FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

LTE
Lehrstuhl für Technische Elektronik

Lehrstuhl für Technische Elektronik
Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel
Prof. Dr.-Ing. Georg Fischer

Masterarbeit

im Studiengang
"Advanced Signal Processing and Communication Engineering (ASC)"

von

Subhasri Mitra

zum Thema

Vulnerable Road User Tracking using 3D Radar Data and Learned Appearance Model from PointNet

Betreuer:	Anand Dubey, M.Sc.
Beginn:	03.08.2020
Abgabe:	26.02.2021

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 26.02.2021

Subhasri Mitra

Kurzfassung

Radar hat viele Anwendungen in fortschrittlichen Fahrerassistenzsystemen und gilt als eine der Schlüsseltechnologien für das hochautomatisierte Fahren. Für das autonome Fahren ist eine zuverlässige Wahrnehmung der Fahrzeugumgebung von entscheidender Bedeutung. Dies beinhaltet die Erkennung, Klassifizierung und Verfolgung von Zielen in der Umgebung eines autonomen Fahrzeugs. In dieser Arbeit wird ein radarbasierter Ansatz zur Erkennung und Verfolgung von gefährdeten Verkehrsteilnehmern (Vulnerable Road Users (VRUs)) in einer automobilen Umgebung vorgestellt. Die Hardware eines Millimeterwellen-Radars (mm-Welle) wird zusammen mit seiner Signalverarbeitungskette in MATLAB® unter Verwendung der Phased Array System Toolbox™ simuliert. Moderne hochauflösende Radarsensoren erzeugen mehrere Reflexionen pro zu erkennendem Objekt, was diese Sensoren besonders für die Aufgabe der 3D-Objekterkennung interessant macht. Es wird ein Ansatz zur Erkennung von 3D-Objekten in einem Fahrszenario vorgestellt, der ausschließlich auf Radardaten basiert und ein neuronales Netz namens PointNet verwendet. Diese Arbeit untersucht außerdem die Möglichkeit, das Aussehen eines Objekts in Form von Objektmerkmalen, die vom PointNet extrahiert werden, auszunutzen, um die Datenassoziation und die Performance eines Multi-Objekt Trackers zu verbessern. Das PointNet wird trainiert und die Leistung der entwickelten Multi-Objekt Tracking Algorithmen wird anhand eines Fahrdatensatzes evaluiert, der mehrere realistische Fahrmanöver von VRUs enthält und mit der Automated Driving System Toolbox™ für automatisierte Fahrsysteme generiert wurde. Die Ergebnisse zeigen das Potenzial der Objekterkennung und verfolgung in hochauflösenden Radardaten mit PointNet.

Abstract

Radar has many applications in advanced driver assistance systems (ADASs) and is considered as one of the key technologies for highly automated driving (HAD). For autonomous driving, reliable perception of the vehicle surroundings is paramount. This includes detection, classification and tracking of targets in the surroundings of an autonomous vehicle. This thesis presents a radar based approach to detect and track Vulnerable Road Users (VRUs) in an automotive setting. A millimeter wave (mm-wave) radar's hardware along with its signal processing chain and propagation environment is simulated in MATLAB® using the Phased Array System Toolbox™. Modern high-resolution radar sensors generate multiple reflections per object of interest, which make these sensors particularly suitable for the 3D object detection task. An approach to detect 3D objects in a driving scenario solely depending on sparse radar data, using a deep neural network called PointNet, is presented. This contribution further investigates the possibility of exploiting the object appearance in the form of object features learned by the PointNet to improve the data association and tracking performance of a multi-object tracker. The PointNet is trained and the performance of the developed multi-object tracking algorithms is evaluated using a driving dataset containing several realistic driving maneuvers of VRUs, generated using MATLAB®'s Automated Driving System Toolbox™. The results show potential of object detection and tracking in high-resolution radar data using PointNet.

Contents

List of Abbreviations

2D	2-dimensional
3D	3-dimensional
ADAS	Advanced Driver Assistance System
CFAR	Constant False Alarm Rate
CNN	Convolutional Neural Network
CW	Continuous Wave
EKF	Extended Kalman Filter
FC	Fully Connected
FMCW	Frequency Modulated Continuous Wave
HAD	Highly Automated Driving
KF	Kalman Filter
MLP	Multi-Layered Perceptron
RCS	Radar Cross Section
ReLu	Rectified Linear Unit
SNR	Signal-to-Noise Ratio
T-Net	Transformation Network
UKF	Unscented Kalman Filter
VRU	Vulnerable Road User

Nomenclature

x	a variable
$f(x)$	a function of variable x
$exp(x)$	exponent of x
t	time
Δt	simulation time step
$\dot{x}$	derivative of x with respect to t
(x,y,z)	a point in 3D
$\mathbf{x}$	a vector. Bold lower case letters represent vectors
x_i	i^{th} element of vector $\mathbf{x}$
$\mathbf{X}$	a matrix. Bold upper case letters represent matrices
X_{ij}	element in the i^{th} row and j^{th} column of matrix $\mathbf{X}$
$[.]^T$	transpose of a matrix
$[.]^{-1}$	inverse of a matrix
$\|.\|_F$	Frobenius norm of a matrix
$\mathbf{I}$	identity matrix
$\hat{.}$	estimated value
$\tilde{.}$	innovation or measurement pre-fit residual

1 Introduction

One of the most dynamic topics in the automotive industry is the development of ADAS toward HAD. The performance and reliability of these systems strongly depend on the capabilities of the environmental sensing. Radar technology has some unique advantages when compared to camera or lidar technologies and has become indispensable for the development of ADASs and HAD. Radar works reliably in bad weather and lighting conditions; can provide accurate and direct measurements of range, relative velocity, angle, RCS of multiple targets; and offers high range coverage of more than 200 meter [1]. Radar is typically used in current ADASs, such as adaptive cruise control (ACC), forward collision avoidance (FCA), lane-change assist, or evasion assist, to name a few [2].

1.1 Motivation

Vulnerable road users (VRUs) is a term used for those actors in a traffic scenario that are at most risk of injury due to collision with other vehicles. VRUs mainly include pedestrians and bicyclists, as these road actors have no protective shield around them which make them most susceptible to road accidents. According to [3], globally, over half of road traffic deaths are vulnerable road users. Therefore, developing autonomous driving systems, that take into account the safety of VRUs, is paramount. For autonomous driving, the perception of vehicle surrounding is an important task. In particular, tracking of multiple targets in a driving scenario using noisy data from sensors, such as radar, lidar and camera, is a crucial task. A standard approach is to preprocess received sensor data in order to generate target detections, which are used as input for a tracking algorithm. Object detection using camera or lidar sensor data has already been widely researched. Being robust to all weather condition, modern high-resolution radar sensors generate multiple detections per target. However, the nature of received radar data is extremely sparse compared to lidar point clouds or camera images. For this reason, it is a challenging task to recognize different objects solely using radar data. Therefore, radar has widely been used in fusion with other sensors like camera and lidar for target detection and tracking in ADAS subsystems [4–6].

This contribution presents a method to detect VRUs in high-resolution radar data using a machine learning approach. Radar data is represented as a point cloud, consisting of lateral and longitudinal position and Doppler velocity. Since radar targets are represented as point clouds, it is desirable to use raw point clouds as input for a neural network. Most existing approaches which process radar data use certain representation transformations in order to use neural networks, e.g. radar data is transformed into a grid map representation. However, PointNet [7] is a novel neural network that makes it possible to directly process point clouds. Although radar data is sparse compared to lidar data, radar data contains strong features in form of Doppler or RCS information. For example, the micro-motion of human movement in the presence of radar illumination creates unique modulations in the received signal known as the micro-Doppler ef-

fect [8]. By analyzing these frequency modulations, one can infer the type of movement being performed by the target. Another advantage of radar data is that it often contains reflections of a target part which is not directly visible, e.g., wheel houses at the opposite side of a vehicle. All these features can be very beneficial to classify radar targets.

For any type of surveillance system where dynamic targets are involved, target tracking is an important component. The objective of a tracking algorithm is to collect data from sensors and filter data from the same object over time into a so-called track. From these tracks, estimated characteristics such as velocities, accelerations and future positions of tracked objects can be obtained. One key step in tracking multiple targets is to associate data to the right tracks. This is referred to as data association. Using radars in an automotive setting may give rise to more than one detection per object per scan. The majority of the target tracking literature focuses on point targets, i.e., targets that give rise to at most one detection. An object that may give rise to several detections is referred to as an extended object. Hence, the area of tracking such objects is denoted as extended target tracking. This thesis will evaluate the capabilities of radar-based target tracking and investigate how to handle extended targets in an automotive setting by extending standard point tracking algorithms. The tracked targets are extended, and shall be represented by their position, velocity, length, width and appearance in the form of object features, learned by the PointNet.

1.2 Outline

Except from the brief introduction to the problem in Chapter 1, this thesis consists of five additional chapters. Chapter 2 introduces the reader to the key concepts used in this work. It gives an overview of radar technology and signal processing techniques, in the context of autonomous driving. It briefly touches upon evolutionary work in object detection using deep neural networks and moves on to introduce the PointNet architecture. Finally, it discusses the target tracking problem and various aspects of it. We conclude the chapter with discussion of metrics used to evaluate the performance of a tracking algorithm.

Chapter 3 describes the simulation model used in this work. It defines the automotive radar hardware and signal processing chain along with all relevant simulation parameters. It also defines the PointNet architecture used in this thesis for target detection and the training dataset used. In the last section, the multi-object tracking algorithms are presented along with their mathematical models.

Chapter 4 presents the results of the object detection and tracking subsystems. The performance of the target tracking algorithms are evaluated and comparison are made among different approaches.

Chapter 5 summarizes the work done in the course of this thesis and provides direction for further research in this area.

Chapter 6 concludes the thesis.

2 Background

This chapter will go through the necessary theory in order to get an understanding of the key concepts covered in this thesis. The basics of radar are touched upon and various aspects of automotive radar signal processing are summarized. In the next section, convolutional neural networks are introduced, which form the backbone of most object detection frameworks. The chapter continues with an elaborate description of the PointNet. In the last sections of the chapter, the target tracking is described along with a summary of existing state-of-the-art algorithms to track targets and the metric used to evaluate the performance of such trackers.

2.1 Radar

A radar can simultaneously transmit and receive Electromagnetic (EM) waves in frequency bands ranging from 3 MHz to 300 GHz [9]. It is designed to extract information (i.e., location, range and velocity) about targets using the EM waves reflected from those targets. Automotive radar [1, 9] systems typically operate in 24 GHz and 77 GHz bands of the EM spectrum, known as Millimeter wave (mm-wave) frequencies so that higher velocity and range resolution can be achieved. Fundamental radar operation involves three main tasks: range (distance), relative velocity, and direction estimation of targets, as discussed next.

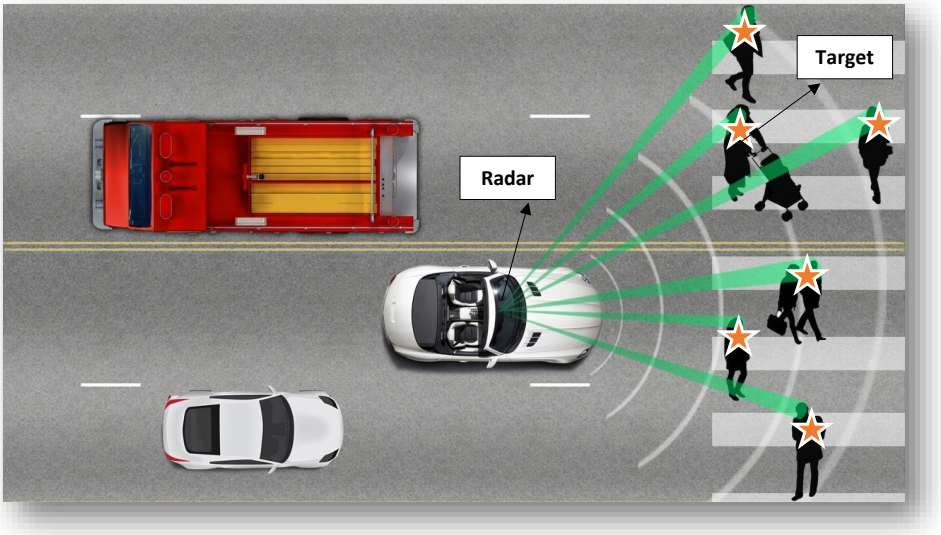


Figure 2.1: A car with an automotive radar sensor that can detect targets in its field of view.

2.1.1 Frequency Modulated Continuous Wave (FMCW) Radar

Continuous wave (CW) radar is a type of radar system where a known stable frequency continuous wave radio energy is transmitted and then received from any reflecting objects. Individual targets can be detected using the Doppler effect, which causes the received signal to have a different frequency than the transmission, allowing it to be detected by filtering out the transmitted frequency. FMCW radar is a special type of radar sensor which radiates continuous transmission power like a CW radar but unlike CW radar, FMCW radar can change its operating frequency during the measurement, that is, the transmitted signal is modulated in frequency (or in phase). Simple CW radar devices without frequency modulation have the disadvantage that they cannot determine target range because they lack the timing mark necessary to allow the system to time accurately the transmit and receive cycle and to convert this into range. Such a time reference can be generated by frequency modulation of the transmitted signal. In this method, a signal is transmitted, which increases or decreases in the frequency periodically. When an echo signal is received, that change of frequency gets a delay Δt. In FMCW radar, the differences in phase or frequency between the actually transmitted and the received signal are measured. For the carrier frequency f_c and FM modulation constant K, a single FMCW pulse within the time duration T can be written as [9],

$$s(t) = e^{j2\pi(f_c+0.5Kt)t} \; 0 \leqslant t \leqslant T, \tag{2.1}$$

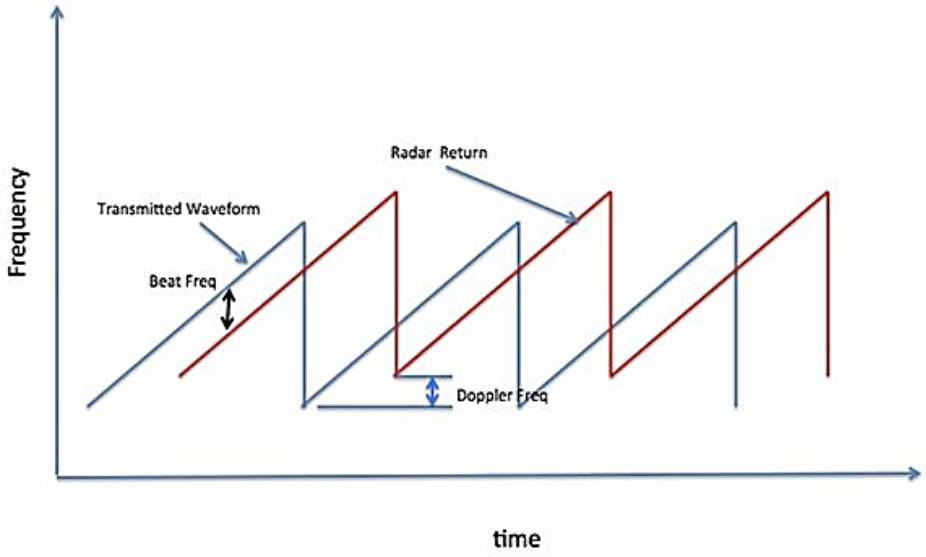


Figure 2.2: A FMCW Waveform with saw-tooth shape pulses [10]. The transmitted signal is shown in blue and the reflected signal in red. The radar measures not only the frequency difference between the transmitted and received signal or beat frequency (caused by the run time), but also the Doppler frequency (caused by the speed).

2.1.2 Radar Signal Processing

Range Estimation

Range estimation [9] is a fundamental characteristic of automotive radars. The range R, to a target, is determined based on the round-trip time delay that the EM waves take to propagate to and from that target: $R = c\tau/2$, where τ is the round-trip time delay in seconds and c is the speed of light in meter per second ($c \approx 3X10^8 m/s$) . Thus, the estimation of τ enables the range measurement.The form of the EM waves (signals) that a radar transmits is important for round-trip time delay estimation. For example, pulse-modulated continuous waves (CWs) consist of periodic and short power pulses and silent periods. Silent periods allow the radar to receive the reflected signals and serve as timing marks for radar to perform range estimation. However, unmodulated CW signals cannot be used for range estimation since they lack such timing marks. Additionally, the signal reflected from a target should arrive before the next pulse starts. Hence, the maximum detectable range of a radar depends on pulse repetition interval T_{PRF}.

Range resolution of a radar denotes the ability to distinguish closely spaced targets. Two targets can be separated in the range domain only if they produce non overlapping returns in the time domain. Hence, the range resolution of a radar is proportional to the pulse width. Although finer pulses provide higher resolution, shorter pulses contain less energy and therefore, poor receiver SNR and detection performance. Pulse compression [9] is a technique used to overcome this problem. The goal of pulse compression is to transmit a long duration pulse of high energy, but to detect a short duration pulse to localize the receive filter output response to one or at most two radar range bins. Early radars accomplished this by transmitting a signal with linear frequency modulation. The pulse would start at a low frequency sinusoid, and increase the frequency over the duration of the radar pulse. This is referred to a "chirp." All digital radars can also perform pulse compression, by using a matched filter. Due to the pulse compression, the range resolution is inversely proportional to the bandwidth of the FMCW signal and is independent of pulse width. For example, the short-range FMCW radar uses ultra wideband (UWB) waveforms to measure small distances with higher resolution.

Velocity Estimation

Estimation of the target velocity [9] is based on the phenomenon called Doppler effect. Suppose a car is moving ahead with a differential velocity v. Due to the existence of relative motion between two cars, the reflected waves are delayed by time $\tau = 2(R \pm vt)/c$. The time dependent delay term causes a frequency shift in the received wave known as the Doppler shift $f_{vd} = (\pm 2v/\lambda)$. The Doppler shift is inversely proportional to wavelength λ, and its sign is positive or negative, depending on whether the target is approaching or moving away from the radar. While this frequency shift can be detected using Continuous Wave (CW) radar, it lacks the ability to measure the targets range. Thus, a pulsed radar configuration that uses frequency modulated FMCW pulses is needed that provides simultaneous range and velocity estimation in multi-target traffic scenarios. Velocity resolution of a radar is the minimum velocity difference between two targets traveling at the same range that the radar can distinguish. It depends on the radar wavelength, the pulse repetition period, and the number of consecutively sampled echoes.

Direction Estimation

Use of wideband pulses such as FMCW provides discrimination of targets in both distance and velocity. The discrimination in direction [9] can be made by means of an antenna array. In a realistic traffic scenario there will be several targets surrounding the radar that collects direct and multi-path reflections from them. In order to spatially resolve equidistant targets and comprehensively represent the traffic scene, angular location of targets should be estimated. Therefore, in automotive radars, the location of a target is often described in terms of a spherical coordinate system (R,θ,ϕ), where (θ,ϕ) denote azimuth and elevation angles, respectively. For this, single antenna radar setup is not sufficient, since the measurements lack the information in terms of angular locations of targets.

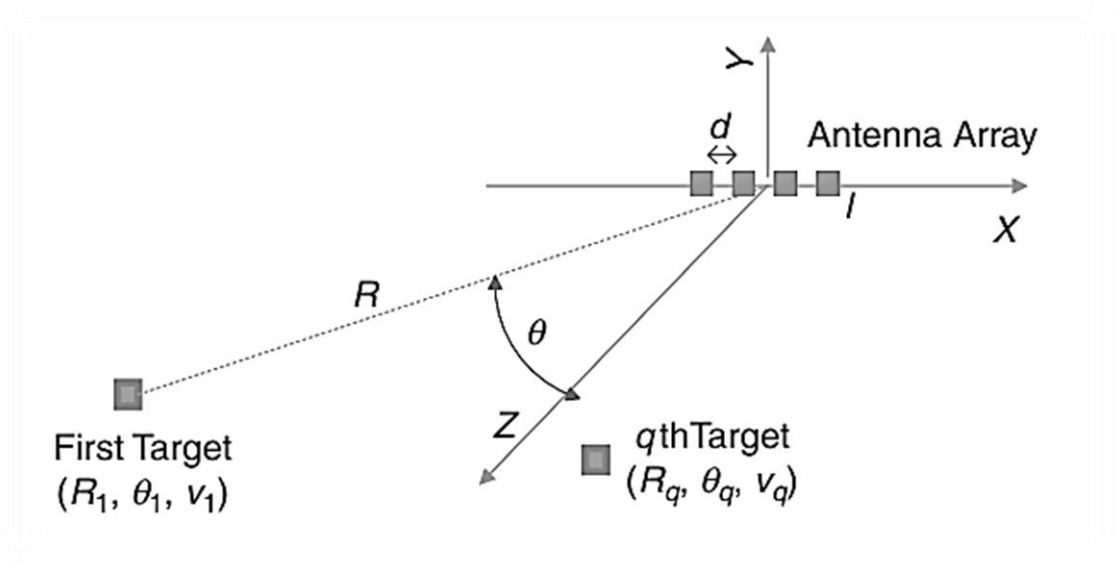


Figure 2.3: The azimuth angle estimation setup using uniform linear antenna array [9].

To enable direction estimation, the radar should collect reflected wave data across multiple distinct dimensions. For example, locating a target using EM waves in 2D requires the reflected wave data from the object to be collected in two distinct dimensions. These distinct dimensions can be formed in many ways using combinations of time, frequency, and space. For instance, a linear antenna array and wideband waveform such as FMCW form two unique dimensions. Additionally, smaller wavelengths in mm-wave bands correspond to smaller aperture sizes and thus, many antenna elements can be densely packed into an antenna array. Hence, the effective radiation beam, which is stronger and sharper, in turn increases the resolution of angular measurements.

Consider an antenna array located in plane $z = 0$, and let l be the abscissa corresponding to each receiver antenna position, as shown in Figure 2.3. Let (R_q,θ_q) be the position of the q^{th} target in spherical coordinates, moving with velocity v_q relative to the radar. Using far field

approximation, for the q^{th} target, the round-trip time delay between a transmitter located at the origin and the receiver positioned at coordinate l is given by [9],

$$\tau_{lq} = \frac{2(R_q + v_q t) + ldsin\theta_q}{c}, \tag{2.2}$$

where d is the distance between antenna elements (usually half the wavelength) arranged in a linear constellation.

Range Doppler Estimation

The signal reflected from a target is conjugately mixed with the transmitted FMCW signal to produce a low-frequency beat signal, whose frequency gives the range of the target. This operation is repeated for P consecutive pulses. 2D waveform in Figure 2.4(a) depict successive reflected pulses arranged across two time indices. The slow time index p simply corresponds to pulse number. On the other hand, the fast time index n assumes that for each pulse, the corresponding continuous beat signal is sampled with frequency f_s to collect N samples within the time duration T. The pulse period is T_0. Assuming single target and neglecting reflected signal distortions, the FMCW radar receiver output as a function of these two time indices is given by [9],

$$d(n,p) \approx exp\left\{j2\pi \left[\left(\frac{2KR}{c} + f_d \right) \frac{n}{f_s} + f_d p T_0 + \frac{2f_c R}{c} \right] \right\} + \omega(n,p), \tag{2.3}$$

Applying discrete Fourier transform across fast time n gives the beat frequency $f_b = \frac{2KR}{c}$ coupled with Doppler frequency f_d. This operation is known as range transform or range gating, which allows the estimation of Doppler shift corresponding to an unique range gate by the application of second Fourier transform across the slow time. A range-Doppler map can be found efficiently by using 2D fast Fourier transform (FFT). For Q number of targets, the three-dimensional (3D) FMCW radar output signal can be obtained by combining equations 2.3 and 2.2 and can be represented as [9],

$$d(l,n,p) \approx \sum_{q=0}^{Q-1} \alpha_q \, exp\left\{j2\pi \left[\left(\frac{2KR_q}{c} + f_{dq} \right) \frac{n}{f_s} + \frac{f_c ld \sin\theta_q}{c} + f_{dq} p T_0 + \frac{2f_c R_q}{c} \right] \right\} + \omega(l,n,p), \tag{2.4}$$

where α is a complex scalar whose magnitude represents attenuation due to antenna gain, path loss, and the RCS of the target and ω is additive white Gaussian noise with zero mean.

Range-Doppler maps are a common representation for radar data. In a Range Doppler map, the target's range is plotted against Doppler velocity, also known as Doppler shift. The magnitude of the reflected signal is illustrated with colors, ranging from blue to red, where red represents the largest reflected values and blue the lowest.

Constant False Alarm Rate (CFAR) Detection

CFAR detection is an adaptive algorithm used in radar systems to detect target returns against a background of noise, clutter and interference [11]. In the radar receiver, the returning echoes are typically received by the antenna, amplified, down-converted and then passed through detector

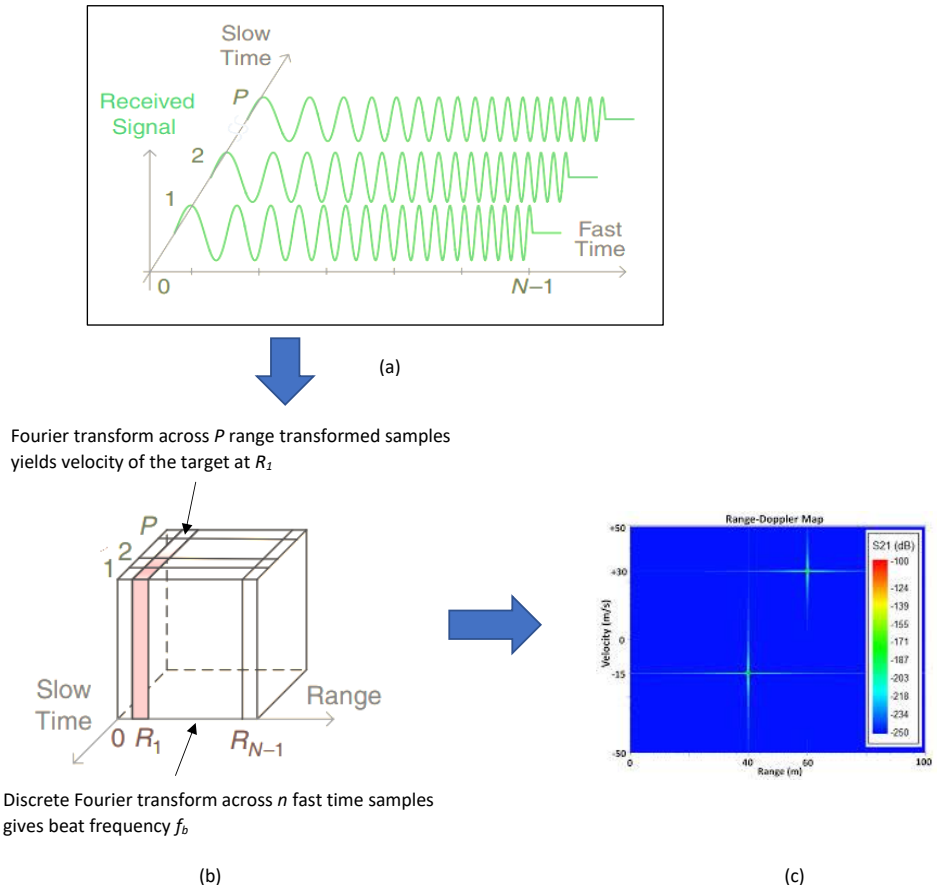


Figure 2.4: 2D joint range-Doppler estimation with FMCW Radar [9].(a) depicts a 2D FMCW reflected waveform along fast and slow time dimensions.(b) shows the process of applying FFT for range and Doppler estimation (c) shows the resulting range Doppler map obtained by applying 2D FFT on (a).

circuitry that extracts the envelope of the signal. This envelope is proportional to the power of the received echo and comprises of the wanted echo signal and the unwanted power from internal receiver noise and external clutter and interference.

The role of the constant false alarm rate circuitry is to determine the power threshold above which any return can be considered to probably originate from a target. If this threshold is too low, then more targets will be detected at the expense of increased numbers of false alarms. Conversely, if the threshold is too high, then fewer targets will be detected, but the number of false alarms will also be low. In most radar detectors, the threshold is set in order to achieve a required probability of false alarm[1]. If the background against which targets are to be detected is constant with time and space, then a fixed threshold level can be chosen that provides a specified probability of false alarm.. However, in most fielded systems, unwanted clutter and

[1]source:`https://en.wikipedia.org/wiki/Constant_false_alarm_rate`

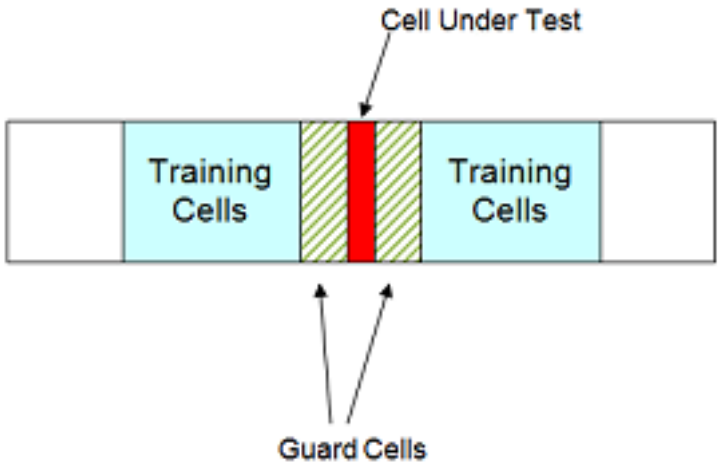


Figure 2.5: The relationship among cells for 1D cell averaging CFAR Detector[2].

interference sources mean that the noise level changes both spatially and temporally. In this case, a changing threshold can be used, where the threshold level is raised and lowered to maintain a constant probability of false alarm. In many phased array systems, because of the cost associated with a false detection, it is desirable to have a detection threshold that not only maximizes the probability of detection but also keeps the probability of false alarm below a preset level. In CFAR, when the detection is needed for a given cell, often termed as the cell under test (CUT), the noise power is estimated from neighboring cells. Then the detection threshold, T, is given by,

$$T = \alpha P_n \tag{2.5}$$

where P_n is the noise power estimate and α is a scaling factor called the threshold factor. From equation (2.5), it is clear that the threshold adapts to the data. It can be shown that with the appropriate threshold factor, α, the resulting probability of false alarm can be kept at a constant, hence the name CFAR. In a cell averaging CFAR detector, noise samples are extracted from both leading and lagging cells (called training cells) around the CUT. The noise estimate can be computed as [12],

$$P_n = \frac{1}{N} \sum_{m=1}^{N} x_m \tag{2.6}$$

where N is the number of training cells and x_m is the sample in each training cell. In general, the number of leading and lagging training cells are the same. Guard cells are placed adjacent to the CUT, both and leading and lagging it. The purpose of these guard cells is to avoid signal components from leaking into the training cell, which could adversely affect the noise estimate. Figure 2.5 shows the relation among these cells for the 1-D case.

[2]source:`https://www.mathworks.com/help/phased/ug/constant-false-alarm-rate-cfar-detection.html;jsessionid=6144c923db1d66f9df1f0e1a6e0a`

2.2 Object Detection and Classification with Machine Learning

Generic object detection aims at locating and classifying existing objects in an image, and labeling them with rectangular bounding boxes to show the confidence of existence. Traditional object detection methods are built on handcrafted features and shallow trainable architectures. Their performance easily stagnates by constructing complex ensembles which combine multiple low-level image features with high-level context from object detectors and scene classifiers. With the rapid development in machine learning, more powerful tools, which are able to learn semantic, high-level, deeper features, are introduced to address the problems existing in traditional architectures [13].

