REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

3D LIDAR BASED ODOMETRY WITH CNN-RNN DEEP LEARNING NETWORK-SIMULATION

17011901 — Muhammed Johar

COMPUTER PROJECT

Advisor
Asst. Prof. Erkan USLU

October, 2022

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank everyone who contributed to the success of this project, especially Professor Asst. Prof. Erkan USLU who supported me and was by my side throughout this project.

Muhammed Johar

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
6DOF	Six Degrees Of Freedom
AI	Artificial Intelligence
GPS	Global Positioning System

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

3D LIDAR Based Odometry with CNN-RNN Deep Learning Network-Simulation

Muhammed Johar

Department of Computer Engineering
Computer Project

Advisor: Asst. Prof. Erkan USLU

In recent years, it's become greatly noticeable how self-drive vehicles and autonomous devices have started to become the main part of our lives. In that sense, every autonomous vehicle requires four essential technology keys: navigation system, path planning, environment perception, and control[1]. Those keys are required to obtain an autonomous state.

In addition; there have been an increased interest in Artificial Intelligence (AI) due to its potentials in practical applications, particularly its subsets Machine Learning (and transitively, Deep Learning).

This project aims to make use of deep learning techniques to produce a model that takes the readings data of a 3D LiDAR scanner as input and estimates as output the change in two poses indicated by LiDAR scans in form of six values that represent the change in 6DOF of a vehicle. A virtual environment is prepared to test the model. The proposed method can be a replacement for position information of the wheel encoders of GPS data when the data is absent.

Keywords: Odometry, Simulation, 3D LiDAR, neural networks

1
Introduction

Localization is one of the fundamental tasks for autonomous robots, which is described as the process of determining where a mobile robot is located with respect to its environment. In that sense; The Odometry provides information about the changing in position and rotation of a robot between two-time instances.

The Odometry of a mobile robot can be extracted by processing the readings coming from sensors. those sensors can be categorized into two groups first category proprioceptive sensors(e.g. encoders, accelerometers, etc.) which provide motion information for the robot. the second category is exteroceptive sensors(e.g. GPS, LiDAR, sonar, camera, etc.) which provide information about the surrounding environment[2].

In this project, we consider extracting the Odometry from the reading provided by the 3D LiDAR sensor. the approach has taken to reach this goal is by exploiting the capability of ML techniques to extract meaningful patterns from raw data in particular conventional neural networks.

In the next chapter, we provide a preview of 3D-LiDAR and conventional neural networks for better understanding.

2
Preview

2.1 LiDAR Sensor

LiDAR as a word is shortcut for (Light Detection And Ranging), sometimes (Light Imaging Detection And Ranging) for the image-like resolution of modern 3D sensors[3]. As a technology, it uses a vertically arranged laser emitters turn around a vertical axis constantly, by gathering 360 rotation of laser beams readings to create a 3D representation of the scanned environment. A typical LiDAR sensor emits pulsed light waves from a laser into the environment. These pulses bounce off surrounding objects and return to the sensor. The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it traveled. the following table shows the specification for 12 of 3D-LiDARs from 4 different manufacturers.

	Velodyne						Hesai		Ouster	RoboSense
	 VLS-128AP*[1]	 VLS-128*[2]	 HDL-64S2[3]	 HDL-32E[4]	 VLP-32C[5]	 VLP-16[6]	 Pandar-64[7]	Pandar-40p[8]	 OS1-64[9]	 RS-Lidar32[10]
Channels	128	128	64	32	32	16	64	40	64	32
FPS[Hz]	5-20	5-20	5-20	5-20	5-20	5-20	10,20	10,20	10,20	5,10,20
Precision[m]	±0.03	±0.03	±0.02[a]	±0.02	±0.03	±0.03	±0.02[c]	±0.02[c]	±0.03[d]	±0.03[c]
Max.Range[m]	245	300	120	100	200	100	200	200	120	200
Min.Range[m]			3	2	1	1	0.3	0.3	0.8	0.4
vFOV[deg]	40	40	26.9	41.33	40	30	40	40	33.2	40
vRes[deg]	0.11[b]	0.11[b]	0.33[a]	1.33	0.33[b]	2.0	0.167[b]	0.33[b]	0.53	0.33[b]
λ[nm]	903	903	903	903	903	903	905	905	850	905
ϕ[mm]	165.5	165.5	223.5	85.3	103	103.3	116	116	85	114
Weight(kg)	3.5	3.5	13.5	1.0	0.925	0.830	1.52	1.52	0.425	1.17
Pointcloud fields from driver	x, y, z, intensity, ring, timestamp	x, y, z, intensity, ring, timestamp	x, y, z, intensity, ring	x, y, z, intensity, ring	x, y, z, intensity, ring	x, y, z, intensity, ring	x, y, z, intensity, timestamp, ring	x, y, z, intensity, timestamp, ring	x, y, z, timestamp, intensity, reflectivity, ring	x, y, z, intensity

Figure 2.1 Acronyms are frame rate (FPS), vertical field-of-view (vFOV), vertical resolution (vRes), laser wavelength (λ), $and diameter \phi$.

2.2 Deep Neural Network

Machine learning techniques have been widely applied in various areas such as pattern recognition, natural language processing, and computational learning. By starting with the simplest Artificial Neural Network from, we can classify ANN to the following:

1. Single-layer ANN or The Perceptron
2. Multi-Layer Perceptrons (MLP)
3. Convolutional Neural Networks (CNN)
4. Recurrent Neural Networks (RNN)

2.2.1 Single-layer ANN or The Perceptron

The Preceptron is the simplest Neural Network and considered as the basic building block for other neural networks. The main functionality to preceptron was based on binary inputs, an binary output is produced makes the preceptron an binary classifier algorithm, with present of weights, bias, and activation function non-binary input and output is produced.

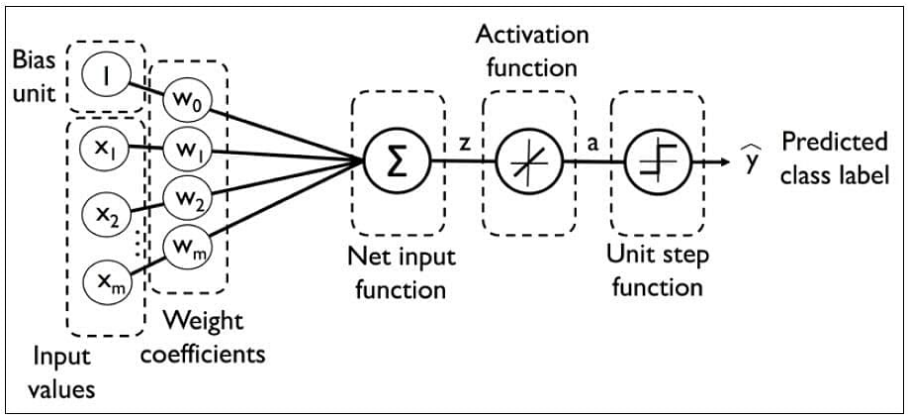

Figure 2.2 The Perceptron

In short, a perceptron is consist of four main parts including input values, weights and bias, net sum, and an activation function.

The process begins by taking all the input values and multiplying them by their weights. Then, all of these multiplied values are added together to create the weighted sum. The weighted sum is then applied to the activation function, producing the perceptron's output.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 2.3 Activation Function Cheatsheet

2.2.2 Multi-Layer Perceptron (MLP)

Multilayer Perceptron (MLP), known as fully connected multi-layer neural network MLPs are the most basic DNNs and classified as feed-forward Artificial Neural Networks(ANN). It has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN. Each layer is a set of nonlinear functions of a weighted sum of all outputs from previous layer. First layer receives input called input layer and last layer takes input called output layer in between of them basically there's hidden layer and could be multiple layers.

