Deep Learning for Object Tracking in 3D Point Clouds

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Jaume Colom Hernandez

In partial fulfilment

of the requirements for the degree in

Telecommunications engineering

Advisor: Aljosa Osep

Supervisor: Josep R. Casas

Barcelona, June 2020

Abstract

Great progress has been achieved in computer vision tasks within image and video, however technological advances in LiDAR sensors have created a whole new area of computer vision research devoted to it. With applications in many industries, such as transportation, agriculture or healthcare.

This thesis studies object tracking in 3D point clouds. Pairs of point cloud observations are feed to a neural network to estimate pose and translation between the observations. Then this estimations, together with external features, are processed with Kalman Filter and RNN to extract spatial-temporal redundancies and improve the results.

The system has been tested in the KITTI dataset, with pre-segmented observations, on different types of objects and paths. The results show that the neural network estimated pose gives a very accurate tracking, but the best results are achieved when combining the estimated pose and translations with a recurrent neural network.

Resum

S’han aconseguit grans avenços en problemes de visió per ordenador en imatge i vide, tot i així els avenços tecnologics en sensors LiDAR han creat una nova área de recerca en visió per ordenador dedicada a ella. Amb aplicacions en varies industries, per exemple transport, agricultura o sanitat.

Aquesta tesis estudia tracking d’objectes en nuvols de punts 3D. Parelles d’observacions de nuvols de punts és processen amb una red neuronal per a estimar posició i translació entre les observacions. Després aquestes estimacions, juntament amb ‘features’ externes, són processades amb filtres de Kalman i RNN per a extraure redundancies espai-temporals i millorar els resultats.

El sistema s’ha testejat amb la base de dades KITTI, amb observacions pre-segmentades, en diferents tipus d’objectes i rutes. Els resultats mostren que l’estimació de posició amb la red neuronal dona molt bons resultats, pero els millors resultats s’aconsegueixen quan és combina les estimacions de posició amb reds neuronals recurrents.

Resumen

Se han conseguido grandes avances en problemas de visión por ordenador en imagen y video, aun así los avances tecnológicos en sensores LiDAR han creado una nueva área de investigación en visión por ordenador dedicada a ella. Con aplicaciones en varias industrias, por ejemplo transporte, agricultura o sanidad.

Esta tesis estudio tracking de objetos en nubes de puntos 3D. Parejas de observaciones de nubes de puntos se procesan con una red neuronal para estimar posición y traslación entra las observaciones. Después estas estimaciones, junto con 'features' externas, son procesadas con filtros Kalman y RNN para extraer redundancias espaciotemporales y mejorar resultados.

El sistema se ha testeado con la base de datos KITTI, con observaciones pre-segmentadas, en diferentes tipos de objetos y rutas. Los resultados muestran que la estimación de posición con la red neural ya da muy buenos resultados, pero los mejores resultados se consiguen cuando se combinan estimación de posición con redes neuronales recurrentes.

Acknowledgements

I want to thank my advisor Aljosa Osep, for his help and attitude during the development of the thesis. Also Prof. Laura Leal for allowing me to write the thesis in her lab, and helping with the initial paperwork. And finally thanks to Quirin Lohr for the technical support he provided.

From UPC I want to thank Josep R. Casas for supervising the project from Spain and helping me successfully face the bachelor thesis.

Finally thanks to my family and friends for their invaluable support and patience.

Revision history and approval record

Revision	Date	Purpose
0	15/06/2020	Document creation
1	29/06/2020	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Jaume Colom Hernandez	jaumecolomhernandez@gmail.com
Aljosa Osep	aljosa.osep@tum.de
Josep R. Casas	josep.ramon.casas@upc.edu

Written by:		Reviewed and approved by:	
Date	15/06/2020	Date	29/06/2020
Name	Jaume Colom	Name	Aljosa Osep
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

1.Introduction

Recently there has been an increase of interest in 3D perception, fuelled by the recent developments of LiDAR sensors [1]. Due to increased availability, reduction of price and increased performance, LiDAR sensors have been gaining popularity in many applications, being robotics and more specifically self-driving cars the star application.

This has driven a scientific movement to solve a myriad of new problems in the context of 3D computer vision, for self-driving cars it means object classification, object tracking, SLAM among others. Other relevant areas of research are object reconstruction [2], neural rendering [3], semantic segmentation [4], image generation [5] or pose estimation [6]. These technologies have a wide range of applications (retail, automotive, healthcare or agriculture) and are still a subject of research.

Deep learning has revolutionised machine learning and computer vision. In the last years we have seen great increases in the performance of traditional computer vision problems such as object recognition [7] or detection. The availability of large datasets [8] together with the computing power needed for training big enough models has brought deep learning at the forefront of research in computer science.

This project will start from the AlignNet [9] publication from Gross et al. , analyze it, and extend it to multiple time-step prediction. This publication proposes a novel approach to point cloud registration, instead of using a classical algorithm it uses a learned approach. It outperforms the classical, computationally expensive methods both in accuracy and computation time.

The current limitation of the AlignNet-3D solution is that it is constrained to predict object pose movement within two frames. To solve this problem an initial solution is proposed. Run AlignNet-3D with all consequent frames and simply add the predicted translation to predict the full translation. To refine this trajectory we try two approaches, Kalman Filter [10] and learned LSTMS [11].

The next iteration is to train a network on the 1024 length vector output of the CanonicalNet [9]. This network will be able to pick a lot more information from the implicit representation. Further improvements imply to have two latent representations one for pose and one for shape, and use both to create a better tracking with the by-product of having a shape representation in the network.

1.1.Statement of purpose

The purpose of this project is to extend to extend siamese tracker [9]. Instead of estimating relative motion between two point sets only, we should take the full history of surface observations into account. This can be posed as sequence-level learning using recurrent neural networks, e.g., Long-Short Term Memory Networks (LSTMs).

The project main goals are:

1.- Learn about deep learning applied to computer vision.

2.- Design and implement a system for object tracking in 3D point clouds.

1.2.Requirements and specifications

The project requires the following technical skills (non-exhaustive list), programming, basic geometry, machine learning and deep learning. Apart, the project requires specific knowledge about the base publication, AlignNet [9], as it will extend the current project with new features.

The project will produce path estimations/predictions based on previous observations of the dataset. Therefore it needs to be able to process the input data, generate the output (paths) and evaluate the data. It needs to do so in an automated fashion as a large number of experiments will be run during the experimentation phase of the project, and the easier it is to run them the faster and more effective the project will be.

1.3.Methods and procedures

The Dynamic Vision and Learning chair (https://dvl.in.tum.de/) at TUM is a newly created research group within the Computer Vision group. It is leaded by Professor Laura Leal (https://dvl.in.tum.de/team/). It works on solving computer vision problems with a deep learning approach; The current areas of research are Multi Object Tracking [12] [13], GAN video generation [14] and minor focus on other areas of computer vision. Previous work has been done at DVL in the context of object tracking, both in 2D and 3D problems, proposing novel solutions to the problem. Also they organize the MOTChallenge [15] in the CVPR conference.

The project is a continuation of the Align-3D [5] publication from Gross et al. It proposes to extend the Siamese tracker to use the full trajectory to estimate motion. It is performed as an independent thesis within the laboratory (DVL), this means that the student works independently in the project while the advisor supervises the work. The main ideas were proposed by the supervisor, in the chair they have a document with different project ideas and I decided to work in one of them.

1.4.Work plan

WP Name: Intensive research	WP ref: 1.1
Major constituent: Research	-
Short description: Read and learn about the current trends relevant to this project. In this case: Multi object tracking, deep learning on point clouds (point-net), point cloud registration, object detection/recognition.	Planned start date: 07/02/2020 Planned end date: 15/02/2020

WP Name: First steps with current model (Align-3D)	WP ref: 1.2
Major constituent: Learning	-
Short description: Read the code, train, reproduce results, adapt to current setup and play with the existing code.	Planned start date: 07/02/2020 Planned end date: 15/02/2020

WP Name: Extensive research	WP ref: 1.3
Major constituent: Research	-
Short description: Read and learn about the current trends relevant (or not) to this project. In this second stage it is both theoretical and practical areas. The plan is that this work will ease and sharpen my skills and it will be reflected in the on-going project.	Planned start date: 10/02/2020 Planned end date: 01/06/2020

WP Name: Kalman Filter with Align3D predictions	WP ref: 2.1
Major constituent: Development	-
Short description: The current model gets the translation and rotation of the ob-	Planned start date: 15/02/2020

ject in two different timepoints. Using this data with many consequent frames we can get the object path. Analyze the results and improve them using Kalman Filter.	Planned end date: 24/02/2020

WP Name: Kalman Filter with estimated positions	WP ref: 2.2
Major constituent: Development	-
Short description: Extending the work of the previous WP the results were not as good as expected. Design, implement and analyze the results of using estimated position (mean, median and NN) for predicting the position.	Planned start date: 24/02/2020 Planned end date: 06/03/2020

WP Name: Kalman Filter further experiments and tuning	WP ref: 2.3
Major constituent: Development	-
Short description: Once having both estimated position and translation there were a lot of option to analyze. Predict positions or translations. Take into account rotation. Extended Kalman Filter.	Planned start date: 09/03/2020 Planned end date: 03/04/2020

WP Name: LSTMs for object tracking	WP ref: 3.1
Major constituent: Development	-
Short description: Use LSTMs to estimate the position of the objects, it acts as a replacement to the Kalman Filter. Design, implement and evaluate the results.	Planned start date: 06/04/2020 Planned end date: 01/05/2020

WP Name: LSTMs with CanonicalNet outputs	WP ref: 3.2
Major constituent: Development	-
Short description: Use LSTMs to estimate the position of the objects, this time based on the 1024 length vector generated by the CanonicalNet. Much larger network, have to see if will have enough data with KITTI dataset.	Planned start date: 01/05/2020 Planned end date: 01/06/2020

WP Name: Further ideas (optional)	WP ref: 3.3
Major constituent: Development	-
Short description: Extend the current model to support full trajectory when doing inference. Current ideas are: add shape information using a latent representation of the shape.	Planned start date: 22/05/2020 Planned end date: 01/06/2020

WP Name: Write thesis	WP ref: 4
Major constituent: Writing	-
Short description: Write the thesis document. Will dedicate full time during the last month to this task.	Planned start date: 01/06/2020 Planned end date: 20/06/2020

1.5.Deviations from initial plan

In the original plan I introduced a short work package called "Object Tracking with current model" that ended mid-march but I continued to work in that area for an extra month.

During the first month I analyzed the Kalman Filter on the outputs of the Align3D, and then I analyzed the performance of a LSTMs based network on the same outputs.

A second reason for the delay was the development of a lot of infrastructure to support the experiments. The first iteration demanded a lot of manual work for each experiment, creating classes, data container, visualizations, etc. Because of this original infrastructure I was quite slow to perform experiments and not progressing as much as I wanted so I developed a novel architecture to hold all the data and help me access the results.

2.Technology review

In this chapter we will review the basic concepts required for understanding this thesis and analyse three scientific publications that lay the groundwork for the development of this project.

2.1.Fundamentals

2.1.1.Machine Learning

Machine learning is an area from artificial intelligence that is dedicated to the design, analysis and development of algorithms and techniques that allow machines to learn from data. It creates programs capable of generalizing behaviours based on patterns or classifications. It is related to the field of statistics, but it also coincides with model building methods, or statistical learning.

Some areas where this type of learning has been applied are applications dedicated to natural language processing, search algorithms, medical diagnosis, bioinformatics, fraud detection and classification.