2.2.1 Neural Networks

Deep neural network represents a type of machine learning where the system uses many layers of nodes that are densely interconnected to derive high-level functions from input information. It means transforming the data into a more creative and abstract component. These networks are of high importance in many applications today, such as computer vision and natural language processing. Most of today's neural nets are organized into layers of neurons, and they are feed-forward, meaning that data moves through them in only one direction. An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data. The fundamental architecture of deep neural networks is based on the fully connected layer, shown in Figure 2.6. This layer contains neurons where each neuron is connected to all neurons in the adjacent layers and not connected to any neuron in the same layer. A neuron is simply a unit that takes several inputs and computes an activation value to pass forward to neurons in the next layer. The overall goal of the network is to approximate the true model $f(\mathbf{x})$, by the network model $h(\mathbf{x})$, based on the input $\mathbf{x}$. This is known as the universal approximation theorem [14]. The activation function introduces non-linear properties to the neural network. Two commonly used activation functions are the sigmoid function, $\sigma(\mathbf{x})$, and the ReLu function. When dealing with classifiers it is preferable to have an activation function in the final layer of the network that yields a probabilistic output. The is achieved for binary classification tasks by the sigmoid function. The softmax function can be seen as a generalization of the sigmoid function, for a classification task with multiple classes. Softmax outputs a vector of probabilities, one for each class.

2.2.2 Convolutional Neural Network

Convolutional neural networks (CNN) is a type of neural network that specializes in image recognition [15] and computer vision tasks. Classic neural network architecture was found to be inefficient for computer vision tasks. Images represent a large input for a neural network. In a classic fully connected network, this requires a huge number of connections and network parameters. A convolutional neural network leverages the fact that an image is composed of smaller details, or features, and creates a mechanism for analyzing each feature in isolation, which informs a decision about the image as a whole. A convolutional neural network consists of an input layer, hidden layers and an output layer. In a convolutional neural network, the

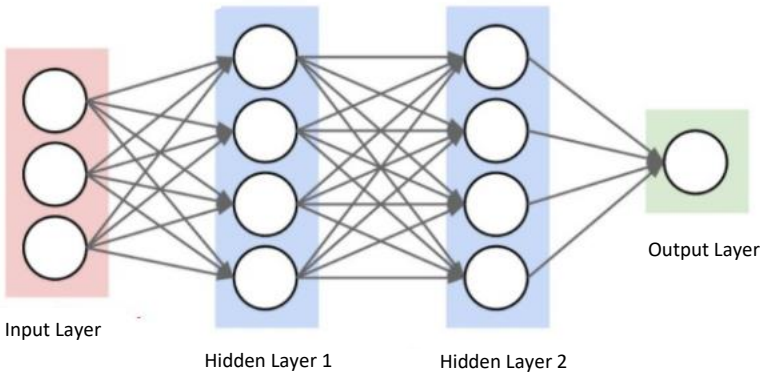


Figure 2.6: A fully connected neural network with an input layer with three inputs, two hidden layers with four neurons per layer and one output[3].

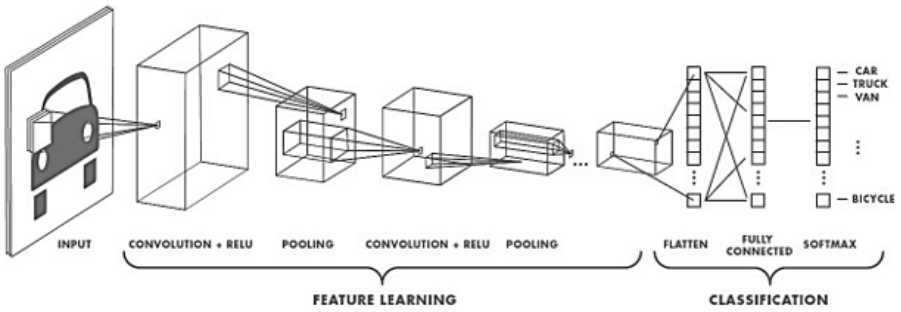


Figure 2.7: A vehicle classifier based on CNN with two convolution and max pooling layers for feature extraction and fully connected layers with softmax activation for classification on the learned features[4].

hidden layers include layers that perform convolutions. This is followed by other convolution layers such as pooling layers and fully connected layers, as shown in Figure 2.7. Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small patches of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel, as shown in Figure 2.8. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. This is followed by ReLu activation. Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and

[3]source:`https://cs231n.github.io/convolutional-networks/`

[4]source:`https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`

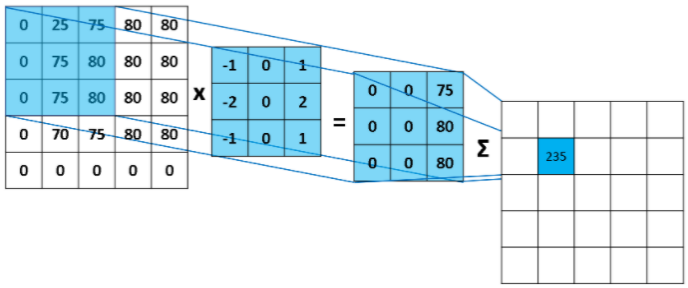


Figure 2.8: The convolution operation between an input matrix and a kernel matrix. The output is a feature map. The information in multiple feature maps is summarized by pooling[5].

max pooling that summarize the average presence of a feature and the most activated presence of a feature, respectively. Lastly, a FC layer takes the output of convolution/pooling and predicts the best label to describe the image.

2.2.3 PointNet

Applying deep learning to 3D point cloud data has many challenges. These challenges include occlusion, which is caused by cluttered scenes or blindsides; noise/outliers, which are unintended points; and point misalignment, etc. The most significant challenges regarding the application of deep learning to point clouds are irregularity, lack of structure and unorderedness. Point cloud data are irregular, meaning that the points are not evenly sampled across the different regions of an object/scene, so some regions could have dense points while others have sparse points. Irregularity can be attenuated by sub sampling techniques, but cannot be completely eliminated. Secondly, point cloud data are not placed on a regular grid. Each point is scanned independently, and its distance to neighboring points is not always fixed. In contrast, pixels in images are represented on a two-dimensional grid, and the spacing between two adjacent pixels is always fixed. A point cloud of a scene is the set of points (usually represented by x, y and z coordinates) obtained around the objects in the scene, and these are usually stored as a list in a file. As a set, the order in which the points are stored does not change the scene represented, therefore, it is invariant to permutation. These properties of point clouds are very challenging for deep learning. Typical convolutional architectures require highly regular input data formats, like those of image grids or 3D voxels, in order to perform weight sharing and other kernel optimizations. Since point clouds or meshes are not in a regular format, they are typically transformed to regular 3D voxel grids [16] or collections of images [17] (e.g, views) before being fed to a deep net architecture. This data representation transformation, however, renders the resulting data unnecessarily voluminous and introduces quantization artifacts that can obscure natural invariances of the data.

PointNet [7] is a network designed to consume point cloud data and perform object classification and part segmentation on the dataset. This is desirable since point clouds resemble the

[5] source:`https://medium.com/@kinisanketh/getting-started-with-cnn-18c03efc7d06`

way raw sensor data is received. Point clouds are simple and unified structures that avoid the combinatorial irregularities and complexities of meshes, and thus are easier to learn from.

In PointNet, each point is processed independently. In the basic architecture a point is represented by 3D coordinates (x,y,z). Additional dimensions, e.g. color and normal, can be added. Unlike pixel arrays in images or voxel arrays in volumetric grids, point cloud is a set of points without specific order. In other words, a network that consumes N 3D point sets needs to be invariant to $N!$ permutations of the input set in data feeding order. In order to make a model invariant to input permutations, the PointNet uses a structure of MLP networks [18] which are shared by all points and a max pooling layer, which is a symmetric function to aggregate information from each point, shown in Figure 2.9. The output from the max pooling forms a vector $[f_1, \ldots, f_k]$, which is a global signature of the input set. A Support Vector Machine (SVM) or MLP classifier can be trained on the shape of global features for classification.

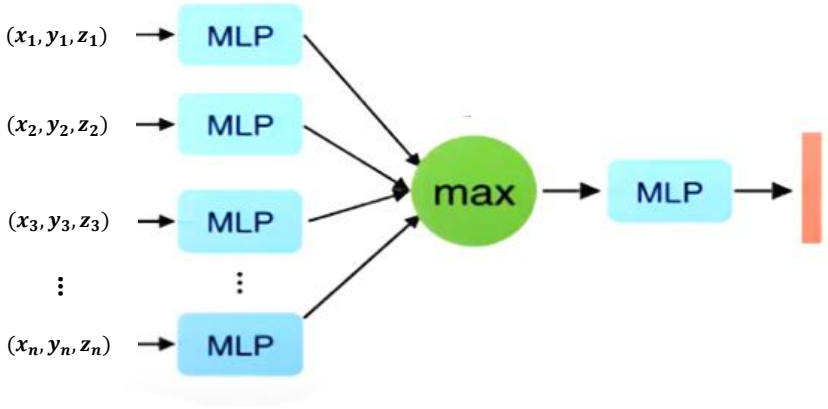


Figure 2.9: Module for order invariance of n input points in three dimensions [19]. The MLP structure is shared, that is, each point uses the same MLP. 'max' stands for max pooling.

As a geometric object, the learned representation of the point set should be invariant to certain transformations. For example, rotating and translating points all together should not modify the global point cloud category. The PointNet achieves this by predicting an affine transformation matrix by a mini T-Net [20], as shown in Figure 2.10, and directly applies this transformation to the coordinates of input points. The mini network itself resembles the big network and is composed of basic modules of point independent feature extraction, max pooling and fully connected layers. The T-Net, which can be thought of as a special type of spatial transformer network (STN), is a light-weight regression PointNet that estimates the true center of the complete object and then transforms the coordinate such that the predicted center becomes the origin. The PointNet inserts another alignment network on point features and predicts a feature transformation matrix to align features from different input point clouds. However,

transformation matrix in the feature space has much higher dimension than the spatial transform matrix, which greatly increases the difficulty of optimization. Therefore, a regularization term is added to the softmax training loss. The feature transformation matrix is constrained to be close to orthogonal matrix,

$$L_{reg} = \left\| \mathbf{I} - \mathbf{A}\mathbf{A}^T \right\|_F^2 \tag{2.7}$$

where $\mathbf{A}$ is the feature alignment matrix predicted by a mini-network. An orthogonal transformation will not lose information in the input, thus is desired. By adding the regularization term, the optimization becomes more stable and our model achieves better performance.

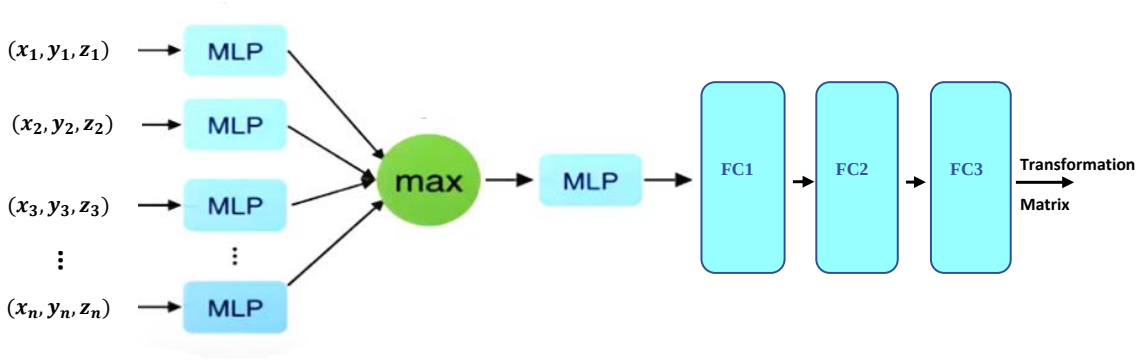


Figure 2.10: Architecture of the T-Net module [19] consisting of shared MLPs, max pooling and fully connected layers. The output is a transformation matrix

2.3 Target Tracking

Target tracking refers to the problem of using sensor measurements to determine the location, path and characteristics of objects of interest. The typical objectives of target tracking are the determination of the number of objects, their identities and their states, such as positions, velocities and in some cases their features. A typical example of target tracking is the radar tracking of aircrafts. There are a number of sources of uncertainty in the object tracking problem that render it a highly non-trivial task. For example, object motion is often subject to random disturbances, objects can go undetected by sensors and the number of objects in the field of view of a sensor can change randomly. The sensor measurements are subject to random noises and the number of measurements received by a sensor from one scan to the next can vary and be unpredictable. Objects may be close to each other and the measurements received might not distinguish between these objects.

2.3.1 Tracking Theory Overview

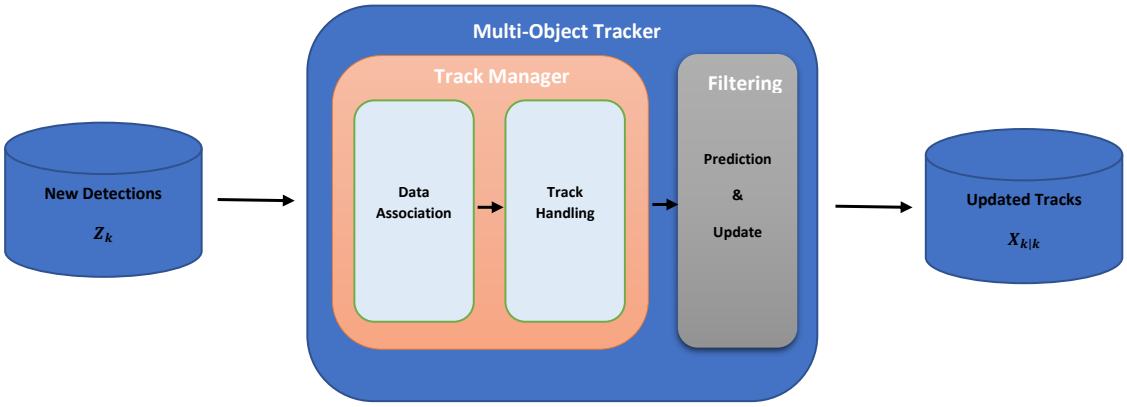


Figure 2.11: Generalized block diagram of a multi-object tracker: input is a new detection $\mathbf{z}_k$ at instant k; output is the updated state of the tracking filter $\mathbf{x}_{k|k}$. It contains modules to perform data association, track handling and filtering

The main objective of target tracking is to recursively estimate the states of the targets detected by one or several sensors. As illustrated in Figure 2.11, a tracking algorithm consists of four main parts, namely prediction, data association, measurement update and track handling. The prediction and measurement update steps are collectively referred to as filtering. Given the posterior state estimate of objects, $\mathbf{x}_{k-1|k-1}$, and their corresponding covariances, $\mathbf{P}_{k-1|k-1}$, at instance $k-1$, the current state estimates are predicted using the motion model that describes the dynamics of the objects. The measurements, $\mathbf{z}_k$, are associated to tracks to determine which measurements stem from which objects represented by tracks. Using the associated measurements, each track is updated with new information. The track handling module initializes new tracks for the non-associated measurements and deletes tracks that have not been updated for a number of time steps. After track handling, the target tracker outputs the current state update $\mathbf{x}_{k|k}$, and the covariance $\mathbf{P}_{k|k}$. This process is repeated at each time step in order to detect and track objects in the observed scene.

2.3.2 Filtering

Recursive Bayesian Estimation

Recursive Bayesian Estimation, also known as a Bayes Filter [21], is a general probabilistic approach for estimating an unknown probability density function (PDF) recursively over time using incoming measurements and a mathematical process model. The true state $\mathbf{x}$ is assumed to be an unobserved Markov process, and the measurements $\mathbf{z}$ are the observations of a Hidden Markov model (HMM). The distribution of interest is the posterior distribution [22],

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = p(\mathbf{x}_k|\mathbf{z}_k,\mathbf{z}_{1:k-1}), \tag{2.8}$$

of a state $\mathbf{x}_k$ at time k, given a set of measurements, $\mathbf{z}_{1:k} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_k\}$. From Bayes' theorem, the conditional probability can be derived as,

$$p(A|B) = \frac{p(B|A)p(A)}{P(B)}, \tag{2.9}$$

where A and B are events and $P(A)$ and $P(B)$ are probabilities for the respective events to occur. Using (2.9), (2.8) can be written as,

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \propto p(\mathbf{z}_k|\mathbf{x}_k,\mathbf{z}_{1:k-1})p(\mathbf{x}_k|\mathbf{z}_{1:k-1}), \tag{2.10}$$

where $p(\mathbf{z}_k|\mathbf{x}_k,\mathbf{z}_{1:k-1})$ can be simplified assuming that sensor noise is independent over time such that $\mathbf{z}_{1:k-1}$ can be disregarded in the expression. The remaining expression $p(\mathbf{z}_k|\mathbf{x}_k)$ is the likelihood function, or in filtering terms, the sensor model. $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is the prior distribution of $\mathbf{x}_k$ and can be obtained by marginalizing over the previous state $\mathbf{x}_{k-1}$ according to,

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1}, \tag{2.11}$$

Since the object dynamics follow a Markov model, the knowledge of the previous state $\mathbf{x}_{k-1}$ is sufficient to describe the future state $\mathbf{x}_k$, i.e., $p(\mathbf{x}_k|\mathbf{x}_{1:k-1}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$. This assumption leads to the simplification,

$$p(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{x}_{k-2},\dots,\mathbf{x}_1,\mathbf{x}_0) = p(\mathbf{x}_k|\mathbf{x}_{k-1}), \tag{2.12}$$

The resulting distribution $p(\mathbf{x}_k|\mathbf{x}_{k-1})$, is called the transition model or motion model. The posterior distribution $\mathbf{x}_{k|k}$ is proportional to the likelihood multiplied with the prior distribution as shown in (2.10). These equations are fundamental for popular filtering methods, such as Kalman filter.

Kalman Filter (KF)

The Kalman Filter (KF) [23] is derived from a Bayesian filter under the assumption that the transition and measurement models are linear and Gaussian. The predicted value in a Kalman Filter is represented by a Gaussian distribution. The predicted value is centered around the mean with the width of the Gaussian denoting the uncertainty in the prediction.

The filter estimates the mean and variance of the Gaussian distribution, represented as the state vector $\mathbf{x}_{k|k}$, and covariance matrix $\mathbf{P}_{k|k}$ at time k. The prediction step can be derived from (2.11) as,

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k, \tag{2.13}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}{\mathbf{F}_k}^T + \mathbf{Q}_k, \tag{2.14}$$

where $\mathbf{F}_k$ is the state transition model which captures the dynamics of the estimated states, $\mathbf{B}$ is the control input model which is applied to the control vector $\mathbf{u}_k$ and $\mathbf{Q}_k$ is the covariance of

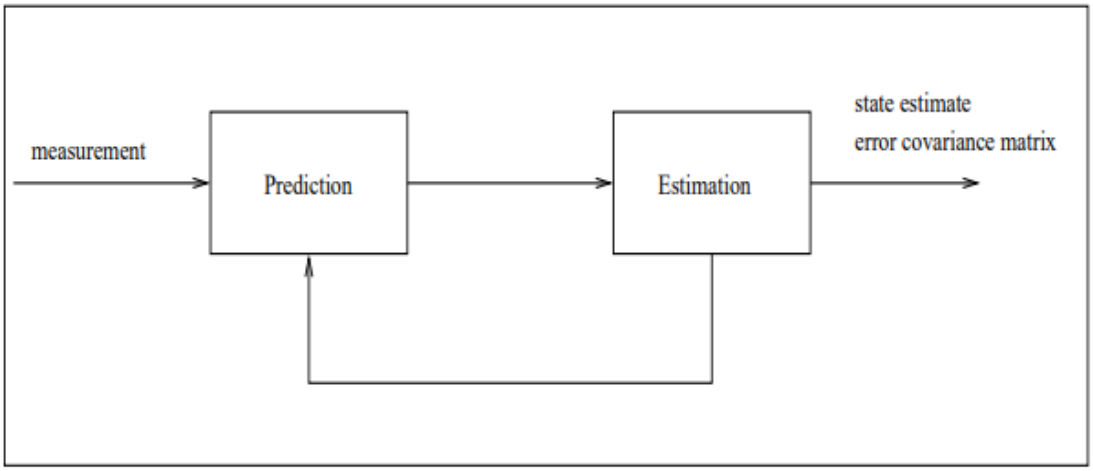


Figure 2.12: Block diagram of the recursive Kalman filter. It uses the current measurement and previous state estimate to estimate the current state of the filter [24].

the process noise which captures the event of model miss match between the real system and the transition matrix. The update step of the filter is given by equations,

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}, \tag{2.15}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} {\mathbf{H}_k}^T + \mathbf{R}_k, \tag{2.16}$$

where $\mathbf{R}_k$ is the covariance of the measurement noise.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} {\mathbf{H}_k}^T {\mathbf{S}_k}^{-1}, \tag{2.17}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k, \tag{2.18}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}, \tag{2.19}$$

where $\tilde{\mathbf{y}}_k$ is called the innovation or measurement pre-fit residual, $\mathbf{S}_k$ is the innovation covariance and $\mathbf{K}_k$ is the optimal Kalman gain. The update step results in posteriors $\hat{\mathbf{x}}_{k|k}$ and $\mathbf{P}_{k|k}$.

The Kalman gain is always been 0 and 1. If the Kalman gain is close to 1, it means the measurements are accurate but the estimates are unstable. If the Kalman gain is small, the error in the measurement is large relative to the error in the estimate. Estimates are more stable and the measurements are inaccurate. As a result, any difference between new data and the prediction will have a smaller effect on the eventual update. This filter runs iteratively, as shown in Figure 2.12, and updates at discrete time steps, whenever new measurements are available.

Non-Linear Kalman Filter

A Kalman filter works with linear functions and Gaussian distributions. As shown in Figure 2.13, if a Gaussian is fed with a linear function then the output is also a Gaussian. When

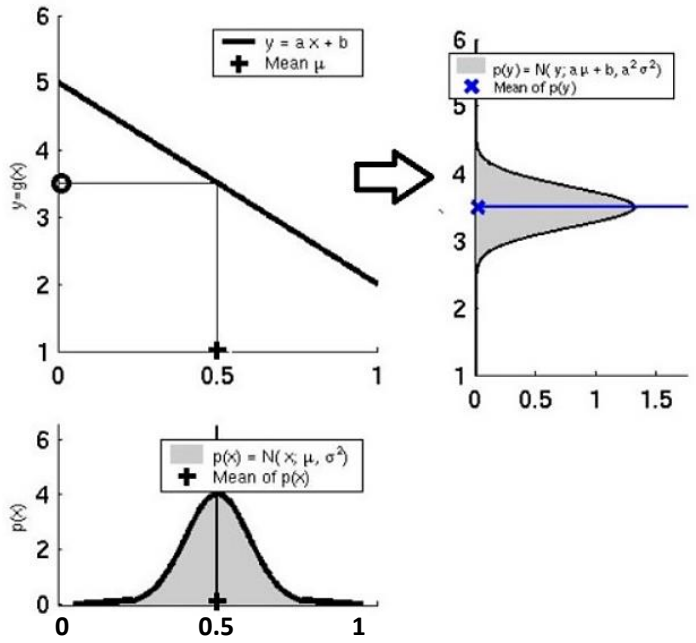


Figure 2.13: When Gaussian is fed with a linear function, then the output is also Gaussian[6].

a Gaussian is fed with a non-linear function, as shown in Figure 2.14, the output is a non-Gaussian distribution. Most real world problems involve non-linear functions. In most cases, the system is looking into some direction and taking measurement in another direction. This involves angles and sine, cosine functions, which are non-linear functions.

Non-linearity destroys the Gaussian and the mean and variance cannot be computed. In the non-linear Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \tag{2.20}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k, \tag{2.21}$$

where $\mathbf{x}_k$ represents the state, $\mathbf{z}_k$ represents the measurement, $\mathbf{w}_k$ and $\mathbf{v}_k$ are process and measurement noises. The noise is assumed to be additive white Gaussian, with a zero mean with covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$. The function f is used to compute the predicted state from the previous estimate and the function h is used to compute the predicted measurement from the predicted state. When filtering with non-linear models, the normal Kalman filter equations can no longer be used. A way to still be able to use Kalman-like algorithms is to perform linearization. This can be done either analytically, which leads to the extended Kalman filter, or statistically, giving the Unscented Kalman filter (UKF).

[6]source:`https://towardsdatascience.com/extended-kalman-filter-43e52b16757d`

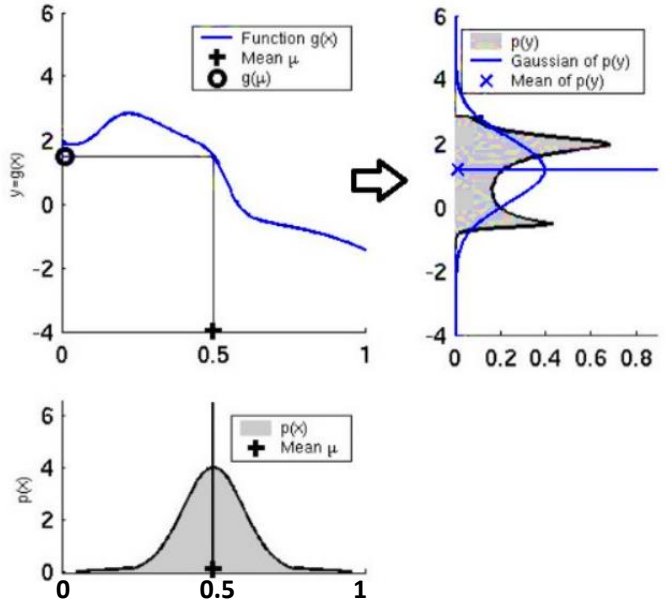


Figure 2.14: When a Gaussian is fed with a non linear function, then the output is non Gaussian[6].

Extended Kalman Filter (EKF)

The extended Kalman filter (EKF) [25] solves the nonlinear filtering problem, shown in Figure 2.15, by computing the Jacobian of f and h with current predicted states, at each time step. The state transition and observation matrices are defined to be the following Jacobians,

$$\mathbf{F}_k = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1},\mathbf{u}_k}, \tag{2.22}$$

$$\mathbf{H}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}, \tag{2.23}$$

Using the linearized prediction function $\mathbf{F}_k$, the prediction step of the EKF is given by,

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1},\mathbf{u}_k), \tag{2.24}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k\mathbf{P}_{k-1|k-1}{\mathbf{F}_k}^T + \mathbf{Q}_k, \tag{2.25}$$

Using the linearized measurement model function $\mathbf{H}_k$, the prediction step of the EKF is given by,

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}), \tag{2.26}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k|k-1}{\mathbf{H}_k}^T + \mathbf{R}_k, \tag{2.27}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}{\mathbf{H}_k}^T{\mathbf{S}_k}^{-1}, \tag{2.28}$$

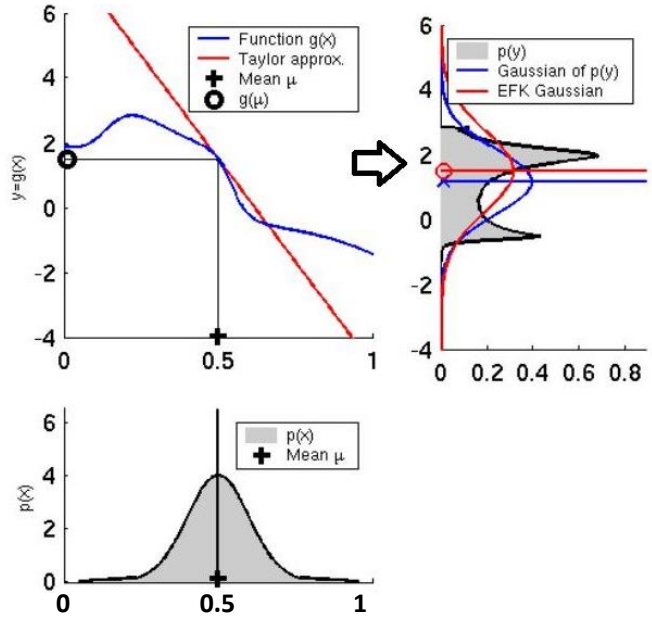


Figure 2.15: To feed a non linear function to a Gaussian, linearization by Taylor approximation is done in extended Kalman Filter. This results in Gaussian output[6].

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k, \tag{2.29}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}, \tag{2.30}$$

Unlike its linear counterpart, the extended Kalman filter in general is not an optimal estimator. Additionally, if the initial estimate of the state is wrong, or if the process is modeled incorrectly, the filter may quickly diverge, owing to its linearization. In the extended Kalman filter, the estimated covariance matrix tends to underestimate the true covariance matrix and therefore risks becoming inconsistent in the statistical sense without the addition of stabilizing noise.

Unscented Kalman Filter (UKF)

The unscented Kalman filter (UKF) [26] addresses the challenges faced by the EFK by using a deterministic sampling approach. The state distribution is again approximated by a Gaussian Random Variable (GRV), but is now represented using a minimal set of carefully chosen sample points. These sample points completely capture the true mean and covariance of the GRV, and when propagated through the true nonlinear system, captures the posterior mean and covariance accurately to the third order Taylor series expansion for any non-linearity. This process is called Unscented Transform (Figure 2.16). The EKF, in contrast, only achieves first order accuracy. Remarkably, the computational complexity of the UKF is the same order as that of the EKF.

The transformation is done as follows. First a number of points from the probability distribution are deterministically chosen. The number of points and their location depend on the dimension of the state space and on the covariance matrix, respectively. These points are then

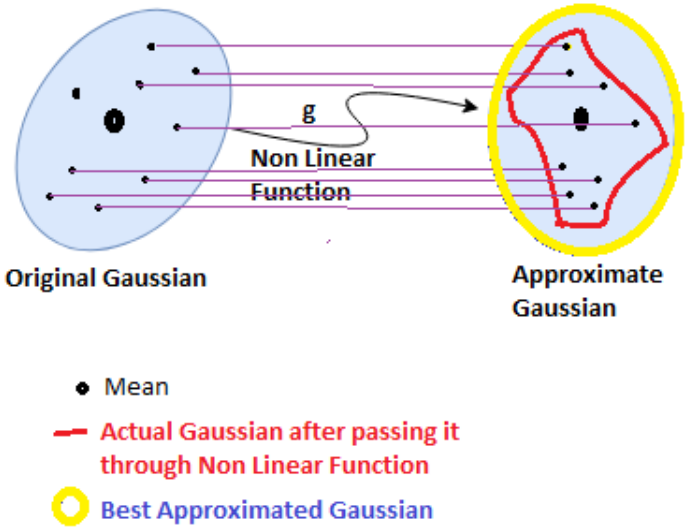


Figure 2.16: UKF takes sigma points from the source Gaussian and passes them through some non linear function and then calculates the new mean and variance of the transformed Gaussian. This process is called unscented transform[7].

transformed through the non-linear function. Finally, based on the transformed points, the mean and covariance matrix of the transformed random variable can be extracted. The deterministic points are referred to as sigma points and are given by the following equations [22],

$$\mathcal{X}^{(0)} = \hat{\mathbf{x}}_k, \tag{2.31}$$

$$\mathcal{X}^{(i)} = \hat{\mathbf{x}}_k + \sqrt{\frac{n}{1-W_0}}\mathbf{P}_i{}^{1/2} \ \forall i = 1,\dots,n \tag{2.32}$$

$$\mathcal{X}^{(i+n)} = \hat{\mathbf{x}}_k - \sqrt{\frac{n}{1-W_0}}\mathbf{P}_i{}^{1/2}, \ \forall i = 1,\dots,n \tag{2.33}$$

Here n is the dimension of the state space and W_0 is a design parameter. The UKF uses the sigma points along with corresponding weights W_i to create the prediction equations,

$$\hat{\mathbf{x}}_{k|k-1} \approx \sum_{i=0}^{2n} f(\mathcal{X}^i{}_{k-1})W_i \tag{2.34}$$

$$\mathbf{P}_{k|k-1} \approx \mathbf{Q}_{k-1} + \sum_{i=0}^{2n} (f(\mathcal{X}^i{}_{k-1}) - \hat{\mathbf{x}}_{k|k-1})(f(\mathcal{X}^i{}_{k-1}) - \hat{\mathbf{x}}_{k|k-1})^T W_i \tag{2.35}$$

where weights W_i are given by

$$W_i = \frac{1-W_0}{2n} \ \forall i = 1,\dots,n \tag{2.36}$$

[7] source:`https://towardsdatascience.com/the-unscented-kalman-filter-anything-ekf-can-do-i-can-do-it-better-ce7c773cf88d`

The update equations are given by

$$\hat{\mathbf{z}}_{k|k-1} \approx \sum_{i=0}^{2n} h(\mathcal{X}_k^i) W_i \tag{2.37}$$

$$\mathbf{P}_{xz} \approx \sum_{i=0}^{2n} (\mathcal{X}^i{}_k - \hat{\mathbf{x}}_{k|k-1})(h(\mathcal{X}^i{}_k) - \hat{\mathbf{z}}_{k|k-1})^T W_i \tag{2.38}$$

$$\mathbf{S}_k \approx \mathbf{R}_k + \sum_{i=0}^{2n} (h(\mathcal{X}^i{}_k) - \hat{\mathbf{z}}_{k|k-1})(h(\mathcal{X}^i{}_k) - \hat{\mathbf{z}}_{k|k-1})^T W_i \tag{2.39}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{P}_{xz}\mathbf{S}_k{}^{-1}(\mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1}) \tag{2.40}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{P}_{xz}\mathbf{S}_k{}^{-1}\mathbf{P}_{xz}{}^T \tag{2.41}$$

where $\mathbf{z}_k$ is the measurement and $\mathbf{P}_{xz}$ is the approximated state-measurement cross covariance.