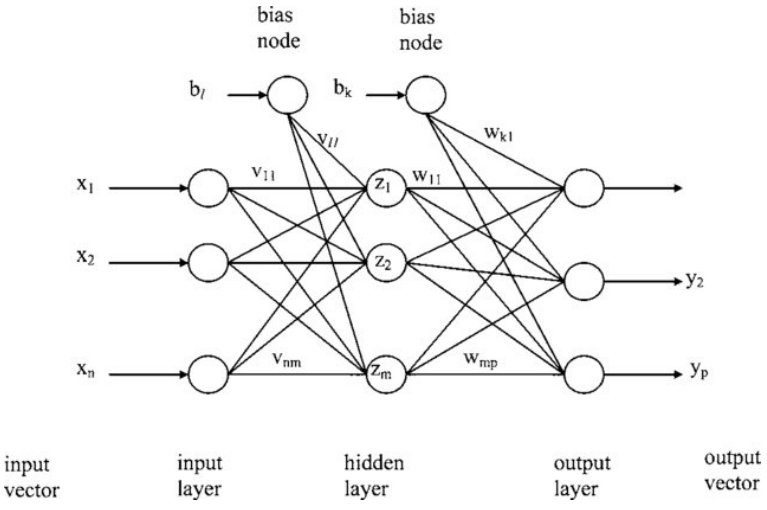

Figure 2.4 Architecture of Multi-Layer Perceptron (MLP)

2.2.3 Convolutional Neural Networks (CNN)

One of the most popular neural networks, mainly deployed to recognize the patterns mostly in images additional to MLP architecture and before the data -represented as pixels and could be other then that- pass to the input layer, they pass through multiple filters to extract the features from the image, those filters generally is composed of convolutional layer, non-linearity layer, pooling layer[4].

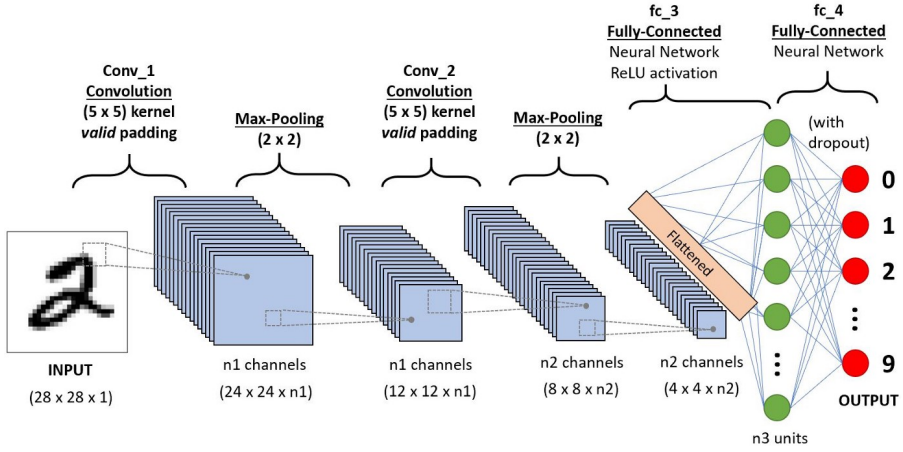

Figure 2.5 Convolutional Neural Networks (CNN)

2.2.4 Recurrent Neural Network (RNN)

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. In this type of these Deep Neural Networks input data flow is very important to solve ordinal or temporal problems in fields such as Natural Language Processing (NLP), language translation, and image captioning. The previous output of this neural network influences along with the current input the result obtained in current output.

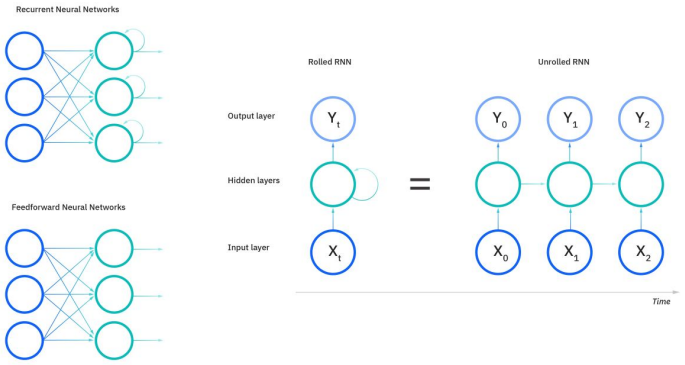

Figure 2.6 Recurrent Neural Network (RNN)

3
Feasibility

In this section, we describe the feasibility study carried out during the development of the project.

3.1 Technical feasibility

Technical feasibility study is the complete study of the project in terms of input, processes, output, fields, programs and procedures.

3.1.1 *Software feasibility*

The operating systems, and development environments are explained in 3.1. part of

Table 3.1 Software options

	Option 1	Option 2
OS	MacOS	Linux
Language	C++	Python
IDE	Visual Studio Code	Pycharm

this project is developed using Python due to its wide variety of "simple to use" and high performance libraries capable of performing complex mathematical operations. As for the operating systems, the simulation and ROS framework will be runnuig in Linux environment, the model could be build and trained either in MacOS or in CoLab environment from Google.

3.1.2 *Hardware feasibility*

Since training a CNN is a compute-intensive task, a machine with a dedicated GPU is recommended for a better performance. However, once the trained model is acquired, the system is expected to run smoothly on any mid-range computer. 3.2.

Table 3.2 Hardware requirements for a healthy system operation

	Option 1	Option 2
Processor	Apple M1	Core i7-2450M
RAM	8 GB	16 GB
Storage	256 GB	512 GB
GPU	M1 GPU	AMD Radeon Pro 560

3.2 Communication feasibility

Git along with GitLab is used for project collaboration, version control and source control. Also, online virtual meetings are conducted using Zoom software.

3.3 Legal feasibility

The project complies with existing laws and regulations, any patents and so on, and does not violate any protected rights.

3.4 Economic feasibility

Expenditures made on behalf of the project are shown in Table 3.3.

Table 3.3 Expenditures table

	amount	Price per unit	Total price	Brand
Computer 1	1	10000 TL	10000 TL	MacBook Air M1
Computer 2	1	7000 TL	7000 TL	MacBook Pro 2017
Software Developer Expense	1	200 TL (daily)	17200 TL	-
Total	-	-	34200 TL	-

Note: Since some development software and platforms are either open-source or provided through student licenses, there has been no expenditure on behalf of software.

3.5 Time feasibility

The design and implementation process of the project is as specified in the Gantt chart in Figure 3.1.

Figure 3.1 Gantt chart

4
System Analysis

In this chapter we will work on tasks of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

4.1 Dataset

The dataset used to train the model is taken from The Kitti Vision website under odometry tab we can find odometry data set (velodyne laser data, 80 GB). The dataset contains 22 sequences. first 11 sequences (00-10) are provided with grand truth trajectories for training and the rest sequences (11-21) are dedicated to testing.

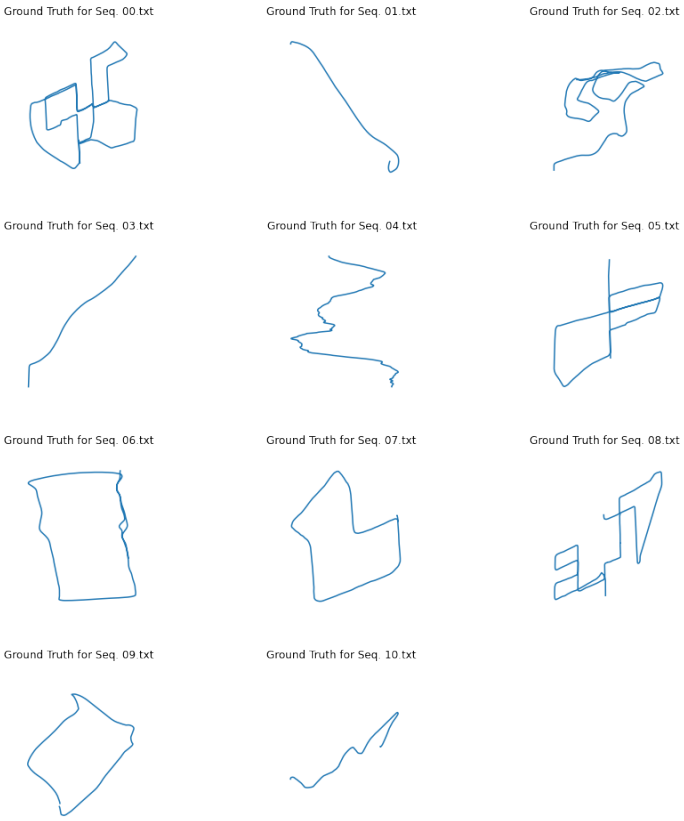

Figure 4.1 The Ground Truth For First 11 Sequences of KITTI dataset

4.1.1 Velodyne LiDAR Setup

The Ground Truth in kitti dataset recorded according to the left gray camera(cam0). The sensors platform setup as showed below:

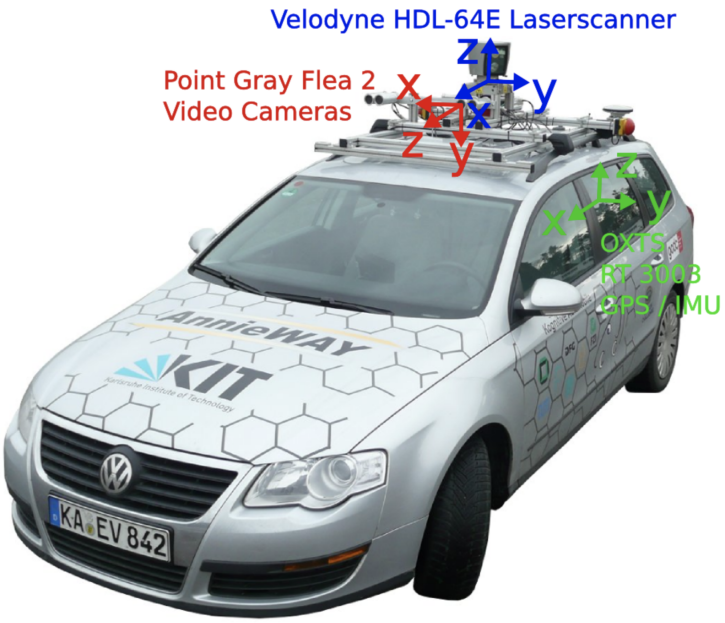

Figure 4.2 Sensors Platform Setup

For that reason the cloud points information will be diffident from the ground truth:

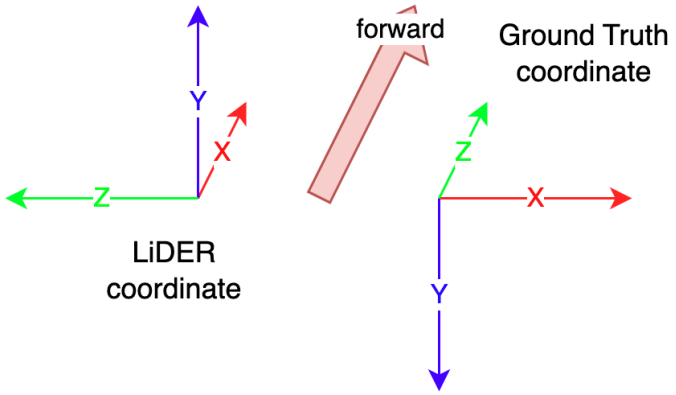

Figure 4.3 Coordinates

so if we will work on the cloud point directly we have to, apply transformation using calib.txt file to rectify the coordinates. and we will get result as the following:

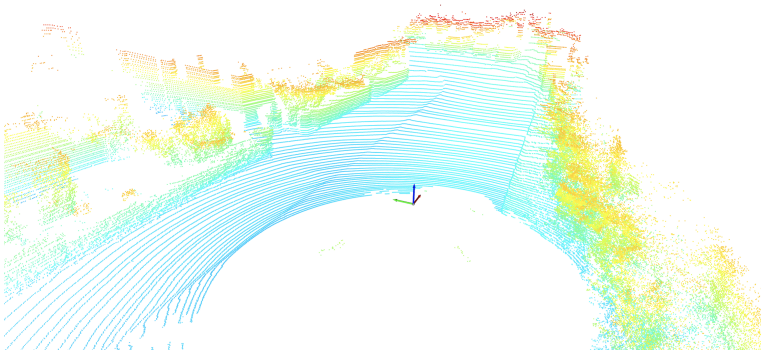

Figure 4.4 Cloud Points Before Rectify

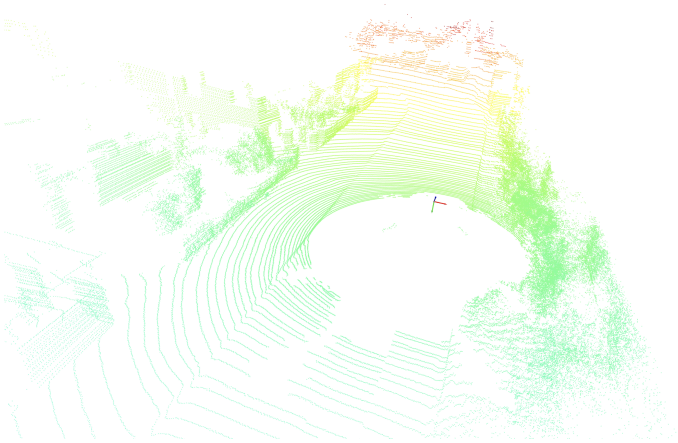

Figure 4.5 Cloud Points After Rectifying

In the next chapter we will talk about data transformation of the cloud points to 2d image spherical projection

5
System Design

In this Chapter kitti dataset transformation will be discussed.

5.1 The Kitti Dataset Transformation

5.1.1 Cloud Points Transformation:

The Kitti Dataset contains Velodyne LiDAR reading as cloud points in form of (x,y,z) and the intensity value of the laser beams which represent the reflectance of the surfaces in the surrounding environment. the transformation that will be applied is transform the value of the points in the cloud-points from (x,y,z) representation form in Cartesian coordinate system to to (R,pitch,raw) representation form in spherical coordinate system. The next step is to apply spherical projection mapping to map every point in the cloud to the corresponding pixel.

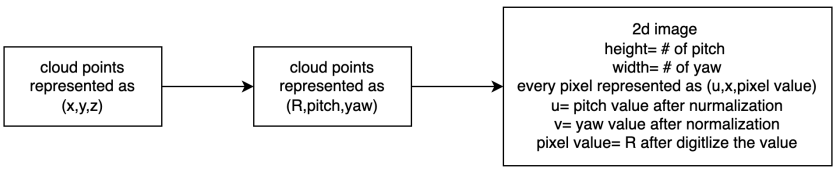

Figure 5.1 Cloud Points Transformation Diagram

the result of this transformation is showed below:

Figure 5.2 First Frame of seq. 00 after the transformation

We can have several representation to the cloud point by changing the pixels values to be the distance in x,y plane which give us the range representation, z value which give us the height representation, and the intensity value give us intensity representation. The result of these transformations is showed respectively below:

Figure 5.3 The Range frame of First Frame in seq. 00

Figure 5.4 The height frame of First Frame in seq. 00

Figure 5.5 The intensity frame of First Frame in seq. 00

Another transformation could be used in our model which is called bird eye view, the result of this transformation is showed as below:

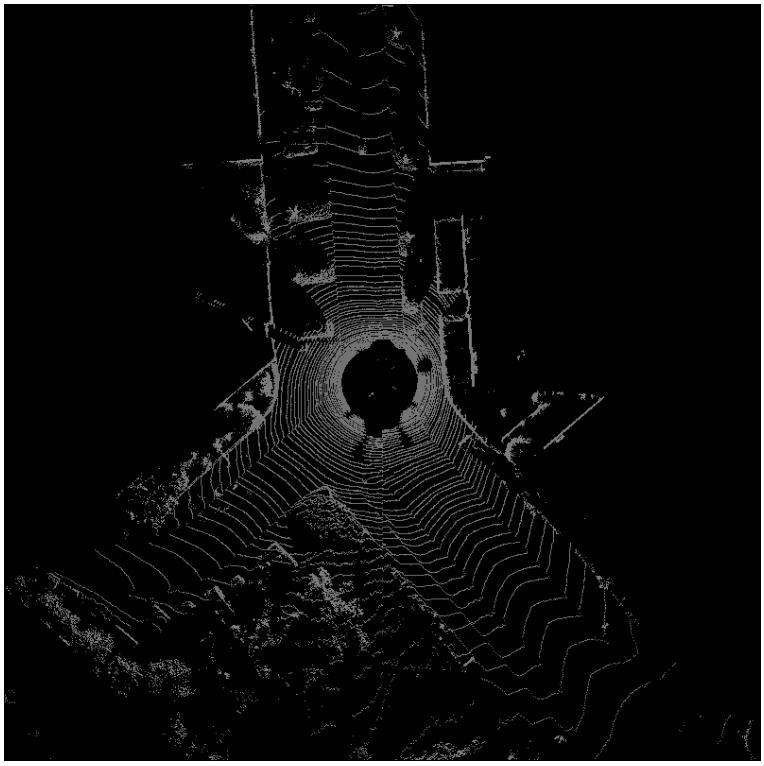

Figure 5.6 The Bird Eye View frame of First Frame in seq. 00

5.1.2 Ground Truth Transformation:

The ground truth values of The Kitti Dataset are stored as trajectories from the origin to the corresponding read. First, we will extract the translation and rotation of the trajectories as 6DOF (x,y,z,α,β,γ) where Alpha is the rotation angle about X axis, Beta is the rotation angle about Y axis, and Gamma is the rotation angle about Z axis.