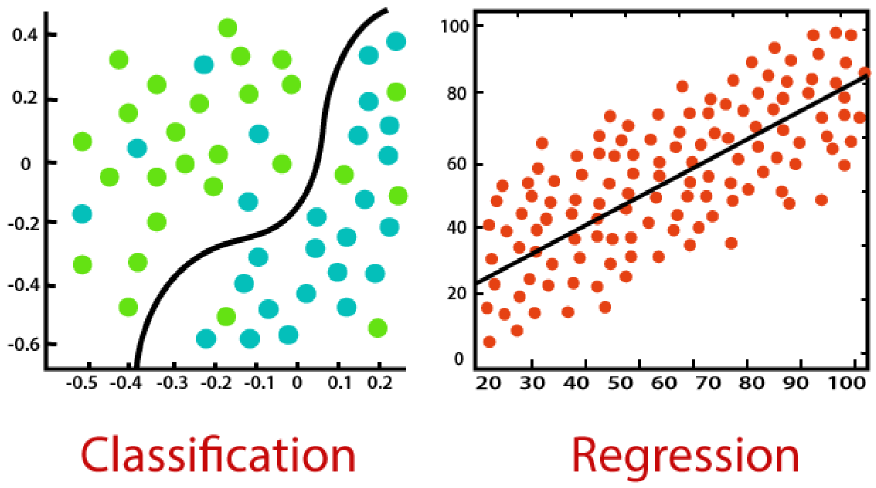


Figure 1: Classification/Regression [16]

Any system that is considered intelligent must have the ability to learn, that is to automatically improve with experience. The programs used are learning systems capable of acquiring high-level knowledge and problem-solving strategies using examples, analogous to how the human mind would do it.

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

- Supervised machine learning: The program is “trained” on a pre-defined set of “training examples”, which then facilitate its ability to reach an accurate conclusion when given new data.
- Unsupervised machine learning: The program is given a bunch of data and must find patterns and relationships therein.

2.1.2.Deep Learning

Deep learning is a subset of machine learning. It attempts to imitate how the human brain can process light and sound stimuli into vision and hearing. A deep learning architecture is inspired by biological neural networks and consists of multiple layers in an artificial neural network made up of hardware and GPUs.

Deep learning uses a cascade of nonlinear processing unit layers in order to extract or transform features (or representations) of the data. The output of one layer serves as the input of the successive layer. In deep learning, algorithms can be either supervised and serve to classify data, or unsupervised and perform pattern analysis.

Among the machine learning algorithms that are currently being used and developed, deep learning absorbs the most data and has been able to beat humans in some cognitive tasks. Because of these attributes, deep learning has become an approach with significant potential in the artificial intelligence space

Computer vision and speech recognition have both realized significant advances from deep learning approaches.

Deep learning algorithms use neural networks to find associations between a set of inputs and outputs. See figure 1 for the basic structure of a neural network.

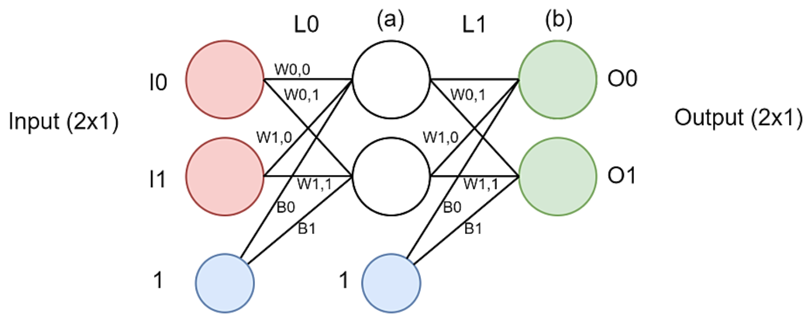


Figure 2: Neural network graphic representation [17]

A neural network is composed of input, hidden, and output layers — all of which are composed of "nodes". Input layers take in a numerical representation of data (e.g. images with pixel specs), output layers output predictions, while hidden layers are correlated with most of the computation.

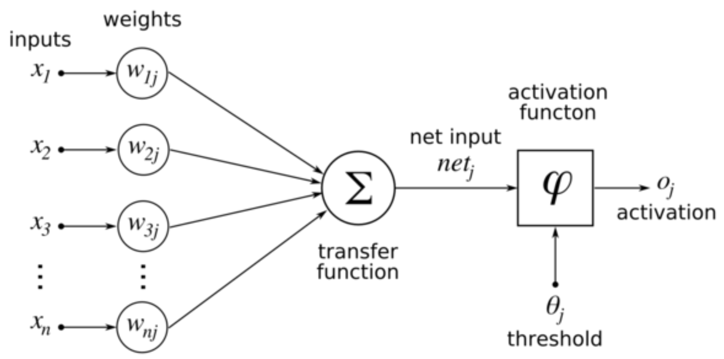


Figure 6: Basic Perceptron/Linear Classifier [18]

Information is passed between network layers through the function shown above. The major points to keep note of here are the tuneable weight and bias parameters — represented by w and b respectively in the function above. These are essential to the actual "learning" process of a deep learning algorithm.

After the neural network passes its inputs all the way to its outputs, the network evaluates how good its prediction was (relative to the expected output) using a loss function. The goal of the network is ultimately to minimize this loss by adjusting the weights and biases of the network. Using back propagation (paper) through gradient descent, the network back-tracks through all its layers to update the weights and biases of every node in the opposite direction of the loss function – In other words, every iteration of back propagation should result in a smaller loss function than before.

Without going into the proof, the continuous updated of the weights and biases of the network ultimately turns it into a precise function approximate – one that models the relationship between inputs and expected outputs.

2.1.2.1.Recurrent neural networks

Recurrent neural networks add a twist to basic neural networks. A vanilla neural network takes a fixed size vector as input which limits its usage in situations that involve a series type input with no predetermined size.

Recurrent Neural Network remembers the past and its decisions are influenced by what it has learnt from the past. Note: Basic feed forward networks "remember" things too, but they remember things they learnt during training. For example, an image classifier learns what a "1" looks like during training and then uses that knowledge to classify things in production.

While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It's part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

In summary, in a vanilla neural network, a fixed size input vector is transformed into a fixed size output vector. Such a network becomes "recurrent" when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. There is no pre-set limitation to the size of the vector. And, in addition to generating the output which is a function of the input and hidden state, we update the hidden sate itself based on the input and use it in processing the next input.

LSTMS

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) [11], and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

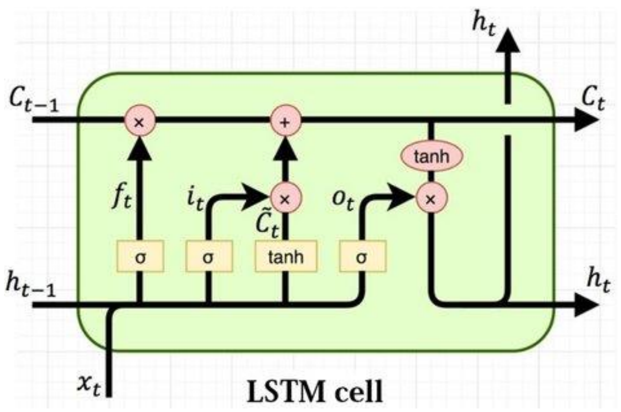


Figure 3: LSTMS cell diagram [19]

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn.

A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, of the flow of information inside the LSTM unit: an input gate, an output gate and a forget gate.

Intuitively, the cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.

<u>2.1.3.Kalman Filter</u>

The Kalman Filter is a powerful mathematical tool that plays an important role when real-world measurements are included in the system you are working with. It was invented by Rudolph Emil Kalman in the late 1950s [20], with the aim of filtering and predicting linear systems.

Basically it is a set of mathematical equations that implement an optimal estimator of the predictor-corrector type. It processes all available measurements, regardless of their precision, to estimate the current value of the variables of interest. This is possible thanks to:

a) Knowledge of the system and dynamic measurement devices,

b) The statistical description of system noises, measurement errors and uncertainties in dynamic models,

c) And any available information about the variables of interest.

```
     KF Algorithm ($\mathbf{x_{t-1}, P_{t-1}, u_t, z_t}$)
     Prediction stage:
1-   Forward state projection.
     $\mathbf{x_t} = \mathbf{A_t x_{t-1}} + \mathbf{B_t u_t}$
2-   Covariance error forward projection.
     $\mathbf{\overline{P}_t} = \mathbf{A_t P_{t-1} A_t^T} + \mathbf{R_t}$
     Update stage:
3-   Compute Kalman Gain.
     $\mathbf{K_t} = \mathbf{\overline{P}_t C_t^T (A_t \overline{P}_t A_t^T + Q_t)^{-1}}$
4-   Update state with measurement ($\mathbf{z_t}$)
     $\mathbf{x_t} = \mathbf{\overline{x}_t} + \boldsymbol{K_t(z_t - C_t \overline{x_t})}$
5-   Update error covariance.
     $\mathbf{P_t} = \boldsymbol{(I - K_t C_t)\overline{P_t}}$
6-   Return $\mathbf{x_t, P_t}$
```

The algorithm

The KF represents a belief or confidence in the state at the time that is given by the mean, $\mathbf{x_t}$ and the covariance, $\mathbf{P_t}$. The input that the KF receives is the belief in **t–1**, represented by $\mathbf{x_{t-1}}$ and $\mathbf{P_{t-1}}$.

To update this belief, the KF also requires the control signals ($\mathbf{u_t}$) and the observations of the environment provided by the sensors ($\mathbf{z_t}$). The output of the KF would be the belief in the instant of time, represented by $\mathbf{x_t}$ and $\mathbf{P_t}$.

Prediction stage, in which belief in the current moment is projected, through odometry. It is just the addition to the robot's displacement and rotation of the instant $t - 1$ of the displacement and the rotation that have occurred since the instant $t - 1$ to t.

Updating stage, in which the re-observed characteristics are considered. Thanks to the estimation of the position provided by the Prediction stage, it is possible to estimate where the feature should be, allowing to correct the position of the robot.

The matrices that appear in the KF algorithm are defined below:

$\mathbf{A_t}$: Matrix that relates the state at time $t - 1$ to the state at time t, in the absence of control signals.

$\mathbf{B_t}$: Matrix that relates the optional control signals to the current state. Rt: Matrix nxn that represents the covariance of the process noise.

$\mathbf{C_t}$: Matrix mxn that relates the current state with the observations of the environment.

$\mathbf{Q_t}$: Matrix nxn that represents the covariance of the noise of the observations.

$\mathbf{K_t}$: Matrix nxm that represents the Kalman Gain. The Kalman gain indicates confidence in the observed characteristics, using their uncertainty together with a measure of the quality of the data provided by the laser and by odometry. If the odometry is good and the data supplied by the laser are not so good, the odometry will have more weight than the observations, giving a low value of the Kalman gain. If the opposite situation were to occur, the Kalman gain would be high, and the observations supplied by the laser would have more weight.

2.2.Relevant research

2.2.1. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation [21]

PointNet [21] is a seminal paper in 3D perception, applying deep learning to point clouds for object classification and part/scene semantic segmentation. Its basis structure has been used for many different applications in 3D perception, extending it from classification/segmentation to many other domains.

Input data

PointNet takes raw point cloud data as input, which is typically collected from either a LiDAR or radar sensor. Unlike 2D pixel arrays (images) or 3D voxel arrays, point clouds have an unstructured representation in that the data is simply a collection (more specifically, a set) of the points captured during a LiDAR or radar sensor scan. In order to leverage existing techniques built around (2D and 3D) convolutions [22] [23] [24], many researchers and practitioners often discretize a point cloud by taking multi-view projections onto 2D space or quantizing it to 3D voxels. Given that the original data is manipulated, this approach can have negative impacts.