2.3.3 Data Association

Data association is the process of associating uncertain measurements to known tracks. One of the major difficulties in the application of multi-target tracking involves the problem of associating measurements with appropriate tracks, especially when there are missing measurements, unknown targets and false alarms (clutter). In a real driving scenario, during a radar scan, a large number of measurements are collected for which the sources are not known. The set of measurements has to be processed by the tracker. In point tracking, it is assumed that each target in the coverage of the scanning radar can produce a maximum of one measurement during a radar scan. However, with modern high resolution radar sensors, it is possible to track extended objects where each target can give rise to multiple measurements which have to be assigned to tracks. Assume that the multi-target tracker has produced a number of tracks, using radar measurements collected during earlier scans. To determine which measurements are likely candidates to be produced by a certain target, a technique called gating is used. A gate or association ellipse (ellipsoid) is positioned at the predicted state vector of the target in the state space [24]. The association function receives the output of the gating function and has to partition the sensor measurements into tracks. In cases where a single measurement is within the gate of a single track, the assignment can be immediately made. For closely spaced targets, it is more likely that conflict situations, as shown in Figure 2.17, will occur. In this figure, association gates, represented by a circle in the (x,y)-plane, are shown for two tracks. Only one measurement can be assigned to each of the targets, indicated by the predicted positions T_1 and T_2. An assignment problem has to be solved to determine the most likely distribution of the measurements $\{M1,...,M6\}$ to the set of targets $\{T1,T2\}$, new targets and false alarms. The association problem can be considered as an optimization problem in which some defined object function has to be minimized. The un-connected measurements do not originate from the same source and are mutually independent, representing false alarms or new targets. Theoretically, it is possible to generate all possible data association hypotheses. The next step is, to define appropriate probability density functions that the source of a measurement is an already

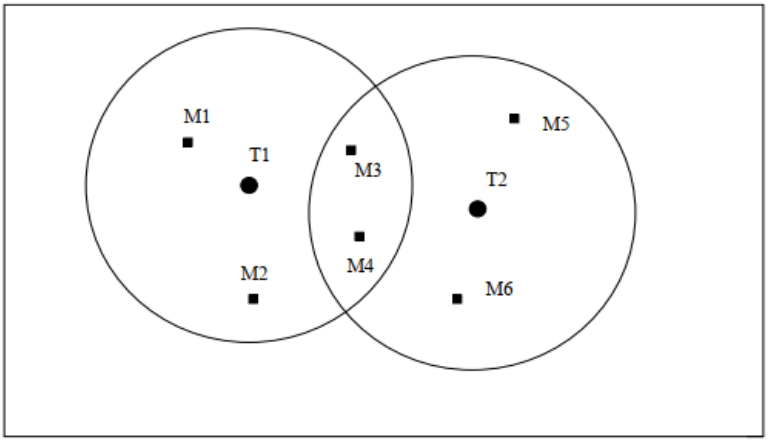


Figure 2.17: A gating example with two tracks T_1 and T_2. To assign measurements $M_1, \ldots, M_6$ to the tracks, circular association gates are positioned around each track. Only one measurement can be assigned to a track by solving an optimization problem [24].

known target, a new target or a false alarm. It is now possible to define a conditional probability function for a data association hypothesis, which expresses the likelihood of the hypothesis given all earlier processed measurements. The hypothesis with the highest probability value is considered to represent the best data association hypothesis. In the next subsections data association methods are discussed.

Data Association Methods

The Global Nearest Neighbors (GNN) method [24], as shown in Figure 2.18, considers all possible measurement to track pairings and generates the most likely assignment hypothesis for the different measurements in a data set. The assignment hypothesis is used to make irrevocable assignments which cannot be influenced by data collected in later scans. GNN produces only a single data association hypothesis Ω^k which is based on the cumulative data set of measurements up to scan k. In the all-neighbors data association approach, after each scan multiple hypotheses are formed and subsequently combined into a single hypothesis. The result is that the track estimate for a given track may contain contributions from more than one measurement. The essence is that all hypotheses are taken into account in the track. The Probabilistic Data Association (PDA) method [27] is based on the assumption that a single target has to be tracked in clutter.

Given that N measurements fall within the gate, PDA forms $N + 1$ hypotheses. The first one corresponds with the case that none of the measurements originated from the target. Using the N measurements and the probabilities that they originate from the target, a composite state estimate for the track is determined, using the extended first order Kalman filter. PDA is based on the assumption that a single target has to be tracked. The method can easily be extended to the multi-target tracking case, resulting in the Joint Probability Data Association (JPDA) method [28]. For complex scenarios with closely spaced targets, JPDA shows a tendency to coalesce the tracks. This is due to the fact that JPDA forms composite measurements as the measurements in the overlap region of two or more gates contribute to the state vector estimates

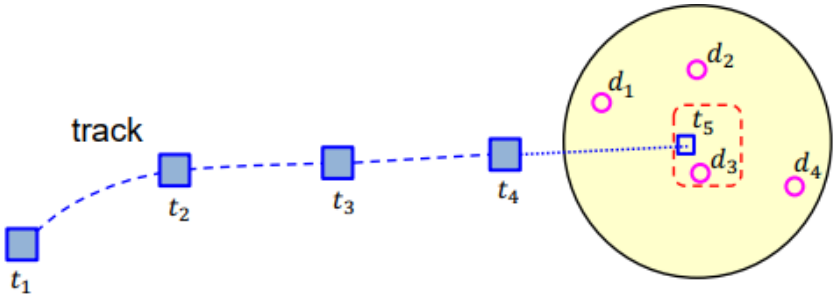


Figure 2.18: An example of global nearest neighbor tracking where each detection $d_1, \ldots, d_4$ in predicted track region t_5, as shown by a circle, is evaluated and best one is associated with the track t_5 based on Mahalanobis distance[8].

of all these targets. Due to the fact that PDA and JPDA average over all possible measurement-track associations, the produced updated track estimates may contain contributions from more than one measurement. Statistically, the tracks produced will probably be sub-optimal. Multiple Hypothesis Tracking (MHT) [29] is a deferred decision logic in which alternative data association hypotheses are formed whenever there are measurement to track conflict situations. Rather then combining these hypotheses, as in the JPDA method, the hypotheses are propagated in anticipation of subsequent data that may resolve the uncertainty. MHT offers the best performance in complicated scenarios under heavy clutter conditions. In terms of computation time, GNN is the most efficient data association algorithm.

2.3.4 Track Handling

Track handling, as shown in Figure 2.19, refers to the processes of track initiation, confirmation, and deletion. The GNN approach starts new tentative tracks on observations that are not assigned to existing tracks. The JPDA approach starts new tentative tracks on observations with probability of assignment lower than a specified threshold. The MHT approach starts new tentative tracks on observations whose distances to existing tracks are larger than a specified threshold. The tracker uses subsequent data to determine which of these newly initiated tracks are valid.

Once a tentative track is formed, a confirmation logic identifies the status of the track. Different logic can be used for track confirmation. When the history logic is used, a track is confirmed if the track has been assigned to a detection for at least M updates during the last N updates. In track score logic, a track is confirmed if its score is higher than a specified threshold. A higher track score means that the track is more likely to be valid. The score is the ratio of the probability that the track is from a real target to the probability that the track is false. In integrated logic, a track is confirmed if its integrated probability of existence is higher than a threshold.

A track is deleted if it is not updated within some reasonable time. The track deletion criteria are similar to the track confirmation criteria. Using history logic, a track is deleted if the track has not been assigned to a detection at least P times during last R updates. When track score

[8] source:`https://www.mathworks.com/videos/case-study-vision-and-radar-based-sensor-fusion-1497301769212.html`

logic is used, a track is deleted if its score decreases from the maximum score by a specified threshold. In integrated logic, a track is deleted if its integrated probability of existence is lower than a specified threshold.

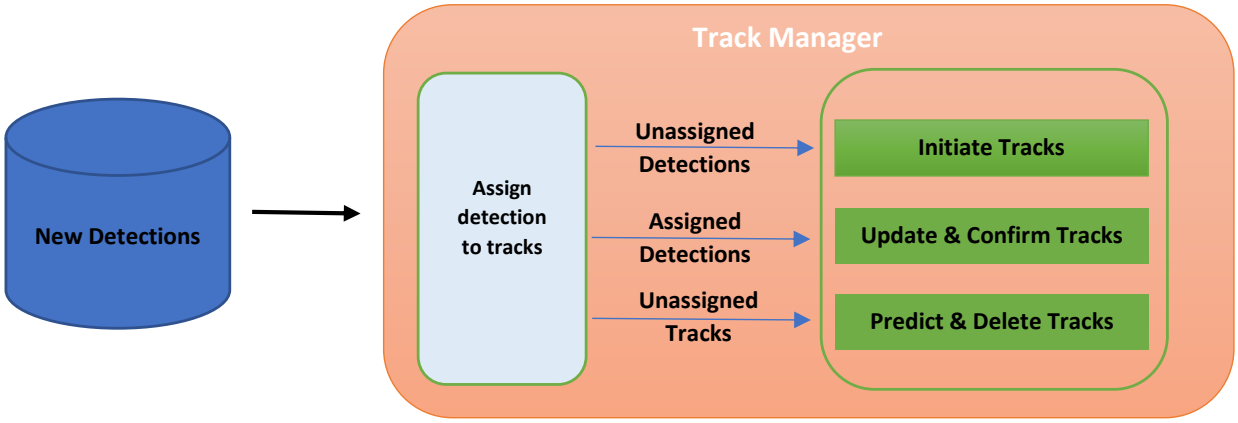


Figure 2.19: The track handling module is shown which assigns new detections to existing tracks or new tracks and deletes old tracks.

2.3.5 Extended Target Tracking

The tracking of an object that might occupy more than one sensor cell, as shown in Figure 2.20, leads to the extended target tracking problem. In extended target tracking [30], the objects give rise to a varying number of potentially noisy measurements from different spatially distributed measurement sources, also referred to as reflection points. The shape of the object is usually unknown and can even vary over time, and the objective is to recursively determine the shape of the object plus its localization parameters. Due to the non-linearity of the resulting estimation problem, tracking a single extended object is in general a highly complex problem for which elaborate nonlinear estimation techniques are required. The extended object state models where the object is located ((x, y) position in 2D or (x, y, z) position in 3D), where it is going (the motion parameters of the object, such as velocity, acceleration, heading and turn-rate), and its spatial extent (shape, size, as well as the orientation of the shape). Extended targets can be tracked by the Probability Hypothesis Density (PHD) filter. It is a multiple target filter for recursively estimating the number and the state of a set of targets given a set of observations. It is based on the Random Finite Set (RFS) formulation [31] in which the collection of individual targets is treated as a set-valued state and the collection of observations as a set-valued observation. PHD filter can operate in environments with false alarm and missed detection and it works by propagating in time the intensity of the target's RFS instead of the full multi-target posterior density. In literature, various implementations have been proposed and their performance compared using different levels of clutter and model uncertainty. The

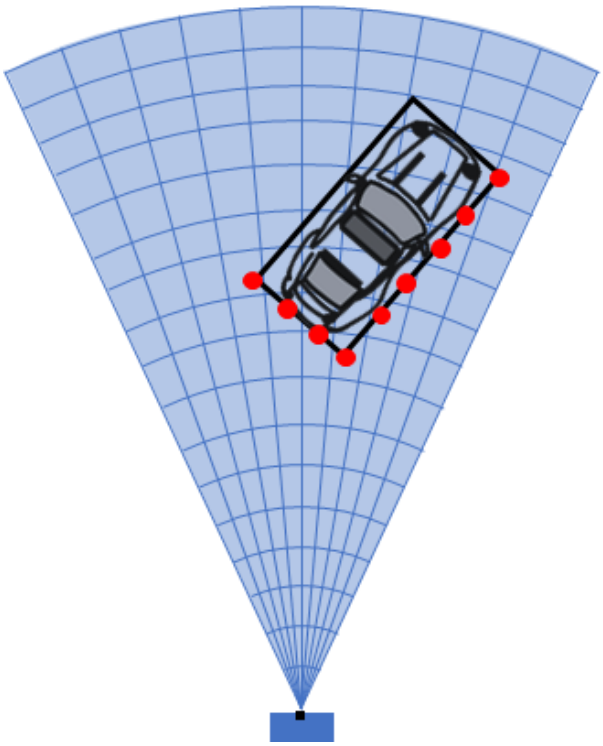


Figure 2.20: Example of an extended target that spans over multiple radar sensor cells resulting in multiple detections[9].

generic sequential Monte Carlo implementation (SMC-PHD) filter [32], generally suffers from a high computational cost as it requires a large number of particles and relies on clustering techniques to provide state estimates. The unreliability of estimates due to inaccuracy introduced by the clustering step and the computational complexity constitute its main drawbacks. To alleviate these problem a closed form solution to the PHD filter recursion, called Gaussian Mixture PHD (GM-PHD) [33] had been proposed. This approach does not require clustering procedures but, since it makes use of the Kalman filter equations, it is restricted to linear-Gaussian target dynamic. When targets show a mildly non-linear dynamic it is generally possible to rely on extensions for the GM-PHD filter using the extended Kalman filter (EK-PHD) or the unscented Kalman filter (UK-PHD) [34] or to use the Gaussian particle implementation of the PHD filter [35].

2.3.6 Normalized Innovation Square Test

The Normalized Innovation Squared (NIS) test is used to assess whether a Kalman filter's noise assumptions are consistent with realized measurements [36]. Consistency is an important property especially in filtering, where the current estimate is used as a prior distribution for future estimation. If an estimate is not consistent then it is overly optimistic about its precision, and

[9] source:`https://www.mathworks.com/help/driving/ug/extended-object-tracking.html`

new measurements have too little influence. In this case the filter can get stuck in an erroneous state. The NIS test can be applied online with real data, and does not require future data, repeated experiments, or knowledge of the true state. If the measurement noise covariance parameter in a Kalman filter is too small relative to the actual noise, the filter gives too much weight to measurements relative to the process model, and estimated state trajectories are overly erratic. On the other hand, if the parameter is too large, the filter gives too little weight to measurements, and its response is sluggish. Bar-Shalom and Li [37] presented the NIS consistency check for Kalman filter, and defined a filter estimate as consistent if its normalized innovation squared (NIS), given as $(\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}))^T \mathbf{S}_k{}^{-1} (\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}))$, follows a chi-square distribution with number of degrees of freedom equal to dimensions of the measurement vector.

3 Method

In this chapter, the main system model used in this thesis is explained. The ego vehicle and sensor coordinate systems are introduced, the driving scenario generation is explained and the structure of the dataset used to train the PointNet model is described. The target tracking algorithms developed in this thesis are described and the mathematical models are elaborately presented at the end of this chapter.

3.1 Simulation Model Overview

An overview of the simulated model is illustrated by Figure 3.1. The simulation model contains three distinct subsystems. The FMCW radar based subsystem, the PointNet based object detection and the target tracking subsystems. The FMCW radar subsystem consists of a transmitter, a receiver, a mixer and an analog-to-digital converter (A/D). A modulated signal is transmitted

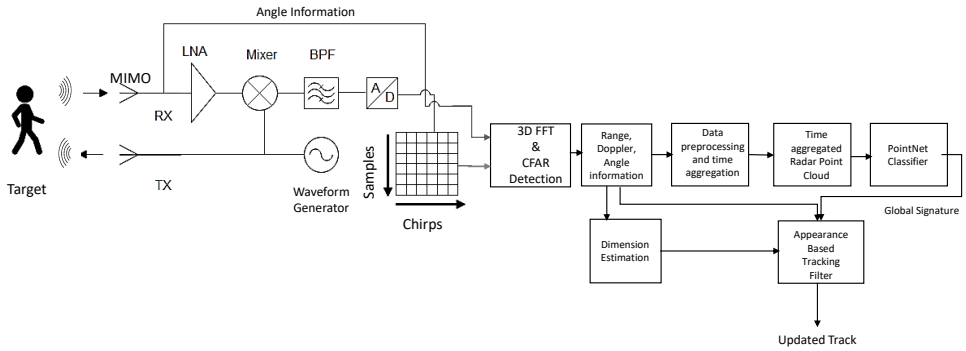


Figure 3.1: The simulation model used in this thesis. It consists of the FMCW radar transceiver and signal processing subsystem, the object detection subsystem and the tracking subsystem.

and received through antennas. The received signal is amplified by a low noise amplifier (LNA) and the transmitted and received signals are multiplied in the time domain. The resulting signal is band pass filtered (BPF) to obtain the immediate frequent (IF) signal. As explained in section 2.1.2, all target parameters of interest (i.e. range, velocity and angle) can be estimated

from the IF signal. This process in repeated for P consecutive pulses. Combining the signals of multiple receive antennas, a radar data cube with three dimensions, namely slow time, fast time and number of antennas, is obtained. In order to resolve targets in range, velocity or azimuth angle, a 3D FFT along the respective dimensions, is applied. A CFAR filter is used to identify target detections. The FMCW radar setup along with all simulation parameters is discussed in detail in section 3.3. The target dimensions (i.e. length and width) can be estimated from the position and angle information and is explained in section 3.7.

In the object detection subsystem, the target parameters are pre-processed to generate radar point clouds which are aggregated for multiple time stamps of the logged radar data to tackle the problem of sparsity. These time aggregated point clouds are fed to the PointNet classifier. Sections 3.8 and 3.10 describe the data pre-processing module and the PointNet architecture, respectively. The tracking filter obtains range velocity, angle and dimension information from the radar based subsystem and the global features representing the appearance of the target from the PointNet based object detection subsystem to track the targets in the driving scenario. The different filtering approaches are discussed in section 3.12.

3.2 Coordinate Systems

The world coordinate system is a fixed universal coordinate system in which all vehicles and their sensors are placed. A world coordinate system is important in global path planning, localization, mapping, and driving scenario simulation. The right-handed Cartesian world coordinate system defined in ISO 8855 is used, where the Z-axis points up from the ground and units are in meters.

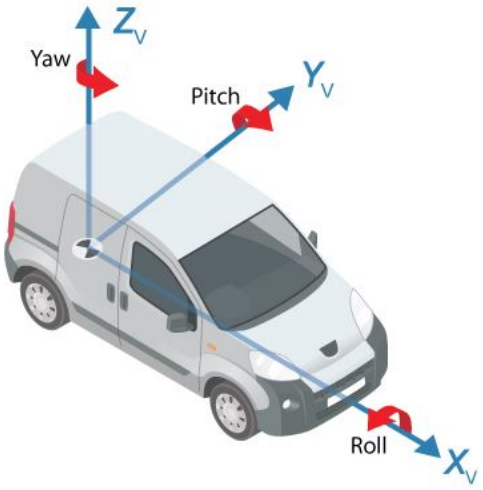


Figure 3.2: Vehicle coordinate system is anchored at the ego vehicle. The origin is on the ground, below the midpoint of the rear axle[1]. (X_Y,Y_V,Z_V) are the three vehicle coordinates.

[1] source:`https://www.mathworks.com/help/driving/ug/coordinate-systems.html`

The vehicle coordinate system (X_V, Y_V, Z_V), as shown in Figure 3.2, is anchored to the ego vehicle. The term ego vehicle refers to the vehicle that contains the sensors that perceive the environment around the vehicle. The X_V axis points forward from the vehicle, the Y_V axis points to the left, as viewed when facing forward, and the Z_V axis points up from the ground to maintain the right-handed coordinate system. The vehicle coordinate system follows the ISO 8855 convention for rotation. Each axis is positive in the clockwise direction, when looking in the positive direction of that axis. Locations in the vehicle coordinate system are expressed in world units, that is, meters. Values returned by individual sensors are transformed into the vehicle coordinate system so that they can be placed in a unified frame of reference. The state of the vehicle can be described using the pose of the vehicle. The steering angle of the vehicle is positive in the counterclockwise direction, as shown in Figure 3.3.

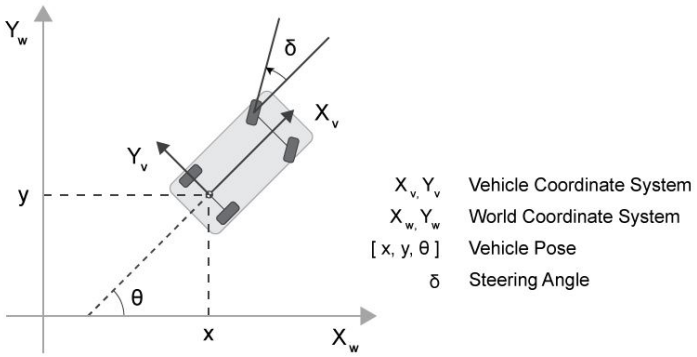
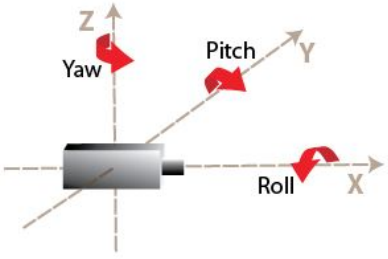
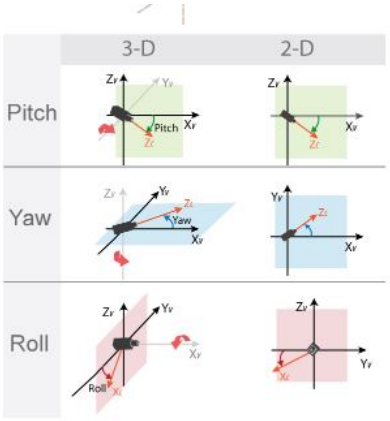


Figure 3.3: The vehicle Pose is given by (x,y,θ) where θ is the angle between the world and vehicle coordinates, and the steering angle is δ. (X_Y,Y_V) indicate the vehicle coordinate system and (X_W,Y_W) indicate the world coordinate system[1].



(a) The sensor coordinate system.



(b) yaw, pitch, and roll angles in sensor coordinate system.

Figure 3.4: Sensor coordinate system is shown in (a). (b) shows the convention for yaw, pitch, and roll angles of sensors. (X_V,Y_V,Z_V) indicate vehicle coordinate system and (X_C,Y_C,Z_C) indicate sensor coordinate system[1].

An automated driving system can contain sensors located anywhere on or in the vehicle. The location of each sensor contains an origin of its coordinate system. The yaw, pitch, and roll angles of sensors follow an ISO convention. These angles have positive clockwise directions when looking in the positive direction of the *Z*, *Y*, and *X* axes, respectively, as shown in Figure 3.4. In this thesis, there is one radar sensor located in front of the ego car and reports detections in radial coordinates, that is, in terms of range, doppler velocity and angle of arrival.

3.3 Radar Sensor Setup

The simulation model in this thesis has one radar sensor mounted at the center of the front grill of the ego vehicle, as shown in Figure 3.5. The radar has a field of view of 160°. The radar sensor operates at a center frequency of 77 GHz. This frequency is commonly used by automotive radars. For short-range operation, the radar must detect vehicles at a maximum range of 50 m in front of the ego vehicle. The radar can resolve objects in range that are at least 0.16 m apart. Because this is a forward-facing radar application, the radar also needs to handle targets with large closing speeds, as high as 230 kmph. Short pulses are needed to attain maximum range resolution and it depends on round-trip time of the signals propagated between the transmitters and receivers, consequently, there is always as a trade-off between the range and velocity resolution. All radar simulation parameters have been listed in Table 3.1. As the FMCW signal occupies wide bandwidth signals, the sampling rate should be at least two times the bandwidth. However, this high sample rate imposes stress on the analog to digital converters due to the power constrains. To reduce the burden on analog to digital converter and to accommodate the power constrains, the sample rate is considered to be the maximum of twice the maximum beat frequency and the sweep bandwidth in this work.

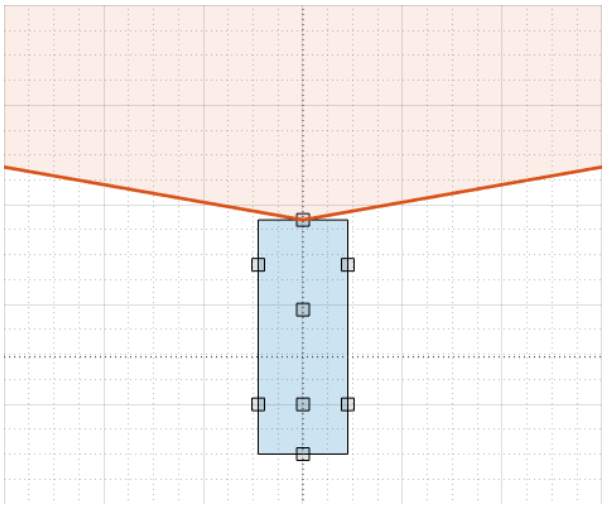


Figure 3.5: An illustration of the radar set up. The area between the solid orange lines illustrate the field of view of the radar.

To model a realistic system setup while keeping the computations simple, a complete radar transceiver, similar to the one described in Figure 3.1, is simulated using the Phased Array

System Toolbox™. The radar is designed to use an FMCW waveform with a bandwidth of 937 MHz. The radar transmits with a peak power of 3 mW via a single isotropic antenna, enabling it to cover a larger area than on receiver side. On the receiver side, a uniform linear array (ULA) with 6 identical receive antennas with half wavelength inter-element spacing is used. The free space channel is used to model the propagation of the transmitted and received radar signals. In a free space model, the radar energy propagates along a direct line-of-sight between the radar and target vehicles.

Table 3.1: FMCW Radar Simulation Parameters used in this thesis.

Parameter	Value
Center frequency of Radar	77 GHz
FMCW Sweep Bandwidth	937 MHz
Sweep time	10μs
Number of sweeps	192
Maximum beat frequency	31 MHz
Maximum received intermediate frequency	31 MHz
Sample Rate	937 MHz
Number of fast samples	625
Number of slow samples	192
Unambiguous range	50 m
Range resolution	16 cm
Unambiguous velocity	230 kmph
Maximum Doppler shift	33 KHz
Number of Tx antennas	1
Number of Rx antennas	6
Antenna element spacing	$\lambda/2$
Transmit power	3 mW
Tx/Rx antenna gain	27 dB
Receiver noise figure	4.5
Radar field of view	-80°to +80°

3.4 Radar Signal Processing Chain

The radar collects multiple sweeps of the waveform on each of the linear phased array antenna elements. These collected sweeps form a radar data cube. These sweeps are coherently processed along the fast and slow time dimensions of the data cube to estimate the range and Doppler information of the targets. The root multiple signal classification (root MUSIC) direction of arrival estimator is used to estimate the direction of arrival of signals detected on a linear phased array. A Hanning window is used to suppress the large side lobes produced by the targets when they are close to the radar. Detections are identified in the processed range-Doppler map using a CFAR detector. The simulation parameters for the radar signal processing chain are enlisted in Table 3.2.

Table 3.2: FMCW Radar Signal Processing Parameters used in this thesis.

Parameter	Value
CFAR threshold factor	20
Number of FFT points (Range)	1024
Number of FFT points (Doppler)	256
RMS resolution for range estimation	0.55

3.5 Driving Scenario Generation

The Driving Scenario Designer app in MATLAB®'s Automated Driving Toolbox™ enables the designing of synthetic driving scenarios for testing autonomous driving systems. Using the app, it is possible to create road and actor models using a drag-and-drop interface and configure sensors mounted on the ego vehicle. These sensors are used to simulate detections of actors and lane boundaries in the scenario and to generate point cloud data from a scenario. The main benefit of using scenario generation and sensor simulation over sensor recording is the ability to create rare and potentially dangerous events and test the vehicle algorithms with them. The following are the steps to create a simple driving scenario:

- Add a curved road to the driving scenario.
- Add lanes to the road. By default, the road is one-way and has solid lane markings on either side to indicate the shoulder. In this thesis, only two-line driving scenarios are used.
- Add a ego vehicle at one end of the road and set its initial position. Set the driving trajectory of the vehicle.
- Adjust the speed of the vehicle as it passes between way-points. Increase the speed of the vehicle for the straight segments and decrease its speed for the curved segments.
- Add a sensor to the ego vehicle. On the sensor canvas, select the location of the sensor.
- Similarly, add other actors to the scenario like cars, bicycles, trucks and pedestrians.

3.6 Visualizing Driving Scenario and Radar Data

Different plots have been used in this thesis to visualize the driving scenario and generated radar detections. A chase plot, as shown in Figure 3.6, plots a driving scenario from the perspective of a vehicle. It has an ego-centric projective perspective, where the view is positioned immediately behind the vehicle.

A bird's eye plot, illustrated in Figure 3.7, plots the bird's eye view or an elevated view of the 2D driving scenario from above, with a perspective as though the observer were a bird, in the close vicinity of the ego vehicle.

A bird's eye plot can display vehicles outlines, radar detections, lane marking and boundaries, point cloud, object tracking results etc. All of the vehicles, detections, and tracks are shown

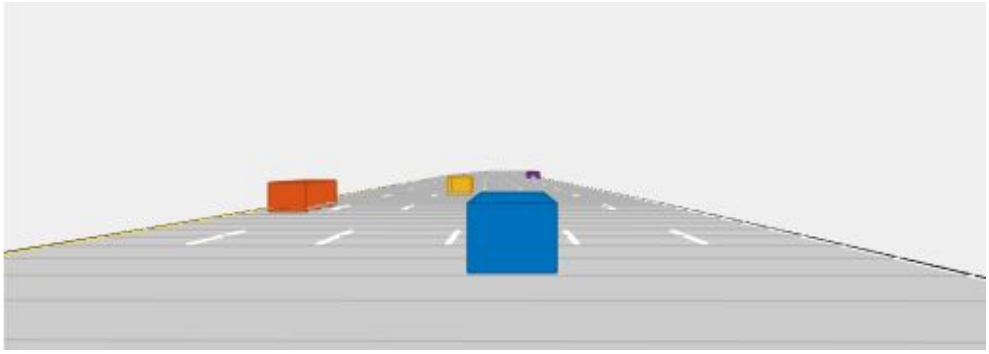


Figure 3.6: A chase plot of a driving scenario with an ego vehicle shown in blue and targets shown in orange, yellow and purple.

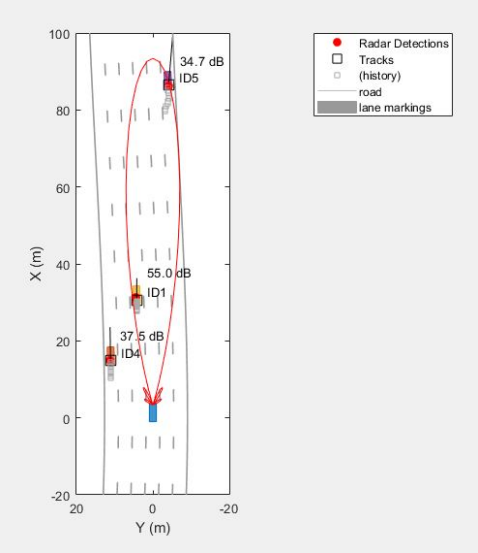
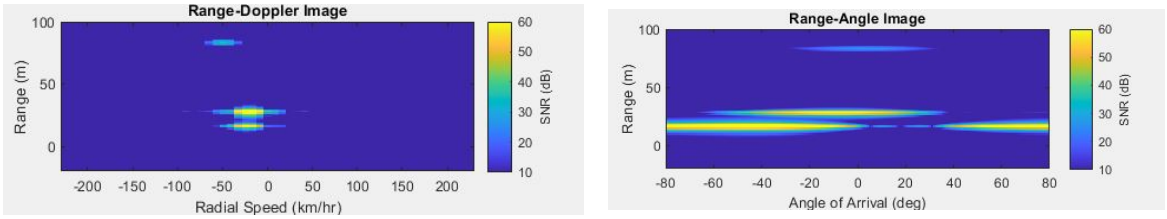


Figure 3.7: Bird's eye plot of the driving scenario illustrated by Figure 3.6. The ego vehicle is shown in blue, the target tracks along with track IDs and history of tracks is also shown.

in the ego vehicle's coordinate reference frame. The estimated signal-to-noise ratio (SNR) for each radar measurement is displayed next to each detection. The vehicle location estimated by the tracker is shown in the plot. Different tracks are identified with unique track IDs. The tracker's estimated velocity for each vehicle is shown as a line pointing in the direction of the vehicle's velocity. The length of the line corresponds to the estimated speed, with longer lines denoting vehicles with higher speeds relative to the ego vehicle. In this thesis, the radar images generated after signal processing are displayed in the form of range-Doppler and range-angle maps, as illustrated by Figures 3.8(a) and 3.8(b), respectively. The range-Doppler plot shows how the received radar echos from the target vehicles are distributed in range and radial speed. The range-angle plot shows how the received target echos are spatially distributed in range and azimuth angle of arrival.



