2021

Deep Learning Assisted Intelligent Visual and Vehicle Tracking Systems

Liang Xu
Virginia Commonwealth University

DEEP LEARNING ASSISTED INTELLIGENT VISUAL AND VEHICLE TRACKING SYSTEMS

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

LIANG XU

M.S., Georgia Institute of Technology

M.S., Wichita State University

B.S, Tianjin University of Technology

Advisor: Ruixin Niu,

Associate Professor, Department of Electrical and Computer Engineering

Virginia Commonwealth University

Richmond, Virginia

August, 2021

Acknowledgements

I would like to express my sincere gratitude to my Ph.D. advisor Prof. Ruixin Niu. It was a pure miracle for me to work with Dr. Niu, after considering giving up my Ph.D. journey. It was Dr. Niu, who recruited me during the challenging time of my research. I would like to thank him again for his continuous support for my Ph.D. study and research. He trained me, inspired my potentials, and made me an eligible Ph.D. Without his guidance and continuous help, this dissertation would not have been possible.

I would also like to thank my committee members, Dr. Alen Docef, Dr. Yanxiao Zhao, Dr. Cang Ye, and Dr. Mulugeta Haile for serving on my advisory committee. Dr. Alen Docef and I both graduated from Georgia Tech. I was Dr. Zhao's teaching assistant for EGRE 364. With her help, I received the Best Teaching Assistant Award of the College of Engineering in that year. I took Dr. Ye's robotic vision class, and I would like to say that was one of the best classes I have ever taken at VCU. Dr. Haile supported our group's research work and provided research advice for the past several years, which are very important for me to complete my Ph.D. program. I would like to thank again all the committee members, for providing constructive comments as early as I proposed my research plan.

I would also like to extend my thanks to other faculty and staff members of the Department of Electrical and Computer Engineering. I want to give special thanks to Dr. Ümit Özgür, who helped me during my challenging time and offered me very important financial support. I would like to also thank my first Ph.D. advisor Dr. Weijun Xiao, who recruited me to start my Ph.D. journey in the beginning. Without him, this journey would not start, and I would not have accomplished these achievements.

Meanwhile, I thank my previous and current lab mates Dongwei Wang, Qinbin Xia, Andrew Ward, Tao Sun, Fereshteh Firouzi, Maitham Alsalman, and other close friends in VCU. They helped me a lot both with my study and life in Richmond. There are also a lot of friends who supported my family locally, and I also enjoyed the time spent with them.

I also want to thank my internship mentors Giancarlo Baldan and Lingji Chen, who both gave me lots of advice during my summer internship in Motional, which is also the company I will join after graduation. I still remember the one-on-one meeting every day. They dedicated one hour with me every workday, helping me go through real-world problems with autonomous vehicles. I came up with a lot of research ideas after the internship. I would like also to thank the company Motional, which hired me after the internship one year before my graduation.

I would like to thank my family members. I thank all of them who always support me during my Ph.D. study. I thank my parents who taught me the value of hard work and education. They let me believe in education which can change a person's life. I thank my parents-in-law who let their lovely daughter married with me, and give us financial supports. Mostly, I want to express my sincere appreciation to my wife Wanning. She dedicates herself to our family, and always supports me with my whimsical ideas, even when I told her I want to quit my job to pursue my Ph.D. degree. She bears my minimum salary and provides everything she can to our family. My son Yuanbao is a cute and lovely boy. I hope he can understand what I write in this dissertation soon, and discuss with me his ideas.

The research work in this dissertation was supported in part by the Air Force Office of Scientific Research (AFOSR) Dynamic Data and Information Processing Portfolio under Grant FA9550-18-1-0362.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

DEEP LEARNING ASSISTED INTELLIGENT VISUAL AND VEHICLE TRACKING SYSTEMS

By Liang Xu

A submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2021.

Advisor: Ruixin Niu,

Associate Professor, Department of Electrical and Computer Engineering

Sensor fusion and tracking is the ability to bring together measurements from multiple sensors of the current and past time to estimate the current state of a system. The resulting state estimate is more accurate compared with the direct sensor measurement because it balances between the state prediction based on the assumed motion model and the noisy sensor measurement. Systems can then use the information provided by the sensor fusion and tracking process to support more-intelligent actions and achieve autonomy in a system like an autonomous vehicle. In the past, widely used sensor data are structured, which can be directly used in the tracking system, e.g., distance, temperature, acceleration, and force. The measurements' uncertainty can be estimated from experiments.

However, currently a large number of unstructured data sources can be generated from sensors such as cameras and LiDAR sensors, which bring new challenges to the fusion and tracking system. The traditional algorithm cannot directly use these

unstructured data, and it needs another method or process to "understand" them first. For example, if a system tries to track a particular person in a video sequence, it needs to understand where the person is in the first place. However, the traditional tracking method cannot finish such a task. The measurement model for unstructured data is usually difficult to construct. Deep learning techniques provide promising solutions to this type of problem. A deep learning method can learn and understand the unstructured data to accomplish tasks such as object detection in images, object localization in LiDAR point clouds, and driver behavior prediction from the current traffic conditions. Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks, and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, and machine translation, where they have produced results comparable with human expert performance. How to incorporate information obtained via deep learning into our tracking system is one of the topics of this dissertation.

Another challenging task is using learning methods to improve a tracking filter's performance. In a tracking system, many manually tuned system parameters affect the tracking performance, e.g., the process noise covariance and measurement noise covariance in a Kalman Filter (KF). These parameters used to be estimated by running the tracking algorithm several times and selecting the one that gives the optimal performance. How to learn the system parameters automatically from data, and how to use machine learning techniques directly to provide useful information to the tracking systems are critical to the proposed tracking system.

The proposed research on the intelligent tracking system has two objectives. The first objective is to make a visual tracking filter smart enough to understand unstructured data sources. The second objective is to apply learning algorithms to improve a tracking filter's performance. The goal is to develop an intelligent tracking system that can understand the unstructured data and use the data to improve itself.

The topics of this dissertation are focused on:

- How to use the unstructured data as inputs to the tracking system.
- How to make the tracking system better by integrating the detection algorithm and prediction method in the loop. We treat the tracking systems not only as individual modules in the whole system but also as interacting with other modules.
- How to improve the tracking system with the data, and let it learn from the real-world scenario.

CHAPTER 1

INTRODUCTION

1.1 Background and Problem Statement

Autonomous systems have become popular due to their potentially huge impact on our society [1, 2]. For example, autonomous vehicles can save a large amount of human labor, improve efficiency in transportation, and lead to higher safety for society. A vital part of autonomous vehicles is their ability to perceive the surrounding environment for safe driving. Its perception system is responsible for detecting many different subjects, such as nearby vehicles, pedestrians walking on the road, the road to drive, traffic lights, and road signs. In an autonomous system illustrated in Fig. 1, the inputs to the perception system are the raw sensor data from cameras, radar, and LiDAR.

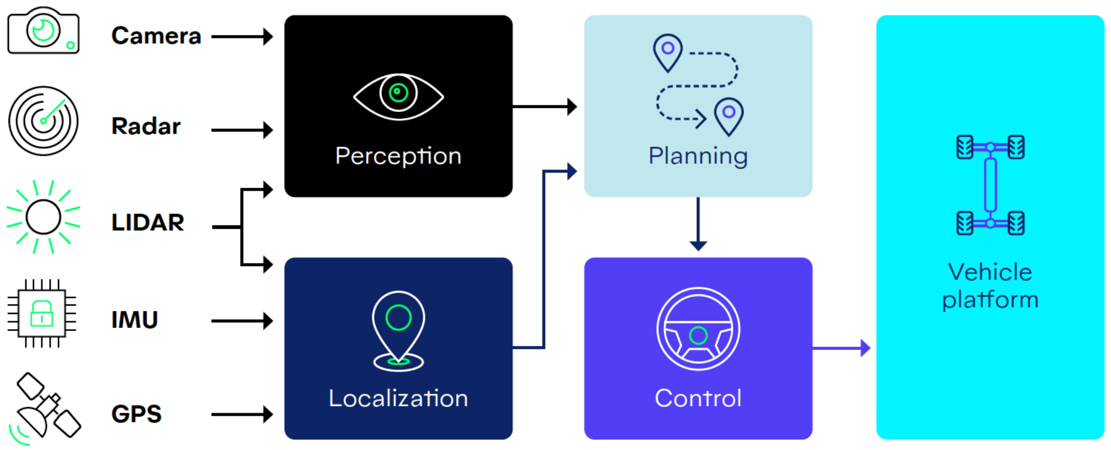


Fig. 1.: High level autonomous system architecture [3].

Inside the perception system, the first task is to understand the raw sensor data.

High-performing computer perception algorithms based on deep learning have been used to understand surrounding objects in real-time, by processing raw sensor data. The second task is to track the objects according to the results from the detection algorithms [2, 4, 5, 6]. The emerging deep learning algorithms can solve problems that cannot be solved previously. The tracking system faces its challenges to use these results accurately and efficiently. Also, how to use the emerging machine learning technology to improve the tracking algorithm itself is still an open problem.

This dissertation has two aspects to improve the tracking performance of an autonomous system with deep learning methods. The first aspect is to incorporate the results from deep learning algorithms into a tracking filter to track objects which cannot be tracked previously. Another aspect is to use learning techniques to improve the tracking algorithm itself.

For the first aspect, I use visual object tracking as a subject to investigate the combination of the tracking algorithm and deep learning detection techniques. In this part, I propose two new visual tracking approaches. One is using semi-supervised learning to improve visual features during visual tracking, and the second is to model visual objects, detected by a deep learning algorithm, as extended targets, and track the objects' kinematic states and shapes simultaneously.

For the second aspect, I propose a method called EKFNet [7], which can transform the extended Kalman filter (EKF) to a recurrent neural network, and learn the unknown system parameters from data. Also, there are two follow-up topics, which are based on the same framework to improve the tracking results. One is using the surrounding environment information to improve vehicle tracking results, which is called Traffic-Aware EKF. Another one is using the LiDAR detection features to estimate the measurement uncertainties for vehicle detections.

In this chapter, some background knowledge about object tracking and deep learning is presented. The EKF formulation is discussed in the beginning, then the following is the introduction to deep learning. In the end, I briefly discuss the proposed learning-assisted tracking methods.

1.2 Nonlinear Filtering and Tracking

Nonlinear filtering is the process of estimating and tracking the state of a nonlinear stochastic system from noisy observation data [8]. As shown in Fig. 2, the red dots represent the measurements, and the goal is to estimate the state of the plane over time by using these measurements.

The filtering process consists of recursively estimating, based on a set of noisy measurements, at least the first two moments of the state vector, governed by a dynamic nonlinear non-Gaussian state-space model. A discrete-time filter consists of a stochastic propagation step (prediction according to the motion model), and an update step using a stochastic observation that links the observation data to the current state vector. In the Bayesian formulation, the filter specifies the conditional posterior probability density function (PDF) of the state at the current time given all the observations up to the current time.

When the dynamic and observation equations are linear and the associated noises are additive and Gaussian, the optimal recursive filtering solution is the Kalman filter (KF) [9]. The most widely used filter for nonlinear systems with Gaussian additive noise is the well-known EKF which requires the computation of Jacobian matrices for state propagation and state update with measurements [8].

1.3 General Bayesian Filter

A nonlinear stochastic system can be defined by a stochastic discrete-time state space transition (motion model) equation:

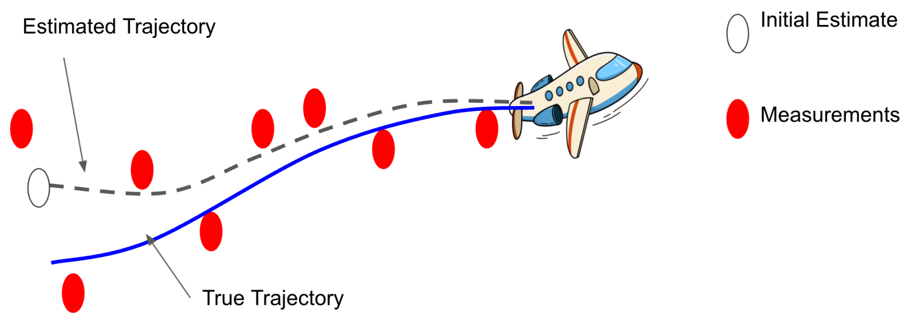


Fig. 2.: Tracking an air plane.

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \mathbf{v}_k) \tag{1.1}$$

and the stochastic observation (measurement) process:

$$\mathbf{z}_k^m = h_k(\mathbf{x}_k, \mathbf{w}_k) \tag{1.2}$$

where at time k, $\mathbf{x}_k$ is the (usually hidden or not observable) system state vector, and $\mathbf{v}_k$ is the process noise vector. $\mathbf{z}_k^m$ is the real observation or measurement, and $\mathbf{w}_k$ is the measurement noise vector. The functions $f_k(\cdot)$ and $h_k(\cdot)$ link the prior state to the current state, and the current state to the observation respectively.

In Bayesian filtering, the problem is to estimate the posterior PDF $p(\mathbf{x}_k|\mathbf{z}_{1:k}^m)$. In the above nonlinear non-Gaussian state-space model, (1.1) specifies the predictive conditional transition PDF, $p(\mathbf{x}_k|\mathbf{x}_{1:k-1}, \mathbf{z}_{1:k-1}^m)$, of the current state given the previous state and all the previous observations. Also, the observation equation (1.2) specifies the likelihood function of the current observation given the current state $p(\mathbf{z}_k^m|\mathbf{x}_k)$. The prior PDF $p(\mathbf{x}_k|\mathbf{z}_{1:k-1}^m)$ is calculated according to Bayes' rule as:

$$p(\mathbf{x}_k|\mathbf{z}^m_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{1:k-1}, \mathbf{z}^m_{1:k-1})p(\mathbf{x}_{k-1}|\mathbf{z}^m_{1:k-1})d\mathbf{x}_{k-1} \tag{1.3}$$

where the previous posterior PDF is identified as $p(\mathbf{x}_{k-1}|\mathbf{z}^m_{1:k-1})$.

The update step generates the posterior PDF from:

$$p(\mathbf{x}_k|\mathbf{z}^m_{1:k}) = p(\mathbf{z}^m_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}^m_{1:k-1})/c \tag{1.4}$$

where c is a normalization constant:

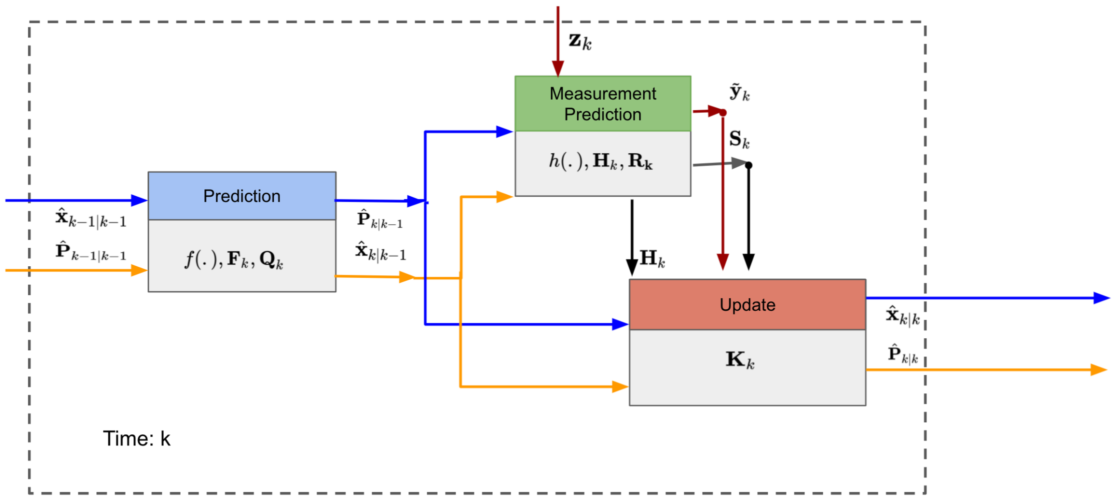
$$c = p(\mathbf{z}^m_k|\mathbf{z}^m_{1:k-1}) = \int p(\mathbf{z}^m_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}^m_{1:k-1})d\mathbf{x}_k \tag{1.5}$$

The filtering problem is to find $p(\mathbf{x}_k|\mathbf{z}^m_{1:k})$, or at least to estimate its first two moments, in a recursive manner.

But for a general Bayesian filtering problem, multivariate integrals in (1.3) and (1.5) cannot be evaluated in closed form. So some form of integration approximation must be made. Various nonlinear Bayesian filters have been proposed, using different numerical approximations for solving the integral in (1.3) and (1.5), or find an approximation of $p(\mathbf{x}_k|\mathbf{z}^m_{1:k})$. The EKF has been used extensively as a nonlinear filter with high computational efficiency and robust performance.

1.4 Extended Kalman Filter

The EKF is an approximation for the general Bayesian filter in Section 1.3. I begin with describing the background of the EKF and its different modules and defining the basic notation and terminology. As shown in Fig. 3, a single EKF operation can be separated to state prediction, measurement prediction, and state update modules. The state prediction module is used to predict the state using the motion model. The measurement prediction module is used for making a prediction for the next measurement and calculating its residual and covariance. The update module is used for updating the predicted state with the latest measurement. The output from the update module is the updated state also called posterior, and it is going to be used as a prior to the following time step, as illustrated in Fig. 4.



$\mathbf{z}_k$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
$\tilde{\mathbf{y}}_k$
$\mathbf{S}_k$
Prediction
$f(.), \mathbf{F}_k, \mathbf{Q}_k$
$\hat{\mathbf{x}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\mathbf{H}_k$
Update
$\mathbf{K}_k$
$\hat{\mathbf{x}}_{k|k}$
$\hat{\mathbf{P}}_{k|k}$
Time: k

Fig. 3.: Extended Kalman Filter.