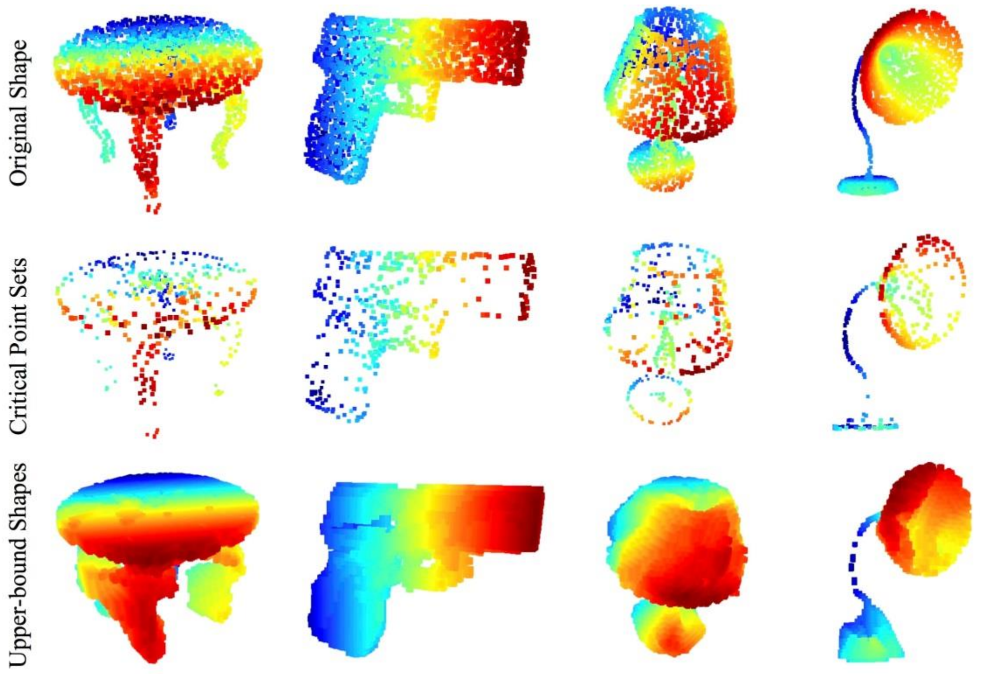


Figure 4: PointNet critical points and shape upper-bound [21]

Network architecture

Given that PointNet consumes raw point cloud data, it was necessary to develop an architecture that conformed to the unique properties of point sets. Among these, the authors emphasize:

- Permutation (Order) Invariance: given the unstructured nature of point cloud data, a scan made up of N points has N! permutations. The subsequent data processing must be invariant to the different representations.
- Transformation Invariance: classification and segmentation outputs should be unchanged if the object undergoes certain transformations, including rotation and translation.
- Point Interactions: the interaction between neighbouring points often carries useful information (i.e., a single point should not be treated in isolation). Whereas classification need only make use of global features, segmentation must be able to leverage local point features along with global point features.

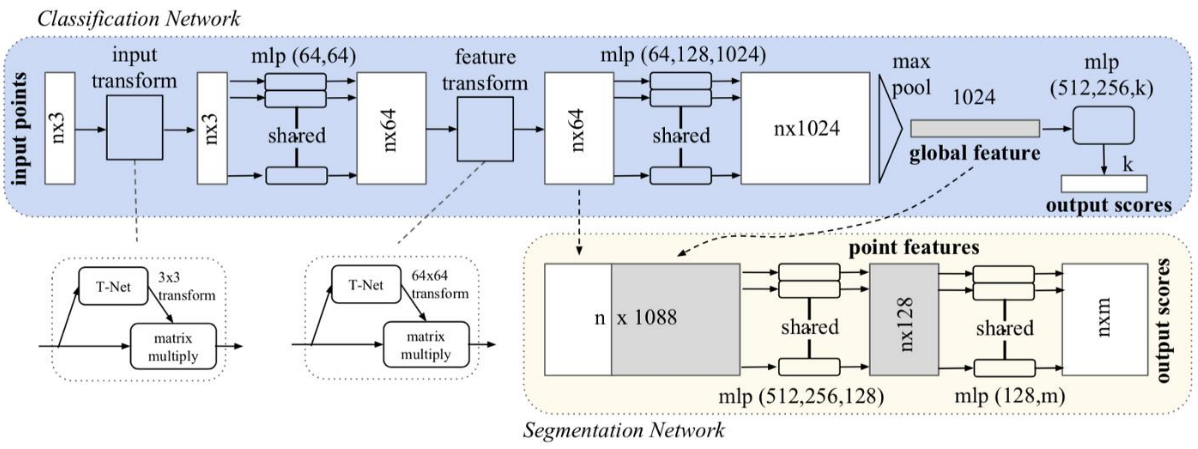


Figure 5: PointNet architecture [21]

Ignoring the application specific part of the network, what point net does is transforming the input data (nx3) to a global feature vector of length 1024. The core network uses a shared multi-layer perceptron (MLP) to map each of the n points from three dimensions (x, y, z) to 64 dimensions. It's important to note that a single multi-layer perceptron is shared for each of the n points (i.e., mapping is identical and independent on the n points). This procedure is repeated to map the n points from 64 dimensions to 1024 dimensions. With the points in a higher-dimensional embedding space, max pooling is used to create a global feature vector in $\mathbb{R}^{1024}$.

Permutation invariance

As mentioned, point clouds are inherently unstructured data and are represented as numerical sets. Specifically, given N data points, there are N! permutations. In order to make PointNet invariant to input permutations, the authors turned to symmetric functions, those whose value given n arguments is the same regardless of the order of the arguments [2]. For binary operators, this is also known as the commutative property. Common examples include:

$$\mathrm{sum}(a, b) = \mathrm{sum}(b, a)$$
$$\mathrm{average}(a, b) = \mathrm{average}(b, a)$$
$$\max(a, b) = \max(b, a)$$

Specifically, the authors make use of the symmetric function once the n input points are mapped to higher-dimensional space, as shown below. The result is a global feature vector that aims to capture an aggregate signature of the n input points. Naturally, the expressiveness of the global feature vector is tied to the dimensionality of it (and thus the dimensionality of the points that are input to the symmetric function). The global feature vector is used directly for classification and is used alongside local point features for segmentation.

Transformation invariance

The classification (and segmentation) of an object should be invariant to certain geometric transformations (e.g., rotation). The "input transform" and "feature transform" are modular sub-networks that seek to provide pose normalization for a given input.

The T-Net is a regression network that is tasked with predicting an input-dependent 3-by-3 transformation matrix that is then matrix multiplied with the n-by-3 input.

The operations comprising the T-Net are motivated by the higher-level architecture of PointNet. MLPs (or fully-connected layers) are used to map the input points independently and identically to a higher-dimensional space; max pooling is used to encode a global feature vector whose dimensionality is then reduced to $\mathbb{R}^{256}$ with FC layers. The input-dependent features at the final FC layer are then combined with globally trainable weights and biases, resulting in a 3-by-3 transformation matrix.

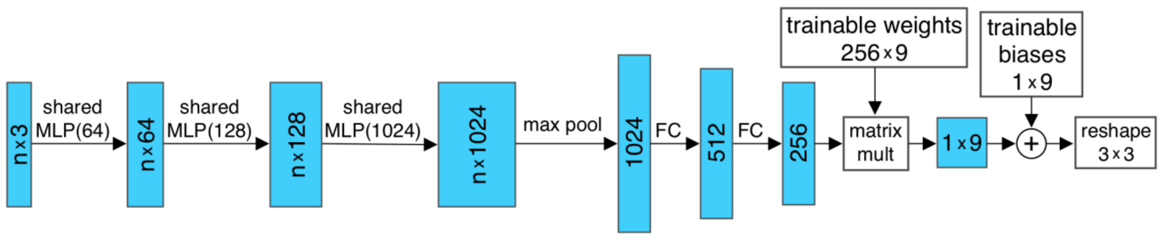


Figure 6: Structure of the Spatial Transformer

The concept of pose normalization is extended to the 64-dimensional embedding space ("feature transform" in Fig. 2). The corresponding T-Net is nearly identical to that of Fig. 8 except for the dimensionality of the trainable weights and biases, which become 256-by-

4096 and 4096, respectively resulting in a 64-by-64 transformation matrix. The increased number of trainable parameters leads to the potential for over fitting and instability during training, so a regularization term is added to the loss function. The regularization term is shown below and encourages the resulting 64-by-64 transformation matrix (represented as A below) to approximate an orthogonal transformation.

$$\mathcal{L}_{reg} = \|I - AA^T\|^2$$

Figure 7: Regularization used in loss function

2.2.2. AlignNet-3D for Fast Point Cloud Registration of Partially Observed Objects [9]

Align3D proposes a novel approach for computing point cloud registration. It is based on learned approach instead of computationally expensive classical algorithms.

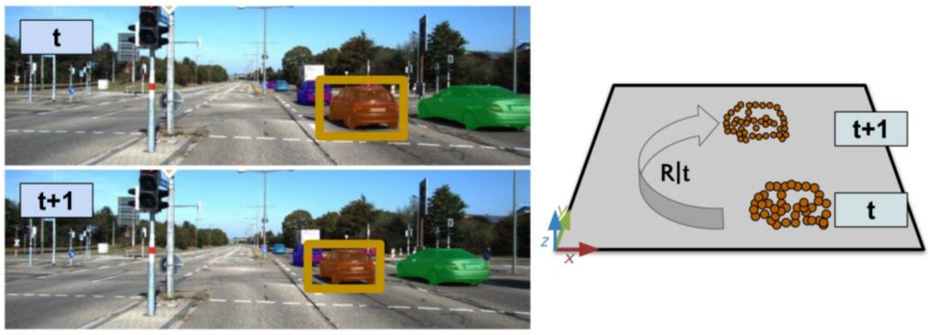


Figure 8: High level overview of Align3D

Given two LiDAR point segments that measure object shape at different times, the network estimates the **relative motion between scans**. The novelty is that it uses a learning approach opposed to purely geometric methods. This gives good performance in adverse conditions such as large temporal gaps.

System architecture

Input data to the network are two 3D point clouds that represent surface measurements of an object, captured at two different timesteps. The point clouds are sampled to ensure the same size. Both point clouds are input to branches of (the same) CanonicalNet, which transform the point clouds to a canonical pose and compute a fixed-dimensional feature vector for each point cloud (see PointNet 3.1.3). To obtain a refined alignment estimate, it concatenates the embeddings of both point clouds and uses a multi-layer perceptron (MLP) to produce the final transform.

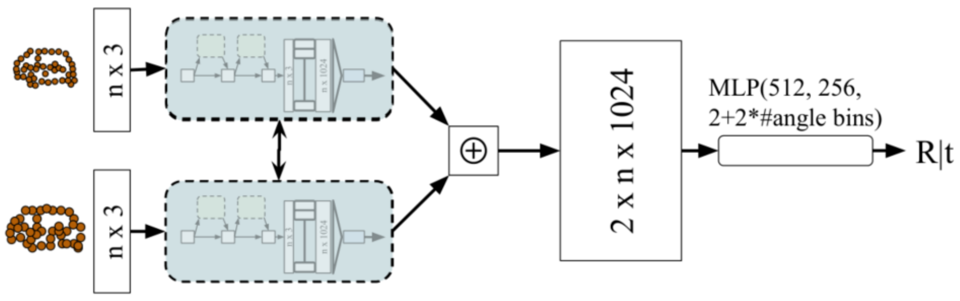


Figure 9: Architecture align3d

To process the sparse point cloud data (within a branch of the siamese network), it uses a PointNet architecture. First normalizes the input point cloud by moving its centroid to the origin. Using T-CoarseNet (Point-Net (64, 128, 256), followed by $MLP_{0.7}$(512, 256)) it predicts an amodal object center and bring the object closer to a canonical pose by moving the predicted center to the origin. After the coarse center estimate, it uses T-FineNet (Point-Net (64, 128, 512), followed by $MLP_{0.7}$(512, 256)) to refine the object center and additionally predict a canonical orientation. To estimate orientation instead of directly regressing the angle to the canonical orientation, it predicts a classification into one of 50 equidistant angle bins between 0 and 2π. Additionally, we predict an angle residual for every angle bin, so that the final angle prediction is computed as $\alpha = i \cdot 2\pi/(\#bins) + res_i$, where i is the predicted angle bin and res_i is the respective predicted residual. The output layer size of an MLP predicting a translation and an angle is therefore $2 + 2 \cdot \#bins$. We re-normalize the point cloud by moving the new amodal center to the origin, followed by a rotation by −α. The final point cloud embedding is predicted by a PointNet(64, 128, 1024) network.

