3D Multi Object Detection with Lidar for Autonomous Driving

Yagmur Cigdem Aktas

Department of Computer Architecture and Technology
University of Burgundy

A Thesis Submitted for the Degree of
MSc in Vision and Robotics (VIBOT) created in Kitware SAS

· 2022 ·

Abstract

Object tracking is a fundamental computer vision problem that refers to a set of methods proposed to precisely track the motion trajectory of an object in a video or in sequenced frames. Multiple Object Tracking (MOT) is a subclass of object tracking where the goal is to track not a single but multiple objects from one or multiple classes.

Multi-object tracking is an important ability for an autonomous vehicle to safely navigate and a lot of different public dataset exists especially for autonomous driving. Among these datasets, the KITTI dataset has been chosen for this work. It has 21 training and 29 test sequences in the tracking dataset which results in 7481 training, and 7518 test frames with camera data (images), Velodyne lidar data (3D point clouds), calibration files, and ground truth labels. To create a tracking pipeline, first, the Object Detection models compatible with Lidar are reviewed. Using the OpenPCDET framework [22], the reviewed models are trained with the KITTI object detection dataset using only Point Cloud data but not images. SECOND [26] object detection model is chosen considering its accuracy and speed performance.

The tracking part has been created by being inspired by two main pieces of research: Probabilistic 3D Multi-Modal, Multi-Object Tracking for Autonomous Driving [5] and AB3DMOT [23] which are both in the tracking-by-detection category and use Kalman Filter. They have different approaches to how to use Kalman predicted trackers to associate them with detected objects. While one [5] uses Mahalanobis Distance with Greedy Algorithm, other [23] prefers IOU with Hungarian algorithm as the matching method. In this work, all of these methods are reviewed, explained, and tested to compare their performance. Their limitations and significant errors which cause to decrease in the performance of tracking are analyzed and various optimizations are proposed.

Finally, it is shown that the accuracy results are increased by %72 for pedestrian, %14 for car class and %16 for cyclist class using Mahalanobis Distance, Optimized Greedy Algorithm with updated maximum age and kalman prediction parameters and without adding Match Deny Mechanism or Ego Motion Compensation which are explained in the chapter 4.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my family, Sahin, Kezban and Burak Aktas for their unconditional love and support, my two lovely friends Emmanouil M. and Emmanouil K., for being such a great accompany to survive in Le Creusot, and my supervisor Lea Vauchier, for her guidance and assistance during this work.

Chapter 1

Introduction

The multi-object tracking (MOT) system performs accurate tracking of obstacles moving in front of or in the surrounding environment of an autonomous vehicle, including vehicle path tracking, non-motor vehicle trajectory tracking, pedestrian trajectory tracking, etc. This subsystem helps self-driving cars make decisions and avoid collisions with objects that may move. The main task of the multi-object tracking algorithm is to track many objects simultaneously and assign and maintain a corresponding ID for each object, which cannot be achieved by only using the object detection algorithm or single object tracking algorithm.

The MOT methods are divided into two main categories according to the fundamental operating logic. While Tracking by Detection (TBD) methods use a system having an independent tracking algorithm applied to a detection model which estimates the bounding box location and orientation of each object in each frame, Joint Detection and Tracking (JBD) methods use a unified system where tracking and detection models are not independent but affecting each other. This work is inspired by two main TBD models and focuses on the optimizations of these models.

After significant improvements in processing Point Clouds using Deep Learning, the tasks like object detection, and object tracking with Point Clouds have gained speed and become a state of the art thanks to its ability to give 3D information of the environment with depth and its wide range over the cameras. Even if there are RGBD types of cameras that produce 3D data with depth, these are not at a competitive level with Point Cloud sensors' (LIDAR, RADAR) range.

We aimed to first analyze object detection models compatible to extract features from Lidar data, and create a tracking mechanism over these detections in a TBD form.

1.1 Problem Statement

Even if there are various types of public autonomous driving datasets [3, 8, 25] including both the image and lidar data for each frame, each dataset has its advantages and disadvantages. It is important to analyze these different datasets and choose the optimal one to continue with.

Since in a TBD system, the detection and tracking algorithms are independent of each other and the tracking algorithm doesn't have any effect on detection results, to obtain a successful tracking model, first, an optimal object detection model should be obtained. For this purpose, various object detection models compatible with working on Point Clouds should be reviewed, and fine-tuned to take a final decision about the detection model to continue with.

Even if the optimal detection model is chosen, misdetections and occlusions are still two main challenges for tracking tasks. Both cause missing detections which make it harder to estimate the tracker's new position. A tracker's trajectory may be fragmented since its detection is lost, and its ID may be changed if the tracker gets matched with another detection while its target detection is lost. Therefore, meanwhile, the tracker model is helpful to handle detection losses thanks to its ability to estimate the next position according to the previous frame's location and its updated velocity, the more the detections are noisy, the more the tracker results get noisy, and difficult to continue with robust estimates.

Another challenge that comes with the dataset is that although Lidar has different advantages over images, its use is not as common as images which causes most of the benchmarks [6,8] to be implemented by taking images as the base. This means both ground truths (2D, and 3D) are in image space and the tracker results should be transformed from lidar to image space to be able to perform an official evaluation of the final model.

In summary, to perform 3D Multi-Object Tracking with Lidar, the following challenges are reviewed:

- Having a good intuition about Lidar data and the autonomous driving dataset to choose an optimal one.
- Making a deep review about Object Detection models compatible with Lidar processing and fine-tune to select the optimal one.
- Implementing a track by detection mechanism using constant velocity Kalman Filter as fundamental step, trying different distance metrics and matching algorithms to analyze their performance. Analyzing the reasons for significant errors to optimize the model.
- Transforming the track results in the correct format and in image space to perform benchmark evaluation.

Chapter 2

Background

In this chapter the technical background about different data types and Lidar, the main object tracking types, the features of the KITTI [8] dataset which is used for this project will be explained to give a better understanding before implementing the tracking mechanism.

2.1 Different Data Types and Lidar

We are more familiar with 2D data types like images we used to see and use in our daily lives. A digital image is either a 1 channel grayscale image describing the environment in pixels having different intensities i.e in a range [0-255], or a 3 channel RGB image having the same type of pixels for red, blue, and green color channels.

Figure 2.1: RGB vs Grayscale Image

Since they are describing the environment in 2D, they don't contain the depth information which is very useful when it comes to specific tasks like autonomous driving where the depth information is useful to obtain the distance the detected object and used for various decision tasks. The 3D information about the environment can be obtained with different devices.

2.1.1 RGB-D Camera

An RGB-D camera is simply a combination of a RGB camera with a depth sensor. It produces a RGB image with the depth map of the image at the same time.

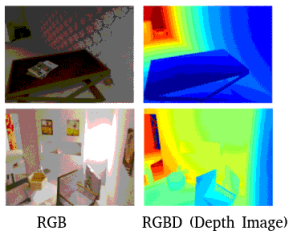

Figure 2.2: RGBD Image and Depth Map (modified and retrieve from [7])

2.1.2 Point Cloud

A point cloud is a digital 3D representation of a physical object or space. It's made of millions of individual measurement points, each one with an x, y, and z coordinate representing their cartesian coordinates in the sensor's frame. Depending on the method used to capture the cloud and the sensors involved, each point can also include RGB color data, or even intensity information. There are two primary methods to capture a point cloud: lidar and photogrammetry.

Photogrammetry

Generating a point cloud with photogrammetry is the process of 3D reconstruction of a scene using multiple cameras (or at least a stereo camera) capturing the 2D images from different angles. Therefore this methodology is based on processing 2D information to obtain a 3D information rather than creating a 3D environment directly.

Lidar

Being the base component of lidar scanner, laser is a device that generates an intense beam of coherent monochromatic light.

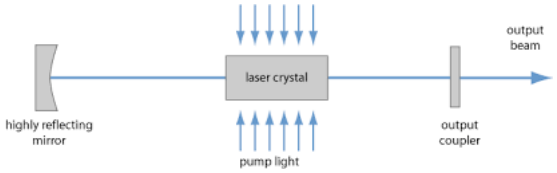

Figure 2.3: Laser Structure

Also called 3-D laser scanning, Lidar(light detection and ranging) is the method for determining ranges by targeting an object or a surface with a laser and measuring the time for the reflected light to return to the receiver. A lidar scanner consists of multiple laser beams which

are sending the light beam at the same time from different angles and receiving the reflected light to calculate the distance of the object the light is reflected from. The scan process refers to the lidar sensor completing one full tour making 360° rotation in z axes to collect the 3D information from the environment for 1 frame.

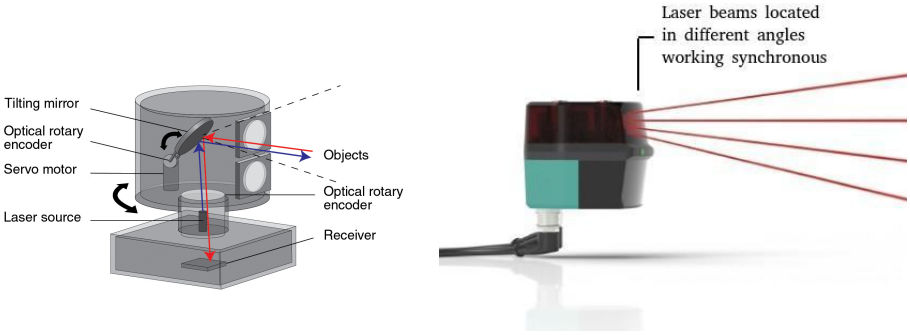

Figure 2.4: Lidar Scanner Structure

The number of the points obtained for 1 frame depends on the laser beam count that the Lidar sensor has i.e a Velodyne H64 Lidar Scanner that has 64 laser beams located collects two times more information (two times more 3d points) than a Velodyne H32 having 32 laser beams. An example of 1 frame Point Cloud obtained by a Laser Scanner and another example colored by height values are as shown in figure 2.5 2.5

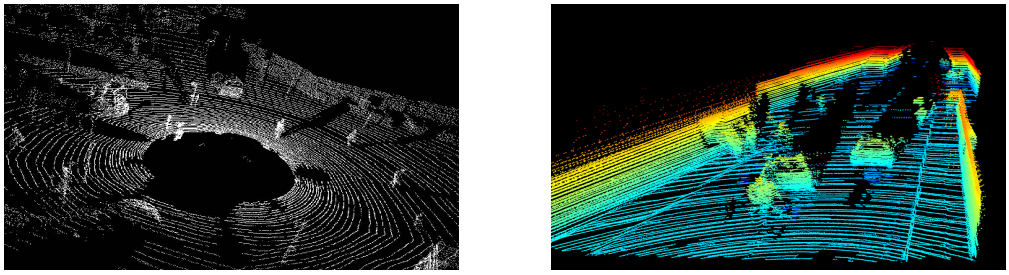

Figure 2.5: Point Clouds Examples (modified and retrieved from [16])

Photogrammetry and Lidar point clouds have both their advantages. Photogrammetric point clouds have an RGB value for each point, resulting in a colorized point cloud which makes it easier to interpret the scene. On the other hand, it is not as accurate as lidar since it requires an acquisition step from 2D data. While photogrammetry has its spesific use cases, in autonomous driving Lidar is the sensor used for most public dataset creation. In this project, the point cloud data refers to point clouds created by Velodyne H64 Lidar Scanner since it is

the one used for KITTI [8] dataset acquisition.

2.2 Object Tracking Types

Object tracking refers to estimating the state of the target object present in the scene from previous information. On a high level of abstraction, there are mainly two levels of object tracking: Single Object Tracking (SOT) and Multiple Object Tracking (MOT)

In SOT, the bounding box of the target object is defined in the first frame. The goal of the algorithm is then to locate the same object in the rest of the frames. SOT belongs to the category of detection-free tracking because one has to manually provide the first bounding box to the tracker. This means that Single Object Trackers should be able to track whatever object they are given, even an object on which no available classification model was trained.

Multiple Object Tracking (MOT) refers to the approach where the tracking algorithm tracks every single object of interest. Initially, the tracking algorithm determines the number of objects in each frame, following that it keeps track of each object's identity from one frame to the next frame until they leave the frame.

Both can be performed with 2D or 3D bounding boxes according to the obtained detection results, independently from the source data(2D or 3D) used for the task.

Figure 2.6: 2D vs 3D Detections on an Image (modified and retrieved from [8])

2.3 KITTI Autonomous Driving Dataset

After a review of three public autonomous driving datasets [3, 8, 25], KITTI [8] dataset has been chosen since it is the first public dataset containing a suite of vision tasks built using an autonomous driving platform. This gives it the advantage of being compatible with almost every framework created for Object Detection, Object Segmentation, or Object Tracking for the researchs about autonomous driving.

The car setup to create this dataset contains:

- 2 RGB camera facing forward (10 Hz)
- 2 Grayscale camera facing forward (10 Hz)
- 1 Velodyne Lidar Sensor (10 Hz, 64 laser beams, 100 m range)

The Lidar Coordinate System and the Ranges :

- x-axes : front, y-axes : left, z axes : up
- The point cloud range used for model training [0, -40, -3, 70.4, 40, 1] (xmin ymin zmin xmax ymax zmax) which gives 70.4 meters for front view, 80 meters for side view being left and right and 4 meters for the height range
- The intensity range [0,1]
- Lidar located at 1.73 m from ground
- Lidar data contains x, y, z, intensity informations and its in .bin format

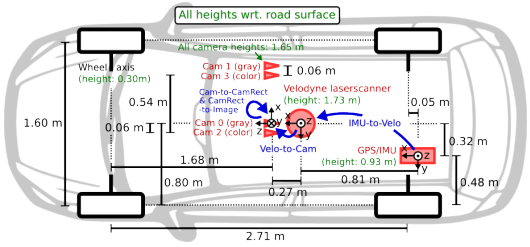

Figure 2.7: KITTI Car Setup (modified and retrieved from [8])

There are different datasets for object detection and object tracking which are slightly in different formats, since a tracking by detection model will be implemented in this project, both datasets are used independently. In the following figure, these two datasets' annotation files are shown:

The dataset contains 8 classes for object detection: Car, Van, Truck, Pedestrian, Person(sitting), Cyclist, Tram, Misc. In this project tracking is performed on Car, Pedestrian and Cyclist classes.