(a) Range-Doppler map.

(b) Range-angle map.

Figure 3.8: Radar images for the driving scenario shown in Figure 3.6.

3.7 Dimension Estimation of Radar Targets

In this thesis, the dimension (length and width) of the VRUs is approximately estimated at each simulation step, using the range, Doppler, angle information of targets. Detections which occur at adjacent range and Doppler cells within the range-Doppler image are associated to a single detection cluster. Each detection cluster is assigned a unique cluster ID which is used to identify the detections assigned to that cluster. The minimum and maximum range bins for each cluster is calculated. At each simulation time step, using the minimum and maximum range and the angle of arrival information for a cluster, the 2D Cartesian coordinates of the corners of the rectangle approximating the cluster is estimated.

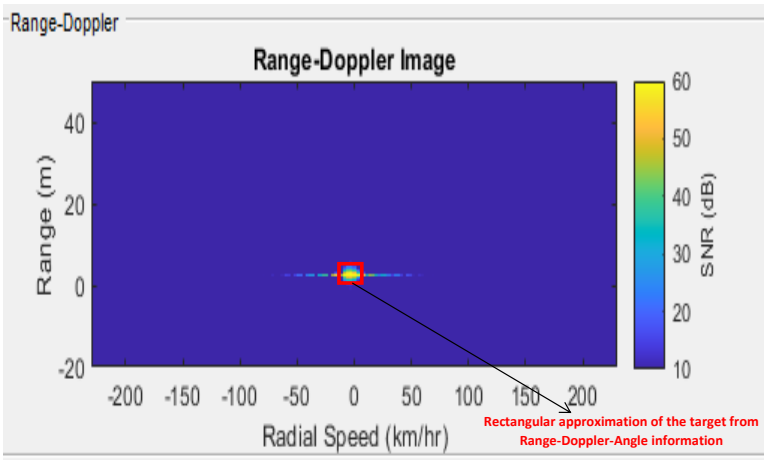
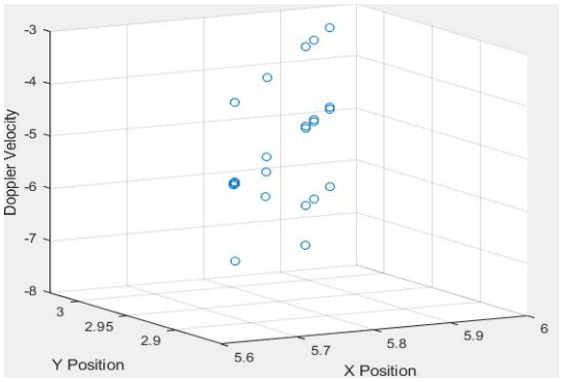


Figure 3.9: Illustration of dimension estimation of targets using rangle, Doppler and angle information. The red rectangle approximates the radar target.

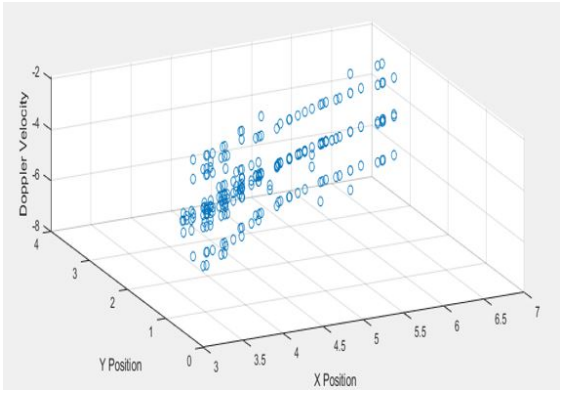
The dimension estimation of the target by this logic is not very accurate and depends on the number of detections generated by a target at any particular time. If a target leads to very few or missed detections in one simulation time step, the estimated dimensions would be very small or zero. Despite this shortcoming, it is worth estimating the target dimension to investigate whether this additional input to the tracking filter will aid the target tracking algorithm to assign and track detections better in ambiguous scenarios where a target is occluded partially or completely, or when targets are in close vicinity of one another etc.

3.8 Radar Point Cloud Generation

Each radar detection contains information of range, azimuth angle and Doppler velocity of targets. This information is processed to generate radar point clouds. Since the ego vehicle is moving, the tracking coordinate system is in motion. A compensation for the ego motion is required to describe targets in the moving coordinate system. Radar point clouds are generated by concatenating X and Y positions with the Doppler velocity, for each detection in one time frame of the logged radar data. The Cartesian X and Y coordinates for detections can be obtained from the polar range and angle measurements. Thus, we have three dimensional point clouds, as illustrated in Figure 3.10(a). In order to cope with the issue of sparsity of the point cloud data, point clouds from ten consecutive time frames of the logged radar data is aggregated in this thesis. Figure 3.10(b) depicts how data aggregation changes the form and information content of a point cloud. Finer structures like curves and lines are more accentuated when the number of frames over with the point cloud data is aggregated, is increased.



(a) Point cloud without data aggregation.



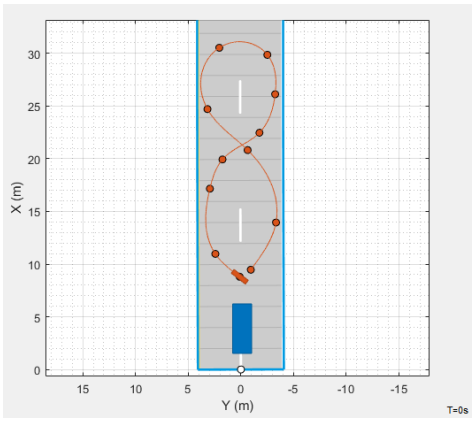
(b) Point cloud with data aggregation.

Figure 3.10: Illustration of a 3D radar point cloud. The coordinates are X and Y positional coordinates and Doppler velocity. (a) and (b) show how data aggregation changes the form and information content of a point cloud.

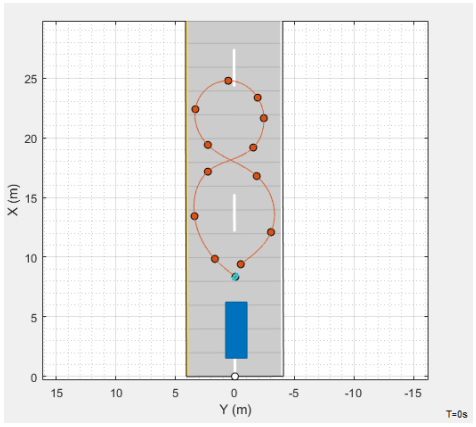
3.9 Dataset Characteristics

The dataset for training the PointNet model is generated synthetically in MATLAB® using the Automated Driving Toolbox™. There are five simple driving scenarios, as shown in Figures 3.11,3.12,3.13 and 3.14, involving a ego vehicle and a VRU in a two-lane highway. The target can be a pedestrian or a bicyclist. The ego vehicle is a car of type sedan with length 4.7 m, width 1.8 m and height 1.4 m. The bicyclist target is 1.7 m in length, 0.45 m in width and 1.7 m in height. The pedestrian actor is 0.24 m in length, 0.45 m in width and 1.7 m in height. The ego vehicle is always stationery and the target moves in front of it following different trajectories. A bicyclist travel can travel with speeds from 9 kmph to 18 kmph and the pedestrian user can have speeds between 3.6 kmph and 10 kmph, without acceleration. In this work, highly non-linear trajectories of the target have been considered so as to cover as many viewing angles from the ego vehicle to the target vehicle as possible. The intuition behind having these trajectories is

to train the PointNet model to detect the targets even from the most crooked viewing angles. Training the PointNet on simplified trajectories create a biased model with low generalization accuracy. The simulation parameters for dataset generation are enlisted in Table 3.3.

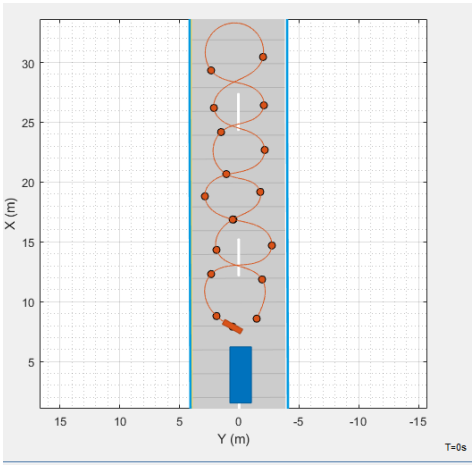


(a) Scenario 1 with bicyclist target.

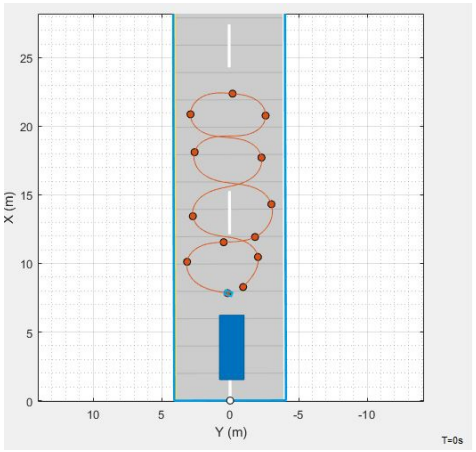


(b) Scenario 1 with pedestrian target.

Figure 3.11: Driving scenario 1 from dataset.



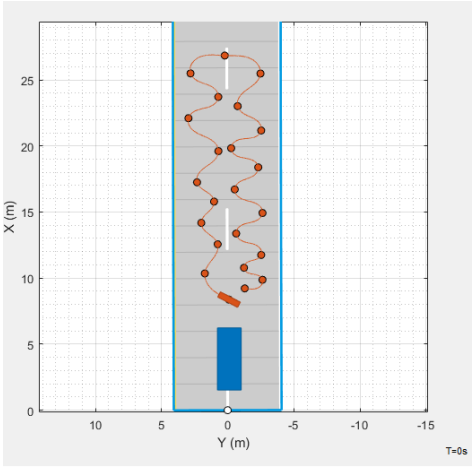
(a) Scenario 2 with bicyclist target.



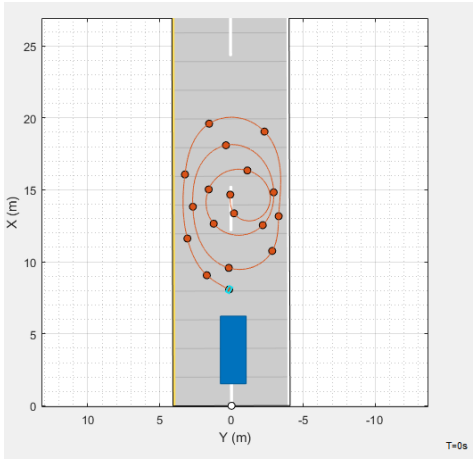
(b) Scenario 2 with pedestrian target.

Figure 3.12: Driving scenario 2 from dataset.

Each scenario simulation runs for 15 seconds with a sample time of 0.1 second. For each simulation, there are 150 frames in total and every 10 consecutive frames of this logged radar data are concatenated to generate time aggregated radar point clouds. Thus, for each simulation, 15 point clouds are generated. For every driving scenario, multiple logging sessions are made. These logging sessions contain variations in the relative velocity between the ego vehicle and the VRU. The same driving scenarios are used for both pedestrian and bicycle road users. The entire dataset has 900 examples of aggregated point cloud data. The entire dataset is split into the training set D_T containing 85% of the examples in the dataset and validation dataset D_V, containing the remaining 15% examples. Since the scenarios are relatively simple, it can at most be considered to be simulations of traffic scenarios in a low traffic sub-urban environment.



(a) Scenario 3 with bicyclist target.



(b) Scenario 4 with pedestrian target.

Figure 3.13: Driving scenario 3 and 4 from dataset.

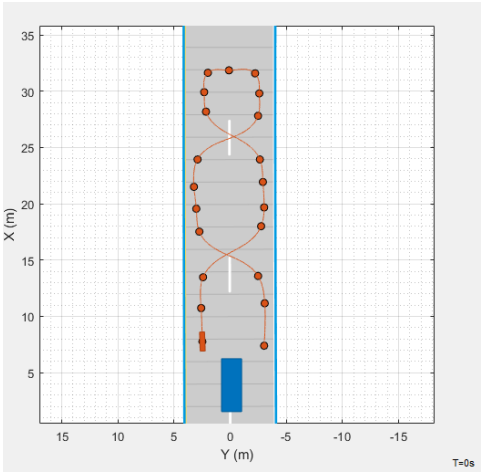


Figure 3.14: Driving scenario 5 with bicyclist target from dataset.

Table 3.3: Simulation Parameters used to generate the training dataset in this thesis.

Parameter	Value
Simulation time	15 sec
Sample or frame time	0.1 sec
Ego car dimension (Length X Width X Height)	4.7 m X 1.8 m X 1.4 m
Pedestrian dimension (Length X Width X Height)	0.24 m X 0.45 m X 1.7 m
Bicyclist dimension (Length X Width X Height)	1.7 m X 0.45 m X 1.7 m
Sensor location in ego coordinates (in m)	(3.4,0,0.2)
No. of frames aggregated for point cloud generation	10
Ego car speed	0 kmph
Pedestrian speed	3.6 to 10 kmph
Bicyclist speed	9 to 18 kmph

3.10 PointNet Model

The PointNet architecture used in this thesis is based on the architecture described in [7]. Since the training set is small, the number of parameters have been reduced as compared to the original PointNet architecture so as to reduce over-fitting. The PointNet implementation still keeps the main architecture of the original PointNet. Only the PointNet classification architecture is implemented. The segmentation architecture has not been explored in this thesis.

The PointNet architecture is illustrated by Figure 3.15. The PointNet model consists of two components. The first component is a point cloud encoder that learns to encode sparse point cloud data into a dense feature vector. The second component is a classifier that predicts the categorical class of each encoded point cloud. The network takes n points from a point cloud as input. Each point in the point cloud is represented in three dimensions, that is, x and y positions and Doppler velocity. Thus, the input size is nX3.

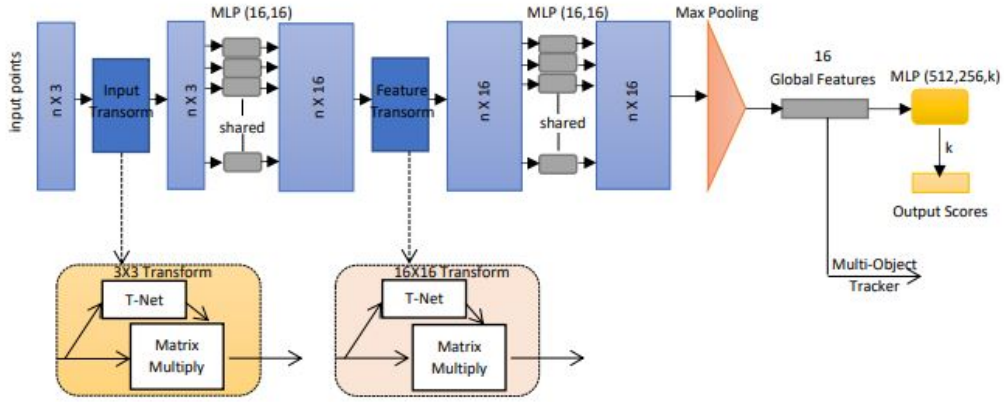


Figure 3.15: PointNet architecture for point cloud classification. The input is n 3D point clouds. MLP(16,16) indicates that there are two layers of multi-layer perceptrons with 16 neurons in each layer. 16 global features are learned by max pooling. The classification MLP has thre layers with 512, 256 and k neurons. In this thesis, k is 2. The output is classification scores for 2 classes.

The PointNet applies an input transformation so as to align the input points. The T-Net, which is a smaller version of the original PointNet, predicts an affine transformation matrix of size 3X3 to align the points in the input point cloud. Each point is multiplied with the transformation matrix and is used as input to the shared MLP with two layers with 16 neurons in each layer. The shared MLP model is implemented using a series of convolution, batch normalization, and ReLU operations. The convolution operation is configured such that the weights are shared across the input point cloud. The size of the input to this MLP block is nX3, as indicated in the figure. Features learned by the MLP block are aligned by a second T-Net module which predicts a 16X16 feature transformation matrix. Each feature point is multiplied with the transformation matrix and is fed to a second shared MLP having the same structure as the first one. The input size to this MLP block is nX16 and the output from this

layer is max-pooled to obtain the global signature for the input point cloud. The number of global features learned is 16. A MLP classifier is trained on the shape of the global features for classification. After the last perceptron, a fully connected operation maps the classification features to the number of classes, which is two, pedestrian and bicyclist. A softmax activation is used to normalize the output into a probability distribution of labels. The global features learned by the PointNet are fed as additional input to the multi-object tracker subsystem.

3.10.1 Data Preprocessing For PointNet

Imbalance in the training data in terms of representation of both classes can make the classifier biased towards one class and prevents the training of a robust classifier. Class imbalance is overcome by oversampling the infrequent class. In this thesis, over-sampling is achieved by randomly duplicating the infrequent class. Duplicating the files to address class imbalance increases the likelihood of over-fitting the network because much of the training data is identical. To offset this effect, data augmentation is applied to the training data, by random affine transformation to the point cloud, randomly removing points, and randomly jittering points with Gaussian noise. Due to the limited number of training samples in the dataset, the model easily overfits the training data in the absence of the augmentation.

The full sized point clouds being too computational demanding, a predefined number of points are to be randomly sampled from each point cloud. Because of the large amount of intra-class and inter-class variability in the number of points per class, choosing a value that fits all classes is difficult. One heuristic is to choose enough points to adequately capture the shape of the objects while not increasing the computational cost by processing too many points. The final pre-processing step is to normalize the point cloud data between 0 and 1 to account for large differences in the range of data values, which can otherwise hinder the convergence of the network during training. Normalization is used for the point cloud data in the training and validation sets. The point clouds in each dataset is a time series and therefore consecutive point clouds might be similar since they are close to each other in time. When creating a training and validation set it is undesirable to have neighboring point clouds in time in both the validation and training datasets, as this might yield an unfairly high accuracy. Therefore the data points have been shuffled by logging session so that data points consecutive in time never occur consecutively both the training set and validation set.

3.11 Training and Validation

For training and validation purpose, the dataset is divided into two parts: training and validation data. The training dataset D_T is used to train the model. The evaluation of the model during training process uses the validation dataset D_V. The validation data consists of same driving maneuvers as the training data. For example, if a maneuver was driven five times, then four of them are added to the training data and the rest to the validation data. This ensures that all trained driving maneuvers are covered within the validation data. When the trained PointNet is integrated in the system model, it makes prediction for driving maneuvers which differ from the trained maneuvers. This is done to to show the generalization ability of models after the training process.

Training is performed with cross-entropy loss to optimize the classification PointNet. The loss includes a regularization term designed to ensure the feature transformation matrix predicted by the PointNet encoder is approximately orthogonal. During training, dropout is applied after each perceptron operation. To guarantee a fixed number of input points, sampling is performed. For classification, 80 radar targets from each point cloud are drawn. When testing the object detector, the same sample size is used. Data augmentation is a helpful concept to avoid model over-fitting. Hence, for all radar targets of a point cloud, data augmentation is applied during training. The point cloud is perturbed using random Gaussian noise with zero mean and standard deviation of 0.02.

Training of the model uses Adam optimizer and performs batch normalization for all layers excluding the last classification layer. The initial learning rate is chosen to be 0.0001. The model is trained for 10 epochs without any drop in the learning rate. The gradient decay factor for the Adam optimizer is 0.9. At the end of each epoch, the model is evaluated against the validation data set and confusion metrics are collected to measure classification accuracy as training progresses. Since the learning data set is small, the model was trained on a 2.4 GHz CPU and took 5 minutes for a batch size of 16.

3.12 Multi-Object Tracking System

In this thesis, three different multi-object tracking algorithms are implemented and their performances are compared against one another. The first implementation is a standard non-linear Kalman filter performing point object tracking on the radar data. It tracks the localization parameters, namely, position, velocity and azimuth angle of point targets. The second tracker implementation, modifies the standard non-linear Kalman filter equations to linearly track the length and width of targets along with their localization parameters. The third implementation is an extension of the second filter and has the additional capability of tracking target appearances by linearly filtering the global target features learned by the PointNet.

3.12.1 Clustering

In conventional tracking approaches such as Kalman filtering, tracked objects are assumed to return one detection per sensor scan. With the development of sensors that have better resolution, such as a high-resolution radar, the sensors typically return more than one detection of an object. Thus, extended objects present new challenges to conventional trackers. Cluster analysis is the task of dividing a set of objects into groups, such that similar objects are placed in the same group. Clustering of the sensor data provides the conventional trackers with a single detection per object. Clustering avoids having more tracks that the number of objects. The clustering of data points can be done directly on detections before feeding clusters of detections to the tracker. Alternatively, point targets can be tracked and the tracks that seem to originate from the same target are clustered. In this thesis, detections in adjacent cells of the range Doppler map are clustered together. Range, Doppler and velocity estimation is performed on the clustered detections to estimate the range, Doppler velocity and angle of arrival for the clusters as a whole along with their variances. The algorithm does so by finding the indices of the largest response value in the cluster and fits a quadratic curve to the magnitudes of the range, Doppler or angle response at this detection and finds the location of the peak.

3.12.2 Localization Based Filtering Algorithm

A filtering algorithm estimates a state vector comprising the parameters of the target, such as position and velocity, based on a measurement model. In target tracking problem, target dynamics is commonly modeled in the radar Cartesian coordinate system, while the measurements are available only in the natural sensor coordinate system, for most cases in polar coordinates. Tracking the targets in Cartesian coordinate system with the measurements described in the sensor coordinate system is a nonlinear state estimation problem. Multiple approaches exist to deal with this nonlinear tracking problem. The approach used in this thesis is tracking in mixed coordinate. This is the most widely used approach. The target dynamics and the measurements are modeled as a nonlinear function. Target state and process noise are in the Cartesian coordinate, yet measurement and its noise are given in the sensor coordinate. Two nonlinear filtering techniques, the extended Kalman filters (EKF) and the unscented Kalman filters (UKF) have been applied here for this framework.

Since a point moving in the two-dimensional physical world can be described by its two-dimensional position and velocity vectors, the state vector of of such a point in the Cartesian coordinate system can be given as $\mathbf{x}_k = [x, y, \dot{x}, \dot{y}]^T$, where $(x,y)^T$ are the position coordinates and $[\dot{x}, \dot{y}]^T$ is the velocity vector. If measurements are available every Δt seconds, assuming a constant velocity model, the state transition from timestep $k-1$ to k is given as

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{w}_k \tag{3.1}$$

where $\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $\mathbf{w}_k$ is a white noise process, with zero mean and standard deviation σ_w and has a "small" effect on $\mathbf{x}_k$, that accounts for unpredictable modeling errors due to turbulence, etc. At each time step, a noisy measurement of the true position of the target is made. Therefore, the target dynamics and measurements are modeled by

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{3.2}$$

where the target state and process noise are in the Cartesian coordinates, but measurement and its additive noise are in the sensor coordinates. The mapping function h between Cartesian and polar coordinates is gives as

$$h\left(\mathbf{x}_k\right) = \begin{bmatrix} \rho \\ \phi \\ \dot{\rho} \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan\left(y/x\right) \\ \frac{x\dot{x}+y\dot{y}}{\sqrt{x^2+y^2}} \end{bmatrix} \tag{3.3}$$

where ρ, ϕ and $\dot{\rho}$ are referred to as the range, bearing angle and range rate, respectively. The measurement noise $\mathbf{v}_k$ is Gaussian, with zero mean and known standard deviations $\sigma_\rho, \sigma_\phi, \sigma_{\dot{\rho}}$, respectively. In order to apply the extended Kalman filter eqs. (2.26) to (2.30) to this mixed model, the function $h(\mathbf{x}_k)$ has to be linearized and the Jacobian matrix $\mathbf{H}_k$ is given as

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial \dot{x}} & \frac{\partial \rho}{\partial \dot{y}} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \dot{x}} & \frac{\partial \phi}{\partial \dot{y}} \\ \frac{\partial \dot{\rho}}{\partial x} & \frac{\partial \dot{\rho}}{\partial y} & \frac{\partial \dot{\rho}}{\partial \dot{x}} & \frac{\partial \dot{\rho}}{\partial \dot{y}} \end{bmatrix} \tag{3.4}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \\ \frac{-y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x-\dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} \end{bmatrix} \tag{3.5}$$

In order to apply the unscented Kalman filter to this model, the matrix $\mathbf{F}$ and the mapping function h, given by eqn.(3.3) have to be substituted in eqs. (2.34) to (2.39).

3.12.3 Dimension and Localization Based Filtering Algorithm

In this modified tracking approach, the state vector of a target in the Cartesian coordinate system can be given as $\mathbf{x}_k = [x, y, \dot{x}, \dot{y}, w, h]^T$, where h and w are the estimated height and width of the target. If measurements are available every Δt seconds, again assuming a constant velocity model, the state transition from timestep $k-1$ to k is given by eqn.(3.1), where $\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$. The target dynamics and measurements are modeled by

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{3.6}$$

. The radar sensor measurements, in this model, are obtained in mixed coordinates, that is, the measurement of range ρ, angle ϕ, and velocity $\dot{\rho}$ at available in polar coordinates but the measurements of width w and height h are available in cartesian coordinates. The measurement function h that performs mapping between the state vector in Cartesian and the measurement vector in mixed coordinates is gives as

$$h\left(\mathbf{x}_k\right) = \begin{bmatrix} \rho \\ \phi \\ \dot{\rho} \\ w \\ h \end{bmatrix} = \begin{bmatrix} \sqrt{x^2+y^2} \\ \arctan\left(y/x\right) \\ \frac{x\dot{x}+y\dot{y}}{\sqrt{x^2+y^2}} \\ w \\ h \end{bmatrix} \tag{3.7}$$

The measurement noise $\mathbf{v}_k$ is Gaussian, with zero mean and known standard deviations $\sigma_\rho, \sigma_\phi, \sigma_{\dot{\rho}}$, 1, 1 respectively. In order to apply the extended Kalman filter eqs. (2.26) to (2.30) to this mixed model, the function $h(\mathbf{x}_k)$ has to be linearized and the Jacobian matrix $\mathbf{H}_k$ is given as

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial\rho}{\partial x} & \frac{\partial\rho}{\partial y} & \frac{\partial\rho}{\partial\dot{x}} & \frac{\partial\rho}{\partial\dot{y}} & \frac{\partial\rho}{\partial w} & \frac{\partial\rho}{\partial h} \\ \frac{\partial\phi}{\partial x} & \frac{\partial\phi}{\partial y} & \frac{\partial\phi}{\partial\dot{x}} & \frac{\partial\phi}{\partial\dot{y}} & \frac{\partial\phi}{\partial w} & \frac{\partial\phi}{\partial h} \\ \frac{\partial\dot{\rho}}{\partial x} & \frac{\partial\dot{\rho}}{\partial y} & \frac{\partial\dot{\rho}}{\partial\dot{x}} & \frac{\partial\dot{\rho}}{\partial\dot{y}} & \frac{\partial\dot{\rho}}{\partial w} & \frac{\partial\dot{\rho}}{\partial h} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial\dot{x}} & \frac{\partial w}{\partial\dot{y}} & 1 & \frac{\partial w}{\partial h} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial\dot{x}} & \frac{\partial h}{\partial\dot{y}} & \frac{\partial h}{\partial w} & 1 \end{bmatrix} \tag{3.8}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 & 0 & 0 \\ \frac{-y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 \\ \frac{y(\dot{x}y-\dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x-\dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

In order to apply the unscented Kalman filter to this model, the matrix $\mathbf{F}$ and the mapping function h, given by eqn.(3.7) have to be substituted in eqs. (2.34) to (2.39).

3.12.4 Dimension, Localization and Appearance Based Filtering Algorithm

In this tracking approach, the state vector of a target in the Cartesian coordinate system is given as $\mathbf{x}_k = [x, y, \dot{x}, \dot{y}, w, h, f_1, f_2, \dots, f_{16}]^T$, where $f_1, f_2, \dots, f_{16}$ are the global features of the target learned by the PointNet. If measurements are available every Δt seconds, assuming constant velocity, the state transition from time step $k-1$ to k is given by eqn.(3.1), where $\mathbf{F}$ is given as $\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \cdots & 0 \\ 0 & 1 & 0 & \Delta t & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$. The target dynamics and measurements are modeled by

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{3.10}$$

The measurement function h that performs mapping between the state vector in Cartesian and the measurement vector in mixed coordinates is gives as

$$h(\mathbf{x}_k) = \begin{bmatrix} \rho \\ \phi \\ \dot{\rho} \\ w \\ h \\ f_1 \\ \vdots \\ f_{16} \end{bmatrix} = \begin{bmatrix} \sqrt{x^2+y^2} \\ \arctan(y/x) \\ \frac{x\dot{x}+y\dot{y}}{\sqrt{x^2+y^2}} \\ w \\ h \\ f_1 \\ \vdots \\ f_{16} \end{bmatrix} \tag{3.11}$$

The measurement noise $\mathbf{v}_k$ is Gaussian, with zero mean and known standard deviations $\sigma_\rho, \sigma_\phi, \sigma_{\dot{\rho}}$, to account for the error in range, angle and range rate respectively. Unit standard deviation is assumed for all other measurements. In order to apply the extended Kalman filter eqs. (2.26) to (2.30) to this mixed model, the function $h(\mathbf{x}_k)$ has to be linearized and the

Jacobian matrix $\mathbf{H}_k$ is given as

$$\mathbf{H}_k = \begin{bmatrix} \frac{\partial \rho}{\partial x} & \frac{\partial \rho}{\partial y} & \frac{\partial \rho}{\partial \dot{x}} & \frac{\partial \rho}{\partial \dot{y}} & \frac{\partial \rho}{\partial w} & \frac{\partial \rho}{\partial h} & \frac{\partial \rho}{\partial f_1} & \cdots & \frac{\partial \rho}{\partial f_{16}} \\ \frac{\partial \phi}{\partial x} & \frac{\partial \phi}{\partial y} & \frac{\partial \phi}{\partial \dot{x}} & \frac{\partial \phi}{\partial \dot{y}} & \frac{\partial \phi}{\partial w} & \frac{\partial \phi}{\partial h} & \frac{\partial \phi}{\partial f_1} & \cdots & \frac{\partial \phi}{\partial f_{16}} \\ \frac{\partial \dot{\rho}}{\partial x} & \frac{\partial \dot{\rho}}{\partial y} & \frac{\partial \dot{\rho}}{\partial \dot{x}} & \frac{\partial \dot{\rho}}{\partial \dot{y}} & \frac{\partial \dot{\rho}}{\partial w} & \frac{\partial \dot{\rho}}{\partial h} & \frac{\partial \dot{\rho}}{\partial f_1} & \cdots & \frac{\partial \dot{\rho}}{\partial f_{16}} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial \dot{x}} & \frac{\partial w}{\partial \dot{y}} & 1 & \frac{\partial w}{\partial h} & \frac{\partial w}{\partial f_1} & \cdots & \frac{\partial w}{\partial f_{16}} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & \frac{\partial h}{\partial \dot{x}} & \frac{\partial h}{\partial \dot{y}} & \frac{\partial h}{\partial w} & 1 & \frac{\partial h}{\partial f_1} & \cdots & \frac{\partial h}{\partial f_{16}} \\ \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \dot{x}} & \frac{\partial f_1}{\partial \dot{y}} & \frac{\partial f_1}{\partial w} & \frac{\partial f_1}{\partial h} & 1 & \cdots & \frac{\partial f_1}{\partial f_{16}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{16}}{\partial x} & \frac{\partial f_{16}}{\partial y} & \frac{\partial f_{16}}{\partial \dot{x}} & \frac{\partial f_{16}}{\partial \dot{y}} & \frac{\partial f_{16}}{\partial w} & \frac{\partial f_{16}}{\partial h} & \frac{\partial f_{16}}{\partial f_1} & \cdots & 1 \end{bmatrix} \tag{3.12}$$

$$\mathbf{H}_k = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{-y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{y(\dot{x}y-\dot{y}x)}{(x^2+y^2)^{3/2}} & \frac{x(\dot{y}x-\dot{x}y)}{(x^2+y^2)^{3/2}} & \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{3.13}$$

In order to apply the extended Kalman filter to this model, the matrix $\mathbf{F}$ and the Jacobian matrix $\mathbf{H}$, given by eqn.(3.11) have to be substituted in eqs. (2.34) to (2.39).