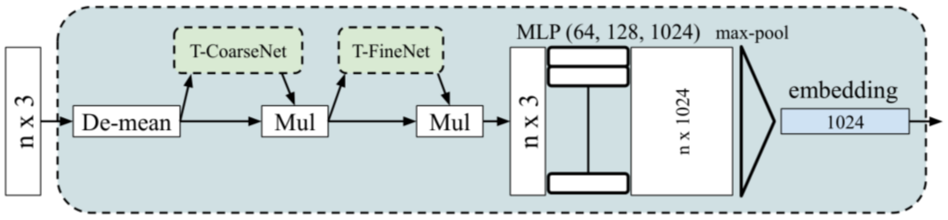


Figure 10: CanonicalNet architecture

Finally, it concatenates the point cloud embeddings computed by the siamese branches (see Fig. 3a). It inputs the combined feature vector of size 2048 to a final $MLP_{0.7}$(512, 256), which predicts refined translation and angle to precisely align the two point clouds in their canonical pose estimates.

Loss function

All the stages of the pipeline are fully supervised. In stage 1 it predicts an amodal center with our first transformer network, T-CoarseNet. As target center it uses the center of the 3D bounding box. In stage 2 it predicts an amodal center and the deviation from a canonical orientation. The target rotation angle is the annotated 3D bounding box orientation. In stage 3 we predict the remaining translation and rotation needed to align the point clouds from the remaining translation and rotation needed to align the point clouds from the concatenated embeddings.

For translation the Huber loss function [25] is used with *δ=1* except for the last stage where *δ=2*. The Huber loss function is defined as $1/2 * x$^2 for $|x| \leq \delta$ and $\delta|x|$- $1/2\ \delta$^2 otherwise. The angle loss if formed by the cross entropy loss for the angle bin classification and a Huber loss for the residual corresponding to the ground truth angle bin. Residuals are predicted normalized within [-1,1], corresponding to angles [−β/2, β/2] with the angle bin size beta = 2pi/#bins.

2.2.3.A Baseline for 3D multi-object tracking [26]

3D Multi-object tracking is an essential component technology for many vision (real-time or not) applications such as autonomous driving, robot collision prediction or video face alignment. In the recent years there has been a great progress on MOT.

Although the accuracy overtime has been significantly improved, it has come at the cost of increasing system complexity and computational cost. This paper develops an accurate, simple and real-time 3D MOT system. It combines the minimal components for 3D MOT and works extremely well, getting state of the art performance on 3D MOT.

It uses an off-the-shelf 3D object detector to obtain oriented 3D bounding boxes from LiDAR point cloud. Then a combination of Kalman Filter and Hungarian algorithm is used for state estimation and data association. Note that this work extends the state space of the Kalman Filter to full domain, including 3D location, size, velocity and orientation of the objects.

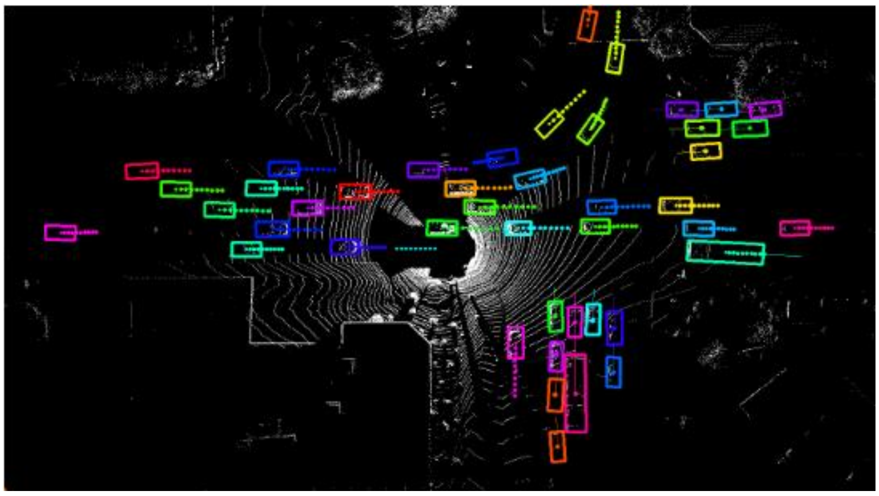


Figure 11: MOT in LiDAR captures (source: [27])

System architecture

The paper presents an online method, the means that at every frame we require only the detection at the current frame and associated trajectories from the previous frame. The system pipeline is illustrated in the next figure. It is composed of (1) 3D detection module to provide the bounding boxes from the LiDAR point cloud; (2) 3D Kalman Filter predicts the object state to the current frame; (3) data association module matches the detection with predicted trajectories; (4) 3D Kalman Filter updates the object state based on the measurement; (5) birth and death memory controls the newly appeared and disappeared trajectories.

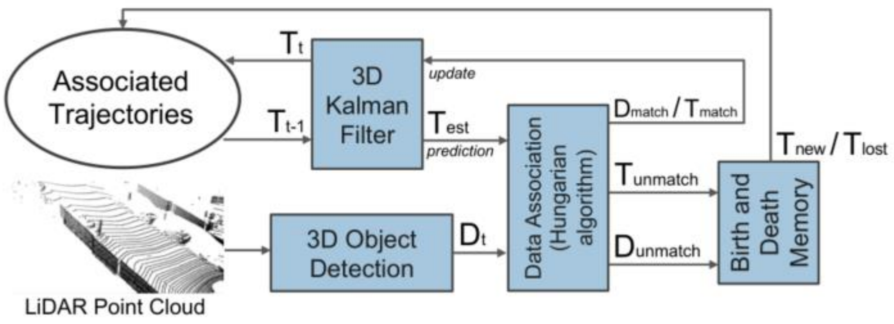


Figure 12: System architecture

3D Object Detection

This publication takes advantage of high quality detection from many successful detectors. It experiments with state-of-the-art 3D detectors on the KITTI dataset. It directly adopts their pretrained models on the training set of the KITTI 3D object detection benchmark. At frame t the output of the 3D detection module is a set of detections $D_t = \{D_t^1, D_t^2, \dots, D_t^n\}$ (n can vary from frame to frame). Each detection is D_t^i is represented as a tuple $(x, y, z, l, w, h, \theta)$.

3D Kalman Filter: State Prediction

To predict the object state in the next frame, we approximate the inter-frame displacement of object using the constant velocity model , independent of camera ego-motion. The state of object trajectory is represented as a 10-dimensional vector $T = (x, y, z, \theta, l, w, h, v_x, v_y, v_z)$ where v_x, v_y, v_z represent the velocity in 3D space. v_θ is not included as it degrades the performance of the whole system.

At every frame all associated trajectories from previous frame $T_{t-1} = \{T_{t-1}^1, T_{t-1}^2, \dots, T_{t-1}^m\}$ are propagated to frame t based on the constant velocity model:

$$x_{est} = x + v_x \quad y_{est} = y + v_y \quad z_{est} = z + v_z$$

As a result for every trajectory the predicted state after propagations to frame t is $T_{est} = (x_{est}, y_{est}, z_{est}, \theta, l, w, h, v_x, v_y, v_z)$ in T_{est}, which will be fed to the data association module.

Data Association

To match the detections D_t with predicted trajectories T_{est}, it applies the Hungarian algorithm. The affinity matrix is computed using the 3D IoU (intersection over union) metric between every pair of detection D_i and T_{est}^j. Then, the bipartite graph matching problem can be solved with the Hungarian algorithm. In addition trajectories are rejected when 3D IoU is less than IoU_{min}.

3D Kalman Filter: State Update

The updated trajectories are computed following the Bayes rule, they are the weighted average between the state space of T_{match}^k (estimated) and D_{match}^k (measurement). The weights are determined by the uncertainty of both the matched trajectory and detection. See Kalman Filter (section 2.1.3).

In addition, it is observed that the naïve weighted average does not work well for orientation. For matched object k, the orientation of its detection D_{match}^k can be nearly opposite to T_{match}^k (differ by π). This is not possible in real motion so the orientation of D_{match}^k or T_{match}^k must be wrong (most possibly D_{match}^k), when computing the IoU this mismatch will lead to low 3D IoU with ground truth. The paper proposes a simple solution to this problem. When the difference of orientation is frater than π/2, add π to the orientation of D_{match}^k so that it is consistent with T_{match}^k.

Birth and Death Memory

It is necessary to manage the birth and death trajectories as new objects might appear and existing disappear. First, it considers all unmatched detection $D_{unmatch}$ as potential objects entering the image. To avoid creating false positives a new trajectory won't be created until t_i appears for the next F_{min} frames. Once the new trajectory is successfully created, we initialize the state of the trajectory T_{new}^p same as its most recent measurement $D_{unmatch}^p$ with velocities set to zero.

Second, it considers all unmatched trajectories $T_{unmatch}$ as potential objects leaving the image. To avoid deleting true positive trajectories which have missing detection at certain frames, we keep tracking each unmatched trajectory $T_{unmatch}^q$ for Age_{max} frames before deleting it from the associated trajectories. Ideally, true positive trajectories with missing detection can be maintained and interpolated by our 3D MOT system, and only the trajectories that leave the image are deleted.

3.Methodology / project development:

3.1.System definition

3.1.1.Full pipeline

The system uses a sequence of pre-segmented point clouds as input data, this means that each observation only contains one object. They are sequentially processed with align-3D network to compute pose and translation estimations, then these estimations together with external features are feed into Kalman Filter or LSTMs to compute the refined routes.

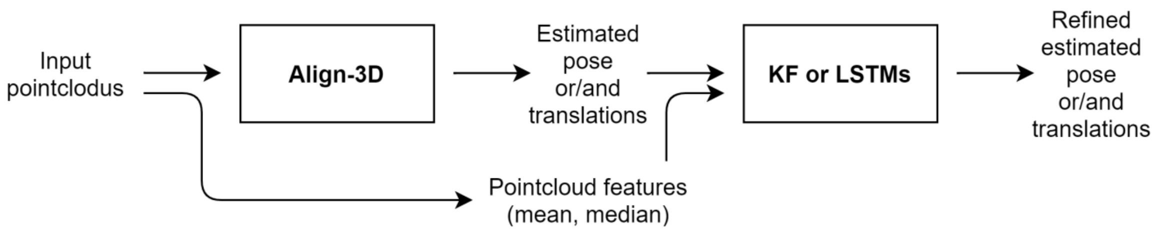

Input pointclodus
Align-3D
Estimated pose or/and translations
KF or LSTMs
Refined estimated pose or/and translations
Pointcloud features (mean, median)

Figure 13: Schematic of the system

3.1.2.Evaluation of the system

To evaluate the system we take the RMSE on the computed routes. It is computed by comparing the estimation with the ground-truth data.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\bar{y_i} - y_i)^2}{n}}$$

3.2.KF for Object tracking

As explained in section 2.1.3, the Kalman Filter is used to estimate a value given noisy measurements. This same approach can be taken to estimate the path of an object given some estimations. This section explores the use of KF for object tracking, both for pose and translation estimation.

We tried various experiments with the Kalman filter, with different input data, output data and whether we used angle information or not. The process noise covariance matrix and the measurement noise covariance matrix parameters are set up using linear search on the training set.

