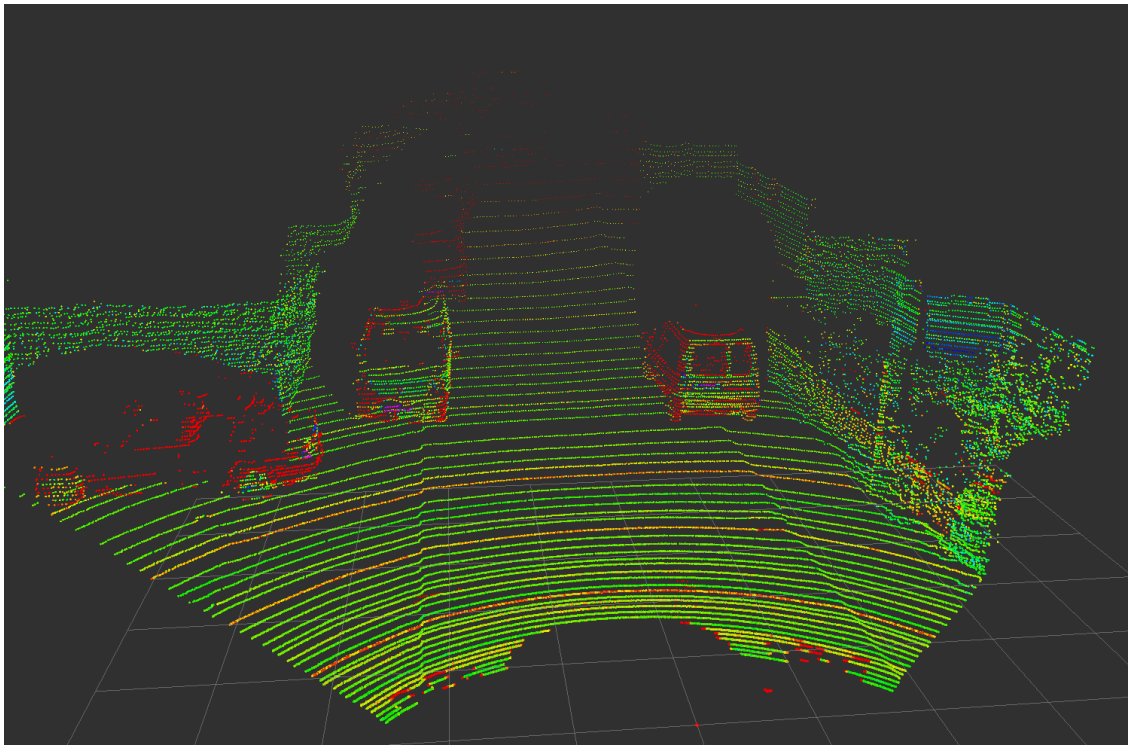

LiDAR Object Detection and Sensor Fusion in Simulation Environments

Sensor modelling towards advancements in Real2Sim - Sim2Real

Bachelors Thesis in Computer science

Mahan Vahid Roudsari

Christopher Höglind

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Bachelor's thesis 2020

LiDAR Object Detection and Sensor Fusion in Simulation Environments

Sensor modelling towards advancements in Real2Sim - Sim2Real

MAHAN VAHID ROUDSARI

CHRISTOPHER HÖGLIND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of *Computer Science and Engineering*
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

LiDAR Object Detection and Sensor Fusion in Simulation Environments
Sensor modelling towards advancements in Real2Sim - Sim2Real

MAHAN VAHID ROUDSARI
CHRISTOPHER HÖGLIND

© MAHAN VAHID ROUDSARI, 2020.
© CHRISTOPHER HÖGLIND, 2020.

Supervisor: Sistek, Sakib, Department of Computer Engineering, Institution of Computer and Information Technology.
Examiner: Duregård, Jonas, Department of Functional Programming, Institution of Computer and Information Technology.

Bachelor's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualized data in RVIZ by deep neural network displaying different intensities depending on reflectance.

Typeset in LaTeX
Gothenburg, Sweden 2020

LiDAR Object Detection and Sensor Fusion in simulation environments
Sensor modelling towards advancements in Real2Sim - Sim2Real
MAHAN VAHID ROUDSARI
CHRISTOPHER HÖGLIND
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

In the world of autonomous driving the environment perception is of the outmost importance. Light detection and ranging(LiDAR) data is one of the most common sensor data in the field of autonomous driving. This thesis will therefore explore LiDAR data as the first steps toward using it in a simulation environment. In order to be able to explore the use of LiDAR, an arsenal of tools and knowledge is needed. This thesis goes through the learning steps of these tools and in which way they fit together to arrive at the result. This thesis includes different approaches of visualizing data using Robotic Operation Systems (ROS) framework and implementing 3D bounding box annotation and also covers creating a convolutional auto encoder and implementing it as a deep neural network. The result is knowledge on how LiDAR data works, how it can be used and estimating some characteristics using a deep neural network. Included in the results are a few visualizations of the data after being trained on a neural network. Further work once this knowledge is gathered would be to implement it into a simulation environment.

Keywords: Sensor modelling, LiDAR, Sensor data visualization,Velodyne point, Bounding box, Autonomous driving, ROS, RVIZ, Deep neural network

Acknowledgements

Thanks to Khalid for technical help, guiding and overall support.
To Sakib for supervision and introducing us to Alexander and Jonas at Sigma.
To Jonas for assigning us to Khalid as company supervisor.
To David Frisk for creating the LATEX-template (CC BY 4.0.)

Gothenburg, June 2020

Contents

List of Figures

Abbreviations

$LiDAR$	Light detection and ranging
$KITTI$	Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago
ROS	Robot Operating System
$RVIZ$	ROS Visualization tool
DNN	Deep Neural Network
OS	Operating System
$PC2$	PointCloud2
CNN	Convolutional Neural Network
CAE	Convolutional Auto Encoder

1
Introduction

This section contains an introduction to the thesis and background to the touched subjects.

1.1 Background

The continuous development of self-driving cars demands exploration in a variety of fields. Sigma Embedded Engineering is working together with Volvo engineers to research environment perception, which is one of the main components in autonomous driving. The research involves development in environment modeling. Gathering the data for environment modeling is done with help of "Light Detection And Ranging" (LiDAR), "Radio Detection And Ranging" (Radar), cameras and ultrasound. Creating these advanced models requires a huge amount of data and to gather this in the real world is expensive and time consuming. What is being developed, is sensor modeling in simulation environments which unlocks the potential to gather the data needed in simulations which in contrast to reality is very cheap. In the simulated environment, tweaking sensor settings, weather condition, road condition or time of day can be done without needing to rely on external factors. Simulation environment is a strong tool in the development of self-driving cars.

1.2 Purpose

The purpose of this thesis is to investigate LiDAR datasets and examine how collected data can be visualized to show applications of LiDAR and the benefits of doing it in a simulation environments.

1.3 Goals and Objectives

To achieve the purpose of this thesis a few different tasks must be done.

- Investigate LiDAR data-sets and learn how to extract the data to a code-able type.
- Learn the ROS framework which the data can be applied to.
- Visualize the data with the RVIZ tool.
- Train a DNN and visually compare its generated data to actual data.

1.4 Limitations

No usage of own sensors will be used to collect data, only already collected data-sets will be used. There will be no in-depth learning on how LiDAR works, only understanding the collected LiDAR data and how to use and modify it. The software developed will be in the form of scripts and enhanced frameworks, and no simulation.

2
Method

This thesis' main focus is gathering data, manipulating it and finally visualizing it. In order to achieve these objectives, a lot of learning and experimenting is required. The learning will be carried out by doing various online courses and many tutorials. This dependency on the knowledge of the subject means the first phase will be heavily focused on learning. This section; Method and theory will include the required topics in order to start and complete this thesis and the methodologies and theories used in perusing this thesis.

2.1 Lidar frames and dataset

Autonomous cars need to have a vision of their surroundings, something like an eye that would illustrate and define objects around them, their location, distance and other properties. Currently LiDARs are one of the most used sensors that would provide such digital vision. LiDAR sensors capture data and stores it as LiDAR frames. These frames always include information about 3D points and timings, but sometimes even provide information such as object labels, GPS data, and even captured pictures. This information can be post-processed in the processing unit so that all the data combined together can lead to 3D vision for the cars.

This thesis will use *Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago* (KITTI) dataset which includes such LiDAR frames. KITTI has captured information such as images, 3D Velodyne points, GPS, calibration and tracklets at a rate of 10Hz [1].

2.2 Researching ROS framework

In order to get start processing LiDAR data there was a need to find a framework on which this thesis will be based on. This thesis will be using Robot Operating System(ROS) which is a flexible framework for robotic software. ROS includes many libraries, tools and documentation which will simplify the progress throughout the thesis [2]. ROS is used in creating advanced robot software and is quite complicated for a beginner to start using. Due to the complexity of this framework, the initial main focus primarily on getting ROS to run on the used systems(Personal computers) and getting familiar with the ROS framework's general usages. The first stage of studying ROS covered simple communication nodes, creating and using custom message types, creating the connection between the communication nodes and creating a client in Python [3].