2.3.1 Data Transformation in KITTI

Since the ground truth labels are made in image space and we use Lidar data to perform this project, it is needed to transform the ground truths from image space to lidar space before

#Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

#Values	Name	Description
1	frame	Frame within the sequence where the object appearers
1	track id	Unique tracking id of this object within this sequence
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries. Truncation 2 indicates an ignored object (in particular in the beginning or end of a track) introduced by manual labeling.
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Figure 2.8: Detection (left) and Tracking (right) Annotation File Format (modified and retrieved from [8])

training the object detection model. Even though most of the frameworks do this transformation already embedded in preprocessing step, it is important to have an intuition for understanding the background process.

The calibration file of the KITTI [8] setup contains the following information:

- P0: Left grayscale camera (also called reference camera) projection matrix in the world frame
- P1: Right grayscale projection matrix from the reference camera frame
- P2: Left RGB camera calibration matrix from the reference camera frame
- P3: Right RGB camera calibration matrix from the reference camera frame
- R0: The rectifying rotation for reference coordinate (rectification makes images of multiple cameras lie on the same plan).
- Tr velo to cam : Point clouds frame to reference camera frame rigid body transformation (translation + rotation) matrix

Considering that the ground truth labels are located in reference camera image plane, the different transformations are performed as follows:

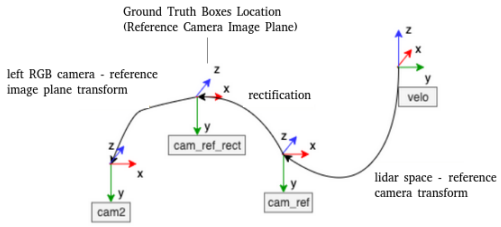

Figure 2.9: KITTI Transformation Map (modified and retrieved from [8])

Bounding boxes to left RGB camera images

proj_mat = P2
boxes_in_image = proj_mat * gt_boxes

Bounding boxes in Lidar Plane

R_inv = np.linalg.inv(R0)
Tr_inv = np.linalg.inv(Tr_velo_to_cam)
proj_mat = R_inv * Tr_inv
boxes_in_lidar = proj_mat * gt_boxes

Lidar Point in Image Plane

proj_mat = P2 * R0 * Tr_velo_to_cam
lidarpoints_in_image = proj_mat * lidar_points

This transform is mostly used for eliminating the lidar space points that stay out of the image boundaries since the lidar space is far wider than the image plane which makes the model obtain a lot of points doesn't contain any ground truth boxes even if some objects of an interested class exist. To train a model without applying this step may cause a huge decrease in performance on the detection model since not labeling the objects is a huge error for object detection model training.
Figure 2.10 shows 3D bounding boxes projected in lidar space.

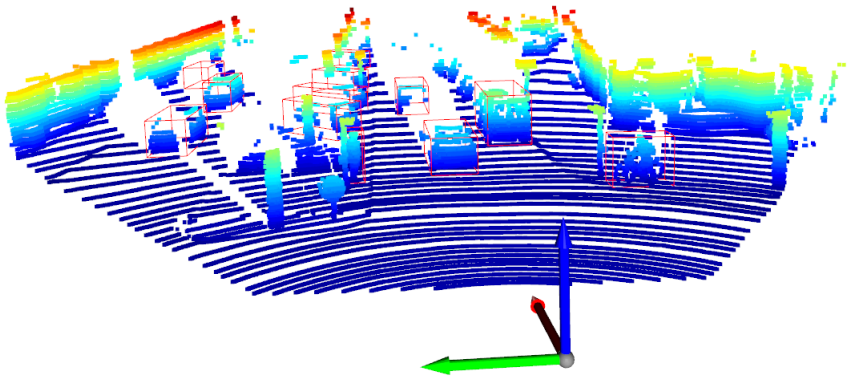

Figure 2.10: 3D Bounding Boxes Projected to Lidar Space (modified and retrieved from [8])

The main advantage of the KITTI dataset over others is that a lot of examples are reachable and most of the frameworks have the support for its format which makes it easier to work with. But also its main disadvantage is that the ground truths are labeled in image space, therefore only the front view can be used for training. This doesn't cause a problem for side views but when the back view is added for the test, it is seen that the results get pretty bad. So the Lidar space range is limited by only the front view.

Chapter 3

State of the art

3D multi-object tracking is essential for autonomous driving. It aims to estimate the location, orientation, and scale of all the objects in the environment over time. This chapter provides the information about related works, state of the art object detection models with Lidar used for this work's detection part, current tracking by detection model created for this project and its components.

3.1 Related Work

The Multi-Object Tracking problem can be considered as a data association problem for detections between the frames in a sequence. To handle this association problem, trackers use various methods for modeling the motion like Kalman [2, 5, 23, 24, 28], Optical Flow, Bounding Box Regression [1], for modelling the similarity like IOU score [2, 23, 28] Mahalanobis distance, and appearance of objects in the scene. Some papers like [24], combine more than one metric to calculate the similarity like the appearance of the object and Mahalanobis distance at the same time. Usually the tracking by detection category architectures use a matching algorithm after calculating the similarity between detections and trackers like Hungarian [2, 23, 24, 28] or Greedy Algorithm [5, 27]. On the other hand, some architectures [1] don't even use a matching algorithm and they pass the estimated tracker with its ID and consider that it is the true place of the tracker in the next frame. They only use a method like the Siamese network to re-identificate a deactivated (if the detection is lost for a while because of the occlusion and the tracker is deactivated but not deleted permanently) tracker.

Being all these methods of tracking by detection category, there are also various methods [13, 29, 30] using joint detection and tracking which means the mechanism has an end to end, unified detection and tracking model.

3.2 Object Detection using Lidar

In this section, 4 main model architectures for processing Lidar data are explained.

3.2.1 Pointnet - 2016

A major breakthrough in recognition and detection tasks on images was due to moving from hand-crafted features to machine-learned features. PointNet [19], the first, an end-to-end deep neural network that learns point-wise features directly from point clouds brought the same breakthrough for point cloud data and after this new architecture, various types of models are proposed to perform object classification, detection and segmentation tasks on the point cloud.

Challenges for processing point clouds :

- The lidar data is unstructured so the model should be permutation invariant (if you have n points you have n! permutations of ordering and processing this data)
- Interaction of points (the relation between neighbor points (local features) and general relation (global features) should be extracted. For classification, the global features are important since the task is to decide for a global class for the whole point cloud. For segmentation, a combination of local and global knowledge is required
- The model should be transform (translation - rotation) invariant (A chair is still a chair when its rotated 90 degrees etc)

Input and Feature Transform

Both input transform and feature transform uses T-Net which is an independent neural network consisting of 3 convolutionals, 1 max-pooling and 2 fully connected layers. They are trained to estimate different transform matrices to be applied by matrix multiplication. Input transform is applied to the first input and the feature transform to the first feature map comes after the input layer. This is the solution to the transform invariant challenge that this paper proposes.

MLP (Multi Layer Perceptron)

Same fully connected layer is applied to each of the n points one by one and the outputs are concatenated.

Max Pooling

The second MLP layer gives n x 1024 output and a maximum pooling layer is applied to it. In this way, the global features are obtained without depending on the order of the points. This is

the solution to the unstructured data challenge that this paper proposes. These global features directly go to classify the point cloud.

Local and global features are concatenated for the segmentation part as an extension of the architecture.

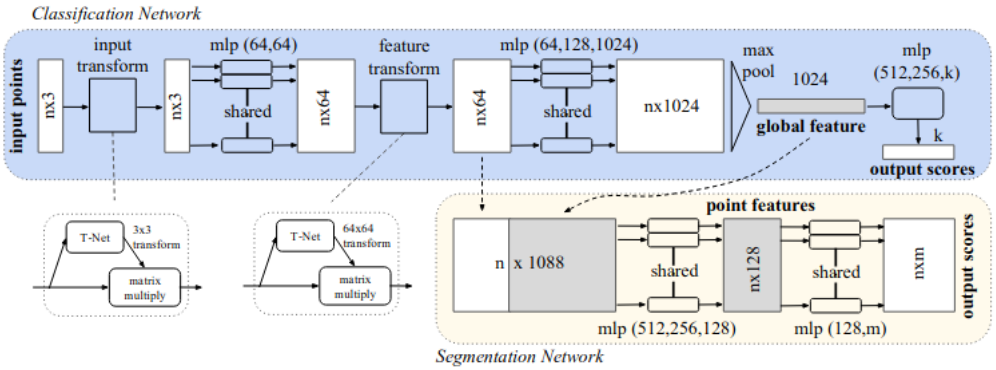

Figure 3.1: Pointnet Architecture (modified and retrieved from [19])

PointNet architecture does not allow to do Object Detection alone but after this paper, various methods to realize object detection on point cloud data has started to come up.

3.2.2 Voxelnet - 2017

Voxelnet is the first paper that performs end to end 3d object detection processing Point Cloud Data using PointNet architecture inside.

PointNet is for processing pretty small point cloud data having about 1000 points referring to a simple object like a chair table etc. If the subject is Autonomous Driving, around 100k points are found in an environment with different objects to detect like Cars, Pedestrians, Cyclists, Van, Truck, Trees, Buildings, etc. This fact makes PointNet method requires too much computational cost and slow speed. In this sense, Voxelnet comes with the idea of dividing the whole Point Cloud in voxels and process them one by one.
It consists of 3 main parts:

- 3D Data to Voxel Partition - Feature Learning Network (The network to process the points in voxels similarly to the Pointnet architecture)
- Convolutional Middle Layers (3D CNN to extract the features)
- Region Proposal Network (Last layer to obtain the detection results)

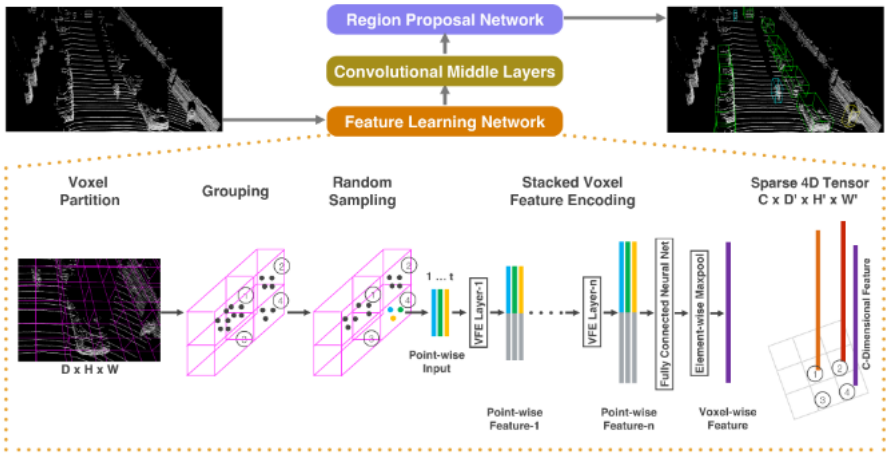

Figure 3.2: Pointnet Architecture (modified and retrieved from [31])

Voxel Partition

Voxel [31] depth, height, and width referring to the z y x-axes are determined. A grid size having D = point cloud z range / voxel depth, H = point cloud y range / voxel height, W = point cloud w range / voxel width is calculated. Being T is the maximum number of points in a voxel, all the points are selected randomly one by one and placed in the nearest voxel. If a voxel is already full, the selected point is ignored. (Random Sampling)

Feature Learning Network

The Feature Learning consists of 2 VFE (Voxelnet Feature Encoding) layers which are identical structures (except for their size being VFE-1(7, 32) and VFE-2(32, 128)) applied in a row. In this sense, one point in a voxel has the following structure :

$$Vin = \{pi = [x_i, y_i, z_i, r_i, x_i - v_x, y_i - v_y, z_i - v_z]T\epsilon R^7\}i = 1...t \tag{3.1}$$

Which explains that every input point in a voxel has 7 features being x,y,z coordinates, reflectance (or intensity), the distance x to the mean point of the voxel, the distance y to the mean point of the voxel, the distance z to the mean point of the voxel
Instead of having n x 3 points like in PointNet, 7 x n points are given as input to the shared Fully Connected Layer and a 16 x n matrix is obtained as output.
Keeping a copy of this output aside as local features, max pooling is applied to obtain global features in 32 x 1 size. These global features are concatenated with local features and the 32 x

n output feature map is passed to the next VFE layer.

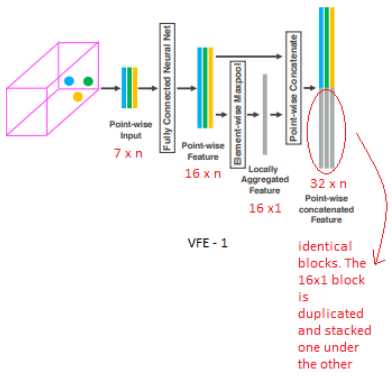

Figure 3.3: Voxelnet Feature Encoding (modified and retrieved from [31])

The same logic is applied this time for a shared Fully Connected Layer and obtained 64 x n matrix is obtained as output. Keeping a copy of this output aside as local features, max pooling is applied to obtain global features in 64 x 1 size. These global features are concatenated with local features and a last maximum pooling is applied to the 64 x n output feature map to obtain the final output of the Feature Learning Network for one voxel.

This output is a 128x1 vector and all the outputs that come from voxels are concatenated through the same locations they come from. This gives us a 128 x D x W x H size 4D tensor. (128 x Grid Size output tensor)

In the paper, following sizes are used for KITTI Dataset:

- Point Cloud Range z: [3, 1] y: [40, 40] x: [0, 70.4]
- Voxel Size vD = 0.4, vH = 0.2, vW = 0.2 meters.
- Grid Size D = 4 / 0.4 = 10 , W = 80 / 0.2 = 400 , H = 70.4 / 0.2 = 352

Which gives 10x400x352 = 1408000 voxels to send the Feature Learning Layer 1 by 1 and a $128 \times 10 \times 400 \times 352$ 4D tensor is obtained. This 4D tensor is then passed to the 3D CNN.