At time k, the state transition and measurement models are provided as follows:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{v}_k \tag{1.6a}$$

$$\mathbf{z}_k^m = h(\mathbf{x}_k) + \mathbf{w}_k \tag{1.6b}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions, and cannot be used directly to update the covariances. Instead, the Jacobian matrices $\mathbf{F}_k$ and $\mathbf{H}_k$ are used. $\mathbf{v}_k$ and $\mathbf{w}_k$ are the process and measurement noises, which are assumed to be zero mean Gaussian random vectors with covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

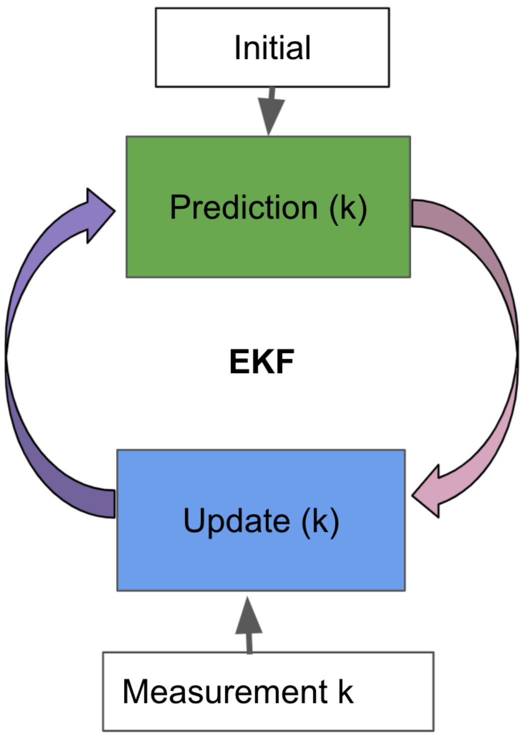


Fig. 4.: Extended Kalman Filter Loop.

The prediction module in Fig. 3 is used for state prediction:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}) \tag{1.7a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{1.7b}$$

The measurement prediction module is used to evaluate the measurement residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{1.8a}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{1.8b}$$

The update module in Fig. 3 provides the posterior state estimate. The Kalman gain

$\mathbf{K}_k$ is calculated in (1.9a):

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{1.9a}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{1.9b}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{1.9c}$$

In the motion and measurement models of a non-linear filter, noise terms are used for compensating the errors made by: (1) model simplification, (2) additional states not modeled, (3) discretization error, (4) model linearization, and (5) Gaussian noise assumptions.

Unlike its linear counterpart, the EKF in general is not an optimal estimator. In addition, if the initial estimate of the state is inaccurate, or if the process is modeled incorrectly, the filter may quickly diverge. Another problem with the EKF is that it tends to underestimate the true covariance matrices and therefore it has the risk of becoming inconsistent in a statistical sense.

1.5 Deep Learning

Deep learning is a subset of machine learning, which is essentially a neural network with many layers, usually more than three. The reason for these layers is to attempt to simulate the behavior of the human brain by activating neurons layer by layer. With some proper method, it can "learn" from large amounts of data, and this process is called training. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to learn the hidden features and relations between the inputs. The features extracted by deep learning have proven to be better than manually selected features [10, 11], and they also can help the network to make better final decisions. Deep learning drives much artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human in the loop [10].

Compared with the traditional machine learning methods, deep learning has its advantages, by the types of data which it works with and the methods which it uses to learn. The extra hidden layers usually perform like a feature extractor, which allows the network to learn from the unstructured data, e.g., images [6, 5], language [12], and point clouds [2, 13]. Through the processes of gradient descent and backpropagation, the deep learning algorithm adjusts its network parameters for better prediction accuracy.

Deep learning models are capable of different types of learning as well. Supervised learning utilizes labeled datasets to categorize or make predictions. This requires some kind of human intervention to label input data. In contrast, unsupervised learning does not require labeled datasets for training. Instead, it detects patterns in the data, clustering them by any distinguishing characteristics automatically. Semi-supervised learning is used for the case with a large amount of unlabeled data and limited labeled data.

1.5.1 Deep Convolutional Neural Networks

Convolutional neural networks (CNNs) have been applied to visual tasks since the late 1980s. However, despite a few applications, they were dormant until the mid-2000s when developments in parallel computing power (GPUs) and large amounts of labeled data brought them to rapid progress. In the past, CNNs are typically applied for image classification tasks.

CNNs are feed-forward networks flow takes place in one direction only, from their inputs to their outputs. Artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain consists of alternating layers of simple and complex cells. Motivated by that, CNN architectures come in several variations. However, in general, they consist of convolutional and pooling layers, which are work together and grouped into modules. Either one or more fully connected layers, as in a standard feed-forward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model, usually more than 3 layers. that is the reason this kind of network is usually called deep CNNs. The convolutional layers are usually used as a feature extractor, and the fully connected layers are used as a classifier. The outputs from the convolutional layers are usually called features, which the fully connected layers use to make the final prediction. Because the convolutional layers are stimulated by 2D shapes, CNN usually works well with visual applications. The features selected by those convolutional layers are usually batter than human selected features [10, 11].

In Fig. 5, a typical CNN architecture is illustrated for an image classification task. An image is an input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations from these operations feed one or more fully connected layers. Finally, the last fully connected layer outputs the class label. Despite this being the most popular fundamental architecture found in the literature, several architecture changes have been proposed in recent years, which improve the accuracy and reduce the computation cost.

1.5.2 Convolutional Layers

In a CNN, the convolutional layers serve as feature extractors, which learn the feature representations of the input images. The neurons in convolutional layers are arranged into feature maps. Each neuron in a feature map has a fixed size receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved

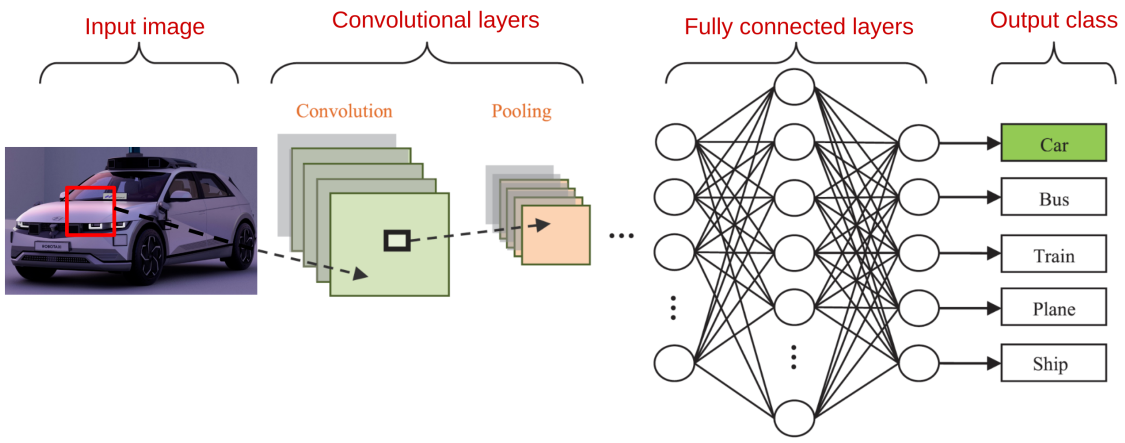


Fig. 5.: Deep Convolutional Neural Networks for Image Classification [14].

with the filter bank with learned weights in order to compute a new feature map, and the convolved results are sent through a nonlinear activation function to form a feature map. All neurons within a feature map have weights that are constrained to be equal. However, different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location.

Because the image usually has a lot of 2D shape features, it is advantageous to use the 2D convolutional operation to encode the features. Several convolutional and pooling layers are usually stacked on top of each other to extract more abstract feature representations. Convolutional layers are usually used in the first several layers to extract the shallow features like circles or lines, and the deep features are extracted by using the filters toward the end of the network.

1.5.3 Fully Connected Layers

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular deep learning networks. Their activations can hence be computed with a matrix multiplication and a bias offset. However, due to the number of parameters it only serves as a classifier at the end of the most of classification networks. The fully connected layers that follow the convolution layers interpret these feature representations and perform the function of high-level reasoning. For classification problems, it is standard to use the softmax or RElU [10, 11] operator to make the prediction.

1.5.4 Back Propagation and Chain Rule

In machine learning, backpropagation is a widely used algorithm for training feedforward neural networks. Generalizations of backpropagation exist for other artificial neural networks. These classes of algorithms are all referred to generically as "backpropagation". In fitting a neural network, backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single input-output example, unlike a naive direct computation of the gradient with respect to each weight individually. This efficiency makes it feasible to use gradient methods for training multilayer networks with large number of weights, updating weights to minimize loss. Gradient descent, or its variants such as stochastic gradient descent, are commonly used. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule[15, 11].

1.5.5 Training

Neural networks in general use optimization algorithms to adjust their trainable parameters in order to obtain the desired output. Backpropagation computes the gradient of an objective function to determine how to change a network's weights in order to minimize errors and improve its performance. Gradient descent is the most popular optimization algorithm that has been used when training a neural network. It minimizes the network's loss function by iteratively moving the training parameter in the direction of steepest descent as defined by the negative of the gradient. A commonly experienced problem with training a neural network is overfitting, which leads to poor performance on a test set after the network is trained on a training set. This affects the model's ability to generalize on unseen data and is a major challenge for neural network training [15].

1.5.6 Recurrent Neural Networks

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a linear directed graph along a temporal sequence. This allows it to explore temporal dynamic behavior between inputs. Derived from feed-forward neural networks, RNNs can use their internal states (memory) to process variable-length sequences of inputs. This makes them applicable to tasks such as handwriting recognition or speech recognition. The basic RNN is shown in Fig. 6, where each step has a input x and output y. The internal states are updated by each step and keep the memory for the next step.

1.6 Proposed Approaches and Structure of the Dissertation

In this section, I briefly discuss the proposed approaches and the structure of this dissertation. In our definition, intelligence tracking has two aspects. The first one is to track the object from unstructured data sources. In this part, visual object tracking is used as an example. The second aspect is how to use data to improve the performance of the tracking algorithm. In this part, I use the EKF as the desired algorithm to be optimized.

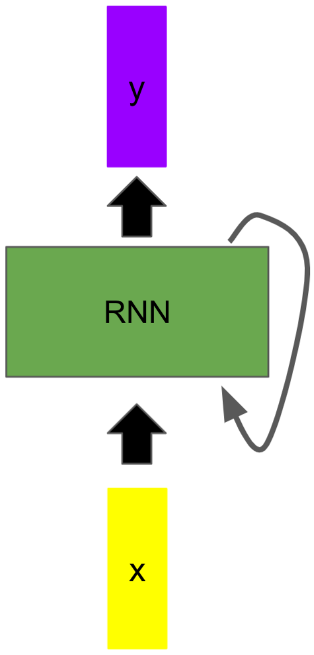


Fig. 6.: Recurrent Neural Network.

1.6.1 Visual Object Tracking

Visual Object Tracking (VOT) is the task of measuring and predicting the trajectory of a specific object at each time step in a video sequence, with the object being given in the first frame. As shown in Fig. 7, the goal is to track the black vehicle, and the vehicle's location and appearance are given in the first frame. The result is marked as a red bounding box that contains the vehicle.

For this topic, I propose two methods to solve this problem. One is based on semi-supervised learning to improve the feature representation, and another one is based on the target modeling technique and extended target tracking methods.

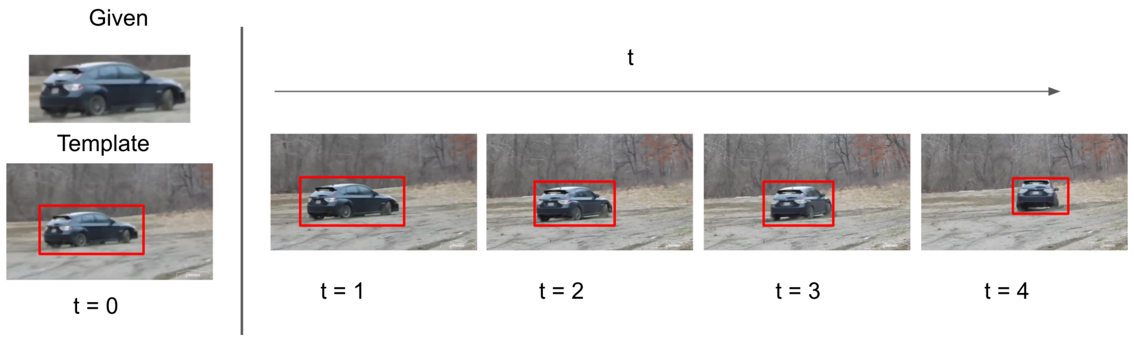


Fig. 7.: visual Object Tracking.

1.6.1.1 Semi-Supervised Visual Tracking Based on Variational Siamese Network

In Chapter 2, I propose a semi-supervised learning framework for visual tracking. The variational auto-encoder has been used as a robust feature encoder to extract visual features for visual object tracking. In this chapter, a new semi-supervised learning framework, variational Siamese neural network, is developed for visual tracking by combining a Siamese network with a variational autoencoder, which supports both supervised and unsupervised training. The learned features are represented as Gaussian distributions in feature space, and the object is represented as a distribution in image space. The similarity between objects' features is measured by an information-theoretic distance.

1.6.1.2 Tracking Visual Object as an Extended Target

This topic is discussed in Chapter 3. Most VOT algorithms treat the 2D visual object as a single point in their output score map, and the bounding box is estimated by a multi-scale search. Furthermore, modern VOT algorithms are based on the concept of tracking-by-detection, which is only focused on the detection step and ignores the object's motion dynamics and the tracking step. A novel modeling technique is proposed to improve object tracking by modeling the visual object as an extended target. The tracking results can help the learning algorithm distinguish the object from the background.

1.6.2 Learning Assisted Tracking Filters

The goal for learning assisted trackers is to use the machine learning technology to improve the tracking system's performance. In this part, I use the deep learning training method to the common EKF and learn the unknown parameters from data.

1.6.2.1 EKFNet: Learning Noise Covariance from Data

The EKFNet is proposed in Chapter 4. In this chapter, I propose a method to reduce the time and manpower to fine-tune an EKF. I propose a new learning framework, EKFNet, for automatically estimating the best process and measurement noise covariance pair for an EKF from the real measurement data. The EKFNet is trained end-to-end by using backpropagation through time (BPTT) with the EKF. The forward operation of EKFNet is the same as the normal EKF operation which will be used during the tracking process. During the offline training, the EKFNet uses the BPTT for passing the gradient flow to each time step and for optimizing the unknown noise statistics parameters.

1.6.2.2 TrafficEKF: A Traffic Aware Kalman Filter

The TrafficEKF is proposed in Chapter 5. Most vehicle tracking algorithms only consider the vehicle's kinematic state but ignore the information about the surrounding environment, which also plays an important role in affecting how the driver controls the vehicle. In addition, how to represent the traffic information and its effect on the vesicle's state is a challenging problem. In this chapter, I propose a tracking method called traffic-aware extended Kalman filter (TrafficEKF), which incorporates not only the vehicle's kinematic dynamics, but also the information from the surrounding environment. The traffic information has been represented by a birds-eye-view rasterized image, with the road shape, traffic light conditions, and other objects inside the field of view. The effect of traffic information on vehicle driving is learned by TrafficEKF from the ground truth data.

1.6.2.3 Uncertainty Aware EKF: Understanding the LiDAR Measurement Uncertainty

This topic is discussed in Chapter 6. The goal of this chapter is to propose an EKF framework called Uncertainty Aware EKF (UA-EKF), which is used for vehicle tracking by understanding the uncommon measurement uncertainties from the LiDAR-based vehicle detections. The UA-EKF has two major parts. One is the ability to estimate the state-dependent measurement noise from LiDAR object detections. Another is to create multiple hypotheses measurements based on the detected vehicle heading. The estimated measurement uncertainties are learned based on the EKFNet, which is proposed in Chapter 4. Both parts are used to compensate for the physical limitations of the LiDAR sensor and object detection algorithm. I also analyze what kind of uncertainty is created by the LiDAR, and how to deal with it.

CHAPTER 2

SEMI-SUPERVISED VISUAL TRACKING BASED ON VARIATIONAL SIAMESE NETWORK

Visual object tracking is the task of estimating the trajectory of a specific object over the time in a video sequence, given the object in the fist frame.The object is given in the first frame by a bounding box, and this is the only information for both detection and tracking. For object tracking in a visual dynamic data-driven application systems (DDDAS) framework, visual appearance features can be extracted by the convolutional neural network, which has been shown to provide a robust feature representation. In this chapter, a new semi-supervised learning framework, variational Siamese neural network, is developed for visual tracking by combining a Siamese network with a variational autoencoder, which supports both supervised and unsupervised training. The learned features are represented as Gaussian distributions in feature space, and the object is represented as a distribution in image space. The similarity between objects' features is measured by an information theoretic distance. The tracking algorithm is based on the detection network's detections to update the object state estimate. Experiment results show that the proposed visual tracking framework outperforms existing state of the art visual tracking approaches.

2.1 Introduction

Visual tracking involves state estimation of objects based on dynamic video data, which serves as a critical component of a visual dynamic data-driven application systems (DDDAS) framework, and is imperative in various visual DDDAS applications, such as camera surveillance and environment monitoring. Further, without assuming a certain model for the video data, deep learning based visual tracking is naturally a data-driven approach. In visual tracking, learning the visual appearance features of an object and searching it in the given image feature space have been investigated recently for detection and tracking of visual objects [16, 17, 18, 19]. The features can be manually selected or extracted by learning algorithms from labeled training dataset. Furthermore, features extracted by the convolutional neural network (CNN) have been shown to represent the object in semantic feature space with robust representation [16].

In this chapter, I develop a new semi-supervised learning framework, the variational Siamese network, by combining the traditional Siamese neural network with a variational autoencoder (VAE). A detector is developed based on this semi-supervised learning framework using both labeled and unlabeled training data. Its first step is to learn the robust feature representation by using an unsupervised deep generative model (VAE). The second step is to use a labeled video sequence to train a visual object detector. The learned features are represented by Gaussian distributions in feature space instead of discrete points as in the traditional methods. The semantic similarity between the object feature and search image feature is measured by using an information theoretic distance metric. Further, I propose to model objects as distributions instead of bounding boxes, which allows measuring the distance continuously. I integrate Kalman filter with our detector, which is used to estimate the object state. To the best of our knowledge, this is the first work where the VAE is used for semi-supervised learning for visual tracking.

To summarize, the main contributions of this are listed below:

- I present a new fully convolutional Siamese network which is based on information theory to optimize the continuous feature space.

- I propose to model the object location and shape as distributions instead of bounding boxes, which allows measuring the distance smoothly.

- I propose a new approach to train the object detector by semi-supervised learning. To the best of our knowledge, this is the first work where variational encoder is used for both supervised and unsupervised learning for visual object tracking.

- I integrate Kalman filter with our detector, which is used to estimate the true object state.

