TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ESA HYTTINEN

3D TRACKING OF OBJECTS IN REAL TIME

Master of Science Thesis

Examiners: prof. Jouni Mattila, prof. Matti Vilkko
Examiner and topic approved on 31 May 2017

ABSTRACT

ESA HYTTINEN: 3D tracking of objects in real time
Tampere University of Technology
Master of Science Thesis, 39 pages
April 2018
Master's Degree Programme in Automation Technology
Major: Process Automation
Examiners: Professor Jouni Mattila, professor Matti Vilkko

Keywords: object tracking, object recognition, point cloud

This thesis aims to explore the problem of object tracking. This included reviewing existing applications and technologies related to the problem, and testing one approach via setting up a system that tracks obstacle location. Also, the suitability of the selected hardware was to be assessed. Obstacle tracking solutions could be used in various tasks with autonomous mobile machines, for example avoiding collisions with the environment.

The system built for this project consisted of a two-dimensional laser scanner mounted on a rotating shaft. Shaft was rotated giving the scanner a nodding motion and therefore making possible to scan a three-dimensional point cloud. The point cloud was used for obstacle position estimation and tracking using tools provided by Point Cloud Library. The system performance was evaluated using a physical object whose position was estimated using the scanner, and moving the object in a controlled and measurable manner.

The system tested within this project was able to track obstacle location. The error in obstacle position was up to 0.15 m, and tracking was delayed up to 0.5 s. The position estimation also tended to have high sudden variations not related to the real movement of the obstacle. The performance was not quite what modern hardware used in similar tasks is capable of, and suggests that either the approach presented here is not optimal, or that there are several areas that have to be improved. The issue with high variation in the position estimate must also be investigated should this line of research be continued in the future.

TIIVISTELMÄ

ESA HYTTINEN: Esineiden 3D-seuranta reaaliajassa
Tampereen teknillinen yliopisto
Diplomityö, 39 sivua
Huhtikuu 2018
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Prosessien hallinta
Tarkastajat: professori Jouni Mattila, professori Matti Vilkko

Avainsanat: esineiden seuranta, esineiden tunnistus, pistepilvi

Tämä opinnäytetyö käsitteli esineiden 3D-seurantaa. Tämä sisälsi olemassa olevien teknologioiden ja ratkaisujen selvitystä, sekä erään menetelmän testaamista pystyttämällä järjestelmä joka seuraa esimerkki esineen sijaintia. Lisäksi, järjestelmän soveltuvuutta esineiden seurantaan tutkittiin ylipäätään. Esineiden seurantaa on mahdollista hyödyntää eri sovelluskohteissa, kuten esimerkiksi esteiden väistämisessä autonomisten työkoneiden tapauksessa.

Järjestelmä, joka rakennettiin tätä projektia varten, käsitti laserskannerin kiinnitettynä nivelöityyn alustaan. Alustaa liikuteltiin niin, että skanneri tuotti kolmiulotteisen pistepilven ympäristöstään. Pistepilven avulla selvitettiin esineen sijainti ja seurattiin sitä. Tämä tapahtui Point Cloud Library –nimisen työkalun avulla. Järjestelmän suorituskykyä ja toimintaa arvioitiin liikuttelemalla fyysisitä esinettä järjestelmän työalueella siten, että esineen paikka mitattiin samalla kun järjestelmä seurasi esineen sijaintia.

Lopputuloksena todettakoon, että järjestelmä kykeni seuraamaan esineen sijaintia. Virhe sijainnissa oli noin 0,15 m, ja viive esineen liikkeiden seurannassa noin 0,5 s. Järjestelmän tuottama sijainnin estimaatti lisäksi vaihteli nopeassa tahdissa, vaikka esine itsessään pysyi paikallaan. Suorituskyky ei ollut sitä, mikä olisi ollut laitteiston ominaisuuksien perusteella odotettava, ja tämä antaa ymmärtää, että joko valittu lähestymistapa ei ole optimaalinen tahi useilla osa-alueilla on parannettavaa. Erityisesti sijaintitiedon vaihtelu on seikka, joka täytyy korjata mikäli tämänkaltaista järjestelmää aiotaan jatkokehittää.

PREFACE

I must thank professors Jouni Mattila and Matti Vilkko, as well as Laboratory of Automation and Hydraulics, for providing the opportunity to do this thesis and for the input necessary for its completion.

Thanks also go to other people I have the privilege of knowing, including, but not limited to, family and friends. Even though some are not even aware of this thesis, their contribution to my well-being has been important.

Tampere, 24.4.2018

Esa Hyttinen

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CAN	Controller Area Network
DATMO	Detection And Tracking of Moving Objects
FOV	Field Of View
ICP	Iterative Closest Point
LaDAR	Light imaging, Detection And Ranging
LiDAR	Light imaging, Detection And Ranging
PCL	Point Cloud library
SLAM	Simultaneous Localization And Mapping
UDP	User Datagram Protocol

1. INTRODUCTION

Mapping the environment and avoiding obstacles are common problems in robotics, and different approaches have been taken to solve them. The increased interest in autonomous vehicles, both cars and other types, is also increasing their relevance. Avoiding obstacles requires that they are first detected somehow, and this requires that there is a sensor for sensing the environment. Currently, a variety of sensors for such tasks are available, for example a digital camera, a radar, or a laser scanner.

The goal of this project was to investigate the problem of tracking object position and orientation real-time in three dimensions. This includes finding out available solutions for this type of task, creating one possible setup and testing its performance and suitability for the task. A great deal of work done within this thesis was to implement a tracking system with resources available, and make it as fast and reliable as possible.

The object tracking researched within this thesis could be used as part of projects regarding autonomous mobile machines. Aim of such project would be for example to make the machine beam to automatically dodge obstacles that are in its working area. Information about obstacle position is obviously a key component for that.

In this thesis, the key components of a system which aims to track a moving obstacle in real time are presented. System performance is thereafter tested with a real setting, and further fields of improvement are discussed.

The system consists of a two-dimensional laser scanner mounted on a rotating shaft. The scanner itself is capable of measuring distances horizontally on a single plane, and since three-dimensional measurements are required, the scanner shaft is rotated to move the scanner in a nodding motion. This will also mean that the dynamics and dimensions of the scanner base rotation must be taken into account. Scanner resolution depends on scanning frequency, and there is a trade-off between them: higher frequency gives faster response, but yields lower resolution.

The scanner data is combined into a point cloud, which represents the environment. The point cloud is then utilized in order to track the obstacle pose within the environment. Point clouds are processed using Point Cloud Library (PCL) [1]. It is an open-source framework designed for handling point cloud data.

Aside from laser scanners, other option for producing point cloud data from environment is a depth camera, which produces depth information (that is, distance from camera to each point in the picture) in addition to regular image. A notable example of one is Microsoft Kinect, whose introduction has made it easy for hobbyists to start building point cloud applications of their own.

There are also devices available that are based on time-of-flight camera technology. These have also capability to produce a full 3D-point cloud at each interval. Time-of-flight cameras rely on infra-red illumination they produce themselves, and bright sunlight can pose a problem with that. They are also more susceptible to dirt and other optical occlusions than laser scanners.

Documentation for PCL describes a method for object tracking [2], however, in that example the data comes from a Kinect camera, so it’s not directly suitable for this project. Both Kinect and PCL have been used in small scale projects working with point clouds [6].

The hardware requirements for the system are a big issue, since it has to be able to handle large amounts of data in short timeframe, as the goal is to have obstacle tracking running live. Another issue is also finding suitable tools and methods to process the point cloud. The system’s mechanical properties set some limitations. Oscillating motion of the scanner cannot be very fast, because its structure does not permit arbitrarily large acceleration, which gives a limit to how fast the tracking can be. Another possible problem is that as the scanner has a fixed location, the data it can acquire from the environment may be incomplete if there are multiple obstacles in front of each other.

Tracking obstacles in two dimensions would be an easier task in many ways. The laser scanner could be stationary, and it would be sufficient to use two-dimensional point cloud. In several applications that track pedestrians, this kind of system is used. Two dimensional point clouds tend to require fewer points to contain necessary information, and therefore the point cloud processing time would be shorter. In addition to that, assuming that the obstacle essentially moves only in two dimensions makes it easier to estimate its movement.

Laser scanners are often used in applications where dimensional attributes or geometry of the environment needs to be measured. Examples include mapping of geological formations, or building interiors and exteriors. Geological mapping especially may require high scanning range, and laser scanners often provide that. Measures [3] mentions the following examples, as early as 1984: measuring water depth from a helicopter, cloud movement or concentration of certain substances in atmosphere locally.

Nowadays laser scanners are also commonly used in autonomous vehicle development, for sensing the environment and localizing the vehicle itself. Also, detecting and tracking pedestrians and other traffic in an outdoor traffic environment is an often-researched application of obstacle detection and tracking. One such example is a review from Mertz et al [11], which gives an overview of their DATMO–system (Detection And Tracking of

Moving Objects), and also lists several other papers researching the same subject. A. Azim and O. Aycard [10] also have done research about situations, where pedestrians and vehicles are tracked in an outdoor environment.

In the case of traffic obstacle tracking, any data about the obstacles is not known beforehand, and both their sizes and locations have to be estimated. There are, however, a number of helpful assumptions that can be made in such scenarios: First, all of the obstacles are located on the ground; second, it can be assumed that they have a certain minimum height, as cars and humans usually are higher than 1m. These mean that horizontally scanning two dimensional laser scanner at a correct level can provide sufficient data for the task. Indeed, most of the papers listed by Mertz et al [11] use one or more 2D laser scanners.

The approach to get 3D point clouds in this thesis is to mount laser scanner in a rotating platform, and apply a nodding-type motion to it. A similar approach is used by A. Harrison and P. Newman [15]. Thielemann et al. [14] explores the possibility to use an external rotating mirror in front of a static 2D laser scanner instead of moving the laser scanner itself, and use the mirror to deflect the beam and acquire three dimensional point cloud. In their paper, however, they only speculate with the idea while their actual setup has a rotating scanner.

A different approach is to move laser scanner in a “roundly swinging” pattern, presented by Yoshida et al. [16]. Their solution also makes it possible to adjust the point density in specific region, allowing the system to better focus on interesting features in the environment.

A thesis with quite similar subject was done in Tampere University of Technology in 2014 [17]. That project was in many ways similar to this thesis. Its goal was obstacle tracking, within the context of mobile machines. The system used here had an indoor test setup with a moving obstacle and laser scanner mounted on fixed platform, whereas the other project had the scanner mounted on a mobile vehicle that was tested outdoors.

This work has relevance considering current interest towards autonomous vehicles, though highest interest on that field is directed towards people transport instead of utility machines. It is almost sure that the ever-growing level of automation will eventually spread into other fields, and interest towards them will also rise.

This thesis is organized as follows: chapter two presents computational and other methods used to generate and process point cloud data, and also methods that are used to register point clouds and subsequently to track the obstacle. Chapter three lists hardware components of the tracking system, and goes through the application specific mathematics. Experiments that are used to test the tracking system are listed in chapter 4, and results of them in chapter 5. Chapter 6 contains conclusions of the thesis and this project overall.

2. METHODS

This chapter gives an overview of the procedures that are relevant with this thesis. First, a short overview to laser scanner technology is given. Point clouds and methods for point cloud processing are another important topic. Also, system kinematics and algorithms, most importantly Kalman filter are presented.

2.1 Laser scanner technology

A laser scanner, in the context of this thesis, is a device that measures distances to objects using a laser beam. Other names along laser scanner include laser range finder [11], or more often either LIDAR or LADAR [3], [11]. A term LIDAR is an acronym from light detection and ranging, and LADAR is used as a synonym for that. Usually, a laser scanner can measure distances almost-simultaneously in many directions. This is done using a rotating mirror that directs the beam radially on a single plane. An example of this type of LIDAR is SICK LMS-511, [5]. There are also devices that can get range measurements from multiple angles in two directions (e.g. horizontally and vertically). That requires more refined mechanics, and the devices capable of that are also more expensive. One such device is manufactured by Velodyne [18].