Convolutional Middle Layers

This step consists of 3 convolutional layers to extract features in 3D.
Conv3D(128, 64, 3,(2,1,1), (1,1,1))
Conv3D(64, 64, 3, (1,1,1), (0,1,1))
Conv3D(64, 64, 3, (2,1,1), (1,1,1))

with respect to the following syntax **Conv3D(cin, cout, k, s, p) cin :** input channel size **cout :** output channel size **k :** kernel size **s :** stride **p :** padding. Being ($64 \times 2 \times 400 \times 352$) is the output 4D tensor size, it is then reshaped to (128 x 400 x 352) to be able to apply 2D convolutions and obtain detection results.

The difference between a 3D and 2D convolution is that the kernel does not move only in 2 direction but 3. In 2D convolution, a kernel with different channels which conforms a 3D kernel again, travels a 3D input from left to right and from top to bottom while in 3D convolution same shape of the kernel travels in depth dimension too.

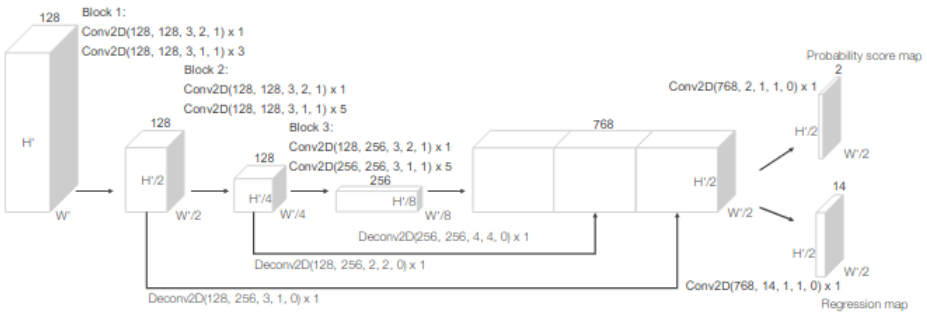

Figure 3.4: Region Proposal Network (modified and retrieved from [31])

Region Proposal Network

Region Proposal Network, first proposed in [20] is in general refers to a neural network used to predict different region proposals (ROI boxes) with objectness scores without any classifying. In Voxelnet [31], a type of RPN is created to produce final detections using the features extracted in Convolutional Middle Layers.

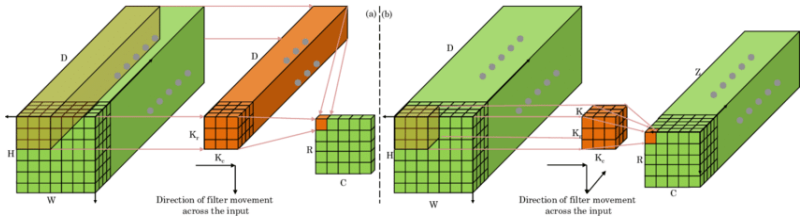

Figure 3.5: 2D vs 3D Convolution (modified and retrieved from [18])

3.2.3 SECOND - 2018

The model architecture is very similar to the Voxelnet [31] with only some little changes which bring improvements to the model speed. The main change is to use 3D Sparse Convolutional Neural Networks instead of Voxelnet's 3D CNN. The logic behind the Sparse CNN is to use only non-zero data in feature maps and apply convolution only on these parts which is very useful for Lidar data having a very sparse structure.

Sparse Convolutional Networks

Sparse CNNs, first proposed in [9] can be thought of as an extension of the idea of sparse matrices. If a large matrix only has a small number of non-zero entries per row and per column, then it makes sense to use a special data structure to store the non-zero entries and their locations; this can both dramatically reduce memory requirements and speed up operations such as matrix multiplication. To forward propagate the network it is calculated two matrices for each layer of the network:

- A feature matrix which a list of row vectors, one for the ground state, and one for each active spatial location in the layer. The width of the matrix is the number of features per spatial location.
- A pointer matrix with a size equal to the spatial size of the convolutional layer. For each spatial location in the convolutional layer, the number of the corresponding row in the feature matrix is stored.

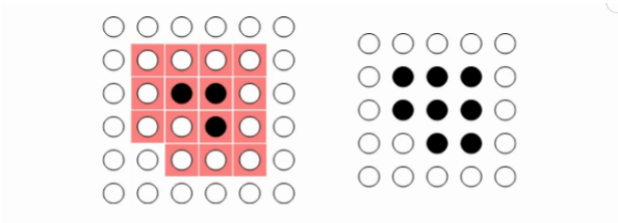

Figure 3.6: Sparse CNN (modified and retrieved from [9])

It is seen that the convolution is applied only to the areas having active points (non-zero data) and as the output all this area become active points. All the other parts staying outside the active area just passed to the output without any process. Since Point Clouds are almost %90 sparse data, to apply Sparse Convolutional Networks is very important and effective to improve

the speed and even increase the effectiveness of feature extraction part since the unnecessary points are not involved.

Submanifold Sparse Convolutional Networks

In this version [10], the output feature map is the same size as the input feature map and an output point becomes active only if it was active in the input data map too. This is also called VSC (valid sparse convolution).

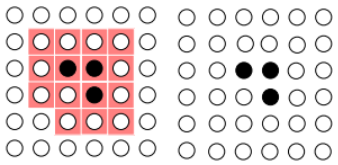

Figure 3.7: Submanifold Sparse CNN (modified and retrieved from [10])

The reason for this change is, that for deep neural networks, the sparsity disappears so much that the performance enhancement of SC disappears too. So protecting the sparsity as it was in the input feature map may help for especially creating sparse deep neural networks. There are 2 different implementations of the paper called as Second (using Submanifold Sparse CNN), and Second Small (using Sparse CNN).

3.2.4 Pointpillar - 2018

Pointpillar is a model for object detection in 3D that enables end-to-end learning with only 2D convolutional layers. Instead of dividing the Point Cloud to 3D Voxels, it divides them to the Pillars (columns). In other words, the grid created on the Point Cloud is not 3D but 2D having only width and height but not depth since the space doesn't get divided in z direction.

One point in one pillar has the following structure with 9 features:

$$Pin = pi = [x_i, y_i, z_i, r_i, x_c, y_c, z_c, x_p, y_p]T\epsilon R^7 i = 1...t \tag{3.2}$$

x, y, z and reflectance r, c subscript denotes distance to the arithmetic mean of all points in the pillar and the p subscript denotes the offset from the pillar x, y center. Therefore, with this structure of not having D in grid, it is obtained CxWxH output 3D tensor and it is therefore possible to apply directly 2D CNN instead of 3D to extract high resolution features.

The model contains similarly to the Voxelnet and Second 3 main steps:

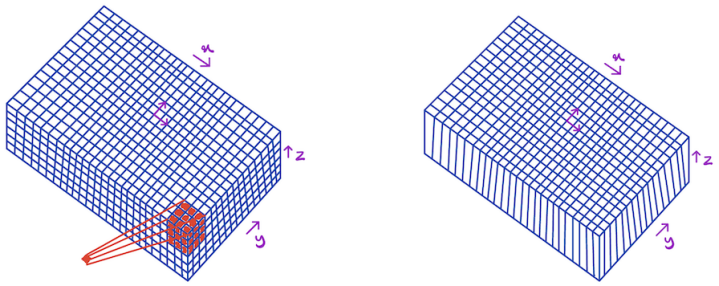

Figure 3.8: Voxel vs Pointpillar (modified and retrieved from [14])

- 3D data to Pillar partition - Pillar Feature Network
- Convolutional middle layers (2D CNN to extract the features)
- SSD (Last layer to obtain detection results)

It is seen that another change the paper does is to use SSD instead of RPN as the object detection head like Voxelnet and Second.

SSD Architecture

While RPN was a two-stage detector, SSD [15] is a one-stage detector. The feature map is used to generate the region proposals using anchors with the same logic as RPN and classification is applied directly to these regions for c classes without having any additional objectness score calculation step.

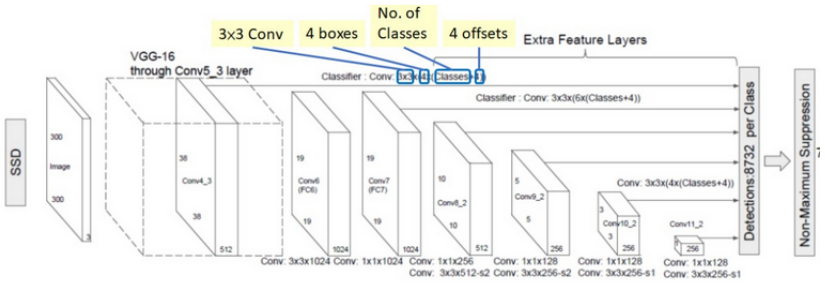

Figure 3.9: SSD Architecture,(modified and retrieved from [15])

3.2.5 Model Selection

All these model architectures have been used to train an object detection model for Car, Pedestrian and Cyclist classes using KITTI object detection dataset.

	Car AP@0.70	Car AP@0.50	Pedestrian AP@0.50	Pedestrian AP@0.25	Cyclist AP@0.50	Cyclist AP@0.25	Inference Time
VoxelNet	85.82 76.03 75.10	92.45 86.70 86 50	52.05 45.42 42.02	72.53 68.20 65.10	78.64 63.79 60.17	83.32 72.80 68.90	0.069 sec per frame
SECOND	87.66 77.42 76.05	94.40 89.74 89.12	52.21 46.77 43.10	75.16 70.20 67.79	80.91 66.10 62.15	86.38 72.91 69.30	0.033 sec per frame
SECOND MultiHead	87.58 77.61 76.00	90.76 89.93 89.18	53.85 48.72 45.07	75.43 72.33 68.62	80.22 68.14 63.17	86.26 78.36 74.48	0.038 sec per frame
Pointpillar	84.19 75.33 71.45	90.64 89.54 88.82	51.89 45.63 41.60	69.20 65.33 61.84	76.30 60.88 57.89	81.57 72.82 68.44	0.024 sec per frame

Table 3.1: Object Detection Models Comparison

The table 3.2.5 shows the accuracy performance of different models with different threshold (i.e @50 means the IOU threshold explained in section **??**, is %50) and for 3 difficulty levels (Easy Medium Hard) for 3 classes Car, Pedestrian and Cyclist.

Usually, one-stage detectors are known to have a better performance on speed but worse accuracy in comparison with a multi-stage detector. It is seen in the table that Pointpillar has a better speed for inference but the accuracy is different between the architectures with RPN like Voxelnet and SECOND is not negligable.

While default architectures apply region proposal layer 1 by 1 separately for each class (which is the reason that they don't apply any image classification after this layer as in original paper [20], since the objectness score gives if there is an object in the proposed region or not and if there is, the class is already known) multi head add parallel region proposal layers to perform at the same time for each class in parallel.

It is seen that the SECOND multihead has near accuracy results to default SECOND architecture but for more sensitive classes like pedestrians and cyclists, its results are quite better without any big change in speed which makes multihead SECOND architecture the optimal model. In conclusion, SECOND Multihead is chosen as the main object detection model to continue with the tracking task. Figure 3.10 shows the evaluation of the SECOND model for 3 classes on 3 difficulty level.

The figures 3.11 3.12 and 3.13 show some example inference outputs with IOU threshold 0.50 for each class.

3.3 Tracking by Detection Model Architecture

As mentioned before, in this work, having [5, 23] as the two base architectures, a tracking by detection architecture is implemented. Since these two methods are using a similar approach (constant velocity Kalman Filter) for motion estimation and only their similarity metrics and matching algorithms are different, after the base of the architecture is implemented, these different intermediate level changes are applied to compare the results.

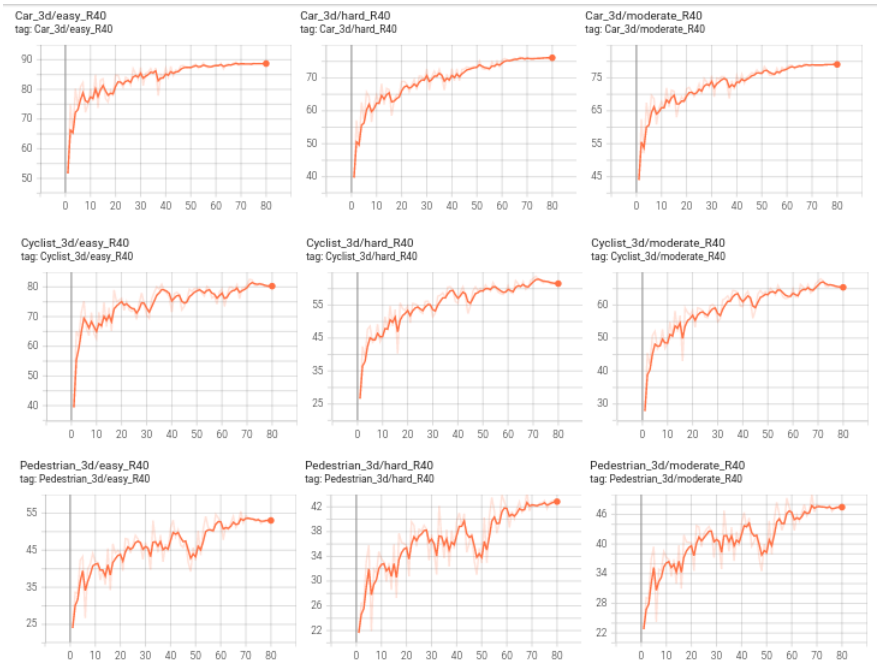

Figure 3.10: SECOND Multihead Model Evaluation Graphs

After choosing the optimal methods between these similarity and matching algorithms, the final model is optimized according to the analyzed weak points.