The result of this transformation for sequence 00 and sequence 01 as the following:

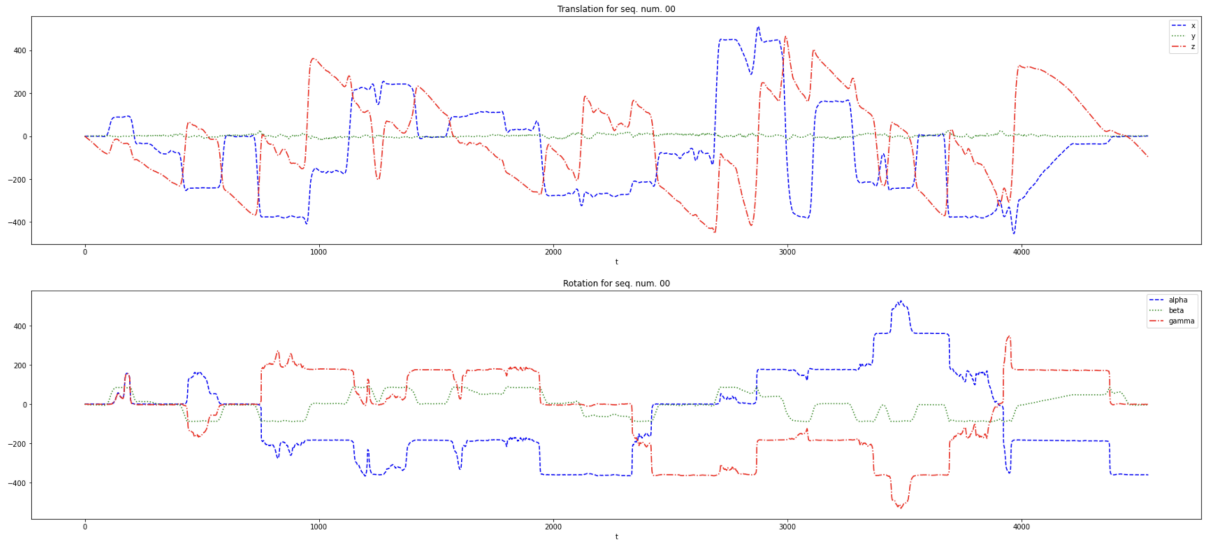

Figure 5.7 Plot of Ground Truth of seq. 00

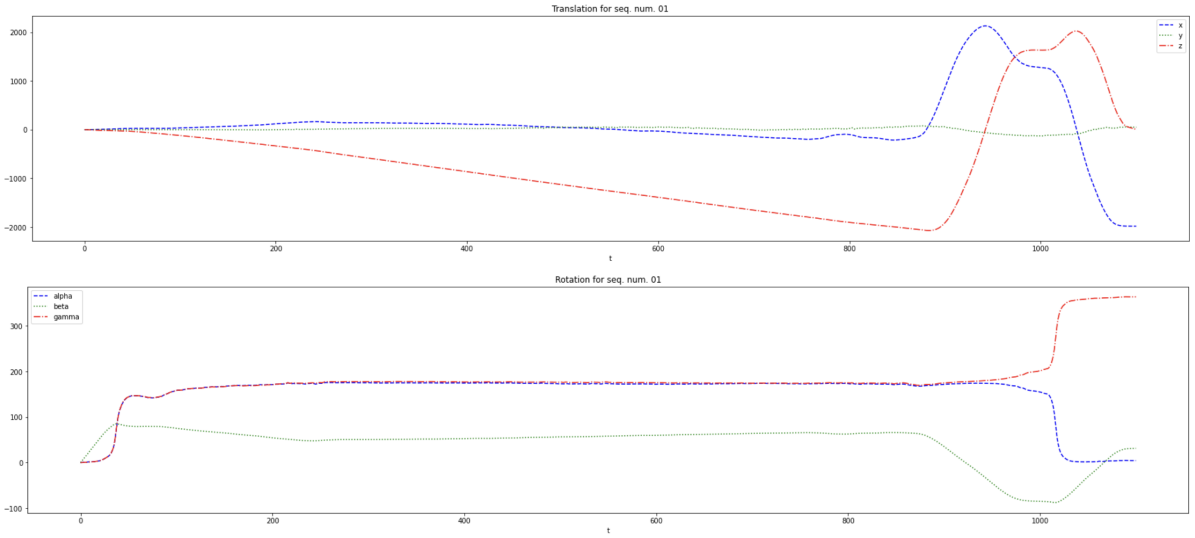

Figure 5.8 Plot of Ground Truth of seq. 01

We can notice that the value of y is almost constant and the values of Alpha and Gamma are bouncing between negative and positive extremes. Since the car is moving in horizontal plane. We can discard y, Alpha, and Gamma values because they don't give any valuable information but they could prevent the model from learning. Thus, Ground Truth's values will be x, z, and Beta.