A laser scanner measures distance by emitting a laser pulse into certain direction, and measuring time until a reflection returns. The distance to nearest obstacle in that direction is then calculated based knowing laser pulse flight time and speed of light. According to this concept, the devices are said to operate with time-of-flight principle. Another possible operating principle is phase shift, which uses continuously illuminating laser with amplitude modulation. [4]

2.2 Point cloud

A point P is an object with three coordinate values. It represents location in Euclidean space. There is not a standard way of formulating a point, but it could be defined as follows:

$$p = \{x, y, z\} \tag{1}$$

Where x, y and z are the respective coordinates. Exact definition of a point is ultimately application specific, and it is possible that a point contains other data in addition to location. Extra data could be, for example, a surface normal vector heading, or colour information. In such case, the point formulation could be

$$p = \{x, y, z, x_n, y_n, z_n\} \tag{2}$$

where x_n, y_n and z_n are components of the normal vector. A point cloud C can then be defined as a set of n points:

$$C = \{p_1, p_2, p_3, \dots, p_n\} \tag{3}$$

If points are acquired from an actual object or an environment, the point cloud then can contain information about the geometry of the object or environment. Points can either be only sampled from surfaces, or there can also be points that represent the inside of objects. Usually, the order of individual points in the cloud is not important, as the points together represent the object or environment in whole.

2.2.1 Filtering of point clouds

Depending of application and use case, a point cloud may contain unwanted data. There are at least two such scenarios: points may be sampled with an unnecessarily high density, and there may exist points that lie outside the region of interest. Both of these issues are relatively straightforward to solve, though.

The problem with extra points outside wanted range is that algorithms that try to find specific features from the cloud will have more possibilities for mismatches. Assuming that the coordinates for region of interest are known, points with coordinates outside that region can simply be discarded. This method is further called pass-through filtering.

The problem with point cloud of too high point density is that it requires more processing power than otherwise similar point cloud with lower point density. Addressing this problem is more complicated. One method is to divide the region into cubic sections of identical size. The size should be chosen so that the desired density is acquired while there exists one point within a section. Then, for each section, all points that are inside the section are discarded and replaced with a single point whose position is arithmetic mean of the discarded points' positions. The result is a point cloud with a point density specified by the cubic section size, and which somewhat represents the original data. This method is briefly explained also in [19], and is called down sampling.

2.3 Kinematics

This section defines some geometric properties, which are further needed when generating point clouds and executing obstacle tracking. First, a method for expressing rotating operations of points in Euclidean space is defined. A point, $\boldsymbol{x}$, consists of three coordinate values that form a vector:

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{4}$$

Rotating the point around origin can be achieved by multiplying the vector by a certain matrix, hereby called a rotational matrix R.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{5}$$

$$\boldsymbol{x}' = R * \boldsymbol{x} \tag{6}$$

In the equation above, $\boldsymbol{x}'$ is the coordinate vector after rotation. A group of multiple points can be rotated around origin in similar fashion, by multiplying the matrix containing the points by the rotation matrix.

Any rotation can also be thought to be a result from a combination of elementary rotations applied around x, y and z-axes. If elementary rotations are known, rotation matrices for each axis can be constructed as follows:

$$R_x = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ \sin\alpha & -\cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{8}$$

$$R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{9}$$

Variables α, β and γ are rotation angles around respective axis. Multiplying these matrices in order results a rotation matrix that represents combination of those three rotations:

$$R = R_x * R_y * R_z \tag{10}$$

This process is also reversible. It is, however, important to make sure that the order in which the rotations are applied is always the same, as not doing that will lead to incorrect results. Equations for getting the rotation angles from a rotation matrix are the following:

$$\alpha = atan2(r_{21}, r_{11}) \tag{11}$$

$$\beta = atan2\left(-r_{31}, \sqrt{r_{32}^2, r_{33}^2}\right) \tag{12}$$

$$\gamma = atan2(r_{21}, r_{11}) \tag{13}$$

Variables α, β and γ are, again, rotation angles around coordinate axes. Function $atan2$ is multi-valued inverse tangent.

In some corner cases, using rotation matrixes and individual rotation angles does not sufficiently describe the system, and a degree of freedom is lost. This is referred often as a gimbal lock –problem. Aside from that, rotation matrix can represent any possible rotation, and therefore any possible orientation.

Next, a method for expressing combination of rotation and lateral translation is presented. In case of an object that contains many points, this combination can be called object's pose. A common way to do this is to use a transformation matrix:

$$T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

Transformation matrix T consists of a 3x3 rotation matrix (in top-left corner) defining the rotation, and three scalars, T_x, T_y and T_z defining the translation. Bottom row is required only to make sure that matrix multiplication can be used to easily calculate transformations, as with rotation matrices. Also the coordinate vector must be of form

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{15}$$

since transformation matrix size is 4x4.

Transformation matrix can be thought of combining two operations: rotation and translation. If a point is first rotated around the origin and then moved laterally, the resulting coordinate vector can be calculated with transformation matrix. Also, as with rotation matrices, multiplying several transformation matrices yield a transformation matrix that represents those transformations applied in multiplication order:

$$\boldsymbol{x}'' = T_2 * T_1 * \boldsymbol{x} = (T_2 * T_1) * \boldsymbol{x} \tag{16}$$

where $\boldsymbol{x}''$ is the point after transformation.

Transformation matrices can be also used to apply translation and rotation to group of points, or, an object. If the object's initial location and orientation are known, its location and orientation after the transformation can be calculated. This makes it possible to use a transformation matrix to store and represent information about object's location and orientation. Any changes in location or orientation can be represented by a transformation matrix as well.

2.4 Object detection and tracking

Detecting an object from an environment, and subsequently tracking its location, is a central problem in many applications. This section explains object detection and tracking from the perspective that there exists a point cloud of the environment.

There are different ways to detect objects from point clouds. The selection of method depends on the situation in general, but also greatly on what is known about the obstacle beforehand. If only a vague description of the object is known, the algorithm must be highly sophisticated, and if the object's properties are well known, less sophisticated algorithms may suffice.

If it is possible to create a point cloud from the object, it will be possible to use that for object detection and tracking. This approach leads to what is called point cloud registration, registration meaning the process of finding out transformation between two or more clouds acquired from one environment [1], [8]. A successful registration means that the point clouds are aligned, or that the features in the two clouds "match". This requires that other point cloud is a model of the object, while the other is a cloud from the environment including the object. Finding the translation between those two will give object's location within the environment. Tracking based on point cloud registration should work well with obstacle of any shape, provided that a reference model is available. If there is only a single object that needs to be tracked, having a guess about object's initial location can also help.

Other methods for obstacle recognition may be based on extracting specific features from the point cloud. For example, building facades can be reconstructed from a point cloud using feature detection [21]. These features may include sharp edges and corners, or certain shapes like a ball, cylinder or a plane. For example, in an application where the aim is to recognize and track pedestrians, a cylinder can be used as an approximation for the shape. This method is only good for situations in which obstacle is composed of relatively simple shapes. Obstacles with more complicated geometry will require more refined methods to extract the features.

Point cloud can also be segmented or clustered, e.g. separated into clusters that likely contain points from single object or feature. Clustering methods are used with two-dimensional point clouds, but according to Klasing [9] there is little research for three-dimensional cloud clustering. Segmenting can be done before other methods are used, which may help especially if the point clouds are large.

2.4.1 Iterative closest point -algorithm

Iterative Closest Point (ICP) is an algorithm for point cloud registration. It was first presented by P. J. Besl and N. D. McKay [7] [1]. This section aims to explain ICP and its properties.

An example of two point clouds matched with ICP is presented in Figure 1. The figure is based on screenshots from PCL visualizer. Point cloud on the left contains points that are sampled from a surface of a box, and point cloud on the right represents points sampled from the environment. The point cloud below is a combination of those two.

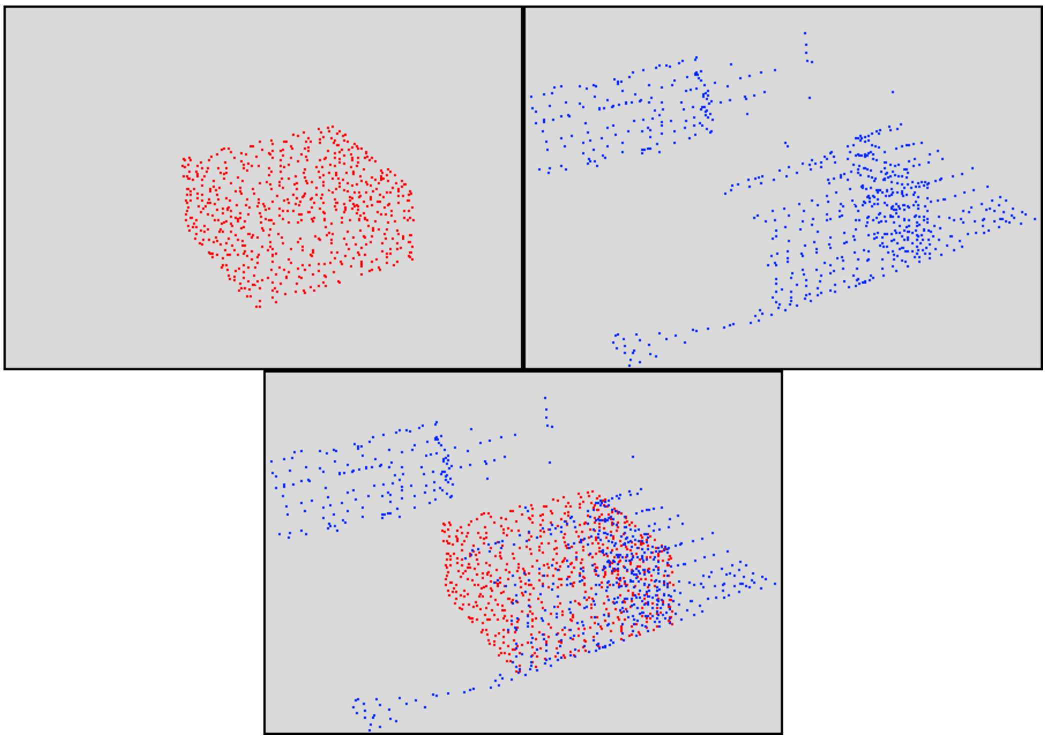

Figure 1. *Matching two point clouds with Iterative Closest Point-algorithm*

ICP takes two point clouds, let's say point cloud A and point cloud B, as its input. The algorithm has essentially two phases. The first phase iterates over every point in A, and for each point finds out the best corresponding point in B using correspondence metric. That is used to test how well the two clouds match. The most straightforward metric is the Euclidean distance between two points. If the point clouds are perfectly aligned and have reasonably high point density, for each point in cloud A there is supposedly a point nearby from cloud B. Therefore, the smaller distances between point pairs, the better matched the clouds are.

[1] Often, "Object modelling by registration of multiple range images" (1991) by Yang Chen and Gerard Medioni is mentioned as another source for ICP-algorithm.

Second phase is to calculate a rigid transformation that best minimizes the distance, or correspondence difference for each pair of points. Then, the transformation is applied to the point cloud, and the cycle is started over again: new corresponding pairs are picked, and new transformation is calculated. Each iteration should "move" the point clouds to a more aligned pose. ICP is executed until a defined threshold is achieved. It can be an upper limit for number of iterations, or a lower limit for how small the consecutive transformation can be.

The algorithm may include some additional restrictions. For example, it is possible to completely ignore point pairs whose initial distance is too high, as these are likely incorrect pairs anyway. Also, it is possible to use additional metrics for correspondence. One of them is using information about surface normal vector, if that is available.

2.4.2 Kalman filter

Kalman filter is a well-known technique to estimate process states in a situation where measurements are noisy. Thrun et al. [12], p.81 states that is was originated by Rudolf Kalman and Peter Swerling in 1960 and 1958, respectively. The filter and its derivatives are widely used in a variety of engineering problems, including obstacle tracking [11].

Theory for Kalman filter is available in detailed form by Thrun in [12] and Labbe in [13], the latter being especially designed for being easy to approach. It is not a major topic in this thesis, and thus only a superficial description is presented here.