2.2 Related Work

This involves three aspects of visual tracking: Siamese network based visual tracking, variational autoencoder, and visual object representation.

Siamese Network Based Visual Tracking. The advantages of Siamese network based tracking include end-to-end training, no need for online training, and high efficiency, attracting a lot attention recently [20, 16]. SiamFC [20] adopts the Siamese network as a feature extractor and introduces the correlation layer to combine feature maps. However, these previous methods are based on measuring the distance between two vector feature points and supervised learning only.

Variational Autoencoder. VAE [21] has been used as a generative model in machine learning field recently. For example, VAE has been used to generate more training samples for visual tracking [22] and extract robust features for object segmentation [23], respectively. Different from [22, 23], in our work, I use the encoder part of the VAE to extract features for object representation in a visual tracking Siamese network. Rather than representing features with fixed points in a fixed dimensional space, an alternative is to represent them with Gaussian distributions. The distribution's variance can represent the ambiguity, which is a desirable property for modeling the feature representation of an image. Also, the variance can help the learning algorithm to smoothly "fill" the semantic space with continuous representation to generalize better. The distribution features can be learned with deep learning based VAE [21].

Visual Object Representation. One of the basic components in object tracking is to represent the object in space and time. In the case of 2D image, the axis-aligned bounding box representation is widely used to identify an object with its approximate location and size [20]. Because the limitation of subtraction of the object from the background, mask representation becomes popular, such as Siamese-Mask [18]. The mask is a dense representation of the object, which needs much more parameters than other representations. Recently, representation of the object as a Gaussian distribution has been proposed [24], in which the mean represents the object location and the covariance matrix represents the object's shape and orientation in a continuous 2D image space.

2.3 Methodology

Tracking visual objects based on the initial frame can be accomplished by similarity learning between the objects and the current image frame. Here I propose a framework to learn the similarity between an exemplar image z and a candidate image x' with the same size, and return a similarity score. Here x' is a sub-image from a larger search image x. I can detect the object in x by testing all the possible locations, and find the highest similarity score. I denote $f_\theta(z, x)$ as the similarity score between two input images z and x, which can be constructed by the fully convolutional variational Siamese network.

2.3.1 Fully Convolutional Variational Siamese Network

I adopt the fully convolutional Siamese network as the base learning structure, which has been successfully applied in tracking scenarios [20, 18, 25, 26]. Instead of cutting a sub-image from the search space and translating it to the feature space, the fully convolutional network can translate the original search space to a more dense grid in a single evaluation. Also, the fully CNN commutes with translation, and based on its output, one can identify the object location in the original image space. The Siamese network has a powerful framework to compare the difference between two unstructured sources. In this , I develop a new semi-supervised learning framework by combining VAE and Siamese network, with feature outputs as Gaussian distributions.

As shown in Fig. 8, function $q_\theta(\cdot|x)$ is a fully convolutional network given image x, and the similarity between the two images is typically calculated by the Siamese architecture in feature space. q_θ transforms the two images to the feature space, represented as Gaussian distributions, with parameters μ and Σ.

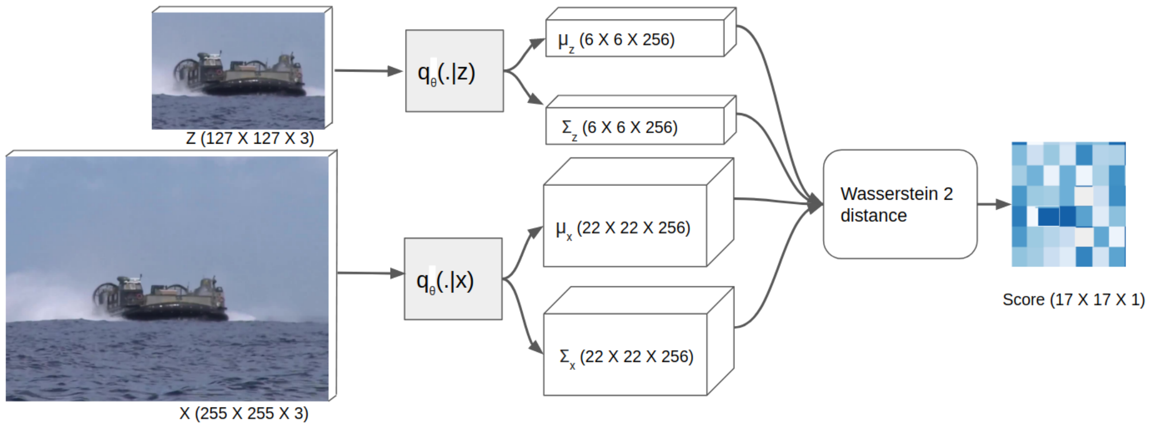


Fig. 8.: Fully convolutional variational Siamese network architecture. z: exemplar image; x: search space. Images are from [27].

2.3.2 Object as Distribution

The object is typically represented by a bounding box in most tracking applications. However, there are several disadvantages with this representation: it is difficult to measure the difference between the proposal bounding box and true label bounding box with different shapes, sizes, and locations; the bounding box provides a binary decision for each pixel with sharp boundaries. In this , I propose to use a normal distribution to parameterize a visual 2D object:

$$\boldsymbol{Z_i = \mathcal{N}(\mu_i, \Sigma_i)} \tag{2.1}$$

where $\mu_i = [x_i \ y_i]^T$, $\Sigma_i = diag(\sigma_{x_i}^2, \ \sigma_{y_i}^2)$, and $x_i, y_i, \sigma_{x_i}, \sigma_{y_i}$ are means and standard derivations. Since most of the labels are axis-aligned, I do not model the correlation between x_i and y_i.

2.3.3 Wasserstein Distance Between Two Gaussian Distributions

I adopt an information theoretic distance measure in this , which is different from the previous work [20]. The object in the feature space is represented by a Gaussian distribution, which allows us to smoothly measure the semantic similarity based on Wasserstein distance [4]. The computation of Wasserstein 2 (W_2^2) is efficient for two Gaussian distributions in $\mathbb{R}^h$.

$$W_2^2(p_1, p_2) = \sum_{i=1}^{h} (\mu_1^i - \mu_2^i)^2 + (\sigma_1^i - \sigma_2^i)^2 \tag{2.2}$$

2.3.4 Variational Autoencoder for Semi-Supervised Training

Our learning framework consists of two steps, generative and discriminative models. The first step is using the unlabeled data for unsupervised learning. The second step is to use the labeled data for learning the similarity between exemplar image and search image, and detect the object.

As shown in Fig. 9, the generative learning can be carried out by a VAE based on variational inference. In this case, I can use the output from the variational fully convolutional network as input to reconstruct the input or the next few frames of the input images. The intermediate variable $s \sim \mathcal{N}(\mu_z, \Sigma_z)$ is the latent variable, which is also the feature extracted by using the network. The unsupervised learning can help the network to find a good feature extractor, and warm up the next task. The deep neural network is usually difficult to train and it is often stuck in a saddle point. By using both unsupervised and supervised learning, I can achieve better training result, as shown later in the .

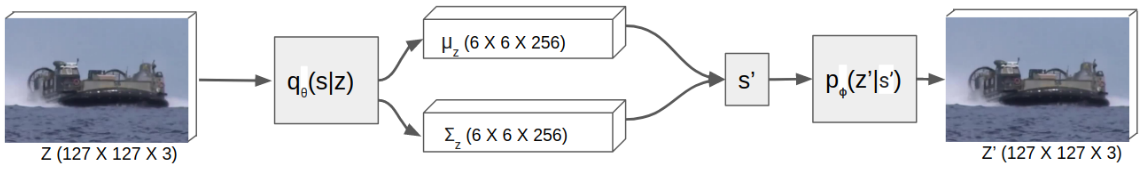


Fig. 9.: Unsupervised learning via VAE. Function q_θ: encoder; function p_φ: decoder. Both of them are fully convolutional neural networks. s' is a sample drawn from the distribution.

The parameters of our encoder-decoder architecture are learned by minimizing the following regularized loss based on variational inference and stochastic gradient descent, which is a sigmoid annealing scheme.

$$-L_{\theta,\phi}(z|z') = -E_{q_\theta(s|z)}(\log p_\phi(z'|s')) + \lambda KL(q_\theta(s|z)||\mathcal{N}(0, I)) \qquad (2.3)$$

The first term encourages the sampled latent space to encode the necessary information to reconstruct the input image, which is called the evidence lower bound (ELBO). The second term, a regularization term, enforces the latent variable to match the standard normal $\mathcal{N}(0, I)$, and to fill the semantic space with a positive definite Σ_z.

The second step of the learning framework is supervised learning based on the labeled tracking datasets. The score map is the final output of the network, and the value should between 0 and 1, the higher the more similar. The distance metric in Section 2.3.3 is from 0 to ∞, the lower the similar. The distance can be transformed by $1 - tanh(x)$ with output from 0 to 1. I call this map O. Ground truth score can be calculated similarly. The value also needs to be rescaled by $1 - tanh(x)$, and this map is T. Our loss can be calculated as follows:

$$Loss = \sum_{w}^{width} \sum_{h}^{hight} \left(O^{h,w} - T^{h,w}\right)^2 \tag{2.4}$$

2.4 Experiments

2.4.1 Implementation Details

Network Architecture: I use AlexNet [28] with slight modifications to output mean and variance. The detailed parameters are listed in Table 1. There are 5 convolutional layers and the last layer has 2 groups of convolutional filters, which outputs the means and variances. A max pooling layer is employed after each of the first two convolutional layers. A ReLU layer follows each convolutional layer except for conv_5_mean.

Training: For unsupervised training I use the dataset Got-10k [27], which has a large number of visual objects. During unsupervised training, only the objects are fed into the architecture presented in Fig. 9. The input image and the output image could be the same or could be T time steps away. So this step can be trained for static images, or for the object in the video.

Tracking: I use a Kalman filter as the tracking algorithm. The object state is defined as $\mathbf{x} = [x,\ y,\ \dot{x},\ \dot{y},\ \sigma_x,\ \sigma_y]^T$, consisting of the position and velocity along

Table 1.: Architecture of variational convolutional embedding function.

	Convolutional Kernel				Features map size	
Layer	Kernel	Chan. map	Stride	Chans.	Exemplar	Search Img
Input				× 3	127 × 127	255 × 255
conv_1	11 × 11	96 × 3	2	× 96	59 × 59	123 × 123
pool_1	3 × 3		2	× 96	29 × 29	61 × 61
conv_2	5 × 5	256 × 48	1	× 256	25 × 25	57 × 57
pool_2	3 × 3		2	× 256	12 × 12	28 × 28
conv_3	3 × 3	384 × 256	1	× 192	10 × 10	26 × 26
conv_4	3 × 3	384 × 192	1	× 192	8 × 8	24 × 24
conv_5_mean	3 × 3	256 × 192	1	× 128	6 × 6	22 × 22
conv_5_var	3 × 3	256 × 192	1	× 128	6 × 6	22 × 22

each direction, and standard deviations of the object distributions in image space.

2.4.2 Evaluation for Visual Object Tracking

Here two tracking challenge datasets are used for evaluations: VOT-2016 [26] and VOT-2018 [25]. I compare the proposed tracking approach against some state-of-the-art approaches, using the official VOT toolkit, and the expected average overlap (EAO), a measure that considers both accuracy and robustness of a tracker. The EAO measures the expected non-reset overlap of a tracker run on a short term sequence. The accuracy is the average overlap during successful tracking periods and the robustness measures how many times the tracker drifts from the target and has to be reset [25]. From Table 2, it is clear that our tracker outperforms the state-of-the-art trackers. Also, its speed is fast and allows real time applications. One tracking example is shown in Fig. 10. As I can see, our proposed approach can track the moving ball and its size and shape accurately.

Table 2.: Comparison with the state-of-the-art tracking approaches. The arrow indicates that the larger/smaller the better.

	Ours	SiamRPN[16]	SCRDCF[29]	STRCF[30]	LSART[31]	ECO[17]
EAO↑	0.339	0.244	0.263	**0.345**	0.323	0.280
Accuracy↑	**0.526**	0.490	0.466	0.523	0.495	0.484
Robustness↓	**0.213**	0.460	0.318	0.215	0.218	0.276
Speed↑	50	**200**	48.9	2.9	1.7	3.7

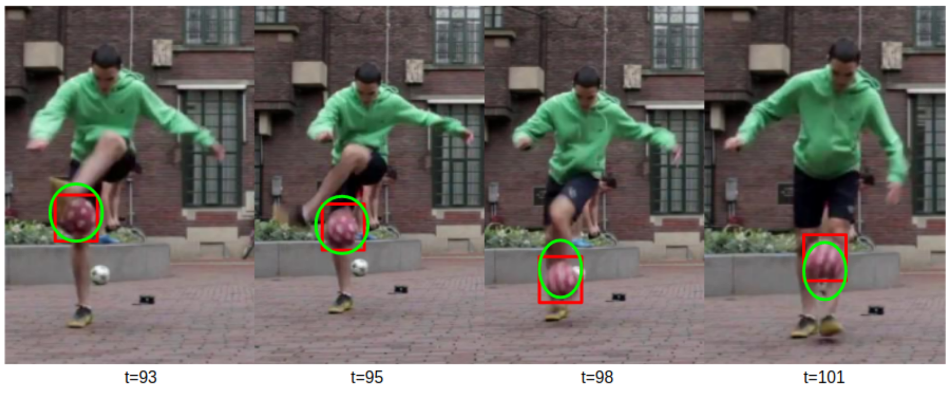


Fig. 10.: Football tracking [25]. Red box: true label; green ellipse: our tracker output with 95% confidence region.

2.4.3 With or Without Unsupervised Learning

The VAE can help us with the training of the detector. In Fig. 11, the supervised learning progress is shown for two different frameworks, with and without the unsupervised learning step respectively. It is clear that on the average, the one with unsupervised learning has a lower loss function, demonstrating the advantage of the proposed semi-supervised learning framework.

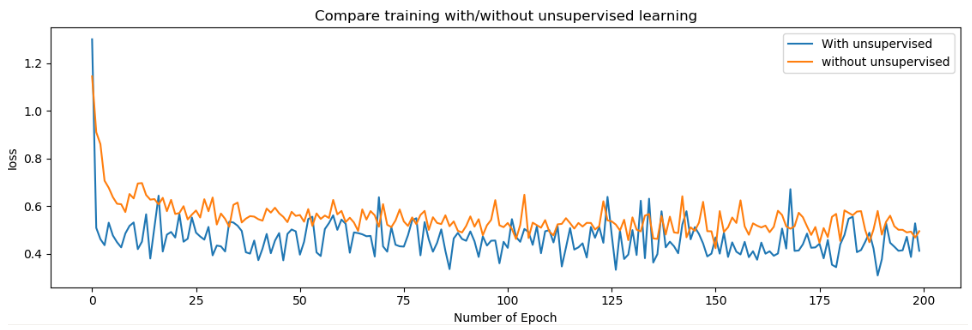


Fig. 11.: The training progress with or without unsupervised learning.

In Table 3, the accuracy and robustness of the tracking results with and without unsupervised learning are compared using dataset VOT2016. It is clear that the approach with unsupervised learning outperforms the one without.

Table 3.: Training results with/without unsupervised learning.

	With Unsupervised	Without Unsupervised
Accuracy↑	**0.520**	0.498
Robustness↓	**0.220**	0.221

2.5 Conclusion

For object tracking in a visual DDDAS framework, I departed from the traditional fully convolutional Siamese network, and developed a variational Siamese network which trains feature embedding through both supervised and unsupervised learning. The embedded features are represented by multivariate Gaussian distributions in a feature space, and the distance between two objects' features is measured by an information theoretic metric (Wasserstein distance). To the best of our knowledge, this is the first work where variational encoder is used for semi-supervised learning for visual tracking. Numerical experiments showed that the proposed visual tracking approach outperforms existing state of the art tracking approaches.

CHAPTER 3

TRACKING VISUAL OBJECT AS AN EXTENDED TARGET

Visual Object Tracking (VOT) is to estimate the trajectory of a specific object in a video sequence, with the tracked object given in the first frame. Most VOT algorithms treat the 2D visual object as a single point in their output score map, and the bounding box is estimated by a multi-scale search. Furthermore, modern VOT algorithms are based on the concept of tracking-by-detection, which is only focused on the detection step and ignores the object's motion dynamics and the tracking step. In this chapter, I address these problems by proposing a novel object modeling concept, and integrating it with a modern target tracking algorithm. Instead of modeling the visual object as a single point, I model it as an extended target with height and width, and the tracking algorithm needs to track both the kinematic parameters and the object's shape. Experiment results are provided on several open available visual tracking benchmarks and the results are compared with state-of-the-art methods.

3.1 Introduction

Learning the visual appearance features for an arbitrary object and searching them in the given image feature space have been investigated recently for tracking and detection of visual objects[32, 16, 19, 33]. Most commonly, the visual object is represented by an axis aligned bounding box which contains the target. However, much improvement have been made recently, it is still a very challenging compare with other vision tasks. On one hand, the target object is only provided in the first frame, which requires the detector to learn a reliable model to distinguish the given object from background by only one valid training sample [17, 20]. On the other hand, the tracking task is under several difficulties such as target appearance and shape change, occlusion, clutter, and illumination variations [27].

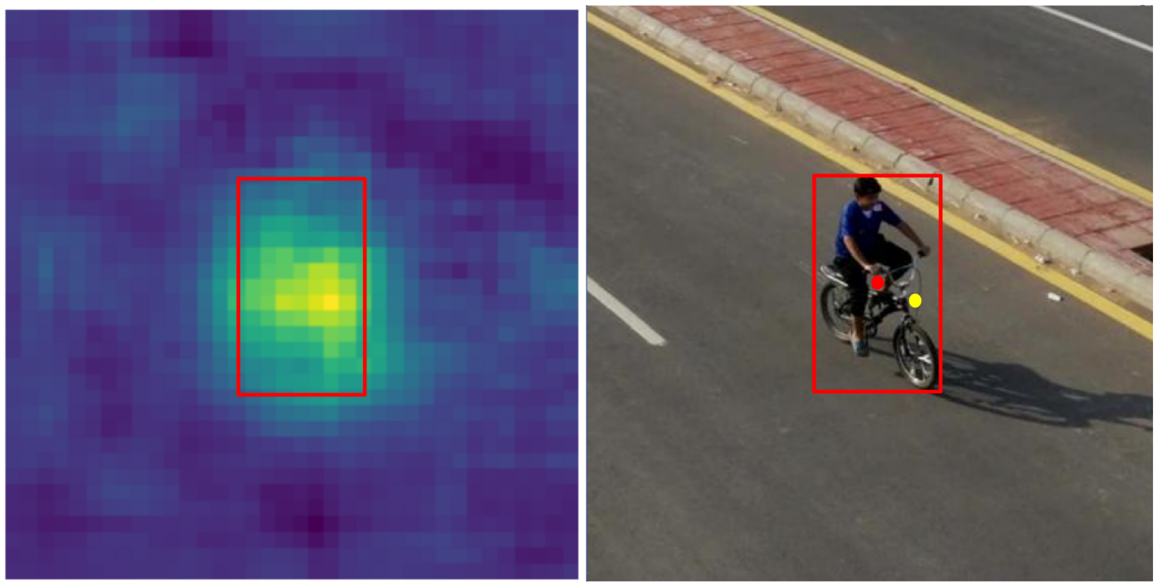


Fig. 12.: Score map and image. Score map is based on the proposed detector. Image is from UVA dataset [34].