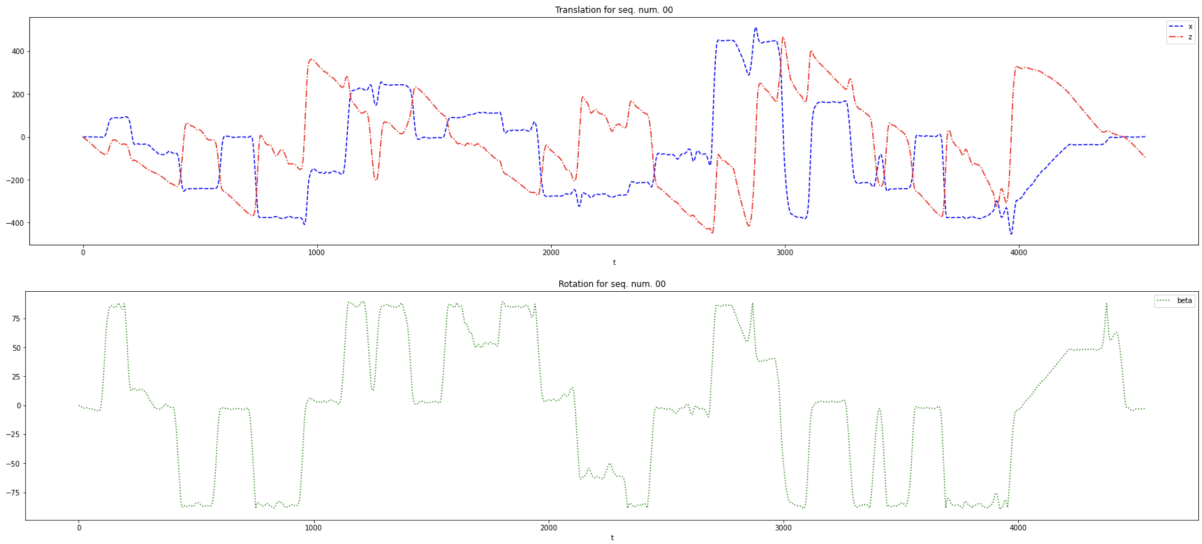

Figure 5.9 Plot of Ground Truth of seq. 00 after discarding y, Alpha,and Gamma

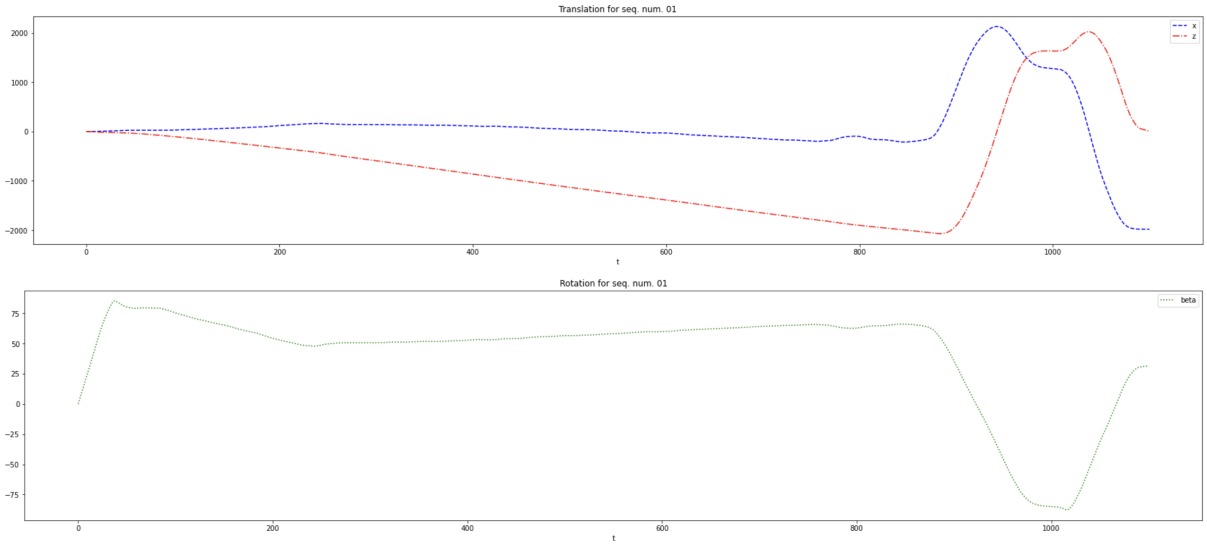

Figure 5.10 Plot of Ground Truth of seq. 01 after discarding y, Alpha,and Gamma

5.2 KittiDataset Class Design

To obtain the ground truth and the paths of images A data-frame was prepared to organize the accessibility to every image and its ground truth. the columns of this dataframe are:

- seq_num : indicates to the number of the sequence.
- ground_truth: list of (x,y,z,α,β,γ).
- bird_eye_view: full path of bird_eye_view images.
- height: full path of height images.
- img360: full path of img360 images.
- intensity: full path of intensity images.
- β: value of β angle about y axis.
- x: value of x.
- z: value of z.

KittiDataset take as arguments: the dataframe , list of desired sequences, bin_size which is the number of classes of the ground truth after calculate the distance between every two consecutive ground truth values and quantize these distances. The Ground Truth is separated to:

1. Translation Ground Truth:
   $distance(GTTrans_t, GTTrans_{t+1}) = \sqrt{(x_t - x_{t+1})^2 + (z_t - z_{t+1})^2}$

2. Rotation Ground Truth: newline $d(GTRot_{t+1}, GTRot_t) = \beta_t - \beta_{t+1}$

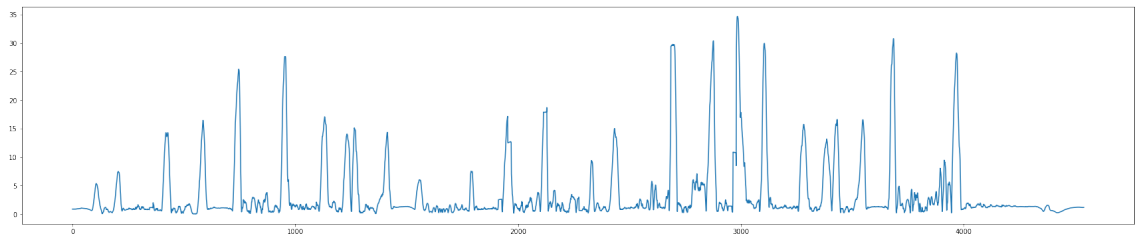

Figure 5.11 Plot of Translation Ground Truth of seq. 00 before quantization.

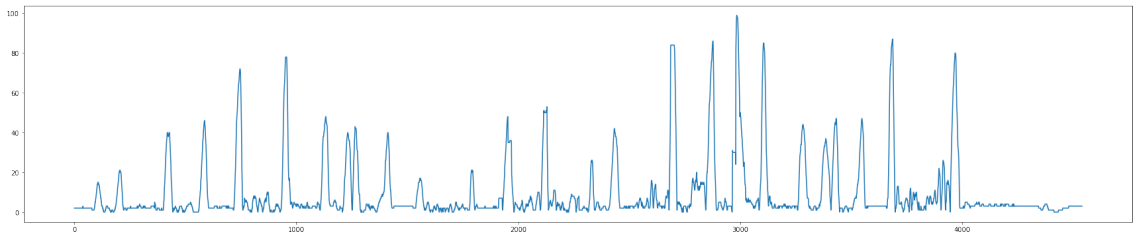

Figure 5.12 Plot of Translation Ground Truth of seq. 00 after 50 bins quantization.

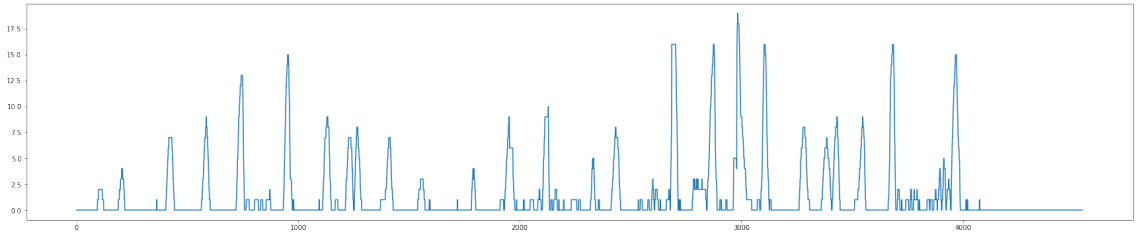

Figure 5.13 Plot of Translation Ground Truth of seq. 00 after 20 bins quantization.

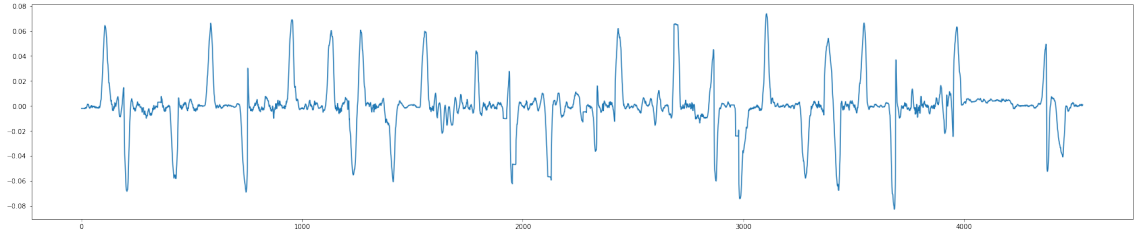

Figure 5.14 Plot of Rotation Ground Truth of seq. 00 before quantization.

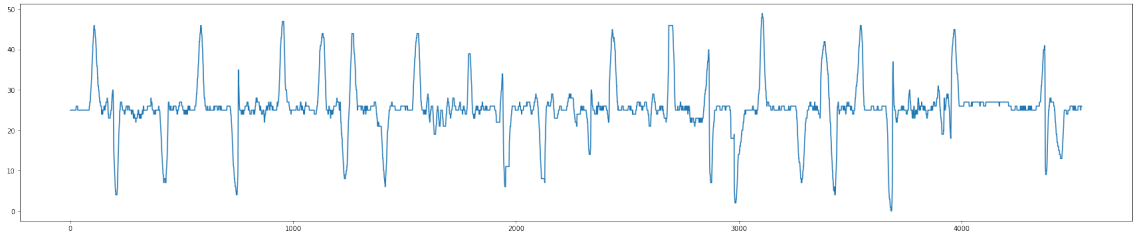

Figure 5.15 Plot of Rotation Ground Truth of seq. 00 after 50 bins quantization.

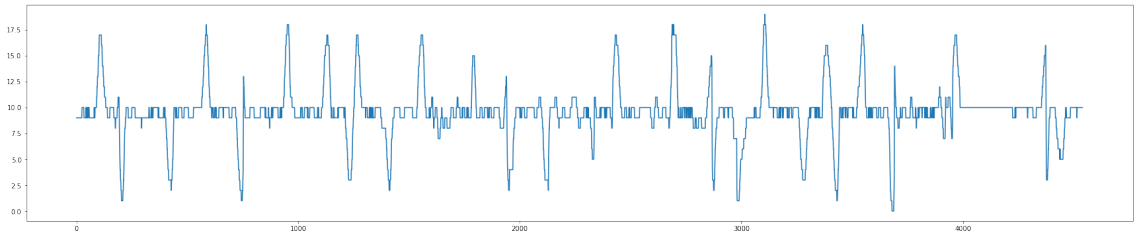

Figure 5.16 Plot of Rotation Ground Truth of seq. 00 after 20 bins quantization.

6
Implementation

After trying different model architectures end up with failure, the recent architecture gives unfitting results.