Let $\boldsymbol{x}_k$ be a vector of n system states at time step k. Kalman filter assumes that states at time step k follow the equation:

$$\boldsymbol{x}_k = \boldsymbol{F} * \boldsymbol{x}_{k-1} + \boldsymbol{B} * \boldsymbol{u}_k + \boldsymbol{w}_k, \tag{17}$$

where $\boldsymbol{F}$ is a state-transition matrix, $\boldsymbol{u}_k$ a vector composed of system inputs if applicable, $\boldsymbol{B}$ the control-input model, and $\boldsymbol{w}_k$ a vector of process noise.

Kalman filter operates with state estimates rather than actual states. Vector $\widehat{\boldsymbol{x}}$ represents the state estimate. Based on the equation above, kalman filter predicts the state estimate from previous estimate:

$$\widehat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F} * \widehat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{B} * \boldsymbol{u}_k. \tag{18}$$

Additionally, a matrix $\boldsymbol{P}$ containing estimated state error covariance, is defined. Matrix $\boldsymbol{P}$ is similarly predicted:

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F} * \boldsymbol{P}_{k-1|k-1} * \boldsymbol{F}^T + \boldsymbol{Q}, \tag{19}$$

where $\boldsymbol{Q}$ is covariance of the process noise.

Equations 18 and 19 form the first step of kalman filter algorithm, and are called prediction step. This step basically tries to predict state values and their error covariance based on previous data and system model.

The second step also takes measurements into account, and adjusts estimates accordingly. First, a residual $\widetilde{\boldsymbol{y}}_k$ is calculated from state estimate $\widehat{\boldsymbol{x}}_{k|k-1}$ and measurement $\boldsymbol{z}_k$:

$$\widetilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H} * \widehat{\boldsymbol{x}}_{k|k-1}, \tag{20}$$

Above, matrix $\boldsymbol{H}$ is observation matrix, and it defines the observation model that represents relation of measurements to states. Then, Kalman gain $\boldsymbol{K}_k$ is calculated as follows:

$$\boldsymbol{S}_k = \boldsymbol{H} * \boldsymbol{P}_{k|k-1} * \boldsymbol{H}^T + \boldsymbol{R}, \tag{21}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1} * \boldsymbol{H}^T * \boldsymbol{S}_k^{-1}. \tag{22}$$

Matrix $\boldsymbol{R}$ is covariance of observation noise. The state estimate is updated using newly-calculated Kalman gain:

$$\widehat{\boldsymbol{x}}_{k|k} = \widehat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k * \widetilde{\boldsymbol{y}}_k. \tag{23}$$

Also, estimate of state error covariance is also updated:

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k * \boldsymbol{H}) * \boldsymbol{P}_{k|k-1}. \tag{24}$$

Equations 20 - 24 form what is usually called the update step of the Kalman filter. As the name could imply, update step updates state estimate and error covariance acquired in prediction step, using new information provided by measurements.

3. SETUP

In this chapter, hardware components of the tracking system and its overall structure are presented. The purpose of this system is to execute the actual tracking process and to have an option to measure how well it performs. The entire architecture is presented in Figure 2, and the subchapters here are arranged similarly. Figure 2 also shows the communication buses between system components.

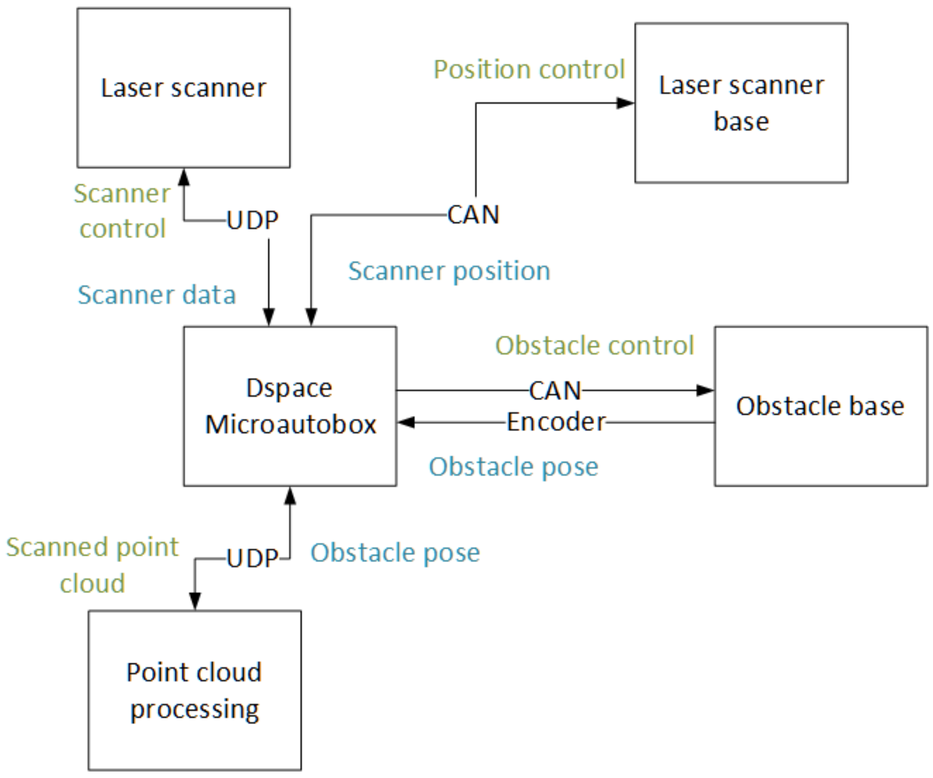

Figure 2. *Full system architecture*

The system contains a laser scanner and laser scanner base to get point cloud data, a point cloud data processor to track the obstacle, and a moving platform that serves as obstacle base that can be used to move obstacle and to verify tracking results. A dSpace Microautobox functions as a central processor and provides the communication interfaces necessary for other components. Experiments in which the system is tested are described in chapter 4.

3.1 Laser scanner

Scanner used in this project is a SICK LMS511-20100 PRO laser measurement sensor. It measures distances based on time-of-flight –principle, and has maximum range of 80 m. Scanner operation is based on a rotating mirror, which deflects the laser beam to desired direction. A scan is executed at each full rotation of scanner's rotating mirror, and contains measured distances to points within scanner range. The actual measuring area can either be scanner's maximum field of view, or be set by user. This is shown in Figure 6. Scanning frequency, determining how often scans are executed can be also chosen by user. Higher frequencies require lower angular resolutions. The frequency-resolution combinations available are presented in Table 1.

Table 1. *Laser scanner resolution and frequency options [5]*

Scanning frequency	Angular resolution	Max. Distance
25 Hz	0.1667°	65 m
25 Hz	0.25°	80 m
35 Hz	0.25°	65 m
35 Hz	0.5°	80 m
50 Hz	0.3333°	65 m
50 Hz	0.5°	80 m
75 Hz	0.5°	65 m
75 Hz	1°	80 m
100 Hz	0.6667°	65 m
100 Hz	1°	80 m

Scanner output is distance values in millimetres, and the data type is 16-bit unsigned integer. Distance values are sent enclosed in UDP packets, one packet containing data for one scan Accuracy of distance measurement is up to $\pm\ 25\ mm$ [5]. Figure 3 shows both laser scanner and the rotating base it is attached to.

The maximum range of the laser scanner is over 60m, and higher that the project would require, as the scanner and obstacle are placed within few meters from each other. This means that the scanner will get data from an area around the working area. Fortunately, most of the generated points can be directly discarded, because the obstacle's approximate position and movement area limits are well known beforehand.

3.2 Scanner base

Scanner is mounted on a horizontal rotating shaft, which is rotated by an electronic motor. There is an absolute encoder attached to the end of the shaft to provide data about shaft angle. Motor is driven by a motor controller. The components are listed in Table 2, and most of them are visible in Figure 3. The motor controller is given commands via CAN bus. Also, the encoder output is read via CAN bus.

Table 2. *Scanner base components*

Motor controller	Maxon ePos 2 24/5
Motor	Maxon brushed DC motor
Gear	Maxon planetary gear, ratio 113:1
Absolute encoder	Posital Fraba, 16 bit

The scanner shaft is rotated alternating the movement between clockwise and counter-clockwise. The result is a motion best described as nodding, and a similar concept is used in [15]. End position angles can bet set freely, but are limited somewhat by the base assembly structure. Nodding motion range directly defines the vertical field of view of the tracking system, and was approximately 30 degrees while testing.

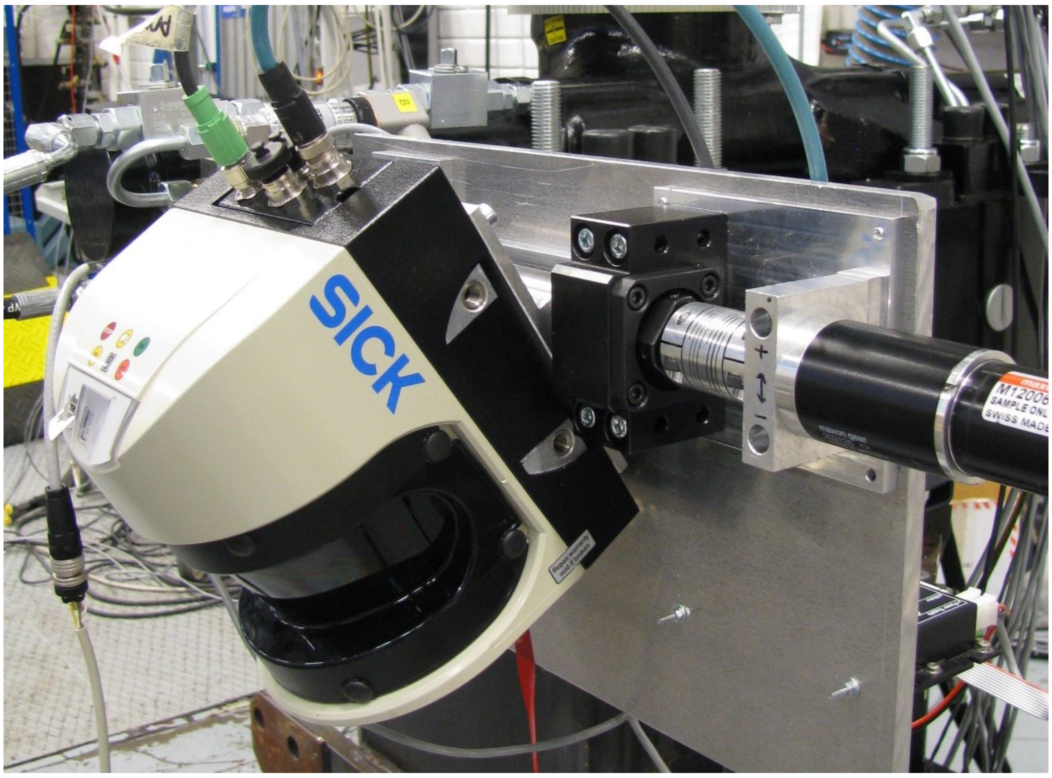

Figure 3. *Laser scanner and rotating base*

3.3 Obstacle base

In order to test functionality of tracking, there is a moving platform to which the object is attached. The platform serves two functions: First, it gives an easy way to move the object in a controlled manner, and second, it gives accurate data about object position at the same time. Tracking results can be verified by comparing position data against tracking data. Part of the moving platform is shown in Figure 4.