2.3 Visualizing and filter LiDAR data in RVIZ

LiDAR data might at first sound simple and straightforward, but in order to even use the data, it must be decoded from binary files into a more usable data type. In this thesis the data will be encoded into Numpy arrays which is a library that supports large multidimensional arrays, matrices and a big variety of mathematical functions [4]. Once the data is transformed into Numpy arrays it will be visualized in the ROS framework using it's powerful visualization tool, RVIZ [5]. Once all the data is visualized, it will be modified and filtered based on multiple specified filters, displaying only the specific filtered points from the binary files.

2.4 Generating annotation types

After visualizing the data in RVIZ it will be necessary to create custom annotation types for further analysis of the data. The annotation type that will be use in this thesis will be object bounding-boxes.

2.5 Researching Deep neural networks

To generate new sensor data based on the KITTI dataset, deep neural networks(DNN) will be used. Such networks can be trained on any type of data and based on their inputs and amount of training they can generate data with high accuracy. DNNs are currently used in many image recognition areas and the model used in this thesis will be based on a similar architecture.

2.6 Keras API

As mentioned there are different versions of TensorFlow where one of the new things with TensorFlow 2.0 is the use of the Keras API. Keras was developed to simplify TensorFlow code. The thesis uses both code with and without Keras as changing working code was unnecessary.

2.7 Designing a Deep neural network

Once the data has been visualized and annotation types have been created a custom-made DNN will trained with the whole data set (1TB+). The DNN should be able to generate new sensor data. This step will require a lot of attention to details and small parameters in order to construct an accurate and acceptable sensor model that generates desirable and accurate sensor data.

2.8 Programming languages and libraries

This thesis will require a lot of programming and in order to keep everything in control and well organized, most of the modules will be written using Python3. Python is a free, object-oriented high-level programming language [6]. The main libraries that will be used are Rospy [7], Numpy [4] and Pyplot [8]. Rospy is the library provided by ROS and Pyplot is a library provided by Python which is mainly used for plotting and displaying data.

2.9 Operating system

In-order to run all the mentioned frameworks, programs and scripts there were two viable options for operating systems(OS): Windows and Linux. Due to the experience on both OS it was decided to use both of them. This decision was based on having more options, releases of software and maximised compatibility. In the main station used, the Linux subsystem was installed on a Windows 10 PC which meant both Linux and Windows could work hand-in-hand. This made it possible to use Windows edition of TensorFlow and Linux distribution of the ROS while having Python running natively in both Linux and Windows.

3
Investigation

This section explains the different datasets and tools used in this thesis. Here. the advantages, disadvantages and context of the datasets and tools are explained.

3.1 The Datasets

At the early stages of the thesis the chois was was between one of the two datasets: KITTI dataset and the Nuscenes dataset. Both datasets were collected by placing sensors on top of a car which then captured LiDAR data during multiple driving sessions. The differences between the datasets were with which sensor it collected the data and in how the data was stored.

3.1.1 KITTI Dataset

The KITTI dataset uses a Velodyne HDL-64E laser scanner to collect its LiDAR data [9]. The following specifications for the sensor are necessary to undestand in order to use the data. The KITTI dataset is divided into different driving sessions and each session has four downloadable data attached to it: "Unsynced+Unrectified" , "Synced+Rectified" , "calibration" and "tracklets" data. In this thesis the last three are essential due to their included information and synchronisation in-between them. The Synced+Rectified data includes camera images and LiDAR data while the calibration data matches the different data to one another and tracklets for identifying different objects in the scenes [6].

On the KITTI website there are links to various tools for using the data with ROS and Python. Due to already having explored the ROS framework, this was the optimal dataset to start working with.

KITTI dataset is not as detailed as Nuscenes in regards to tracklets which was noticed once the tracklet objects were explored. However, because the thesis' main interest was in exploring and experimenting, the KITTI dataset was well suited for this application.

3.1.2 Nuscenes Dataset

The Nuscenes dataset uses a spinning LiDAR with 64 channels compared to the KITTI one which uses 32. Extra channels are required in order to achieve a 360 degree view around the car. This leads to bigger dataset size due to the number of captured points but the LiDAR collection is smaller which is another reason to use KITTI if only working with velodyne points. Nuscenes does not introduce any tools for working with ROS which is another reason it was not used even considering its superior tracklet data.

3.2 LiDAR frames

As previously mentioned the selected dataset for this thesis was KITTI and this section contains specifications from the KITTI dataset's LiDAR frames. Note that LiDAR data is usually very similar to one another with small differences in the captured data (order of information and the information itself). The following information is gathered from KITTI Foundation. [1]

3.2.1 Velodyne points

Velodyne points are points that match the real life data collected from the LiDAR sensor as 3D Velodyne pointclouds. This data is stored in floating point binaries which were then parsed into Python. Each point is stored with its (x,y,z) positional coordinates and a reflectance value r (float ranging between 0 - 1).

3.2.2 Images

The image data can be downloaded as gray-scale stereo and color stereo sequences(0.5 Mega pixels, stored in png format) and be used as a reference visualization to see if the LiDAR data matches reality.

3.2.3 Tracklets

The downloaded tracklet data contains 3D bounding box information which tracks between different objects in the frames represented in Velodyne coordinates. These positional coordinates are extracted to be used in multiple Python scripts to visualize bounding boxes in RVIZ and PyPlot.

3.2.4 Calibration, 3D GPS and IMU data

KITTI includes lots of text information which contains calibration data for synchronisation between Camera and GPS and synchronisation between Camera and Velodyne points. This calibration data is used for processing the raw data into a synchronised version. In this thesis the data used is mostly already calibrated and ready for experimenting. The next set of data includes location, speed and acceleration of the vehicle which will be used in order to correctly map objects and tracklets with the velodyne data.

3.3 ROS Framework

Most of the visualization and communication in this thesis is performed in the ROS framework due to its popularity among autonomous vehicle company's and Sigma's desire to base this thesis on. The following section includes the followed methodology in ROS communication and visualization.

3.3.1 Communication

The ROS framework has its own communication procedure and in order to be able to use it, learning this communication model was essential. ROS communicates using nodes and topics. These nodes in ROS can have two different types, publisher nodes and subscriber nodes. Publisher nodes act like clients where they can send data and be the source of a communication while subscriber nodes can be considered the clients and have to rely on receiving the information from the subscriber nodes. In order for a communication to be possible between two nodes, they have to expect and define a communication subject thus creating the need for ROS topics. ROS topics define the "topic" of the communication between two nodes, for example, Velodyne pointcloud in this case.

The simplest communication will need one publisher node, one subscriber node and a topic between them. Once it was possible to filter the pointclouds and add markers this is what the final ROS communication looked like:

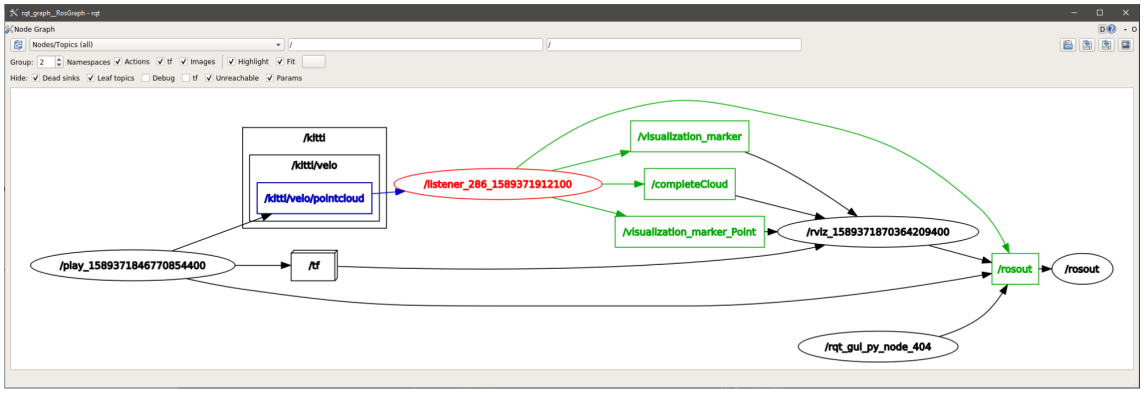

Figure 3.1: Communication between ROS communication nodes. The **blue** node indicates a **subscriber** node while the **green** nodes indicates **subscribers** and the **red** node indicates a **topic**.

3.3.2 Rosbag

One of the simplest ways to display raw data in ROS using RVIZ is displaying the whole dataset without any modification. This is typically done by using a program/script to convert the raw binary data into a "bag" format which is supported by ROS. The bag format is a file type that can be used to playback the data as a publisher node in ROS. In the first step of visualization for this thesis, kitt2bag [10] is used in order to convert the KITTI raw data into a bag file. This conversion requires the raw dataset, the synchronisation data and the calibration data. Once the bag file is created, ROS bag library is used to create a publisher node that sends data at a defined frequency. Initially the publisher was able to run almost 30 frames per second but once the modification scripts were in place it was necessary to lower the frequency to 5 frames per second to be able to keep up with the calculations and keep the scripts in sync.

3.3.3 RVIZ

As mentioned earlier, the main visualizations will be done using RVIZ. RVIZ has many different display types and in this case the primary display types will be: PointCloud2(PC2), Images and Markers. The Velodyne points in KITTI's dataset all have their world position attached to them which makes them suitable to be used with PC2. An image data type is simply the captured pictures from the cameras utilizing the KITTI cars. As mentioned before, these images have been captured from multiple directions for every single frame in both colored and gray-scale.