3.3.1 Flow Chart of Tracking Mechanism

The first model is created with almost the same way of [5] using Kalman Filter as motion estimation step, Mahalanobis Distance as similarity metric and Greedy Algorithm for matching. *hits* variable is used to count how many times the tracker got a match in its total life cycle, *time_since_last_update* variable is used to count when the last time the tracker got a match and *max_age* is the threshold to determine how long a tracker can be alive without having a match. The main logic of the architecture is as follows:

1. Using SECOND Multihead, detect Car, Pedestrian, Cyclist objects in the first frame. Initialize a new tracker for each detection with the same location and unique ID. A newly created tracker will have 0 for velocity variables (dx, dy, dz, da) in the Kalman State x, since the velocity can't be calculated only 1 image. The location of the detection box is passed to the state x without any change.

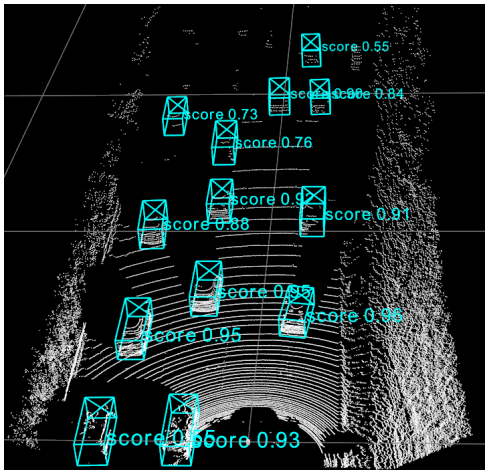

Figure 3.11: Example Inference Output with Multiple Car Objects Detected

2. Predict these tracker's locations in the next frame with Kalman Prediction step. Increase "time_since_last_update" variable by 1. The tracker's next location is calculated using a constant velocity model:

   $x_{t+1} = x_t + dx_t$
   $y_{t+1} = y_t + dy_t$
   $z_{t+1} = z_t + dz_t$
   $a_{t+1} = a_t + da_t$
   $w_{t+1} = \mathrm{w}$
   $h_{t+1} = \mathrm{h}$
   $l_{t+1} = \mathrm{l}$
   $dx_{t+1} = dx_t$
   $dy_{t+1} = dy_t$
   $dz_{t+1} = dz_t$
   $da_{t+1} = da_t$

3. Calculate the Mahalanobis Distance between all the tracker objects and detections in the second frame.

4. Using the Greedy Algorithm, match the detections and trackers according to their distance. Lower distance means a better match.

5. Update the trackers (Kalman update step) according to their matched detections' locations. Therefore the final equation of a tracker's next frame state becomes as follows:

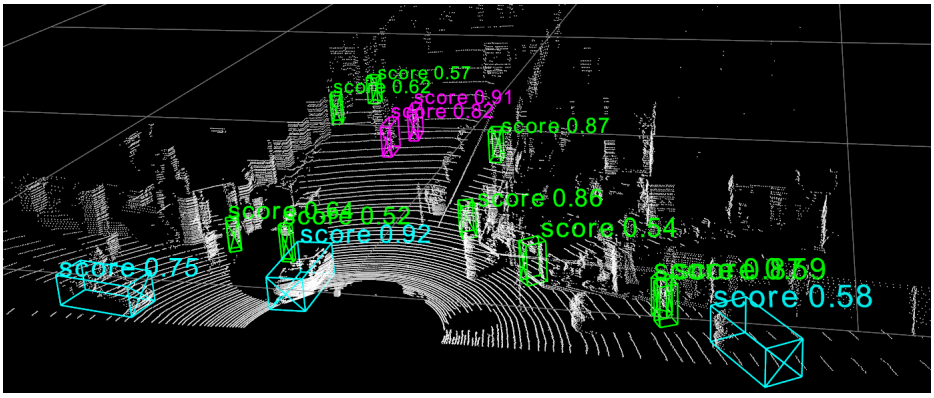

Figure 3.12: Example Inference Output with Multiple Car (blue), Pedestrian (green) and Cyclist (pink) Objects Detected

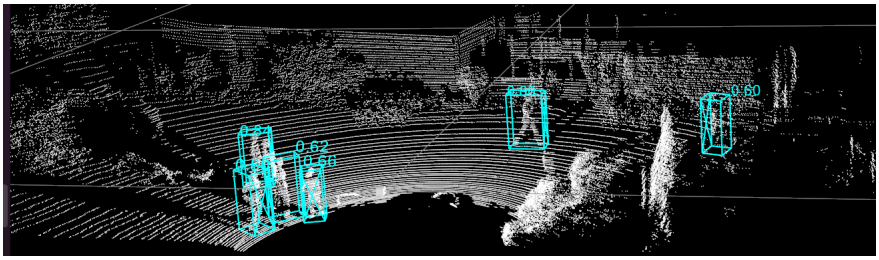

Figure 3.13: Example Inference Output with Multiple Pedestrian Objects Detected

$$x_{t+1} = x_t + dx_t + qx_t$$
$$y_{t+1} = y_t + dy_t + qy_t$$
$$z_{t+1} = z_t + dz_t + qz_t$$
$$a_{t+1} = a_t + da_t + qa_t$$
$$w_{t+1} = \mathrm{w}$$
$$h_{t+1} = \mathrm{h}$$
$$l_{t+1} = \mathrm{l}$$
$$dx_{t+1} = dx_t + qdx_t$$
$$dy_{t+1} = dy_t + qdy_t$$
$$dz_{t+1} = dz_t + qdz_t$$
$$da_{t+1} = da_t + qda_t$$

Where the unknown linear and angular acceleration are modeled as random variables (qxt , qyt , qzt , qat) and (qdxt , qdyt , qdzt , qdat) that follow a Gaussian distribution

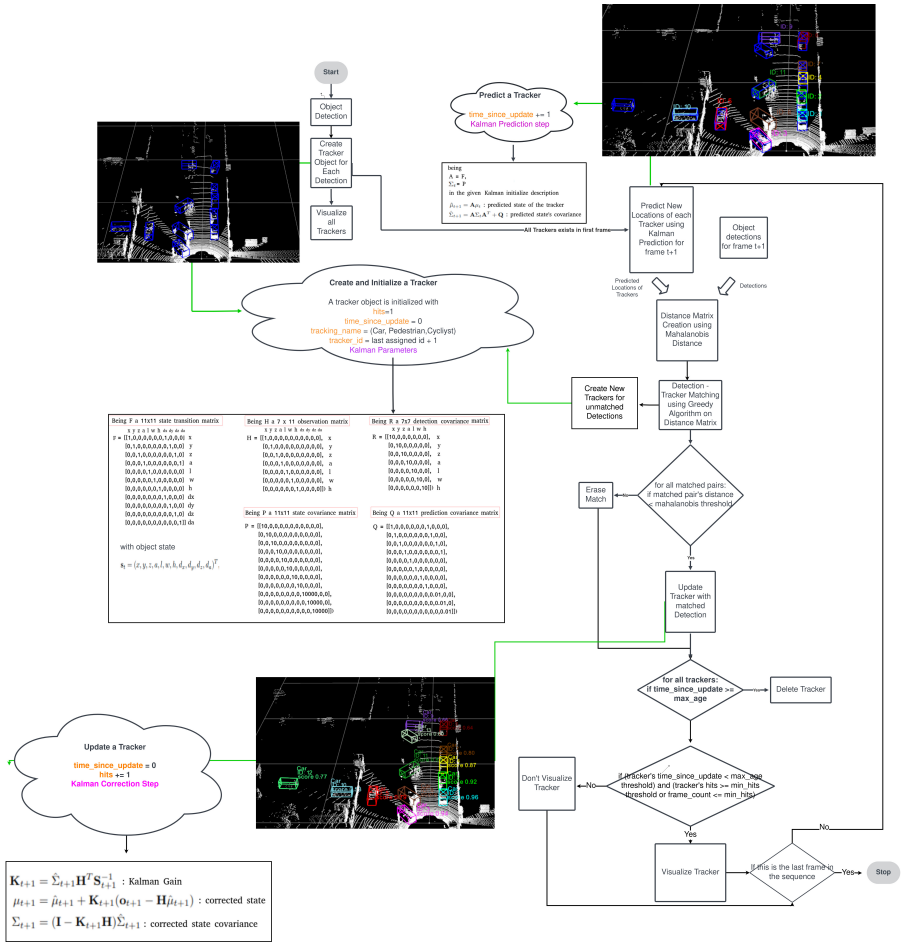

Figure 3.14: Flowchart of the first model implementation

with zero mean and covariance Q. The linear and angular velocities (dxt, dyt dzt, dat) are calculated by the difference between the x y z a, values at frame t+1 and t.

For matched trackers, the hits counter is augmented by 1 and *time_since_last_update* is reset to 0 which prolongs the time of the tracker's life.

6. For trackers not having a match, don't apply the Kalman Update mechanism, just leave the parameters as in the prediction step.

7. After this step, the creation or deletion of a tracker has some differences than the first frame as explained below:

   - If the current frames are among the first 5 frames, visualize the trackers. If not,

check if the tracker has more hits than the *min_hit* threshold to verify if the tracker is a reliable one.

- Delete the trackers having *time_since_last_update* > *max_age*, so the ones didn't get any match longer than the determined threshold.
- Create a new tracker if a detection in frame t stayed unmatched with previous trackers. Visualize it without any condition if the current frames are among the first 5 frames otherwise keep it in the background, just as a record of the program and visualize if and only if it has more hits counted than *min_hit* threshold

3.3.2 Kalman Filter

The Kalman Filter [11] is one of the most important and common estimation algorithms consisting of 3 main steps as Prediction, Measurement and Update. Having various fields of usage like location and navigation systems, control systems, computer graphics and much more, the usage of Kalman Filter for tracking systems will be covered in this part.

Prediction

Having the prior knowledge of the tracker locations at frame t x_t and the state covariance at frame t (p_t), the locations of the trackers at the frame t+1 (x_{t+1}) are estimated with state covariance at frame t+1 (p_{t+1}). In AB3DMOT [23] paper, the state of the tracker consists of 10 variable as follows:

x_t = [x, y, z, theta, l, w, h, dx, dy, dz] being
x = center point of the box in x axis
y = center point of the box in y axis
z = center point of the box in z axis
a = the angle between the object's facing direction and the x-axis
l = length of the box
w = width of the box
h = height of the box

dx, dy, dz = represent the difference of (x, y, z) between the current and the previous frame.
The state covariance p_t is the uncertainty of the state calculated in x_t.

Measurement

Measurement refers to the detections coming from SECOND model for the frame t+1. In our case, these detections are 3D box locations in a 1x7 vector form:
detection = [x, y, z, a, l, w, h] being

x = center point of the box in x axis
y = center point of the box in y axis
z = center point of the box in z axis
a = the angle between the object's facing direction and the x-axis
l = length of the box
w = width of the box

h = height of the box

Using the state prediction, state covariance prediction and measurement covariance, the measurement prediction (o'_{t+1}) and the uncertainty of the predicted measurements (S_{t+1}) are calculated. Figure 3.15 shows the correct meanings of a 3D box dimensions.

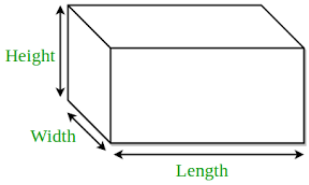

Figure 3.15: Width, length and height of a 3D box

Update

The last step is to update the state covariance and predicted states of the trackers using the detections to obtain the final states of the trackers at frame t+1 (x_{t+1} and p_{t+1}). For this step the Kalman Gain K_{t+1} is calculated using the uncertainty of the predicted measurements and the predicted state covariance The obtained x_{t+1} and p_{t+1} then use as prior knowledge of the current frame to calculate the next frame states until all the frames in the sequence are processed.

The logic and the usage of the Kalman Filter in [5] are pretty similar to AB3DMOT's Kalman Filter implementation. The only difference is that the state vector is defined with 1 more parameter da, being the difference of the angle a between the current and the previous frame. Therefore the state vector x is an 11x1, state covariance matrix P is an 11x11, state transition matrix F is an 11x11 and observation matrix H is a 7x11 matrix instead of 10x1, 10x10, 10x10, 7x10 as explained above and in [23].

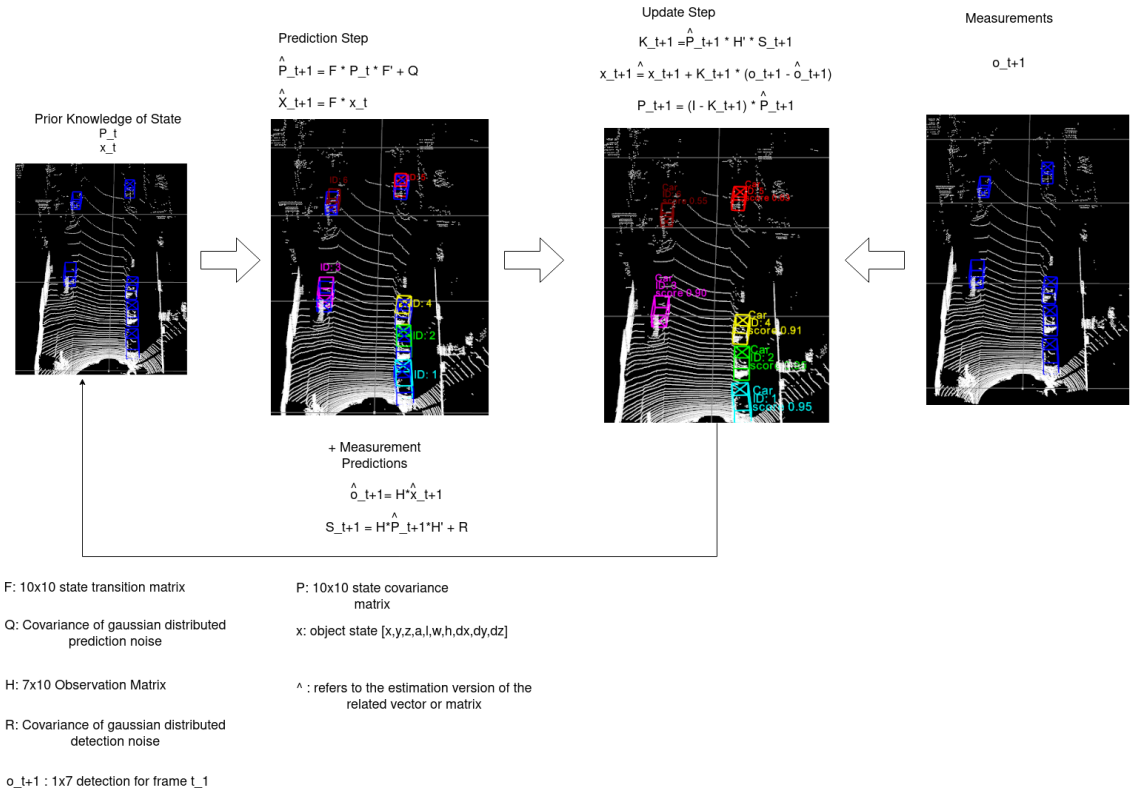

Figure 3.16: Complete Flowchart of Kalman Filter for Tracking

3.3.3 IOU Metric

To perform the update step of Kalman, it is needed to find which Kalman predicted tracker location belongs to which detection for frame t+1 as we can see from the formula of x t+1 calculation. Intersection over Union (IoU) is a very common metric used usually to calculate the accuracy of detection over its ground truth. It is a number from 0 to 1 that specifies the amount of overlap between the predicted and ground truth bounding box. The same logic is used for calculating the overlap between all the detections and the estimated tracker locations to create a distance matrix obtaining the IOU score between all the detections and trackers. This distance matrix then will be used to pick matches.