6.1 CNN ONLY Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 65, 1025]             608
       BatchNorm2d-2         [-1, 32, 65, 1025]              64
              ReLU-3         [-1, 32, 65, 1025]               0
         AvgPool2d-4          [-1, 32, 32, 512]               0
           Dropout-5          [-1, 32, 32, 512]               0
            Conv2d-6          [-1, 64, 16, 256]          18,496
       BatchNorm2d-7          [-1, 64, 16, 256]             128
              ReLU-8          [-1, 64, 16, 256]               0
         AvgPool2d-9            [-1, 64, 5, 85]               0
          Dropout-10            [-1, 64, 5, 85]               0
          Flatten-11                [-1, 27200]               0
           Linear-12                   [-1, 64]       1,740,864
      BatchNorm1d-13                   [-1, 64]             128
             ReLU-14                   [-1, 64]               0
          Dropout-15                   [-1, 64]               0
           Linear-16                   [-1, 50]           3,250
          Softmax-17                   [-1, 50]               0
          Flatten-18                [-1, 27200]               0
           Linear-19                   [-1, 64]       1,740,864
      BatchNorm1d-20                   [-1, 64]             128
             ReLU-21                   [-1, 64]               0
          Dropout-22                   [-1, 64]               0
           Linear-23                   [-1, 50]           3,250
          Softmax-24                   [-1, 50]               0
================================================================
Total params: 3,507,780
Trainable params: 3,507,780
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.51
Forward/backward pass size (MB): 63.63
Params size (MB): 13.38
Estimated Total Size (MB): 77.52
----------------------------------------------------------------
```

Figure 6.1 CNN only Model's Summary .

6.1.1 Training Results

Until now the results are not satisfying and we keep trying to improve the model to get better results.

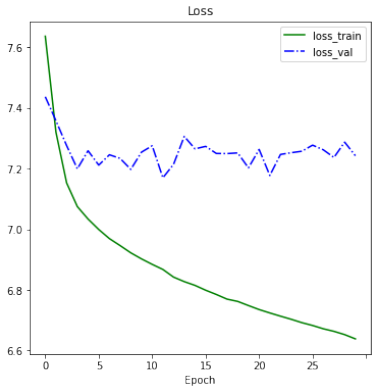

Figure 6.2 CNN only Model's result: loss .

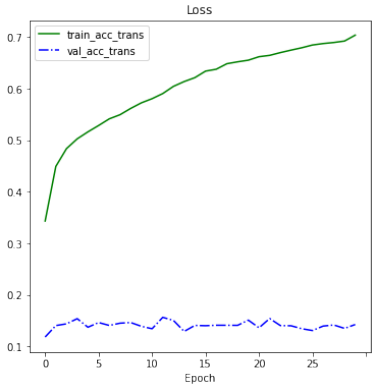

Figure 6.3 CNN only Model's result: translation accuracy.

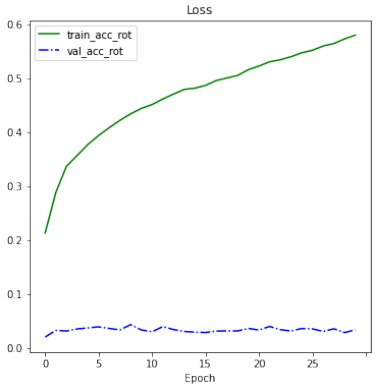

Figure 6.4 CNN only Model's result: rotation accuracy.

6.2 CNN-RNN Model

6.2.1 Training Results

After adding RNN layers to the model the results still not satisfying.