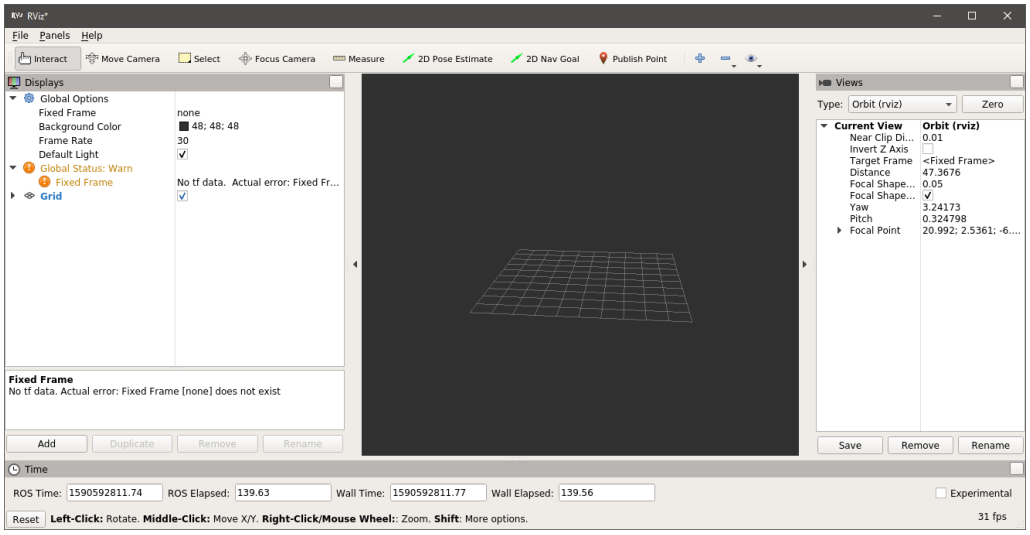

Figure 3.2: Figure displaying RVIZ graphical user interface.

3.3.3.1 PointCloud2

The PC2 display type draws the dataset as points in the world which makes it easy to get an overall view of the LiDAR data in RVIZ. The following figure is visualizing PC2 from a single unmodified LiDAR frame using RVIZ. It's clear that cars can be seen on the left side of the frame but the background walls and other smaller details are much harder to interpret.

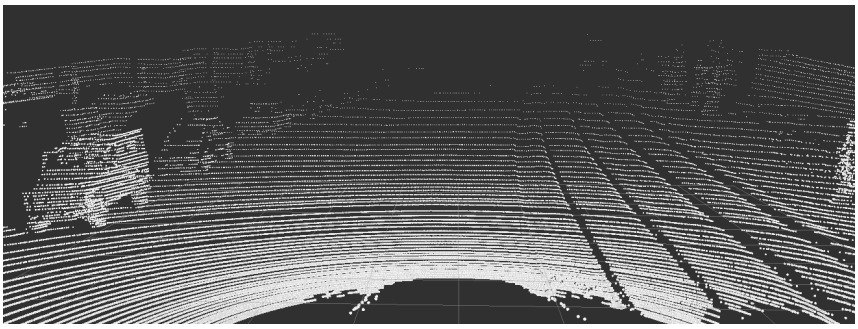

Figure 3.3: An example of a LiDAR frame from KITTI displaying PC2 in RVIZ.

3.3.3.2 Image

Image is another data type that is used in this thesis' data visualization due to the amount of information that can be extracted from them. The image data types can be shown simultaneously as the PC2 data in the sidebar which helps to understand the PC2 and interpret the points. As can be seen the same cars are visible here on the following two pictures but now it's much clearer to interpret house walls, trees and the rail tracks.

Figure 3.4: An example of a grayscale image captured by the KITTI rig.

Figure 3.5: An example of a colored image captured by the KITTI rig.

3.3.3.3 Marker

Another powerful datatype in RVIZ is the marker data type. This data type is essential for visualization of data and further easing the interpretation of the objects. In this thesis multiple types of markers were used for visualising data, especially when displaying the 3d bounding box annotation. In this thesis the following markers were used and combined with the PC2 to create the 3d bounding box annotation.

The **Line strip** marker is able to draw multiple lines between defined points. For example in order to draw a cube, rectangle or even a polygon.

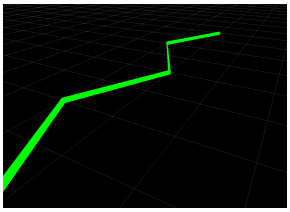

Figure 3.6: A Line Strip marker in RVIZ.

The **Sphere list** marker has the ability to store multiple spheres in an array and display them all in the same LiDAR frame. This marker has the ability to have different properties for each sphere which enables the possibility to display different colors depending on the object's classification.

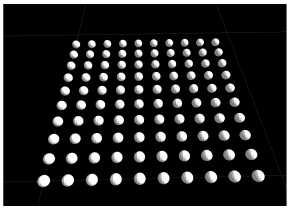

Figure 3.7: Sphere list marker displayed in RVIZ.

The **Arrow** marker draws an arrow from point A to point B. This marker can also have different colors and sizes and was used to show orientation of an object.

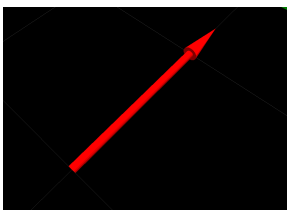

Figure 3.8: An arrow marker displayed in RVIZ.

3.4 Annotation types and views

A main part of this thesis consisted of visualizing data thus making the annotation types another important subject. This section will further explain the different annotations and visualizations used and the reasoning behind them.

3.4.1 Object bounding boxes annotation

Object bounding box annotation is an annotation type used to display edges of a cube containing an object. This annotation type is primarily used to verify the synchronization of the dataset and tracklet and to display simple but important information about each object. This information can vary from displaying only the "bounding-box" to including the center of each object, their classification and orientation.

3.4.2 Extracting infromation from tracklets

In this thesis plenty of information is received and parsed from tracklets. These tracklets contain data about objects which includes the position of each corner for a bounding box and a classification which defines the type of object (i.e. car, pedestrian or bike). The tracklets are used in order to calculate position, size and orientation of each object. This data is then exported as a custom numpy array which included the necessary information.

X	Y	Z	Width	Length	Height	Orientation	Classification

Figure 3.9: Format of the generated numpy array including essential information about each object.

3.4.3 Plotting the generated numpy array

Once the numpy array containing the information is constructed, the data is then passed to another Python script. This script then used PyPlot and a polygon formula in order to draw lines for the bounding box. The bounding boxes are then assigned different colors depending on their classification. As a final step, a sphere is added to the center of each box to mark their origin, and a normalized vector is drawn as a line to illustrate the forward orientation of the objects.

3.4.4 Visualizing the generated numpy array

Next step of visualizing the bound boxes is to display the data in RVIZ. This step consists of reforming and updating the script in (section 3.4.3) and adding the necessary functions. These functions and methods consist of creating ROS subscriber nodes, retrieving information about LiDAR frames, synchronising the live data and tracklet and finally overlayin the bounding boxes over the PC2. In this step LINE_STRIP is then used to draw bounding boxes while SPHERE_LIST visualizes the center of the objects and their classification type using different colors.

3.5 Deep Neural Networks

The finishing piece which binds all the above tools and datasets together in this thesis is deep neural networks. The following section shows the very basic building blocks of a deep neural network which is needed to build a working DNN. Due to the main focus of the thesis being placed on LiDAR data only the very basics of DNNs were explored, thus only an overview will be covered below.

3.5.1 Neural Network

Visualising LiDAR data is one thing but in order to let the program learn characteristics of the LiDAR data, which for future work is important, the use of machine learning is necessary. The basics of machine learning is that the program learns by training from examples and understand important features. This is done by setting up the architecture for the computer, providing it with data and letting it solve the hard calculations. The type of network that this thesis uses is a Convolutional neural network (CNN). This is due to the CNNs being superior in image reconstruction, compared to other neural network types. After building a CNN it will be trained using a large LiDAR dataset. As it is an image reconstruction the goal is for the output to be very similar to the input.

3.5.2 TensorFlow

TensorFlow is an open-source platform used for numerical computation and machine learning. As TensorFlow uses Python for it's front-end, it is the suitable framework for this thesis as all the ROS code is also written in Python. TensorFlow is used to train and run DNNs of various types. TensorFlow is explored with the intent to build a neural network that can learn from the LiDAR data scripts and estimate an intensity depending on the Velodyne points positions. TensorFlow 2.0 is fairly new and was released in October 2019, therefore there are many compatibility issues with other frameworks. This thesis will work with v.2.0 in a v.1.0 compatibility mode. [11]

3.5.3 Convolutional Neural Network

The type of Convolutional nerual network which is constructed to handle the LiDAR data is a Convolutional Auto Encoder (CAE). The CAE architecture is simple and a good first step into DNN. The main focus here is generating similar LiDAR visualizations from random frames in KITTI's dataset.
The CAE needs three major parts to function: encoding, decoding and loss calculation. The encoding is the compression of the input into the the CAE. The decoding is the decompression of the input, similar to the encoding with the exception of being able to choose the last output value, chosen from the inputs. The loss function indicates how well the training works by measuring the error between the outputted value to desired value.