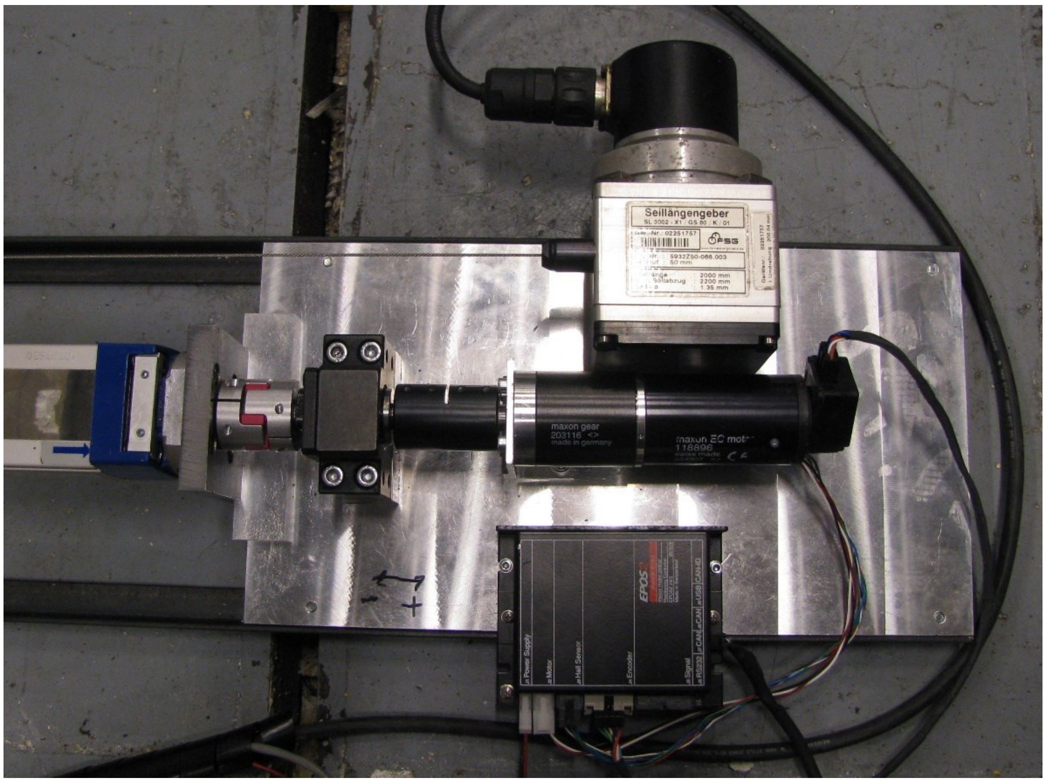

Figure 4. *Linear slide motor and position measurement device*

Moving platform consists of an electric motor, motor controller and a linear slide, which is operated by the motor. A cable length measurement device is used to measure position of the linear slide. The motor assembly and one end of the linear slide are shown in Figure 6. The components are also listed in Table 3.

Table 3. *Moving platform components*

Motor controller	Maxon ePos 2 24/5
Motor	Maxon brushless DC motor
Gear	Maxon planetary gear, ratio 15:1
Cable length measurement device	Pepperl+Fuchs SL3002-X1/GS80
Rotary encoder	Pepperl+Fuchs RVI58N

Motor controller is a similar one than used in the laser scanner base. Similarly, CAN is used as a primary communication method with it; that is, the operating commands to the motor are sent via CAN. Linear slide position measurement is realized by a retracting cable that is wound around a drum. Drum rotation is measured by a rotary encoder. Encoder output is a standard two-channel encoder pulse output.

3.4 Microautobox and Simulink model

CAN communication, laser scanner data gathering and encoders' measurements are done by a dSpace microautobox 1401. CAN communication is used to send drive commands to motor controllers for both scanner rotating base and linear slide. CAN is also used to read absolute encoder data from scanner rotation. Laser scanner transmits its scan data via UDP, which is then read by Microautobox and converted to point coordinates according to calculations in section 3.4.1. Microautobox also sends the point data forward via Ethernet/UDP connection.

Usage of the microautobox is closely tied to matlab. The software was developed by compiling the Simulink model into a real-time executable for the microautobox.In this project, the Simulink model incorporates the following: laser scanner data reading, encoder data reading, calculating coordinates based on the former two, and finally filtering of the pose measurement. In addition to that, it does the most of additional communication duties, like sending appropriate drive commands to the motors.

3.4.1 Kinematics

Scanner was rotated to get three dimension scans. Absolute encoder provides accurate data of scanner rotation angle ϕ. Combining that with rotating mirror angle θ and distance d for each point, it is possible to calculate Cartesian coordinates for each point.

Coordinates of the points in point clouds will be calculated in relation to one global base coordinate system. Coordinates will be acquired by taking into account rotation of laser scanner internal mirror, offsets of scanner base, scanner rotation, and finally offsets to global base coordination from scanner base. Figure 5 presents an overview of the basic idea behind calculations below and simplified representation of the scanner system.

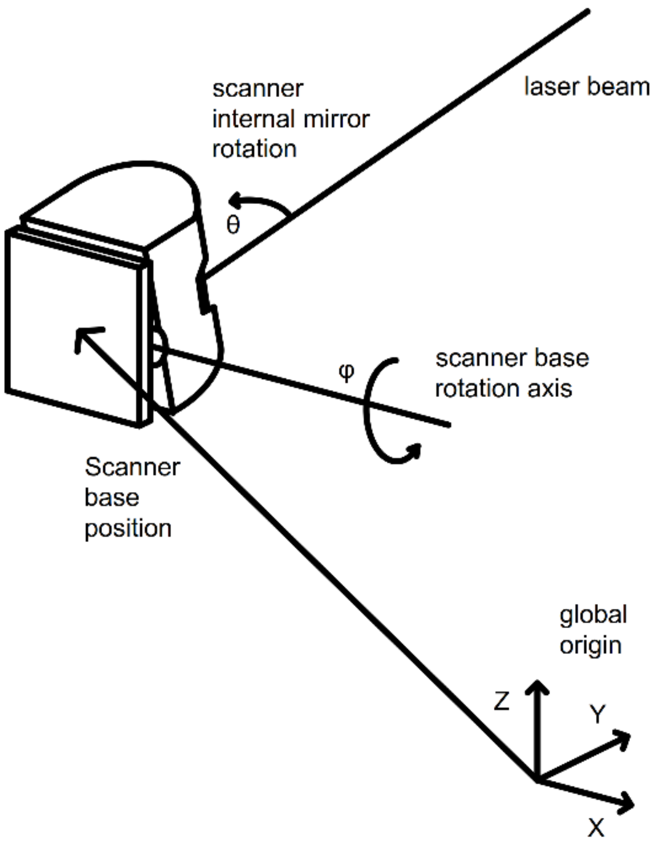

Figure 5. *Scanner base in relation to global origin*

Measured points of one scan are located on a plane, since the scanner operates in 2D. The scanning direction is tied to scanner pose. Angle θ for each point along scanner's internal mirror rotation can be calculated if user-defined measurement area and scanner's angular resolution are known. Figure 6 shows a not-in-scale example of laser measurement in relation to measurement area and scanner maximum measurement area.

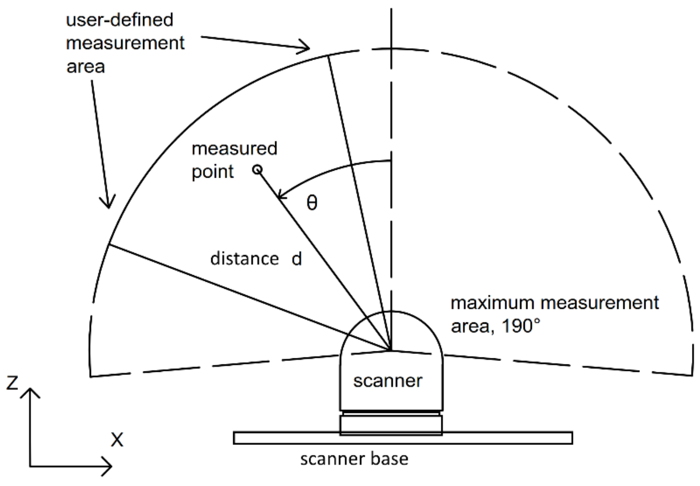

Figure 6. *Laser scanner measurement area, scanner drawn pointing upwards*

Scanner internal mirror angle θ and scanner base rotation angle ϕ are set so that when both would be zero, the laser would point straight upwards. This leads to rotation θ being around y-axis and rotation ϕ being around x-axis. Rotation matrices R_θ and R_ϕ for θ and ϕ can be then defined as follows:

$$R_\theta = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{25}$$

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{26}$$

Transformations from global origin to scanner base origin T_{base} and from scanner base origin to scanner optical origin $T_{scanner}$ are both translations, so the orientation won't change. They can therefore be defined as x, y and z offsets. Translation matrix $T_{scanner}$ for scanner optical origin offset from scanner base origin, shown in Figure 7, is simply

$$T_{scanner} = \begin{bmatrix} 0 \\ 0.027 \\ 0.117 \end{bmatrix} (m) \tag{27}$$

And translation matrix T_{base} (Figure 5)

$$T_{base} = \begin{bmatrix} -0.41 \\ 0.46 \\ -0.15 \end{bmatrix} (m) \tag{28}$$

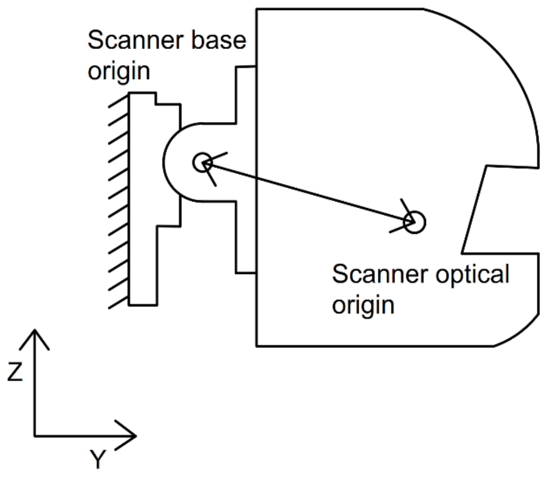

Figure 7. *Scanner optical origin in relation to scanner base origin, side view*

As laser is considered to point upwards before applying any rotation or transformation, the distance value d represents z-coordinate of initial coordinates X_{init}:

$$X_{init} = \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix} \tag{29}$$

Final coordinates X_{final} can be calculated by first applying rotation R_{θ}, then translation $T_{scanner}$, then rotation R_{ϕ} and finally translation T_{base} to initial coordinates X_{init}:

$$X_{final} = T_{base} + \{R_{\phi} * [T_{scanner} + (R_{\theta} * X_{init})]\} \tag{30}$$

3.4.2 Rotation compensation

Rotating movement of the scanner causes two problems, which have to be taken into account. First, as there is certainly a delay between scanning and sending the data via UDP, the encoder data does not correspond fully to scanner data. Second, as scanner's internal rotating mirror has finite speed, different points in single scan are actually scanned in different time and therefore different scanner shaft angle. The latter won't be too much of a problem, if rotation speed is low enough. The former, however must be compensated, especially if the point cloud is to contain points from scans in which the rotation direction is different. If rotation direction and speed would be constant all the time, the delay could be compensated by adding offset into angle measurements.

The chosen scanning frequency for laser scanner was 75 Hz. It is assumed that this is also the rotation frequency for scanner's internal mirror. The duration T_{rot} for one full rotation is then

$$T_{rot} = \frac{1}{75\ Hz} = 13.3\ ms. \tag{40}$$

As the scanner has a field of view of 190 degrees, the time T_{FOV} between the first and the last point of single scan is

$$T_{FOV} = \frac{190}{360} * T_{rot} = 7.0\ ms. \tag{41}$$

This is a bare minimum estimate for full delay; there is also delay originating from the internal processing time within scanner, and delay from communication, in this case UDP.

Nodding movement periodic time was 0.6 seconds, and the sequence had a motion pattern that resembles sine wave. The rotation speed ω_{rot} at its maximum was

$$\omega_{rot} = 3.09\frac{rad}{sec}. \tag{42}$$

This value was manually set at the motor controller, and is a result of sine-like motion pattern and the aforementioned 30-degree angle difference.

If we assume an arbitrary delay T_{delay} of 10 ms, the positional error x_{err} would be approximately

$$x_{delay} = 2\,m * \sin\left(\omega_{rot} * T_{delay}\right) = 0.062\,m, \tag{43}$$

assuming also arbitrary obstacle distance of two meters from scanner.

The problem was clearly visible while doing initial testing with the scanner system, and applying a correction did provide clear improvement to the quality of the point clouds. Unfortunately, the exact quantity of delays are difficult to find out. The solution here was to pick a compensation value by hand, and adjust it so that the point cloud looked as correct as possible.

In this application it is assumed that the delay for sending measurements of each scan is constant. Also is assumed that sample time of angle ϕ measurement is higher than the delay. Compensation for delay is made so that instead of using the latest measurement for angle (ϕ_t), a weighted average of two most recent values (ϕ_t and $\phi_{t_{-1}}$) is used:

$$\phi_{compensated} = w_{old} * \phi_{t_{-1}} + w_{new} * \phi_t \tag{44}$$

$$w_{old} + w_{new} = 1 \tag{45}$$

where w_{old} is weight of old measurement, w_{new} is weight of new measurement and $\phi_{compensated}$ is delay compensated value of angle measurement. This results a linear interpolation, which is a good approximation especially if rotating speed is near constant.