3.2.1.Align3D translations

The first experiment was to use the outputs of align3d (translations) as observable state and estimate translation. The state is represented by pose and velocities.

$$state = (x, y, v_x, v_y)$$

$$obs.\ state = (vx, vy)$$

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2.2.Pose estimation – Mean, median and predicted

The next iteration was to use pose estimation together with Kalman Filter to estimate the pose. In this experiment we first tried to estimate the pose with taking the mean and the median of the point cloud. Then we added estimated pose from Align3D stage 2.

$$state = (x, y, v_x, v_y)$$

$$obs.\ state = (x, v)$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2.3.Fusing data with KF

The final stage with Kalman Filter was to use both estimated pose and estimated translation together with Kalman Filter. In this stage we are fusing together pose and translations estimations.

$$state = (x, y, v_x, v_y)$$

$$obs.\ state = (x, y, v_x, v_y)$$

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3.LSTMs for Object tracking

We evaluated LSTMs performance for the object tracking task. Unlike Kalman Filters, LSTMs make no assumptions about the type of motion of the object, so they should be able to capture both linear and non-linear motion. Furthermore, because of the recurrent nature of the neural network, the LSTM can incorporate a window of the previous history when learning to predict the future position of an object. The LSTM learns a regression based on the previous N observations, where the observations are point clouds and the outputs are the estimated pose.

3.3.1.Architecture of the network

The architecture of the network is a model that has a single hidden layer of LSTM units and an output layer used to make the predictions. As inputs we use estimated pose, estimated translation or both. The number of units in the hidden layer is studied in the results section.

3.3.2.Huber loss

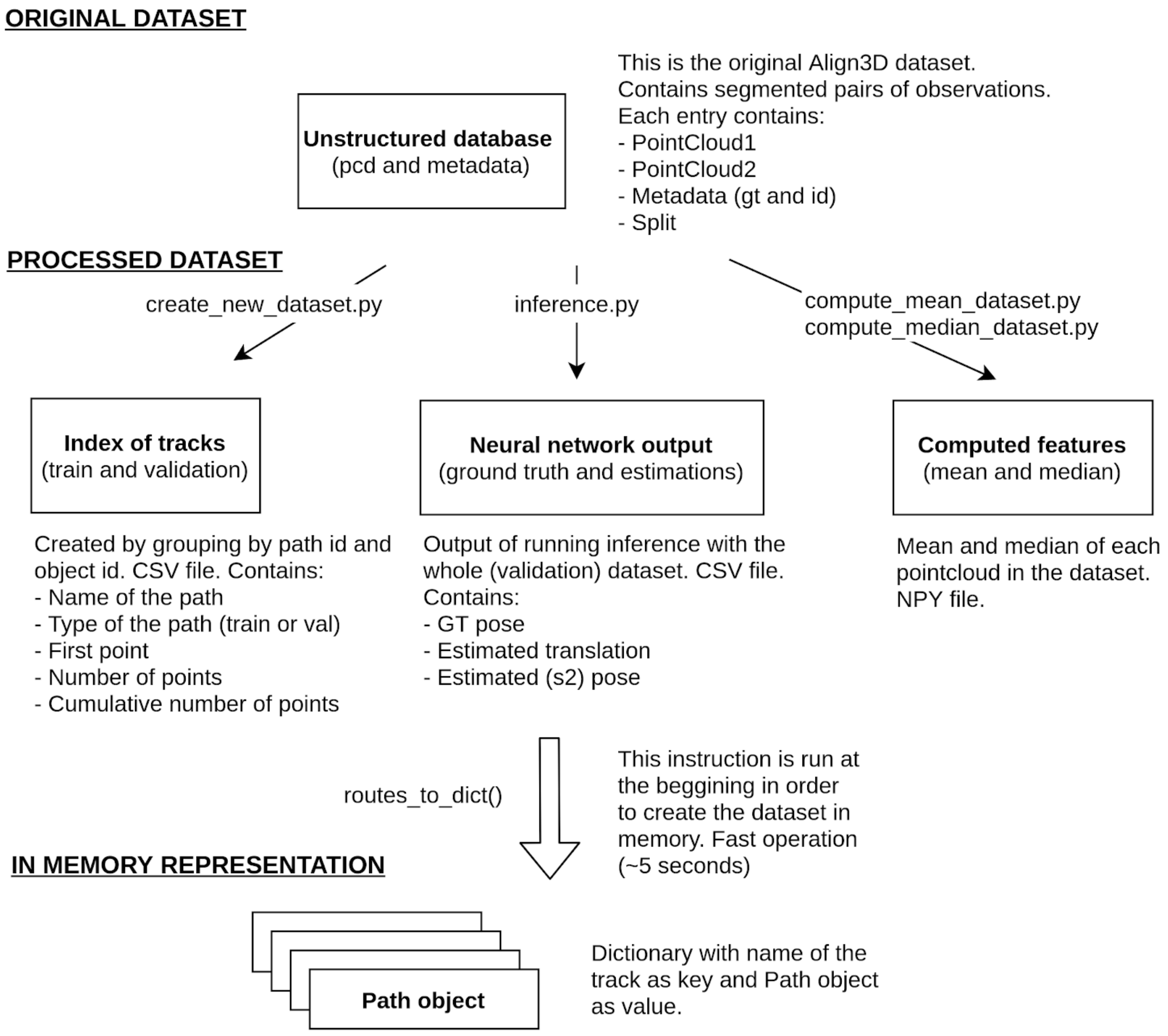
The Huber loss [28] is used for training the network. It is less sensitive to outliers in data than the squared error loss. The function is quadratic for small values of *a*, and linear for large values. With equal values and slopes of the different sections at the two points where *|a| = δ*.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| < \delta \\ \delta\left(|a| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases}$$

3.4.Implementation details

3.4.1.Dataset creation

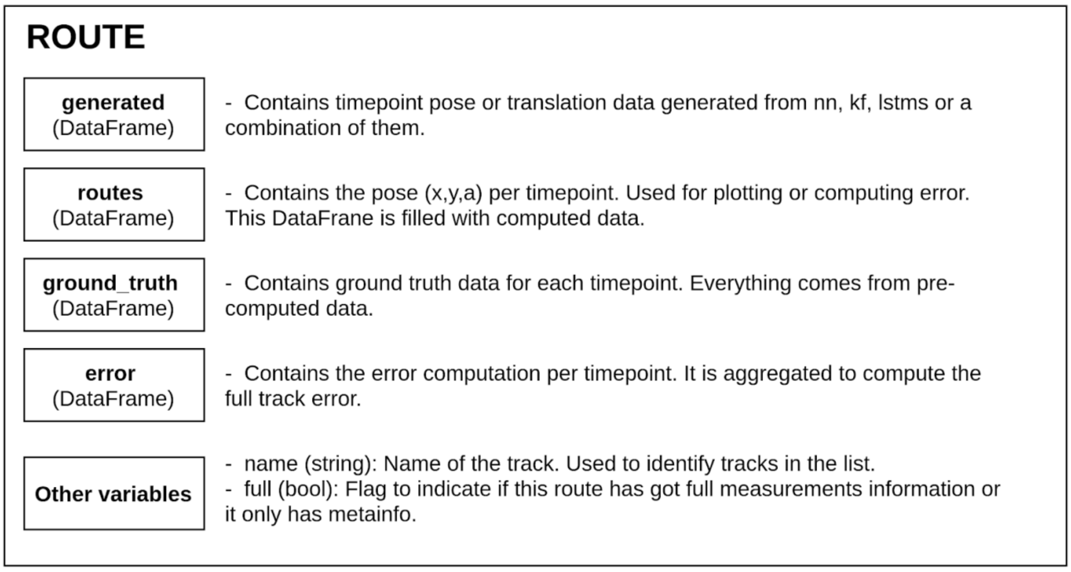
The original Align3D datasets were designed for evaluating point-cloud translation between two observations. They are already segmented; this means that each observation contains only one object. Each entry in the dataset contains two temporally adjacent observations (point clouds) and a json file with metadata containing the path id, object id and ground truth data.


ORIGINAL DATASET
Unstructured database
(pcd and metadata)
This is the original Align3D dataset.
Contains segmented pairs of observations.
Each entry contains:
- PointCloud1
- PointCloud2
- Metadata (gt and id)
- Split
PROCESSED DATASET
create_new_dataset.py
inference.py
compute_mean_dataset.py
compute_median_dataset.py
Index of tracks
(train and validation)
Neural network output
(ground truth and estimations)
Computed features
(mean and median)
Created by grouping by path id and object id. CSV file. Contains:
- Name of the path
- Type of the path (train or val)
- First point
- Number of points
- Cumulative number of points
Output of running inference with the whole (validation) dataset. CSV file. Contains:
- GT pose
- Estimated translation
- Estimated (s2) pose
Mean and median of each pointcloud in the dataset. NPY file.
routes_to_dict()
This instruction is run at the beggining in order to create the dataset in memory. Fast operation (~5 seconds)
IN MEMORY REPRESENTATION
Path object
Dictionary with name of the track as key and Path object as value.

For our problem we are interested in having full trajectories of the same object. We need to group by recording and object identification number. The proposed solution is based on creating an index of the different routes, running inference on the whole (ungrouped) dataset, computing extra features (mean and median), then loading in memory the estimations, ground truth data and extra features based on the index of the different routes.

The output of the neural network contains: ground truth pose (both point clouds), align3d output (translation), and stage 2 predicted pose (both point clouds) for each entry. The index of the routes contains: name (path and object id), type (train or val), first point, number of points, and cumulative sum of the number of points.

The in-memory representation is based on a class containing: name, flag for indicating if it is a full route (contains measurements or just metadata) and dataframes for generated routes (kf, lstms), ground_truth (straight from NN output), routes (used for plotting and error measurement) and error (error computation per timepoint).



ROUTE

generated (DataFrame)	- Contains timepoint pose or translation data generated from nn, kf, lstms or a combination of them.
routes (DataFrame)	- Contains the pose (x,y,a) per timepoint. Used for plotting or computing error. This DataFrane is filled with computed data.
ground_truth (DataFrame)	- Contains ground truth data for each timepoint. Everything comes from pre-computed data.
error (DataFrame)	- Contains the error computation per timepoint. It is aggregated to compute the full track error.
Other variables	- name (string): Name of the track. Used to identify tracks in the list. - full (bool): Flag to indicate if this route has got full measurements information or it only has metainfo.

Figure 14: Class schematic

During our experiment, we thought that adding new features to the model for object tracking could be a good idea. The first/simplest iteration was to add the mean of the observations as input to the Kalman filter and combine that data with the estimated translations. The procedure to do this is to first compute the features, store them as an external file, and then add them to the in-memory representation. This way we have an extendable, decoupled way of adding extra information to the model. Eventually, we added the mean, median, and the prediction from stage 2 (align3d) of the observations. All of them are added to the generated data frame.

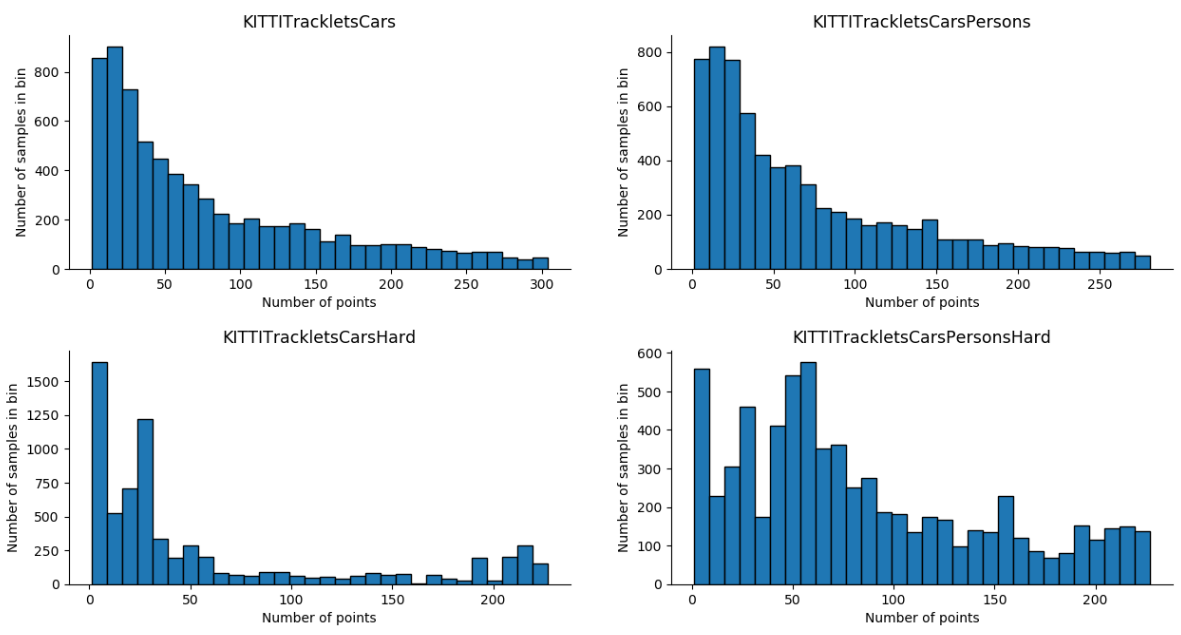
3.4.2.The dataset

In this section we analyse the existing align3D dataset, both in the raw/unstructured observation domain and also with routes.