3.5.4 Training an DNN

To improve the training of a CAE changes are made to the hyperparameters. In the CAE the hyperparameters are: Code size, number of layers, number of nodes per layer and the loss function. The two hyperparameters that will be experimented on in this thesis are number of layers, how many layers of encoding/decoding the data goes through, and the loss function. Another thing to keep in mind when training a DNN is the gradient decent, commonly named the steps of the optimizer. Optimizers are algorithms/methods that changes the attribute of the neural network. TensorFlow has a few built in options of optimizers to choose from and the one used in thesis is named Adam Optimizer.

3.5.5 GPU instead of CPU

When running a training session it requires a lot of processing power from the computer's side. Luckily TensorFlow calculations are able to be run on the GPU which is better for images calculations. The smaller training set in the thesis was made up of 60 000 images and training it on a CPU would not be time efficient.

4

Implementaion

In this section the result of each part of the thesis will be explained and presented.

4.1 Visualizing unmodified velodyne points

The unmodified version of velodyne points were visualized in RVIZ using the rosbag format. The binary files and calibration information were combined together using Kitti2bag tool and the bag file was created. During this process the tracklet information was separated from the other files thus losing synchronization between the tracking information and velodyne points. The bag file was then visualized in RVIZ by using the "rosbag play" command using default parameters. Once the communication nodes were created and accessible, the data was selected by choosing PC2 data type in RVIZ and selecting the corresponding communication topic. The following snapshot represents a single frame of the bag file.

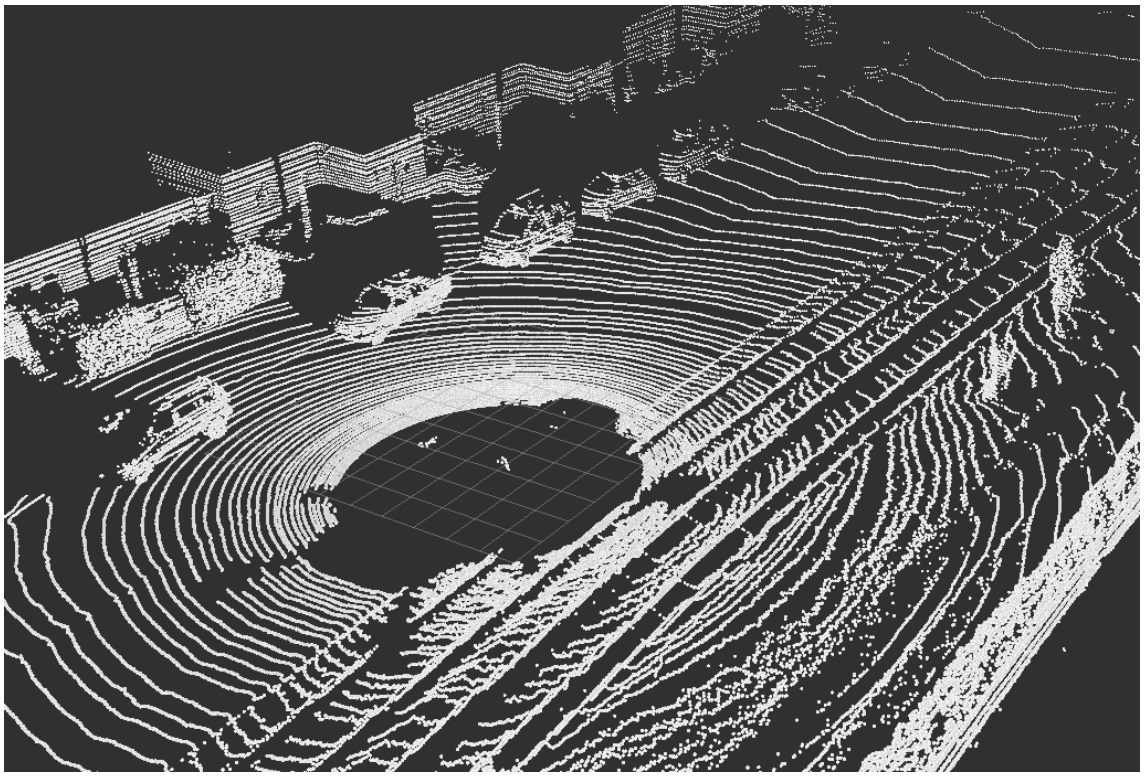

Figure 4.1: A LiDAR frame's pointcloud2 displayed in RVIZ.

4.2 Visualizing modified velodyne points

In order to visualize the modified version of velodyne points the process is more advanced and requires more steps and scripts which will be explained in this section. Visualizing this data-set requires creating the filters, applying them to the PC2 and then displaying them using a custom created communication node instead of the bag file.

4.2.1 Interpreting the binary files

As mentioned earlier in the thesis, Velodyne points are stored in binary files. For each frame of the LiDAR data there is a corresponding binary file, which includes the points, their position and reflectance. Using a Python script these files were read and later transformed into numpy arrays. These numpy arrays made it possible to be able to modify the large arrays and have access to them.

4.2.2 Filtering layers

The first step to filter the data into layers was to acquire deeper knowledge of the LiDAR data and the way it was recorded. In specifications it was mentioned that the LiDAR camera that was used to record KITTI data-set, was spinning clockwise. This information might at first not seem important but by really understanding this small detail and considering its result it was concluded that the data is sorted into "rings" that are captured in a clockwise circular pattern. This information makes it possible to set a defined border between each layer by keeping track of which quadrant the points' location will translate into. Using a mathematical formula and a Python script it was possible to define where a new layer starts thus making it possible to filter the numpy arrays into layers. The following snapshot visualizes the layers 3, 6, 9, 12 for a specific frame.

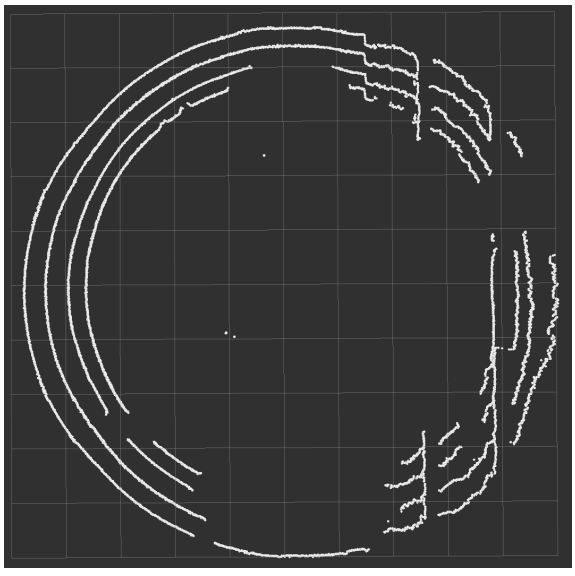

Figure 4.2: Pointcloud2 filtered by layers. (Layers 3, 6, 9, 12)

4.2.3 Filtering Columns

Filtering the data into columns were done by combining the information from layer filtering (4.2.2) with basic trigonometric formulas. This filter creates slices of LiDAR data in order to check if an object intersects with the vehicle's path or any other direction from it. To create the slices, each point was sorted by their angles from the origin (in this case the LiDAR camera on the vehicle) and then filtered again by their layers. This snapshot displays ninety different layers of four different columns, note that you can see that the points in top left are not shaped as perfect circles indicating their collisions with other objects.

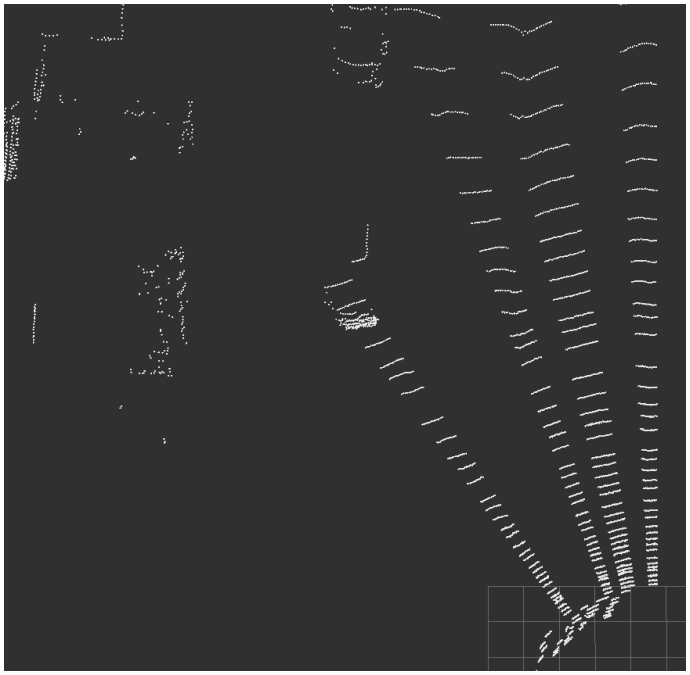

Figure 4.3: A Line Strip marker in RVIZ.

