwise they are pushed to the second test. The second test performs the same patch projection methodology, like the one discussed above, only instead of color images, the points are projected into the depth images.

Note however, that before performing and patch comparisons on depth images, the images should have comparable intensities. That is, a depth map from a camera position will have different intensities for a static object than a depth map from another camera position. This is because the sensor is in motion and at every scan the scene depth is perceived in a different way. For this reason before performing patch similarities check, the intensities of the depth maps should coincide for the same static object. To perform the normalization we used the following formula:

$$D'_l = D_l + \frac{||\mathbf{t}_k - \mathbf{t}_l||}{d_{l,max}} \tag{5.5}$$

where $D'_l$ and $D_l$ is the new and old intensity values of depth map $l$, $\mathbf{t}_k$ and $\mathbf{t}_l$ are translation vectors of cameras $k$ and $l$ respectively, and $d_{l,max}$ is the maximum points' depth distance from the camera $l$. We used only the translation vectors since the baseline between considered camera frames was small enough to avoid using also the rotation of the cameras. The patches are then extracted, although with a smaller side length to speed up computation, and resized in the same way as before. Finally we check the patches similarity with the SSD metric. This metric is suitable in this case because depth maps are not affected by illumination changes, as in transitions from shadow to lit areas, the depth maps remain the same.

The tests check each mobility point if its corresponding patches are similar. If a point positively passes the tests, then it is considered a moving point and thus a true positive. This happens when patch similarity error is greater then a given threshold. The same threshold is used in both tests for simplicity, although the second test, due to smaller error values, have an additional coefficient to lower the said threshold even more.

The second test is used for points on low-textured surfaces, as those have patches of almost identical neighborhood intensities and thus are hard to spot on a big moving object. Consider a uniformly painted moving van in front of the car where the sensor and camera reside. This scenario will almost certainly fail in the first test, as the patches will be uniform in both sampled images, but it will succeed in the second step, since the depth intensities of the patches will be significantly different. The second test serve somehow as a backup plan if the first test fails.

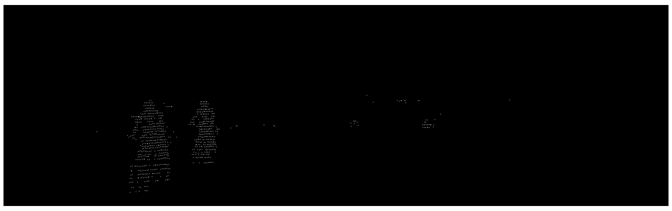


*Figure 5.3: A result of the sparse change mask after the validation check*

## 5.3 Change mask extraction

The points that successfully pass the validation tests are stored in a binary image. Once all the points are tested, the binary image is none other then the chage mask, although still sparse. The sparse change mask can be seen in Figure 5.3. The validation tests are not perfect, and there could still be some false positives. These points most frequently come from vegetation, such as bushes and leafs. However, there can also be false positives due to sensor and camera noise.

Considering these issues, we propose to first perform a neighbor check on the sparse change mask. This check filters out sparse pixels which do not have any close neighbors. A moving object, once detected, will seldom be represented by sparse points. This is why this step is performed. We used a simple box filter and thresholding the result to the pre-defined number of neighboring pixels. This the remaining sparse pixels, which are almost certainly false positives which were not detected by the tests.

Note however that dynamic objects close to the camera and lidar will generate pixels with smaller density than objects far away. To overcome this problem, we could discretize the depth and use a set of binary sparse masks for each discrete depth value. Then perform the filtering for each of these masks indipendently. We have tried this method but, while it gives slightly better results, it is computationally more expensive and the quality/speed trade-off is not good.

To create the dense change mask, we used morphological operators of disk shaped dilation and erosion several times with decreasing disk size. And as a final step, a gaussian low pass filter is performed to obtain a smoother mask. The result of this method can be seen in Figure 5.4.

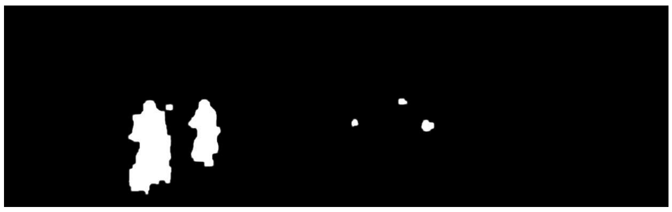


*Figure 5.4: The sparse change mask after the post-processing step results in a dense change mask*

# Chapter 6

# Experimental results

To prove the efficiency of our system, we have designed a set of experimental setups.

## 6.1 System Setup

### 6.1.1 Dataset

For our testing purposes, we have used the KITTI dataset [23]. This dataset was created using a rig of sensors and cameras mounted on a vehicle, as see Figure 3.1.

- 2 × PointGray Flea2 grayscale cameras (FL2-14S3M-C)
- 2 × PointGray Flea2 color cameras (FL2-14S3C-C), 1.4 Megapixels, 1/2″ Sony ICX267 CCD, global shutter
- 4 × Edmund Optics lenses, 4mm, opening angle  90 , vertical opening angle of region of interest (ROI)   35°
- 1 × Velodyne HDL-64E rotating 3D laser scanner, 10 Hz, 64 beams, 0.09 degree angular resolution, 2 cm distance accuracy, collecting  1.3 million points/second, field of view: 360° horizontal, 26.8° vertical, range: 120 m

The cameras, the lidar and the GPS/IMU are all synchronized, and generate a set of data each $1/10s$. The cameras are

### 6.1.2 Hardware and software used

The experiments were executed on a notebook PC having the following configuration:

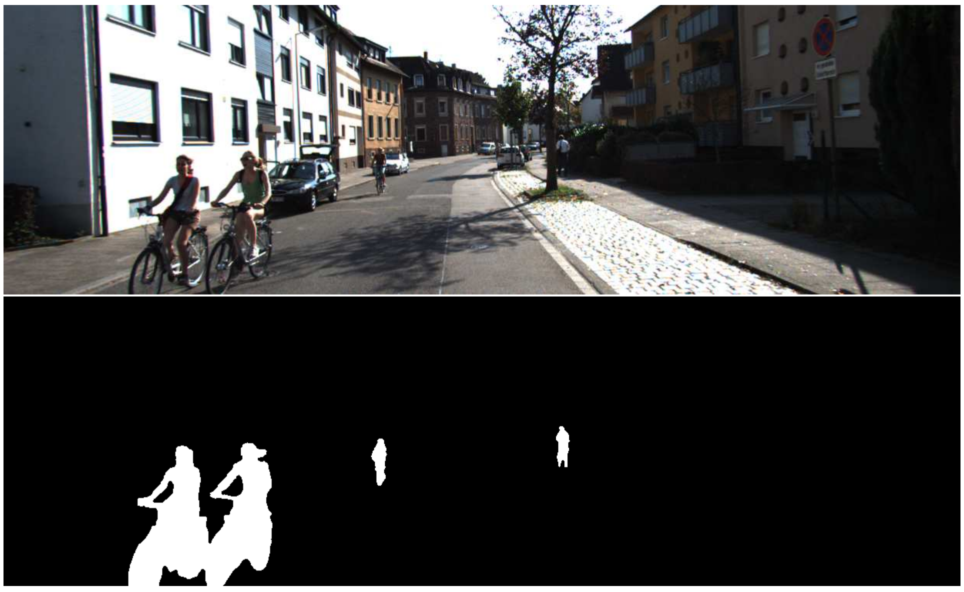


*Figure 6.1: An example of a color image and its corresponding ground truth change mask.*

- Processor: Intel Core i7-3537u, 2GHz, 3.1GHz max Frequency, 2 Cores, 4 Threads
- RAM: 8GB DDR3-1600L
- Storage type: Solid State Drive (SSD)

The system was created and the experiment was performed on a Ubuntu 14.04 operating system. For this system Point Cloud Library (PCL) [49] was used, as it contains many useful algorithms to manipulate point cloud data.

### 6.1.3 Ground truth considerations

To perform the performance analysis, a reference is needed to assest the quality and performance of the results. For this reason we manually annotated the dataset to create the ground truth change masks. The masks were created by selecting by hand moving objects in the images captured by the color camera on the left in the KITTI sensor rig (see cam 2 in Figure 3.1).

An example of the ground truth image mask can be seen in Figure 6.1. These change masks have been created considering the following factors:

1. The far moving objects are seen in images, but they are unreacheable by the lidar.

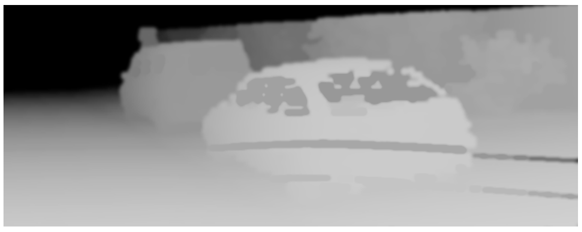


*Figure 6.2: The visible gap between the two laser banks.*

2. Vegetation will most probably be detected as dynamic objects, due to their leafs complex geometry.
3. The selection of moving objects cannot be perfect, as there are many small details that cannot be selected individually. An example is the bicycle wheels, which can be selected entirely or just their tyres.

To avoid future problems caused by the first factor, we selected the objects only once they have entered the scene at a given distance from the camera. To understand when an object has entered into the scene, we have used the same methodology as for the patch extraction. The amount of pixels $h$ in an image corresponding to a given height $H$ in real world at a given distance $d$ is computed by the formula:

$$h = \frac{H}{d} f_{xy} \tag{6.1}$$

where $f_{xy}$ is the focal length in pixels, it is given by the camera intrinsic matrix. By knowing the object's height in real world, and by measuring this height in pixels, we are able to infer its distance from the camera in real world. The distance is given by the point cloud bounding box length minus the distance of the camera from the lidar on the sensor rig. In our case we have found that a height of 1.70m in real world corresponds to circa 48 pixels. For example when a moving person in an image has a height of 48 pixels or more, it is annotated as ground truth.

As for the problems caused by the presence of vegetation, we have simply decided to leave the vegetation as static objects, which is the real case. Related to moving object selection, we have decided to slightly over-select the objects having parts difficult to classify or select, such as bicycle wheels.

There is another problem which is worth mentioning. The Velodyne lidar is composed of a rotating head, which houses two blocks of 32 lasers. These blocks generate points from range measurements, however the transition between the points of the two blocks is not seamless. There is a visible gap between the two, as it can be seen in a depth map image, rendered from the Velodyne points, in Figure 6.2. This factor creates issues as the experiments will never give a perfect score when comparing the proposed algorithm results to the ground truth.

### 6.1.4 Experiments setup

We designed a semi-automatic system to test the performance and robustness of the proposed algorithm. There three types of algorithms we have decided upon: only 3D change detection testing, only 2D changes validation testing and the two methods combined testing.

The testing of the 3D change detection is performed in the following way:

1. To create a comparable ground truth for the change detection results, all points from a point cloud are projected into the an image using the matrix of the same camera on which the ground truth was annotated.

2. This sparse image of the projected points is compared to the ground truth mask with a logical AND operator. This gives us the ground truth values, which are the amount of points projected onto the ground truth moving object.

3. The same procedure is done with the results of the change detection. This gives the amount of points which the change detection got right.

4. We then proceed to compare the ground truth values with our algorithm values.

In order to have a fair comparison between algorithms, we have done the same test with the results from the algorithm of Vallet et al. The results of the comparison between algorithm are showed and discussed in the following section.

We have designed an experimental setup to test the effectiveness of the validation tests both for the worst case scenario, that is, considering all points of the point cloud as dynamic, and normal scenarios with only dynamic points given by the change detection algorithm. The testing is performed as follows:

1. The considered points are projected onto the images using the camera matrix.

2. The validation tests and post-processing are performed on those points and images.

3. The resulting binary mask is directly compared with the ground truth masks.

To illustrate the experimental results we use the Receiver Operating Characteristic (ROC) curve and the Precision-Recall (PR) curve. These curve shows the performance of a binary classification system, which our case, since we want to classify points as being either dynamic or stationary. Both these curves are based on the confusion matrix (see table below).

| | actual positive | actual negative |
|---|---|---|
| predicted positive | True Positive (TP) | False positive (FP) |
| predicted negative | False Negative (FN) | True Negative (TN) |

For our system, the positive outcome of a test means a point is non-stationary. The actual positive and negative values are given by the ground truth. For each experiment there is a short description how to interpret the TP, FP, FN and TN.

To define ROC and PR curves we first have to define some ratios. Recall (or Sensitivity) is the hit rate of dynamic points being classified as such. It is defined in the following way:

$$R = \frac{TP}{TP + FN} \tag{6.2}$$

Specificity (S) is the hit rate of correctly identified stationary points. It is defined as follows:

$$S = \frac{TN}{TN + FP} \tag{6.3}$$

Precision (P) is the ratio of identified true positives over all predicted true positives. It is given by:

$$P = \frac{TP}{TP + FP} \tag{6.4}$$

The ROC curve shows the relationship between Recall and Specificity, while PR curve shows the relationship between Precision and Recall. These curves are deeply related to each other, as noted by [17], when comparing two algorithms with these curves, an algorithm will prevail over the second algorithm in ROC curve if and only if it fails in the PR curve to the second.