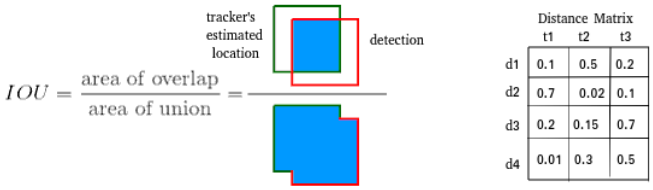

Figure 3.17: IOU and Distance Matrix

3.3.4 Mahalanobis Distance

Mahalonobis distance is the distance between a point and a distribution. And not between two distinct points. It is a multivariate equivalent of the Euclidean distance. In this task, it is used to calculate the disttance between the detection and predicted tracker location instead of IOU in previous model. The distance matrix is created again between each detection and tracker to be applied an matching algorithm on. In contrary from IOU, the lower the distance, the better.

Mahalanobis Distance m:

$$m = \sqrt{(\mathbf{o}_{t+1} - \mathbf{H}\hat{\mu}_{t+1})^T \mathbf{S}_{t+1}{}^{-1} (\mathbf{o}_{t+1} - \mathbf{H}\hat{\mu}_{t+1})}.$$

$\mathbf{o}_{t+1}$: detections

$\hat{\mathbf{o}}_{t+1} = \mathbf{H}\hat{\mu}_{t+1}$: kalman predictions

$\mathbf{S}_{t+1} = \mathbf{H}\hat{\Sigma}_{t+1}\mathbf{H}^T + \mathbf{R}$: uncertanty of the predicted object detection

Figure 3.18: Manahalobis Distance

3.3.5 Greedy Match Algorithm

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result. The algorithm never reverses the earlier decision even if the choice is wrong. The logic of the algorithm is as follows:

1. All the detections are randomly selected and paired with the tracker having lowest distance.
2. Repeat the same step for 2. lowest, 3. lowest, ... n. lowest options for all the detections.
3. Travel the pair list and accept the detection - tracker pair as a match if neither the detection nor the tracker has been selected before.

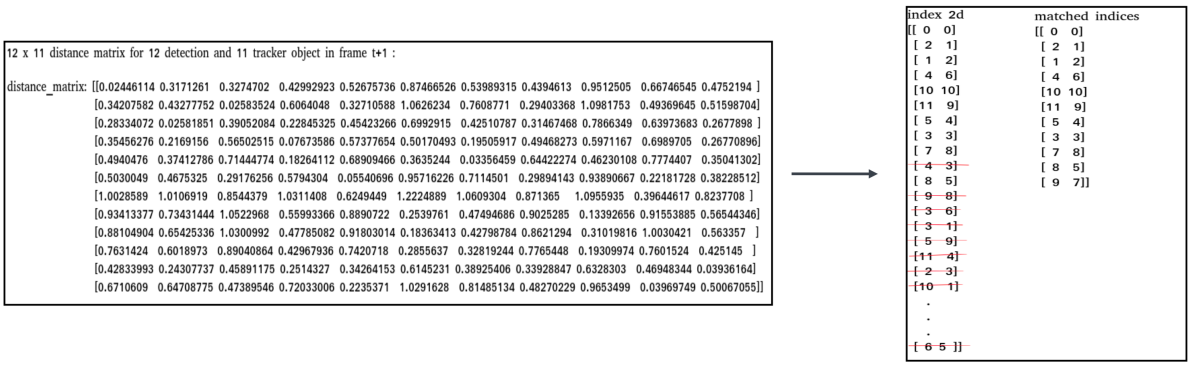

Figure 3.19: Greedy Match Algorithm

Weak Point of Greedy Match

Since the algorithm is based on finding the optimal solution for the detections selected in random order, a tracker may be matched with a detection since its the best choice for the detection, but not for the tracker if it has a lower distance with another detection. In that case, the error is not fixed and the tracker is unable to unmatch and rematch with its best choice.

3.3.6 Hungarian Algorithm

The Hungarian Algorithm is used to find the minimum cost in assignment problems. Using this algorithm, we can find the matches between detection and trackers by working on distance

matrix. Since the bigger the IOU score, the better for this matching case, the distance matrix is updated by 1-IOU_score to find pairs giving globally minimum cost. The logic of the Hungarian Algorithm is as follows:

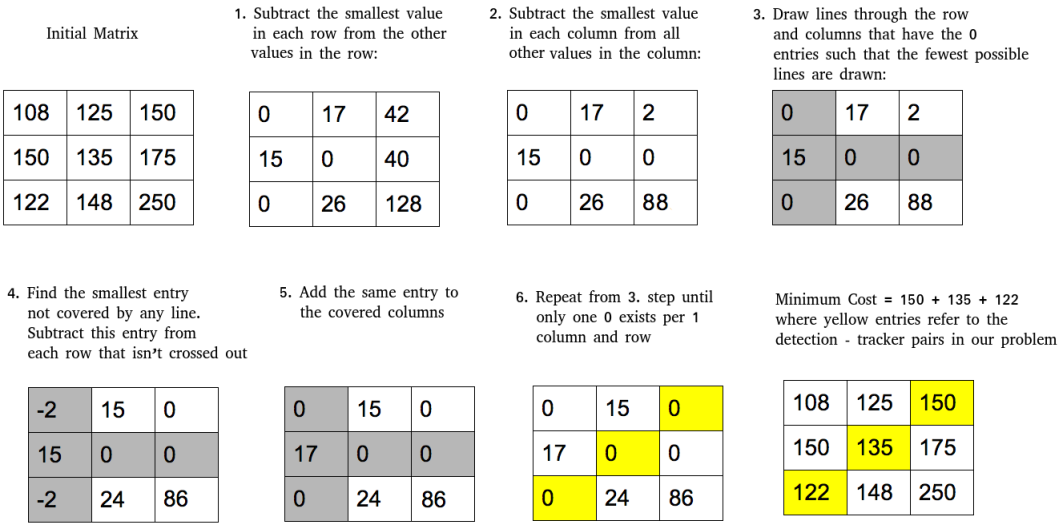

Figure 3.20: Hungarian Match Algorithm

Weak Point of Hungarian Match

Since the goal of the algorithm is to find the global minimum cost, not every time the best column entry is paired with each row. Which means sometimes even if the global cost is the minimum, not every detection gets paired with the optimal choice of tracker as we can see that in the first row, although the best choice is the first row (108), the last column (150) is chosen.

Chapter 4

Contributions

In this chapter, the analyses and optimizations made after implementing the base tracking mechanism are explained. For each analysis, an optimization is proposed. Additionally, some challenges that contain more complex problems and the proposed solutions are explained.

4.1 Optimizations

4.1.1 Overlapping Tracking Boxes

There are some false positives occurs in tracking even if it does not exist in detection.

Reason : Low Mahalanobis Threshold

The detection and its potential tracker have the Mahalanobis distance at 0.14541635 which is higher than the default threshold 0.1. Since only the matches lower than this default threshold are accepted, this match is not accepted. If a detection stays unmatched and it's within a few initial frames, a new tracker is created for it. Since we had already a tracker for the same detection and this tracker is not older than the *max_age* limit (5), we see both tracker boxes in the frame.

The figure 4.1 shows an example of this error being the left side is the detections and the right side is tracking results.

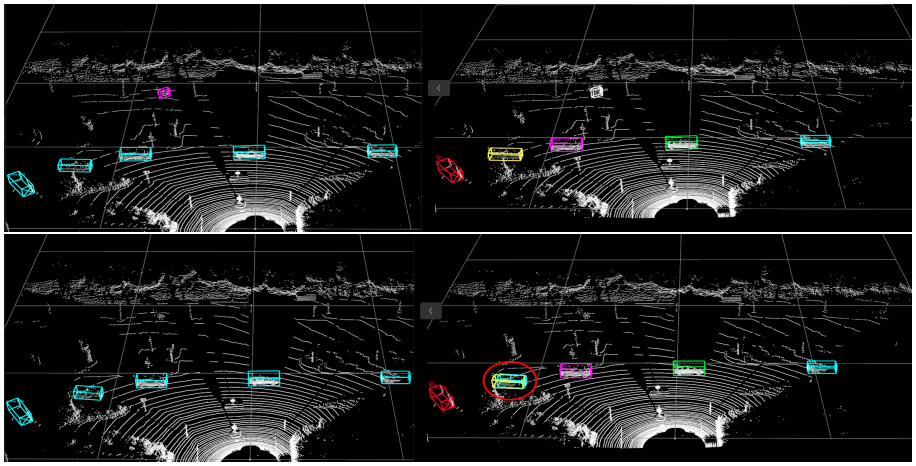

Figure 4.1: Overlapping Tracking Boxes

Optimization:

The threshold has been changed to see different results. The more this threshold is increased, the less we obtain overlap problem since close detections and tracker predictions gets matched without any problem. On the other hand, using a more flexible threshold brings another problem : When a tracker stays unmatched because the detection is lost for that frame, a new possibility of being matched for this tracker with a not so close detection (mostly a false prediction detection) can come up.

The figure 4.2 shows an example of this error being the left side is the detections and the right side is the tracking results.

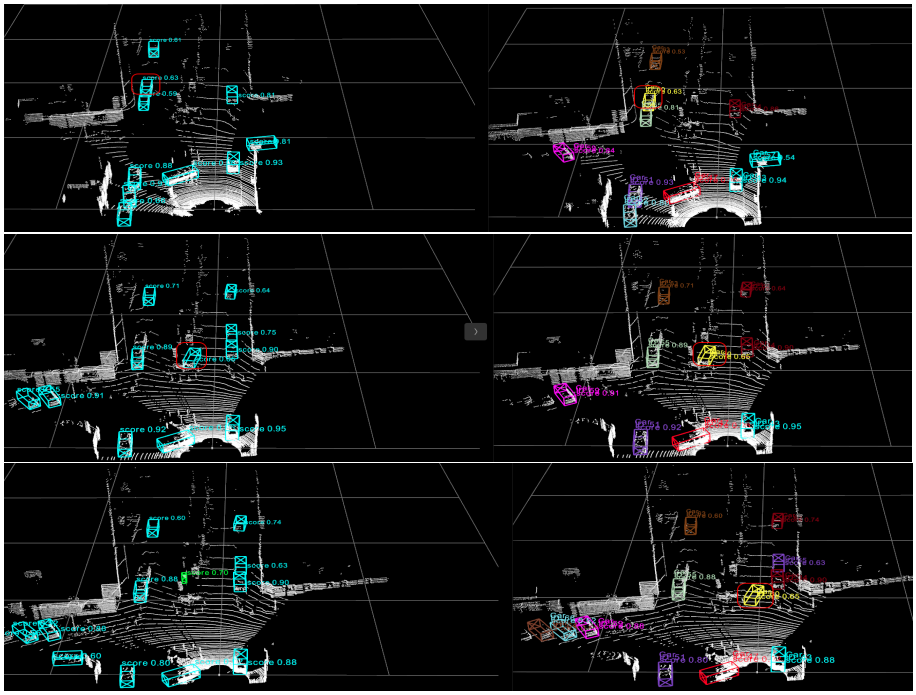

Figure 4.2: Flexible Mahalanobis Threshold

Match Deny Mechanism

After examining the change on trackers' positions, we decided to use the distance between the last frame's position and the current frame's position to decide if the tracker is moving abnormal. Now the algorithm denies the matches coming from greedy algorithm if the distance is higher than 4m. Since the match is denied, the updated tracker is sent to the last location and the state of the tracker is updated for the next Kalman prediction. The threshold is calculated as follows:

- In usual cases we obtain the distance between last and current frame around 1.5.
- Since the Lidar sensor work at 10Hz, we obtain 1 frame per 0.1 second.
- 0.1 second - 1.5m : 3600 second - 54000m
- So if a car moves 1.5 m between the frames, this corresponds to 54km/h which is very reasonable.

- According to my analyses, these false matches has around 11m distance which corresponds to 396km/h speed for a car which is very clearly wrong.
- Instead of putting the threshold so high as near 11, we decided to use more realistic one to catch some other false matches which may still occur with lower than 11m distance and set the threshold to 4.5 which corresponds to 162km/h

As a result, figure 4.3 shows the effect of the match deny mechanism on the same scene.