4.2.4 Pointcloud2 filtered by columns.

Once the data was filtered into layers and columns (4.2.2 and 4.2.3) the output was shaped into a numpy array meaning it was not possible to directly visualize it as a bag file. Once the "rosbag play" command was not an option, a workaround solution was built around the ROS communication nodes and manually creating the topics and the data being transferred.

4.2.5 Publishing PointCloud2 into ROS

The first step of this process was to create a publisher node which required a node name, data type and a queue size. The node name was unique for each filter type while the data type and queue size were constant (PC2 and one million). This resulted in a publisher node capable of publishing up to one million points as a PC2. Due to the real-time aspects of the ROS framework every publisher node requires correct headers to ensure synchronisation between multiple data types. The headers were then constructed using std_msgs as message type, correct timestamp in milliseconds and a unique frame_id. Std_msgs is just a default message type in Python while frame_id is just a unique number for each new frame.

Once the publisher node was constructed and ready to be used, one last step was necessary in order to input the actual filtered data into the publisher node, transforming the numpy array into a PC2. This was achieved by using a method called Create_cloud_xyz32() which would create the data using the recently created header and the filtered data itself. This method was later replaced by a custom method to provide extra customizeability to parse more parameters such as reflectance and colors. Once the data was in correct format the publisher node could finally publish the data into the ROS framework. Using RVIZ, the publisher node was then selected and data was visualized as expected.

4.3 Creating Object bounding box annotation

In order to differentiate different objects from the PC2 in the data-set it was necessary to use a simple annotation with clear borders that defined each object while providing important information. Object bounding box annotation was the preferred way to do this. Using this annotation, boxes were drawn around the objects and each classification had its own colors. While tracklets provide information such as corners of the object, some mathematical functions are necessary in order to calculate the center of each box and their orientation. Center of each box was calculated by averaging the sum of each of the eight corners of the object using the following formula.

$$n = 8 : Center_{xyz} = \frac{1}{n} \cdot \sum_{0}^{n} (x_i, y_i, z_i) \tag{4.1}$$

Figure 4.4: Mathematical equation for finding center of objects depending on their bounding box corners. X, Y, Z are the positions of each corner while.

Once the center of the objects were defined it was possible to calculate the orientation of the object by using trigonometric functions. This way the object bounding boxes could be drawn using the size, orientation and center of the objects. The tracklets provided information about object classification which was used to display different boxes with unique colors in order to easier differentiate the different object types. The object bounding box was implemented in both PyPlot as a standalone script for a single frame and in RVIZ with live data.

4.4 Designing the Deep Neural Network

The investigation on DNNs revealed that for this project a CAE was the optimal choice as the objective of the DNN is to reconstruct an image with correct reflectance data. The DNN takes the Velodyne points of the LiDAR frames and reconstructs their reflectance. To see how well it works, it outputs the reflectance to see the difference from the validation data, Where the difference will be shown in the color of the points. Below one of the final visualizations will be shown and in the appendix a few more visualizations from the outputted data can be viewed.

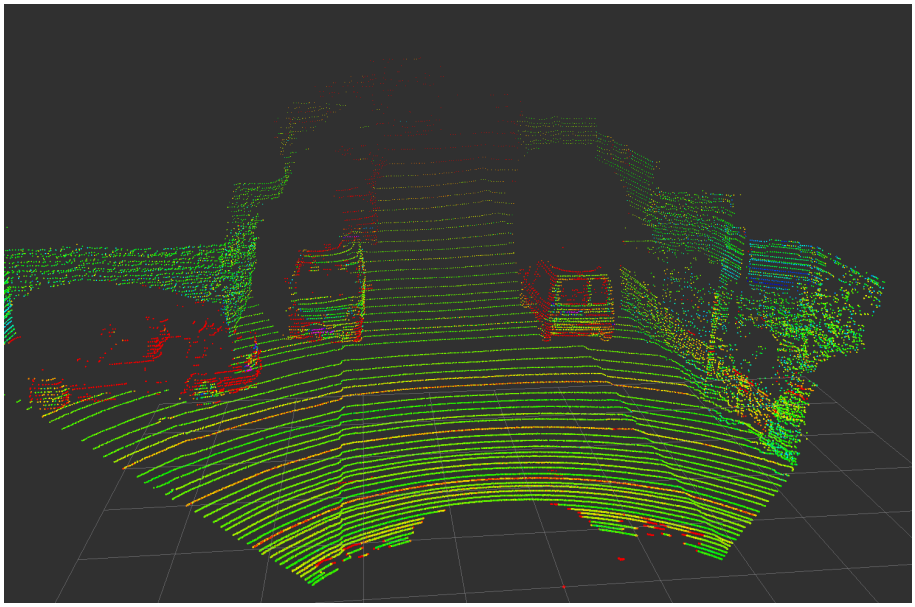

Figure 4.5: Visualization of the LiDAR data in Rviz after being trained on the CAE.

4.4.1 Hyperparameters

Deciding the hyperparameters of the CAE was the next step. As mentioned in the theory chapter there are 4 different hyper parameters and in this case the number of layers and the loss function was the focus during the experiments. After some trial and error two convolution layers were chosen in both encoder and decoder which was enough for this case. The loss function used is a mean error calculation which takes the the original value minus the reconstructed output value:

$$\frac{1}{n}\sum_{0}^{batchsize}(original - reconstructed)^2 \tag{4.2}$$

Figure 4.6: Mathematical equation for calculating the mean square error.

4.4.2 Early stop

Training a DNN more does not always result in a more accurate model. This mean it is possible to train a model with a really low loss and result is a much higher loss value thus overshooting the desired training epoch. A solution to this problem is to keep track of local minimum values for loss and keep one value as a global minimum. Every time a new global minimum is reached the result can be saved and TensorFlow can store the current training stage. This makes it possible to roll back and use the best result of a whole training session. Once this early stop method was implemented, the results were much better and the final loss was greatly minimized.

4.4.3 Adam Optimizer

For the optimizer the built in Adam Optimizer with step of 1e-4 was used. Experiments with bigger and smaller steps were done and a method of decreasing steps as the result get close to minimum was tested, the result of the 1e-4 step was good enough as expected so in the end it was locked in as default for the training.
Screenshots from the terminal during a training session can be viewed in the appendix and below is an example table of how the data from a training session may look like:

Training Session		
Epoch	Current Loss	Early stop
40	83	83
80	0.50	0.47
100	0.25	0.25

Table 4.1: Table showing example epochs with their current loss and the early stop

5
Result

5.1 Object Bounding Box

One of the main focuses of this thesis was to visualize data and extract important information. The object bounding box was one of the final visual results. The bounding box implementation in PyPlot confirmed the correctness of the extracted tracklet data and implementation of different mathematical functions. This implementation also includes on-screen information about the size of each object, their classification and orientation. The figure shown below was the final implementation of object bounding box annotation in PyPlot.

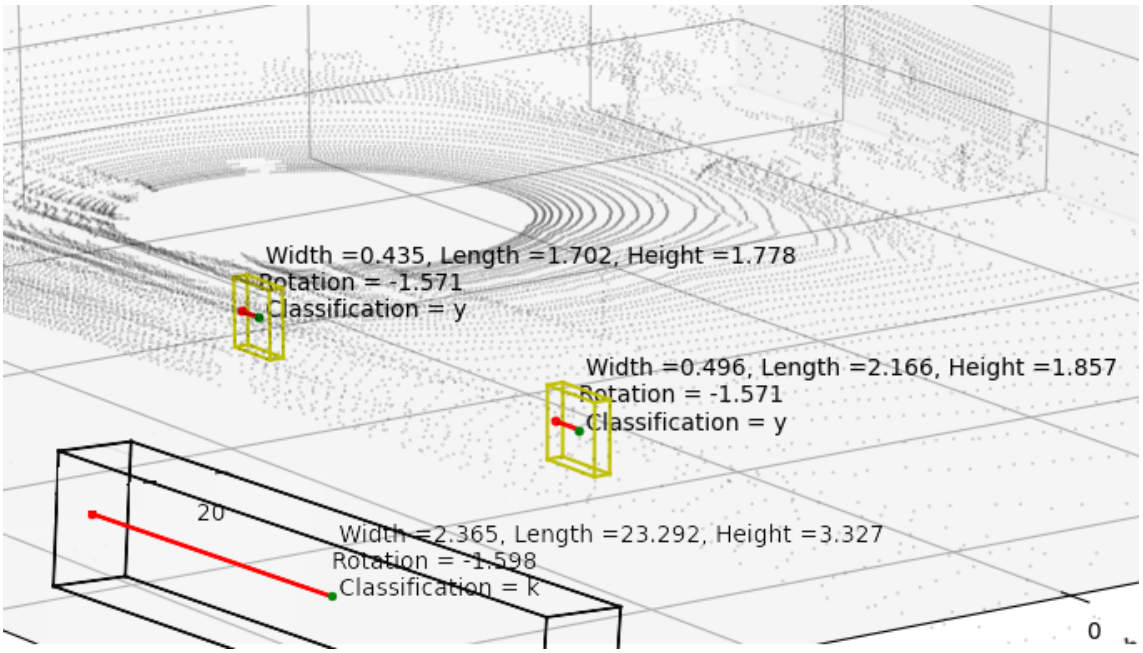

Figure 5.1: Screenshot of Object bounding box annotation in pyplot.