## 6.2 Experimental Results

In the following we show and discuss the experimental results. Since we have annotated the dynamic objects only in the Dataset 0095 from KITTI, we have performed the algorithms also on this dataset. We have used 220 frames from that Dataset, starting from frame 35 and ending with 255, as these are the starting and ending frames where dynamic objects appear.

We have the following parameters fixed for all experiments:

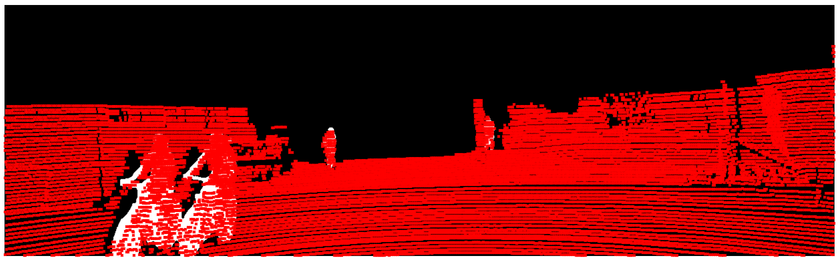
- 3D detection part:
  - Minimum number of sample scans = 20 (10 before and 10 after the reference scan)
  - Bounding box side length = 30m
  - Octree resolution = 0.3m (this is the minimum leaf node size)
  - Lower and upper bound for occupancy mass discretization: $r_{inf} = 0.6$ and $r_{sup} = 0.8$
  - Ground points grid resolution = 0.4m
  - Ground points maximum slope = 22%
- 2D validation part:
  - Real world color patch height = 0.15m
  - Real world depth patch height = 0.05m
  - Sparse to dense depth map dilation disk shape radius = 5px
  - Camera to camera frame distance for patch similarity check = 2 frames

### 6.2.1 3D change detection results

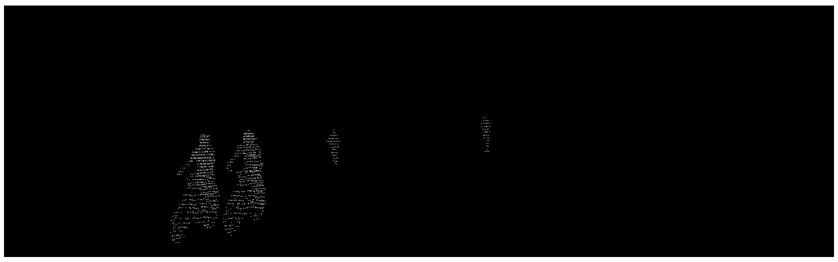
Before showing and explaining the results, a good definition of what is the confusion mask for this experiment should be given. We consider ground truth positive values (GTP), the pixels from projected points which pass the AND operation with the ground truth mask. An example of such resulting pixels can be seen in Figure 6.3. For ground truth negative values (GTN), we perform the AND operation with the negated ground truth mask.

Once the ground truth values are obtained, we do the same process with the non-stationary points from the 3D change detection step (see Figure . Here we define the values of confusion matrix as follows:

- True Positives TP = number of white pixels after the AND operation with the ground truth mask
- False Positives FP = number of white pixels after the AND operation with negated ground truth mask



(a) Projected points on ground truth mask



(b) Result of the AND operation between projected points and ground truth mask

*Figure 6.3: Ground truth values extraction.*

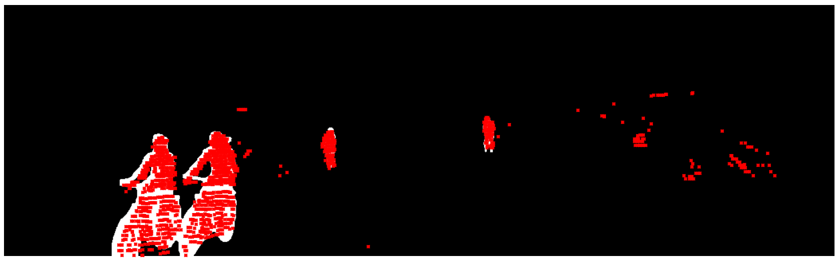


*Figure 6.4: An example of projected dynamic points over the ground truth change mask.*

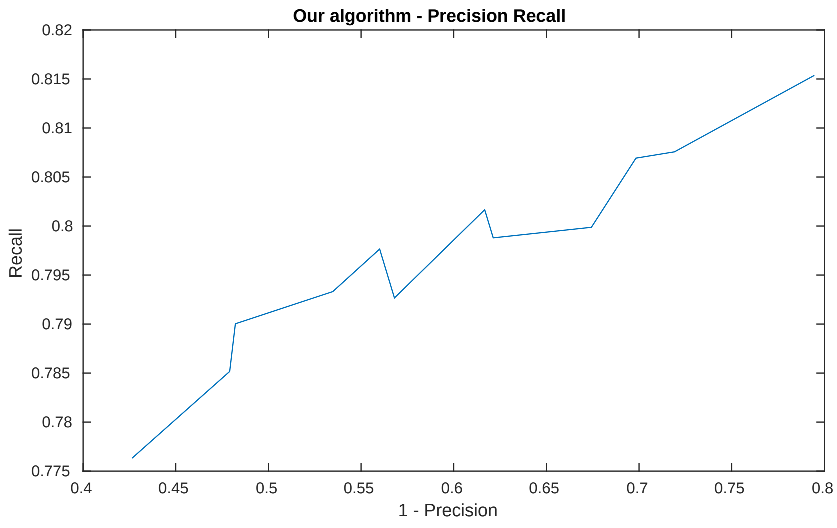


*Figure 6.5: Precision and Recall of our algorithm.*

- False Negatives FN = GTP - TP
- True Negatives TN = GTN - FP

Once we have these values, we can show both the ROC and PR curves. The change detection results were obtained by varying the $\sigma_r$ and $\sigma_\theta$, as these parameters define the elipsoid form of the occupancy space in our algorithm. The values were set to: $\sigma_r \in \{0.1, 0.15, 0.2\}$ and $\sigma_\theta \in \{0.2°, 0.3°, 0.4°, 0.5°\}$. The Figure 6.5 the Precision and Recall curve is shown.