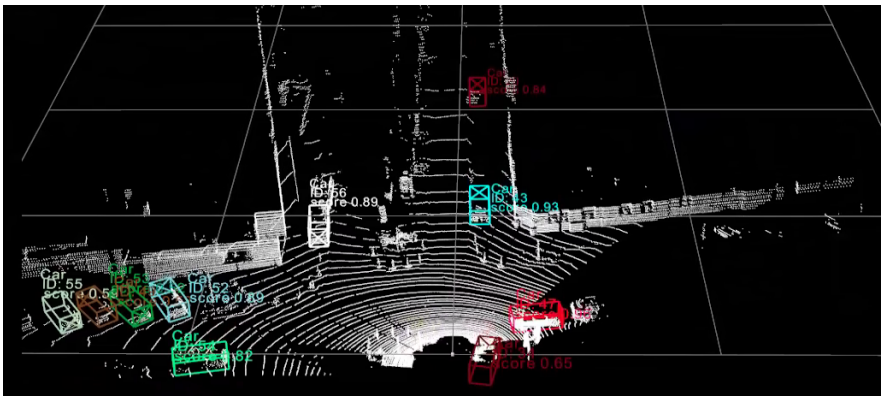

Figure 4.3: Match Deny Mechanism

4.1.2 False Positive on Detection Passes Directly to the Tracking

In general, if a detection does not match with the already existing tracker, the detection is considered a new object, and a new tracker is created directly for this object. But it's needed to decide whether it's a correct object to track or a false positive. For that purpose, instead of visualizing the new tracker directly, we keep "track of trackers" and decide that it's a good track if only the tracker gets match more min_hits (3). The figure 4.4 which belongs to the second frame of a scene shows a false positive detection(left) and obtains a tracker(right).

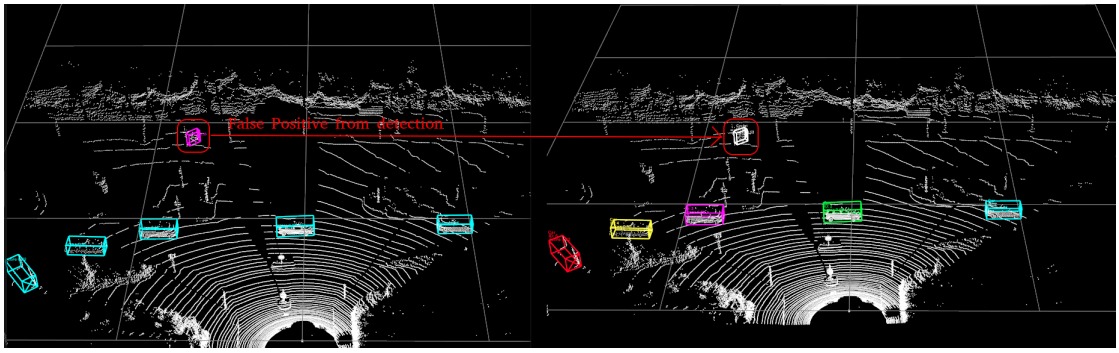

Figure 4.4: False Positive

Reason : The Weak Point of the Starting Frames

The rule explained above is not applied for the initial frames since we can't expect a tracker takes 3 times match in first or second scene and if the rule is applied, the first 3 frames would be totally empty. This makes the initial frames weaker and all the detections passes directly to the tracking results.

Optimization :

Since initial frames are exempt from the general conditions which make the tracking system weaker to allow passing wrong detections to the tracker, I added a threshold for initial frame's detections, especially for the classes more fragile like Cyclist and Pedestrian. Only the detections having a prediction score higher than this threshold (0.65) are accepted to create trackers for initial frames.

4.1.3 Old False Positive Tracker Comes back without Detection

With a frame_count variable, we keep track of which frame we are. For the first frames, we accept the detections directly, to begin with. General conditions to create a new tracker for a new detection are applied if and only if we are beyond of first few initial frames. The figure 4.5 shows that even though there is not any detection for this false cyclist, its false tracker comes back to the scene.

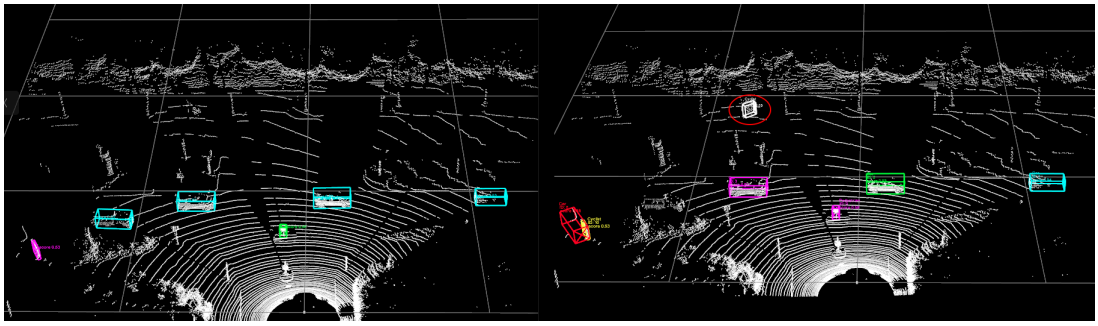

Figure 4.5: Old False Positive

Reason : Different frame count parameter for each class

This frame count is kept individually for each class (which is a bug for me), and even if we are out of frame count limit in general, if we see a cyclist 2nd time in the 100th frame, still this frame count limit is applicable and it lets all the detections or old trackers pass.

Optimization :

The frame count parameter in original code was different for each class. It has been changed to be global.

4.1.4 Tracker Box Comes Bad If the Detection is Lost for a Few Frame

The Kalman prediction without detection update doesn't work very well if the detection is lost while the object is rotating. The figure 4.6 shows an example case.

Reason : Kalman Filter's Constant Velocity

Since it is a constant velocity Kalman Filter, the last speed calculated between the last frame and the previous frame becomes directly the new dx dy dz da for the next frame Kalman prediction. This logic works well if the object does not change its orientation significantly since the last frame or its detection is matched. Otherwise the box gets pretty incorrect.

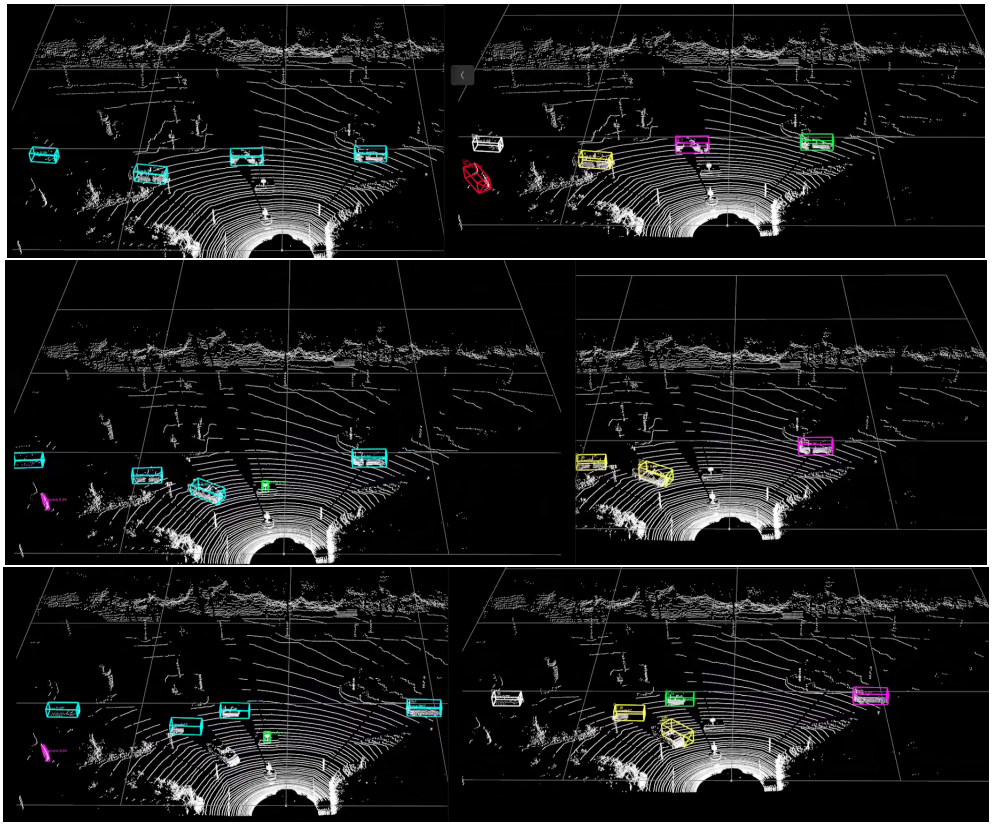

Figure 4.6: Bad Predicted Tracker Boxes

Optimization :

Since the main problem is about prediction uncertainty, the prediction noise is augmented for velocity parameters in matrix Q by replacing 0.01 (default initialization) to 1. The figure 4.7 shows the effect of this change.

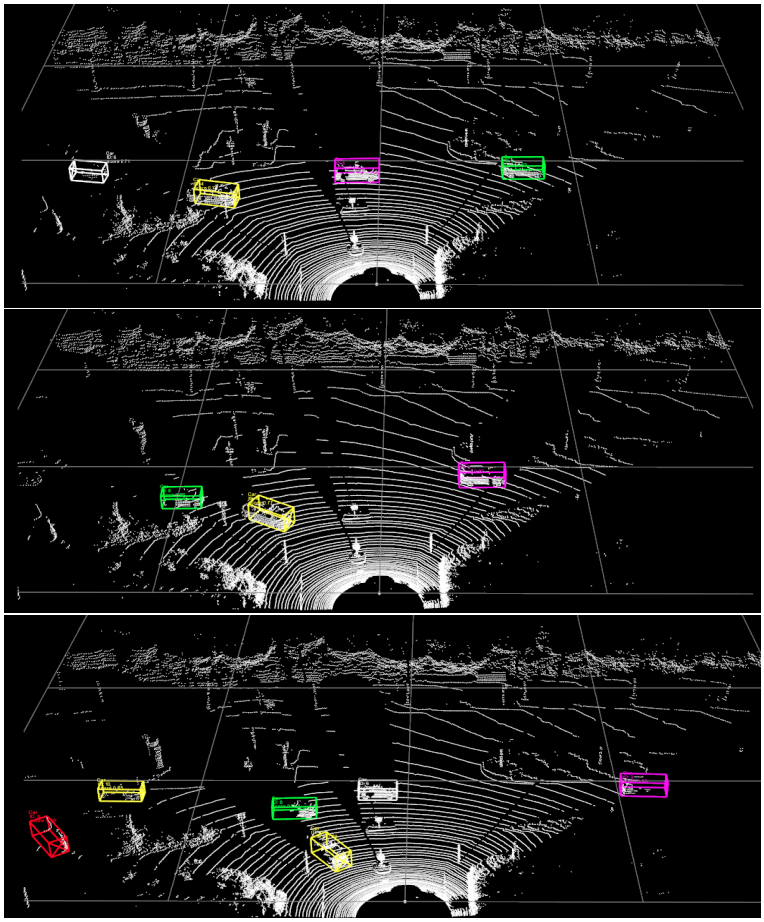

Figure 4.7: Kalman Filter Prediction Noise Increased

4.1.5 Losing Trackers too Early

A lot of example case shows that a tracker gets removed too fast, (after only 2 frame that they didn't have any match) and this situation causes to have different ID trackers for the same object because of even small occlusions or lost of detections.

Reason : Low Maximum Age Threshold

The maximum age threshold is so low that it even doesn't compensate small occlusions occurs for a few frame.

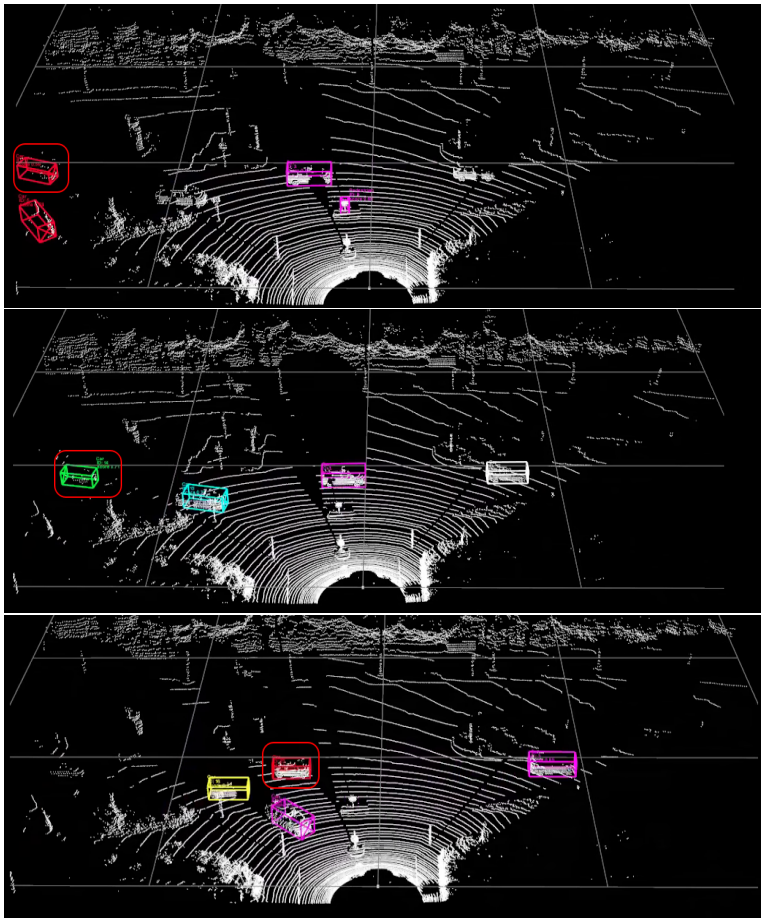

Figure 4.8: Frame 8 - car ID 7 — Frame 11 - car ID 13 — Frame 25 - car ID 23

Optimization :

Considering there are occlusion cases where we lost detection for a tracker but the object is still there even more than 100 frames, different experiments have been made to find the optimal value for this parameter. It has seen that for higher values than 10, causes to have too much trackers at the background and having more mismatches, ID switches etc. Therefore, it is decided to visualize a tracker if it is not unmatched for more than 5 frames and delete it if it is for more than 10 frames.