Once the bounding box implementation worked with pyplot the next step was combining the bounding box data with the pointcloud2 live data. This was achieved using the subscriber node in order to retrieve information which was then processed together with the tracklets containing information about each object. The result included the center of each object colored based on their classification and a box constructed of the eight edges calculated using the object's size and orientation. The

outputted data was then projected onto the pointcloud2 using the publisher nodes and this was the achieved object bounding box annotation in RVIZ. The following figure displays the final result of the object bounding box annotation in RVIZ which was the final step of starting to use the deep neural network.

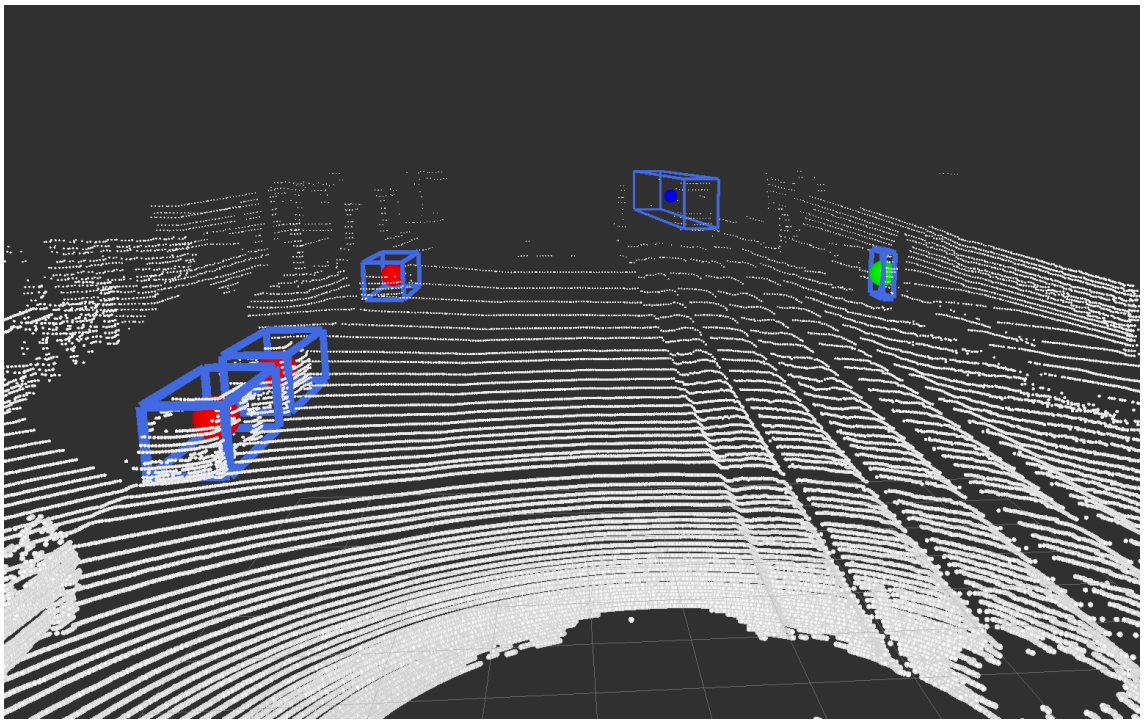

Figure 5.2: Final version of object bounding box annotation in RVIZ.

5.2 Generating reflectance using DNN

Final step of this thesis includes training a DNN and visually comparing it to original data. Once the DNN was designed it was time to train the DNN and generate the data. After some experimenting with different number of epochs and batches the following results were created. In-order to visually compare the data a frame with original reflectance will be illustrated on the left side while the generated data will be shown on the right side. Note that in the first picture were the DNN was only trained for 100 epochs and the difference compared to the original image is very noticeable (Loss of 85%). The pattern is basically a made of random noise and no co-relation could be found. Once the DNN was trained with around 10.000 epochs on the other hand the similarities were clear and the concept was proven. The loss at this point was around 5%.

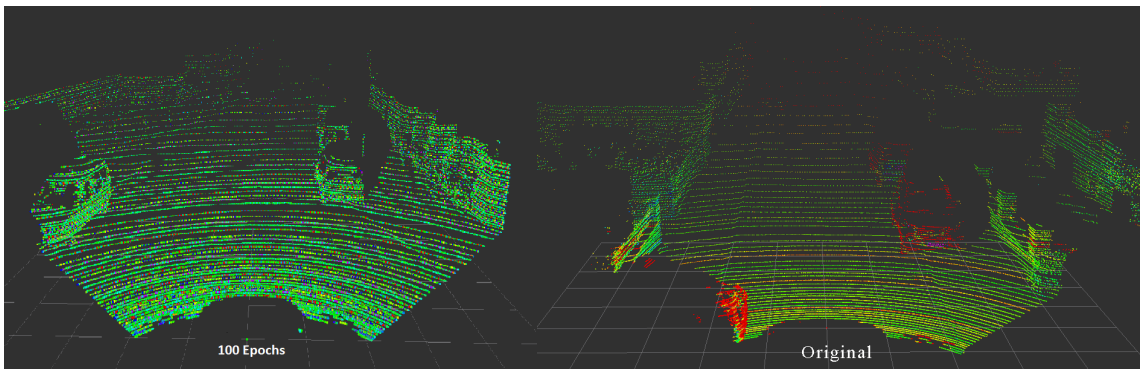

Figure 5.3: Generated data after 100 epochs compared to original data.

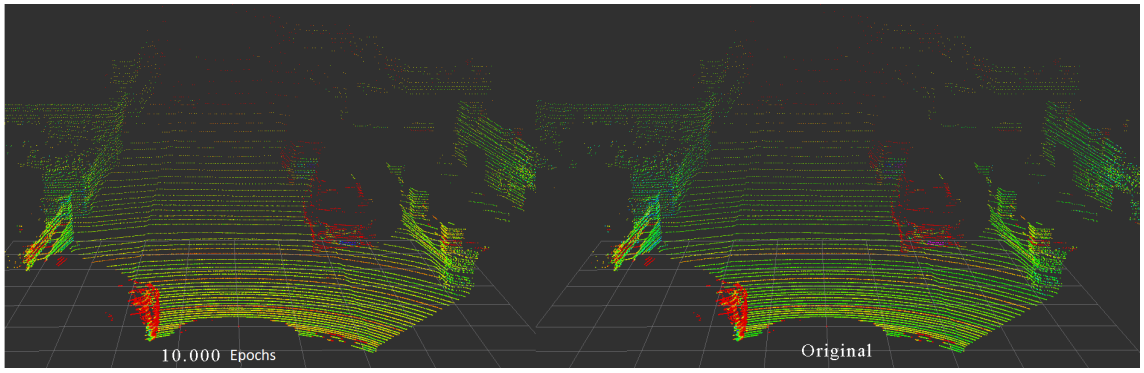

Figure 5.4: Generated data after 10000 epochs compared to original data.

6
Discussion

This section will discuss different aspects of the thesis and the end result of it.

6.1 Workflow

Due to the thesis requiring lots of knowledge about Deep neural networks and the ROS framework, it was planned that the first phase of the workflow was dedicated to learning ROS and DNN. Installing and setting up an environment with ROS framework was very time consuming which significantly stretched the first phase of learning. Once the environment was successfully setup and the learning process had begun the KITTI database was introduced which would be used as main testing data for further learning the ROS framework. During this time the thesis consisted of only a few pages which was a result of the time dedication to the learning phase. At this point it was crucial to try and achieve an adequate result in order to be able to construct a satisfactory thesis.

This step of the process consisted of setting up an environment compatible with correct version of Tensorflow, Ros and Python which proved quiet tedious. Once the environment was ready multiple days were dedicated into designing and creating a deep neural network, implementing the model and training it. At the end the result was more than adequate and the rest of the time was dedicated into writing the thesis. The following Figure illustrates the rough timing and partitions of the workflow.

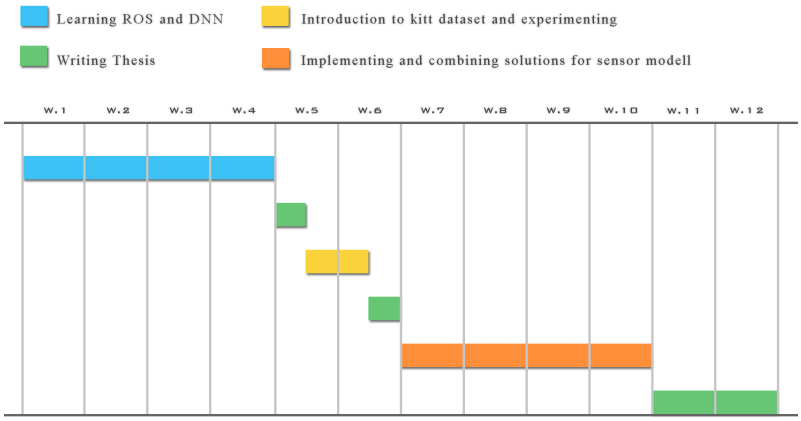

Figure 6.1: Gant schema matching the actual workflow of the thesis.