The values in the graph are averaged over a series of 220 frame (lidar point clouds and images). Each point correspond to a given parameters pair. The table below summarizes the points on the curve.

| $\sigma_r(m)$ | $\sigma_\theta(^\circ)$ | Precision | Recall |
|---|---|---|---|
| 0.1 | 0.2 | 0.2057 | 0.8153 |
| 0.1 | 0.3 | 0.3017 | 0.8069 |
| 0.1 | 0.4 | 0.3833 | 0.8017 |
| 0.1 | 0.5 | 0.4400 | 0.7977 |
| 0.15 | 0.2 | 0.2809 | 0.8076 |
| 0.15 | 0.3 | 0.3787 | 0.7988 |
| 0.15 | 0.4 | 0.4653 | 0.7933 |
| 0.15 | 0.5 | 0.5178 | 0.7900 |
| 0.2 | 0.2 | 0.3257 | 0.7999 |
| 0.2 | 0.3 | 0.4320 | 0.7927 |
| 0.2 | 0.4 | 0.5209 | 0.7852 |
| 0.2 | 0.5 | 0.5733 | 0.7764 |

*Table 6.1: Our algorithm PR values*

Before commenting the results, we note that for this dataset, the point cloud registration algorithm fails for the last few frames, therefore affecting the results with many more false positives.

Note how the precision increases while recall decreases when $\sigma_r$ is increasing. This is because slowly moving objects will have a significantly lower points distance between one frame and another, and the $\sigma_r$ computes the occupancy along the laser ray. This leads to assigning occupied mass values to more points along the ray, as $\sigma_r$ increases, which translates to fewer true negatives and fewer false positives. Note the same trending with $\sigma_\theta$. This is somehow counter-intuitive, because with increasing $\sigma_\theta$ should increase the empty mass values of the points. This is in part true, but our algorithm however uses the $\sigma_\theta$ also as neighborhood search parameter. This translates into larger number of neighbors being considered, and thus more possibility to find occupied points, since their occupied mass does not depend on $\sigma_\theta$.

We have performed the same experiment with the Vallet et al. algorithm, although the $\sigma_\theta$ was not used here because it depends on $\sigma_r$ in their algorithm. To compensate the missing values, we tried to vary the $\lambda_\theta$ parameter instead. This parameter represents the angular resolution in their algorithm. The resulting Precision-Recall curve can be seen in Figure 6.6. Their values are summarized in the following table:

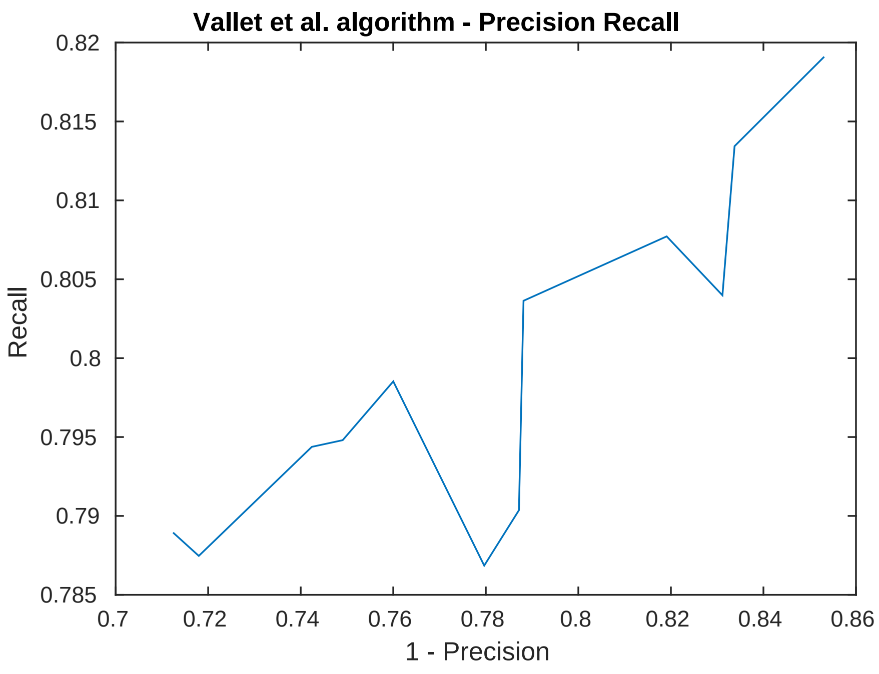


*Figure 6.6: Precision and Recall of Vallet et al. [65] algorithm.*

| $\sigma_r(m)$ | $\lambda_\theta(^\circ)$ | Precision | Recall |
|---|---|---|---|
| 0.1 | 0.2 | 0.1689 | 0.8040 |
| 0.1 | 0.3 | 0.1809 | 0.8077 |
| 0.1 | 0.4 | 0.1662 | 0.8134 |
| 0.1 | 0.5 | 0.1470 | 0.8191 |
| 0.15 | 0.2 | 0.2128 | 0.7904 |
| 0.15 | 0.3 | 0.2509 | 0.7948 |
| 0.15 | 0.4 | 0.2400 | 0.7985 |
| 0.15 | 0.5 | 0.2119 | 0.8036 |
| 0.2 | 0.2 | 0.2203 | 0.7869 |
| 0.2 | 0.3 | 0.2820 | 0.7875 |
| 0.2 | 0.4 | 0.2874 | 0.7889 |
| 0.2 | 0.5 | 0.2576 | 0.7944 |

*Table 6.2: Vallet et al. PR values*

As it can be seen, their algorithm has more Recall than ours, but just by a small margin. However, their algorithm has less Precision than ours.