4.1.6 Greedy Match Algorithm Review

It is mentioned that both Greedy and Hungarian match algorithms have some weak points. Since it is decided to continue with greedy algorithm, I updated the method to cover its main weak point. According to that:

- If an already matched tracker is chosen as the lowest distant tracker for another detection, instead of ignoring it and to assign next lowest tracker to that detection, it is checked if the tracker is assigned to a detection having higher distance.
- If the assigned detection has a higher distance than the current possible pair, the old match is removed and the tracker is assigned to this later coming detection.
- If a detection loses its tracker by this new rule, the standard greedy algorithm steps continue to be applied for this detection by adding it to the list of detections that have not yet been matched to any tracker.

4.1.7 Mobile Lidar Case - Ego Motion Compensation

In case of losing the detection for a few frames, beside of wrong match problem explained and solved in section Mahalanobis Threshold - Match Deny Mechanism, when the ego vehicle (the car having the Lidar sensor) is not stable but moving, constant velocity Kalman Predictions doesn't work as good as in fixed lidar car case.

Reason :

When the lidar is mobile, the coordinates of the object in lidar frame changes even if the object is not moving due to the its relative velocity. If the detection is missing in a frame that ego vehicle has a significant changes in its speed, the Kalman Predictions made by constant velocity from the previous frame can't take in consideration. This problem causes that Kalman Prediction is calculated with wrong speeds (dx dy dz da) and to place the tracker in a wrong location at the next frame. If we have a correct detection match with this false calculated kalman box, we don't see anything wrong due to the correction step. But if we don't have the detection, since the tracker is not corrected but just updated by Kalman Prediction, we see that the object moves in a wrong direction.

Linear Ego Motion Effect

When the marked tracker is examined in the figure 4.9, it is seen that the object moves in the same direction with ego motion even if its actually stable. (It can't move forward with its left side anyway)

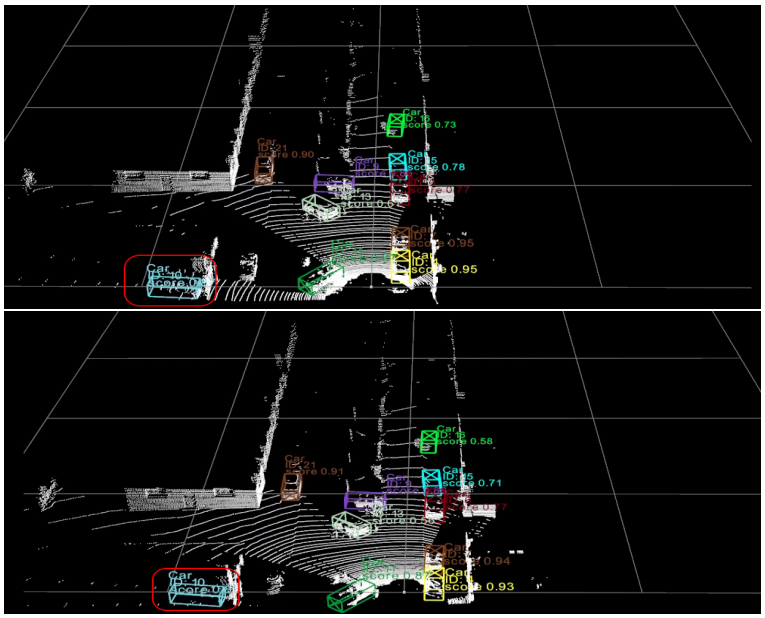

Figure 4.9: Linear Ego Motion

Angular Ego Motion Effect

When the marked tracker is examined in the figure 4.10, it is seen that it orientates in the same way with ego motion even if its stable.

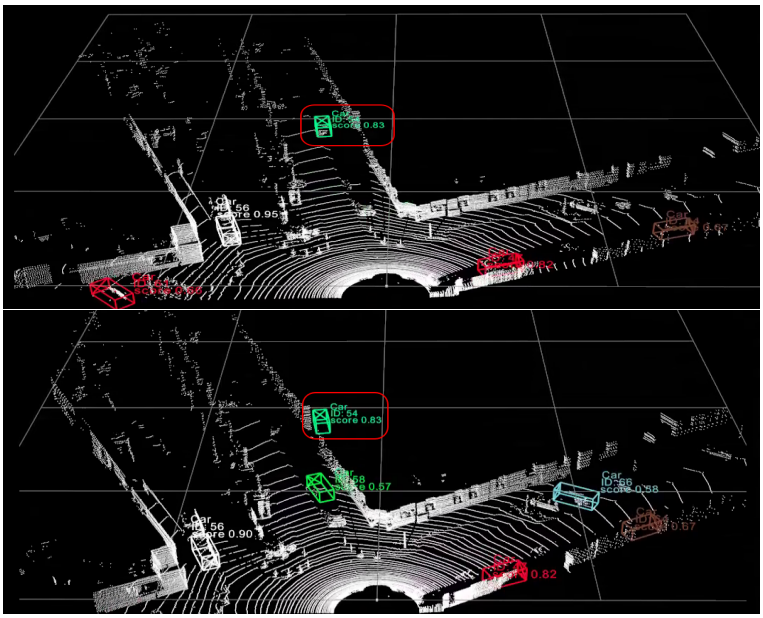

Figure 4.10: Angular Ego Motion

Optimization:

To handle with mobile lidar case, the velocity of the lidar vehicle is used to update the position of Kalman Predictions. For that purpose, linear and angular velocities given in OXTS files in KITTI Tracking dataset is used.

- Orientation

  The rotation matrix is calculated using the yaw angle speed of lidar vehicle. This matrix is applied to the Kalman Prediction boxes to correct the object locations in lidar frame The angular speed of the vehicle is given in OXTS file as wz in rad/s unit. Counter clockwise direction is accepted as positive direction.

- Translation

  The translation of the lidar vehicle is calculated by linear speed in x and y directions. The object is translated in the opposite direction of this movement to correct the position. The linear speed of the vehicle is given in OXTS file as vf and vl in m/s unit.

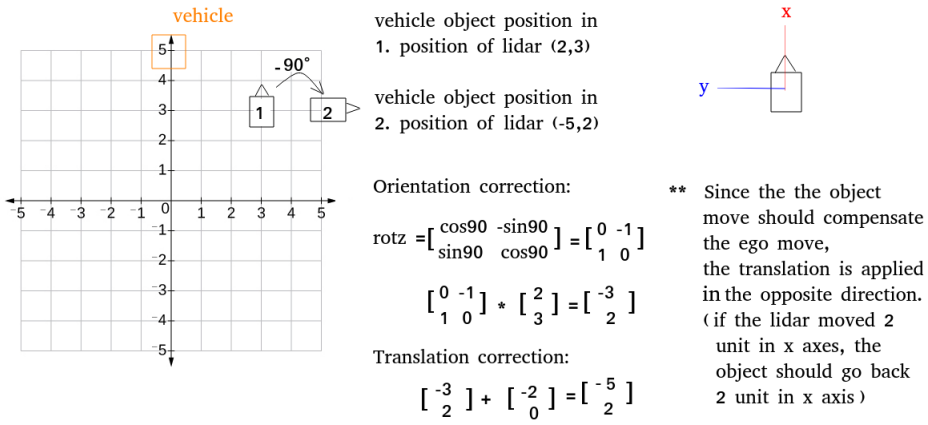

Figure 4.11: Ego Motion Compensation Logic

4.2 Challenges

Some optimizations are more challenging than others due to the problem's complexity. In this chapter, this additional situations, the background of this specific optimizations and their challenging parts will be explained.

4.2.1 Local Match Deny Mechanism

Different sequences have different properties depending on where the sequence is taken (city, auto road, urban, etc). The Lidar case (stationary or mobile), the object case(stationary or mobile, if mobile, the direction according to the Lidar) effects the relative speed of the object which is very important to determine the Mahalanobis threshold as well as the Match Deny Mechanism threshold. Figure 4.12 shows the effect of different example cases on the absolute and relative velocity of an object.

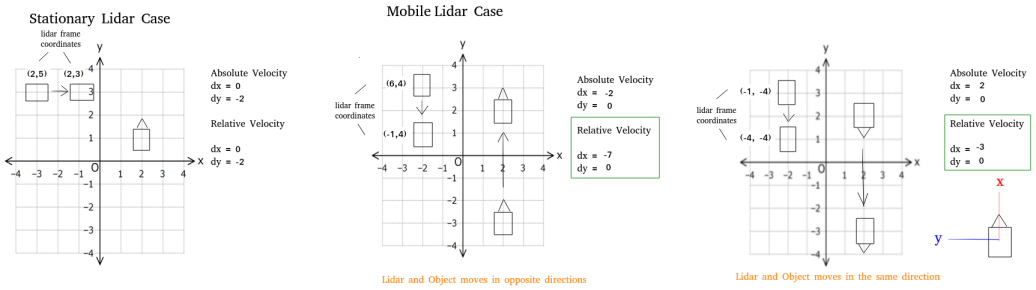

Figure 4.12: Absolute and Relative Velocity for Stationary and Mobile Lidar Cases

In other words, a reasonable match deny the threshold, which eliminates the wrong matches by looking at the displacement of the tracker between the current and previous frame, which may change depending on the lidar-object motion. A car object moving with the same absolute velocity may have different relative velocities according to the direction of the move.
As an example case, in the following frames, the Kalman Predictions and the Detections are visualized where we see a true match is denied since the distance between matched tracker and the detection is 5.76 which makes 207 km/h. In stationary lidar or mobile lidar + moving car in a parallel direction case, this distance would be a clear sign for the false match, but in this case it's a true match since 207 km/h relative velocity corresponds to an absolute velocity around 100 km/h, which is still a reasonable speed but denied by the match deny mechanism. The figure 4.13 shows a new detection (blue) and its tracker (brown) initialized directly at the same location.

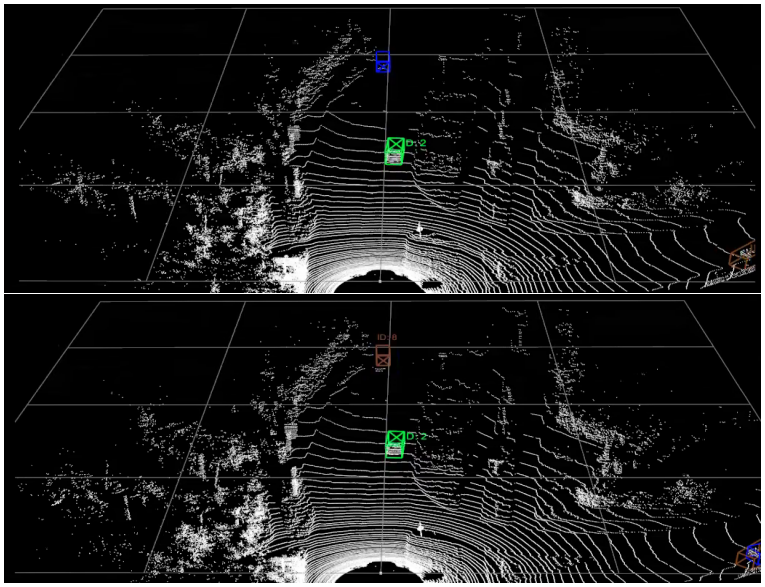

Figure 4.13: New Detection (blue) and its Tracker (brown)

The figure 4.14 shows the Kalman prediction for the same tracker in the next frame. Since the velocity variables in the state vector is initialized with 0 and its updated after the first match, it is seen that Kalman Prediction comes at the same location with the last frame meanwhile the car didn't stay stable and moved through the lidar vehicle. Therefore we obtain Kalman Prediction and Detection for the next frame with a distance 5.76.

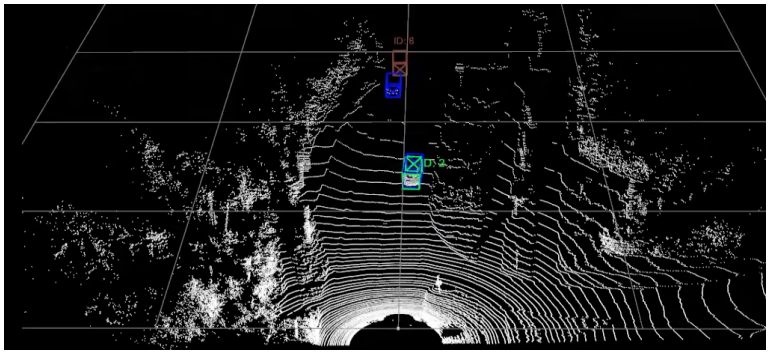

Figure 4.14: Kalman Prediction for the Next Frame

Since the match is denied and the Kalman Prediction is not corrected, the tracker predictions (brown) and detections (blue) boxes continues to diverge as shown in figure 4.15

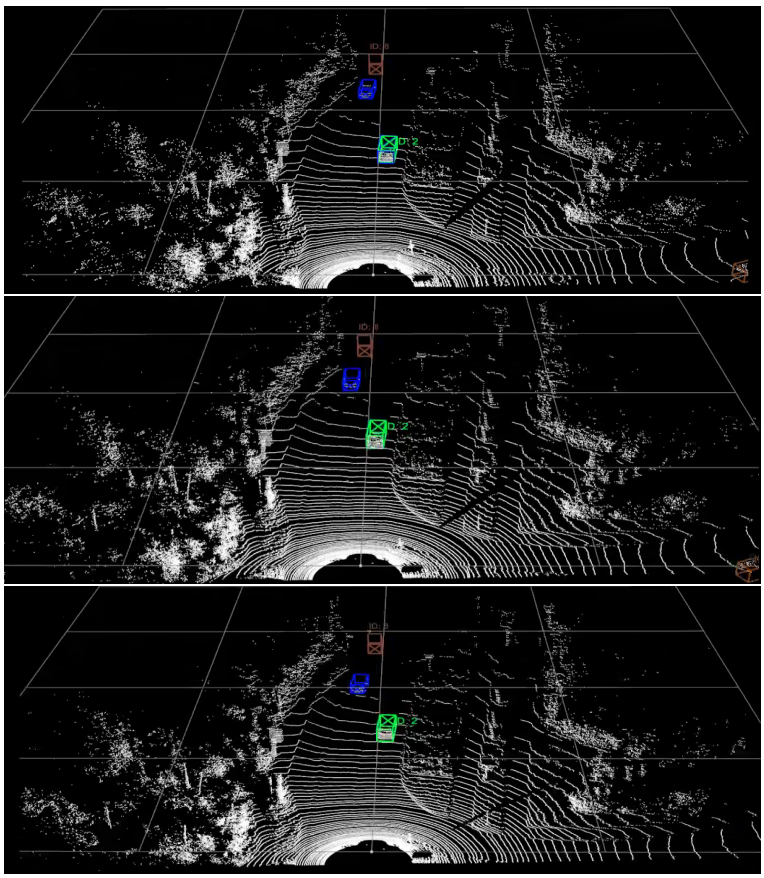

Figure 4.15: Divergence of Tracker Predictions and Detections

Match deny mechanism is created to eliminate some false matches. These false matches are coming especially in crowded scenes like city scenes and a small threshold is needed to eliminate false matches since the cars don't move so fast in general. On the other hand in the scenes having not a lot of objects like highways, where we see just a few car objects moving faster than city scenes, not false matches occur like in city scenes and to not eliminate the true matches a higher threshold should be chosen or the match denies mechanism should be removed. To not lose the true matches independently if the lidar is mobile or not, if the object moves parallel or through the lidar, we decided to use the acceleration of trackers instead of putting a fixed threshold which is optimal for different scenes.