A fundamental problem with rotating scanner is that there are contradicting goals to optimize shaft rotation speed and therefore nodding motion periodic time. Obviously, if the shaft rotation is faster, the scanner can get data from the scanning region faster, and overall response time to changes in environment, or in this case more specifically to movement of the obstacle, is faster. However, lowering rotation speed will reduce the need to compensate for delays between rotation angle and laser scanner measurements. Thus, the higher the rotation speed, also higher is the need to artificially compensate measurement delays. There is significant pressure to make rotation speed as high as possible, since it is a major source of the overall system delay.

The system also features a compensation for scanner rotation between individual laser pulses. The error that comes from that would be quite small; given frequency of $75\ Hz$ and resolution of 0.5°, the time T_{pulse} between two pulses is

$$T_{pulse} = \frac{1}{75\ Hz} * \frac{0.5°}{360°} = 18{,}5\ \mu s \tag{46}$$

and positional error as above

$$X_{pulse} = 2\ m * \sin(\omega_{rot} * T_{pulse}) = 0.114\ mm \tag{47}$$

3.4.3 Kalman filter implementation

The point cloud registration gives obstacle pose estimate as a 4x4 transformation matrix. It was noticed that the estimate, while otherwise reasonably accurate, was quite noisy. A Kalman filter, described in chapter 2.4.2, was constructed to get rid of excess noise in the position estimate. The filter was implemented in the Simulink model running on micro-autobox. It could have been incorporated into the linux pc as well, but doing it in matlab was more straightforward.

As Kalman filter works with linear state and observation models, there is actually no guarantee that a Kalman filter would be a best solution for this project. Especially obstacle rotation model is certainly nonlinear, so a filter with better capability to handle nonlinearity, like an extended Kalman filter, would most likely be a better solution. However, there are a few reasons why Kalman was chosen: Plain Kalman filter was sufficiently easy option to implement, and it proved improved results even while likely not optimal. Also, designing the best possible filter specifically for tracking was not a main objective of this thesis, and in that sense using resources for other, possibly better solutions was not necessary. If the project would be addressed more carefully, and if requirements for obstacle tracking results would be stricter, researching for other types of filters would have a higher priority.

The system was modeled around obstacle motion. Obstacle was considered to move in any direction, and to have any arbitrary rotation. The x, y, and z – coordinates were filtered individually, and rotation was filtered with one filter. Filtering rotation was done by first calculating elementary rotation angles from the 3x3 rotation matrix, and using them as measurement with the Kalman filter to estimate said angles.

System states were obstacle position, velocity and acceleration in each direction, along with rotation velocities and accelerations around x-, y-, and z-axes. Position, velocity and acceleration are here denoted by x, $\dot{x}$, $\ddot{x}$, y, $\dot{y}$ and so on; rotation angles and angular velocities by $\alpha, \dot{\alpha}, \beta, \dot{\beta}, \gamma, \dot{\gamma}$.

Kalman filter system states for position estimate were (only the filter for x-direction is presented here, y and z were fully similar)

$$\boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix}. \tag{48}$$

The system model for Kalman filter assumed constant acceleration, therefore system matrix $\boldsymbol{F}$ would be

$$\boldsymbol{F} = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & s \\ 0 & 0 & 1 \end{bmatrix}, \qquad (49)$$

where s was system sample time. This motion model does not describe the real situation perfectly: Even though acceleration was most of the time constant, zero or nonzero, it would change occasionally. The only way for the model to take that into account is via process noise.

Matrix $\boldsymbol{B}$ contained only zeroes, since this system is considered to have no inputs. For process noise covariance $\boldsymbol{Q}$, a suitable value was found with experimenting. The same was true for observation noise covariance $\boldsymbol{R}$. The following values were used for $\boldsymbol{Q}$ and $\boldsymbol{R}$:

$$\boldsymbol{Q} = \begin{bmatrix} 1 * 10^{-6} & 0 & 0 \\ 0 & 1 * 10^{-6} & 0 \\ 0 & 0 & 1 * 10^{-6} \end{bmatrix} \qquad (50)$$

$$\boldsymbol{R} = 0.1 \qquad (51)$$

A separate Kalman filter was added for orientation estimate smoothing, represented by elementary rotation angles α, β and γ. States of the system were angles and angular velocities, and the system model assumed that the angular velocities would stay constant.

$$\boldsymbol{x} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} \qquad (52)$$

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 0 & s & 0 & 0 \\ 0 & 1 & 0 & 0 & s & 0 \\ 0 & 0 & 1 & 0 & 0 & s \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (53)$$

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} * 10^{-7} \tag{54}$$

$$\boldsymbol{R} = 0.25 \tag{55}$$

3.5 Point cloud processing

This section focuses on processing of actual point clouds. There are two main areas to govern: hardware and software and algorithmic details. The hardware was realized as a virtual PC running Ubuntu Linux. Software and therefore algorithms were implemented using Point Cloud Library.

3.5.1 Point cloud construction

Points obtained from a single scan of the laser scanner lie at a plane, and also the amount of them is very low, 761 at maximum [5]. Therefore points from multiple scans taken with different scanner orientation must be combined to form a three dimensional point cloud. This is also needed in order to have enough points to locate and track the object. The real time constraints require that point cloud must be kept constantly updated.

In this project, the point cloud used basically contains a buffer of fixed amount of previous scans. Updating the cloud can be done simply by inserting points from new scan into cloud and discarding points that are from the oldest scan. In this approach, the relevance of data inside point cloud varies, depending on how old the point in question is.

3.5.2 Linux Pc

Point Cloud Library is essentially a collection of C++ code. The environment to run it under was chosen to be a PC with a linux as operating system. Its operating system was Ubuntu 14.04, which was chosen for its long time support. Code was compiled using tools shipped with default Ubuntu installation. Operations done with PCL include forming point cloud from points of multiple scans, using ICP-algorithm to register scanned point cloud and reference model point cloud, and storing information about current estimate of tracked object position.

The PC was a virtual PC running under windows operating system. It does not fulfil strict real-time requirements. Ideally, a real-time computer, for example a sufficiently powerful soft-PLC, or a real-time Linux kernel, would be more suitable for this kind of task. It was apparent that the capability of the virtual machine was not as high as would have been desired. However, problems that occurred were not such that they would be directly caused by the environment being non-real-time.

3.5.3 Point cloud library

Point cloud library (PCL) is an open-source C++ library providing operations for point cloud data processing. PCL was presented in 2011 by Radu B. Rusu and Steve Cousins [1]. The latest version at the time of writing this thesis was 1.8, which was first released in 2016. In this thesis, however, version 1.7 was used.

PCL has a multitude of features. It provides means to store and visualize point clouds, and has many point cloud processing algorithms implemented. In this thesis the PCL is used to store point cloud data, filter out points in located unwanted area, down sample point clouds to smaller amount of points, and to register two clouds using ICP-algorithm. All these features have an existing implementation in PCL, which made it an easy task to use them.

3.5.4 Iterative closest point

In this project the tracking realized by registration of two point clouds, as mentioned above. There is a variety of algorithms available for point cloud registration. One of them is ICP (Iterative Closest Point) [7]. It is default point cloud registration function in Matlab, and Mitra [7] mentions it being "popular". Basic functionality of ICP is introduced in subchapter 2.4.1. Its purpose is essentially to find the rigid transformation between two point clouds that most accurately "matches" their points.

There are also other algorithms for similar task, that do not focus on points, but rather some features that are estimated from a point cloud. They tend to be more complicated, and might require different strategies depending on the features that the user expects to find. For example, if the scene contains a lot of straight surfaces, it could be beneficial to look for them. ICP, On the other hand, does not care about underlying geometry. There is also an option to use surface normal vectors in addition to just point coordinates. An example of point cloud with surface normals is shown in Figure 8. ICP was chosen here over other algorithms due to its ease of implementation, and because it proved to be fast enough for the hardware used.

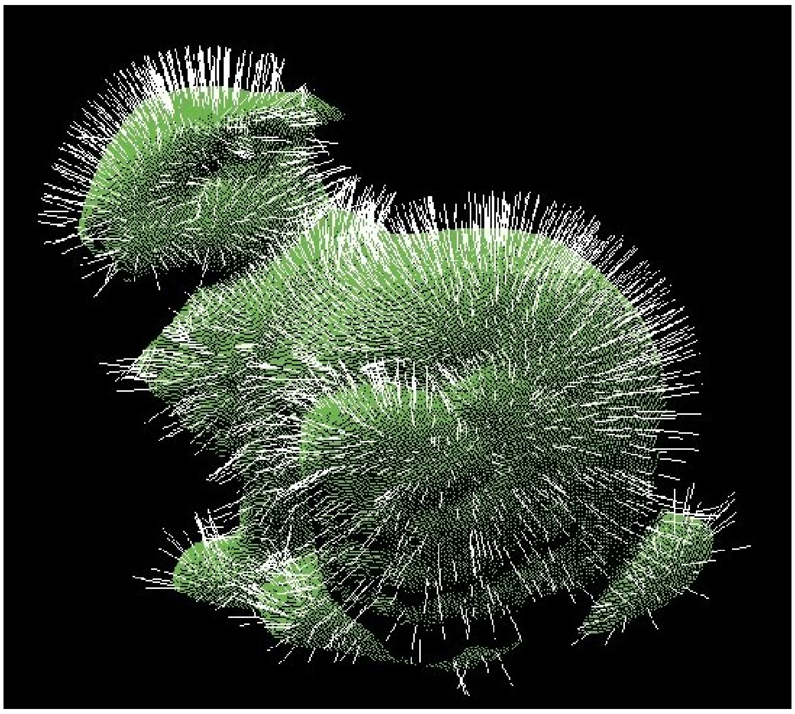

Figure 8. *A point cloud with normal vectors shown, taken from [20].*

The output of ICP-algorithm in PCL is a 4x4 transformation matrix. The matrix represents the reference point cloud's position and orientation after matching it with the environment. If the initial position and orientation are known, the current position and orientation can be calculated with the transformation matrix. An example of two point clouds matched with ICP is shown in Figure 1.

The ICP-algorithm allows adjusting of several parameters: maximum number of iterations, minimum transformation threshold, minimum distance threshold, and maximum correspondence distance. Maximum number of iterations is simply an upper limit for iterations, after which the algorithm is expected to be converged. Minimum transformation threshold also gives a limit for how long the algorithm runs: if point clouds are already matched well together, the new transformation is very small and algorithm can be assumed to have converged. The same goes for minimum distance threshold: if point clouds are well matched, the distance between corresponding points is small and algorithm is considered to be converged.

3.6 Communication overview

Distance data is transmitted from the laser scanner to microautobox over UDP protocol. The laser scanner data sheet specifies the details on how the data structure is formed [5]. Essentially one scan produces one UDP packet, in which there are some general configuration parameters, and the distance values for each configured points one after another.

UDP is also used in communication between microautobox and linux-pc. Microautobox sends simply the three coordinates for each point, but due to technical limitations the coordinates have to be sent in two packets.

The rotary encoder information is accessed via a can bus. CAN is also used to control laser scanner rotation and the linear slide. Both of them utilize a similar Maxon ePos motor controller, which uses CANopen as its communication protocol. The CAN messaging happens mostly in one direction, that is, the microautobox sends control messages as needed, and doesn't mostly care about what the devices send back. CANopen messages are composed within the simullink code that runs on microautobox.

4. CONDUCTED EXPERIMENTS

This chapter contains description about the experiments with which the tracking system was tested. The test setup overview is presented in **Error! Reference source not found.**. I t consists of a laser scanner, a scanner rotating base, and a moving platform with obstacle.

Experiments were executed by running the tracking system, and simultaneously moving the obstacle using the moving platform. Position measurement from moving platform was recorded along with tracking system's estimate about the position. This allowed the comparison of the two values, and therefore evaluation of the tracking performance.

There were two different experiments. The main difference between the two is that the implementation in first one does not have any additional smoothing of position estimate. Instead, the tracking estimate is directly data from point cloud matching algorithm. Second test featured an additional Kalman filter smoothing the position estimate. Other parameters and test setup overall were largely similar between the two experiments.