We have also generated the ROC curve for the same values introduced before, and it can be seen in Figure 6.7. Note the very high specificity, this is because the rate of True Negative values is much more higher than the

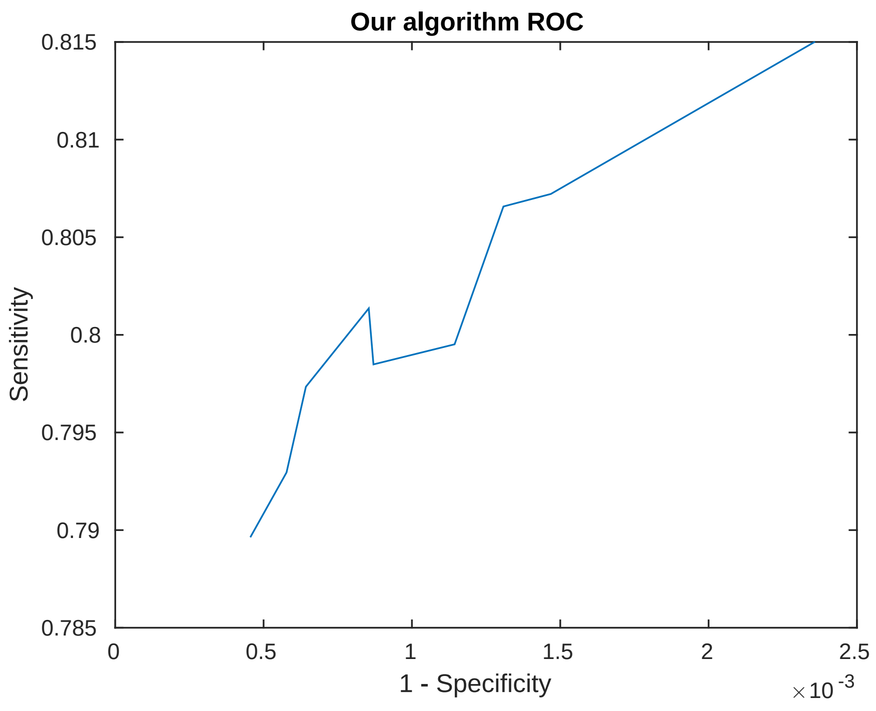


*Figure 6.7: Receiver Operating Characteristic (ROC) of our algorithm.*

False Positive values rate.

As a matter of timing, our algorithm is almost one order of magnitude faster then the Vallet et al. algorithm. These timings are from the same experiments with the same values presented before. Note that we have removed the ground points also in their algorithm and used the same point cloud registration algorithm. Moreover, both algorithms were parallelized with OpenMP directives. The resulting timings can be seen in the following table 6.3:

| Alg | $\sigma_\theta/\lambda_\theta(^\circ)$ | average | total | average pre-processing | total pre-processing |
|---|---|---|---|---|---|
| Proposed | 0.2 | 0.490 | 100.036 | 0.599 | 133.058 |
| | 0.3 | 0.536 | 109.504 | 0.616 | 136.852 |
| | 0.4 | 0.605 | 123.562 | 0.666 | 147.778 |
| | 0.5 | 0.672 | 137.150 | 0.721 | 160.176 |
| Vallet | 0.2 | 2.945 | 615.520 | 3.017 | 653.858 |
| | 0.3 | 4.518 | 970.535 | 4.757 | 1003.17 |
| | 0.4 | 6.505 | 1414.42 | 6.933 | 1444.28 |
| | 0.5 | 9.262 | 2024.43 | 9.924 | 2056.28 |

*Table 6.3: Execution time results*

The first set of rows are the timing results of our algorithm, and the second one theirs. All timing results are in seconds. The execution time increases when $\sigma_\theta/\lambda_\theta$ rises because the search radius of our algorithm was set to $2 \times \sigma_\theta$ and $2 \times \lambda_\theta$ for theirs. The pre-processing times are the change detection execution time with ground points removal time and spatial indexing of the point clouds, such as creation of the kd-trees. Such big difference in execution times is mainly due to octree optimization in our algorithm, and some small time gains are also due to preliminary assignment of occupancy masses to unreacheable points.

### 6.2.2 Worst case for validation testing

We tried lauching the validation tests with all points of the point cloud marked as non-stationary points. This was done in order to see how many points would pass the tests. This experiment and the following one are a bit different from the one before. After projecting the 3D points into the images and performing the validation tests and the post-processing step, the values TP, TN, FP and FN are computed in the following way:

- True Positives TP = number of white pixels after the AND operation with the ground truth mask (gt mask)
- False Positives FP = number of white pixels - TP
- False Negatives FN = number of white pixels in gt mask - TP
- True Negatives TN = number of black pixels in gt mask - FP

From these values the curves ROC and PR are easily computed. The validation step is especially sensitive to point cloud alignment and synchronization errors as can be seen in the exemple in Figure 6.8. In the figure

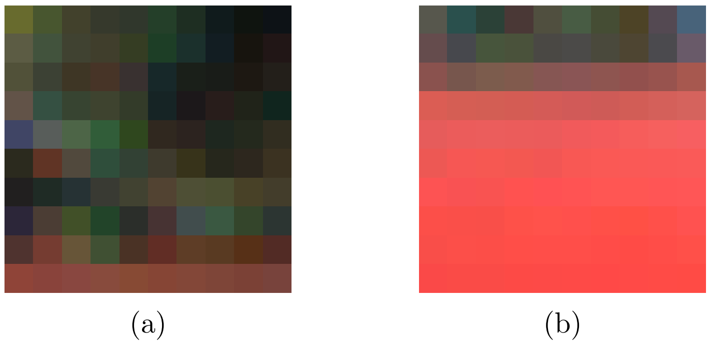


*Figure 6.8: An example of extracted patches in a bumpy part of the road. They correspond to the same 3D point. The patch (a) is from the image taken* $0.1$ *seconds later than the image of patch (b).*

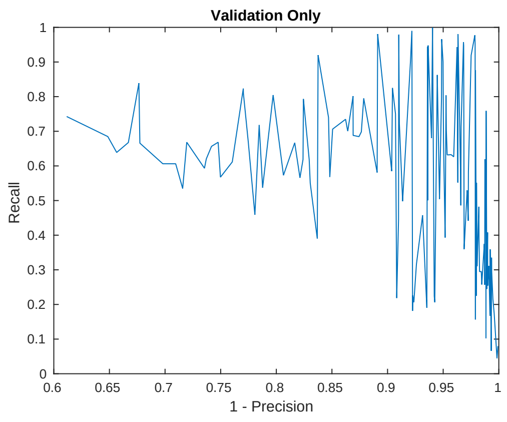
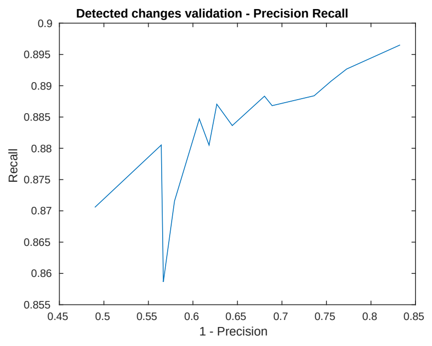


*Figure 6.9: PR curve of the worst case scenario for validation method.*

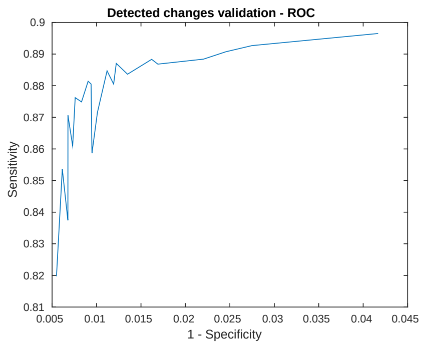
these patches are at a distance of 2 frames from each other, note the significant shift of patch colors. This is because the vehicle was on a bumpy part of the road when the images were captured. This effect nullifies the effectiveness of the validation test for some points in the scene.