Number of points per observation statistics

	MEAN	MEDIAN	STD
KITTI_CARS	395.18	108.0	772.94
KITTI_CARS_PERSONS	381.35	102.0	761.85
KITTI_CARS_HARD	148.51	55.0	163.94
KITTI_CARS_PERSONS_HARD	193.44	113.0	199.65

Histogram number of points in observations (capped at p80)

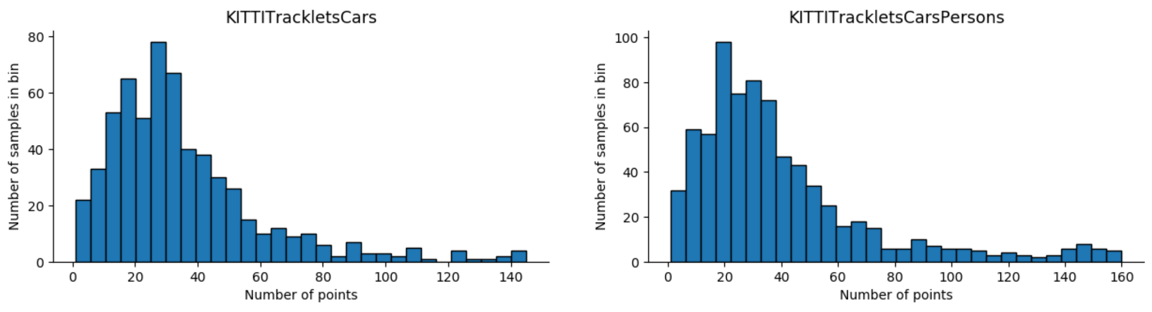


In the standard datasets (not hard) we have more points per observation and also a very long tail with some observations with thousands of points. Also they present the same distribution. In the case of the hard datasets we get fewer points per observation, and lower variability too.

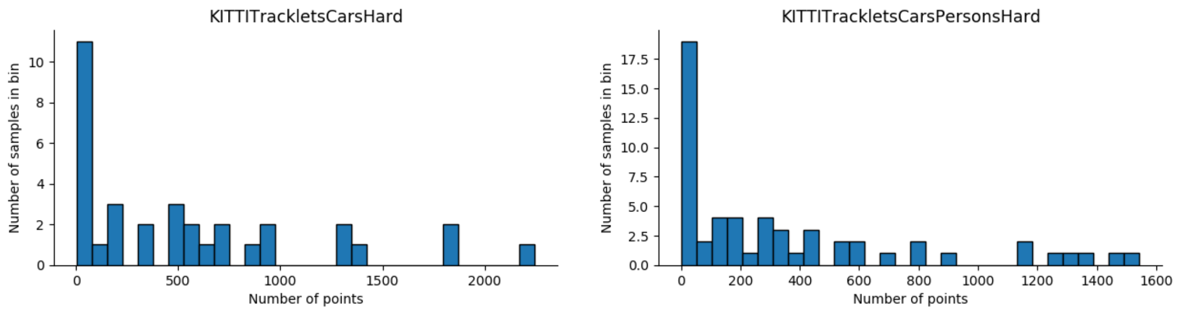
Number of observations per route statistics

	MEAN	MEDIAN	STD
KITTI_CARS	46.37	30.6	57.39
KITTI_CARS_PERSONS	50.80	32.0	59.89
KITTI_CARS_HARD	805.55	487.0	1209.37
KITTI_CARS_PERSONS_HARD	491.53	280.0	734.13

Histogram number of observations in routes (capped at p95)



Analyzing the number of observations it is clear that the kitti_cars and kitt_cars_persons datasets are very similar, both the statistics and the distributions are really close. The hard datasets contain some very long routes, that is why the statistics and distributions are like this.



Number of pairs and routes per dataset

	N.PAIRS			N_ROUTES		
	TRAIN	**VAL**	**TOTAL**	**TRAIN**	**VAL**	**TOTAL**
KITTI_CARS	20518	8790	29308	463	169	632
KITTI_CARS_PERSONS	28463	12072	40535	561	237	798
KITTI_CARS_HARD	20000	9000	29000	22	14	36
KITTI_CARS_PERSONS_HARD	20000	9000	29000	39	20	59

Class analysis per dataset

	Van		Car		Pedestrian	
	Obs.	**Route**	**Obs.**	**Route**	**Obs.**	**Route**
KITTI_CARS	3236	55	26072	577	0	0
KITTI_CARS_PERSONS	3236	55	26072	577	11227	166
KITTI_CARS_HARD	4817	5	24813	31	0	0
KITTI_CARS_PERSONS_HARD	2678	4	10970	31	15352	24

From the last two tables we see that the kitti_cars_persons is just the kitti_cars dataset plus some pedestrian observations. And that the hard datasets contain fewer and longer routes.

3.4.3.Angle analysis and correction

From analysing the straight Align3d results, something that surprised me was the inacurate results it got on angle estimation, especially compared to pose estimation.

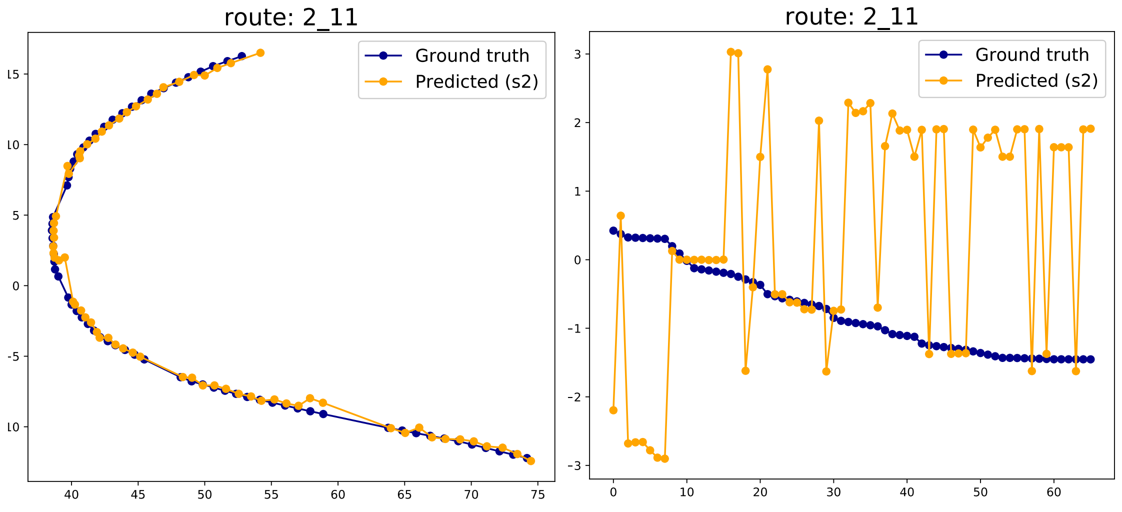


Figure 15: Pose and angle estimation in route 2_11

The error on the whole dataset is very high by default. Then we tried a few techniques to improve the results. The first technique was to normalize the estimations to [-π, π]. The second was to check if there was brusque changes (>π/2) in the angle, as they cannot happen in the real world from frame to frame, and adding pi when this happened. And the third one was to compute the trajectory angle $atan(\Delta y, \Delta x)$ and check that the estimated angle does not differ with the trajectory angle by (>π/2), in case it was we added π.

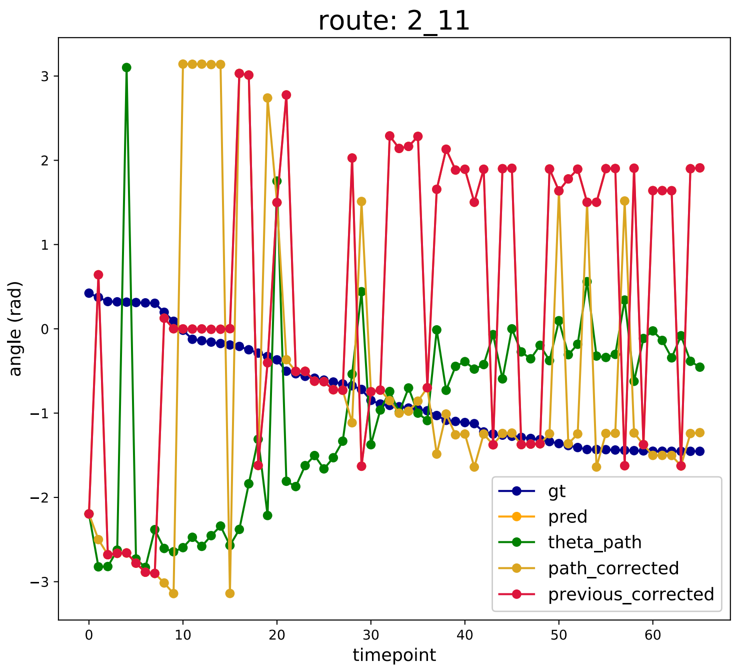


Figure 16: Angle corrections on route 2_11

After some experiments and applying the different methods outlined in the previous paragraph we didn't manage to significantly improve the angle estimations. Even the estimated theta is way different than the ground truth, (in many cases) this is because the camera is moving and the object we are tracking is still. The conclusion is that these angle estimations are too inaccurate to work with them, and we have dismissed for future experiments.

3.4.4.Evaluation framework

Creating routes from translation - relative and absolute routes

Align3d generates translations from two pairs of observations. This presents the problem of how to evaluate object tracking given only translations. One solution could be to just evaluate the translations of the different observations, but this means limiting the scope of the object tracking to only estimate translations between frames. Furthermore in this case we are only evaluating the error on a sample by sample basis, we are not taking into account the accumulated error from the previous frame by not taking a reference point.

The approach we chose to follow tries to solve both problems by computing an estimated route, whether if we are estimating translations or directly estimating the pose. In the latter case, the processing is straightforward. But in the case of estimating translation a problem is presented, how do we create the route if we only have estimated translations?

Initially, we created two solutions. The first was to create a relative route, this means taking the previous point in the ground truth data and adding the translation. Essentially evaluating the translation estimation without taking into account accumulating error. The second was to take the ground truth data for the first point and keep adding the estimated translations for each. In this case, we are taking into account the outcomes of all the estimations. Eventually, we concluded that the correct approach was to compare the estimated absolute routes with the directly pose estimated routes.

Computing error

As indicated in Figure X the class contains a DataFrame with the ground truth data for x, y and angle. Then it is easy to compute the estimation error based on any loss function for a single route.

Taking advantage of the data frames we add columns with the error at each observation and value. Therefore we have a registry of all the measurements, estimations, and errors at any level of granularity.