6.2 Environmental Aspects

The thesis does not go all the way to making the environmental aspect a reality however it builds the foundation for it. When knowing how LiDAR data work and how to build and train a DNN an application for it could be used to simulate it and try to develop it into real life. Because gathering data from simulations instead of real life would require significantly require less energy. In both cases the computer would still need to do loads of work.
In a simulated environment different scenarios could be explored by modifying the current data. This eliminates the need for a car with cameras driving around to gather new data.

6.3 Ethical Aspects

The DNN generated sensor data has decent accuracy if compared to actual sensor data which means it can be used for testing and simulation. As with anything autonomous that could affect human life it should be handled with great care and caution. This sensor data will mainly be used for conducting non-commercial tests due to all the error possibilities and low sample size. Once the DNN has been improved and trained on a much larger data set and been proven to be accurate, can the consequences and ethics be considered. For this to be considered, there must also be a need for commercial usage. Sensor data can have a serious impact in an autonomous system which creates the necessity of human supervision.

6.4 Result of thesis

The results may not look like much in regards to the applications towards the field of LiDAR. It does however show good knowledge and understanding of the field. This knowledge can be used to further explore and develop more advance solutions, in both DDNs and later simulations.

7
Conclusion

The purpose that was set out to be completed was in the end done to satisfaction. The learning curve was steep and with each new software a new level of complexity was introduced. The method of doing basic tutorials and working through simpler tasks, in order to fully understand how to use the tools in regards to this thesis was indeed a good way to go about this. As the field of autonomous driving with the use of LiDAR sensors is still growing and is constantly begin developed, the workflow ran into some troubles. Even though many of the tutorials were outdated, they went through the fundamental knowledge needed and they were the ones recommended by experienced engineers. So for each tutorial the softwares had to be installed on a compatible version. This work was both tedious and unfortunate and could possibly have been avoided by planning which versions were going to be touched in the thesis. Although some of the knowledge of which versions worked came during the learning phase, all non-working versions could not be avoided.

7.1 Visualization

The creation of the different visualizations worked as checkpoints to show that each part was understood and a step closer to being able to use the whole dataset in a meaningful way. The first visualization where the whole dataset was converted to a bagfile and played on RVIZ was a base for the rest of the visualizations. Going from basic tutorial visualizations to manipulate the LiDAR data to only show some parts of it was the first real step towards gaining some results towards our objective and, as it was completed a big puzzle piece of understanding the LiDAR dataset was in the right place.

Once the data was well-explored, adding new tools to use the data to in a more meaningful way was introduced, in the form of object recognition using bound box annotation. Fetching the data and plotting it was a big win but getting it to show as the visualisation of the whole scene would prove challenging. Using the marker tool and ROS nodes the data was transferred onto RVIZ, forming boxes around cars, bikers and trams in the LiDAR scene. While running the scene with boxes, there was a delay which resulted in the boxes starting to fall behind drawing boxes of cars where no cars were shown on the LiDAR. This error took a while to resolve because there were no errors in the code. The error occurred as the rosbag loop started again, the subscriber node gathering the data for the point cloud visualiza-

tion was got its data faster then the tracklet data that the boxes got its data from. The solution was turning down the rate of which the visualization was shown and setting a flag in the code to let both run through the whole scene before visualizing anything new.

7.2 Final thoughts

At this point the ROS system was well-explored, the Python scripts for working and visualizing the LiDAR data functioned as planned and only touching up the DDN remains. Understanding how the DNN functions and using TensorFlow was very far apart, running tutorials and courses online gave results almost instantly while creating something from scratch required plenty of time and modifications. Understanding how to tweak the code and making it suit the project took more time and a big help was the experienced engineer setting up roadblocks that needed to be conquered before going further. This led to a well thought-out design, and a result very close to what was set up in the goals, however as this was learnt a feeling arose that only the surface of the world of DNN was touched.

Bibliography

[1] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[2] Open-Robotics, "About ros," Available at https://www.ros.org/about-ros/ (2020/04/20).

[3] A. Ghatak, "Ros tutorials," Available at http://wiki.ros.org/ROS/Tutorials (2020/04/28).

[4] Numpy, "Numpy," Available at https://numpy.org/ (2020/05/10).

[5] W. Woodall, "rviz," Available at http://wiki.ros.org/rviz (2020/05/28).

[6] Python-Software-Foundation, "Python," Available at https://www.python.org/ (2020/04/22).

[7] G. Hoorn, "Rospy," Available at http://wiki.ros.org/rospy (2020/04/22).

[8] Matplotlib, "Pyplot," Available at https://matplotlib.org/api/pyplot_api.html (2020/04/22).

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Sensor setup," Available at http://www.cvlibs.net/datasets/kitti/setup.php (2020/05/28).

[10] Tomas789, "Kitti2bag," https://github.com/tomas789/kitti2bag.

[11] TensorFlow, "Tensorflow," Available at https://www.tensorflow.org/ (2020/05/29).

A

Appendix 1

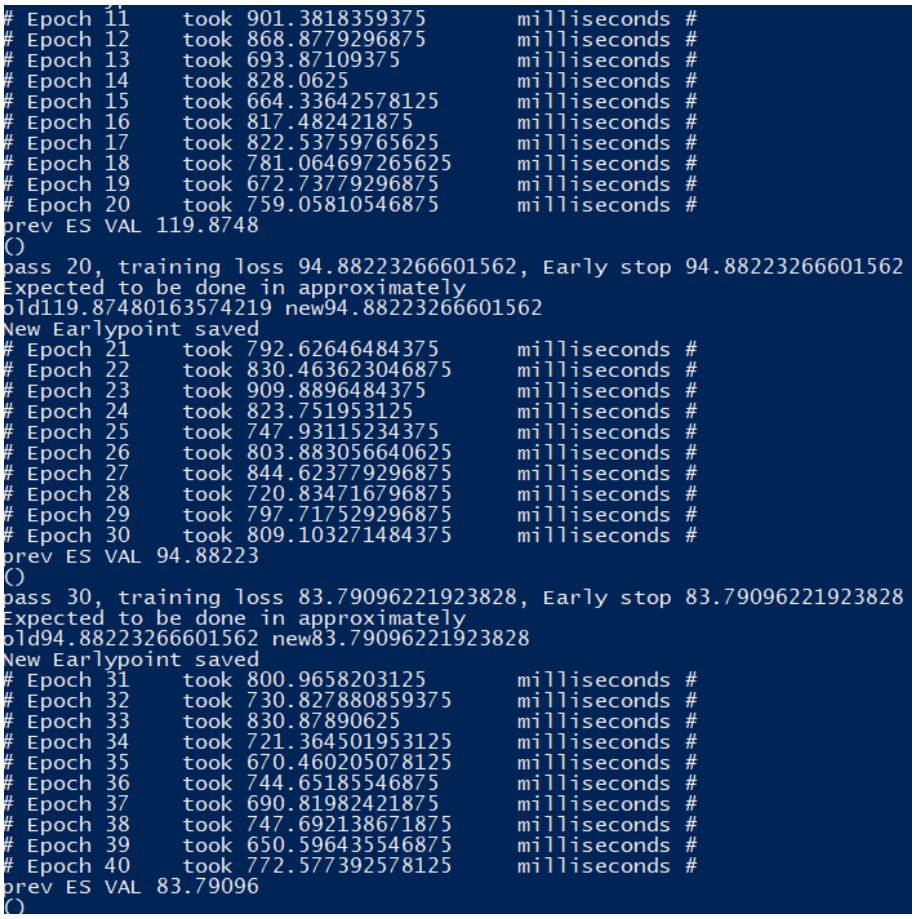

Figure A.1: Screenshot 1 from terminal window when running CAE training session.

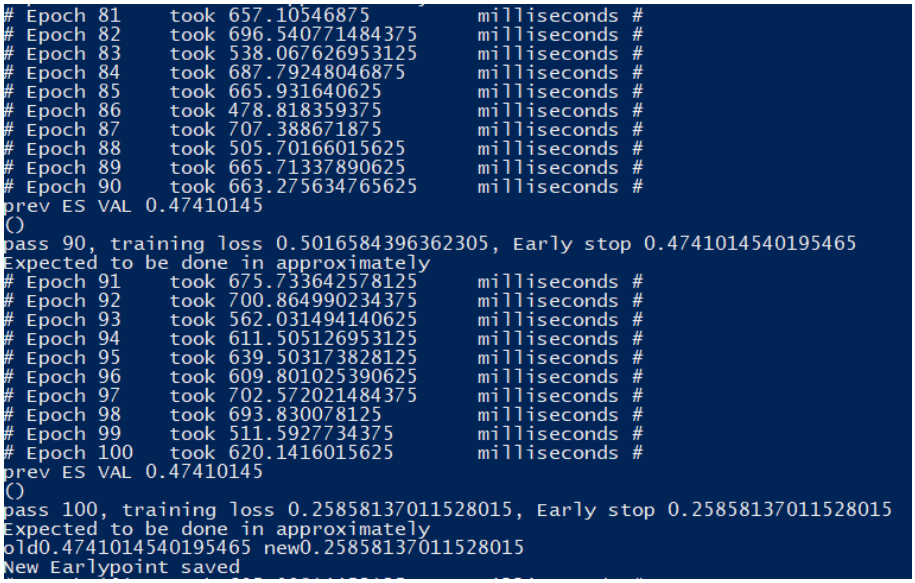

Figure A.2: Screenshot 2 from terminal window when running CAE training session.