In Figure 6.9 the Precision Recall curve is given for the worst case. Notice the anomaly where precision is low. This is due to the validation tests failing in some frames, especially the last few frames where the registration failed.

This experiment proved to be also very computationally intensive, as there were on average about 40000 points projected into the images, as opposed to around 2000 on average, and extracting such amount of patches and performing the NCC on them proved to be very slow (circa 30 seconds per frame on unoptimized matlab code).



(a)



(b)

*Figure 6.10: PR (a) and ROC (b) curves of the dynamic points validation.*

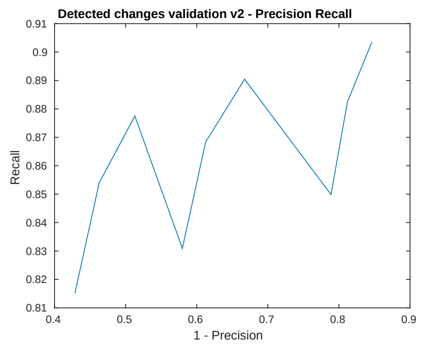
### 6.2.3 Results of validation of dynamic points

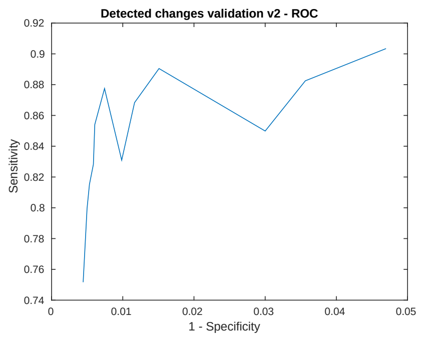
The final experiment we did was to prove the effectiveness of combining the change detection with the validation tests. For this experiment we performed the 3D change detection algorithm with the subsequent points validation tests. Overall the experiment is similar to the last one, although now the dynamic points are projected and validated. The PR and ROC curves can be seen in the Figure 6.10.

For this experiment we have used a threshold $\tau$ for color patch similarity set to 0.1, and for depth patch similarity set to $0.2 \cdot \tau$.

We have also experimented with a fixed set of $\sigma_r$ and $\sigma_\theta$, and a varying threshold $\tau$. The results of this experiment can be seen in the Figure 6.11. The falling spikes correspond to low $\sigma_r$ and low $\sigma_\theta$ and the precision rises with rising $\tau$ at the expense of the falling recall. The values of $\tau$ were set to $\{0.05, 0.15, 0.2\}$, while the $(\sigma_r,\ sigma_\theta)$ were set to $\{(0.1, 0.2), (0.2, 0.3), (0.25, 0.4)\}$.

The ROC curve demonstrates our algorithm is a good classifier, as it has a high Recall (true positive rate) and a high Specificity, which is also called true negative rate. In the Figure 6.12, all the above graphics are shown on the same axis. There we can see a clear advantage of our algorithm by combining the 3D detection of dynamic points with their subsequent validation. As a final result, we show in Figure 6.13 a change mask applied to a red channel of a grayscale image.





(a) (b)

*Figure 6.11: PR and ROC curves of the dynamic points validation when varying the similarity threshold.*

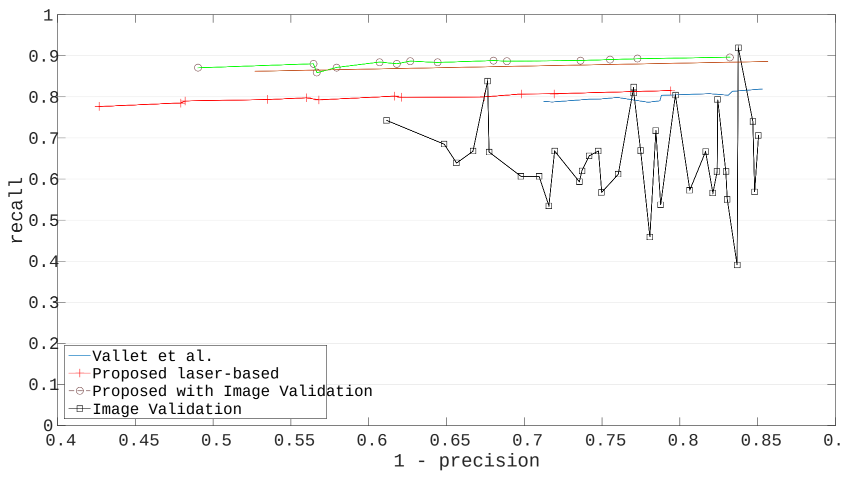


*Figure 6.12: Precision and Recall curves of all the experiments on the same graphic.*

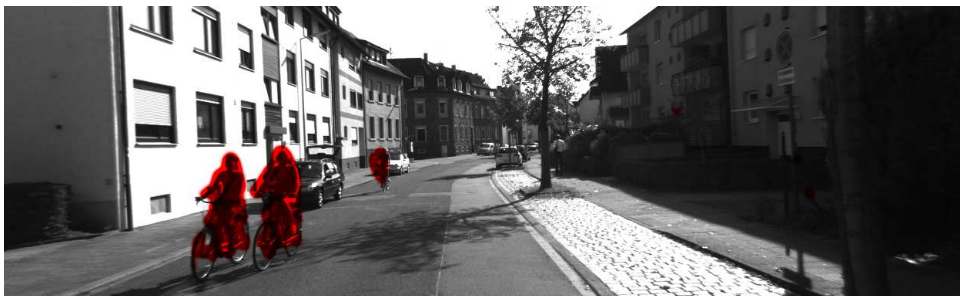


*Figure 6.13: A change mask applied to the red channel in a grayscale image.*

# Chapter 7

# Conclusions and future work

In this thesis, we presented a novel changes detection system in 3D lidar based point clouds with the subsequent image based validation of the detected changes. Our detection algorithm performs in under a second per frame, and it is almost one order of magnitude faster than the considered algorithm by Vallet et al. [65], thanks to the octree optimization speed up and other small enhancements. The experimental results have shown our algorithm to a be a good classifier, and it can output good change masks in difficult scenarios, like the data acquired by a moving sensor rig with a significant distance between two frames (1m on average) and low frame rate for camera images (10 frames per second). Our system is able to take point cloud scans and their corresponding color images and output static point clouds (withoud dynamic objects) and change masks from images. We have also designed a novel way of selecting ground points when various types of slopes are present in the scene. Moreover our system can be instructed to optimize the output point clouds in terms of point density, and thus in space occupancy, for an eventual further usage.