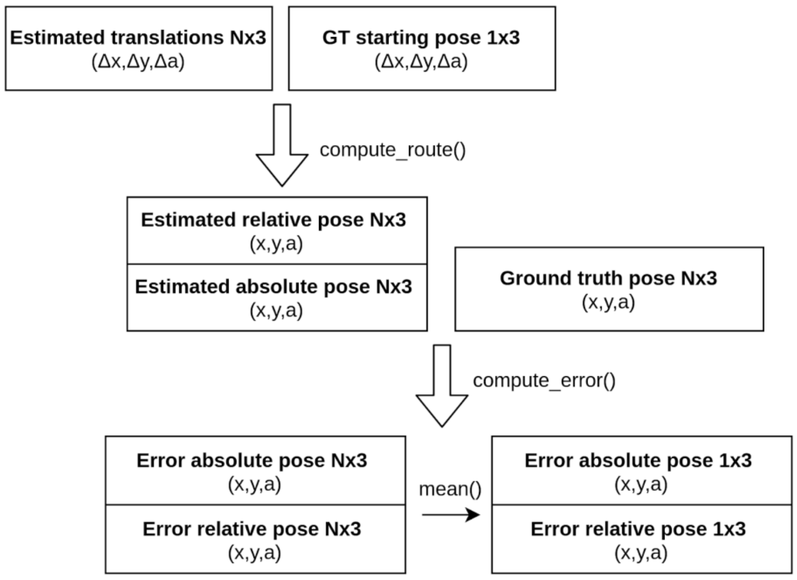


Figure 17: Compute error diagram

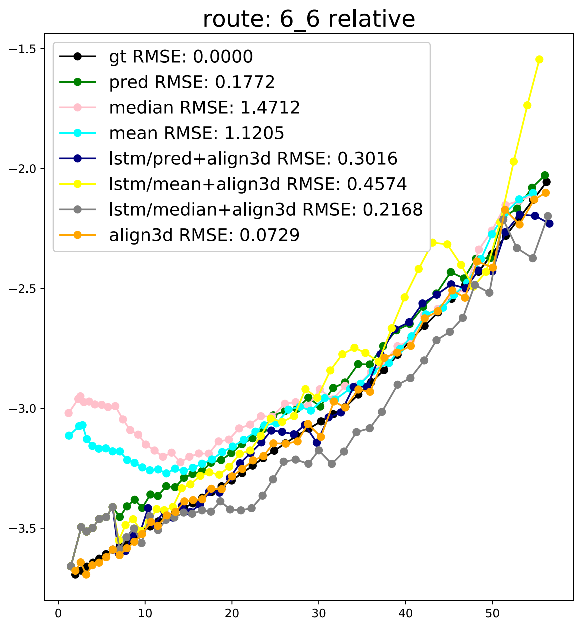
To create a more general and easier to use framework this operation is done automatically for all the routes in the dataset, both relative and absolute. The dataset keeps track of the different routes with the name_routes parameter. It is a simple class that contains two lists, one for relative routes and one for absolute ones. Based on the names on name_routes it computes the error for all the observations and routes in the dataset.

3.4.5.Visualization of the results

To qualitatively analyse the results we created a visualization tool. It takes all the routes from the routes dataframe and plots them in a dual plot, containing the relative and absolute pose. Note that for the methods that directly compute pose, the relative and absolute pose are the same.

There is also the option to visualize the angle in the plot. It paints an arrow in the direction of the angle on each point. The length is computed by taking the magnitude of the translation vector and computing a percentage of its.

The visualizations are implemented using Matplotlib [29] and 3Djs [30]. They are interactive so we can focus on each route and zoom as much as needed.



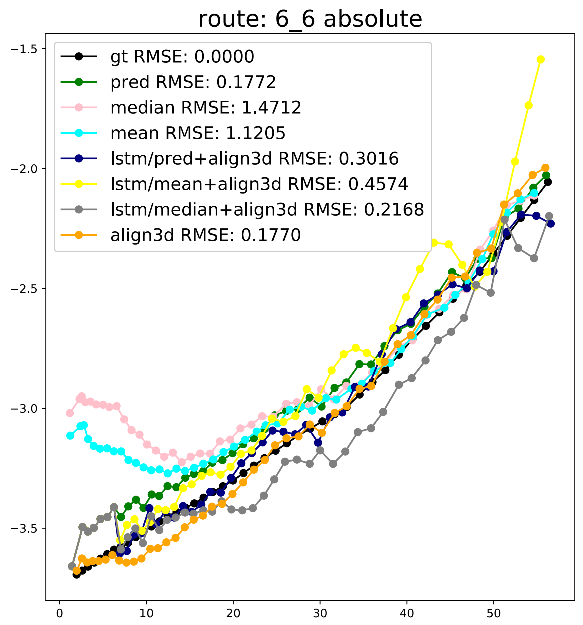


Figure 18: Example plot

3.4.6.Open source

All the code used for creating the datasets, visualizing the results, running inference on AlignNet-3D, running Kalman Filter/LSTM and evaluating the results can be found in the following link: https://github.com/jaumecolomhernandez/3d-object-tracking-lstms.

4.Results

4.1.Direct estimation

4.1.1.Tables of results

Results pose estimation

ROUTE	RMSE
Mean	1.0275
Median	1.2736
Pred	0.4868

Results AlignNet-3D (translation)

ROUTE	RMSE
Align3D	0.9055

4.1.2.Discussion

In this plot we see that mean and median are dependent on having a good/complete observation of the object. At the beginning of the path we see that it separates a lot from the ground truth, but through the trajectory it gets closer. The predicted pose from AlignNet-3D is very consistent through all the trajectories and that is the reason why it has got such a low RMSE in the results table.

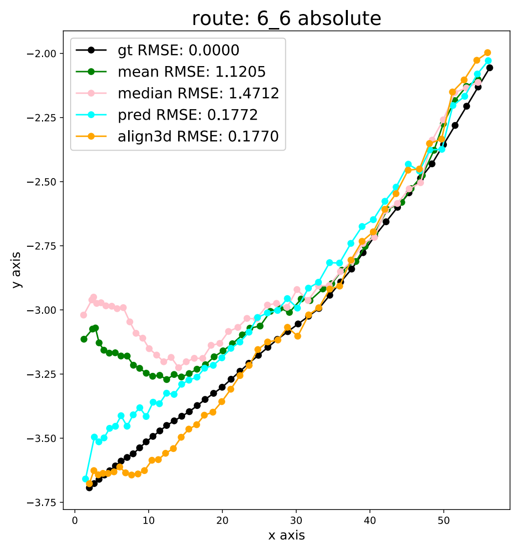


Figure 19: Pose estimation and AlignNet-3D in route 6_10

The estimated translations from AlignNet-3D are quite accurate. Although the problem is that in some cases the translation error adds up and the final estimated trajectory has a bad performance.

4.2.Kalman Filter

4.2.1.Tables of results

Results pose estimation and kalman filter (pose)

ROUTE	RMSE
KF/Mean	2.0889
KF/Median	2.3012

KF/Pred	1.6867

Results align3D and kalman filter (translation)

ROUTE	RMSE
KF/Align3D	5.9775

Results fusing data with kalman filter (translation)

ROUTE	RMSE
KF/Mean+Align3D	2.0624
KF/Median+Align3D	2.3014
KF/Pred+Align3D	1.4434

Results fusing data with kalman filter (pose)

ROUTE	RMSE
KF/Mean+Align3D	0.7406
KF/Median+Align3D	0.7526
KF/Pred+Align3D	0.6780

4.2.2.Discussion

In the case of Kalman filter applied to pose estimations (mean, median and pred) we see that the performance degrades a lot, almost doubling the RMSE.

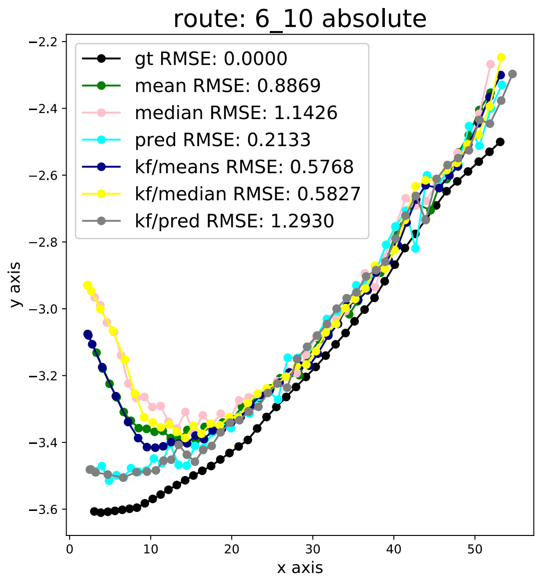


Figure 20: Kalman Filter + pose estimations in route 6_10

For Kalman Filter applied to AlignNet-3D the results are even worse, the RMSE jumps to near 6. As we can see in Figure 22, what happens is that the translations overshoot on the initial direction. In the cases where the trajectory is straight there is not any problem, but in the cases where the trajectory curves it is prominent.

The big improvement comes when we apply kalman filter to pose and translation information, as both inputs are fused in a better estimate. In Figure 24 we see a plot with the resulting route from fusing mean, median and prediction with AlignNet-3D translation estimations. Once again the best results come from using the predicted pose. Althought in all cases the results improve a lot from just using the AlignNet-3D or the Pred estimation.

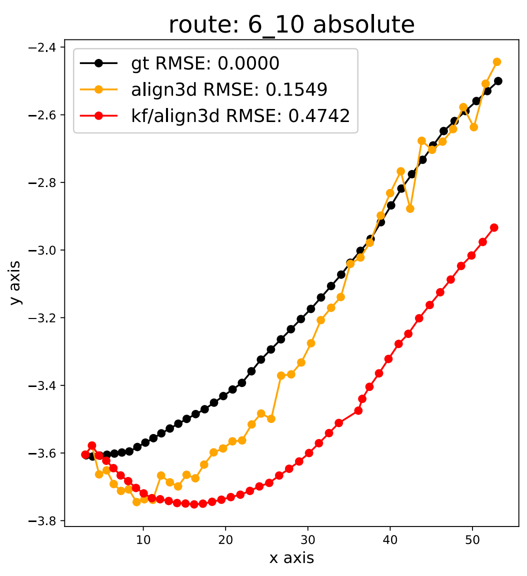


Figure 21: Kalman Filter + translation

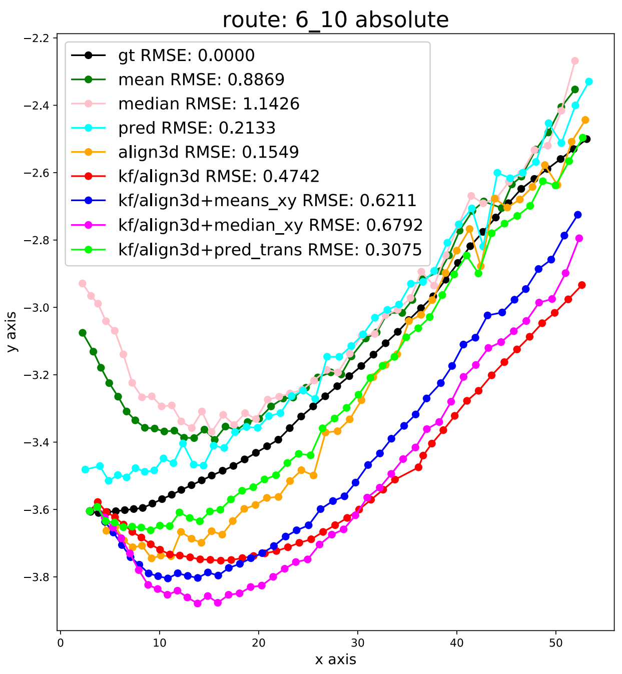


Figure 22: Kalman Filter + pose + translation

4.3.LSTMs

4.3.1.Tables of results

Results pose estimation and LSTMs (pose)

ROUTE	RMSE
LSTMs/Mean	0.8202
LSTMs/Median	0.8460
LSTMs/Pred	0.4842

Results AlignNet-3D and LSTMs (translation)

ROUTE	RMSE
LSTMs/Align3D	0.9619

Results fusing data with LSTMs (pose)

ROUTE	RMSE
LSTMs/Mean+Align3D	0.4997
LSTMs/Median+Align3D	0.4994
LSTMs/Pred+Align3D	0.4207

Results fusing data with LSTMs (translation)

ROUTE	RMSE
LSTMs/Mean+Align3D	1.4197
LSTMs/Median+Align3D	1.4265
LSTMs/Pred+Align3D	1.0895

4.3.2.Discussion

In Figure 25 we see LSTMs applied to pose estimation. We see that there is a big improvement in mean and median poses, but with predicted (s2) pose it improves marginally.