```
OdometryEstimator(
  (conv1): Sequential(
    (0): Conv2d(2, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): AvgPool2d(kernel_size=2, stride=2, padding=0)
    (4): Dropout(p=0.05, inplace=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): AvgPool2d(kernel_size=3, stride=3, padding=0)
    (4): Dropout(p=0.05, inplace=False)
  )
  (rnn_trans): LSTM(27200, 128, num_layers=3, batch_first=True, dropout=0.1)
  (rnn_rot): LSTM(27200, 128, num_layers=3, batch_first=True, dropout=0.1)
  (fc_network_rot): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=128, out_features=64, bias=True)
    (2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): Dropout(p=0.05, inplace=False)
    (5): Linear(in_features=64, out_features=50, bias=True)
    (6): Softmax(dim=-1)
  )
  (fc_network_trans): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=128, out_features=64, bias=True)
    (2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ReLU()
    (4): Dropout(p=0.05, inplace=False)
    (5): Linear(in_features=64, out_features=50, bias=True)
    (6): Softmax(dim=-1)
  )
)
```

Figure 6.5 CNN-RNN Model's Summary .

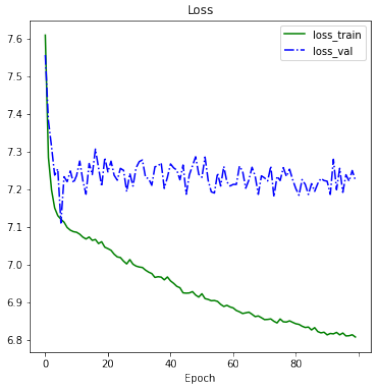

Figure 6.6 CNN-RNN Model's result: loss .

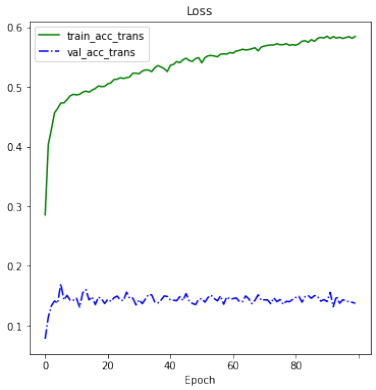

Figure 6.7 CNN-RNN Model's result: translation accuracy.

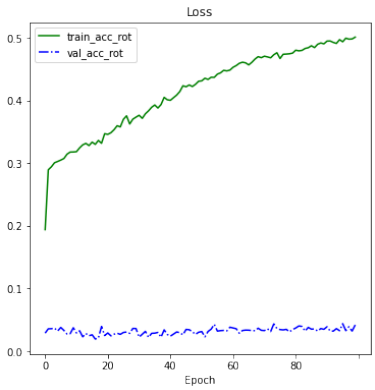

Figure 6.8 CNN-RNN Model's result: rotation accuracy.

6.2.2 Test Results Discussion