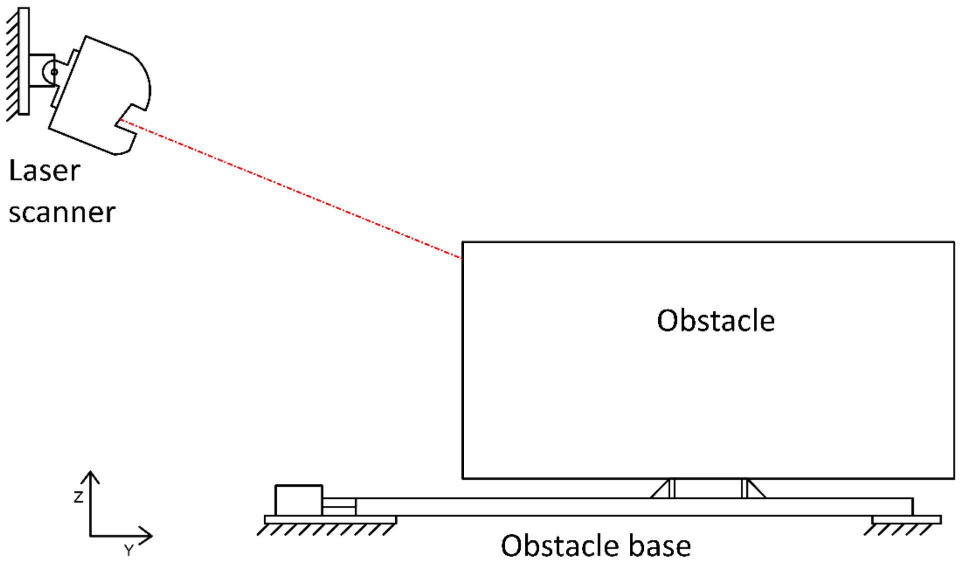

Figure 9. *Overview of the test setup*

The goal of the experiments was to get object tracking work in some level. No speed constraints for object movement were given, and also tracking rate (response time) was accepted to be low. The results are reviewed in chapter 5.

4.1 General properties of the experiment and setup

The experiments were conducted using the system setup described in chapter 3. Using single test setup means that most properties were similar throughout all of the experiments. Some minor details did vary, and variations are listed in this chapter. Also, some details could have been varied, but were kept similar.

4.1.1 Reference cloud

The obstacle used in these experiments was an actual rectangular box, so it was easy to make a reference point cloud based on its dimensions. The reference cloud contained randomly sampled points at all six faces of an ideal box shape. The amount of points on each surface was the same, which resulted different point density at each surface; however, applying down sampling to the cloud did lessen this effect.

As the scanner gets points from the surface of the environment, the reference had only points at box surface. Depending on the obstacle orientation, the scanner can only see one side of the obstacle at a time. The obstacle was positioned so that three faces were visible towards the scanner at all times. Also, the viewing direction in relation of the obstacle did not change while obstacle moved. This did likely make position tracking easier, opposed to situation where only two or one sides of the obstacle would be visible.

4.1.2 Initial pose

With all of the tests, the obstacle's initial pose was known to a certain degree. All three coordinates x, y and z were known with an error of less than 0.5m, and orientation with errors in either rotation angle of less than 20 degrees. Within the limits of this project, knowing initial pose is an acceptable assumption. This might not be the case with real applications.

Knowing the initial pose was critical part of the success in the first place. If the initial pose was too much incorrect, the registration algorithm had a tendency to either not converge at all, or to converge to pose that did not correspond with obstacle's real location.

4.1.3 Measurements and Simulink model configuration

Acquiring measurements was done using Microautobox and ControlDesk-software. The software provided means of recording values of any variables from the Simulink model. In these experiments, the following variables were recorded: Obstacle pose from the tracking system, obstacle pose after applying a filter, and position measurement from moving platform. Obstacle pose was represented as a 4x4 matrix in both cases, and moving platform position was a scalar value. Sampling frequency for the measurements was determined by the Simulink system sample time, which was 4ms.

4.1.4 Linear slide movement, test sequence

The tests were conducted using the linear slide to move obstacle, and to get accurate measurement from the position. While the system would, in theory, be able to track all kinds of movements including rotation, this setup had some limitations. Like the name implies, linear slide can provide only linear motion. Moving obstacle with non-linear track was therefore not tested. Another uncertainty will remain on whether and how well the system could track obstacle rotations. This couldn't be tested, because the linear slide was not able to rotate the obstacle. Linear slide was positioned so that its movement direction matched as close as possible to one of the selected global coordinate frame axes. This was done because it would help analysing the recorded data, since only one coordinate value in estimate should change. Moreover, the orientation estimate should stay constant.

The linear slide had a fixed maximum speed, and tracking performance above that could not be tested. The test sequence consisted of movement from middle position to one end of the slide, then movement to other end of the slide and then movement back to starting position in the middle. The movement speed was constant and similar in each experiment, though the direction changed.

Moving platform was moved from one end to another, according to a certain sequence. Sequence consisted of a constant acceleration to a constant speed, and a similar deceleration to halt. Maximum speed in either direction was about 0.1 m/s. The sequence duration was about 30 seconds. During the sequence, tracking system was active and provided estimate of the obstacle pose.

4.1.5 Adjustable parameters

There are quite many parameters that could be adjusted in the testing process. They can be sorted into different categories based on which system component they affect. The linear slide, for example, can have different motion profiles, of which the ones that were used are described above. Laser scanner has scanning settings, PCL has parameters for ICP-algorithm and point cloud configuration etc. The most important ones for the tests are listed in this section.

Laser scanner was set to operate at a scanning frequency of 75 Hz. Scanning resolution was 0.5 degrees, and horizontal field of view was limited between 0 and 90 degrees. Frequency and resolution are connected so that higher resolution would mean lower frequency. Resolution directly affects horizontal point density, and frequency respectively affects vertical point density. The chosen combination brought sufficient amount of points in horizontal direction while maintaining a sufficient resolution in vertical direction.

Nodding movement of the scanner had a periodic time of 0.6 seconds. Movement range was approximately 30 degrees. Movement profile was nearly sinusoidal. The period was detected as a sweet spot addressing scanning vertical resolution, overall tracking response time and also mechanical properties. Trying a lower period with similar range resulted in higher rotation velocity, and eventually slippage of the scanner shaft.

A number of most recent scans were combined as a point cloud, which was further used in tracking. The amount of scans was, in both tests, 45. This appeared to be a good choice, as it resulted a buffer length being same as the period of the nodding movement:

$$\frac{45}{75\,Hz} = 0.6\,s$$

This way, each point cloud would theoretically contain a similar set of data from the environment. If this were not the case, the buffer would have different areas from the environment scanned with different density or not at all, depending on at which point of the nodding cycle the buffer is sampled.

The point cloud was filtered using the methods mentioned in 2.2.1. Because only a certain region of the scanned environment was interesting, point cloud was pass-through-filtered to contain only the interesting region. Region was about 3m wide, 3m long and 1m high, and was located so that the obstacle was about in the middle of it. In addition to that, both the reference cloud and the scanned cloud were down sampled in order to reduce the amount of points and thus computational load with ICP-algorithm. Down sample filtering density was 0.09m in all of the experiments. A noticeable lag was observed if the points were sampled more densely, which hints that the point density does affect computation time, and that the system processing power is an important design aspect.

4.2 Test with smoothing

Another experiment featured similar motion sequence with moving platform. The difference was that the Kalman filter application was used to reduce output variations. The system model and covariance matrices used with the Kalman filter are presented in section 3.4.3. The system model was relatively simple, and assumed that there was constant acceleration and rotation. Values of covariance matrices was a result of trial and error, and it is unlikely that they are very well optimized for the solution. This is an area with a lot of opportunities to improvement; however, even with sub-optimal parameters an improvement of results was achieved.

5. RESULTS

This chapter references the results that were achieved with tests depicted in chapter 4. There were essentially two tests: one with a tracking implemented, using point cloud matching with ICP, and second with otherwise similar configuration, but enhanced with a kalman filter to smooth position estimate noise. A general idea about system performance can be derived from the first data set, and the second is the best result that was achieved with resources and methods used in this thesis.

There was a relatively large problem with getting points from all possible three faces of the box. Because box's top face was close to laser scanner height, the scanner could not have accurate measurements from the top face. This made point cloud matching more difficult, and subsequently the result worse. If the obstacle would be allowed to move freely, it would be possible that there are even less faces visible towards the scanner. In optimal situation, the system would be so robust that it could work with such issues as well.

It was seen that the initial guess of the pose estimate needs to be relatively close (e.g. within 10-20 cm) from the true value. Otherwise the algorithm might easily find a local minimum that is not a valid match. Once that happens, it is very unlikely that the pose estimate ever converges to correct value, because the previous, bad result is given as an input for new iteration and algorithm will end up in the same local minimum.

Increasing both buffer size and rotation period to a similar amount did lessen the disturbations with pose estimate. There were still some unknown sudden changes while the obstacle was moving, but with stationary obstacle the result was clearly better. Lowering the obstacle speed did reduce measurement noise as expected; looks like noise is relative to the movement speed.

Results for both the case with only the tracking algorithm as the estimation source, and the case with additional filter that was to smooth the measurements are presented here. It is worth noting that those are two entirely different measurements, and therefore for example the obstacle moving sequence is not completely identical between them.

This is also visible in some of the measurements, where the average value is not same even though it theoretically should be.

First, Figure 10 shows both cases' position estimates of y-coordinate, along with a position measurement obtained from the linear slide moving the obstacle. These two together allow the evaluation of the tracking performance.

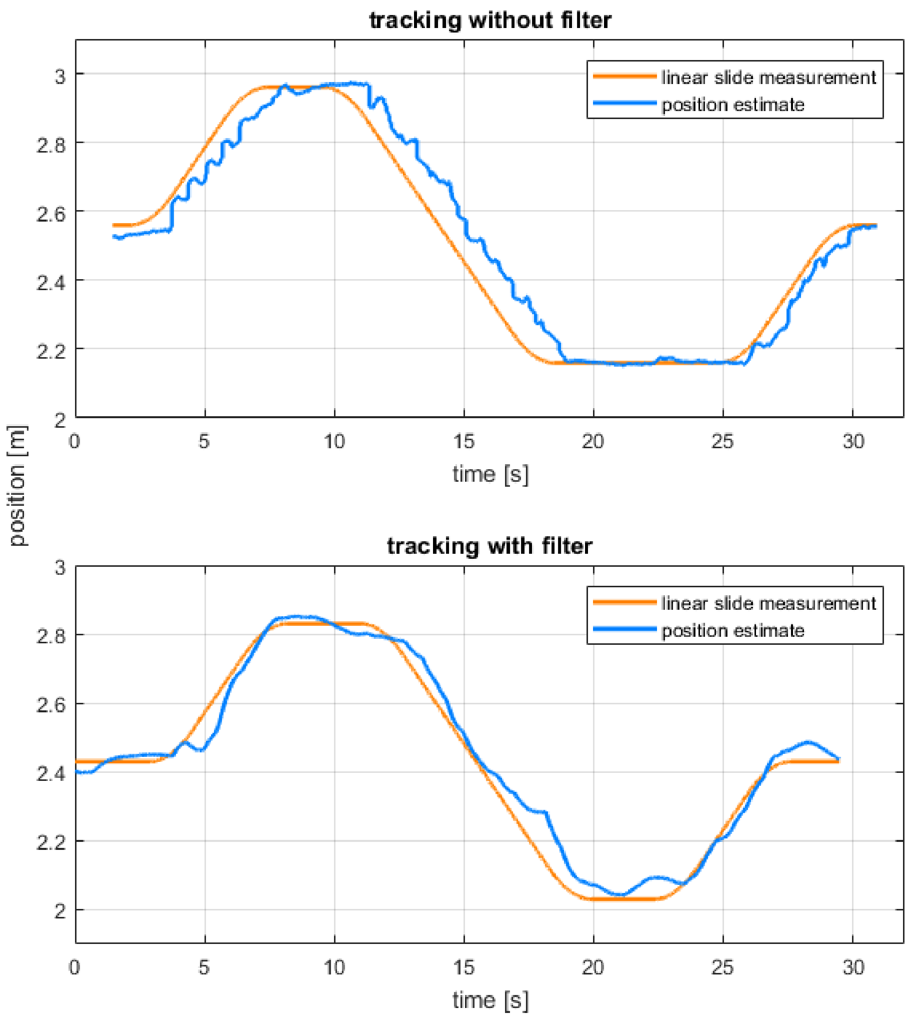

Figure 10. *Position measurement and tracking estimate*

There was a significant amount of noise in the resulting pose estimate. A position estimate in one dimension could jump suddenly up to 10 cm. Another problem was that there was a certain delay after the estimate would react to real obstacle movement. The delay was approximately 0.5 seconds, and that would result a locational error being approximately 0.05 m, as obstacle speed was 0.1 m / second.