4 Results

In this chapter, the results are presented. The performance of the PointNet is evaluated on the training dataset. The performance of the PointNet for different sizes of the input point cloud is investigated. A driving scenario containing a ego car and a single target is used for evaluation. This is a simple scenario where data assignment problem does not need to be solved because all detections arise from the same target. The detailed results of the two multi-object tracking algorithms developed during this thesis, are presented, along with the results from standard Kalman filter algorithm for comparison purpose. Later, a more complex driving scenario is investigated with an ego vehicle and two targets in a crossover driving maneuver. The data association and tracking performance of the appearance based filter is evaluated for this scenario. This scenario is chosen because it covers a critical maneuver where one target is occluded by the other for some time frames. It also addresses the scenario where closely spaced targets are in the field of view of the radar sensor, and the association of sensor detections to tracks is ambiguous.

4.1 Training and Validation Performance

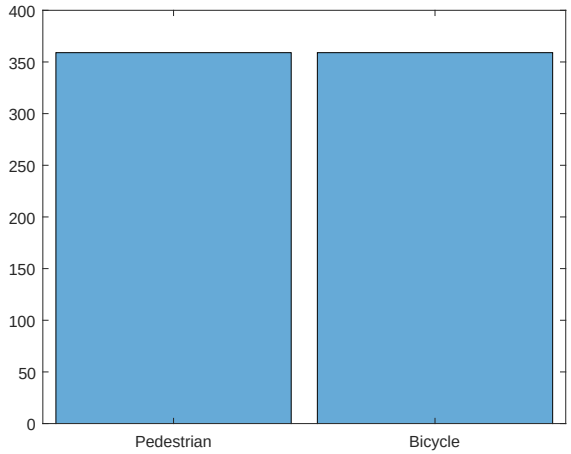


Figure 4.1: Histogram showing the class distribution in training dataset D_T. The dataset is balanced with 359 training examples per class.

The PointNet model, as described in Chapter 3, is trained with the training dataset D_T and validated against the validation dataset D_V. The validation accuracy of the model is 74% when the number of input points sampled from the point clouds is 80. In order to prevent the PointNet

from being biased towards any particular class and to train a robust classifier, a balanced dataset is used for training, with 359 examples in each class, as shown in Figure 4.1. A way to address class imbalance is by oversampling the infrequent classes. The network is trained for 10 epochs.

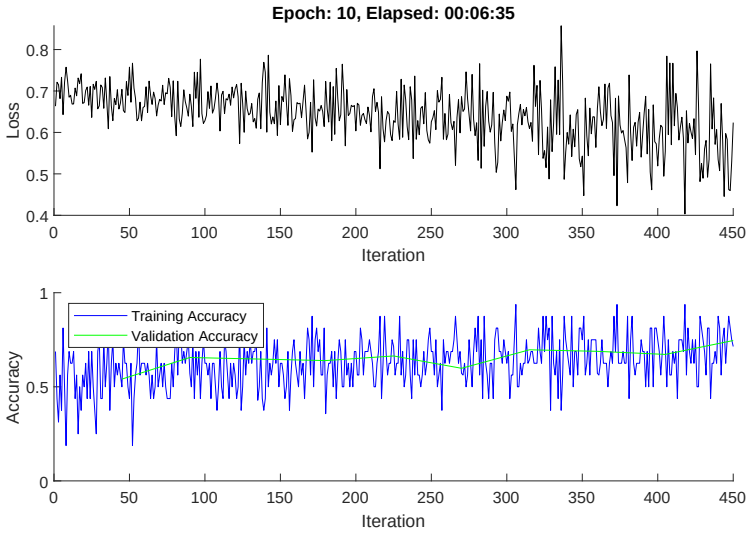


Figure 4.2: Training progress plot on dataset D_T for 10 epochs or 450 iterations. The training loss is plotted in the top figure and the figure in the bottom shows the plot for training and validation accuracy.

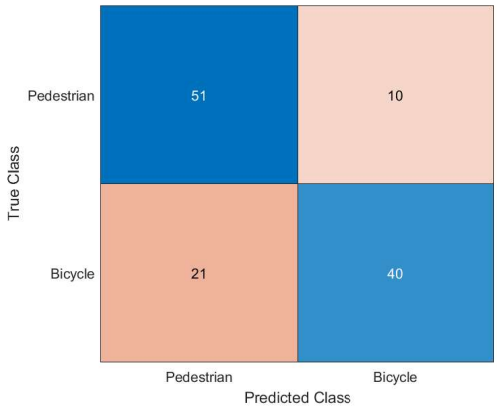


Figure 4.3: Validation confusion matrix for dataset D_V with input sample size of 80 points per point cloud, when trained on dataset D_T.

As shown in the training progress plot, Figure 4.2, the training converges fast and does not change significantly after the first 200 iterations but the validation accuracy improves slowly till

epoch 10, beyond which, overfitting sets in and the validation accuracy drops. The validation confusion matrix is shown in Figure 4.3. The PointNet classifies better on the class pedestrian as compared to the class bicycle, based on the learned features.

4.2 Effect of sample size on performance

To enable batch processing of data during training, a fixed number of input points are sampled from each point cloud. The optimal number of points depends on the data set and the number of points required to accurately capture the shape of the object. The minimum, maximum, and mean number of points per class help to choose the number of points to be sampled. Table 4.1 shows these values for dataset D_T. The point clouds are sparse for both classes since we have detections from a single radar sensor.

Table 4.1: Distribution of points per point cloud per class in training dataset D_T. It shows the minimum, maximum and average number of points per point cloud for both classes.

classes	numObservations	minPointCount	maxPointCount	meanPointCount
Pedestrian	359	186	250	232
Bicycle	359	217	246	234

From Table 4.1, it is evident that class class pedestrian has higher intra-class variability in the number of points per point cloud than class bicycle. Because of the intra-class and inter-class variability in the number of points per cloud, choosing a value that fits all classes is difficult. One heuristic is to choose enough points to adequately capture the shape of the objects while not increasing the computational cost by processing too many points.

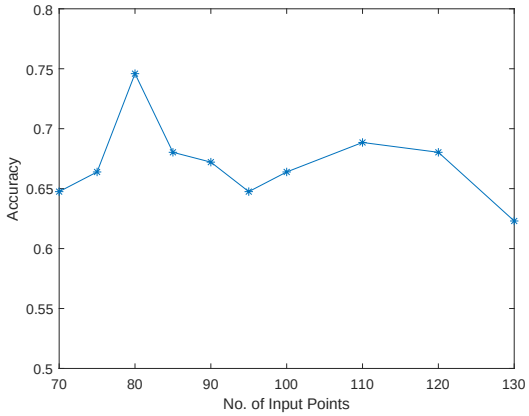


Figure 4.4: Plot of validation accuracy of the PointNet versus the input sample size when trained on D_T by sampling different number of input points from the training examples.

Figure 4.4 shows the effect of the input sample size on the validation accuracy of the PointNet. The validation accuracy is highest for an input sample size of 80 and decreases on decreasing the sample size. On increasing the sample size beyond 80, the validation accuracy decreases,

then increases slightly, and then decreases again. This behavior can not be explained with certainty because a very small dataset of sparse point cloud data has been considered in this thesis. The general intuition is, an increased sample size should enhance the performance of the network up to the mean point cloud size, beyond which, the performance should not change significantly. In this work, increasing the number of input points sampled from the point cloud beyond 80 does not significantly change the classification accuracy for the class pedestrian but substantially reduces the validation accuracy for the class bicycle. This behaviour is shown in Figure 4.5, where the validation confusion matrix is presented for the case when the point net is trained on 110 input points sampled from the point clouds. This reduces the overall validation accuracy to 69%.

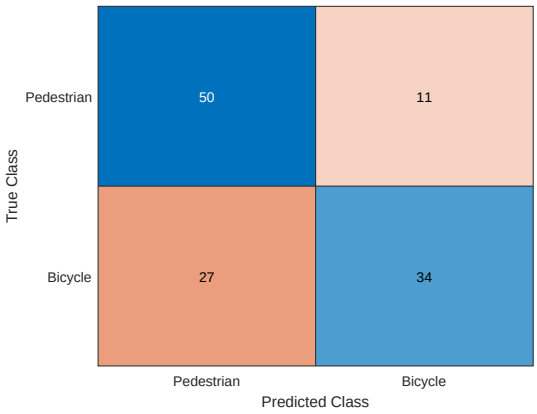


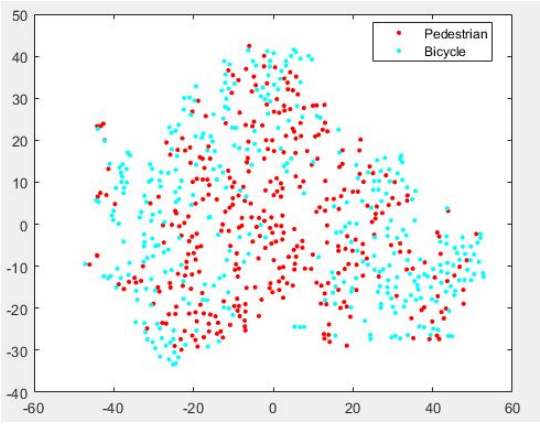
Figure 4.5: Validation confusion matrix when the PointNet is trained on D_T and validated on D_V and input sample size is 110.

4.3 Dataset Visualization By Dimension Reduction

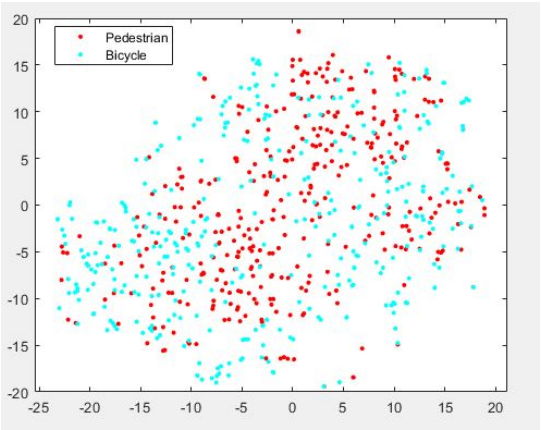
In this section, the 16 global features learned by the PointNet for each training example from D_T is visualized in a 2D space by dimension reduction using the t-distributed stochastic neighbor embedding (t-SNE) plot. t-SNE is a nonlinear dimension reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. The trained featured are visualized in 2D space using four different distance metrics to see which one offers a better seperation between the two classes in the training dataset.

Figure 4.6 shows the t-SNE plots when the distance metric is chosen as Euclidean and Mahalanobis, respectively. Figure 4.7 shows the t-SNE plots with distance metrics Cosine and Chebyshev.

The Euclidean and Mahalanobis distance metric offers better separation between the two classes than the Cosine and Chebyshev metrics. Overall, the seperation between the two classes is poor, which explains the low classification accuracy of the PointNet classifier. When a k-

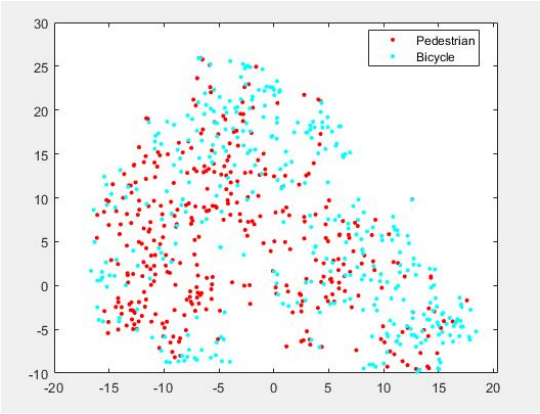


(a) t-SNE plot with Euclidean distance metric.

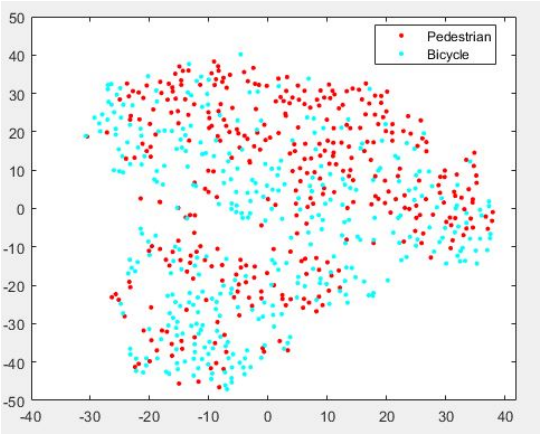


(b) t-SNE plot with Mahalanobis distance metric.

Figure 4.6: 16-dimensional global features learned by the PointNet for training dataset D_T are plotted by dimensionality reduction, where the chosen distance metric is (a) Euclidean (b) Mahalanobis.



(a) t-SNE plot with Cosine distance metric.



(b) t-SNE plot with Chebyshev distance metric.

Figure 4.7: 16-dimensional global features learned by the PointNet for training dataset D_T are plotted by dimensionality reduction, where the chosen distance metric is (a) Cosine (b) Chebyshev.

nearest neighbor algorithm is trained on the global features of D_T with the Euclidean distance metric and five-fold cross validation, the validation loss is found to be 30%.

4.4 Performance Evaluation of Tracking Filters

To evaluate the performance of the target tracking filters, the driving scenario 1 from the training dataset, as shown in Figure 3.11(b), is considered. The ego vehicle is stationary and the target is a pedestrian road user, walking at 5.4 kmph, following a eight-shaped trajectory. The initial distance between the ego vehicle and the target is 6 meter. The scenario is run for 1 or 2 seconds with a simulation time step of 0.1 second. Thus, the tracking algorithms are evaluated for 10 or 20 frames of the driving scenario.

4.4.1 Localization Based Filtering Algorithm

The tracking results of the standard extended kalman filter and the unscented kalaman filter algorithms are presented for the driving scenario (Figure 3.11(b)). The tracked trajectory of the pedestrian target using EKF can be seen from the bird's eye plot in Figure 4.8. The error between the filter estimates and the true measurements, in range, Doppler velocity and angle of arrival, plotted against time, is shown in Figure 4.9. The normalized innovation square metric, along with the lower and upper bounds, is illustrated in Figure 4.10.

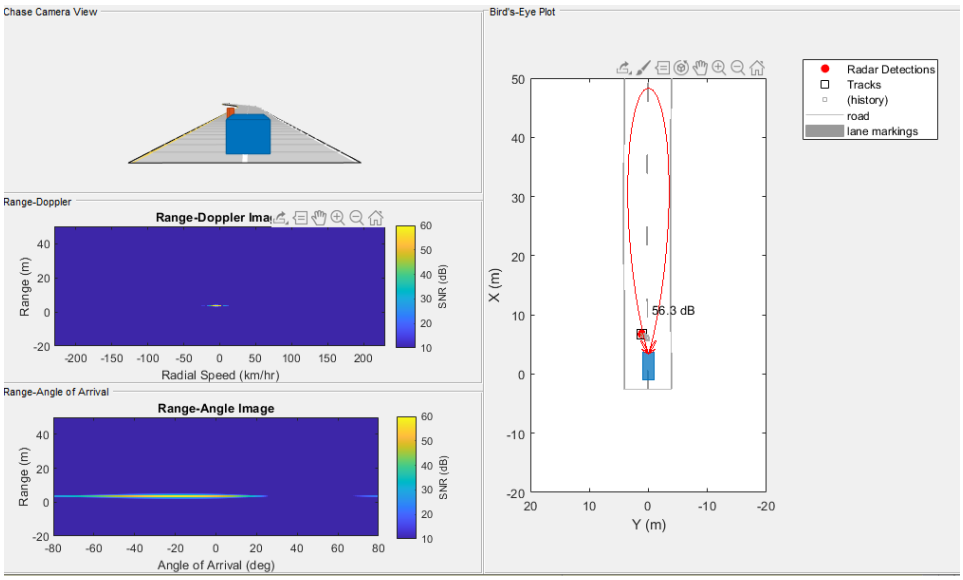


Figure 4.8: Tracking by extended Kalman filter. The top left figure shows the chase camera view of the driving scenario in Figure 3.11(b). The middle and bottom left figures show the range Doppler and range angle plots. Tracking is shown by the bird's eye plot in the right figure. The red circle represents the point target and the rectangles show the current and past tracks of the target.

The estimation error in position and angle are zero in the first simulation time step because the lateral and longitudinal position as well as the heading angle in the state vector are initialized with the true measurements in the first time step. Since the pedestrian target follows a curved trajectory with sharp turns and only the present state estimate and measurement are used to update the filter and no history of state estimates is used, the position estimation error increases

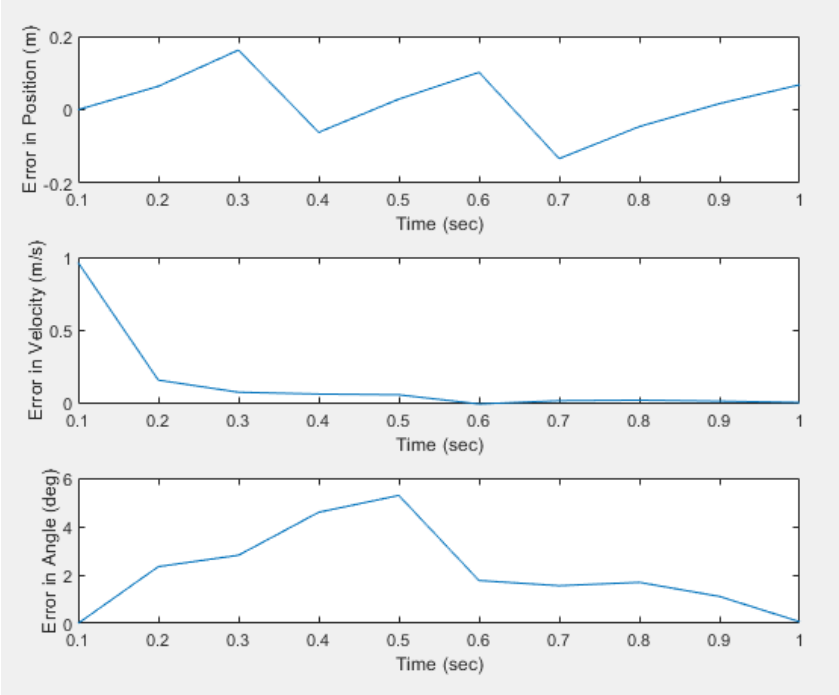


Figure 4.9: Estimation error plotted versus time for extended Kalman filter. The figure on top is the plot for position error, that is, the difference between the ground truth and the filter estimates. The middle figure is the plot for velocity error and the lowest plot shows the angle error.

gradually and varies from 0.2 meter to -0.1 meter and is occasionally close to zero, when the the target follows a straight trajectory for consecutive simulation time steps. The lateral and longitudinal velocities in the state vector, are however, initialized to zero in the first simulation time step and hence, the error in velocity is the highest in the first time frame and quickly drops to zero in the next frames because the target moves without acceleration. So, the EKF tracks the velocity of the target reliably. The estimation error in angle is zero at 0.1 sec, similar to the position, and increases as the target moves away from the field of view of the radar while following it's eight shaped trajectory. The angle error reduces as the target again moves within the line-of-sight of the radar.

The NIS for the EKF, shown in Figure 4.10, is the highest at 0.1 sec and drops in the subsequent time steps and stays well within the upper and lower bounds of the permissible NIS error which suggests that the EKS's noise assumptions are consistent with the sensor measurements. The NIS error overshoots the upper bound at 0.5 sec because the target makes a sharp turn in its trajectory, resulting in a high angle estimation error, as can be seen from Figure 4.9.

The bird's eye plot in Figure 4.11 shows the tracked trajectory of the same target by unscented kalman filter algorithm. The performance of the UKF for this scenario is evaluated based on the estimation error in range, Doppler velocity and angle of arrival, plotted across time, which is shown in Figure 4.12. Figure 4.13 shows the normalized innovation square metric, along with the lower and upper bounds.

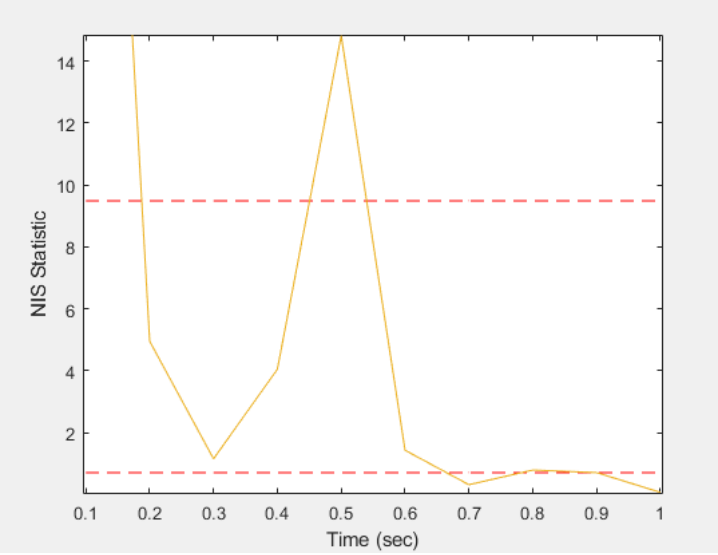


Figure 4.10: Normalized Innovation Square (NIS) plotted versus time for extended Kalman filter.

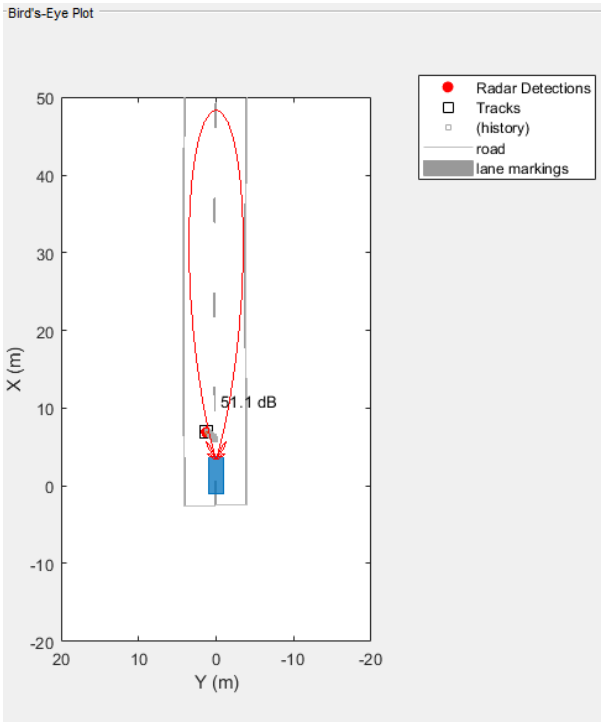


Figure 4.11: Tracking by unscented Kalman filter. The red circle represents the point target and the black squares show the current and past tracks.

The position and angle estimates are initialized to the true measurements in the first time step and the directional velocity estimated were initialized to zero, as in the EKF algorithm. The error estimation error plots, shown in Figure 4.12, resemble the plots for EFK. The position estimation errors are slightly higher for the UKF. The velocity estimates are close to the ground

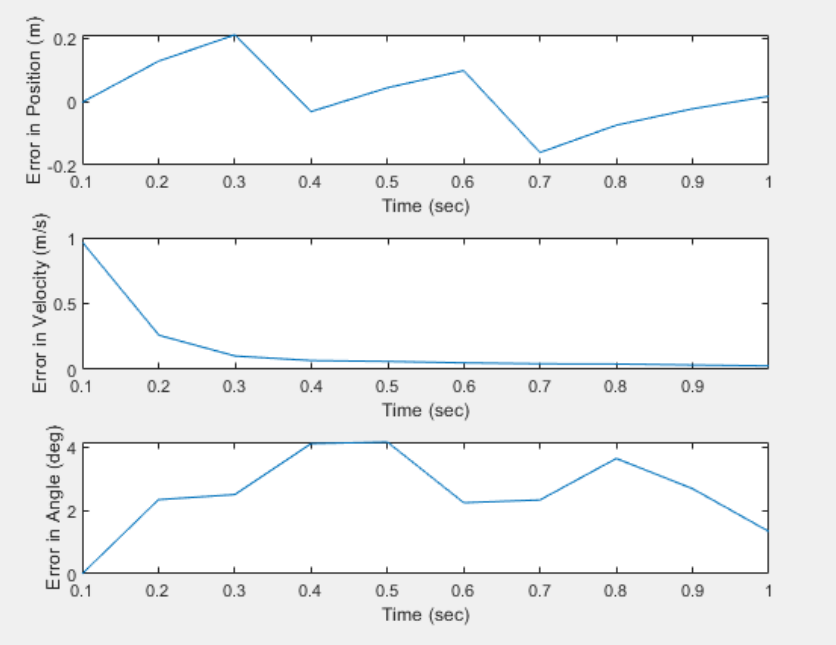


Figure 4.12: Estimation error plotted versus time for unscented Kalman filter. The top figure shows position error, the middle figure is the plot for velocity error and the lowest plot shows the angle error.

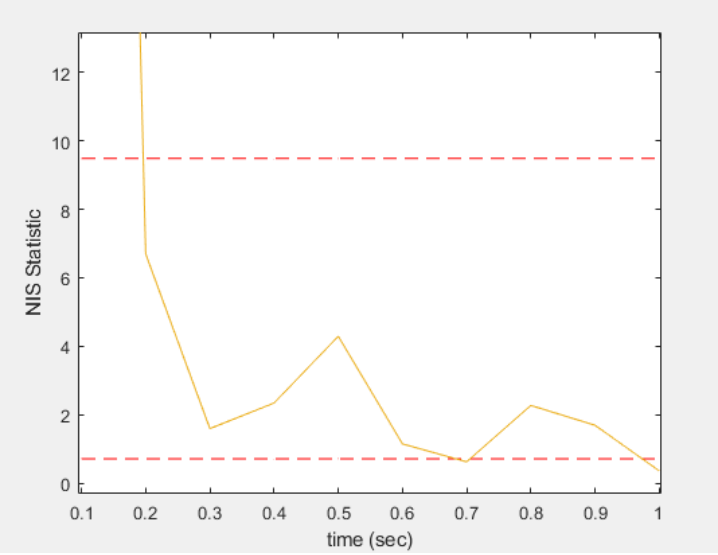


Figure 4.13: Normalized Innovation Square (NIS) plotted versus time for unscented Kalman filter.

truth after the first few frames and remains stable thereof. This is because the target moves with constant velocity, without acceleration. The angle estimates are unstable and the estimation error increases when the target takes sharp turns and decreases when the target maintains a

straight trajectory. The angle estimates are less stable and has more peaks and dips than the estimates by the EKF, but the angle estimation errors are lower as compared to the EKF.

The NIS plot for the UKF is shown in Figure 4.13. The NIS is the highest for the first frame and drops immediately in the next frames as the filter produces more stable estimates based on sensor measurements. The NIS error stays within the upper and lower bounds throughout the tracking phase, which suggests that the filter's noise assumptions are consistent with the sensor measurements. Based on the NIS metric, the UFK outperforms the EKF for this driving scenario.

4.4.2 Localization and Dimension Based Filtering Algorithm

The tracked trajectory of the target using the dimension based EKF algorithm, that tracks target localization and extent, is shown in Figure 4.14.

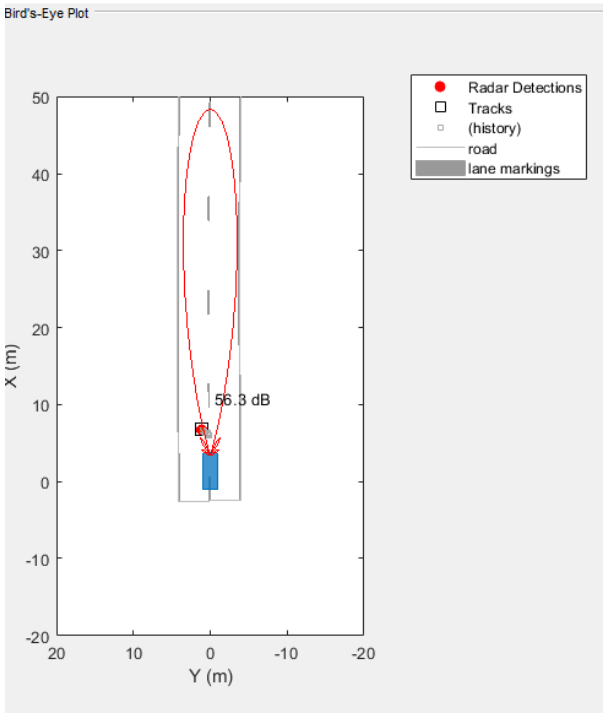


Figure 4.14: Tracking by dimension based extended Kalman filter. The red circle represents the point target and the black squares show the current and past tracks.

Figure 4.15 shows the estimation errors in position, velocity, angle and dimension of the target. The estimation errors in target localization closely resemble the errors in the basic EKF algorithm. The error in target length and width fluctuate on either side of zero, indicating unstable estimates. This is expected because the length and width measurements coming from the radar sensor are not accurate representation of the target dimension but an approximation, which highly depends on the number of radar returns at any simulation time step. So, the filter cannot match the sensor measurements by linear estimation. The negative errors in length indicate that the true length as measured by the sensor is smaller than the estimated length. Thus, the filter sometimes overestimates (negative error) and sometimes underestimates (positive er-

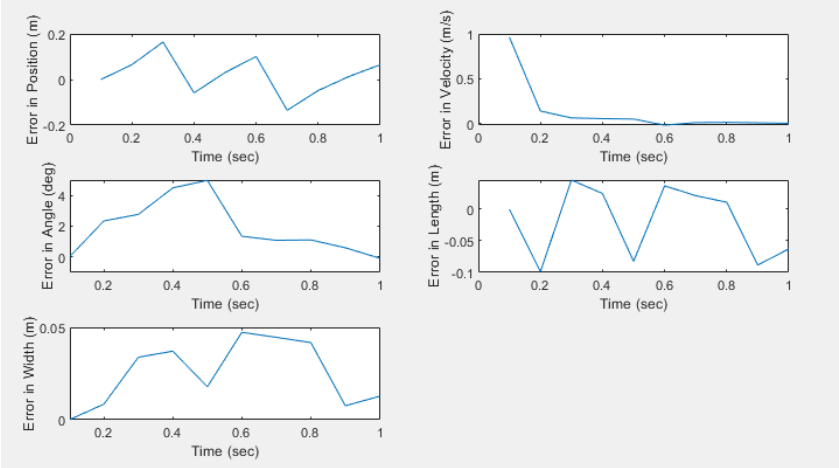


Figure 4.15: Estimation error plotted versus for dimension based extended Kalman filter. The top left plot shows position error, the top right plot is for velocity error, the middle left plot is for angle error, the middle right figure shows error in target length and the bottom right plot shows error in target width.