Commonly, the visual tracker based on the tracking-by-detection paradigm, which focus on developing a powerful classifier and modeling the searching algorithm as a regression problem [20, 19, 16, 35, 18]. The feature model used in classifier is usually based on a deep convolutional neural network (CNN), which can embed the visual object in a lower dimension space remain the key features to distinguish them from the background. The output from the regression is a score map $s_\theta(z, x) \in \mathbb{R}^2$, which indicates the similarity between the target template z and given search image x using parameter θ. The location y^* which has the highest similarity score is then selected as the estimated target center location for current frame, $y^* = \arg\max_y s_\theta(z, x)$. However, the bounding box parameter vector is in $\mathbb{R}^4$, and the height and width of the bounding box are usually estimated by a multi-scale search [20, 19] or maximize the Intersection over Union (IoU) [36, 37, 32]. Both methods are based on $y*$ and the bounding box size in the previous frame.

First of all, the center of the bounding box is not equal to the highest similarity point in the score map, as shown in Fig. 12. The red bounding box is the ground truth with a red point as center. However, the score map indicate the center as the yellow point. Second, visual object is an extended target, only estimate the center is insufficient to track the target's location and shape. From the Fig. 12, only selecting the highest score point from a cluster of detection wast the information contained in other higher score detection.

In this chapter, I model the visual object as an extended target, and the proposed tracking algorithm simultaneous estimation of the kinematic state and the shape parameters based on a varying number of noisy detections.

The contribution of this chapter are:

- I provide a deep analysis of CNN based tracker and prove that single point detection is not sufficient compare with multi-point detection.

- I model the visual object as an extended target and propose a detection method not only based on the highest score.

- I integrated an extended target tracking algorithm with CNN visual object detection.

- I derive an analytical elliptical gating equation for extended object to discard the clutter detections.

3.2 Modeling Visual Object

There are two groups of approaches for VOT one is based on the Siamese Network [20, 16, 35, 18], another one is based on the Discriminative Correlation Filter (DCF) [38, 36, 37, 32]. For Siamese Network based trackers, the same CNN has been applied to extract features both from the template z and search image y under same weights, and the weights are keep fixed during tracking. However, for DCF based trackers, some of the convolutional kernels are learned online from a series of training samples constructed during tracking. Similarly, large portion of the parameters are trained offline, and keep fixed during tracking. Both of those methods are CNN based and they model the visual object as a single point and perform multi-scale search for the estimation of height and width of the bounding box. In this chapter, I proposed a method which can model the visual object as an extended target. Also, the detector is constructed from a DCF based detector with multiple detection points.

3.2.1 General Formulation

The proposed detection network is based on the popular ATOM tracker [32] showed in Fig. 13. Here, I use the pre-trained ResNet [39] as a backbone network represented as $\phi(.)$, and the convolutional classifier is trained during the tracking. For detailed fast online learning and training samples generation please check the ATOM chapter [32].

The fully convolutional target classifier constructed by 2 fully convolutional layers. The score map for each input image is calculated by convolutional operation $*$ with the equitation below:

$$s_\theta(x) = (w_\theta * \phi(x)) \tag{3.1}$$

where, w_θ is the weight for convolution kernel, and $\phi(x)$ represents the feature ex-

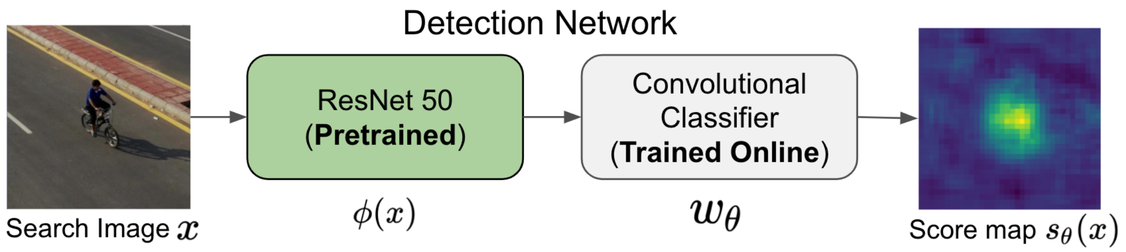


Fig. 13.: An overview of the detection network.

tracted from the search image using the pre-trained backbone network. The w_θ is learned online by a series of tracked history target appearance and pseudo labels. The loss function is the square difference between the output scores s and pseudo labels (α) at each evaluated locations $y \in \mathbb{R}^2$, $L(s, \alpha) = (s - \alpha)^2$. The pseudo labels constructed by using Gaussian function centered at y^*:

$$\alpha(y, y^*) = \exp^{-\frac{|y-y^*|^2}{2\sigma^2}} \tag{3.2}$$

σ is a hyper parameter, which controls the spread of the pseudo labels in the search images. Fig. 14 shows the created pseudo label under different σ, the smaller the σ, the more concentrated the target area will be used for training. The highest pseudo label score is created at y^*, where the score map has the highest score value. Therefore, based on how the pseudo labels are created, the detector is trained to predict the most similar patch in the feature space $\phi(x)$ of the future frames.

3.2.2 Analysis on CNN based Detector

Since both the classifier and the backbone network in (13) are created by fully convolutional neural network, the score map and the input image maintain strict translation invariance, namely $s_\theta(x[\Delta_j]) = s_\theta(x)[\Delta_j]$, where Δ_j is the translation shift operator, which ensures the pseudo label works for training and center detection

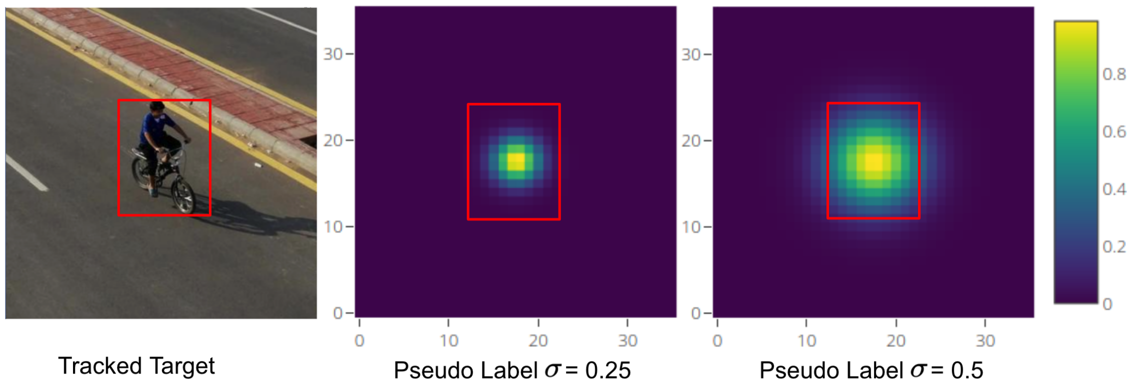


Fig. 14.: Tracked target and its pseudo labels at different σ.

[35]. Also, the target center detection is an ill defined problem, due to the appearance change, osculation and camera orientation [36].

The detector constructed by this concept naturally implies three intrinsic restrictions. 1. The appearance of the target center is changing and its does not represent the center of the bounding box in the future box, as in Fig. (12). 2. A small σ encourages the detector to focus only on the features at the center area of the object, and to ignore the off center contents, as shown in Fig. (14). 3. The padding in deep CNN will destroy the translation invariance that only exists in no padding networks [35], and it will lead to bias on center point detection. However, padding is inevitable in modern network, which is to ensure the network to go deeper and extract rich feature representations.

3.2.3 Detector with Multiple Detection Points

To avoid these limitations in Section 3.2.2, I propose a detector with multiple detections, instead of the traditional methods with single detection. The multiple detections are also based on the score map, which have been selected as exceedances above a certain threshold ζ. In this chapter, the pseudo labels are created based on a larger σ ($\sigma = 0.5$), to cover the whole target area for the training sample. As shown in Fig. 15, from the raw score map, I first apply thresholding to get multiple detections, which are projected back to the original search image. The dimension of score map is usually smaller than the search image, so a certain translation is performed to project the score image back to the search image. Also, translation invariance ensures the proper detection location on the search image. The red dots represent the detections in the search image, and the green ellipse is the extended target.

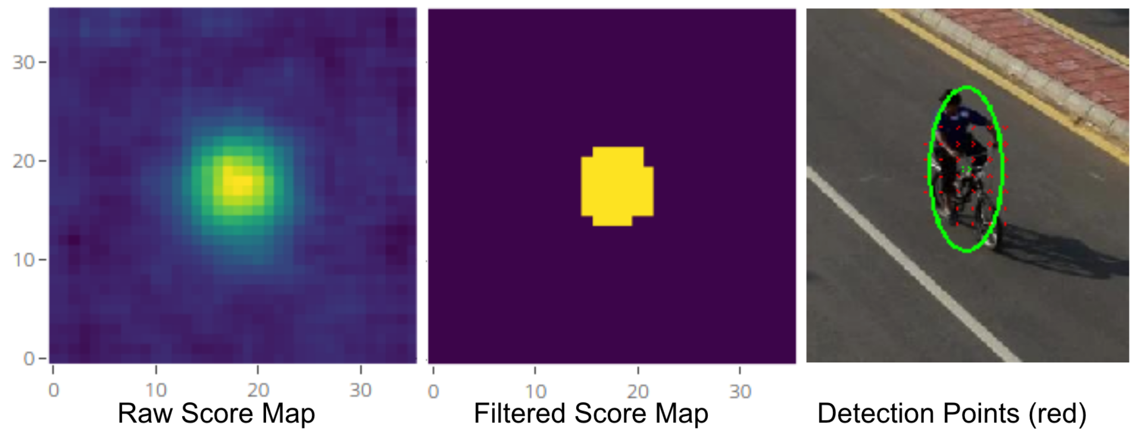


Fig. 15.: Detection with multi-detection points.

I can use the receptive field knowledge to explain the detections. The receptive field in a fully CNN is the region of the input space that corresponds to a particular cell in the final score map. Like in (16), there are three figures with grid, input image, intermediate feature value and the output score map. The blue color region is the target I want to distinguish in the score map. If the score map marked as 1, which means the receptive field in the input image have high probability contain parts of the object.

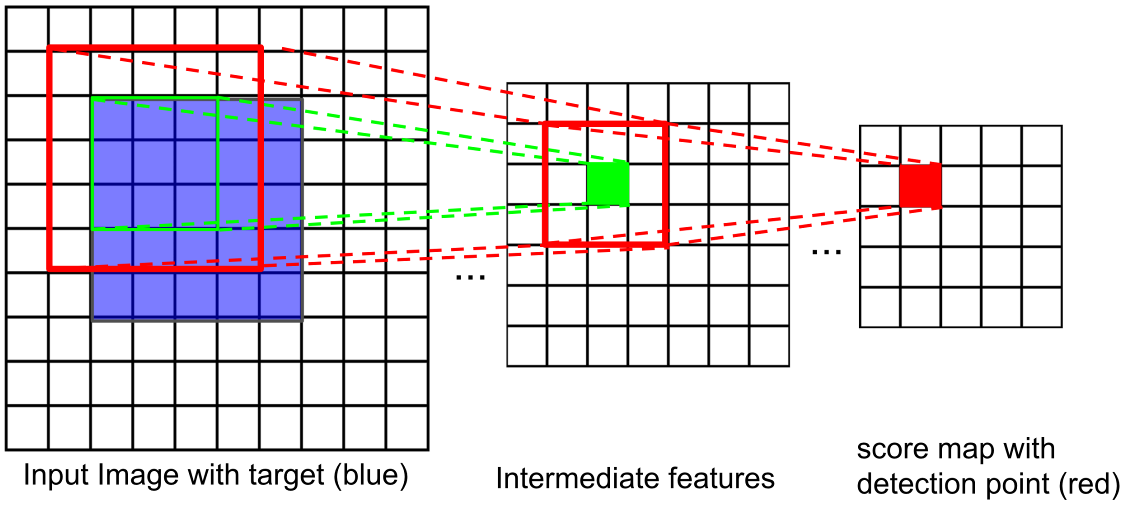


Fig. 16.: The detection point and receptive field with CNN.

3.2.4 Modeling Visual Object as An Extended Target

In this section I model the visual object as an extended target, by using multi-detection points.

3.2.4.1 States Representation for Extended Target

As illustrated in Fig. 17, the visual object has been modeled as an elliptical extended target. The kinematic state r_k and elliptical shape parameters p_k of the object at time k:

$$\mathbf{r}_k = [\mathbf{m}_k^T, \dot{\mathbf{m}}_k^T]^T \in \mathbb{R}^4 \tag{3.3}$$

$$\mathbf{p}_k = [\gamma_k, l_{k,1}, l_{k,2}]^T \in \mathbb{R}^3 \tag{3.4}$$

The kinematic state r_k consists of the center $m_k \in \mathbb{R}^2$ and velocity $\dot{m_k} \in \mathbb{R}^2$. And the shape parameter p_k contains the counterclockwise angle γ_k from the x-axis, and two semi-axes lengths $l_{k,1}$ and $l_{k,2}$. This ellipse modeling of extended target is a common parameterize for modern extended object tracking [40, 41, 42, 43].

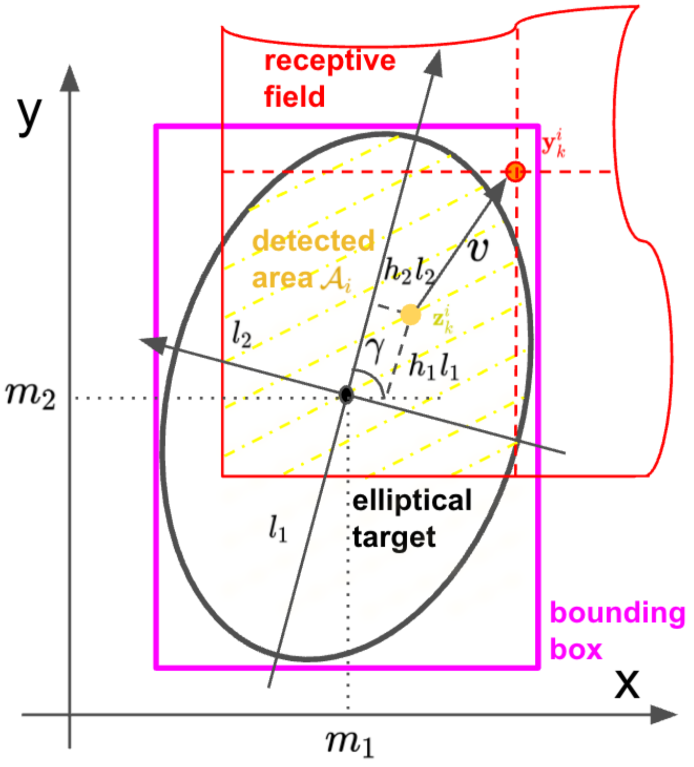


Fig. 17.: Extended target modeling.

3.2.4.2 Measurement Model

Different from the point object, the extended object gives a varying number of independent detection points on the searched image illustrated as red dot in Fig. 15, which can be treated as two-dimensional Cartesian detection. At time k:

$$\mathcal{Y}_k = \{\mathbf{y}_k^i\}_{i=1}^{n_k} \tag{3.5}$$

$$y_k^i = \mathbf{z}_k^i + \mathbf{v}_k^i \tag{3.6}$$

where $\mathbf{y}_k^i$ is the ith detection point on the image which is projected from the filtered score map in Fig. 15, n_k is the total number of detections at time k. Each detection

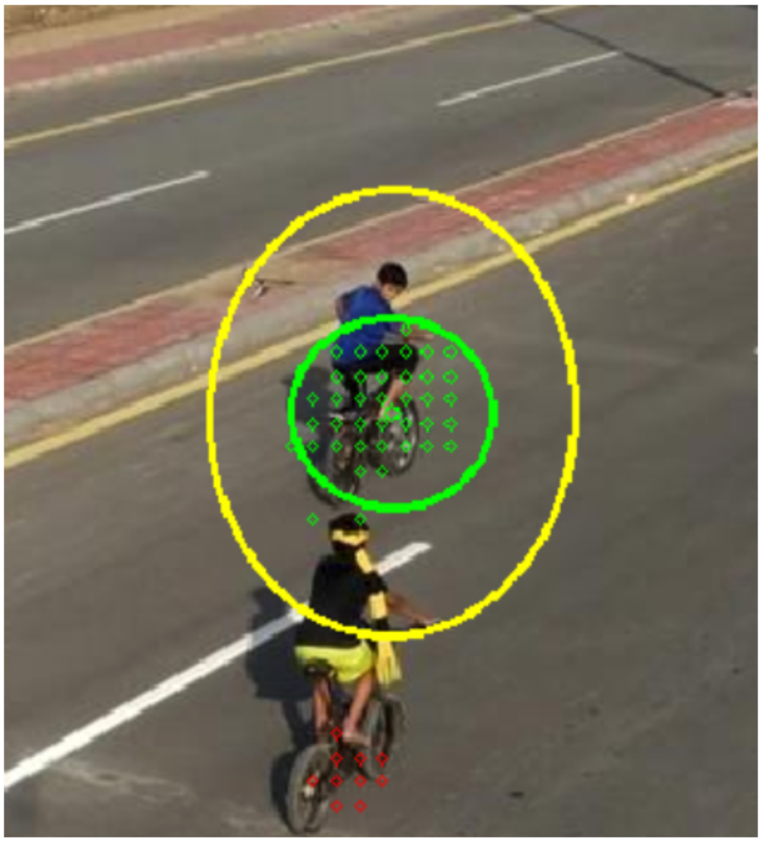


Fig. 18.: Elliptical gating.