It can be seen that using a Kalman filter does suppress some of the estimate noise, especially when measuring the direction of obstacle movement. However, there is still visible delay of approximately one second. Also, delay and the presence of some noise still leads to tracking error of up to more than 0.15m.

Overall, based on the result in Figure 10, the tracking appears to work. Position estimate follows actual position with some accuracy, and at a quick glance the biggest issue seems to be that there is delay between position estimate and linear slide position measurement. The linear slide measurement is assumed to be accurate and to have insignificant delay.

The error between linear slide measurement and the estimate, shown in Figure 11, reveals some important issues: the estimate, even though it would be correct in average, varies fast within a short period of time.

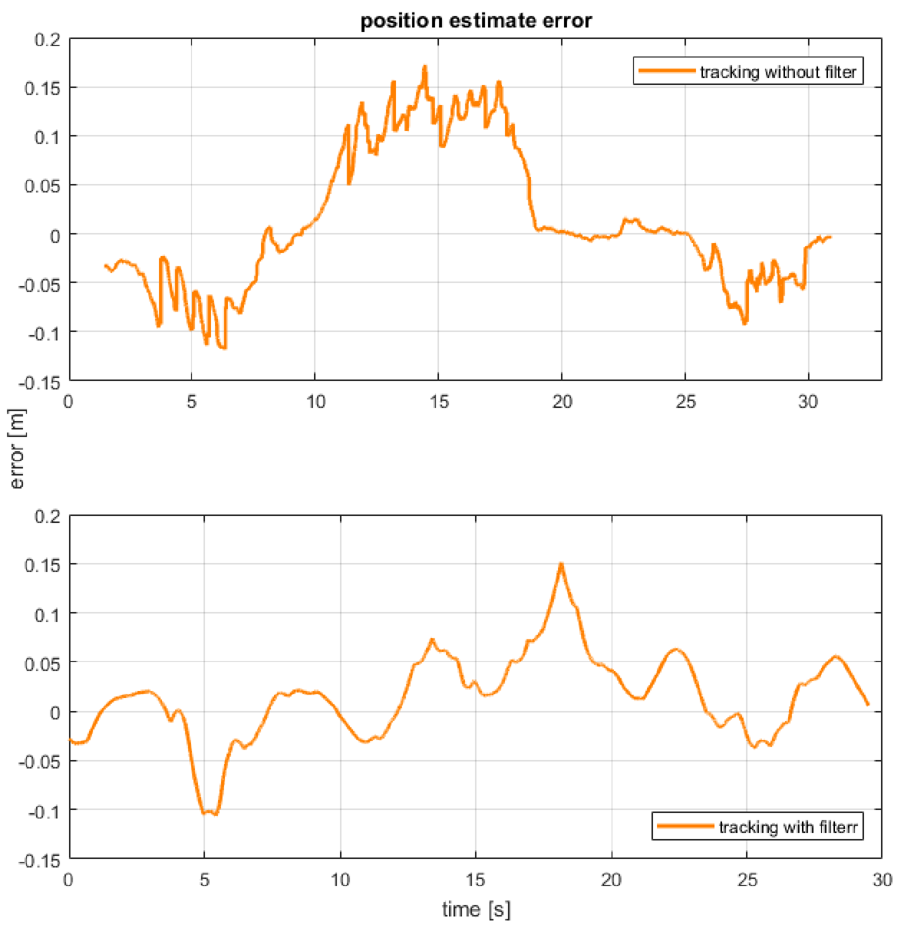

Figure 11. *Tracking position estimate error*

The error can move up to 0.1 m in a time less than a second. Such behavior is highly non-desirable. Moreover, the obstacle itself does not move that fast, and the variations present are not related to real world situation.

With the filter added, tracking error stays within ±0.1m, except for one peak at approximately 17 seconds into the sequence. Error also is somewhat originated from delay, because it slightly depends on movement direction.

Position estimates in other directions are presented in Figure 12 and Figure 13. There is no other measurement available as reference, but both should stay constant, because the moving platform moved the obstacle in only the direction corresponding to y-axis.

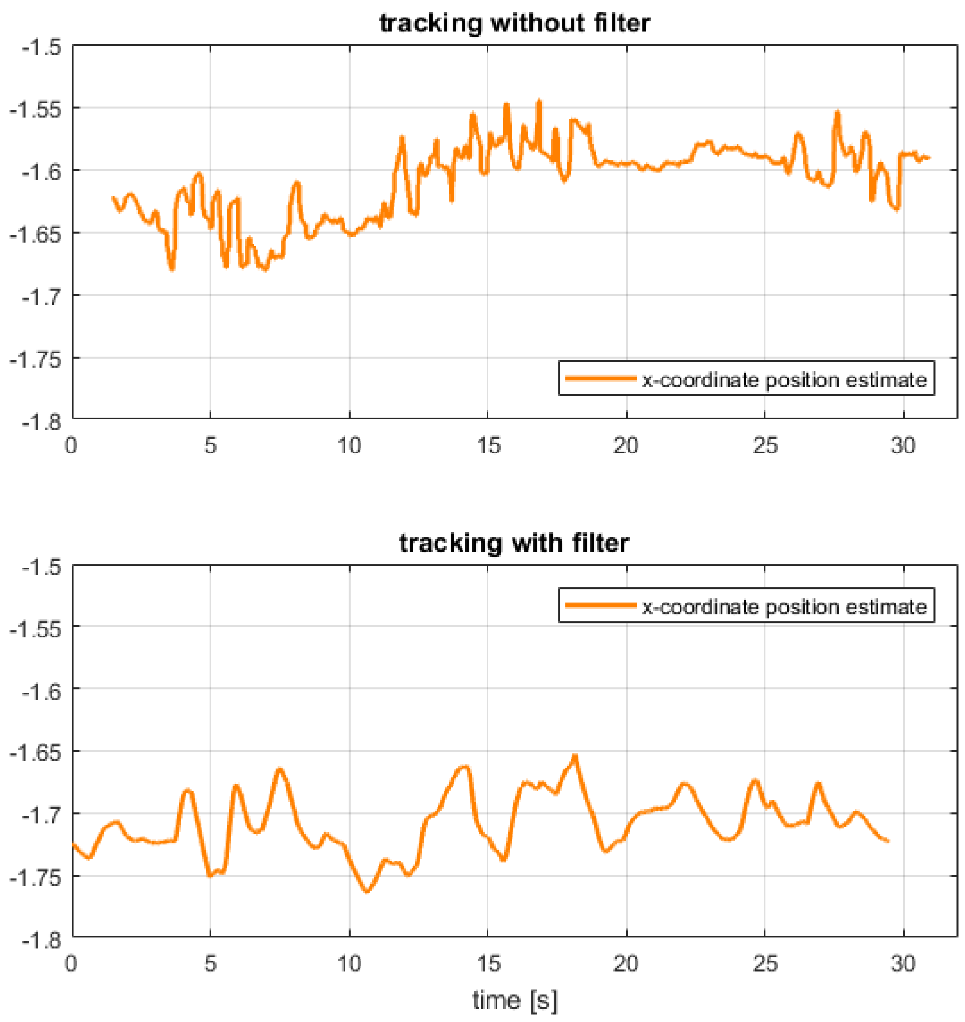

Figure 12. *X-coordinate of position estimate*

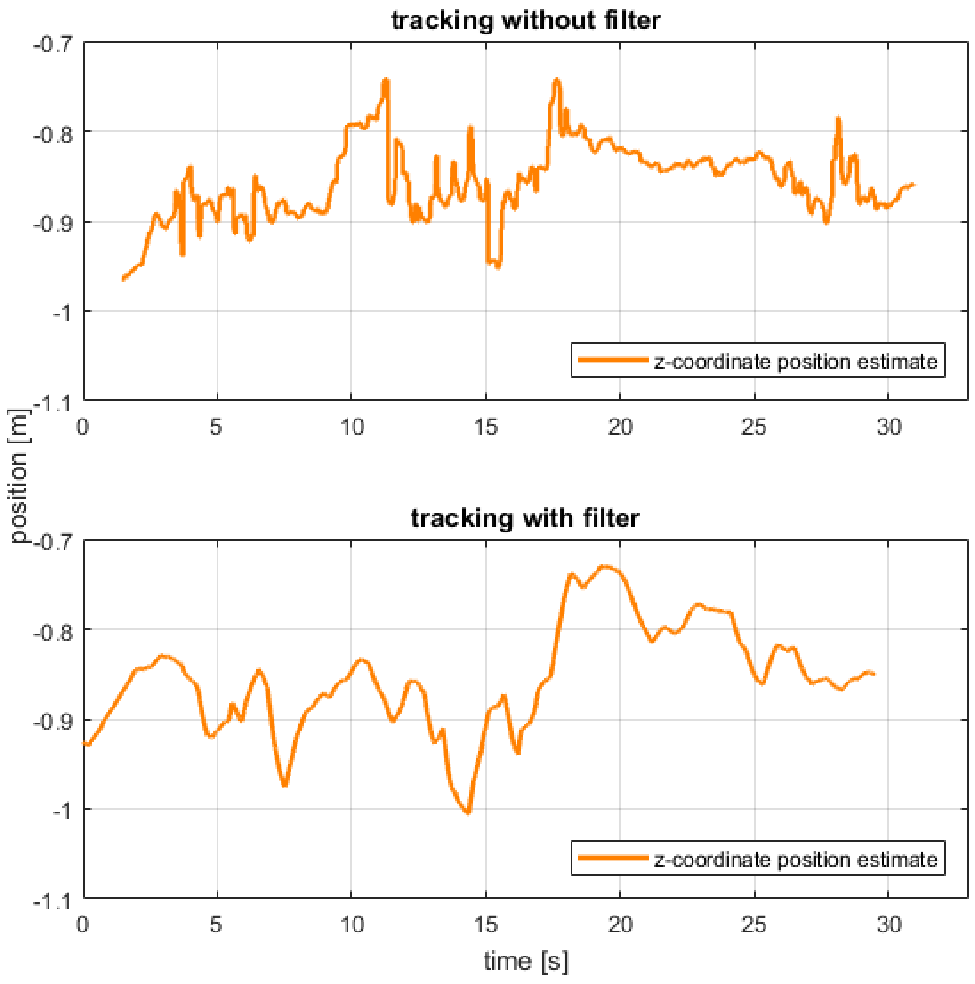

Figure 13. *Z-coordinate of position estimate*

The estimate for z-coordinate is the clearly more noisy than estimates for x or y. The difference between minimum and maximum value exceeds 0.2 m, whereas with x-coordinate (Figure 12) it was approximately 0.14 m.

In addition to obstacle position, its orientation was estimated. Since the linear slide did not provide any means to rotate the obstacle, the orientation stayed constant, and the estimate should also do so. Orientation was acquired as a 3x3 rotation matrix, but for visualization of the data it was converted to three angles, representing rotations around x, y, and z-axes. The results are shown in Figure 14, Figure 15 and Figure 16, respectively.