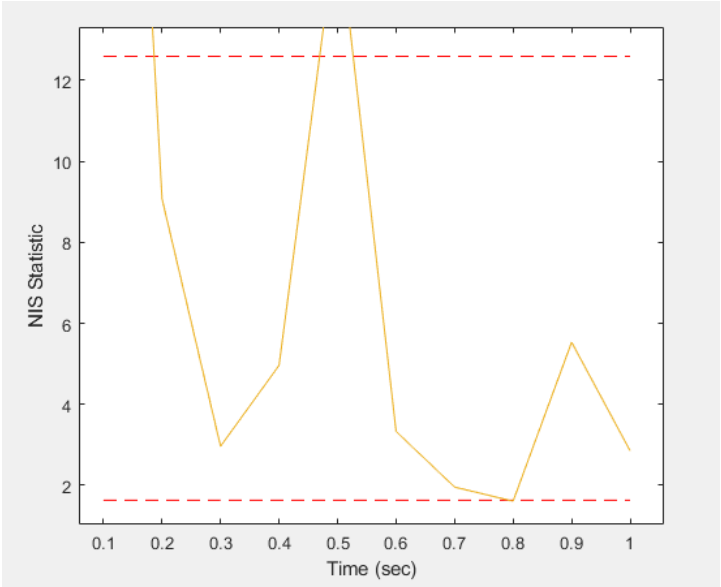


Figure 4.16: Normalized Innovation Square (NIS) plotted versus time for dimension based extended Kalman filter.

ror) the target length. The error in width, is however, always positive, so the filter estimates are lower than the measurements.

The NIS plot, shown in Figure 4.16, closely resembles the NIS plot for a standard EKF expect for the fact that the NIS metric is more pronounced at all times. This can be attributed to the inability of the modified filter to reliably estimate the target dimensions as the true model of the noise in dimension measurements is not known. The NIS is highest for the first time step.

It overshoots the upper bound at the fifth simulation time step, indicating that the combined process and measurement noise levels is low. The NIS stays within the upper and lower bounds for all other frames, indicating consistent estimates.

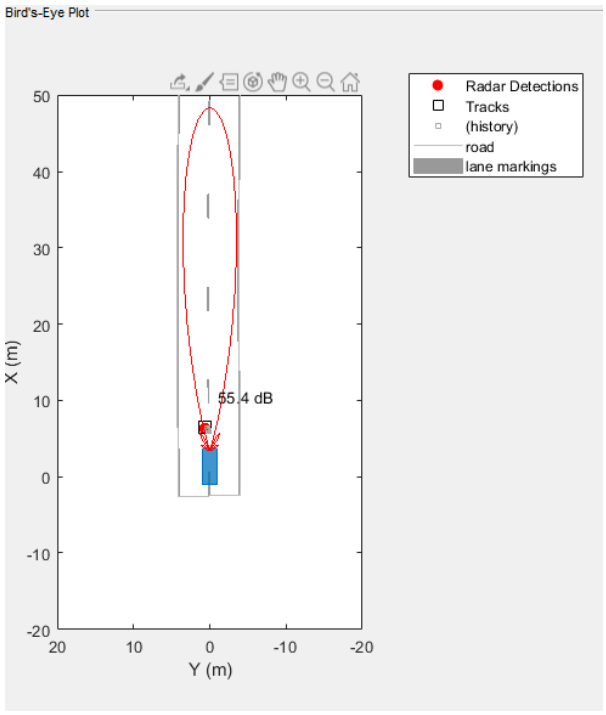


Figure 4.17: Tracking by dimension based unscented Kalman filter. The red circle represents the point target and the black squares show the current and past tracks.

The tracked trajectory of the dimension based unscented Kalman filter in illustrated by Figure 4.17. The estimation error plot, as shown in Figure 4.18, resembles the error plots for the standard UKF and the angle estimation error is lower as compared to the EKF. The error plots for dimension estimation resemble the plots for the modified EKF, however, the UKF has a higher tendency to underestimate the target length and the EKF has a higher tendency to overestimate it. The error in width is less pronounced for the modified UKF.

The NIS statistic plot for the dimension based UKF is shown in Figure 4.19. The normalized innovation squared is lower as compared to the modified EKF across all time steps. The error stays within permissible bounds for all time steps except 0.7 sec and 1 sec, where the error slightly dips below the lower bound, indicating that the combined process and measurement noise levels is too high.

4.4.3 Localization, Dimension and Appearance Based Filtering Algorithm

In this approach, the driving scenario, as illustrated in Figure 3.11(b), is run for 1.9 sec or for 19 simulation steps. In the first 9 time steps (that is, upto 0.9 sec), the point cloud data from the pedestrian target is aggregated and there is no tracking. Tracking starts at 1 sec and continues for ten time steps up to 1.9 sec. During tracking, at each time step, a new aggregated point cloud

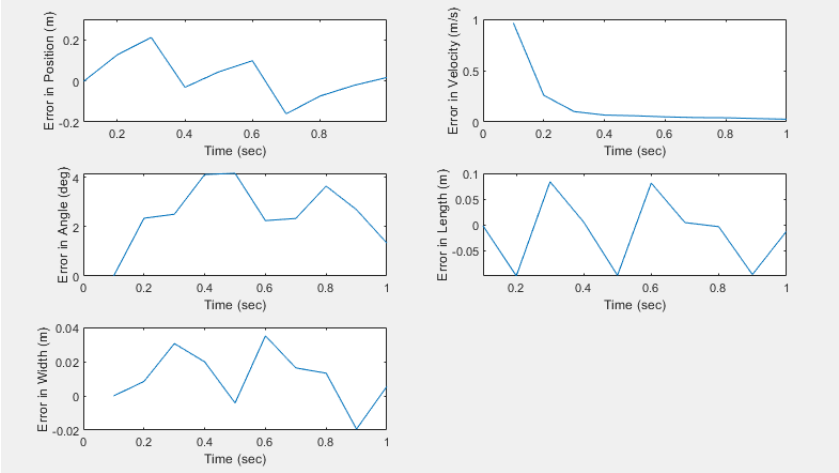


Figure 4.18: Estimation error plots for dimension based unscented Kalman filter. The top left plot shows position error, the top right plot is for velocity error, the middle left plot is for angle error, the middle right figure shows error in target length and the bottom right plot shows error in target width.

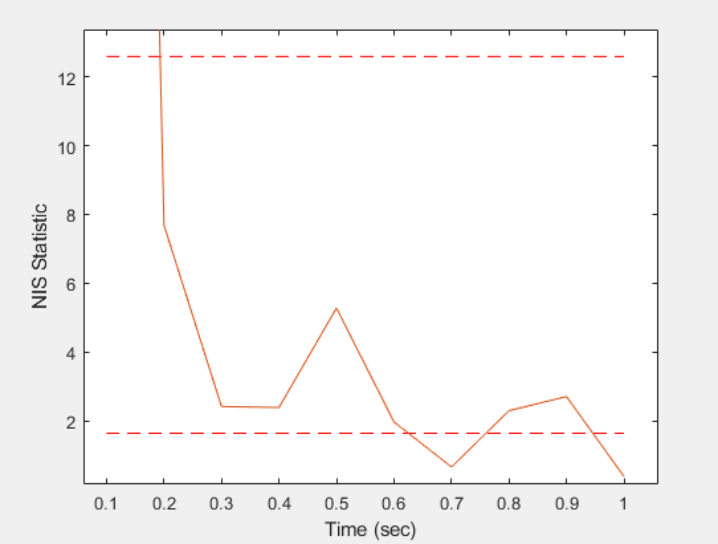


Figure 4.19: Normalized Innovation Square (NIS) plotted versus time for dimension based unscented Kalman filter.

is generated by aggregating the point cloud data for the ten most recent time steps. That is, at 1.5 sec, the aggregated point cloud contains radar returns from 0.6 sec to 1.5 sec. At 1.6 sec, the aggregated point cloud contains radar returns from 0.7 sec to 1.6 sec, and so on. This aggregated point cloud is fed to the PointNet and the 16 global features learned by the PointNet are fed to the tracking algorithm. The tracked trajectory of the pedestrian target by the appearance based EKF is illustrated by the bird's eye plot in Figure 4.20. Figure 4.21 shows the estimation errors plotted against time. The estimation errors are shown only for 3 out of 16 target fea-

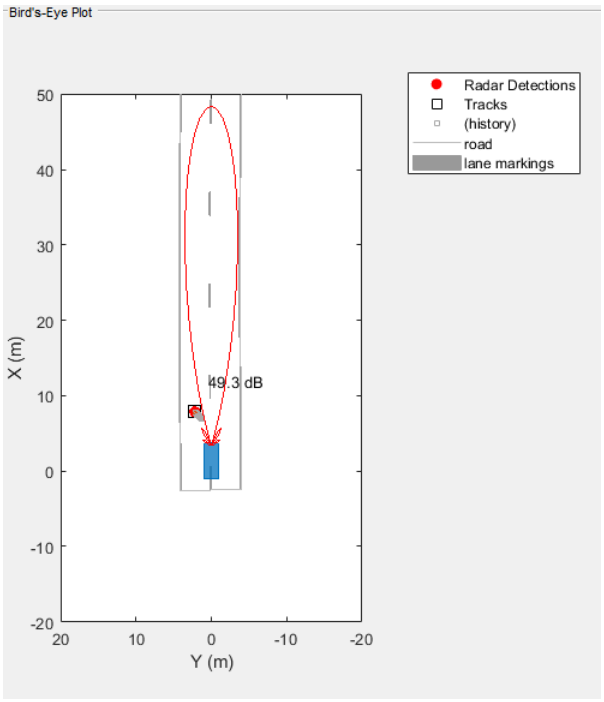


Figure 4.20: Tracking by appearance based extended Kalman filter. The red circle represents the point target and the black squares show the current and past tracks.

tures, for brevity. Except for velocity, all estimates are initialized to the true measurements at 1 sec. The position, velocity, angle and dimension estimation performance is the same as other trackers discussed before. The feature estimates are unstable, which is expected, because the PointNet predicts the features by approximating a non-linear function. These features cannot be matched by the filter by linear estimation. Nevertheless, the appearance model is beneficial when a target gets occluded for some frames, resulting in missed detections. Even when no true measurements are available, the filter can still estimate the appearance of the target using past estimates. The performance of the appearance based EKF based on NIS is illustrated by Figure 4.22. The normalized innovation squared value dips below the lower bound at 1.2 sec and overshoots at 1.5 sec and 1.9 sec. Thus, the filter performance is satisfactory for seven out of ten frames. These sharp peaks and dips show that the noise assumptions of the filter and not consistent with the measurement noise. This is because the noise model for target dimensions and feature measurements is not known. For experimental purposes, a variance is heuristically assumed and it is assumed to be same for all feature measurements. The variance chosen is the one that results in the best NIS values.

This tracking approach is repeated with a appearance based UKF and the tracked trajectory is shown by the bird's eye plot in Figure 4.23. The estimation error plots are shown in Figure 4.24 and the performance is comparable with that of the feature tracking EKF. The NIS plot, in Figure 4.25, also shows similar characteristics as for the feature tracking EKF.

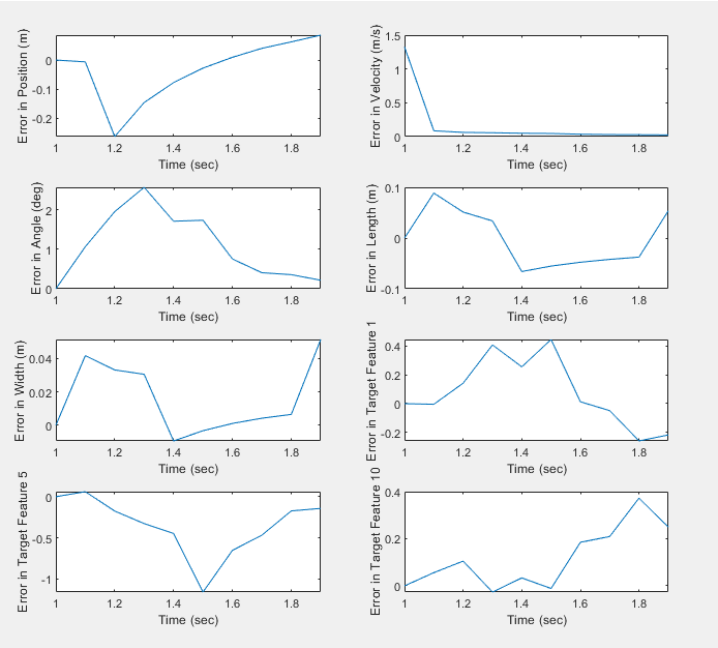


Figure 4.21: Estimation error plotted versus time for appearance based extended Kalman filter.The top left plot shows position error, the top right plot is for velocity error, the second row left plot is for angle error, the second row right figure shows error in target length, the third row left plot shows error in target width, third row right plot shows error in target feature 1 and the two bottom plots show the errors in target features 5 and 10, respectively.

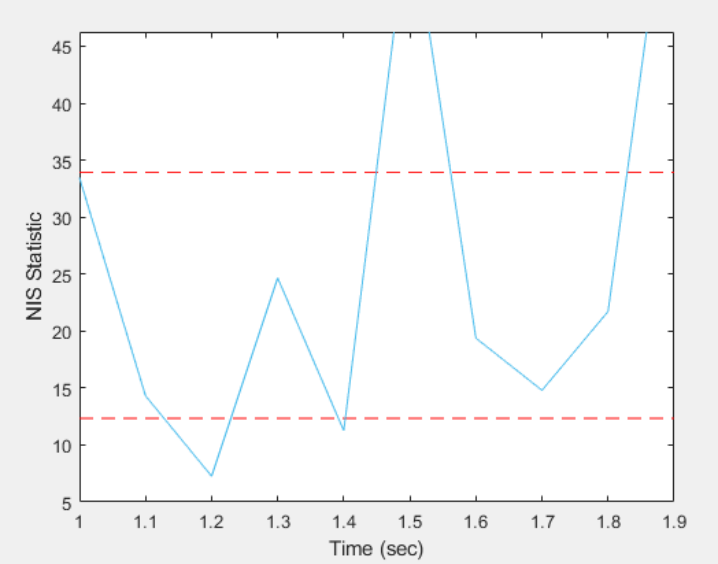


Figure 4.22: Normalized Innovation Squared plotted versus time for appearance based extended Kalman filter.

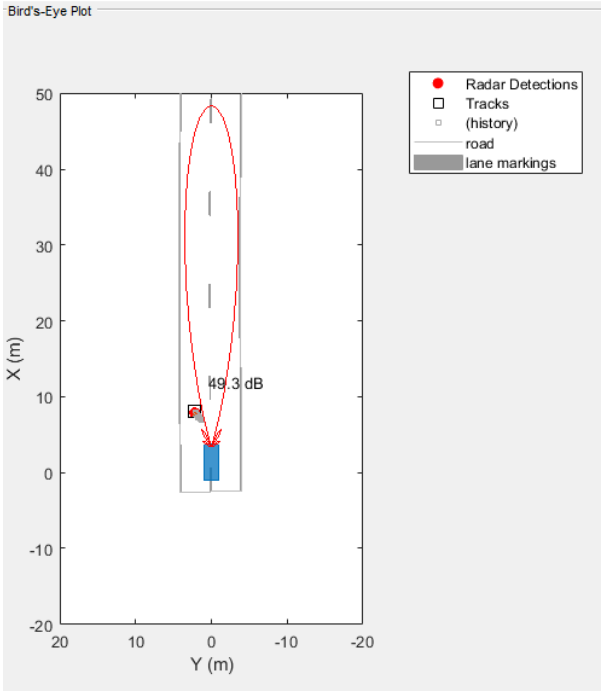


Figure 4.23: Tracking by appearance based unscented Kalman filter. The red circle represents the point target and the black squares show the current and past tracks.

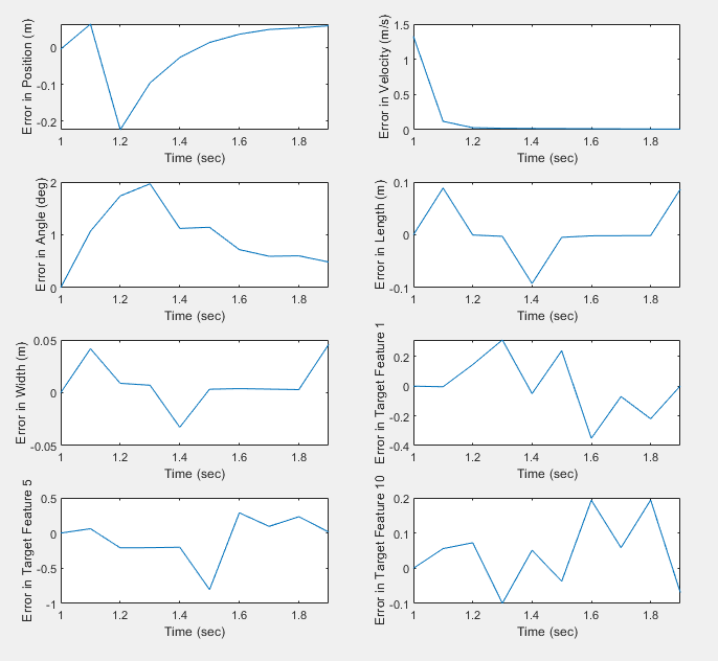


Figure 4.24: Estimation error plotted versus time for appearance based unscented Kalman filter. The top left plot shows position error, the top right plot is for velocity error, the second row left plot is for angle error, the second row right figure shows error in target length, the third row left plot shows error in target width, third row right plot shows error in target feature 1 and the two bottom plots show the errors in target features 5 and 10, respectively.

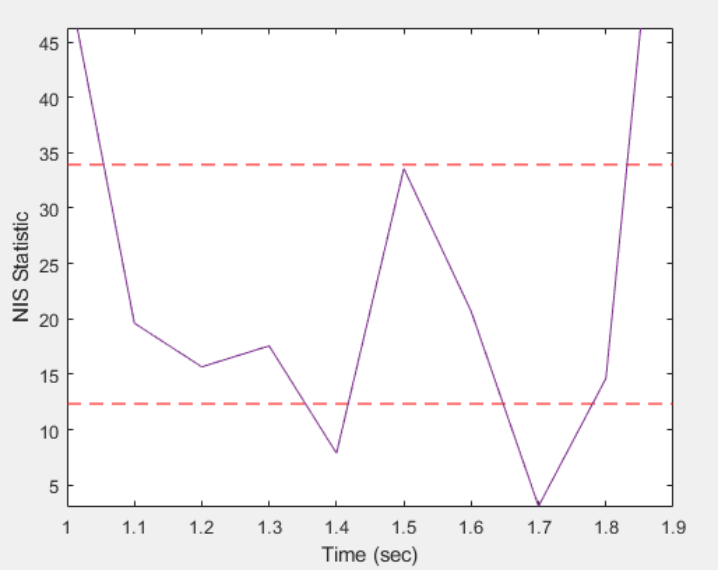


Figure 4.25: Normalized Innovation Squared plotted versus time for appearance based unscented Kalman filter.

4.5 Performance Comparison of Tracking Filters

Figure 4.26 shows the error in position between the ground truth and the estimates of the EKF based tracking algorithms, namely, the dimension based EKF and the appearance based EKF. The performance of these algorithms is compared with the baseline algorithm. Once again, driving scenario 3.11(b) is used for evaluation and tracking starts at 1 sec, for all three filtering algorithms. The localization error in position is the least for the appearance based EKF, whereas no significant improvement is seen in the dimension based EKF as compared to the baseline algorithm. Thus, augmenting the appearance of the target with its localization parameters improves the quality of the estimates coming from the filtering algorithm. Similar observations can be made when the localization error in position of the UKF based algorithms are compared with baseline algorithm. The error plot for the UKF based algorithms are shown in Figure 4.27.

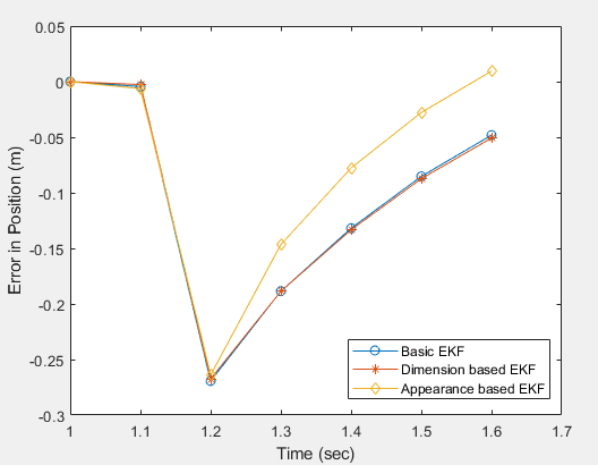


Figure 4.26: The error between the estimated and the ground truth position of the target for EKF based tracking algorithms.

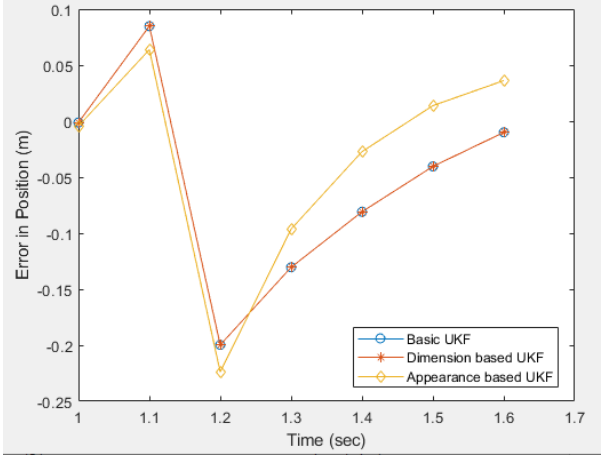


Figure 4.27: The error between the estimated and the ground truth position of the target for UKF based tracking algorithms.

4.6 Tracking a Crossover Maneuver

In this section, the performance of the appearance based EKF is evaluated for a driving scenario with a pedestrian and a bicyclist target involved in a crossover maneuver. The driving scenario is illustrated by Figures 4.28. The simulation is run for 3.2 sec, that is, 32 time steps. The ego vehicle is stationary and the targets are in front of it. The bicyclist has an initial distance of 7.8 meter from the ego vehicle and moves at a constant speed of 10.8 kmph. The pedestrian has an initial distance of 7 meter from the ego vehicle and moves at a speed of 8 kmph. The pedestrian target is walking in the right lane and the bicyclist is riding on the left lane. The bicyclist crosses over the pedestrian and goes to the right lane, while the pedestrian crosses over to the left lane. The crossover maneuver is a critical scenario where both targets are closely spaced with similar localization parameters and one target is occluded or missed detected for few frames. For the first ten simulation steps, both targets are on separate lanes with sufficient

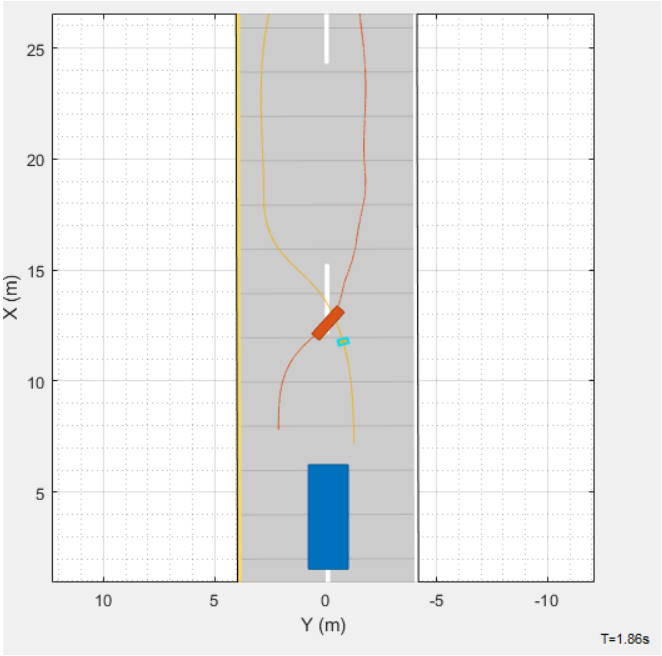


Figure 4.28: Driving scenario with crossover maneuver. The ego car is shown in blue, the bicyclist target in orange and the pedestrian target in sky blue. Crossover happens at 1.86 sec.

spacing between them, and the radar sensor generates two distinct clusters of detections.The detections are aggregated for the first ten time steps up to 1 sec and the aggregated point cloud data is fed to the PointNet. The PointNet generates global signatures for both targets which are fed into the tracker. Tracking starts at 1 sec, and at each subsequent simulation time step, new set of features are learned by the PointNet for both targets, based on the point cloud data aggregated from the most recent ten frames. This continues till 1.7 sec. At 1.8 sec, the crossover occurs and the two targets are in close proximity. One target is occluded by the other and the radar generates a single cluster of detections. To assign this cluster of detection to the right target, the following approach is used:

The single cluster of detection that is generated, is aggregated with the point cloud data of the past nine frames of both the targets. These aggregated point clouds are fed to the PointNet, which then generates two sets of feature vectors. The euclidean distance between this new feature vector and the feature estimate at the current time step by the tracking algorithm, is calculated for both targets. The detected cluster is assigned to the target with the lower euclidean distance. For the other target, there is a missed detection. The intuition is to associate the detection to the target which best resembles it.

Figure 4.29 shows the tracked trajectories for both targets. For the bicyclist target, the tracker is successful in maintaining the track despite the crossover. This indicates that the tracker can successfully associate data to the right target despite the ambiguity arising from closely spaced targets. As the pedestrian target is occluded by the bicyclist for few frames, missed detections are associated with it. The tracking of the pedestrian is resumed once detections are again available.

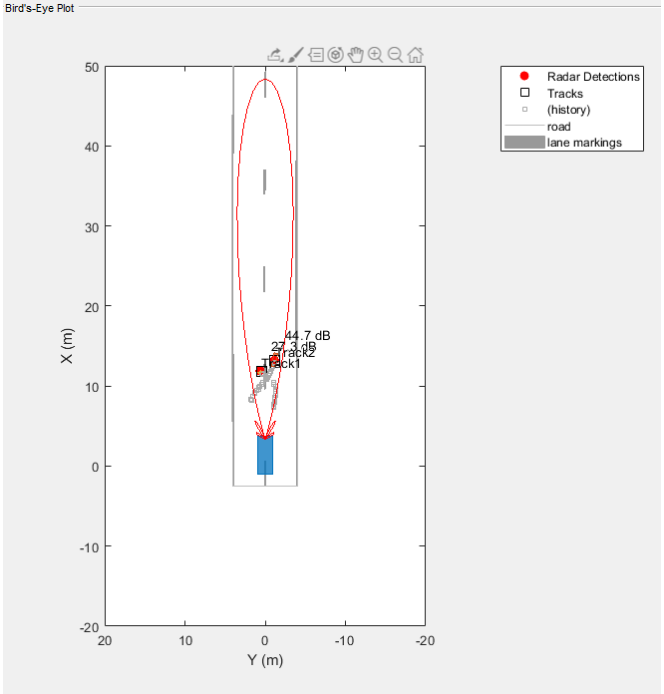


Figure 4.29: Tracking a crossover maneuver by appearance based extended Kalman filter. The red circles represents the point targets and the black squares show the current and past tracks. Track ID 1 is assigned to the bicyclist target and track ID 2 is assigned to the pedestrian target.

5 Discussion

This chapter comments on the performance of the PointNet and suggests improvements. The performance of the developed tracking algorithms are discussed. The results are summarized and directions for future work is shown.

5.1 PointNet Based Object Detection Approach

Unlike in previous works on target detection using PointNet, where thousands of training examples have been used to train the PointNet, in this thesis, a small but balanced dataset of only 718 training examples have been used. This is in line with the recent focus in the deep learning community, to resolve the "paradox" that extremely large, high capacity neural networks are able to simultaneously memorize training data and achieve good generalization error. In a number of experiments, Zhang et al. [38] demonstrated that large neural networks were capable of both achieving state of the art performance on image classification tasks, as well as perfectly fitting training data with permuted labels. In order to have a small but diverse dataset covering as many viewing angles from the radar as possible, complex driving maneuvers involving sharp turns are considered. A slight ability of the PointNet to generalize to new unseen driving scenarios is observed when tested on the crossover scenario, illustrated by Figure 4.28. A network which is guessing randomly and equally often a positive class and a negative class, would achieve an accuracy 50%. It must be kept in mind when evaluating the performance of any binary classifier. The PointNet achieved a validation accuracy of 74% which is decent result considering the fact that the dataset is limited and the point clouds are sparse. Applying cross validation can further push the generalization accuracy. Despite having balanced classes, the PointNet performs significantly better on the pedestrian class with a classification accuracy of 84% as compared to the bicycle class, where the classification accuracy is just 66%.

The point clouds consists of radar returns from a single radar sensor and do not provide an accurate representation of the object shape. Even though point cloud aggregation is used to add temporal information, it does not improve the knowledge of shape of the target. Fusing data from multiple radar sensors will ensure a 360° field of view and produce richer point clouds that provide more accurate representation of the targets. Furthermore, in this thesis, only five driving scenarios have been considered. Greater variety in driving scenarios with more complicated driving maneuvers would improve the generalization accuracy of the PointNet. In order to reduce the computational complexity of the feature based tracking algorithm, the PointNet was designed to learn only 16 global features from the input point cloud. The PointNet classifier performs classification based on these 16 features that summarize the input, obtained by max pooling. The t-SNE plots in Figure 4.6,4.7 show that the two classes are not well separated based on these 16 global features. Thus, any classifier would perform unsatisfactorily based on these features. Even a kNN classifier suffers a validation loss of 30%.

Max pooling is a rather simple technique to reduce the complexity of the embedding space and learn the most important features out of it. Thus by design, information is definitely lost when passed through a max pool layer [39]. It is worth evaluating the performance of the PointNet classifier by learning more global features or by using a combination of local and global features, as used in the original PointNet segmentation network. The decision to use PointNet was mainly due to the appealing property that it works on point clouds directly. In contrast, volumetric deep learning approaches lack the information on the lowest level. They render the point cloud unnecessary voluminous. Although PointNet allows to use point clouds as input for deep learning, it cannot capture local structure induced by the metric space points live in, therefore making it unlikely to learn fine grained patterns or to understand complex scenes. This is due to the max pooling operation which takes the full cloud as input to produce a global feature. However, exploiting local structure has proven to be important for the success of convolutional architectures. A CNN is able to progressively capture features at increasingly larger scales along a multi-resolution hierarchy. At lower levels neurons have smaller receptive fields whereas at higher levels they have larger receptive fields. The ability to abstract local patterns along the hierarchy allows better generalization to unseen cases. The PointNet++ handles this limitation. In PointNet++ [40], a PointNet network is applied recursively to a nested partitioning of the input point set. The approach is somehow similar to CNN with images. The first layer kernels see very local data (small receptive field), but the deeper layer have larger receptive field. Another notable neural net working on point cloud data is PointCNN. The PointCNN [41] approach proposes to learn a transformation that would learn a canonical representation of the input points invariant to permutations. From there, standard convolution could be applied. As the authors precised, the invariance to permutations is far from perfect, but they still claim to achieve state of the art results on principal point cloud datasets. It must be investigated if these architectures might yield better results than PointNet.

5.2 Dataset Improvement

The quality of the dataset is crucial to train a robust classifier. The generalization accuracy of the PointNet classifier determines the accuracy of the feature based tracking algorithm. The presented dataset is limited. The dataset contains only one target per radar measurement cycle. The dataset is idealized and contains no clutter. A further restriction is that the object has always the same ground truth size. That is, the given dataset is made of pedestrian or bicyclist targets of a fixed dimension only. The current dataset includes objects of two classes only. For future research, the dataset should be extended to other classes of road users like cars, trucks, etc. The classes can be defined broadly using labels which include different objects. For example the class vehicle may contain different types of vehicles like cars, trucks, cranes, pumps etc. This would promote variety in the dataset and help the network to generalize better. The dataset is synthetically generated in MATLAB® and it not an accurate representation of real traffic data. Real driving datasets would involve critical scenarios such as partially and completed occluded targeted, reflections from static obstructions like guard rails on a highway, etc. As deep learning is a heavily data-driven approach, the top bottleneck in radar-based applications is the availability of publicly usable data annotated with ground truth information. Only the very recent nuScenes dataset [42] provides non-disclosed type of 2D radar sensor with sparsely populated 2D points but without the sampled radar ADC data required for deep radar detection

whereas, Astyx HiRes2019 dataset [43] provides 3D imaging radar data that contains only 546 frames with ground-truth labels, which is relatively too small for objection detection. Manually generating large amounts of data by simulation is a tedious task and results in a biased dataset. Data from a single radar sensor is very sparse and in the dataset used, there were many frames with no or very few detections. The fusion of radar measurements with different time stamps, as well as measurements from multiple radar sensors is crucial for generating a rich dataset.