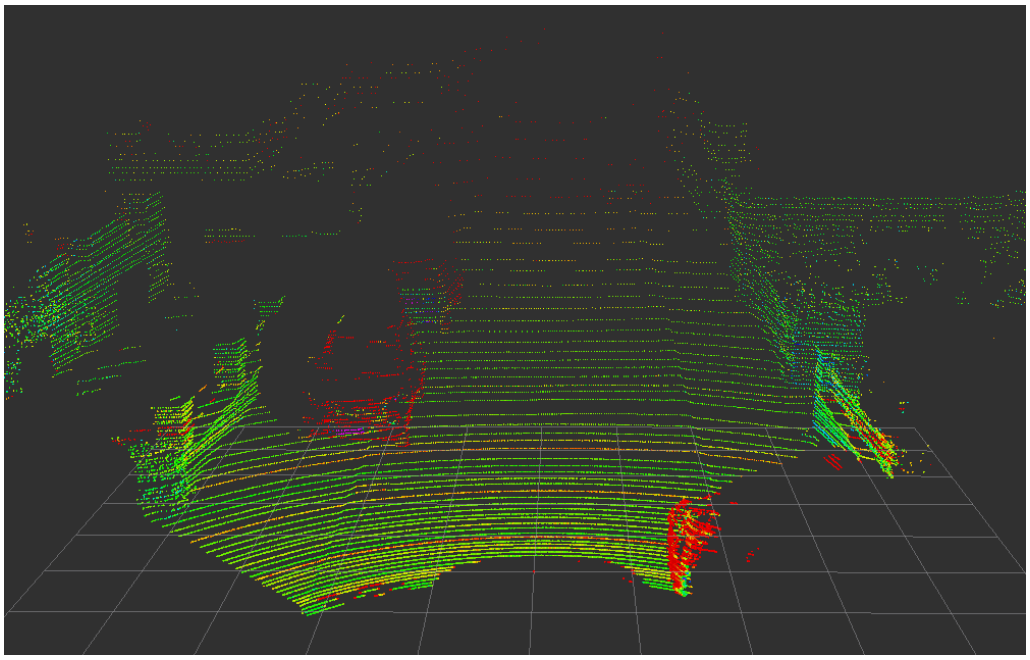

Figure A.3: Example 1 of original frame's intensity displayed in RVIZ.

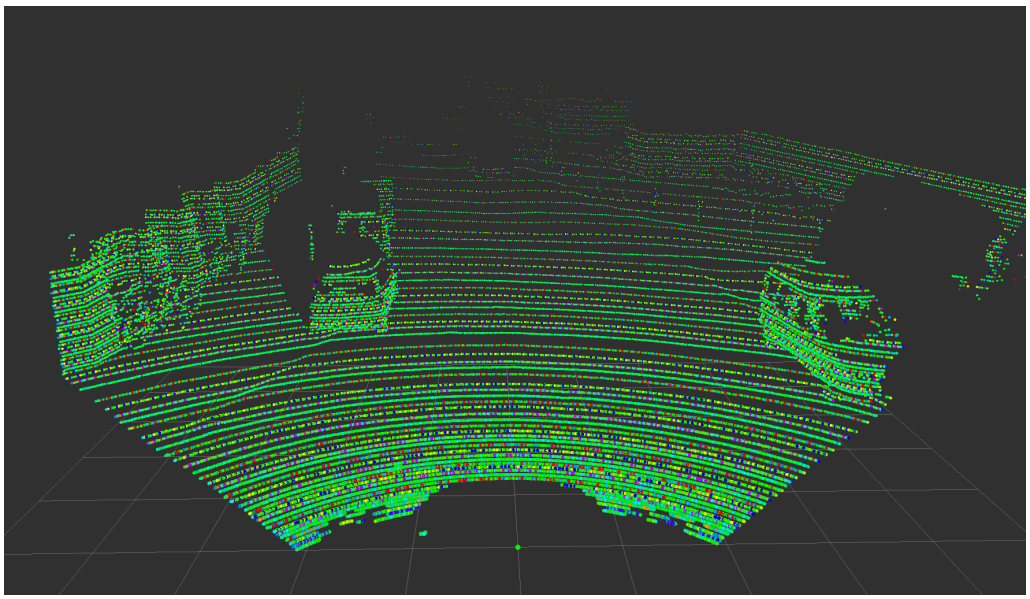

Figure A.4: Example 1 of generated frame's intensity by DNN in early stage of training displayed in RVIZ. (Loss 0.75)

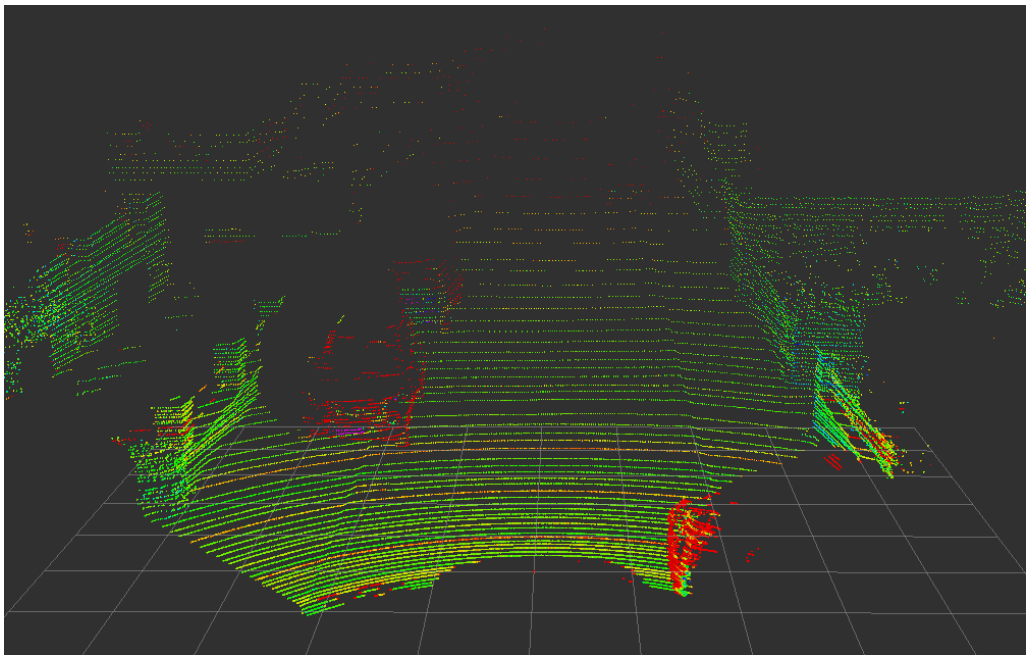

Figure A.5: Example 2 of original frame's intensity displayed in RVIZ.

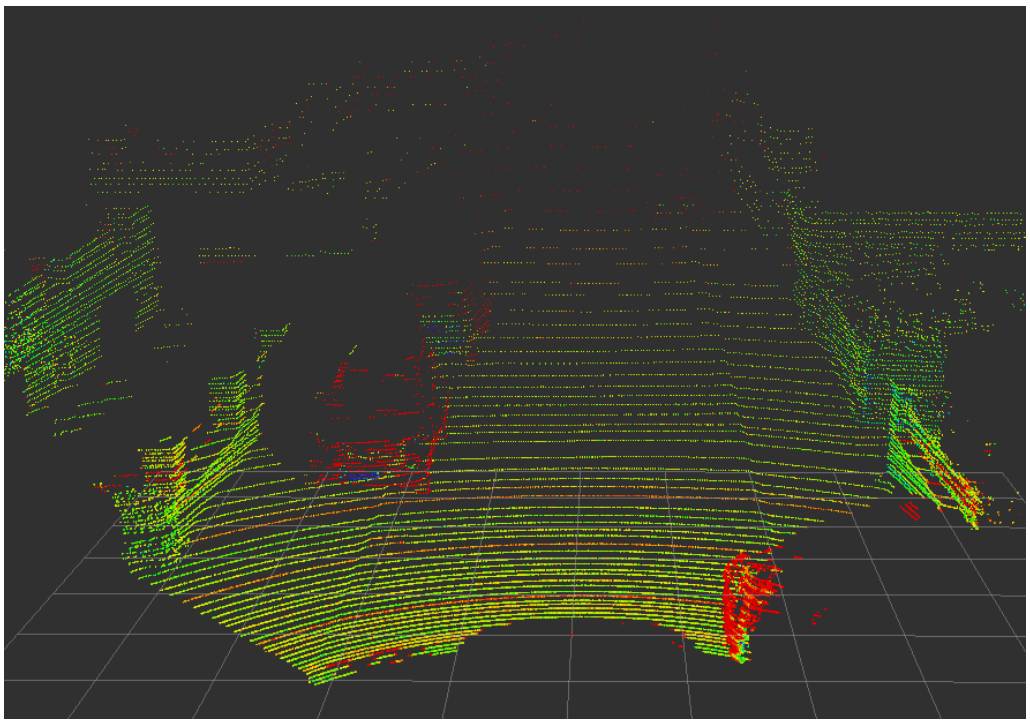

Figure A.6: Example 2 of generated frame's intensity by DNN in final stage of training displayed in RVIZ.(Loss 0.05)