$\mathbf{y}_k^i$ is generated from a detected area $\mathcal{A}_i^k$ in color yellow, whose center is $\mathbf{z}_k^i$. The real detection point is stimulated by using the receptive field $\mathcal{F}_i^k$ and generate the detection $\mathbf{y}_k^i$ as descried in section 3.2.3. The size of the receptive filed depends on the CNN structure, but the size of the object and reflected area are changing during tracking. Also, the distance between $\mathbf{z}_k^i$ and $\mathbf{y}_k^i$ is smaller towards the center and getting bigger towards the edge of the object, which is depends on the overlapping area between the object and receptive field. Therefor, I model this measurement corruption by additive Gaussian measurement noise $\mathbf{v}_k^i$ with covariance of $\mathbf{C}^v$.

The center of overlapping area $\mathbf{z}_k^i$ is originated on the object, and follows a Gaussian spatial distribution according to t how the detection network works and pseudo label created in Fig. 14. The larger the overlapped area the more easy for detection network to distinguish this area as an object, and vice versa. So I model this as a multiplicative error term $\mathbf{h}_i$, and assume $\mathbf{h}_k^i \sim \mathcal{N}(\mathbf{0}, \mathbf{C}^h)$. By assuming the extended target as an elliptical with shape parameters $\mathbf{p}_k$ and kinematic states $\mathbf{r}_k$, any point $\mathbf{z}_i$ on the object follows the below equation:

$$\mathbf{z}_k^i = \mathbf{H}\mathbf{r}_k + \underbrace{\begin{bmatrix} \cos\gamma_k & -\sin\gamma_k \\ \sin\gamma_k & \cos\gamma_k \end{bmatrix} \begin{bmatrix} l_{k,1} & 0 \\ 0 & l_{k,2} \end{bmatrix}}_{:=\mathbf{S_k}} \underbrace{\begin{bmatrix} h_{k,1}^i \\ h_{k,2}^i \end{bmatrix}}_{:=\mathbf{h}_k^i} \tag{3.7}$$

$$\mathbf{y}_k^i = \mathbf{H}\mathbf{r}_k + \mathbf{S}_k\mathbf{h}_k^i + \mathbf{v}_k^i \tag{3.8}$$

$\mathbf{H} = [\mathbf{I}_2, \mathbf{0}]$ select the location states, $\mathbf{S_k}$ represents the orientation and size of the object. The axis aligned bounding box estimation can be generated according to this elliptical like in Fig. 17. The dynamic model is an linear model, where the location updated according to speed and shape remains the same as previous.

$$\mathbf{r}_{k+1} = \mathbf{A}_k^r\mathbf{r}_k + \mathbf{w}_k^r \tag{3.9}$$

$$\mathbf{p}_{k+1} = \mathbf{A}_k^p\mathbf{p}_k + \mathbf{p}_k^r \tag{3.10}$$

where, $\mathbf{A}$ is the process matrices, w is the process noises.

3.3 Tracking An Extended Target

This section introduce the closed-form tracking framework based on the MEM-EKF* [41]. In order to deal with the high nonlinearities and multiplicative noise in measurement (3.7) the kinematic states and shape parameters are decoupled and updated separately, for detailed information please check the reference [41].

3.3.1 Tracking Framework

For a given frame, the first task is to predict the object's location and extract a small patch as search space from the big image like illustrated in Fig. 19. The detection network use this small patch to get multiple detections. I use an elliptical gating to filter out clutter and other detection points and treat the point inside the gate as detections ,then update both the kinematic and shape states according to MEM-EKF*.

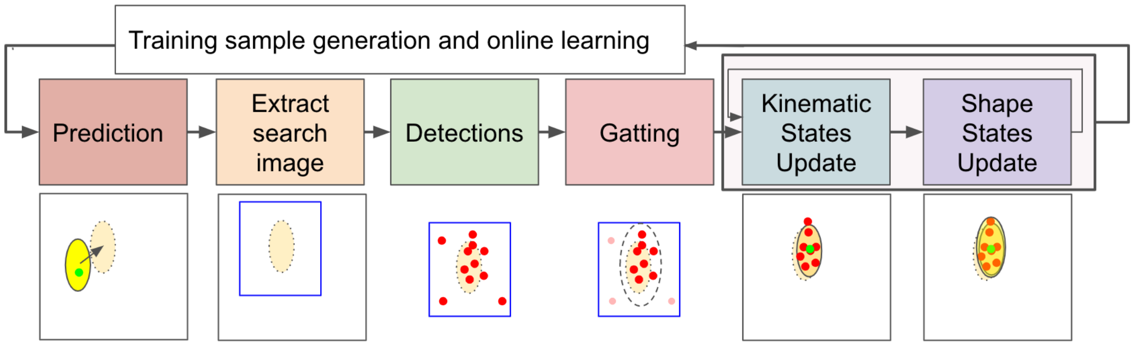


Fig. 19.: Detection and tracking framework.

3.3.2 Elliptical Gating

Gating is a technique to disregard detection as clutters, and in Kalman Filter the ellipsoidal gate is a natural choice also simple to implement. According to (3.8), the measurement is linear to the kinematic state but nonlinear with the shape parameters. In order to approximate the covariance the shape uncertainty has to be approximated by Taylor expansion with respect to $\mathbf{p}$ at predicted shape parameters $\bar{\mathbf{p}}_k$.

$$\mathbf{S}_k \mathbf{h}_k^i \approx \underbrace{\bar{\mathbf{S}}_k \mathbf{h}_k^i}_{\mathrm{I}} + \underbrace{\begin{bmatrix} (\mathbf{h}_k^i)^T \,\widehat{\mathbf{J}}_1(\bar{\mathbf{p}}_k) \\ (\mathbf{h}_k^i)^T \,\widehat{\mathbf{J}}_2(\bar{\mathbf{p}}_k) \end{bmatrix} (\mathbf{p} - \bar{\mathbf{p}}_k)}_{\mathrm{II}} \tag{3.11}$$

where $\widehat{\mathbf{J}}_1$ and $\widehat{\mathbf{J}}_1$ are the Jacobian matrices of the first and second row of $\mathbf{S}$ with respect to $\mathbf{p}$, and evaluated at the $\bar{\mathbf{p}}_k$, and $\bar{\mathbf{S}}_k$ is constructed by $\bar{\mathbf{p}}_k$. The covariance of $\mathbf{S}_k\mathbf{h}_k^i$ can be approximated by sum of covariance of I and II, which defined as $\mathbf{C}^{\mathrm{I}}$ and $\mathbf{C}^{\mathrm{I}}$, where:

$$\mathbf{C}^{\mathrm{I}} = \bar{\mathbf{S}}_{k-1}\mathbf{C}^h\bar{\mathbf{S}}_{k-1}^T \tag{3.12}$$

$$\mathbf{C}^{\mathrm{II}}[m,n] = tr\left\{\overline{\mathbf{C}}_k^p \left(\widehat{\mathbf{J}_n}(\bar{\mathbf{p}}_k)\right)^T \mathbf{C}^h\widehat{\mathbf{J}_m}(\bar{\mathbf{p}}_k)\right\} \tag{3.13}$$

where $m, n \in 1, 2$, evaluated at every position in a $\mathbf{C}^{\mathrm{II}}$ matrix, $\overline{\mathbf{C}}_k^p$ is the prediction covariance for shape parameter at k. The derivation of (3.13) can be found in reference [41] with some modifications. The measurement covariance can be summarized as:

$$\mathbf{C}_k^y = \underbrace{\mathbf{H}\overline{\mathbf{C}}_k^r\mathbf{H}^T}_{A} + \underbrace{\mathbf{C}^{\mathrm{I}} + \mathbf{C}^{\mathrm{II}}}_{B} + \underbrace{\mathbf{C}^v}_{C} \tag{3.14}$$

where $\overline{\mathbf{C}}_k^r$ is the predicted covariance for kinematic states. (3.14) has three terms, A is the uncertainty due to the kinematic model, and B is the uncertainty caused by shape estimation and C is the uncertainty between the real measurement $\mathbf{y}_k^i$ and $\mathbf{z}_k^i$. Then the distance between the measurement $\mathbf{y}_k^i$ and predicted elliptical kinematic states $\bar{\mathbf{r}}_k$ is:

$$(d_k^i)^2 = \left(\mathbf{y}_k^i - \mathbf{H}\bar{\mathbf{r}}_k\right)^T (\mathbf{C}_k^y)^{-1}\left(\mathbf{y}_k^i - \mathbf{H}\bar{\mathbf{r}}_k\right) \tag{3.15}$$

Disregard $\mathbf{y}_k^i$ as a clutter detection if $(d_k^i)^2 > G$, and G is a threshold. The probability that the object measurement is outside the gate is $P_G = Pr[(d_k^i)^2 > G]$, and $(d_k^i)^2 \sim \chi^2(2)$ because the measurement is 2 dimension. G can be defined by a desired value for P_G, like $P_G = 90\% \Rightarrow G = 4.7$, which means 90% confidence the detection point should inside the gate. Like illustrated in Fig. 18, the yellow elliptical is the gate, and the greed dots are the detection pass the gate red ones are discarded as clutter.

3.3.3 Prediction and Updating

The prediction is a standard Kalman filter prediction due to the motion models in (3.9), (3.10) are linear. However, the measurement update are complicated because of the high nonlinearities and multiplicative noise in the measurement model. I use the MEM-EKF* [41] to incorporate with the multiple detection points and update both the kinematic states and shape parameter in the framework. The detections $\{\mathbf{y}_k^i\}_{i=1}^{n_k}$ are incorporated sequentially by first updating the kinematic states then updating the shape parameters by single time scan. For detailed MEM-EKF* updating techniques please check the reference [41].

3.4 Experiment

Table 4.: Comparison with the state-of-the-art tracking approaches. The arrow indicates that the larger/smaller the better.

	Ours(with IOU[32])	Ours	SiamRPN[16]	ATOM[32]	SemiSiam[19]	ECO[17]
EAO↑	**0.413**	0.409	0.244	0.401	0.339	0.280
Accuracy↑	**0.608**	0.601	0.490	0.590	0.526	0.484
Robustness↓	**0.202**	0.202	0.460	0.204	0.213.	0.276

The detection algorithm integrated with *pytracking* framework [44] and based on ATOM tracker [32] on a PC with an Intel Xeon, 32G RAM, Nvidia RTX 2080Ti.

3.4.1 Implementation Details

I used the pre-trained setup and online training sample and learning method as same with ATOM[32], .In order to learn the features for the entire template, I choose a large $\sigma = 0.5$, like in Fig. 14. And the detection threshold ζ has been manually set to 0.55 according to the σ. For the size of elliptical gating I choose the probability that the object measurement is outside the gate is $P_G = 90\%$ then the $G = 4.7$. $\mathbf{C}^h = \frac{1}{4}\mathbf{I}_2$ as proposed in [41], and $\mathbf{C}^v = 2.5\mathbf{I}_2$ to indicate the distribution of $\mathbf{v}$. The initial kinematic states $\mathbf{r}_0$ and shape parameters $\mathbf{p}_0$ are based on the first frame with small covariance matrices, because it is the ground truth.

3.4.2 Compare with State-of-Art

Here is tracking challenge dateset used for evaluations: VOT-2018 [25]. I compared the proposed tracking approach against some state-of-the-art approaches, using the official VOT toolkit [25], and the expected average overlap (EAO), a measure that considers both accuracy and robustness of a tracker. The EAO measures the expected non-reset overlap of a tracker run on a short term sequence. The accuracy is the average overlap during successful tracking periods and the robustness measures how many times the tracker drifts from the target and has to be reset [25]. I also integrated the result with using the IOU-Net refinement [32] which provide a better result also, which shown on the fist column in 4. The proposed method out preformed the base line tracker ATOM by all the metric. From Table 4, it is clear that our tracker outperforms the state-of-the-art trackers. Also, our tracker run around 30 frames per second (FPS) which also be treated as online algorithm. Also, the proposed method can be implemented to almost all the CNN based trackers. which detect the object based on the score map.

3.5 Conclusion

In this work I modeled the visual object as an extended target. Both of the kinematic states and shape parameters are updated based on multiple detection points. I also provide a deep analysis why the single point target assumption is not efficient. To the best of our knowledge, this is the first work where model the visual object as an extended target and provide a closed-loop tracking detection framework. Numerical experiments showed that the proposed visual tracking approach outperforms existing state of the art tracking approaches.

CHAPTER 4

EKFNET: LEARNING SYSTEM NOISE STATISTICS FROM MEASUREMENT DATA

In this chapter, to reduce the time and manpower to fine-tune an extended Kalman filter (EKF), I propose a new learning framework, EKFNet, for automatically estimating the best process and measurement noise covariance pair for an EKF from real measurement data. The EKFNet is trained end-to-end by using backpropagation through time (BPTT) with the EKF. The forward operation of EKFNet is the same as the normal EKF operation which will be used during the tracking process. During the offline training, the EKFNet uses the BPTT for passing the gradient flow to each time step and optimizing the unknown noise statistics parameters. The proposed method can choose among several optimization criteria, such as maximizing the likelihood, minimizing the measurement residual error, or minimizing the posterior state estimation error either with or without the ground truth data. I illustrate the proposed method's performance using real GPS data, which outperforms existing methods and a manually tuned EKF.

4.1 Introduction

Popular nonlinear filters for state estimation include the extended Kalman filter (EKF) [9, 45, 7], the unscented Kalman filter (UKF) [46], and the particle filter (PF) [9]. In practice, the EKF is widely used due to its much lower computational load than the PF and UKF. Different from the linear assumption required by the Kalman Filter, in an EKF, the motion and observation models do not have to be linear functions of the state but need to be differentiable with respect to the state. The motion model is used for predicting the object's dynamic behavior according to the current state. The measurement model maps the state space into the measurement space. In the EKF, these nonlinear models need to be linearized, so that the Kalman filter's recursive framework can be readily utilized.

In the motion and measurement models of a non-linear filter, noise terms are used for compensating the errors made by: (1) model simplification, (2) additional states not modeled, (3) discretization error, (4) model linearization, and (5) Gaussian noise assumptions. In practice, implementing the EKF is often difficult due to the difficulty of getting a good estimate of the motion and measurement noise covariances. Much research has been performed to estimate these covariances from data. However, it takes a significant amount of time and manpower to manually fine-tune the EKF and acquire proper system noise statistics used by the filter. It is thus desirable to develop an approach that automatically finds the best noise statistics for an EKF.

4.1.1 Previous Related Work

The EKF requires knowledge of the noise statistics for uncertainty propagation and the calculation of the filter gain. However, the ground truth noise covariance matrices are generally unknown, and need to be estimated/tuned. In [47], an auto-tuning framework for a Kalman smoother by using the proximal optimization method was introduced. However, the noise covariance pair, which is good for Kalman smoother, performs poorly for Kalman filtering [48]. Different objective functions have been proposed in [48] for tuning the EKF, but the optimization method proposed is neither efficient nor guaranteed to converge. In [49], a nonlinear programming method was proposed for tuning, but it only applies to the Kalman filter (KF). Using Bayesian optimization, a new way to find the globally optimal covariance was presented in [50].

However, it becomes computationally prohibitive for a large training data set. In [51] a six-step solution to filter tuning was provided, but it is only applicable to linear Kalman Filters. How to find the best process and measurement noise covariance matrices for the EKF is still under development.

Some state estimation methods by training the recurrent neural network (RNN) have been developed in several papers [52, 53, 54, 55, 56, 57, 58, 59]. But, the focus of these publications was on learning either the relationship between the states over different time steps [55, 54, 60, 61] or integrating unstructured measurement data to the state space [54]. In [62], a data-driven KF has been proposed, in which the uncertainty propagation steps are replaced by a neural network to calculate the Kalman filter gain. Recently, an emerging trend in data-driven particle filter is developed to learn the motion model and measurement model also through RNNs [63, 64, 65, 60, 66, 67]. Differentiable particle filters (DPFs) were developed in [63, 64], which use backpropagation to learn the motion and measurement models and noise covariance matrices [60]. However, since the particle filter is not differentiable between time steps, the DPFs have to either use finite difference with much higher computational complexity [64] or only train the filter based on a single time step from t to $t+1$, not end to end [63]. In the DPFs publications, both the motion and measurement models are assumed to be unknown, which need to be learned from the data. This is not necessary and will be a waste of resources in the applications where these models are known.

In many system tracking problems, e.g. target tracking in a radar system, both the measurement and motion models are well known, which can be easily constructed by physical modeling and do not have to be learned from data. In such problems, using the learning power to learn the motion and measurement models is wasteful and not guaranteed to converge. Further, the learned motion and measurement models can be easily over-fitted against the training data when one has to learn in a high-dimensional hypothesis space with limited datasets. Therefore, in this chapter , our research is focused on learning the best noise statistics for an EKF with given motion and measurement models.

4.1.2 Contributions

As discussed earlier, in many tracking problems, both the state and measurement models are already given. Even though sometimes in these system models, approximations have been used to convert physical models to mathematical equations, they can still precisely characterize the system. Furthermore, the system noise in the model can be used to account for/absorb the approximation errors. In the EKFNet, instead of learning the system models, the statistic parameters used by the state and measurement models are learned based on the backpropagation through time (BPTT) approach [68]. BPTT is implemented to drive the gradients backward, to optimize the EKF's performance. In summary, the new contributions in this chapter are:

- I construct a new framework for the EKF to learn the noise covariance with or without ground truth data.
- I efficiently drive the gradient backward, and all gradients are calculated by computation graphs.
- I provide four different loss functions for learning the noise covariance, which can be combined or used individually.
- I evaluate the proposed method using real GPS data for vehicle tracking.

The rest of this chapter is organized as follows. In Section 4.2, EKF and the proposed method are described. In Section 4.3, the detailed backpropagation for each model and different loss functions have been discussed. In Section 4.4, the dataset, training parameters, and results are presented. The conclusion and future work are provided in Section 4.5.

4.2 Methodology

In this chapter, I propose a framework for the EKF to learn the system noise covariance matrices, which is trained by the BPTT and gradient descent methods [69, 70, 15]. In this section, the basic EKFNet forward operation is discussed in the beginning. Then the loss functions used for training the EKFNet are presented. The loss functions can be constructed by different optimization criteria, depending on particular applications.