5.3 Selecting Relevant Features

When the dataset is limited, explicit feature selection might be an essential step to avoid overfitting. The dataset used in this thesis is solely based on 3D point clouds without additional information. It is likely that adding more information in the form of radar cross section value or angle of arrival information, would drastically increase the performance and robustness of the PoinNet. In [44], the authors have considered radar point clouds represented as a set of four-dimensional points to to classify and localize objects in 2D space. High dimensional point clouds increases the computational complexity but for small to medium datasets, it can be easily incorporated. In [45], the authors extended the input dimensions to 10 where the first 3 dimensions are (x, y, z). The second 3 dimensions correspond to the spherical coordinates per point (ρ, ϕ, θ). The inputs 7 through 10 are the Doppler velocity per point , the reflection magnitude , the velocity of the ego vehicle, and the turn rate of the ego vehicle respectively. The spherical coordinates are included to allow the network to learn connections between points that are easier to represent in the spherical coordinate system. For example two points with the same azimuth and elevation would only differ in depth, such a relationship is linear and easier to learn, unlike in the Cartesian system where this relationship is a function of the square root of the sum of squares. This increases the dimensions of the shared multi-layer perceptron layers and thus allow for better encoding of features. As a result, the feature transformation network dimensions are also increased. Deeper and more complex network architecture can help reveal more relationships between the data.

5.4 Target Tracking

All three target tracking algorithm discussed in this thesis, essentially employ point tracking. The point target assumption is true for scenarios with multiple remote objects that are far away from the sensor, e.g., as in radar-based air surveillance. In such scenarios, an object is not always detected by the sensor, and if it is detected, at most one sensor resolution cell is occupied by the object. For autonomous driving applications, objects are in the near-field of sensors, thereby rendering the point target assumption invalid. Since a short range radar has been used in this thesis, the driving scenarios that have been considered, always had the target VRU within 30 meters of the radar sensor, leading to multiple detections per target. When few detections are present per object, it is difficult to determine the reflection point on the object. In tracking, it is problematic to estimate states based on a single measurement, whose point of origin is unknown. For instance, it can be that the algorithm uses the front right corner as reference point and updates with a detection from front left side of the car. In this case, it is interpreted as a movement in the wrong direction. Even if a measurement would be accurate, it would still

be possible that states are estimated incorrectly because the origin of the detection in terms of reflection point is not known. For the localization based tracking algorithm, the range and range rate estimates are more reliable than the angle estimates. The range estimates are smaller in the first frames, when the target is close to the ego vehicle, resulting in positive estimation errors. As the distance between the ego vehicle and the target increases, the estimates are larger than measurements, leading to negative estimation errors. The errors in range, range rate and angle reduce with increasing distance from the target. This also gives an indication of that the single point target assumption is valid for estimating at long ranges.

The point target assumption does not account for the extension of objects. This is a problem when tracking extended objects in general. The dimension based tracking algorithm relies on the fact that the entire extension of the target is captured by the sensor, in order to produce correct estimates. Since accurate measurements without clutter never yield a rectangle which is larger than the actual size, this dimension estimation approach is prone to underestimate the size of the target for all situations [22]. In an automotive application it is important not to underestimate the size of vehicles, hence it is problematic to estimate based on this condition. When the target is closest to the ego vehicle, the error in length is most negative, indicating that the estimates are larger than the measurements. As the target moves away from the ego car, the length estimation improves. The positive width estimation error indicates that the tracking algorithm always underestimates the target width. The width estimation, in contrast to the length estimation, is in general poor at long ranges. A sophisticated approach will be to model the object extent by a simple geometric primitive, such as a rectangle or an ellipse of a certain size that best represents the target dimensions. The most advanced approach is to construct a measurement model that is capable of handling a broad variety of both different shapes and different measurement appearances. While such a model would be most general, it could also prove to be overly computationally complex.

The feature based tracking algorithm relies on the accuracy of the PointNet. As the PointNet learns only 16 global features for the entire target, much of the target information is lost and the feature vector is not the most accurate representation of the input point cloud. Since the underlying model for feature prediction by the PointNet is unknown to the feature based tracker, the reliability of the feature estimates is compromised. This is an expected result and it is worth noting that even though the feature estimates are not the most reliable representation of the target appearance, the feature estimates of the same target would resemble closely and the learned appearance of different targets would be distinguishable. This would ensure that even if a target is missed detected in some frames, the feature based tracking algorithm could perform data association to targets despite missed detections. From the results in Figure 4.29, it can be concluded that the appearance based tracking algorithm handles the data association task well.

Another key problem of the Kalman filter is that its estimation performance is greatly influenced by the values of the system parameters and covariance matrices of the process noise $\mathbf{Q}$ and the measurement noise $\mathbf{R}$. According to the Kalman filter theory, $\mathbf{Q}$ and $\mathbf{R}$ have to be obtained by considering the stochastic properties of the corresponding noises [46], however, since these are usually not known, in most cases, the covariance matrices are used as weighting factors (tuning parameters). These matrices are tuned manually by trial and error methods which are very tedious procedures. The modeling inaccuracies in $\mathbf{Q}$ and $\mathbf{R}$ causes the NIS to overshoot or undershoot at some time instants, for all three filters.

5.5 Future Work

A take for future work would be to extend the dataset and test the performance of the PointNet for many different classes of road users. Additionally, objects in the dataset belonging to the same class will have different sizes to examine the impact of object dimension. A larger dataset with an increased variety of driving scenarios containing critical driving maneuvers like crossings, u-turns, roundabouts, takeovers, etc., could increase the generalization performance of the PointNet. Data from multiple radar sensor should be fused for many time stamps to generate richer point clouds. Adding more features to the point clouds to generate higher dimensional point cloud data is another area of investigation that could be of interest. Additionally, the performance of other neural networks like PointNet++ or PointCNN should also be evaluated on the dataset and comparison should be made among different object detectors to see which has the best performance in ”radar-only” object detection. Once satisfactory performance is obtained from simulation, the network performance has to be evaluated on a real dataset, with multiple targets and more clutter per frame.

A more sophisticated approach to track the dimension of the target is worth investigating. In the current approach, if there are no or few detections at any instant, the estimated dimensions of the target, as reported by the sensor, is very low. This causes the Kalman filter to underestimate the target dimension. A better approach will be to update the filter only when the number of detections reported by the radar is more than a minimum . Since it is not desirable to underestimate the size, updating size to a larger estimation should be easier than updating to a smaller size. This property has to be incorporated in the variance of the measurement noise [22]. The performance of such a tracker is to be compared to that of a GM-PHD tracker and rectangular target model that can handle multiple detections per object per sensor, without the need to cluster these detections first. A GM-PHD is initialized with multiple possible sizes and their relative weights using multiple components.

To generate reliable estimates using non-linear Kalman filter, it is important to know the accurate characterization of the uncertainties in the state dynamics and in the measurements. In this thesis, the parameters of the noise densities associated with these uncertainties are, however, treated as ‘tuning parameters’ and adjusted in an ad hoc manner while carrying out state and parameter estimation. To avoid this tedious and erroneous process, in future work, the process and measurement noises should be estimated by solving an optimization problem. In [47], the authors have developed two approaches based on the maximum likelihood estimates (MLE) framework for estimation of the noise covariances from the operating data when the state estimation is carried out using EKF. This is especially important for the feature based tracking algorithm as is it not possible to tune the process and measurement noise for all features collectively on a trial and error basis.

6 Conclusion

In this thesis, an automotive radar has been modeled in MATLAB®. The received sensor data is processed by a signal processing chain to generate radar point clouds. The radar point clouds are aggregated for multiple time stamps and are fed into a deep learning based object detector called PointNet. The global features learned by the PointNet are fed to a feature based tracking algorithm. It can be concluded that the PointNet performs decently, given the small driving dataset it has been trained on. However, in order to have real world applications, the PointNet has to be trained on a diverse dataset containing numerous critical driving maneuvers. In existing research, the PointNet classifier has been proved to achieve accuracy as high as 96%. Further training is needed to make the PointNet network used in this thesis, to achieve comparable accuracy. The performance of the PointNet, should also be compared to other advanced point cloud based object detectors like PointNet++ or PointCNN, which claim to exploit the local structures in the point cloud.

It can be concluded that, integrating the appearance parameters with the localization parameters in the state vector improves the performance of the tracking algorithm. The appearance based target tracking algorithm successfully handles the data association task despite multiple missed detections due to partial or complete occlusion. However, the PointNet performance is still improvable by expanding the dataset. Once the accuracy of the learned features is reliable, the performance of the feature based tracking algorithm can be re-evaluated to see if the performance is comparable with that of a state-of-the-art PHD tracker. If so, this approach would greatly reduce the computational complexity of the multi-object tracking and can be implemented in a real world VRU tracking system.

Bibliography

[1] F. Engels, P. Heidenreich, A. M. Zoubir, F. K. Jondral, and M. Wintermantel, "Advances in Automotive Radar: A framework on computationally efficient high-resolution frequency estimation," *IEEE Signal Processing Magazine*, vol. 34 (2), pp. 36–46, 2017 (cit. on pp. 1, 3).

[2] F. Engels, P. Heidenreich, A. M. Zoubir, F. K. Jondral, and M. Wintermantel, "Advances in Automotive Radar: A framework on computationally efficient high-resolution frequency estimation," *IEEE signal processing magazine*, vol. 34 (2), 2017 (cit. on p. 1).

[3] N. Dasanayaka, K. F. Hasan, C. Wang, and Y. Feng, "Enhancing Vulnerable Road User Safety: A Survey of Existing Practices and Consideration for Using Mobile Devices for V2X Connections," 2020 (cit. on p. 1).

[4] Y.-W. Hsu, Y.-H. Lai, K.-Q. Zhong, T.-K. Yin, and J.-W. Perng, "Developing an On-Road Object Detection System Using Monovision and Radar Fusion," *Energies*, vol. 13, 2020 (cit. on p. 1).

[5] F. Nobis, M. Geisslinger, M. Weber, J. Betz, and M. Lienkamp, "A Deep Learning-based Radar and Camera Sensor Fusion Architecture for Object Detection," pp. 1–7, 2019 (cit. on p. 1).

[6] R. Devagiri, N. C. Iyer, and S. Maralappanavar, "Real-time RADAR and LIDAR Sensor Fusion for Automated Driving," Springer Singapore, S. Agarwal, S. Verma, and D. P. Agrawal (Eds.), pp. 137–147, 2020 (cit. on p. 1).

[7] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *CoRR*, vol. abs/1612.00593, 2016 (cit. on pp. 1, 12, 40).

[8] V. C. Chen, "Micro-Doppler effect of micromotion dynamics: a review," vol. 5102, pp. 240–249, 2003 (cit. on p. 2).

[9] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34 (2), pp. 22–35, 2017 (cit. on pp. 3–8).

[10] E. P. Lam, "Clutter mitigation scheme in presence of wind-blown foliage for FMCW radar," in *Radar Sensor Technology XXII*, K. I. Ranney and A. Doerry (Eds.), International Society for Optics and Photonics, vol. 10633, SPIE (cit. on p. 4).

[11] L. Scharf and C. Demeure, "Statistical Signal Processing: Detection, Estimation, and Time Series Analysis," 1991 (cit. on p. 7).

[12] M. A. Richards, "Fundamentals of Radar Signal Processing." McGraw-Hill Professional, 2005 (cit. on p. 9).

[13] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," 2019 (cit. on p. 10).

[14] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The Expressive Power of Neural Networks: A View from the Width," *CoRR*, vol. abs/1709.02540, 2017 (cit. on p. 10).

[15] M. Valueva, N. N. Nagornov, P. A. Lyakhov, G. Valuev, and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Math. Comput. Simul.*, vol. 177, pp. 232–243, 2020 (cit. on p. 10).

[16] T. Hinks, H. Carr, L. Truong-Hong, and D. Laefer, "Point Cloud Data Conversion into Solid Models via Point-Based Voxelization," *Journal of Surveying Engineering*, vol. 139, pp. 72–83, Jan. 2012 (cit. on p. 12).

[17] S. Milz, M. Simon, K. Fischer, and M. Pöpperl, "Points2Pix: 3D Point-Cloud to Image Translation using conditional Generative Adversarial Networks," *ArXiv*, vol. abs/1901.09280, 2019 (cit. on p. 12).

[18] J. Nazzal, I. El-Emary, and S. Najim, "Multilayer Perceptron Neural Network (MLPs) For Analyzing the Properties of Jordan Oil Shale," *World Applied Sciences Journal*, vol. 5, Jan. 2008 (cit. on p. 13).

[19] C. Garcia and M. Lerjefors, "Radar Based Classification of Vulnerable Road Users," 2019 (cit. on pp. 13, 14).

[20] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," in *NIPS*, 2015 (cit. on p. 13).

[21] Z. Chen, "Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond," *Statistics*, vol. 182, Jan. 2003 (cit. on p. 15).

[22] E. Henriksson and V. Kardell, "Radar-based target tracking for 360-degree environmental perception," 2016 (cit. on pp. 15, 21, 70, 71).

[23] R. Kalman, "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic," 1960 (cit. on p. 16).

[24] S. S. Blackman, "Multiple-target tracking with radar applications." 1986 (cit. on pp. 17, 22, 23).

[25] C. Montella, "The Kalman Filter and Related Algorithms: A Literature Review," May 2011 (cit. on p. 19).

[26] E. Wan and R. Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," *The Unscented Kalman Filter for Nonlinear Estimation*, vol. 153-158, pp. 153–158, Feb. 2000 (cit. on p. 20).

[27] Y. bar-shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *Control Systems, IEEE*, vol. 29, pp. 82–100, Jan. 2010 (cit. on p. 23).

[28] S. H. Rezatofighi, A. Milan, Z. Zhang, Q. Shi, A. Dick, and I. Reid, "Joint Probabilistic Data Association Revisited," pp. 3047–3055, 2015 (cit. on p. 23).

[29] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19 (1), pp. 5–18, 2004 (cit. on p. 24).

[30] K. Granström, M. Baum, and S. Reuter, "Extended Object Tracking: Introduction, Overview, and Applications," *Journal of Advances in Information Fusion*, vol. 12, Dec. 2017 (cit. on p. 25).

[31] B. Kalyan, K. W. Lee, S. Wijesoma, D. Moratuwage, and N. M. Patrikalakis, "A random finite set based detection and tracking using 3D LIDAR in dynamic environments," pp. 2288–2292, 2010 (cit. on p. 25).

[32] B.-N. Vo, S. Singh, and A. Doucet, "Sequential Monte Carlo implementation of the PHD filter for multi-target tracking," *Proceedings of 6th International Conference Information Fusion*, vol. 2, pp. 792–799, Feb. 2003 (cit. on p. 26).

[33] B. .-. Vo and W. .-. Ma, "The Gaussian Mixture Probability Hypothesis Density Filter," *IEEE Transactions on Signal Processing*, vol. 54 (11), pp. 4091–4104, 2006 (cit. on p. 26).

[34] X. Wang and J. Wang, "Simulation analysis of EKF and UKF implementations in PHD filter," pp. 1–6, 2016 (cit. on p. 26).

[35] D. Clark, B. Vo, and B. Vo, "Gaussian Particle Implementations of Probability Hypothesis Density Filters," in *2007 IEEE Aerospace Conference*, 2007, pp. 1–11 (cit. on p. 26).

[36] R. Piché, "Online tests of Kalman filter consistency," *International Journal of Adaptive Control and Signal Processing*, vol. 30 (1), pp. 115–124, 2016 (cit. on p. 26).

[37] Y. bar-shalom, X.-R. Li, and T. Kirubarajan, "Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software." Jan. 2004 (cit. on p. 27).

[38] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *CoRR*, vol. abs/1611.03530, 2016 (cit. on p. 67).

[39] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing between Capsules," Curran Associates Inc., pp. 3859–3869, 2017 (cit. on p. 68).

[40] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," Curran Associates Inc., pp. 5105–5114, 2017 (cit. on p. 68).

[41] Y. Li, R. Bu, M. Sun, and B. Chen, "PointCNN:Convolution On X-Transformed Points," *CoRR*, vol. abs/1801.07791, 2018 (cit. on p. 68).

[42] H. Caesar, V. Bankiti, A. H. Lang, and S. Vora, "nuScenes: A multimodal dataset for autonomous driving," *CoRR*, vol. abs/1903.11027, 2019 (cit. on p. 68).

[43] M. Meyer, "Automotive Radar Dataset for Deep Learning Based 3D Object Detection," Jan. 2019 (cit. on p. 69).

[44] A. Danzer, T. Griebel, M. Bach, and K. Dietmayer, "2D Car Detection in Radar Data with PointNets," pp. 61–66, 2019 (cit. on p. 69).

[45] M. Chamseddine, J. Rambach, O. Wasenmüller, and D. Stricker, "Ghost Target Detection in 3D Radar Data using Point Cloud based Deep Neural Network," Oct. 2020 (cit. on p. 69).

[46] Y. Laamari, K. Chafaa, and B. Athamena, "Particle swarm optimization of an extended Kalman filter for speed and rotor flux estimation of an induction motor drive," *Electrical Engineering*, vol. 97, pp. 129–138, Jun. 2015 (cit. on p. 70).

[47] V. Bavdekar, A. Deshpande, and S. Patwardhan, "Identification of process and measurement noise covariance for state and parameter estimation using extended Kalman filter," *Journal of Process Control*, vol. 21, pp. 585–601, 2011 (cit. on p. 71).

List of Figures

List of Tables

A Appendix

A.1 Simulate Radar Hardware and Signal Processing Chain

```
close all
clc
% Set random number generator for repeatable results
rng(2017);

% Compute hardware parameters from specified long-range requirements
fc = 77e9;                                  % Center frequency (Hz)
c = physconst('LightSpeed');                % Speed of light in air (m/s)
lambda = c/fc;                              % Wavelength (m)

rangeMax = 50; % Maximum range (m)
tm = 30*range2time(rangeMax,c); % Chirp duration (s)
rangeRes = 0.16; % Desired range resolution (m)
bw = range2bw(rangeRes,c); % Corresponding bandwidth (Hz)
% Set the sampling rate to satisfy both the range and velocity requirements
% for the radar
sweepSlope = bw/tm;                         % FMCW sweep slope (Hz/s)
fbeatMax = range2beat(rangeMax,sweepSlope,c);   % Maximum beat frequency (Hz)

vMax = 230*1000/3600;                       % Maximum Velocity of targets (m/s)
fdopMax = speed2dop(2*vMax,lambda);         % Maximum Doppler shift (Hz)

fifMax = fbeatMax+fdopMax;                  % Maximum received intermediate
    frequency (IF) (Hz)
fs = max(2*fifMax,bw);                      % Sampling rate (Hz)

% Configure the FMCW waveform using the waveform parameters derived from
% the long-range requirements
waveform = phased.FMCWWaveform('SweepTime',tm,'SweepBandwidth',bw,'SampleRate',fs);
Nsweep = 192;

% Model the antenna element
antElmnt = phased.IsotropicAntennaElement('BackBaffled',true);

% Construct the receive array
Ne = 6;
rxArray = phased.ULA('Element',antElmnt,'NumElements',Ne,'ElementSpacing',lambda/2)
    ;

% Form forward-facing beam to detect objects in front of the ego vehicle
beamformer = phased.PhaseShiftBeamformer('SensorArray',rxArray,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'Direction',[0;0]);

% Direction-of-arrival estimator for linear phased array signals
doaest = phased.RootMUSICEstimator(...
    'SensorArray',rxArray,...
    'PropagationSpeed',c,'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',1);

% Scan beams in front of ego vehicle for range-angle image display
angscan = -80:80;
beamscan = phased.PhaseShiftBeamformer('Direction',[angscan;0*angscan],...
```

```
    'SensorArray',rxArray,'OperatingFrequency',fc);
antAperture = 6.06e-4;                          % Antenna aperture (m^2)
antGain = aperture2gain(antAperture,lambda);    % Antenna gain (dB)

txPkPower = db2pow(5)*1e-3;                     % Tx peak power (W)
txGain = antGain;                               % Tx antenna gain (dB)

rxGain = antGain;                               % Rx antenna gain (dB)
rxNF = 4.5;                                     % Receiver noise figure (dB)

% Waveform transmitter
transmitter = phased.Transmitter('PeakPower',txPkPower,'Gain',txGain);

% Radiator for single transmit element
radiator = phased.Radiator('Sensor',antElmnt,'OperatingFrequency',fc);

% Collector for receive array
collector = phased.Collector('Sensor',rxArray,'OperatingFrequency',fc);

% Receiver preamplifier
receiver = phased.ReceiverPreamp('Gain',rxGain,'NoiseFigure',rxNF,'SampleRate',fs);

%Define Radar Signal Processing Chain
Nft = waveform.SweepTime*waveform.SampleRate/15;% Number of fast-time samples
%Nft = waveform.SweepTime*waveform.SampleRate;
Nst = Nsweep;                                   % Number of slow-time samples
Nr = 2^nextpow2(Nft);                           % Number of range samples after
    processing
Nd = 2^nextpow2(Nst);                           % Number of Doppler samples after
    processing
rxsig1 = zeros(Nft,Ne);
rngdopresp = phased.RangeDopplerResponse('RangeMethod','FFT','DopplerOutput','Speed
    ',...
    'SweepSlope',sweepSlope,...
    'RangeFFTLengthSource','Property','RangeFFTLength',Nr,...
    'RangeWindow','Hann',...
    'DopplerFFTLengthSource','Property','DopplerFFTLength',Nd,...
    'DopplerWindow','Hann',...
    'PropagationSpeed',c,'OperatingFrequency',fc,'SampleRate',fs/15);

%CFAR Detector
% Guard cell and training regions for range dimension
nGuardRng = 4;
nTrainRng = 4;
nCUTRng = 1+nGuardRng+nTrainRng;

% Guard cell and training regions for Doppler dimension
dopOver = round(Nd/Nsweep);
nGuardDop = 4*dopOver;
nTrainDop = 4*dopOver;
nCUTDop = 1+nGuardDop+nTrainDop;

cfar = phased.CFARDetector2D('GuardBandSize',[nGuardRng nGuardDop],...
    'TrainingBandSize',[nTrainRng nTrainDop],...
    'ThresholdFactor','Custom','CustomThresholdFactor',db2pow(13),...
    'NoisePowerOutputPort',true,'OutputFormat','Detection index');

% Perform CFAR processing over all of the range and Doppler cells
freqs = ((0:Nr-1)'/Nr-0.5)*fs;
rnggrid = beat2range(freqs,sweepSlope);
iRngCUT = find(rnggrid>0);
iRngCUT = iRngCUT(iRngCUT<=Nr-nCUTRng+1);
iDopCUT = nCUTDop:(Nd-nCUTDop+1);
[iRng,iDop] = meshgrid(iRngCUT,iDopCUT);
idxCFAR = [iRng(:) iDop(:)]';

rmsRng = sqrt(12)*rangeRes;
%range estimation of target
rngestimator = phased.RangeEstimator('ClusterInputPort',true,...
```

```
    'VarianceOutputPort',true,'NoisePowerSource','Input port','RMSResolution',
        rmsRng);
%doppler estimation of target
dopestimator = phased.DopplerEstimator('ClusterInputPort',true,...
    'VarianceOutputPort',true,'NoisePowerSource','Input port','NumPulses',Nsweep);

%Model Free Space Propagation Channel
channel = phased.FreeSpace('PropagationSpeed',c,'OperatingFrequency',fc,...
    'SampleRate',fs,'TwoWayPropagation',true);
```

A.2 Create a driving scenario

```
function [scenario,egoVehicle,radarParams,tgts] = createDrivingScenario(c,fc,
    egoSpeed,targetSpeed)
%c is the speed of light
%fc is the carrier frequency of FMCW Wave
%egoSpeed is the speed of the ego vehicle
%targetSpeed is the speed of the target vehicle
% Construct a drivingScenario object.
scenario = drivingScenario;
scenario.SampleTime = 0.1;
scenario.StopTime = 1.1;

% Add all road segments
roadCenters = [0 0 0;
    50 0 0;
    100 0 0;
    250 20 0;
    500 40 0];
laneSpecification = lanespec(2, 'Width', 4); %two lane highway with each lane being
     4 meter in width
road(scenario, roadCenters, 'Lanes', laneSpecification);

% Add the ego vehicle
egoVehicle = vehicle(scenario, ...
    'ClassID', 1, ...
    'Position', [2.53 -0.12 0.01]);

% Add the non-ego actors
pedestrian = actor(scenario, ...
    'ClassID', 4, ...
    'Length', 0.24, ...
    'Width', 0.45, ...
    'Height', 1.7, ...
    'Position', [8.35 -0.04 0.01], ...
    'RCSPattern', [-8 -8;-8 -8]);
waypoints = [8.35 -0.04 0.01;
    9.85 1.68 0.01;
    13.43 3.39 0.01;
    17.16 2.27 0.01;
    19.18 -1.54 0.01;
    21.64 -2.43 0.01;
    23.36 -1.91 0.01;
    24.78 0.56 0.01;
    22.39 3.32 0.01;
    19.4 2.27 0.01;
    16.79 -1.83 0.01;
    12.09 -3.03 0.01;
    9.4 -0.49 0.01];
speed = targetSpeed;
trajectory(pedestrian, waypoints, speed);
%characterise the radar sensor in the ego car
radarPos = [3.4 0 0.2]';
radarYaw = 0;
radarPitch = 0;
radarRoll = 0;
radarAxes = rotz(radarYaw)*roty(radarPitch)*rotx(radarRoll);
```

```
radarParams = struct( ...
    'Frame', drivingCoordinateFrameType.Spherical, ... %radar detections in
        spherical coordinates
    'OriginPosition',radarPos, ...
    'OriginVelocity',zeros(3,1), ...
    'Orientation', radarAxes, ...
    'HasElevation', false, ...
    'HasVelocity', true, ...
    'RMSBias',[0 0.25 0.05]);

tgts = createPointTargets(scenario,c,fc);
end

function tgts = createPointTargets(scenario,c,fc)
% Returns point targets modeled using the |phased.BackscatterRadarTarget|
% object. The point target models are loaded from the point target models
% stored in the driving scenario's actor profiles.

profiles = actorProfiles(scenario);
profiles = profiles(2:end); % First profile is the ego car, not a target
numTgts = numel(profiles);

azAngles = profiles(1).RCSAzimuthAngles;
elAngles = profiles(1).RCSElevationAngles;
rcsPatterns = NaN(numel(elAngles),numel(azAngles),numTgts);
for m = 1:numTgts
    rcsPatterns(:,:,m) = db2mag(profiles(m).RCSPattern);
end
tgts = phased.BackscatterRadarTarget( ...
    'RCSPattern', rcsPatterns, ...
    'AzimuthAngles', azAngles, ...
    'ElevationAngles', elAngles, ...
    'PropagationSpeed', c,...
    'OperatingFrequency', fc);
end
```

A.3 Generate time-aggregated point clouds

```
%create driving scenario
[scenario,egoCar,radarParams,pointTgts] = createDrivingScenario(c,fc,
    egoVehicleSpeed,targetVehicleSpeed1,targetVehicleSpeed1);
tgtProfiles = actorProfiles(scenario); %target profiles
tgtProfiles = tgtProfiles(2:end); %profiles of all actors in the scenario except
    ego vehicle
tgtHeight = [tgtProfiles.Height];

% Run the simulation loop
sweepTime = waveform.SweepTime;
nFrame = 1;
pc = {};
while advance(scenario)

    % Get the current scenario time
    time = scenario.SimulationTime;

    % Get current target poses in ego vehicle's reference frame
    tgtPoses = targetPoses(egoCar);
    tgtPos = reshape([tgtPoses.Position],3,[]);
    tgtPos(3,:) = tgtPos(3,:)+0.5*tgtHeight; % Position point targets at half of
        each target's height
    tgtVel = reshape([tgtPoses.Velocity],3,[]);

    % Assemble data cube at current scenario time
    Xcube = zeros(Nft,Ne,Nsweep);
    for m = 1:Nsweep

        % Calculate angles of the targets viewed by the radar
```

```
27          tgtAngs = NaN(2,numel(tgtPoses));
28          for iTgt = 1:numel(tgtPoses)
29              tgtAxes = rotz(tgtPoses(iTgt).Yaw)*roty(tgtPoses(iTgt).Pitch)*rotx(
                    tgtPoses(iTgt).Roll);
30              [~,tgtAngs(:,iTgt)] = rangeangle(radarParams.OriginPosition,tgtPos(:,
                    iTgt),tgtAxes);
31          end
32
33          % Transmit FMCW waveform
34          sig = waveform();
35          [~,txang] = rangeangle(tgtPos,radarParams.OriginPosition,radarParams.
                Orientation);
36          txsig = transmitter(sig);
37          txsig = radiator(txsig,txang);
38          txsig = channel(txsig,radarParams.OriginPosition,tgtPos,radarParams.
                OriginVelocity,tgtVel);
39
40          % Propagate the signal and reflect off the target
41          tgtsig = pointTgts(txsig,tgtAngs);
42
43          % Collect received target echos
44          rxsig = collector(tgtsig,txang);
45          rxsig = receiver(rxsig);
46
47          % Dechirp the received signal
48          rxsig = dechirp(rxsig,sig);
49
50          %decimate dechirped signal
51           for p = 1:Ne
52              rxsig1(:,p) = decimate(rxsig(:,p),15,'fir');
53          end
54          rxsig = decimate(rxsig,16);
55          % Save sweep to data cube
56         Xcube(:,:,m) = rxsig1;
57         %Xcube(:,:,m) = rxsig;
58          rxsig1 = zeros(Nft,Ne);
59          % Move targets forward in time for next sweep
60          tgtPos = tgtPos+tgtVel*sweepTime;
61      end
62
63      % Calculate the range-Doppler response
64      [Xrngdop,rnggrid,dopgrid] = rngdopresp(Xcube);
65
66
67      % Beamform received data
68      Xbf = permute(Xrngdop,[1 3 2]);
69      Xbf = reshape(Xbf,Nr*Nd,Ne);
70      Xbf = beamformer(Xbf);
71
72
73      Xbf = reshape(Xbf,Nr,Nd);
74      % Detect targets
75      Xpow = abs(Xbf).^2;
76
77      [detidx,noisepwr] = cfar(Xpow,idxCFAR);
78
79      % Cluster detections
80      clusterIDs = helperAutoDrivingRadarSigProc('Cluster Detections',detidx);
81
82      % Estimate azimuth, range, and radial speed measurements for individual
            detections
83      [azest1,azvar1,snrdB1] = helperAutoDrivingRadarSigProc('Estimate Angle',doaest,
            conj(Xrngdop),Xbf,detidx,noisepwr,NaN(size(detidx,1)));
84      [rngest1,rngvar1] = rngestimator1(Xbf,rnggrid,detidx,noisepwr);
85
86      [rsest1,rsvar1] = dopestimator1(Xbf,dopgrid,detidx,noisepwr);
87
88      pc{nFrame}.rngdopval(:,:,:) = [rngest1.*cos(azest1*pi/180),rngest1.*sin(azest1*
            pi/180),rsest1];
```

```
89
90      nFrame = nFrame+1;
91  end
92
93  %Aggegating Radar Data for 10 consecutive frames and storing in xls files
94  pntcld = {};
95  a  = 1;
96   for k=1:nFrame-1
97       if k<=10
98           %aggregate detections from 10 past frames
99           pntcld = [pntcld ; (pc{k}.rngdopval)];
100      else
101          for p=k-10:k
102           %aggregate detections from 10 past frames
103           pntcld = [pntcld ; (pc{p}.rngdopval)];
104          end
105     end
106     if k/10 >= 1
107         pntcld = cell2mat(pntcld);
108         %store aggregated point cloud in .xls file for training the PointNet
109         %later
110         writematrix(pntcld1,strcat('Scenario_5_Pedestrian_Target_Ego',num2str(
                egoVehicleSpeed),'_Pedestrian_Crossover_',num2str(targetVehicleSpeed),'_
                ',num2str(a),'.xls'));
111         movefile(strcat('Scenario_5_Pedestrian_Target_Ego',num2str(egoVehicleSpeed),
                '_Pedestrian_Crossover_',num2str(targetVehicleSpeed),'_',num2str(a),'.
                xls'),'C:\Subhasri Mitra\Crossover Tracking\CrossoverTest1\Pedestrian');
112         pntcld = {};
113         a=a+1;
114     end
115  end
```