Mean Speed Calculation for Match Deny Threshold

Initialize the tracker with mean speed = 0 when the first detection comes. At the first match, assign the object location's distance between the current and last frame as the mean speed. After the first match check if the new distance is higher than the mean_speed + threshold. If yes, deny the match, if not update the mean speed.

4.2.2 Absolute vs Relative Speed in Kalman Update

When a Kalman predicted tracker is matched with detection, the update step is applied to update the state and the state covariance matrices. Since the velocity is calculated in relative speed, the velocity parameters in the vectors are updated according to the relative speed which takes the ego-motion into account already. This situation makes the ego-motion compensation step false if it is already updated by Kalman's update step since the tracker box is already in the correct place having the relative velocity. To prevent translating and orienting the tracker boxes two times (1 by Kalman update 1 by ego-motion compensation), a control mechanism is added to check if the calculated relative velocity is already similar to the velocities that come from Kalman vectors. The ego-motion compensation is applied only if the calculated relative velocity is significantly different from the Kalman vectors which is the case mostly if detection is lost at a frame it changed its velocity significantly.

Chapter 5

Conclusion

In this chapter, the results of different experiments made until the final model obtained are explained. Therefore some qualitative outputs are shown.

5.1 Quantitative Results

Quantitative Results are obtained using AB3DMOT's proposed 3D Multi Object Tracking Evaluation implementation which is based on KITTI 2D Multi Object Tracking Evaluation implementation.

5.1.1 Evaluation Metrics

Since the tracking task has slightly different goals than the tasks like image classification or object detection, the standard evaluation metrics are different than these tasks too. In this section, common evaluation metrics and their meanings is explained.
↓ : The less the better ↑ : The more the better **False Alarm Rate (FAR↓)** : The rate of false alarms (false positives) per frame.
Multi Object Tracking Accuracy (MOTA↑) : Accuracy metric for multi object tracking is calculated as follows:

$$1 - \frac{\sum_t FN_t + FP_t + IDSW_t}{\sum GT_t} \tag{5.1}$$

Multi Object Tracking Precision (MOTP↑) : Precision metric multi object tracking is calculated as follows:

$$\frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{5.2}$$

c_t : Total matches made between ground truth and the detection output

d_t : Distance between the localization of objects in the ground truth and the detection output

MT↑: Number of mostly tracked trajectories, i.e the tracker has the same label for at least 80% of its life span.

ML↓: Number of mostly lost trajectories, i.e the tracker is not tracked for at least 20% of its life span.

IDSW↓: Number of identity switch, is counted if a ground truth target i is matched to track j and the last known assignment was k 6 = j

Recall (REC↑) : The number of correctly matched detections divided by the total number of detections in ground truth.

Precision (PRE↑) : The number of correctly matched detections divided by the total number of output detections

Fragmentation: The number of interrupted tracks. It is counted when a track is missed for a frame.

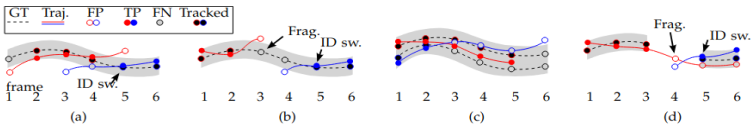

Fig. 9: Four cases illustrating tracker-to-target assignments. (a) An ID switch occurs when the mapping switches from the previously assigned red track to the blue one. (b) A track fragmentation is counted in frame 3 because the target is tracked in frames 1-2, then interrupts, and then reacquires its 'tracked' status at a later point. A new (blue) track hypothesis also causes an ID switch at this point. (c) Although the tracking results is reasonably good, an optimal single-frame assignment in frame 1 is propagated through the sequence, causing 5 missed targets (FN) and 4 false positives (FP). Note that no fragmentations are counted in frames 3 and 6 because tracking of those targets is not resumed at a later point. (d) A degenerate case illustrating that target re-identification is not handled correctly. An interrupted ground truth trajectory will typically cause a fragmentation. Also note the less intuitive ID switch, which is counted because blue is the closest target in frame 5 that is not in conflict with the mapping in frame 4.

Figure 5.1: Fragmentation vs IDSW (modified and retrieved from [17]

	MOTA↑	MOTP↑	IDSW↓	Frag↓	ML↓	MT↑
PT	0.6937	0.7047	0	613	0.1028	0.4184
PT + OG	0.7241	0.7032	0	322	0.0985	0.4597
PT + OG + M0.4	0.7617	0.6729	10	194	0.0940	0.5266
PT + OG + M0.4 + KP	0.7776	0.7100	0	122	0.0851	0.5638
PT + OG + M0.4 + KP + MA	0.8103	0.7082	0	64	0.0816	0.6206
PT + OG + M0.4 + KP + MA + MDM	0.7993	0.7103	2	58	0.0851	0.5975
PT + OG + M.04 + KP + MA + EMC	0.7916	0.7089	246	311	0.0816	0.6170
PT + OG + M.04 + KP + MA + MDM + EMC	0.7916	0.7099	63	124	0.0904	0.5762
PT + M0.4 + HU + KP + MA	0.8076	0.7089	4	67	0.0780	0.6152
PT + IOU0.25 + HU + KP + MA	0.7956	0.7056	1	50	0.1046	0.6011

Table 5.1: Evaluation on Car Class

5.1.2 Evaluation Results

The tables 5.1.2, 5.1.2, 5.1.2 show the effects of different experiments on our base tracking model with the following shortcuts:

- PT : Probabilistic Tracking Model [5] default configurations (with Mahalanobis Distance and Greedy Algoritm)
- OG : Optimized Greedy
- M0.4 : Mahalanobis Optimization applied on PT
- KP: Kalman Prediction Noise optimization
- MDM : Match Deny Mechanism
- EMC : Ego Motion Compensation
- HU : Hungarian Algorithm
- GD : Greedy Algorithm

	MOTA↑	MOTP↑	IDSW↓	Frag↓	ML↓	MT↑
PT	0.6283	0.6736	0	33	0.0811	0.5946
PT + OG	0.6290	0.6720	0	26	0.9021	0.6023
PT + OG + M0.4	0.6294	0.6360	0	22	0.1081	0.6216
PT + OG + M0.4 + KP	0.6541	0.7413	0	19	0.1081	0.6757
PT + OG + M0.4 + KP + MA	0.7262	0.7229	0	9	0.0270	0.7568
PT + OG + M0.4 + KP + MA + MDM	0.7251	0.7228	0	9	0.0270	0.7568
PT + OG + M.04 + KP + MA + EMC	0.7289	0.7247	0	8	0.0270	0.7838
PT + OG + M.04 + KP + MA + MDM + EMC	0.7278	0.7246	0	8	0.0270	0.7838
PT + M0.4 + HU + KP + MA	0.7251	0.7226	0	9	0.0270	0.7568
PT + IOU0.25 + HU + KP + MA	0.6224	0.7369	0	10	0.5946	0.2703

Table 5.2: Evaluation on Cyclist Class

	MOTA↑	MOTP↑	IDSW↓	Frag↓	ML↓	MT↑
PT	0.3977	0.6426	1	422	0.2994	0.0419
PT + OG	0.4265	0.6378	1	404	0.2756	0.0452
PT + OG + M0.4	0.4423	0.6365	3	378	0.2545	0.0516
PT + OG + M0.4 + KP	0.4630	0.6343	4	302	0.2395	0.0599
PT + OG + M0.4 + KP + MA	0.7122	0.6293	38	250	0.1796	0.4731
PT + OG + M0.4 + KP + MA + MDM	0.6852	0.6334	11	188	0.1916	0.4371
PT + OG + M.04 + KP + MA + EMC	0.7135	0.6278	42	259	0.1737	0.4671
PT + OG + M.04 + KP + MA + MDM + EMC	0.6765	0.6317	39	209	0.1856	0.4311
PT + M0.4 + HU + KP + MA	0.6866	0.6254	26	278	0.1737	0.4491
PT + IOU0.25 + HU + KP + MA	0.6641	0.6284	2	184	0.3234	0.3353

Table 5.3: Evaluation on Pedestrian Class

For Pedestrian and Car classes, we see that Greedy Match, Mahalanobis Threshold, Kalman Prediction, Maximum Age optimizations improves the MOTA, Fragmentation, ML and MT metrics while increasing Mahalanobis Threshold all alone has a negative effect on multi object tracking precision and ID switch. Similarly, increasing the maximum age threshold is indispensable for accuracy but it has a bad influence on ID switch metric. Match Deny Mechanism has a controversial effect since we see a small improvement in MOTP, fragmentation and ML results but same size of disimprovement on MOTA, IDSW and MT metrics. On the other hand, even if the Ego Motion Compensation improves the spesific cases detected during qualitative experiments, it seems that it decreases the performance on mostly all the metrics (except MOTP and MT). It is seen that the Hungarian algorithm doesn't perform better than Optimized Greedy with the optimal configurations (PT + OG + M0.4 + KP + MA) and it has slightly worse performance. For Cyclist class, besides achieving very similar improvements with the optimizations, it is seen that Ego Motion Compensation has a better effect for Cyclist class where all the metrics obtain better results without any exception.

As the conclusion, we see the optimizations may give slightly different effects on different classes but especially greedy match, mahalanobis distance, kalman prediction step and maximum age optimizations are necessary for all cases. On the other hand, match deny mechanism and ego motion compensation doesn't give a robust improvement on the quantitative results even they were effective to solve some spesific errors detected. Therefore, even if they don't significantly effect the model speed where the tracking model speed is 95ms per frame without additional Match Deny and Ego Motion Compensation steps and 99ms per frame with them, it is decided to stay in simpler version without these two optimizations.

5.2 Future Work

This work is dedicated to analyze Kalman Filter based tracking by detection mechanism. Although satisfying results are obtained between the first and optimized versions of the implementation, three interesting steps are considering as future work.

5.2.1 Joint Detection and Tracking Methods

Joint Detection and Tracking methods [21, 29, 30] are the second category of object tracking algorithms where a deep learning model which unifies both the detection and tracking pipelines. To work and analyze with these category will be the second future work to compare the performance with this project.

5.2.2 Fusion

While image and lidar data have different advantages and disadvantages. Lidar has a huge range compared to image but it is sparse, images can distinguish and interpret color but sensitive to the light unlike Lidar. To benefit from both data may increase the model's performance by making up for each other's shortcomings. To implement a tracking by detection model, a fusion object detection model [4] which fuses Lidar and camera information in order to perform 3D object detection will be reviewed. Another interesting paper [12] which fuse these two different data type to create an joint detection and tracking methods will be taken as reference.

Bibliography

[1] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. Tracking without bells and whistles. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, October 2019.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. 2016.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving, 2019.

[4] Can Chen, Luca Zanotti Fragonara, and Antonios Tsourdos. Roifusion: 3d object detection from lidar and vision, 2020.

[5] Hsu-kuang Chiu, Jie Li, Rares Ambrus, and Jeannette Bohg. Probabilistic 3d multi-modal, multi-object tracking for autonomous driving. 2020.

[6] Patrick Dendorfer, Aljosa Osep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth, and Laura Leal-Taixé. MOTChallenge: A benchmark for single-camera multiple target tracking. *International Journal of Computer Vision*, December 2020.

[7] Huazhu Fu, Dong Xu, Stephen Lin, and Jiang Liu. Object-based rgbd image co-segmentation with mutex constraint. pages 4428–4436, 06 2015.

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

[9] Ben Graham. Sparse 3d convolutional neural networks, 2015.

[10] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks, 2017.

[11] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[12] Aleksandr Kim, Aljoša Ošep, and Laura Leal-Taixé. Eagermot: 3d multi-object tracking via sensor fusion, 2021.

[13] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[14] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2018.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[16] Yassine Maalej, Sameh Sorour, Ahmed Abdel-Rahim, and Mohsen Guizani. Tracking 3d lidar point clouds using extended kalman filters in kitti driving sequences. *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018.

[17] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking, 2016.

[18] Sparsh Mittal and Vibhu . A survey of accelerator architectures for 3d convolution neural networks. *Journal of Systems Architecture*, 115, 01 2021.

[19] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[21] Bing Shuai, Andrew G. Berneshawi, Davide Modolo, and Joseph Tighe. Multi-object tracking with siamese track-rcnn, 2020.

[22] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. 2020.

[23] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. Ab3dmot: A baseline for 3d multi-object tracking and new evaluation metrics. 2020.

[24] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric, 2017.

[25] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, Yunlong Wang, and Diange Yang. Pandaset: Advanced sensor suite dataset for autonomous driving, 2021.

[26] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18:3337, 10 2018.

[27] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. 2020.

[28] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box, 2021.

[29] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FairMOT: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11):3069–3087, September 2021.

[30] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points, 2020.

[31] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection, 2017.