4.2.1 EKFNet

I begin with describing the background of EKF filtering and different modules for the EKFNet, and defining the basic notations and terminology. The forward operation of the EKFNet is the same as the EKF, and separated into three different modules: the state prediction, measurement prediction, and state update modules, as illustrated in Fig. 20. The state prediction module is used for predicting the state using the motion model. The measurement prediction module is applied to make a prediction for the next measurement and calculate its residual and covariance matrix. The update module updates the system state by combining the predicted state with the measurement residual. The output from the update module is the updated state, also called the posterior, which along with its corresponding covariance matrix, will be used as the prior to the next time step.

4.2.1.1 State Transition and Measurement Models

In the EKF the state transition and observation models do not have to be linear functions. At time k, let us denote the system state as $\mathbf{x}_k$, and the measurement as $\mathbf{z}_k$. The motion and measurement models are provided as follows:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{v}_k(\mathbf{x}_{k-1}) \tag{4.1a}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{w}_k \tag{4.1b}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions, and cannot be used directly to update the covariances. Instead, their corresponding Jacobian matrices $\mathbf{F}_k$ and $\mathbf{H}_k$ are used. $\mathbf{v}_k(\mathbf{x}_{k-1})$ and $\mathbf{w}_k$ are the process and measurement noises, which are assumed to be independent zero-mean multivariate Gaussian random variables with covariance matrices $\mathbf{Q}_k(\mathbf{x}_{k-1})$ and $\mathbf{R}_k$, respectively. The goal is to learn $\mathbf{Q}_k(\mathbf{x}_{k-1})$ and $\mathbf{R}_k$ with real data. In general it is assumed that the process noise is state and time dependent, and measurement noise is time dependent.

4.2.1.2 Prediction Module

The prediction module in Fig. 20 is used for state prediction. Notation $\hat{\mathbf{x}}_{t|m}$ represents the estimate of $\mathbf{x}$ at time t given the measurements up to time m. The state prediction equations are:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}) \tag{4.2a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k(\hat{\mathbf{x}}_{k-1|k-1}) \tag{4.2b}$$

$\mathbf{Q}_k(\cdot)$ is time and state dependent process noise covariance matrix. For the prediction module, the input is the posterior from the previous state, and the output is the predicted state and its covariance according to the state transition model in (4.1a).

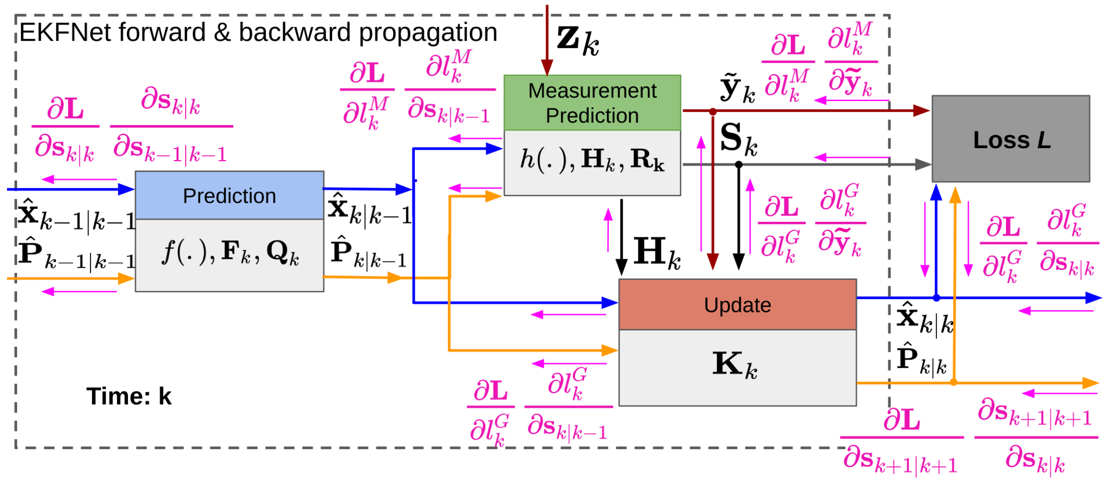

EKFNet forward & backward propagation
$\mathbf{z}_k$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
Prediction
$f(.), \mathbf{F}_k, \mathbf{Q}_k$
Update
$\mathbf{K}_k$
Loss L
$\hat{\mathbf{x}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k-1|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\hat{\mathbf{P}}_{k|k-1}$
$\tilde{\mathbf{y}}_k$
$\mathbf{S}_k$
$\mathbf{H}_k$
$\hat{\mathbf{x}}_{k|k}$
$\hat{\mathbf{P}}_{k|k}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{s}_{k|k}}$ $\frac{\partial \mathbf{s}_{k|k}}{\partial \mathbf{s}_{k-1|k-1}}$
$\frac{\partial \mathbf{L}}{\partial l_k^M}$ $\frac{\partial l_k^M}{\partial \mathbf{s}_{k|k-1}}$
$\frac{\partial \mathbf{L}}{\partial l_k^M}$ $\frac{\partial l_k^M}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial \mathbf{L}}{\partial l_k^G}$ $\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial \mathbf{L}}{\partial l_k^G}$ $\frac{\partial l_k^G}{\partial \mathbf{s}_{k|k}}$
$\frac{\partial \mathbf{L}}{\partial l_k^G}$ $\frac{\partial l_k^G}{\partial \mathbf{s}_{k|k-1}}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{s}_{k+1|k+1}}$ $\frac{\partial \mathbf{s}_{k+1|k+1}}{\partial \mathbf{s}_{k|k}}$
Time: k

Fig. 20.: EKFNet diagram. $\mathbf{s} = \{\hat{\mathbf{x}}, \mathbf{P}\}$.

4.2.1.3 Measurement Prediction Module

For the measurement prediction module in Fig. 20, the task is to evaluate the measurement residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{4.3a}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{4.3b}$$

where $\tilde{\mathbf{y}}_k$ is the residual and $\mathbf{S}_k$ is the residual covariance, which can be used in Section 4.2.2 to calculate the measurement loss function. $\mathbf{R}_k$ is the measurement noise covariance matrix.

4.2.1.4 Update Module

Based on the previous two steps, the update module in Fig. 20 updates the posterior state estimate. First, the Kalman gain $\mathbf{K}_k$ is calculated as follows.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{4.4}$$

Once the Kalman gain is obtained, the state estimate is updated with the measurement residual and its corresponding covariance matrix as follows.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{4.5a}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{4.5b}$$

Finally, the updated state estimate and its covariance will be passed to the next step as the prior. The same operations will be repeated recursively over time until the last measurement is incorporated by the EKF to update the state estimate.

4.2.2 Loss Functions

In order to achieve a better EKF performance, I present four optimization criteria, which can be used to optimize the noise covariances either with or without ground truth system states.

4.2.2.1 Measurement Residual

A loss function can be constructed using the l_2 norm of the measurement residual. The measurement residual $\tilde{\mathbf{y}}_k$ is the output of the measurement prediction module, defined in Section 4.2.1.3, and the loss function based on measurement residuals can be calculated from time 1 to T as follows.

$$L^M_{Res} = \sum_{k=1}^{T} \|\tilde{\mathbf{y}}_k\|_2^2 \tag{4.6}$$

4.2.2.2 Mean Squared Error (MSE) of State Estimate

The MSE of the updated state is calculated between the ground truth $\mathbf{x}_k^G$ and a projection from $\hat{\mathbf{x}}_{k|k}$, and the corresponding loss function is:

$$L_{MSE}^{G} = \sum_{k=1}^{T} \|\mathbf{x}_k^G - g_k(\hat{\mathbf{x}}_{k|k})\|_2^2 \tag{4.7}$$

where $g_k(\cdot)$ is a projection function that transforms the full state to the sub-space of the ground truth. Note that high-end sensor data or human-annotated references are often used as ground truth. Hence, the ground truth is expensive to obtain in some applications.

4.2.2.3 Measurement Residual Log-Likelihood

The measurement residual is only concerned with itself but ignores its covariance. At each time, the measurement residual covariance $\mathbf{S}_k$ is also calculated by the EKF. So hereby maximizing the likelihood, I consider not only the residual but also its uncertainty. By taking the logarithm of the likelihood, I can deal with addictions instead of multiplications. The negative log-likelihood of the measurements is provided in (4.8).

$$\begin{aligned} L_L^M &= -\log\left(\prod_{k=1}^{T} p(\mathbf{z}_k|\mathbf{z}_{0:k-1})\right) \\ &= -\sum_{k=1}^{T} \log p(\mathbf{z}_k|\mathbf{z}_{0:k-1}) \\ &= -\sum_{k=1}^{T} \left(\log\left(|\mathbf{S}_k|\right) + \tilde{\mathbf{y}}_k\mathbf{S}_k^{-1}\tilde{\mathbf{y}}_k + const\right) \end{aligned} \tag{4.8}$$

where the last step is due to the Gaussian approximation used for the measurement residual in the EKF. Compared with (4.6), in (4.8) the measurement residual covariance $\mathbf{S}_k$ is incorporated in the loss function.

4.2.2.4 Posterior Log-Likelihood

The MSE between the ground truth and state also only concerns with the error itself, but ignores its covariance. By considering the posterior's uncertainty which is the posterior covariance $\mathbf{P}_{k|k}$, the negative log-likelihood of the ground truth is provided in (4.9).

$$\begin{aligned} L_L^G &= -\log\left(\prod_{k=1}^{T} p(\mathbf{x}_k^G|\mathbf{z}_{0:k})\right) \\ &= -\sum_{k=1}^{T} \log p(\mathbf{x}_k^G|\mathbf{z}_{0:k}) \\ &= -\sum_{k=1}^{T} \left(\log\left(\left|\mathbf{P}_{k|k}\right|\right) + \tilde{\mathbf{x}}_k \mathbf{P}_{k|k}^{-1} \tilde{\mathbf{x}}_k + const\right) \end{aligned} \tag{4.9}$$

where

$$\tilde{\mathbf{x}}_k = \mathbf{x}_k^G - g_k(\hat{\mathbf{x}}_{k|k}) \tag{4.10}$$

Again, the last step of (4.9) is derived because of the Gaussian assumption used for $\tilde{\mathbf{x}}_k$ in the EKF.

The constant terms in (4.8) and (4.9) can be ignored during the backpropagation because these terms have zero gradients with respect to the optimization variables (the noise covariance matrices). The physical meaning of (4.8) and (4.9) is that the EKF should be tuned to maximize the likelihood corresponding to the measurements or to the updated state estimates. Because in (4.6) and (4.8), only the measurement data are needed, the optimization criterion based on the measurement residual or the measurement likelihood, can be applied without the ground truth data.

4.2.2.5 Loss Functions with Regularization

In many applications, it is often desirable not only to minimize the loss functions defined earlier in this section but also to encourage an optimization solution with other properties, such as its closeness to a manually optimized solution. Hence, I propose two regularized loss functions, based on either the measurement residuals/ MSE errors or the negative log-likelihoods:

$$L_{MSE} = aL^{G}_{MSE} + bL^{M}_{Res} + \lambda_1 r_1(\mathbf{Q}) + \lambda_2 r_2(\mathbf{R}) \tag{4.11}$$

$$L_{Like} = aL^{G}_{L} + bL^{M}_{L} + \lambda_1 r_1(\mathbf{Q}) + \lambda_2 r_2(\mathbf{R}) \tag{4.12}$$

Each loss function consists of two parts: the first two terms are loss functions and the last two terms are regularization functions ($r(\cdot)$ functions). The loss functions encourage the filter output to minimize the residual/MSE error or maximize the likelihood. The regularization functions are problem-dependent but have to be differentiable. For example, it can encourage the covariance matrices to be close to the manually optimized results [47]. a, b, λ_1, and λ_2 are the weights for loss functions or regularization functions and they are hyper parameters. In applications without ground truth, the first term can be ignored, i.e. $a = 0$.

4.3 Back Propagation Through Time

BPTT is a gradient-based method for training a directed computation graph based on time sequence, which is widely used to train the RNN [68, 15, 70]. The EKF can be deemed as a directed graph, which is built by internal states and measurement data of variable lengths. The training data for the EKF are a sequence of T input-output pairs $\{\mathbf{z}_k, (\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, \tilde{\mathbf{y}}_k, \mathbf{S}_k)\}_{k=1}^{T}$. For the purpose of optimizing the EKF performance, BPTT begins with unfolding the EKF over time, and each node uses the same variables to construct $\mathbf{Q}$ and $\mathbf{R}$, as discussed later in Section 4.4.2. Then the backpropagation algorithm is used to find the gradients of the loss function with respect to (w.r.t.) the parameters in $\mathbf{Q}$ and $\mathbf{R}$. Finally, the gradient descent optimization algorithm can be used to update the parameters based on the gradients

[69, 70, 15].

The backpropagation flow is shown in Fig. 20. The pink equations and left arrows indicate the backward gradient flow. At each step k, the gradients are derived from: (1) the loss function l_k^M for each time k, which is based on the measurement residual and its covariance, and can be constructed by either measurement residual L_{Res}^M or residual log-likelihood L_L^M at time step k. (2) loss function l_k^G for time step k, which is based on the posterior error w.r.t. the ground truth and its covariance, it could be either the MSE of the estimated state L_{MSE}^G or the negative posterior log-likelihood L_L^M at time k, and (3) gradients from time step $k+1$.

In this section, a detailed backpropagation procedure for each module and each loss function will be discussed. The gradients, which can be used to optimize the process and measurement noise covariance matrices, are constructed at the end of backpropagation. Because backpropagation is based on the chain rule for the gradient [15], here I only list the gradient results w.r.t. the intermediate parameters.

4.3.1 Back Propagation in Update Module

The forward operation for the update module has been discussed in Section 4.2.1.4. The gradients input to the update module are those of the loss function w.r.t. to the posterior states $\hat{\mathbf{x}}_{k|k}$ and its covariance $\mathbf{P}_{k|k}$. As shown in Fig. 21, the gradient can be constructed by the following equations:

$$\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}} = \frac{\partial L}{\partial \hat{\mathbf{x}}_{k+1|k+1}} \frac{\partial \hat{\mathbf{x}}_{k+1|k+1}}{\partial \hat{\mathbf{x}}_{k|k}} + \frac{\partial L}{\partial l_k^G} \frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k}} \tag{4.13a}$$

$$\frac{\partial L}{\partial \mathbf{P}_{k|k}} = \frac{\partial L}{\partial \mathbf{P}_{k+1|k+1}} \frac{\partial \mathbf{P}_{k+1|k+1}}{\partial \mathbf{P}_{k|k}} + \frac{\partial L}{\partial l_k^G} \frac{\partial l_k^G}{\partial \mathbf{P}_{k|k}} \tag{4.13b}$$

In (4.13a) and (4.13b), the first term on the right hand side is the gradient flow arriving from time step $k+1$, and the second term is the gradient w.r.t. the state

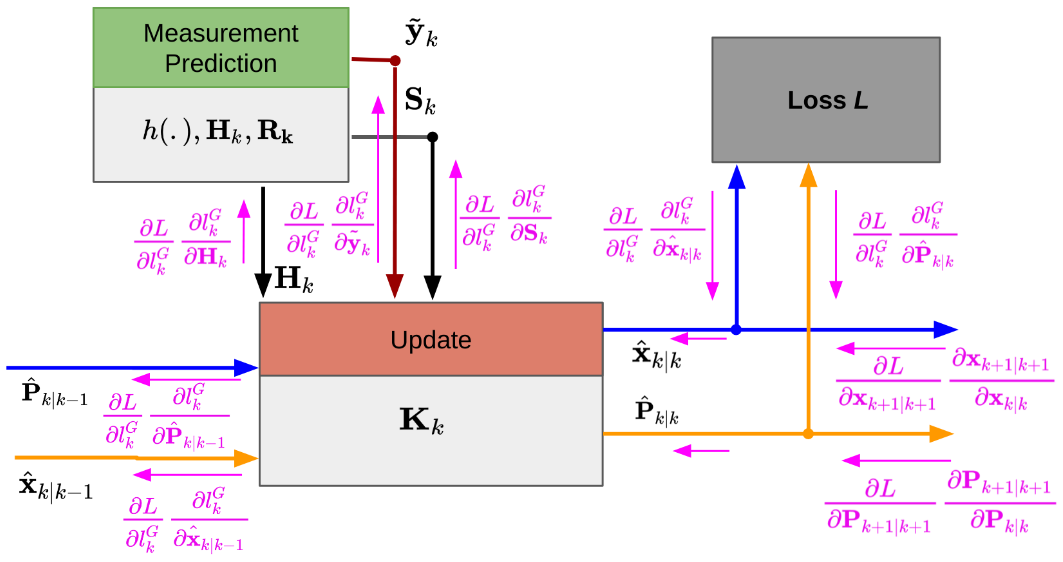

Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
$\tilde{\mathbf{y}}_k$
$\mathbf{S}_k$
Loss L
$\mathbf{H}_k$
Update
$\mathbf{K}_k$
$\hat{\mathbf{P}}_{k|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\hat{\mathbf{x}}_{k|k}$
$\hat{\mathbf{P}}_{k|k}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{H}_k}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k}}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{P}}_{k|k}}$
$\frac{\partial L}{\partial \mathbf{x}_{k+1|k+1}}\frac{\partial \mathbf{x}_{k+1|k+1}}{\partial \mathbf{x}_{k|k}}$
$\frac{\partial L}{\partial \mathbf{P}_{k+1|k+1}}\frac{\partial \mathbf{P}_{k+1|k+1}}{\partial \mathbf{P}_{k|k}}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{P}}_{k|k-1}}$
$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k-1}}$

Fig. 21.: Backpropogation for the Update Module.

estimation error for the current step k denoted as l_k^G. The state estimation error is the difference between the state estimate and the true state. The value for the second term depends on the loss function. If the loss function is constructed without using the ground truth state value, this gradient is 0. For simplicity, U_B stands for the backward operation for the update module. As shown in Fig. 21, the gradient flow for the update module can be summarized below:

$$\left(\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k-1}}, \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{P}_{k|k-1}}, \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}, \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k}, \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{H}_k}\right) = U_B\left(\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}}, \frac{\partial L}{\partial \mathbf{P}_{k|k}}\right) \tag{4.14}$$

There are five terms output from the update module. The first two terms are the gradients w.r.t. the predicted state $\hat{\mathbf{x}}_{k|k-1}$ and its covariance $\mathbf{P}_{k|k-1}$, which will flow back to the prediction module. The rest three terms are the gradients of the loss function w.r.t the residual $\tilde{\mathbf{y}}_k$, its covariance $\mathbf{S}_k$, and the measurement matrix $\mathbf{H}_k$, which will flow back to the measurement prediction module.