A.4 PointNet simulation

Code availabe at: Mathworks:Point Cloud Classification Using PointNet Deep Learning

A.5 Filtering algorithms

Extended Kalman Filter implementation

```
1  function [X,P,error] = implementEKF1(prevState,prevStateCov,currentMeas,R)
2  % Inputs:   stateTrans: state transition matrix
3  %           prevState: state at k-1
4  %           prevStateCov:state covariance at k-1
5  %           currentMeas: current measurement at k
6  %           measFunc: measurement function handle
7  %           measJac: measurement function Jacobian
8  %           Q: process noise covariance
9  %           R: measurement noise covariance
10 % Output:   X: state at k
11 %           P: state covariance at k
12 %           error: Normalized Innovation Squared
13
14
15 if (all(prevState == 0))
16     [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
17      prevStateCov = 0.01*eye(4);
18 end
19 %state Transition Matrix
20 dt = 0.1;
21 sigma = 1.0;
22 %State Transition Matrix
23 stateTrans = eye(4);
```

```
stateTrans(1,3) = dt;
stateTrans(2,4)= dt;
%Process Covariance Matrix
Q = sigma .* [(dt^4)/4, 0, (dt^3)/2, 0;...
      0, (dt^4)/4, 0, (dt^3)/2;...
      (dt^3)/2, 0, (dt^2), 0;...
      0, (dt^3)/2, 0, (dt^2)];
%state prediction
x_est = stateTrans * prevState;
P_est = (stateTrans * prevStateCov) * stateTrans' + Q;
% Update step
P_x = x_est(1);
P_y = x_est(2);
V_x = x_est(3);
V_y = x_est(4);
%Measurement Function
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
measFunc = zeros(3,1);
measFunc(1:3,1) = [theta; rho; rho_dot];
%Error in measurement
y = currentMeas - measFunc;

% Measurement Jacobian
x_ = x_est(1);
y_ = x_est(2);
vx_ = x_est(3);
vy_ = x_est(4);
measJac = zeros(3,4);
measJac(1:3,1:4) = [-y_/(x_^2 + y_^2), x_/(x_^2 + y_^2),0,0;...
            x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2),0,0;...
            (y_*(vx_*y_ - vy_*x_))/((x_^2 + y_^2)^1.5),(x_*(vy_*x_ - vx_*y_))/((x_
                ^2 + y_^2)^1.5),x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2)];

S = measJac * P_est * measJac'  + R;
%kalman Gain
K = P_est * measJac' * inv(S);
X = x_est + K*y;
P = P_est - K*measJac*P_est;
%calculateMetric(prevState,S,y,frame_count_new);

%NIS Metric
error = y'*inv(S)*y;
end
```

Unscented Kalman Filter implementation

```
function [x,P,error]=implementUKF1(prevState,prevStateCov,currentMeas,R)
% UKF   Unscented Kalman Filter for nonlinear dynamic systems
% [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance, P
% for nonlinear dynamic system (for simplicity, noises are assumed as additive):
%           x_k+1 = f(x_k) + w_k
%           z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%       v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:   f: function handle for f(x)
%           prevState: "a priori" state estimate
%           prevStateCov: "a priori" estimated state covariance
%           h: fanction handle for h(x)
%           currentMeas: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance
%           error: Normalized Innovation Squared
if (all(prevState == 0))
    [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
```

```
21      prevStateCov = 0.009*eye(4);
22  end
23  %state Transition Matrix
24  dt = 0.1;
25  sigma = 0.1;
26  %State Transition Matrix
27  fstate = [1,0,dt,0;...
28                0,1,0,dt;...
29                0,0,1,0;...
30                0,0,0,1];
31  %Process Covariance Matrix
32  Q = sigma .* [(dt^4)/4, 0, (dt^3)/2, 0;...
33        0, (dt^4)/4, 0, (dt^3)/2;...
34        (dt^3)/2, 0, (dt^2), 0;...
35        0, (dt^3)/2, 0, (dt^2)];
36
37  % Q = 0.02*eye(6);
38  L=numel(prevState);                                    %numer of states
39  m=numel(currentMeas);                                    %numer of measurements
40  alpha=1e-3;                                         %default, tunable
41  ki=0;                                               %default, tunable
42  beta=2;                                             %default, tunable
43  lambda=alpha^2*(L+ki)-L;                            %scaling factor
44  c=L+lambda;                                         %scaling factor
45  Wm=[lambda/c 0.5/c+zeros(1,2*L)];                   %weights for means
46  Wc=Wm;
47  Wc(1)=Wc(1)+(1-alpha^2+beta);                       %weights for covariance
48  c=sqrt(c);
49  X=sigmas(prevState,prevStateCov,c);
50  %disp(X);
51  %sigma points around x
52  f = fstate*X;
53  [x1,X1,P1,X2]=ut(f,X,Wm,Wc,L,Q);                  %unscented transformation of process
54  % X1=sigmas(x1,P1,c);                                  %sigma points around x1
55  % X2=X1-x1(:,ones(1,size(X1,2)));
56  %deviation of X1
57  hmeas = measurementFunction(X);
58  [z1,Z1,P2,Z2]=ut(hmeas,X,Wm,Wc,m,R);               %unscented transformation of measurments
59  P12=X2*diag(Wc)*Z2';                                %transformed cross-covariance
60  K=P12*inv(P2);
61  x=x1+K*(currentMeas-z1);                                    %state update
62  P=P1-K*P12'*P1;  %change made:last term added
63  %NSI Metric
64  error = (currentMeas-z1)'*inv(P2)*(currentMeas-z1);
65  end
66  %covariance update
67  function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
68  %Unscented Transformation
69  %Input:
70  %         f: nonlinear map
71  %         X: sigma points
72  %        Wm: weights for mean
73  %        Wc: weights for covraiance
74  %         n: numer of outputs of f
75  %         R: additive covariance
76  %Output:
77  %         y: transformed mean
78  %         Y: transformed smapling points
79  %         P: transformed covariance
80  %        Y1: transformed deviations
81  L=size(X,2);
82  y=zeros(n,1);
83  Y=zeros(n,L);
84  for k=1:L
85      Y(:,k)=f(:,k);
86      y=y+Wm(k)*Y(:,k);
87  end
88  Y1=Y-y(:,ones(1,L));
89  P=Y1*diag(Wc)*Y1'+R;
```

```
end
function X=sigmas(x,P,c)
%Sigma points around reference point
%Inputs:
%       x: reference point
%       P: covariance
%       c: coefficient
%Output:
%       X: Sigma points
A = c*chol(P)';
Y = x(:,ones(1,numel(x)));
X = [x Y+A Y-A];
end

function hmeas = measurementFunction(X)
%calculate the measurement function at the sigma points
hmeas = zeros(3,size(X,2));
for i = 1:size(X,2)
P_x = X(1,i);
P_y = X(2,i);
V_x = X(3,i);
V_y = X(4,i);
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
hmeas(:,i) = [theta; rho; rho_dot];
end
end
```

A.6 Dimension Based Tracking Simultion

Dimension Based EKF

```
function [X,P,error] = implementEKF1(prevState,prevStateCov,currentMeas,R)
% Inputs:   stateTrans: state transition matrix
%           prevState: state at k-1
%           prevStateCov:state covariance at k-1
%           currentMeas: current measurement at k
%           measFunc: measurement function handle
%           measJac: measurement function Jacobian
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   X: state at k
%           P: state covariance at k
%           error: Normalized Innovation Squared

if (all(prevState == 0))
    [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
     prevState(5:6) = currentMeas(4:5);
     prevStateCov = 0.009*eye(6);
end
%state Transition Matrix
dt = 0.1;
sigma = 1.5;
%State Transition Matrix
stateTrans = eye(6);
stateTrans(1,3) = dt;
stateTrans(2,4)= dt;
%Process Covariance Matrix
Q = sigma .* [(dt^4)/4, 0, (dt^3)/2, 0, 0, 0;...
      0, (dt^4)/4, 0, (dt^3)/2, 0, 0;...
      (dt^3)/2, 0, (dt^2), 0, 0, 0;...
      0, (dt^3)/2, 0, (dt^2), 0, 0;...
      0, 0, 0, 0, 0.0001, 0;...
      0, 0, 0, 0, 0, 0.0001];
%state prediction
```

```
x_est = stateTrans * prevState;
P_est = (stateTrans * prevStateCov) * stateTrans' + Q;
% Update step
P_x = x_est(1);
P_y = x_est(2);
V_x = x_est(3);
V_y = x_est(4);
w = x_est(5);
h = x_est(6);
%Measurement Function
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
measFunc = zeros(5,1);
measFunc(1:3,1) = [theta; rho; rho_dot];
measFunc(4:5,1) = x_est(5:6);
%Error in measurement
y = currentMeas - measFunc;

% Measurement Jacobian
x_ = x_est(1);
y_ = x_est(2);
vx_ = x_est(3);
vy_ = x_est(4);
measJac = zeros(5,6);
measJac(1:3,1:4) = [-y_/(x_^2 + y_^2), x_/(x_^2 + y_^2),0,0;...
            x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2),0,0;...
            (y_*(vx_*y_ - vy_*x_))/((x_^2 + y_^2)^1.5),(x_*(vy_*x_ - vx_*y_))/((x_
                ^2 + y_^2)^1.5),x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2)];

measJac(4:5,5:6) = eye(2);
S = measJac * P_est * measJac'  + R;
%kalman Gain
K = P_est * measJac' * inv(S);
X = x_est + K*y;
P = P_est - K*measJac*P_est;
%NIS
error = y'*inv(S)*y;
end
```

Dimension Based UKF

```
function [x,P,error]=implementUKF1(prevState,prevStateCov,currentMeas,R)
% UKF   Unscented Kalman Filter for nonlinear dynamic systems
% [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance, P
% for nonlinear dynamic system (for simplicity, noises are assumed as additive):
%           x_k+1 = f(x_k) + w_k
%           z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%       v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:   f: function handle for f(x)
%           prevState: "a priori" state estimate
%           prevStateCov: "a priori" estimated state covariance
%           h: fanction handle for h(x)
%           currentMeas: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance
%           Normalized Innovation Squared
if (all(prevState == 0))
    [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
     prevState(5:6) = currentMeas(4:5);
     prevStateCov = 0.009*eye(6);
end
%state Transition Matrix
dt = 0.1;
sigma = 0.1;
```

```
%State Transition Matrix
fstate = [1,0,dt,0,0,0;...
              0,1,0,dt,0,0;...
              0,0,1,0,0,0;...
              0,0,0,1,0,0;...
              0,0,0,0,1,0;...
              0,0,0,0,0,1];
%Process Covariance Matrix
Q = sigma .* [(dt^4)/4, 0, (dt^3)/2, 0, 0, 0;...
      0, (dt^4)/4, 0, (dt^3)/2, 0, 0;...
      (dt^3)/2, 0, (dt^2), 0, 0, 0;...
      0, (dt^3)/2, 0, (dt^2), 0, 0;...
      0, 0, 0, 0, 0.0001, 0;...
      0, 0, 0, 0, 0, 0.0001];

% Q = 0.02*eye(6);
L=numel(prevState);                         %numer of states
m=numel(currentMeas);                       %numer of measurements
alpha=1e-3;                                 %default, tunable
ki=0;                                       %default, tunable
beta=2;                                     %default, tunable
lambda=alpha^2*(L+ki)-L;                    %scaling factor
c=L+lambda;                                 %scaling factor
Wm=[lambda/c 0.5/c+zeros(1,2*L)];           %weights for means
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);               %weights for covariance
c=sqrt(c);
X=sigmas(prevState,prevStateCov,c);
%disp(X);
%sigma points around x
f = fstate*X;
[x1,X1,P1,X2]=ut(f,X,Wm,Wc,L,Q);          %unscented transformation of process
% X1=sigmas(x1,P1,c);                         %sigma points around x1
% X2=X1-x1(:,ones(1,size(X1,2)));
%deviation of X1
hmeas = measurementFunction(X);
[z1,Z1,P2,Z2]=ut(hmeas,X,Wm,Wc,m,R);       %unscented transformation of measurments
P12=X2*diag(Wc)*Z2';                        %transformed cross-covariance
K=P12*inv(P2);
x=x1+K*(currentMeas-z1);                                %state update
P=P1-K*P12'*P1;  %change made:last term added
%NIS
error = (currentMeas-z1)'*inv(P2)*(currentMeas-z1);
end
%covariance update
function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
%Unscented Transformation
%Input:
%        f: nonlinear map
%        X: sigma points
%       Wm: weights for mean
%       Wc: weights for covraiance
%        n: numer of outputs of f
%        R: additive covariance
%Output:
%        y: transformed mean
%        Y: transformed smapling points
%        P: transformed covariance
%       Y1: transformed deviations
L=size(X,2);
y=zeros(n,1);
Y=zeros(n,L);
for k=1:L
    Y(:,k)=f(:,k);
    y=y+Wm(k)*Y(:,k);
end
Y1=Y-y(:,ones(1,L));
P=Y1*diag(Wc)*Y1'+R;
end
```

```
function X=sigmas(x,P,c)
%Sigma points around reference point
%Inputs:
%       x: reference point
%       P: covariance
%       c: coefficient
%Output:
%       X: Sigma points
A = c*chol(P)';
Y = x(:,ones(1,numel(x)));
X = [x Y+A Y-A];
end

function hmeas = measurementFunction(X)
%calculate the measurement function at the sigma points
hmeas = zeros(5,size(X,2));
for i = 1:size(X,2)
P_x = X(1,i);
P_y = X(2,i);
V_x = X(3,i);
V_y = X(4,i);
w = X(5,i);
h = X(6,i);
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
width = w;
height = h;
hmeas(:,i) = [theta; rho; rho_dot; width; height];
end
end
```

A.7 Feature Based Tracking Simultion

Feature Based EKF

```
function [X,P,error] = implementEKF1(prevState,prevStateCov,currentMeas,R)
% Inputs:   stateTrans: state transition matrix
%           prevState: state at k-1
%           prevStateCov:state covariance at k-1
%           currentMeas: current measurement at k
%           measFunc: measurement function handle
%           measJac: measurement function Jacobian
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   X: state at k
%           P: state covariance at k
%           error: Normalized Innovation Squared

if (all(prevState == 0))
    [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
     prevState(5:22) = currentMeas(4:21);

prevStateCov = 0.05*eye(22);
end
%state Transition Matrix
dt = 0.1;
%sigma = 30;
%State Transition Matrix
stateTrans = eye(22);
stateTrans(1,3) = dt;
stateTrans(2,4)= dt;
%Process Covariance Matrix
Q = 0.0003*eye(22,22);
%state prediction
x_est = stateTrans * prevState;
```

```
P_est = (stateTrans * prevStateCov) * stateTrans' + Q;
% Update step
P_x = x_est(1);
P_y = x_est(2);
V_x = x_est(3);
V_y = x_est(4);
w = x_est(5);
h = x_est(6);
%Measurement Function
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
width = w;
height = h;
measFunc = zeros(21,1);
measFunc(1:5,1) = [theta; rho; rho_dot; width; height];
measFunc(6:21,1) = x_est(7:22);
%Error in measurement
y = currentMeas - measFunc;

% Measurement Jacobian
x_ = x_est(1);
y_ = x_est(2);
vx_ = x_est(3);
vy_ = x_est(4);
measJac = zeros(21,22);
measJac(1:3,1:4) = [-y_/(x_^2 + y_^2), x_/(x_^2 + y_^2),0,0;...
            x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2),0,0;...
            (y_*(vx_*y_ - vy_*x_))/((x_^2 + y_^2)^1.5),(x_*(vy_*x_ - vx_*y_))/((x_
                ^2 + y_^2)^1.5),x_/sqrt(x_^2 + y_^2),y_/sqrt(x_^2 + y_^2)];

measJac(4:21,5:22) = eye(18);
S = measJac * P_est * measJac'  + R;
%kalman Gain
K = P_est * measJac' * inv(S);
X = x_est + K*y;
P = P_est - K*measJac*P_est;
%calculateMetric(prevState,S,y,frame_count_new);

%NSI Metric
error = y'*inv(S)*y;
end
```

Feature Based UKF

```
function [x,P,error]=implementUKFModified(prevState,prevStateCov,currentMeas,R)
% UKF   Unscented Kalman Filter for nonlinear dynamic systems
% [x, P] = ukf(f,x,P,h,z,Q,R) returns state estimate, x and state covariance, P
% for nonlinear dynamic system (for simplicity, noises are assumed as additive):
%           x_k+1 = f(x_k) + w_k
%           z_k   = h(x_k) + v_k
% where w ~ N(0,Q) meaning w is gaussian noise with covariance Q
%       v ~ N(0,R) meaning v is gaussian noise with covariance R
% Inputs:   f: function handle for f(x)
%           prevState: "a priori" state estimate
%           prevStateCov: "a priori" estimated state covariance
%           h: fanction handle for h(x)
%           currentMeas: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance
%           error: Normalized Innovation Squared
if (all(prevState == 0))
    [prevState(1), prevState(2), ~] = sph2cart(currentMeas(1),0,currentMeas(2));
     prevState(5:22) = currentMeas(4:21);

prevStateCov = 0.032*eye(22);
```

```
end

%Process Covariance Matrix
dt = 0.1;
sigma = 1;
%State Transition Matrix
fstate = eye(22);
fstate(1,3) = dt;
fstate(2,4)= dt;
%Process Covariance Matrix
Q = sigma*0.001*eye(22,22);
L=numel(prevState);                          %number of states
m=numel(currentMeas);                        %number of measurements
alpha=1e-3;                                  %default, tunable
ki=0;                                        %default, tunable
beta=2;    %default, tunable
lambda=alpha^2*(L+ki)-L;
c=L+lambda;                                  %scaling factor
Wm=[lambda/c 0.5/c+zeros(1,2*L)];            %weights for means
Wc=Wm;
Wc(1)=Wc(1)+(1-alpha^2+beta);                %weights for covariance
c=sqrt(c);
X=sigmas(prevState,prevStateCov,c);
%sigma points around x
f = fstate*X;
[x1,X1,P1,X2]=ut(f,X,Wm,Wc,L,Q);           %unscented transformation of process
%deviation of X1
hmeas = measurementFunction(X);
[z1,Z1,P2,Z2]=ut(hmeas,X,Wm,Wc,m,R);         %unscented transformation of
    measurments
P12=X2*diag(Wc)*Z2';                         %transformed cross-covariance
K=P12*inv(P2);
x=x1+K*(currentMeas-z1);                     %state update
P=P1-K*P12'*P1;
%NSI Metric
y=currentMeas-z1;
error = (currentMeas-z1)'*inv(P2)*(currentMeas-z1);
end
%covariance update
function [y,Y,P,Y1]=ut(f,X,Wm,Wc,n,R)
%Unscented Transformation
%Input:
%        f: nonlinear map
%        X: sigma points
%       Wm: weights for mean
%       Wc: weights for covraiance
%        n: numer of outputs of f
%        R: additive covariance
%Output:
%        y: transformed mean
%        Y: transformed smapling points
%        P: transformed covariance
%       Y1: transformed deviations
L=size(X,2);
y=zeros(n,1);
Y=zeros(n,L);
for k=1:L
    Y(:,k)=f(:,k);
    y=y+Wm(k)*Y(:,k);
end
Y1=Y-y(:,ones(1,L));
P=Y1*diag(Wc)*Y1'+R;
end
function X=sigmas(x,P,c)
%Sigma points around reference point
%Inputs:
%       x: reference point
%       P: covariance
%       c: coefficient
```

```
%Output:
%       X: Sigma points
A = c*chol(P)';
Y = x(:,ones(1,numel(x)));
X = [x Y+A Y-A];
end

function hmeas = measurementFunction(X)
%calculate the measurement function at the sigma points
hmeas = zeros(21,size(X,2));
for i = 1:size(X,2)
P_x = X(1,i);
P_y = X(2,i);
V_x = X(3,i);
V_y = X(4,i);
theta = atan(P_y/P_x);
rho =  sqrt(P_x*P_x + P_y*P_y);
rho_dot = (P_x*V_x + P_y*V_y) / rho;
hmeas(:,i) = [theta; rho; rho_dot; X(5:22,i)];
end
end
```

A.8 Tracking in real-time

All helper functions availabe at: Radar Signal Simulation and Processing for Automated Driving

```
% Run the simulation loop
sweepTime = waveform.SweepTime;
nFrame = 1;
pc = {};
pntcld = {};
nis_error = zeros(20,1);
a  = 0;
ekf_tracks = {};
prevState = zeros(10,22); %initialise previous state of kalman filer
prevStateCov = zeros(22,22,6); %initialise previous state covariance of kalman
    filer
object_features = load('TestedFeatures1.mat'); %load pre-extracted global features
    from PointNet
while advance(scenario)

    % Get the current scenario time
    time = scenario.SimulationTime;

    % Get current target poses in ego vehicle's reference frame
    tgtPoses = targetPoses(egoCar);
    tgtPos = reshape([tgtPoses.Position],3,[]);
    tgtPos(3,:) = tgtPos(3,:)+0.5*tgtHeight; % Position point targets at half of
        each target's height
    tgtVel = reshape([tgtPoses.Velocity],3,[]);

    % Assemble data cube at current scenario time
    Xcube = zeros(Nft,Ne,Nsweep);
    for m = 1:Nsweep

        % Calculate angles of the targets viewed by the radar
        tgtAngs = NaN(2,numel(tgtPoses));
        for iTgt = 1:numel(tgtPoses)
            tgtAxes = rotz(tgtPoses(iTgt).Yaw)*roty(tgtPoses(iTgt).Pitch)*rotx(
                tgtPoses(iTgt).Roll);
            [~,tgtAngs(:,iTgt)] = rangeangle(radarParams.OriginPosition,tgtPos(:,
                iTgt),tgtAxes);
        end

        % Transmit FMCW waveform
        sig = waveform();
```

```
36          [~,txang] = rangeangle(tgtPos,radarParams.OriginPosition,radarParams.
                Orientation);
37          txsig = transmitter(sig);
38          txsig = radiator(txsig,txang);
39          txsig = channel(txsig,radarParams.OriginPosition,tgtPos,radarParams.
                OriginVelocity,tgtVel);
40
41          % Propagate the signal and reflect off the target
42          tgtsig = pointTgts(txsig,tgtAngs);
43
44          % Collect received target echos
45          rxsig = collector(tgtsig,txang);
46          rxsig = receiver(rxsig);
47
48          % Dechirp the received signal
49          rxsig = dechirp(rxsig,sig);
50
51          %decimate dechirped signal
52           for p = 1:Ne
53              rxsig1(:,p) = decimate(rxsig(:,p),15,'fir');
54          end
55          rxsig = decimate(rxsig,16);
56          % Save sweep to data cube
57         Xcube(:,:,m) = rxsig1;
58         %Xcube(:,:,m) = rxsig;
59          rxsig1 = zeros(Nft,Ne);
60          % Move targets forward in time for next sweep
61          tgtPos = tgtPos+tgtVel*sweepTime;
62      end
63
64      % Calculate the range-Doppler response
65      [Xrngdop,rnggrid,dopgrid] = rngdopresp(Xcube);
66
67
68      % Beamform received data
69      Xbf = permute(Xrngdop,[1 3 2]);
70      Xbf = reshape(Xbf,Nr*Nd,Ne);
71      Xbf = beamformer(Xbf);
72
73
74      Xbf = reshape(Xbf,Nr,Nd);
75      % Detect targets
76      Xpow = abs(Xbf).^2;
77
78      [detidx,noisepwr] = cfar(Xpow,idxCFAR);
79
80      % Cluster detections
81      clusterIDs = helperAutoDrivingRadarSigProc('Cluster Detections',detidx);
82
83      % Estimate azimuth, range, and radial speed measurements
84      %with clustering
85      [azest,azvar,snrdB] = helperAutoDrivingRadarSigProc('Estimate Angle',doaest,
            conj(Xrngdop),Xbf,detidx,noisepwr,clusterIDs);
86      azvar = azvar+radarParams.RMSBias(1)^2;
87
88      %without clustering
89      [azest1,azvar1,snrdB1] = helperAutoDrivingRadarSigProc('Estimate Angle',doaest,
            conj(Xrngdop),Xbf,detidx,noisepwr,NaN(size(detidx,1)));
90      %with clustering
91      [rngest,rngvar] = rngestimator(Xbf,rnggrid,detidx,noisepwr,clusterIDs);
92      %without clustering
93      [rngest1,rngvar1] = rngestimator1(Xbf,rnggrid,detidx,noisepwr);
94      rngvar = rngvar+radarParams.RMSBias(2)^2;
95      %with clustering
96      [rsest,rsvar] = dopestimator(Xbf,dopgrid,detidx,noisepwr,clusterIDs);
97      %without clustering
98      [rsest1,rsvar1] = dopestimator1(Xbf,dopgrid,detidx,noisepwr);
99      % Convert radial speed to range rate for use by the tracker
100     rrest = -rsest;
```

```
101     rrvar = rsvar;
102     rrvar = rrvar+radarParams.RMSBias(3)^2;
103     %store pointcloud in
104     pc{nFrame}.rngdopval(:,:,:) = [rngest1.*cos(azest1.*pi/180),rngest1.*sin(azest1
            .*pi/180),rsest1];
105
106
107     % Estimate clusterSize from Range Doppler Map
108     clusterDim = estimateClusterDimension(clusterIDs,detidx,rnggrid,dopgrid,azest);
109
110
111     % Assemble object detections for use by tracker
112     numDets = numel(rngest);
113     dets = cell(numDets,1);
114     for iDet = 1:numDets
115         %detections for built-in tracker
116         dets{iDet} = objectDetection(time,[azest(iDet) rngest(iDet) rrest(iDet)
                ]',...
117             'MeasurementNoise',diag([azvar(iDet) rngvar(iDet) rrvar(iDet)]),...
118             'MeasurementParameters',{radarParams},...
119             'ObjectAttributes',{struct('SNR',snrdB(iDet))});
120     end
121
122 %%
123     %Feature Based Filtering Approach
124         if nFrame>=10 % filtering starts from the 10th simulation steps
125               a=a+1;
126             %PointNet implementation code not supported by Matlab2019 hence
                    features have been pre-extracted in Matlab2020
127             feature = object_features.feature{1,a};
128           %end
129             for iDet = 1:numDets
130                 ekf_tracks{nFrame}.currentMeas{iDet} = [azest(iDet)*pi/180 rngest(
                        iDet) rrest(iDet) clusterDim(1,iDet) clusterDim(2,iDet) feature
                        '];
131                 ekf_tracks{nFrame}.R{iDet} = diag([azvar(iDet) rngvar(iDet) rrvar(
                        iDet) 0.03*ones(1,18)]); %0.03*ones(1,18) for EKF  %0.001*ones
                        (1,18) for UKF
132                 [state,stateCovariance,error,y] = implementEKF1(prevState(iDet,:)',
                        prevStateCov(:,:,iDet),ekf_tracks{nFrame}.currentMeas{iDet}',
                        ekf_tracks{nFrame}.R{iDet});
133                 ekf_tracks{nFrame}.prevState{iDet} = state;
134                 ekf_tracks{nFrame}.prevStateCov{iDet} = stateCovariance;
135                 ekf_tracks{nFrame}.measError{iDet} = y;
136                 prevState(iDet,:) = state;
137                 prevStateCov(:,:,iDet) = stateCovariance;
138                 nis_error(a+10) = error;
139                 %converting the position and velocity of targets from sensor to ego
                         coordinates
140                 Xradar = [ekf_tracks{nFrame}.prevState{iDet}(1,1);ekf_tracks{nFrame
                        }.prevState{iDet}(2,1);0];
141                 Xego = radarParams.Orientation*Xradar+radarParams.OriginPosition;
142                 radVel = [ekf_tracks{nFrame}.prevState{iDet}(3,1);ekf_tracks{nFrame
                        }.prevState{iDet}(4,1);0];
143                 velEgo = radarParams.Orientation*radVel+radarParams.OriginVelocity;
144                 pos_ego(iDet,:) = Xego(1:2);
145                 ekf_tracks{nFrame}.targPos{iDet,:} = pos_ego(iDet,:);
146                 ekf_tracks{nFrame}.targVel{iDet,:} = velEgo(1:2)';
147                 ekf_tracks{nFrame}.stateCovariance{iDet} = prevStateCov(:,:,iDet);
148                 ekf_tracks{nFrame}.labels{iDet} = int2str(iDet);
149                 % Update Display
150                 helperAutoDrivingRadarSigProc('Update Display',egoCar,dets,cell2mat
                        (ekf_tracks{nFrame}.targPos),...
151                 dopgrid,rnggrid,Xbf,beamscan,Xrngdop);
152                 % Publish snapshot after 10 frames
153                 helperAutoDrivingRadarSigProc('Publish Snapshot',time>=1.1);
154             end
155
156         end
```

```
157 %%
158     nFrame = nFrame+1;
159 end
160 %plot NIS metric
161 calculateNSIMetric(nis_error(11:20,1),nFrame-1);
```

A.9 Miscellaneous Functions

Dimension Estimation from Range-Doppler image

```
1  function clusterDim = estimateClusterDimension(clusterIDs,detidx,rnggrid,dopgrid,
       azest)
2  % Inputs:   clusterIDs: 1-by-L vector of cluster IDs assigned to each detection
3  %                       index in detidx.
4  %           detidx:     2-by-L matrix of detection indices generated from the
5  %                       beamformed range-Doppler image, Xbf. Each column of detidx
       identifies the
6  %                       range and Doppler cell in the range-Doppler image where a
       detection was
7  %                       found as [rngidx; dopidx].
8  %           rnggrid:    range information from range doppler map
9  %           dopgrid:    doppler information from range doppler map
10 %           azest:      angle of arrival estimate
11 % Output:   clusterDim: 2-by-1 array of length and width estimate
12 detCluster = [];
13 k = 1;
14 rngBin = [];
15 dopBin = [];
16 clusterIdx = unique(clusterIDs);
17 cart_x = [];
18 cart_y = [];
19 clusterDim = zeros(2,numel(clusterIdx));
20     for i = 1:size(clusterIdx)
21         for j = 1:size(detidx,2)
22             if clusterIDs(1,j) == clusterIdx(i)
23                 detCluster(:,k) = detidx(:,j);
24                 k = k+1;
25             end
26         end
27         for p = 1:size(detCluster,2)
28             rngBin(p) = rnggrid(detCluster(1,p),1);
29             dopBin(p) = dopgrid(detCluster(2,p),1);
30         end
31
32         for p = 1:size(detCluster,2)
33             [x,y,~] = sph2cart(azest(i)*pi/180,0,rngBin(p));
34             cart_x(p) = x;
35             cart_y(p) = y;
36         end
37       k = 1;
38       clusterDim(1,i) = max(cart_x) - min(cart_x);
39       clusterDim(2,i) = max(cart_y) - min(cart_y);
40       rngBin = [];
41       dopBin = [];
42       cart_x = [];
43       cart_y = [];
44       detCluster = [];
45     end
46 end
```

Calculate normalized squared innovation

```
1  function  calculateNISMetric(error,frame_count_new)
```

```
stateVec_len =22; %length of state vector
x=10:frame_count_new;
Ubound = chi2inv(.95, stateVec_len);
Lbound = chi2inv(.05, stateVec_len);
plot(x.*0.1,repmat(Ubound, 1,length(x)), 'r--') %upper bound
axis([1 frame_count_new*0.1 5 Ubound+Lbound])
hold on
plot(x.*0.1,repmat(Lbound, 1,length(x)), 'r--') %lower bound
hold on
plot(x.*0.1,error)
xlabel('Time (sec)');
ylabel('NIS Statistic');
end
```