Our system was designed to accomodate a novel urban reconstruction algorithm, proposed by Romanoni et al. [46]. with variable density point clouds cleaned from any moving objects as input. As a future work, we intend to explore new heuristics in selecting points subsets for the octree optimization step. Also a more advanced registration algorithm is needed, since our change detection algorithm robustness relies entirely on the precision of such algorithm. An optical flow from images with point cloud registration method combination could be investigated for a better overall registration. We also intend to incorporate this algorithm as a pre-processing phase for an urban reconstruction algorithm. The change masks could be used for texturing the reconstructed model.

# Bibliography

[1] K Anttila, S Kaasalainen, A Krooks, H Kaartinen, A Kukko, T Manninen, P Lahtinen, and N Siljamo. Radiometric calibration of tls intensity: Application to snow cover change detection. *International Society for Photogrammetry and Remote Sensing, The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2011.

[2] Maha M Azab, Howida Shedeed, Ashraf S Hussein, et al. A new technique for background modeling and subtraction for motion detection in real-time videos. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 3453–3456. IEEE, 2010.

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

[4] Pietro Azzari, Luigi Di Stefano, and Alessandro Bevilacqua. An effective real-time mosaicing algorithm apt to detect motion through background subtraction using a ptz camera. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 511–516. IEEE, 2005.

[5] Emmanuel P. Baltsavias. A comparison between photogrammetry and laser scanning. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 54(2-3):83 – 94, 1999.

[6] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[7] Simone Bianco, Gianluigi Ciocca, and Raimondo Schettini. How Far Can You Get By Combining Change Detection Algorithms?, May 2015.

[8] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. IEEE, 2003.

[9] Thierry Bouwmans. Recent advanced statistical background modeling for foreground detection-a systematic survey. *Recent Patents on Computer Science*, 4(3):147–176, 2011.

[10] Thierry Bouwmans. Background subtraction for visual surveillance: A fuzzy approach. *Handbook on Soft Computing for Video Surveillance*, pages 103–134, 2012.

[11] Brutzer, S. and Hoferlin, Benjamin and Heidemann, Gunther. Evaluation of Background Subtraction Techniques for Video Surveillance. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1937–1944. IEEE, 2011.

[12] Thomas Butkiewicz, Remco Chang, Zachary Wartell, and William Ribarsky. Visual analysis and semantic exploration of urban lidar change detection. In *Computer Graphics Forum*, volume 27, pages 903–910. Wiley Online Library, 2008.

[13] Yu-Ting Chen, Chu-Song Chen, Chun-Rong Huang, and Yi-Ping Hung. Efficient hierarchical method for background subtraction. *Pattern Recognition*, 40(10):2706–2715, 2007.

[14] Gustave Choquet. Theory of capacities. In *Annales de l'institut Fourier*, volume 5, pages 131–295. Institut Fourier, 1954.

[15] Robert T Collins, Alan Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, Osamu Hasegawa, Peter Burt, et al. *A system for video surveillance and monitoring*, volume 2. Carnegie Mellon University, the Robotics Institute Pittsburg, 2000.

[16] Dubravko Culibrk, Oge Marques, Daniel Socek, Hari Kalva, and Borko Furht. Neural network approach to background modeling for video object segmentation. *Neural Networks, IEEE Transactions on*, 18(6):1614–1627, 2007.

[17] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[18] Arthur P Dempster. Upper and lower probabilities induced by a multi-valued mapping. *The annals of mathematical statistics*, pages 325–339, 1967.

[19] Fida El Baf, Thierry Bouwmans, and Bertrand Vachon. Foreground detection using the choquet integral. In *Image Analysis for Multimedia Interactive Services, 2008. WIAMIS'08. Ninth International Workshop on*, pages 187–190. IEEE, 2008.

[20] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. In *Computer VisionâECCV 2000*, pages 751–767. Springer, 2000.

[21] Enrique J Fernandez-Sanchez, Javier Diaz, and Eduardo Ros. Background subtraction based on color and depth using active sensors. *Sensors*, 13(7):8895–8915, 2013.

[22] Enrique J. Fernandez-Sanchez, Leonardo Rubio, Javier Diaz, and Eduardo Ros. Background subtraction model based on color and depth cues. *Machine Vision and Applications*, pages 1–15, 2013.

[23] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[24] D Girardeau-Montaut, M Roux, R Marc, and G Thibault. Change detection on points cloud data acquired with a ground laser scanner. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(part 3):W19, 2005.

[25] Eric Hayman and Jan-Olof Eklundh. Statistical background subtraction for a mobile observer. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 67–74. IEEE, 2003.

[26] Berthold KP Horn, Hugh M Hilden, and Shahriar Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *JOSA A*, 5(7):1127–1135, 1988.

[27] KH Hsiao, JK Liu, MF Yu, and YH Tseng. Change detection of landslide terrains using ground-based lidar data. In *XXth ISPRS Congress, Istanbul, Turkey, Commission VII, WG*, volume 7, page 5, 2004.

[28] YZ Hsu, H-H Nagel, and G Rekers. New likelihood test methods for change detection in image sequences. *Computer vision, graphics, and image processing*, 26(1):73–106, 1984.

[29] Pierre-Marc Jodoin. Changedetection.net - a video database for testing change detection algorithms, 2012.

[30] Pakorn KaewTraKulPong and Richard Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Video-based surveillance systems*, pages 135–144. Springer, 2002.

[31] Soo Wan Kim, Kimin Yun, Kwang Moo Yi, Sun Jung Kim, and Jin Young Choi. Detection of moving objects with a moving camera using non-panoramic background model. *Machine vision and applications*, 24(5):1015–1028, 2013.

[32] Andrew HS Lai and Nelson HC Yung. A fast and accurate scoreboard algorithm for estimating stationary backgrounds in an image sequence. In *Circuits and Systems, 1998. ISCAS'98. Proceedings of the 1998 IEEE International Symposium on*, volume 4, pages 241–244. IEEE, 1998.