Based on the chain rule, the gradient can be calculated analytically. The outputs to the prediction module are listed below:

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k-1}} = \frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}} \tag{4.15}$$

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{P}_{k|k-1}} = (\mathbf{I}-\mathbf{K}_k\mathbf{H}_k)^T\frac{\partial L}{\partial \mathbf{P}_{k|k}} + \frac{\partial L}{\partial \mathbf{K}_k}\left(\mathbf{S^{-1}}\right)^T\mathbf{H}_k \tag{4.16}$$

where:

$$\frac{\partial L}{\partial \mathbf{K}_k} = \frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}}\tilde{\mathbf{y}}_k^T - \frac{\partial L}{\partial \mathbf{P}_{k|k}}\mathbf{P}_{k|k-1}\mathbf{H}_k^T \tag{4.17}$$

$$\tag{4.18}$$

$\frac{\partial L}{\partial l_k^G}$ is the gradient for loss function L w.r.t the state error l_k^G at time step k. The gradient for the predicted covariance $\mathbf{P}_{k|k-1}$ is calculated based on the two equations (4.4) and (4.5b).

The gradients flowing back to the measurement prediction module are:

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k} = \mathbf{K}_k^T\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}} \tag{4.19}$$

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k} = -vec^{-1}\left[\left(\mathbf{S}_k^{-T}\otimes\mathbf{S}_k^{-T}\right)vec\left(\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k^{-1}}\right)\right] \tag{4.20}$$

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{H}_k} = \mathbf{K}_k^T\frac{\partial L}{\partial \mathbf{P}_{k|k}}\mathbf{P}_{k|k-1}^T + \left(\mathbf{P}_{k|k-1}\frac{\partial L}{\partial \mathbf{K}_k}\left(\mathbf{S}_k^{-1}\right)^T\right)^T \tag{4.21}$$

where

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k^{-1}} = \mathbf{P}_{k|k}\mathbf{H}_k^T\frac{\partial L}{\partial \mathbf{K}_k} \tag{4.22}$$

In (4.20), $\otimes$ represents the Kroneker product [71], and $vec(\cdot)$ denotes the column-wise vecterization of a matrix. If $\mathbf{S}_k$ is an $m\times m$ matrix, then in (4.20), $\mathbf{S}_k^{-T}\otimes\mathbf{S}_k^{-T}$ is an $m^2\times m^2$ matrix, and after vecterization, $vec\left(\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k^{-1}}\right)$ is an $m^2\times 1$ vector.

The product of the above mentioned terms, $\left(\mathbf{S}_k^{-T} \otimes \mathbf{S}_k^{-T}\right) vec\left(\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k^{-1}}\right)$ is an $m^2 \times 1$ vector, which needs to be reshaped back into an $m \times m$ matrix by using $vec^{-1}(\cdot)$, the inverse operation of $vec(\cdot)$. In deriving (4.20), the differential of matrix inverse [71] has been used:

$$d(\mathbf{S}_k^{-1}) = -(\mathbf{S}_k^{-T} \otimes \mathbf{S}_K^{-T})d(\mathbf{S}_k) \tag{4.23}$$

The gradient for $\mathbf{S}_k$ is from the backpropagation for (4.4).

4.3.2 Back Propagation in Measurement Prediction Module

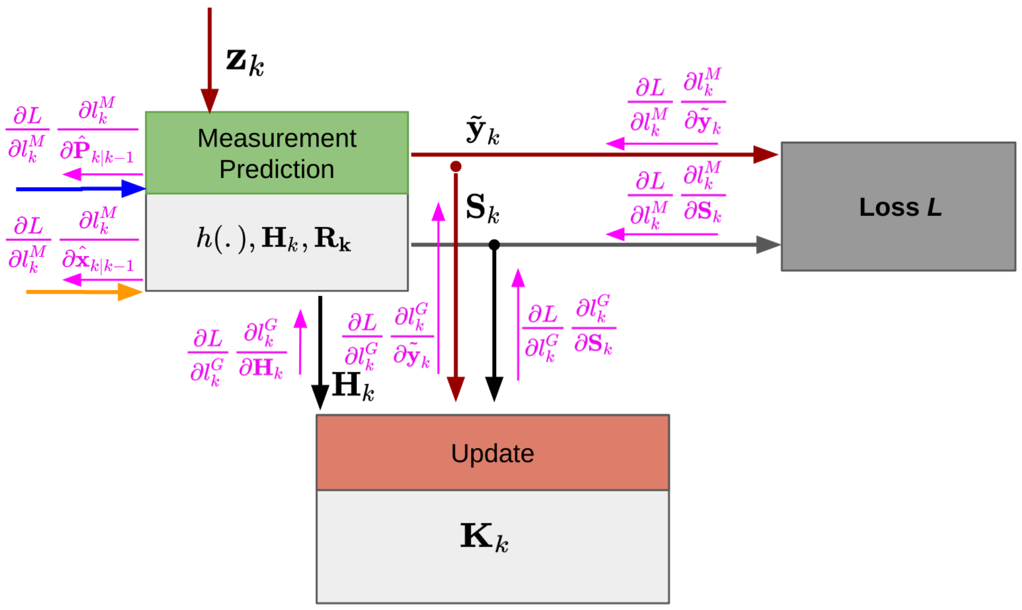

$\mathbf{z}_k$
$\frac{\partial L}{\partial l_k^M}\ \frac{\partial l_k^M}{\partial \hat{\mathbf{P}}_{k|k-1}}$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
$\frac{\partial L}{\partial l_k^M}\ \frac{\partial l_k^M}{\partial \hat{\mathbf{x}}_{k|k-1}}$
$\tilde{\mathbf{y}}_k$
$\frac{\partial L}{\partial l_k^M}\ \frac{\partial l_k^M}{\partial \tilde{\mathbf{y}}_k}$
$\mathbf{S}_k$
$\frac{\partial L}{\partial l_k^M}\ \frac{\partial l_k^M}{\partial \mathbf{S}_k}$
Loss L
$\frac{\partial L}{\partial l_k^G}\ \frac{\partial l_k^G}{\partial \mathbf{H}_k}$
$\mathbf{H}_k$
$\frac{\partial L}{\partial l_k^G}\ \frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial L}{\partial l_k^G}\ \frac{\partial l_k^G}{\partial \mathbf{S}_k}$
Update
$\mathbf{K}_k$

Fig. 22.: Back propagation in Measurement Prediction Module.

The forward operation for the measurement prediction module has been discussed in Section 4.2.1.3. The gradients input to the measurement prediction module are the gradients of loss function w.r.t. to the residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$, and the gradient outputs from the update module which have been discussed in Section

4.3.1. As shown in Fig. 22, the gradient can be constructed by three terms for both the residual $\tilde{y}$, $\mathbf{S}_k$ and $\mathbf{H}_k$ which is given in (4.21). The inputs can be constructed according to the equation below.

$$\frac{\partial L}{\partial \tilde{\mathbf{y}}} = \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k} + \frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \tilde{\mathbf{y}}_k} \tag{4.24a}$$

$$\frac{\partial L}{\partial \mathbf{S}_k} = \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{S}_k} + \frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \mathbf{S}_k} \tag{4.24b}$$

where the first terms on right hand side of (4.24a) and (4.24b) are the outputs from the update module. The second terms on the right hand side of (4.24a) and (4.24b) are the derivative w.r.t. the residual error at time step k. l_k^M represents the loss function due to the measurement prediction for the current step. Those values depend on what loss functions have been chosen also.

For the measurement prediction module, I can simply summarize the inputs and outputs as:

$$\left(\frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \mathbf{P}_{k|k-1}}, \frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \hat{\mathbf{x}}_{k|k-1}}\right) = M_B\left(\frac{\partial L}{\partial \tilde{\mathbf{y}}}, \frac{\partial L}{\partial \mathbf{S}_k}, \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{H}_k}\right) \tag{4.25}$$

where M_B is the backward operation for the measurement prediction module.

The gradients propagate back to the measurement prediction module:

$$\frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \hat{\mathbf{x}}_{k|k-1}} = -\left(\mathbf{H}_k^T\frac{\partial L}{\partial \tilde{\mathbf{y}}}\right)^T + \frac{\partial \mathbf{B}_k}{\partial \hat{\mathbf{x}}_{k|k-1}} \tag{4.26}$$

$$\frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \mathbf{P}_{k|k-1}} = \mathbf{H}_k^T\frac{\partial L}{\partial \mathbf{S}_k}\mathbf{H}_k \tag{4.27}$$

where

$$\mathbf{B}_k = \left((\mathbf{P}_{k|k-1})^T\mathbf{H}_k^T\frac{\partial L}{\partial \mathbf{S}_k}\right)^T + \frac{\partial L}{\partial \mathbf{S}_k}\mathbf{H}_k\mathbf{P}_{k|k-1}^T + \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{H}_k} \tag{4.28}$$

$\mathbf{B}_k$ is the gradient $\frac{\partial L}{\partial \mathbf{H}_k}$. However, $\mathbf{H}_k$ is also a function of the predicted state $\hat{\mathbf{x}}_{k|k-1}$.

So the gradient of $\mathbf{B}_k$ w.r.t. each predicted state $\hat{\mathbf{x}}_{k|k-1}$ also needs to be calculated, which is the second term on the right hand side of (4.26).

Because the measurement noise covariance is involved in the measurement prediction module, the gradients w.r.t. the measurement noise covariance $\mathbf{R}_k$ can be calculated during backpropagation:

$$\frac{\partial L}{\partial \mathbf{R}_k} = \frac{\partial L}{\partial \mathbf{S}_k} \tag{4.29}$$

4.3.3 Back Propagation in the Prediction Module

The forward operation in the prediction module has been discussed in Section 4.2.1.2. As shown in Fig. 23, the gradients propagating back to the prediction module are the gradients of the loss function w.r.t. to the predicted state $\hat{\mathbf{x}}_{k|k-1}$ and its covariance $\mathbf{P}_{k|k-1}$, which are constructed by combining outputs from the measurement prediction module and update module, according to the equation below.

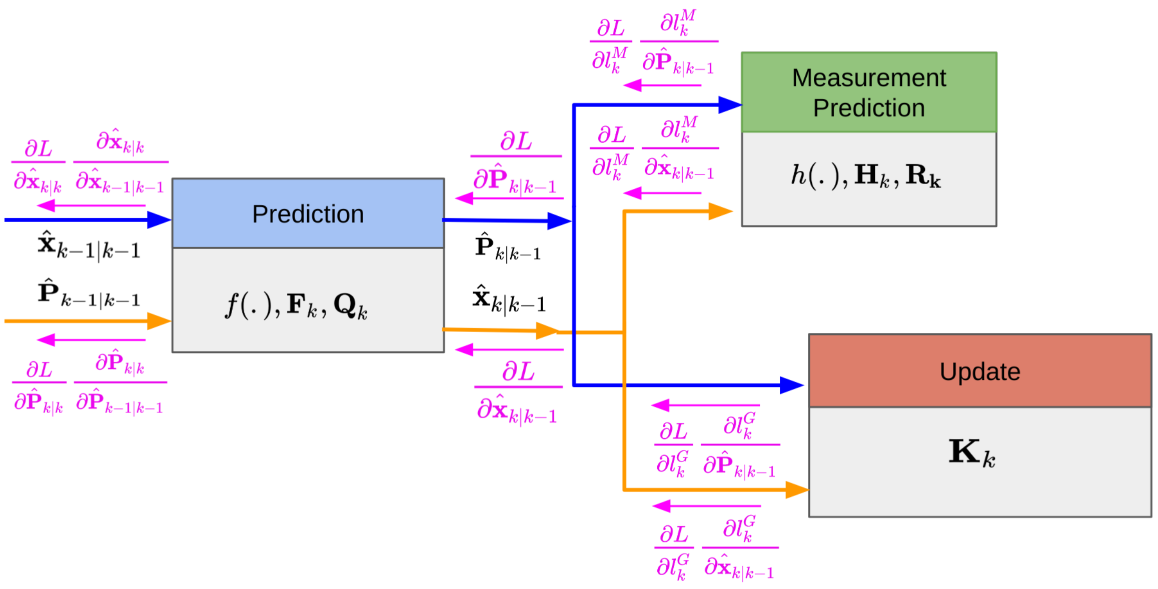
$$\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k-1}} = \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k-1}} + \frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \hat{\mathbf{x}}_{k|k-1}} \tag{4.30a}$$

$$\frac{\partial L}{\partial \mathbf{P}_{k|k-1}} = \frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{P}_{k|k-1}} + \frac{\partial L}{\partial l_k^M}\frac{\partial l_k^M}{\partial \mathbf{P}_{k|k-1}} \tag{4.30b}$$

By combining the outputs from (4.14) and (4.25), the backward propagation for the prediction module can be summarized as:

$$\left(\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}}\frac{\partial \hat{\mathbf{x}}_{k|k}}{\partial \hat{\mathbf{x}}_{k-1|k-1}}, \frac{\partial L}{\partial \mathbf{P}_{k|k}}\frac{\partial \mathbf{P}_{k|k}}{\partial \mathbf{P}_{k-1|k-1}}\right) = P_B\left(\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k-1}}, \frac{\partial L}{\partial \mathbf{P}_{k|k-1}}\right) \tag{4.31}$$

where P_B is the backward operation for the prediction module. The output gradient will be passed to the update module at the time step $k-1$.



Prediction
$f(.), \mathbf{F}_k, \mathbf{Q}_k$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
Update
$\mathbf{K}_k$
$\hat{\mathbf{x}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}} \frac{\partial \hat{\mathbf{x}}_{k|k}}{\partial \hat{\mathbf{x}}_{k-1|k-1}}$
$\frac{\partial L}{\partial \hat{\mathbf{P}}_{k|k}} \frac{\partial \hat{\mathbf{P}}_{k|k}}{\partial \hat{\mathbf{P}}_{k-1|k-1}}$
$\frac{\partial L}{\partial \hat{\mathbf{P}}_{k|k-1}}$
$\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k-1}}$
$\frac{\partial L}{\partial l_k^M} \frac{\partial l_k^M}{\partial \hat{\mathbf{P}}_{k|k-1}}$
$\frac{\partial L}{\partial l_k^M} \frac{\partial l_k^M}{\partial \hat{\mathbf{x}}_{k|k-1}}$
$\frac{\partial L}{\partial l_k^G} \frac{\partial l_k^G}{\partial \hat{\mathbf{P}}_{k|k-1}}$
$\frac{\partial L}{\partial l_k^G} \frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k-1}}$

Fig. 23.: Back Propagation in the Prediction Module.

$$\frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k}} \frac{\partial \hat{\mathbf{x}}_{k|k}}{\partial \hat{\mathbf{x}}_{k-1|k-1}} = \mathbf{F}_k^T \frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k-1}} + \frac{\partial \mathbf{C}_k}{\partial \hat{\mathbf{x}}_{k-1|k-1}} \tag{4.32}$$

$$\frac{\partial L}{\partial \mathbf{P}_{k|k}} \frac{\partial \mathbf{P}_{k|k}}{\partial \mathbf{P}_{k-1|k-1}} = \mathbf{F}_k^T \frac{\partial L}{\partial \mathbf{P}_{k|k-1}} \mathbf{F}_k \tag{4.33}$$

where

$$\mathbf{C}_k = \frac{\partial L}{\partial \mathbf{P}_{k|k-1}} \mathbf{H}_k \mathbf{Q}_k^T + \left(\mathbf{Q}^T \mathbf{H}^T \frac{\partial L}{\partial \mathbf{P}_{k|k-1}} \right)^T \tag{4.34}$$

$\mathbf{C}_k$ is the gradient of the loss function w.r.t $\mathbf{F}_k$. Because $\mathbf{F}_k$ is constructed by $\hat{\mathbf{x}}_{k|k}$, the gradient also needs to be calculated during backpropagation, which is the second term on the right-hand side of (4.32). The first term in (4.32) is derived based on the forward operation (4.2a), where the gradient of the dynamic model is the Jacobian matrix $\mathbf{F}_k$.

Because the process noise covariance $\mathbf{Q}_k$ is involved in the prediction module.

The derivative w.r.t. the process noise covariance matrix for the current step is:

$$\frac{\partial L}{\partial \mathbf{Q}_k} = \frac{\partial L}{\partial \mathbf{P}_{k|k-1}} \tag{4.35}$$

4.3.4 Back Propagation of Loss Function

The gradients for the loss function are different when different optimization criteria are chosen. Among the four different loss functions, some of them only have gradients w.r.t the state, and some of them have gradients w.r.t both the state and its covariance.

4.3.4.1 Back Propagation of Residual Error

The measurement residual is calculated without using the ground truth data. The forward operation of this part is shown in (4.6). If the loss function only considers the measurement residual, the gradient flowing back to each time steps is listed below:

$$\frac{\partial L}{\partial l_k^M} \frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k} = 2\tilde{y}_k \tag{4.36}$$

$$\tag{4.37}$$

The gradient of the loss function w.r.t $\mathbf{S}_k$ is zero in this case.

4.3.4.2 Back Propagation of Negative Log Residual Likelihood

In this section, the back propagation operation for the negative log residual likelihood function is derived, by considering not only the measurement residual $\tilde{\mathbf{y}}_k$, but also its covariance $\mathbf{S}_k$. The loss function used here is defined in (4.8). The gradients calculated from backward propagation flow back to the measurement prediction module. The gradients of the loss function w.r.t. $\mathbf{S}_k$ and $\tilde{\mathbf{y}}_k$ are shown below:

$$\frac{\partial L}{\partial l_k^M}\frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k} = -\mathbf{S}_k^{-1}\tilde{y}_k \tag{4.38}$$

$$\frac{\partial L}{\partial l_k^M}\frac{\partial l_k^G}{\partial \mathbf{S}_k} = \frac{1}{2}\left[\mathbf{S}_k^{-1} - \mathbf{S}_k^{-1}\tilde{y}_k\tilde{y}_k^T\mathbf{S}_k^{-1}\right] \tag{4.39}$$

4.3.4.3 Back propagation of MSE of State Estimate

To find the state estimation error, the ground truth data are required for the system state. The gradient from this section flow back to the update module and the loss function w.r.t. $\mathbf{P}_{k|k}$ is 0 in this case. The gradient flowing back to each time step can be calculated as follows:

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k}} = 2\tilde{x}_k \tag{4.40}$$

where $\tilde{\mathbf{x}}_k$ is defined in (4.10), which is the error between the state and ground truth.