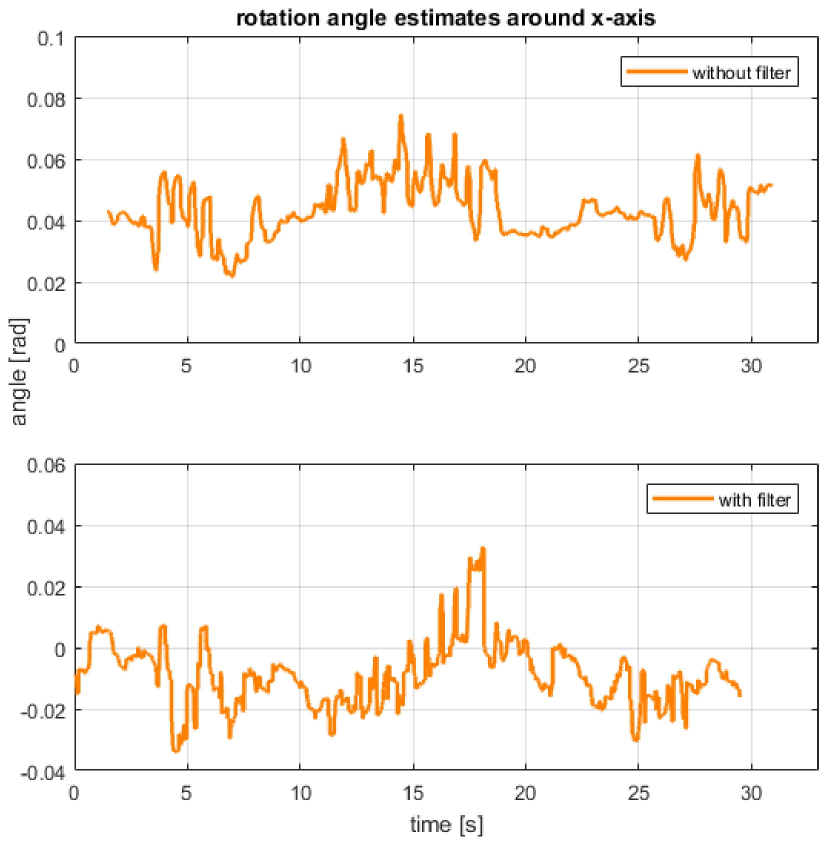

Figure 14. *Rotation angle estimate around x-axis*

There is noise in the rotation angle estimates also. In both cases, the variations in rotation estimate are less than 0.08 radians. This results to errors of similar range of less than in the position estimate itself: If we assume that obstacle position estimate is correct, orientation estimate error is 0.04 radians and obstacle dimension is 1 m, the result is approximately 0.04 m of error, at most, due to incorrect rotation

The smoothing effect of the filter is not as clearly visible as with positional coordinates. All other rotation measurements share these properties.

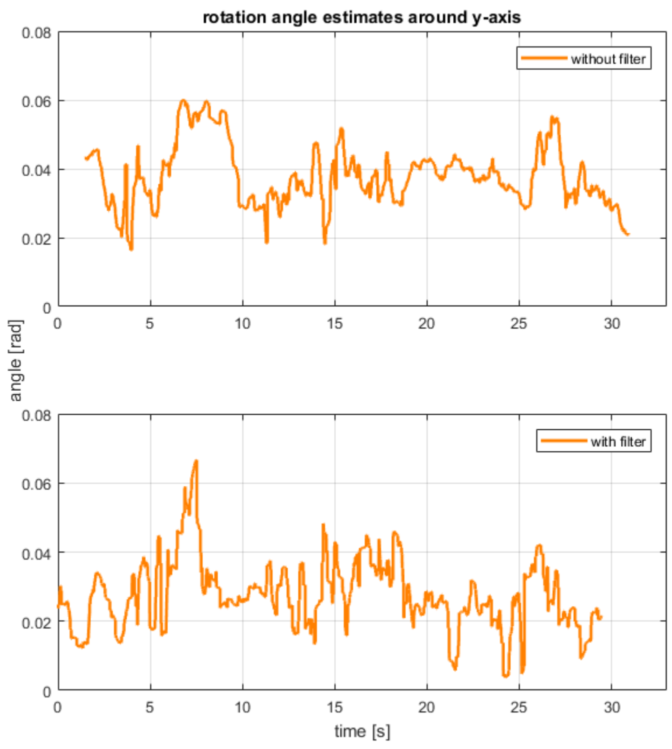

Figure 15. *Rotation angle estimate around y-axis*

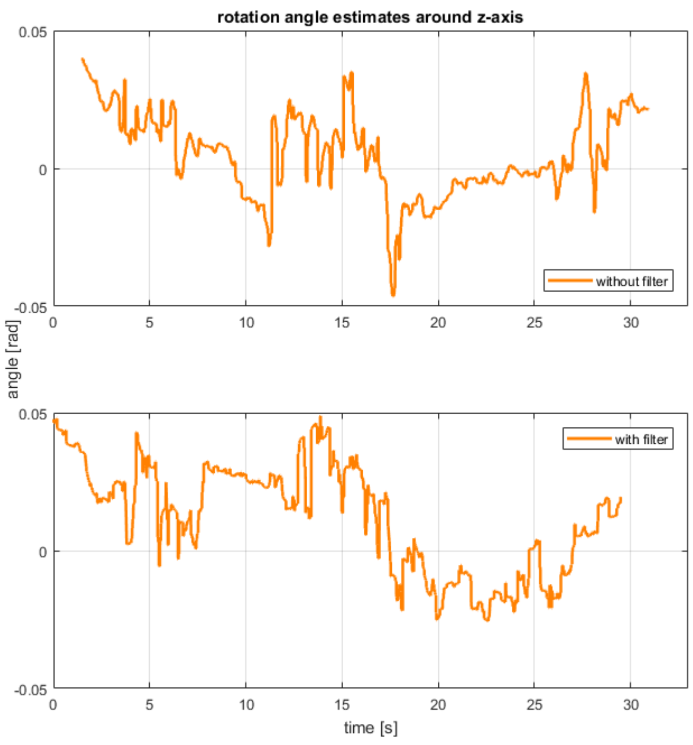

Figure 16. *Rotation angle estimate around z-axis*

6. CONCLUSIONS

This project addressed the topic of three-dimensional obstacle tracking. The aim was to find out whether obstacle tracking is realizable goal using a laser scanner, and to find out aspects that have to be taken into account while doing so. The work is relevant as there is a growing trend of autonomous machinery and automation in general within the industry, and obstacle tracking as a problem is important component and research question with those.

During the course of this project, a tracking system using a laser scanner was built. The system components and theoretic background were introduced in chapters 2 and 3. Also, other options for the laser scanning sensor were shortly reviewed, and various methods for the sensor data processing were reviewed. System functionality was tested using a physical obstacle and tracking its position. Results from the tests were reviewed, and compared to similar research done before. Based on the experiences learnt via system building and testing, possible future improvements for the current system were proposed.

Tracking system used a rotated laser scanner to produce three-dimensional point cloud from the environment. From this point cloud, the obstacle position was extracted using Iterative Closest Point –algorithm. The resulting coordinate value variations were smoothed using a Kalman filter.

Test setup featured a box-shaped obstacle moving along a linear track. Obstacle was moved while the system was actively tracking it position. The linear track provided measurement of the obstacle position, which made it possible to evaluate tracking performance.

The system did track obstacle position with maximum error of 0.15 m. The tracking delay was approximately one second. While testing the tracking without coordinate value smoothing, it became evident that there were significant fluctuations in the obstacle position estimates, especially while the obstacle was moving. These fluctuations were not similar in each direction, but did exceed 0.1 m from the assumed correct value, and happened under a period of one second. There were similar kind of fluctuations in the orientation estimate as well; expressed in rotation angles the amount was up to 0.04 radians.

This leads to two thoughts. First is that there are some design or configuration flaws in the laser scanner system, point cloud generation process or the tracking algorithm itself. For example, scanner movement could lead to point cloud not representing the environment correctly, and this cloud directly lead to obstacle position estimate to fluctuate as well. The second thought is that if the goal is an accurate position estimate, that goal can be achieved via smoothing out the fluctuations. Using a Kalman filter to smooth the fluctuations did improve actual tracking results.

One major problem was response rate of the tracking system. The 3D scan can’t be instantaneous due to physical limits with the laser scanner rotation assembly. Modifying the system so that the scanner rotation is faster would reduce overall delays. The drawback there is that the significance of rotation measurement and scanner delay compensation grows, and that aspect should be properly fixed until speed increase would give any benefits. One possibility is to use a laser scanner with native three-dimensional scanning ability. That would remove the need to rotate the scanner, and eliminate all errors that come from rotation process. The downside of that solution is a higher price.

The results obtained with this project show that it is possible to realize obstacle tracking with system based on laser scanner. However, the actual performance of the system at this point isn’t what one would expect with the technology available. Scanner accuracy would imply higher tracking accuracy than obtained, and scanning frequency would imply higher response rate than what was achieved.

REFERENCES

[1] R. B. Rusu and S. Cousins, 3D is here: Point Cloud Library (PCL), 2011 IEEE International Conference on Robotics and Automation, Shanghai, 2011, pp. 1-4.

[2] Tracking object in real time, Point Cloud Library documentation, Available: http://pointclouds.org/documentation/tutorials/tracking.php.

[3] R. M. Measures, Laser Remote Sensing: Fundamentals and Applications. New York: Wiley, 1984.

[4] A. Kukko, Mobile Laser Scanning – System development, performance and applications, Doctoral dissertation, Finnish Geodetic Institute, 2013, pp. 31-33.

[5] Operating Instructions, LMS5xx Laser Measurement Sensors, SICK AG, Germany, 2015, Available: https://www.sick.com/media/dox/4/14/514/Operating_instructions_Laser_Measurement_Sensors_of_the_LMS5xx_Product_Family_en_IM0037514.PDF.

[6] K. Litomisky and B. Bhanu, Removing Moving Objects from Point Cloud Scenes, in Advances in Depth Image Analysis and Applications. Lecture Notes in Computer Science, vol 7854, Springer, Berlin, Heidelberg, 2013.

[7] N. J. Mitra, N. Gelfand, H. Pottmann, and L.Guibas, Registration of point cloud data from a geometric optimization perspective, in Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing (SGP '04), ACM, New York, NY, USA, 2004, pp. 22-31.

[8] P. J. Besl and N. D. McKay, A method for registration of 3-D shapes, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 2, 1992, pp. 239-256.

[9] K. Klasing, D. Wollherr and M. Buss, A clustering method for efficient segmentation of 3D laser data, 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, 2008, pp. 4043-4048.

[10] A. Azim and O. Aycard, Detection, classification and tracking of moving objects in a 3D environment, 2012 IEEE Intelligent Vehicles Symposium, Alcala de Henares, 2012, pp. 802-807.

[11] C. Mertz et al., Moving object detection with laser scanners, Journal of Field Robotics, vol. 30, 2013, pp. 17–43.

[12] S. Thrun, W. Burgard, and D. Fox, Probabilistic robotics, MIT Press, Cambridge, MA, 2005.

[13] R. Labbe, Kalman and Bayesian filters in Python, 2017, Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.

[14] J. T. Thielemann, A. Berge, Ø. Skotheim and T. Kirkhus, Fast high resolution 3D laser scanning by real-time object tracking and segmentation, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, 2012, pp. 3899-3906.

[15] A. Harrison and P. Newman, High quality 3D laser ranging under general vehicle motion, 2008 IEEE International Conference on Robotics and Automation, Pasadena, CA, 2008, pp. 7-12.

[16] T. Yoshida, K. Irie, E. Koyanagi and M. Tomono, 3D laser scanner with gazing ability, 2011 IEEE International Conference on Robotics and Automation, Shanghai, 2011, pp. 3098-3103.

[17] F. A. Estiri, 3D Object Detection and Tracking Based On Point Cloud Library Special Application In Pallet Picking For Autonomous Mobile Machines, Master's thesis, Tampere University of Technology, 2014.

[18] Data sheet, Velodyne HDL-64E, 2017, Available: http://velodynelidar.com/docs/datasheet/63-9194_Rev-F_HDL-64E_S3_Data%20Sheet_Web.pdf.

[19] Downsampling a PointCloud using a VoxelGrid filter, Point Cloud Library documaentation, Available: http://pointclouds.org/documentation/tutorials/voxel_grid.php#voxelgrid.

[20] S. Holzer, et al. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp. 2684-2689.

[21] S. Pu, G. Vosselman, Knowledge based reconstruction of building models from terrestrial laser scanning data, ISPRS Journal of Photogrammetry and Remote Sensing, Volume 64, Issue 6, 2009, pp 575-584.