This may be because the predicted poses are much more accurate than the mean and median poses, so the network has more difficulties to learn the correction.

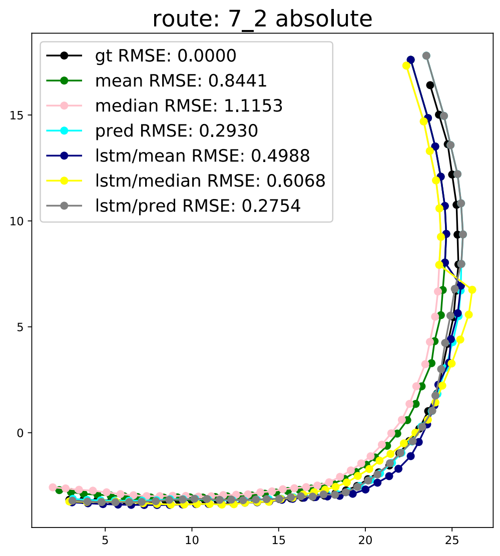


Figure 23: LSTMs + pose

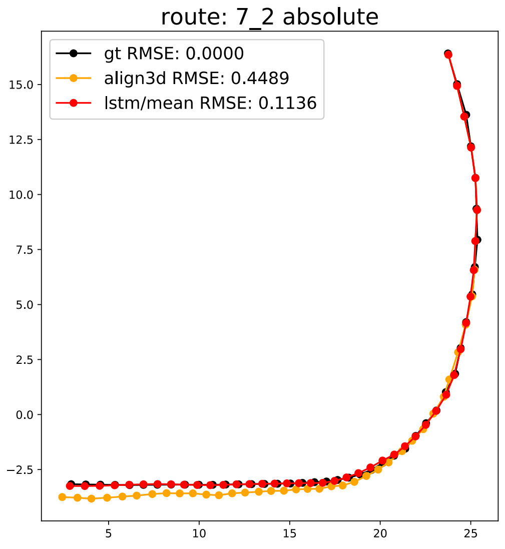


Figure 24: LSTMs + translation

In the next Figure, we see LSTMs applied to the AlignNet-3D translation estimations. In this case we see a big improvement respect to Kalman Filter, while in the previous case the results drifted a lot due to the filter, now they improve. Although that applied to the whole dataset the results are marginally the same.

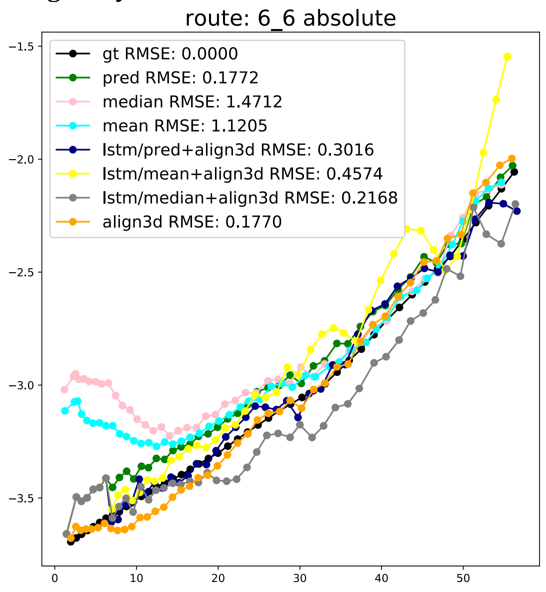


Figure 25: LSTMs + pose and translation estimation

In the last plot we see LSTMs with pose and translation estimation. The results are significantly better than anything seen up to now. In the case that uses predictions as pose estimation it gets an RMSE on the whole dataset of 0.42, which is significantly lower than the 0.48 of the AlignNet-3D pose estimations.

5.Budget

This thesis is not a prototype, but a research exercise on extending a current publication related to object tracking. It is entirely developed in Python and uses free open-source libraries.

The development costs come from the salaries paid to engineers. In this case it was a senior engineer supervising the project 2 hours per week and a junior engineer (me) working 25 hours per week on the project. The project had a full duration of 25 weeks.

Person	Hours per week	Cost per hour	Full cost (25 weeks)
Senior engineer	2 h/week	50 €/h	2500 €
Junior engineer	25 h/week	15 €/h	9375 €
		TOTAL	11875 €

6.Conclusions and future development:

The main objective of this thesis was to analyse and extend the AlignNet-3D publication [9] from estimating the relative motion between two observations to do object tracking in full trajectories. Moreover, another objective of the project was to become familiar with deep learning and computer vision concepts.

The results obtained in the project are satisfactory. We have accomplished the first objective, developing a full framework to estimate the trajectories in the KITTI dataset, running many experiments with the raw results from AlignNet-3D, with the Kalman Filter and also with recurrent neural networks. The best results are achieved with using pose and translation estimation with LSTMs to directly estimate pose, with an RMSE error of 0.42 on the whole dataset. It is relevant that the results from just using the AligNet-3D pose estimations (stage 2) are already very accurate, achieving an RMSE error of 0.48 without the need of any extra processing.

Regarding the second objective I also consider it accomplished. I have gone from having little experience with computer vision and point clouds to understanding the relevant publications in the area, knowing how to structure a project of this scope, preparing/analysing the datasets and performing experiments on a new problem.

Finally, as future work, the next step is to create an encoder-decoder architecture to encode a representation of the object of analysis, similar to the proposal of [31]. So that each observation adds information to the latent representation and this is used to improve the tracking.

Bibliography

[1] Hao Jiang, Zongwei Liu Fuquan Zhao, "Recent development of automotive LiDAR technology, industry and trends," in *ICDIP 2019*.

[2] Xianfeng and Laga, Hamid and Bennamoun, Mohammed Han, "Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[3] Sebastian Weiss and Mengyu Chu and Nils Thuerey and Rüdiger Westermann, Volumetric Isosurface Rendering with Deep Learning-Based Super-Resolution, 2019.

[4] Jonathan and Shelhamer, Evan and Darrell, Trevor Long, "Fully Convolutional Networks for Semantic Segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[5] Aaron and Kalchbrenner, Nal and Espeholt, Lasse and kavukcuoglu, koray and Vinyals, Oriol and Graves, Alex van den Oord, "Conditional Image Generation with PixelCNN Decoders," in *Advances in Neural Information Processing Systems*, 2016.

[6] Kaiyu Yang, Jia Deng Alejandro Newell, "Stacked Hourglass Networks for Human Pose Estimation," in *European Conference on Computer Vision*, 2016.

[7] Ilya Sutskever, Geoffrey E. Hinton Alex Krizhevsky, "ImageNet Classification with Deep Convolutional Neural Networks," in *Neurips 2012*.

[8] Alex Krizhevsky, Learning multiple layers of features from tiny images, 2009.

[9] Johannes Gross and Aljosa Osep and Bastian Leibe, "AlignNet-3D: Fast Point Cloud Registration of Partially Observed Objects," in *International Conference on 3D Vision (3DV)*, 2019.

[10] Rudolph Emil Kalman, A New Approach to Linear Filtering and Prediction Problems, 1960.

[11] Jürgen Schmidhuber Sepp Hochreiter, Long short-term memory, 1997.

[12] Aljosa Osep, Yutong Ban, Radu Horaud, Laura Leal-Taixe, Xavier Alameda-Pineda Yihong Xu, "How To Train Your Deep Multi-Object Tracker," in *Computer Vision and Pattern Recognition (cs.CV)*, 2020.

[13] Laura Leal-Taixé Guillem Brasó, "Learning a Neural Solver for Multiple Object Tracking," in *Computer Vision and Pattern Recognition (cs.CV)*, 2020.

[14] Ismail Elezi, Laura Leal-Taixé Maxim Maximov, "CIAGAN: Conditional Identity Anonymization Generative Adversarial Networks," in *Computer Vision and Pattern Recognition (cs.CV)*, 2020.

[15] Patrick Dendorfer and Hamid Rezatofighi and Anton Milan and Javen Shi and Daniel Cremers and Ian Reid and Stefan Roth and Konrad Schindler and Laura Leal-Taixé, MOT20: A benchmark for multi object tracking in crowded scenes, January 2020, arXiv 0712.0157.

[16] javatpoint.com, Regression vs classification machine learning , https://www.javatpoint.com/regression-vs-classification-in-machine-learning.

[17] Jaume Colom, Neural network implementation, https://github.com/jaumecolomhernandez /simple-net.

[18] Nitesh Goyal, Perceptron Algorithm, https://mlforanalytics.com/2018/04/29/implementation-of-perceptron-algorithm-using-python/.

[19] Christopher Olah, Understanding LSTM Networks, 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[20] Rudolph Emil Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME--Journal of Basic Engineering*, vol. 82, no. 82, pp. 35-45, 1960.

[21] Charles R. Qi and Hao Su and Kaichun Mo and Leonidas J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Computer Vision and Pattern Recognition (cs.CV)*, 2016.

[22] S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao Z. Wu, "3D ShapeNets: A Deep Representation for Volumetric Shape Modeling," in *IEEE Conference on Computer Vision and Pattern Recognition* , 2015.

[23] Charles R. Qi and Hao Su and Matthias Niessner and Angela Dai and Mengyuan Yan and Leonidas J. Guibas, "Volumetric and Multi-View CNNs for Object Classification on 3D Data," in *IEEE*

Conference on Computer Vision and Pattern Recognition , 2016.

[24] Sebastian Scherer Daniel Maturana, "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[25] P. J. Huber, "Robust estimation of a location parameter," in *The Annals of Mathematical Statistics*., 1964, pp. 73-101.

[26] Xinshuo Weng and Kris Kitani, "A Baseline for 3D Multi-Object Tracking," in *Computer Vision and Pattern Recognition (cs.CV)*, 2019.

[27] Bin Yang, Raquel Urtasun Wenjie Luo, "Fast and Furious: Real Time End-to-End 3D Detection, Tracking and Motion Forecasting with a Single Convolutional Net," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake, UT, USA, 2018.

[28] Peter J. Huber, Robust Estimation of a Location Parameter, 1964, Annals of Statistics. 53 (1): 73–101.

[29] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90--95, 2007.

[30] Vadim Ogievetsky, Jeffrey Heer Michael Bostock, D3: Data-Driven Documents, 2011.

[31] Jesus Zarzar, Bernard Ghanem Silvio Giancola, "Leveraging Shape Completion for 3D Siamese Tracking," in *Computer Vision and Pattern Recognition (cs.CV)*, 2019.

[32] Wenhan Luo and Junliang Xing and Anton Milan and Xiaoqin Zhang and Wei Liu and Xiaowei Zhao and Tae-Kyun Kim, Multiple Object Tracking: A Literature Review, 2014, arXiv:1409.7618.

[33] Maria Cristina Munuera Raga, Filtro de Kalman y sus aplicaciones, 2018, TFG Grau matemàtiques UB.

[34] David Pampliega Ruiz, Algoritmo de SLAM para el ROMEO-4R, 2008, PFC Ingeniero de Telecomunicación.

[35] Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, 2015, http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[36] Lisa Tagliaferri, An Introduction to Machine Learning, 2017, https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning.

[37] James Liang, An introduction to Deep Learning, 2018, https://towardsdatascience.com/an-introduction-to-deep-learning-af63448c122c.

[38] Nick McCrea, An Introduction to Machine Learning Theory and Its Applications, -, https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer.

[39] Loic and Simonovsky, Martin Landrieu, "Large-Scale Point Cloud Semantic Segmentation With Superpoint Graphs," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.