[33] Horng-Horng Lin, Tyng-Luh Liu, and Jen-Hui Chuang. A probabilistic svm approach for background scene initialization. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 893–896. IEEE, 2002.

[34] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 2004.

[35] Martin Magnusson. The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection. 2009.

[36] Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[37] O Monserrat and M Crosetto. Deformation measurement using terrestrial laser scanning data and least squares 3d surface matching. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1):142–154, 2008.

[38] Hiroshi Murakami, Katsuto Nakagawa, Taku Shibata, and Eiji Iwanami. Potential of an airborne laser scanner system for change detection of urban features and orthoimage development. *International Archives of Photogrammetry and Remote Sensing*, 32:422–427, 1998.

[39] Nuria M Oliver, Barbara Rosario, and Alex P Pentland. A bayesian computer vision system for modeling human interactions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):831–843, 2000.

[40] Donovan H. Parks and Sidney Fels. Evaluation of Background Subtraction Algorithms with Post-Processing. In *AVSS*, pages 192–199. IEEE Computer Society, 2008.

[41] Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.

[42] Shi Pu, Martin Rutzinger, George Vosselman, and Sander Oude Elberink. Recognizing basic structures from mobile laser scanning data for road inventory studies. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(6):S28–S39, 2011.

[43] Rongjun Qin and Armin Gruen. 3d change detection at street level using mobile laser scanning point clouds and terrestrial images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 90:23–35, 2014.

[44] R.J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. *Image Processing, IEEE Transactions on*, 14(3):294–307, 2005.

[45] Stephen E Reutebuch, Hans-Erik Andersen, and Robert J McGaughey. Light detection and ranging (lidar): an emerging tool for multiple resource inventory. *Journal of Forestry*, 103(6):286–292, 2005.

[46] Andrea Romanoni and Matteo Matteucci. Incremental reconstruction of urban environments by edge-points delaunay triangulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015*, 2015.

[47] Andrea Romanoni, Matteo Matteucci, and Domenico G Sorrenti. Background subtraction by combining temporal and spatio-temporal histograms in the presence of camera movement. *Machine vision and applications*, 25(6):1573–1584, 2014.

[48] Paul L Rosin and Efstathios Ioannidis. Evaluation of global image thresholding for change detection. *Pattern Recognition Letters*, 24(14):2345–2356, 2003.

[49] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[50] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: Science and Systems*, volume 2, 2009.

[51] Kari Sentz and Scott Ferson. *Combination of evidence in Dempster-Shafer theory*, volume 4015. Citeseer, 2002.

[52] Jacopo Serafin and Giorgio Grisetti. Using augmented measurements to improve the convergence of icp. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 566–577. Springer, 2014.

[53] Glenn Shafer et al. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, 1976.

[54] Yaser Sheikh, Omar Javed, and Takeo Kanade. Background subtraction for freely moving cameras. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1219–1225. IEEE, 2009.

[55] Mohamad Hoseyn Sigari, Naser Mozayani, and H Pourreza. Fuzzy running average and fuzzy background subtraction: concepts and application. *International Journal of Computer Science and Network Security*, 8(2):138–143, 2008.

[56] Ashbindu Singh. Review article digital change detection techniques using remotely-sensed data. *International journal of remote sensing*, 10(6):989–1003, 1989.

[57] Kurt Skifstad and Ramesh Jain. Illumination independent change detection for real world image sequences. *Computer Vision, Graphics, and Image Processing*, 46(3):387–399, 1989.

[58] Andrews Sobral and Antoine Vacavant. A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos. *Computer Vision and Image Understanding*, 122:4–21, 2014.

[59] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[60] J Tian, P Reinartz, P d'Angelo, and M Ehlers. Region-based automatic building and forest change detection on cartosat-1 stereo imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 79:226–239, 2013.

[61] Ying-Li Tian, Max Lu, and Arun Hampapur. Robust and efficient foreground analysis for real-time video surveillance. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 1182–1187. IEEE, 2005.

[62] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 255–261. IEEE, 1999.

[63] Rim Trabelsi, Fethi Smach, Issam Jabri, and Ammar Bouallegue. Multimodal background modeling using rgb-depth features. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 884–892. Springer, 2014.

[64] Antoine Vacavant, Thierry Chateau, Alexis Wilhelm, and Laurent LequiÃ¨vre. A benchmark dataset for outdoor foreground/background extraction. In Jong-Il Park and Junmo Kim, editors, *Computer Vision - ACCV 2012 Workshops*, volume 7728 of *Lecture Notes in Computer Science*, pages 291–300. Springer Berlin Heidelberg, 2013.

[65] Bruno Vallet, Wen Xiao, and Mathieu Brédif. Extracting mobile objects in images using a velodyne lidar point cloud. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:247–253, 2015.

[66] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[67] T Vögtle and E Steinle. Detection and recognition of changes in building geometry derived from multitemporal laserscanning data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35(B2):428–433, 2004.

[68] Aloysius Wehr and Uwe Lohr. Airborne laser scanningâan introduction and overview. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 54(2-3):68 – 82, 1999.

[69] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780–785, 1997.

[70] Lu Xia, Chia-Chih Chen, and Jake K Aggarwal. Human detection using depth information by kinect. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, pages 15–22. IEEE, 2011.

[71] Wen Xiao, Bruno Vallet, and Nicolas Paparoditis. Change detection in 3d point clouds acquired by a mobile mapping system. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(2):331–336, 2013.

[72] Kwang Moo Yi, Kimin Yun, Soo Wan Kim, Hyung Jin Chang, and Jin Young Choi. Detection of Moving Objects with Non-stationary Cameras in 5.8ms: Bringing Motion Detection to Your Mobile Device. In *CVPR Workshops*, pages 27–34. IEEE, 2013.

[73] Lotfi A Zadeh. *On the validity of Dempster's rule of combination of evidence.* Electronics Research Laboratory, University of California, 1979.

[74] Reem Zeibak and Sagi Filin. *Change detection via terrestrial laser scanning.* PhD thesis, Technion-Israel Institute of Technology, Faculty of Civil and Environmental Engineering, 2008.

[75] Hongxun Zhang and De Xu. Fusing color and texture features for background model. In *Fuzzy Systems and Knowledge Discovery: Third International Conference, FSKD 2006, Xi'an, China, September 24-28, 2006. Proceedings*, pages 887–893. Springer, 2006.

[76] Liang Zhou and George Vosselman. Mapping curbstones in airborne and mobile laser scanning data. *International Journal of Applied Earth Observation and Geoinformation*, 18:293–304, 2012.