After plotting the paths of ground truth values and predicted values. we can see the reason behind the poor result taken from the model. which is the ground truth, precisely the rotation ground truth, was calculated wrong, as a consequence of the misunderstanding of how the Euler Angles were applied to achieve the rotation. the right way to apply the rotation is after every rotation of one of the angles the coordinates must be changed according to that rotation rather than being fixed.

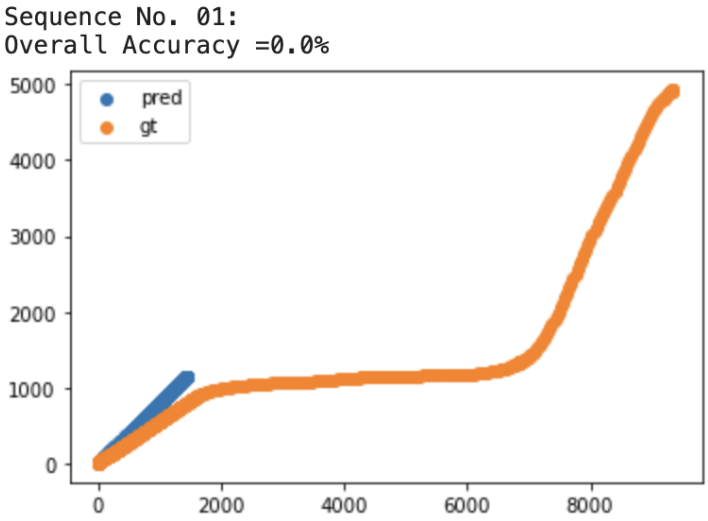

Figure 6.9 The predicted path and the ground truth of the sequence 01 with overall accuracy = 0

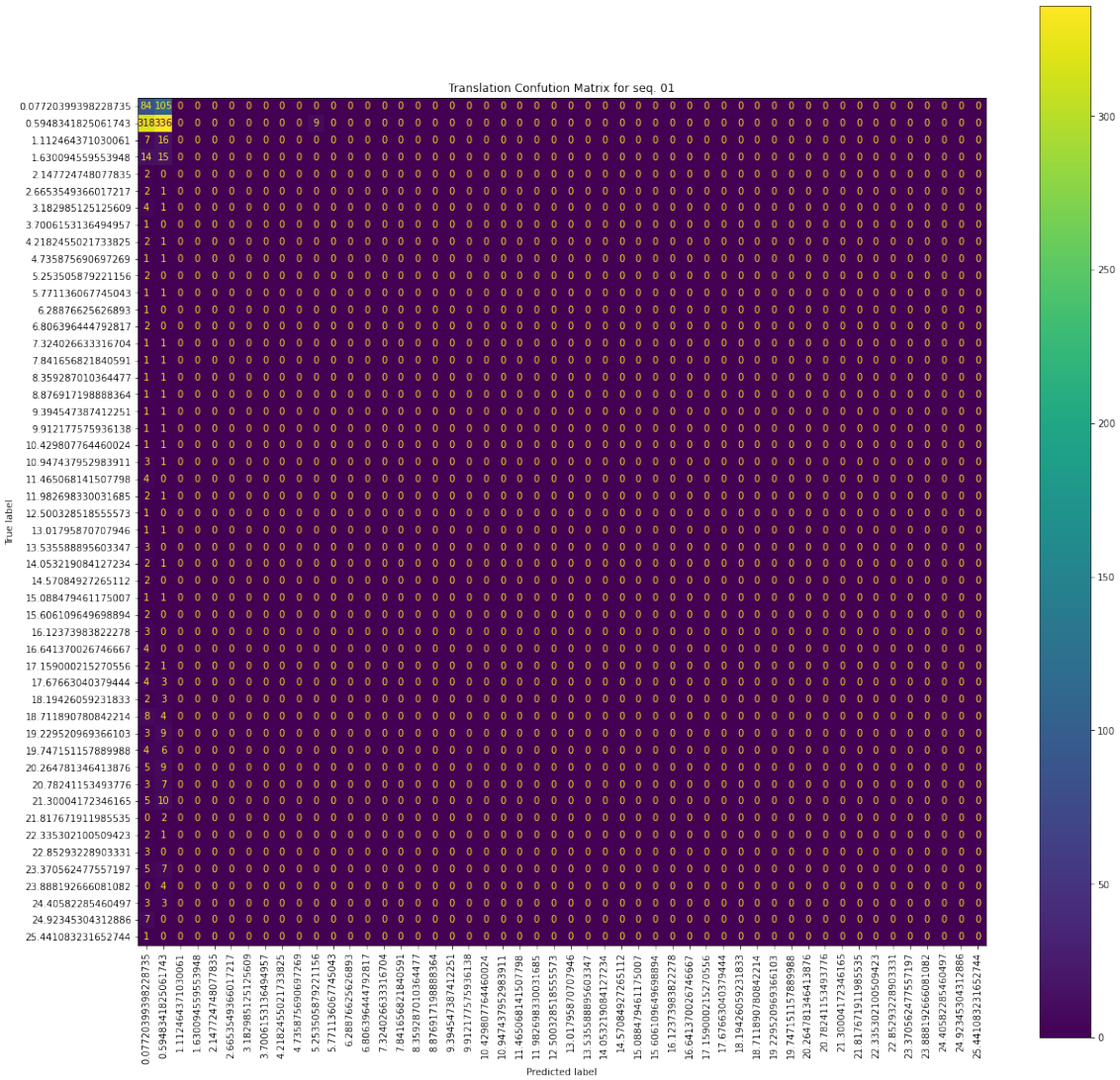

Figure 6.10 The Confusion Matrix for The Translation, seq. 01

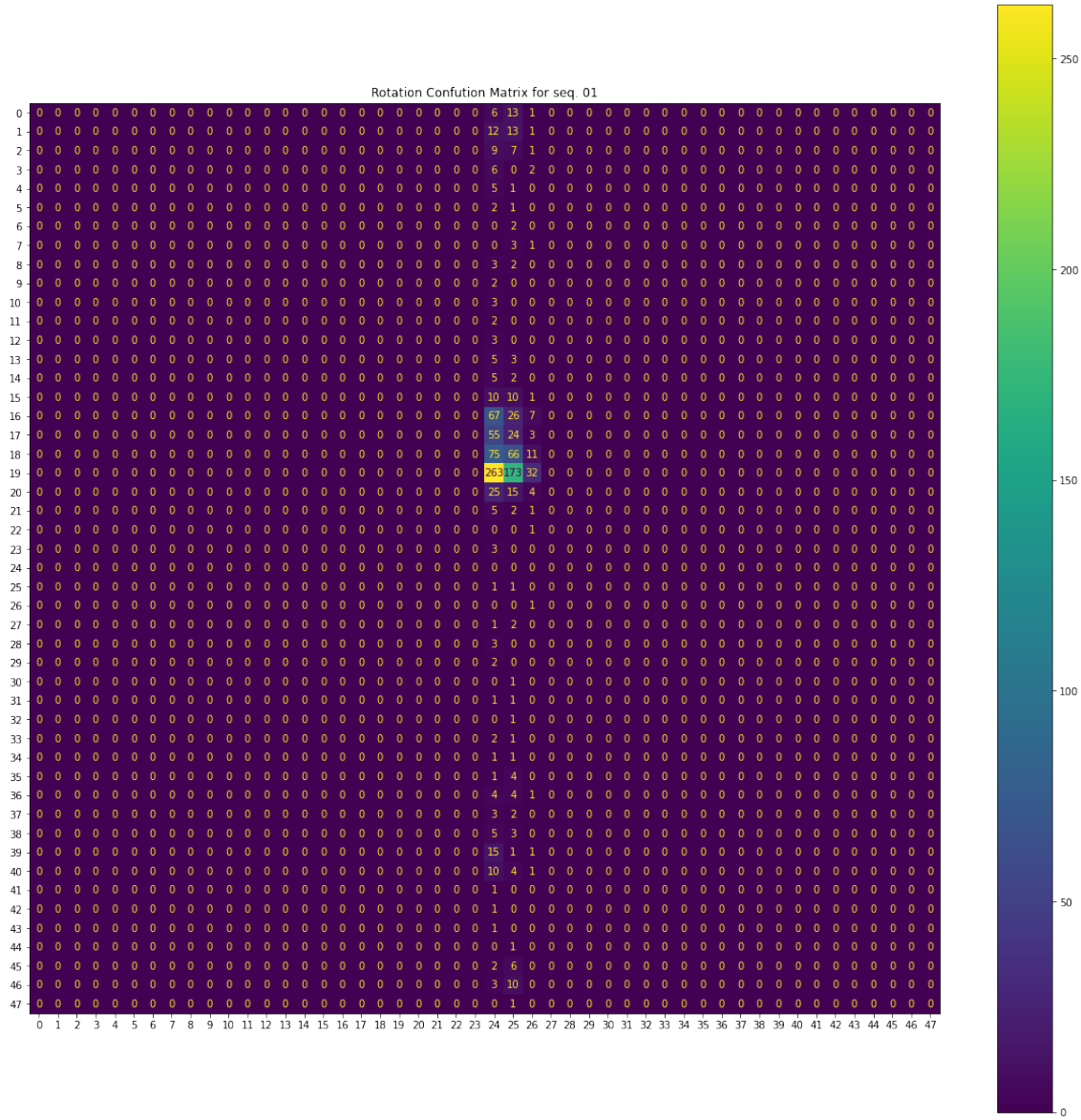

Figure 6.11 The Confusion Matrix for The Rotation, seq. 01

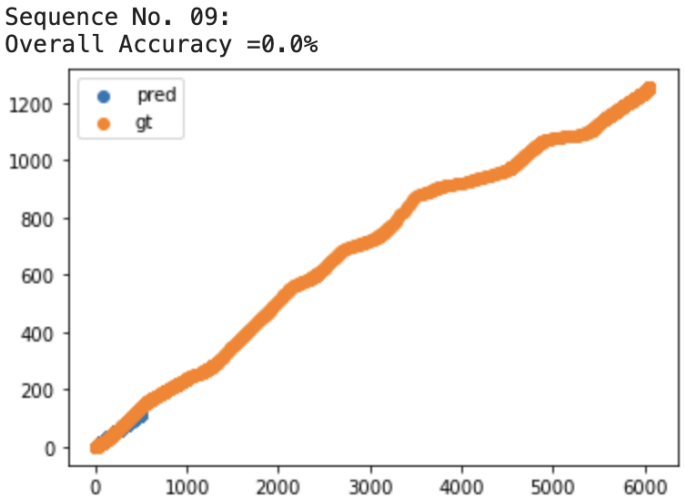

Figure 6.12 The predicted path and the ground truth of the sequence 09 with overall accuracy = 0

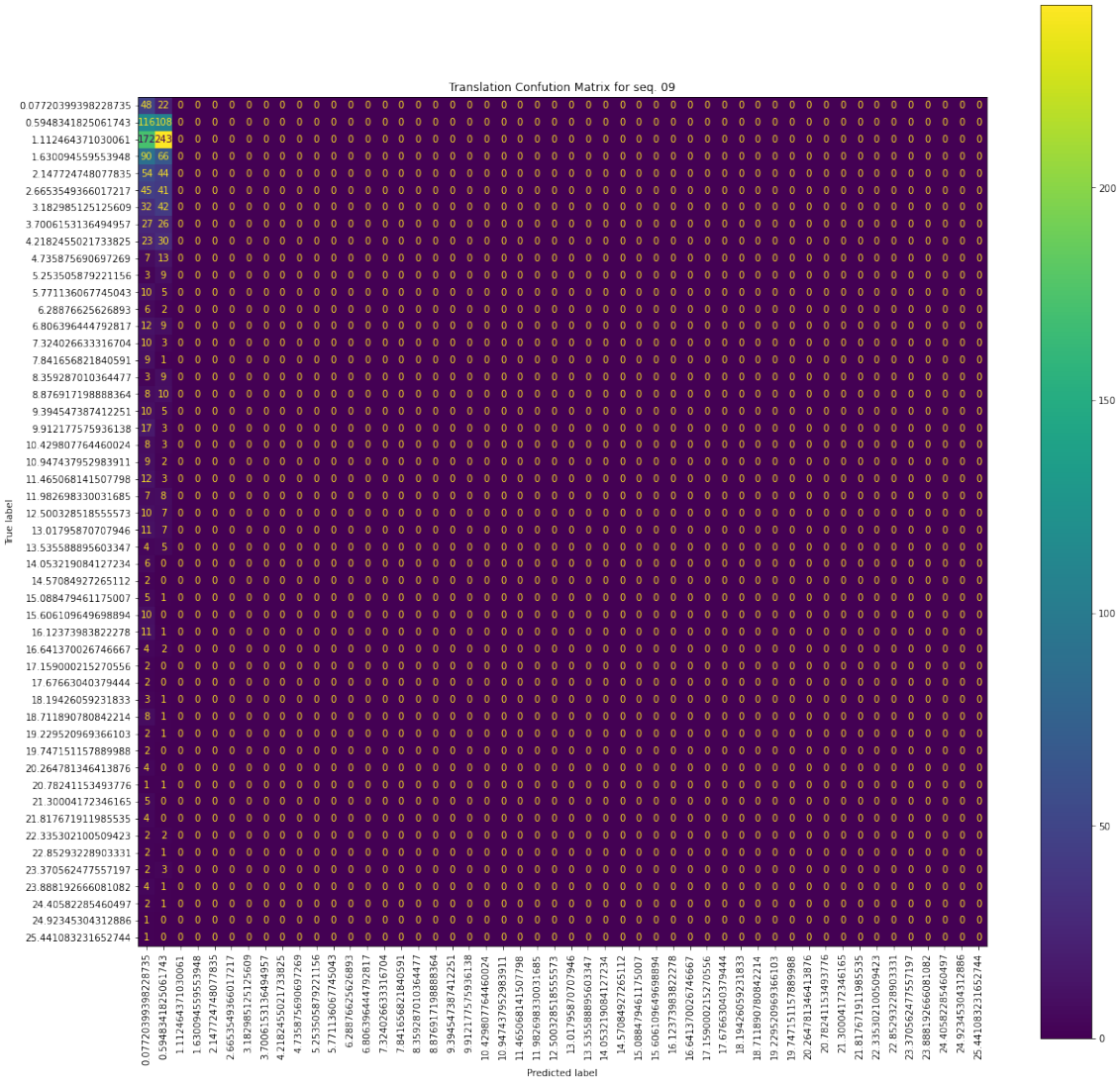

Figure 6.13 The Confusion Matrix for The Translation, seq. 09

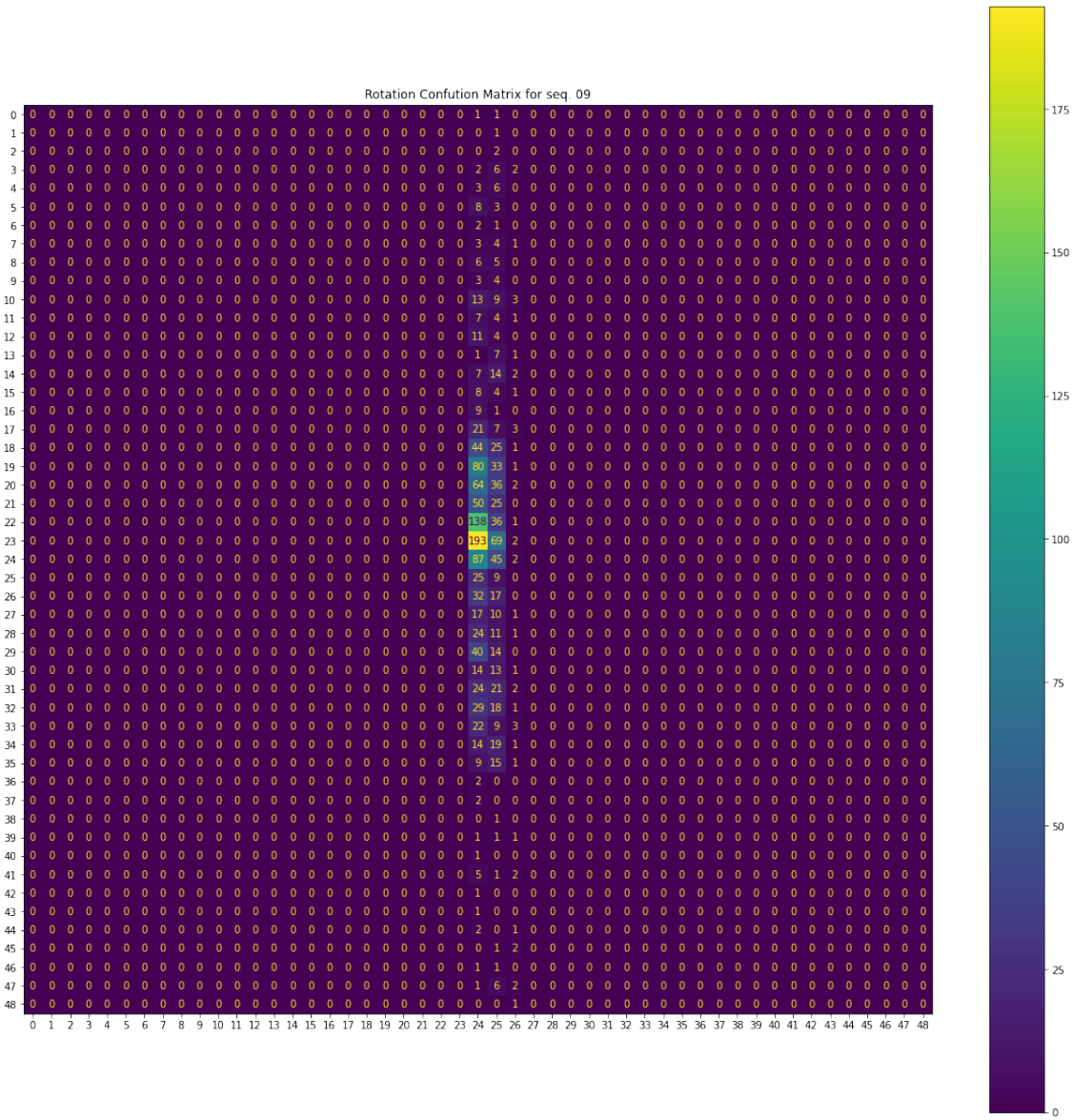

Figure 6.14 The Confusion Matrix for The Rotation, seq. 09

References

[1] J. Zhao, B. Liang, and Q. Chen, "The key technology toward the self-driving car," *International Journal of Intelligent Unmanned Systems*, 2018.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011, p89.

[3] A. Carballo *et al.*, "Libre: The multiple 3d lidar dataset," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1094–1101. DOI: 10.1109/IV47402.2020.9304681.

[4] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.