4.3.4.4 Back propagation of Posterior State Log-Likelihood

To evaluate the posterior state likelihood, one needs the state ground truth data. The back propagation chain rule for the posterior state log-likelihood has been derived and given below:

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \hat{\mathbf{x}}_{k|k}} = -\mathbf{S}_k^{-1}\tilde{x}_k \tag{4.41}$$

$$\frac{\partial L}{\partial l_k^G}\frac{\partial l_k^G}{\partial \mathbf{P}_{k|k}} = \frac{1}{2}\left[\mathbf{P}_{k|k}^{-1} - \mathbf{P}_{k|k}^{-1}\tilde{x}_k\tilde{x}_k^T\mathbf{P}_{k|k}^{-1}\right] \tag{4.42}$$

where $\tilde{\mathbf{x}}_k$ has been defined in (4.10).

4.3.5 Derivative of Loss w.r.t. the Noise Covariance Matrices

For most of the cases, the training dataset has multiple tracks, with possibly different track duration T^is. The derivatives w.r.t. $\mathbf{Q}$ and $\mathbf{R}$ are calculated at each step as in (4.43a) and (4.43b) respectively for each track i and they are accumulated over time steps and different tracks by using (4.35) and (4.29):

4.29:

$$\frac{\partial L}{\partial \mathbf{Q}} = \sum_{i=1}^{N} \sum_{k=1}^{T^i} \frac{\partial L^i}{\partial \mathbf{Q}_k^i} \tag{4.43a}$$

$$\frac{\partial L}{\partial \mathbf{R}} = \sum_{i=1}^{N} \sum_{k=1}^{T^i} \frac{\partial L^i}{\partial \mathbf{R}_k^i} \tag{4.43b}$$

where i represents the track identity, and N is the total number of the tracks. T^i is the total number of steps for the current track i.

Many optimization methods have been constructed based on gradients. The variables can be iteratively updated towards the optimal direction by gradient descent [15, 70].

4.4 Experiments

Here I provide an example where a vehicle is tracked in a 2D space, using data from KITTI dataset [72]. The GPS measurements include longitude, latitude, and orientation. The ground truth states are generated by using extended Kalman smoother [9] at the original sampling rate of 100 Hz, and the measurements are generated by down-sample the original data to 2 Hz and adding small Gaussian noise for testing the algorithm.

This section begins with the experiment setup for both the motion and measurement models. Then some comparison between different optimization criteria has been discussed. In addition, I will test the proposed algorithm with imperfect motion and measurement models.

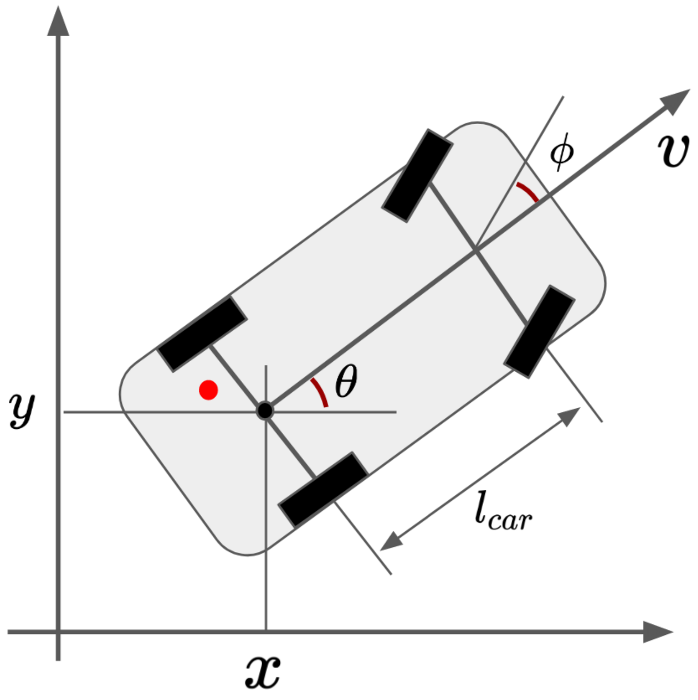


Fig. 24.: Vehicle Motion Model, l_{car} is 2.71m.

4.4.1 Motion and Measurement Models

4.4.1.1 Motion Model

Motion models have been used to improve the accuracy and stability of motion estimates in vehicle applications such as vehicle tracking or navigation. The vehicle is assumed to comply with a certain motion model which describes its dynamic behavior. The choice of the motion model is crucial and greatly affects the motion estimation performance. In this work, the use of a motion model that more closely represents the vehicle motion by the application of nonholonomic constraints is proposed [73, 74, 75, 76]. As shown in Fig. 24 (1), the states are:

$$\mathbf{x}_k = [x_k, y_k, \theta_k, v_k, \phi_k]^T \tag{4.44}$$

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & -\sin(\theta_k)\cos(\phi_k)v_k\Delta t & \cos(\theta_k)\cos(\phi_k)\Delta t & -\cos(\theta_k)\sin(\phi_k)v_k\Delta t \\ 0 & 1 & \cos(\theta_k)\cos(\phi_k)v_k\Delta t & \sin(\theta_k)\cos(\phi_k)\Delta t & -\sin(\theta_k)\sin(\phi_k)v_k\Delta t \\ 0 & 0 & 1 & \sin(\phi_k)\Delta t/l_{car} & \cos(\phi_k)v_k\Delta t/l_{car} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.46}$$

where x_k and y_k are the rear center coordinates, θ_k is the global vehicle heading, v_k is the linear velocity, ϕ_k is the steering angle. I assume the vehicle is a front-wheel drive, the motion model $f(\cdot)$ governs how the state transition according to current states and time [76, 73]:

$$x_{k+1} = \cos(\theta_k) \times \cos(\phi_k) \times v_k \times \Delta t + x_k \tag{4.45a}$$

$$y_{k+1} = \sin(\theta_k) \times \cos(\phi_k) \times v_k \times \Delta t + y_k \tag{4.45b}$$

$$\theta_{k+1} = \sin(\phi_k) \times v_k \times \Delta t/l_{car} + \theta_k \tag{4.45c}$$

$$v_{k+1} = v_k \tag{4.45d}$$

$$\phi_{k+1} = \phi_k \tag{4.45e}$$

The Jacobian matrix for motion model could be find in (4.46). It is constructed at each time step.

In the motion model, I assume the steering angle ϕ_k and velocity v_k remain constant between each updating. This model is usually called Constant Turn Rate and Velocity (CTRV) model [77]. The main feature of the kinematic model of a car like vehicle is the presence of two nonholonomic constraints in (4.45) refraining the front/rear wheels from moving slide ways.

4.4.1.2 Measurement Model

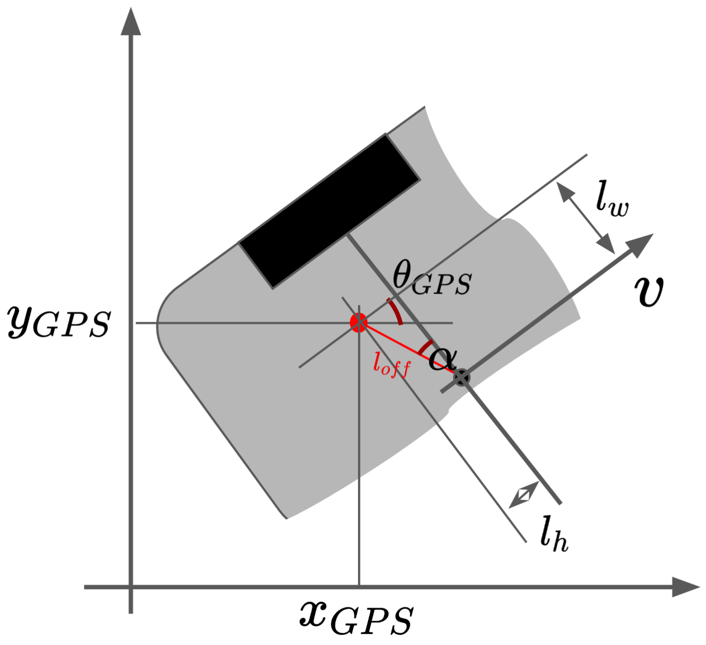


Fig. 25.: Vehicle Measurement Model, l_w is 0.32m and l_h is 0.05m .

By using the GPS data, the measurements are the center position x_k^{GPS}, y_k^{GPS} , heading angle θ_k^{GPS} of the vehicle:

$$\mathbf{z}_k = [x_k^{GPS}, y_k^{GPS}, \theta_k^{GPS}]^T \tag{4.47}$$

The measurement model $h(\cdot)$ is provided in (4.48), in which $l_{off} = 0.323$ is the distance between the GPS receiver and the vehicle rear center, and $\alpha = 0.154$ is the angle between x-axis and the line of sight between the GPS receiver and the rear center, as illustrated in Fig. 25.

$$x_k = x_k^{GPS} - l_{off}\cos(\hat{\theta}_{k|k-1} + \alpha + 0.5\pi) \tag{4.48a}$$

$$y_k = y_k^{GPS} - l_{off}\sin(\hat{\theta}_{k|k-1} + \alpha + 0.5\pi) \tag{4.48b}$$

$$\theta = \theta_k^{GPS} \tag{4.48c}$$

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & l_{off}\sin(\hat{\theta}_{k|k-1} + \alpha + 0.5\pi) \\ 0 & 1 & 0 & 0 & -l_{off}\cos(\hat{\theta}_{k|k-1} + \alpha + 0.5\pi) \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.49}$$

4.4.2 Process and Measurement Noise Covariance Matrices

For simplicity, the model assumes the constant velocity and constant steering rate, which is not realistic in real tracking applications. The state-dependent process noise covariance matrix is given in (4.50), where σ_A is the maximum acceleration and σ_{SR} is the maximum steering rate due to physical limitations. The definition of $\mathbf{G}(\cdot)$ is provided in (4.51), which is the dynamic model for acceleration and steering rate [76]. The measurement noise covariance is provided in (4.52).

$$\mathbf{Q}_k(\mathbf{x}_{k|k}) = \mathbf{G}(\mathbf{x}_{k|k})\mathbf{diag}(\sigma_A^2, \sigma_{SR}^2)\mathbf{G}(\mathbf{x}_{k|k})^T + \mathbf{diag}(\sigma_x^2, \sigma_y^2, \sigma_\theta^2, \sigma_v^2, \sigma_\phi^2) \tag{4.50}$$

$$\mathbf{G}(\mathbf{x}_{k|k}) = \begin{bmatrix} \frac{1}{2}(\Delta t)^2\cos(\phi_{k|k}) & \frac{1}{2}(\Delta t)^2\sin(\phi_{k|k}) & 0 & \Delta t & 0 \\ 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 & \Delta t \end{bmatrix}^T \tag{4.51}$$

$$\mathbf{R}_k = \mathbf{diag}(\sigma_{GPS_x}^2, \sigma_{GPS_y}^2, \sigma_{GPS_\theta}^2) \tag{4.52}$$

All the σs in (4.50) and (4.52) are variables, which are used to construct the covariance matrices.

4.4.3 Experiment Results

The whole dataset is separated to training, validation, and testing sets. The validation sets is used during the training to prevent over fitting.

4.4.3.1 Vehicle Trajectory Estimates

In Fig. 26, an example trajectory from the testing set is shown. The yellow trajectory is estimated by the EKF optimized manually, with large errors occurring when the vehicle turns. The blue trajectory is the estimate obtained by the EKFNet, which does not overshoot during the turning period, and is much closer to the ground truth. The loss function used in this case is (4.11). In Fig. 27, the positional root mean square error (RMSE) is presented by combining MSEs associated with the estimates of the vehicle coordinates x and y. From this figure, it is clear that the trained EKF has a lower estimation error than the original EKF, especially during vehicle turns.

4.4.3.2 Loss Function Comparison: MSE of State Estimate vs. Posterior Log-likelihood

As mentioned before, the loss function can be chosen to either include the second order statistic of the measurement residual/state estimation error or not. In this subsection, I discuss the difference between using a loss function based on the MSE of the state estimate error by using (4.11) and setting $b = 0$, and that based on the posterior log-likelihood by using (4.12) and setting $b = 0$. In Fig. 28, the histograms for the positional RMS error and negative posterior log-likelihood are shown. The lower the

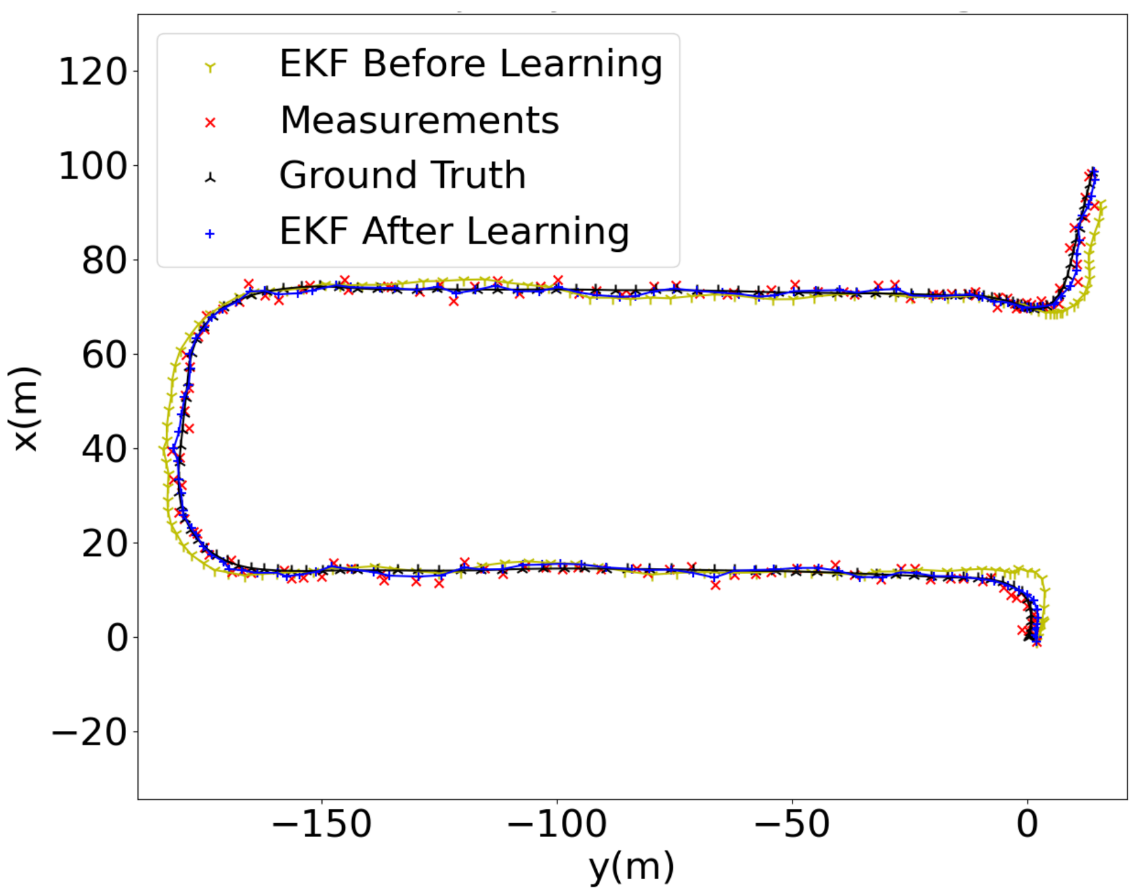


Fig. 26.: Estimated vehicle trajectories before and after training.

better for both of the metrics. The manually tuned results are displayed in blue in both of the figures. From Fig. 28(a) the long tail of the histograms for manually tuned EKF can be observed, implying large estimation errors occur with significant probabilities. In Fig. 28(b), for manually tuned EKF, most of negative posterior log-likelihoods are distributed between 4 and 8. The manually tuned covariance matrices are used to initialize the proposed training algorithm.

In the first case, the loss function is based on state estimation MSE L^{G}_{MSE}, and the corresponding RMS error histogram is shown in Fig. 28 in green. From those graphs, both the errors move toward better performance. If the loss function is created

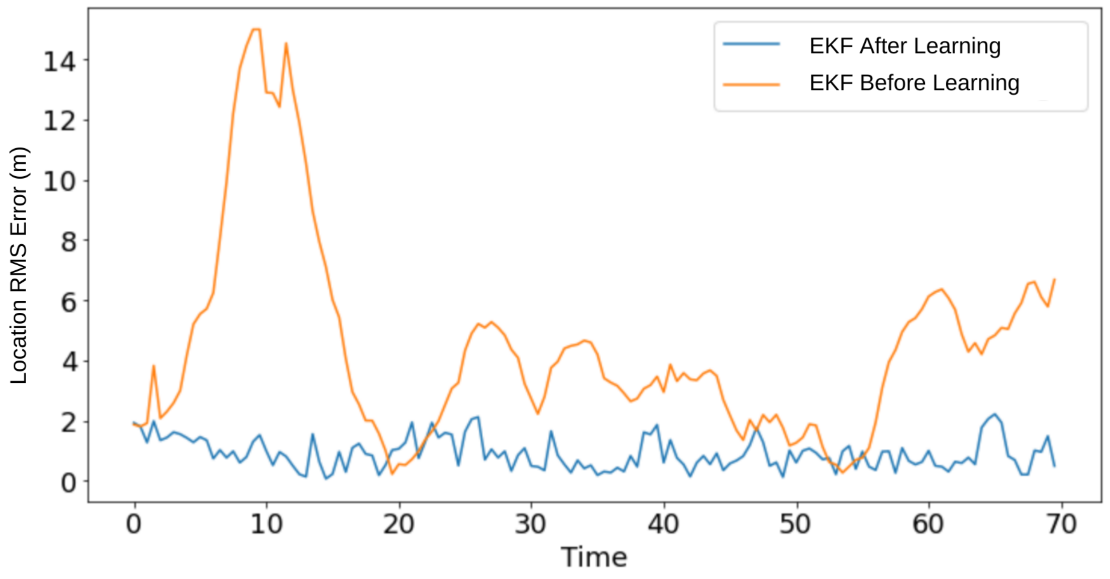
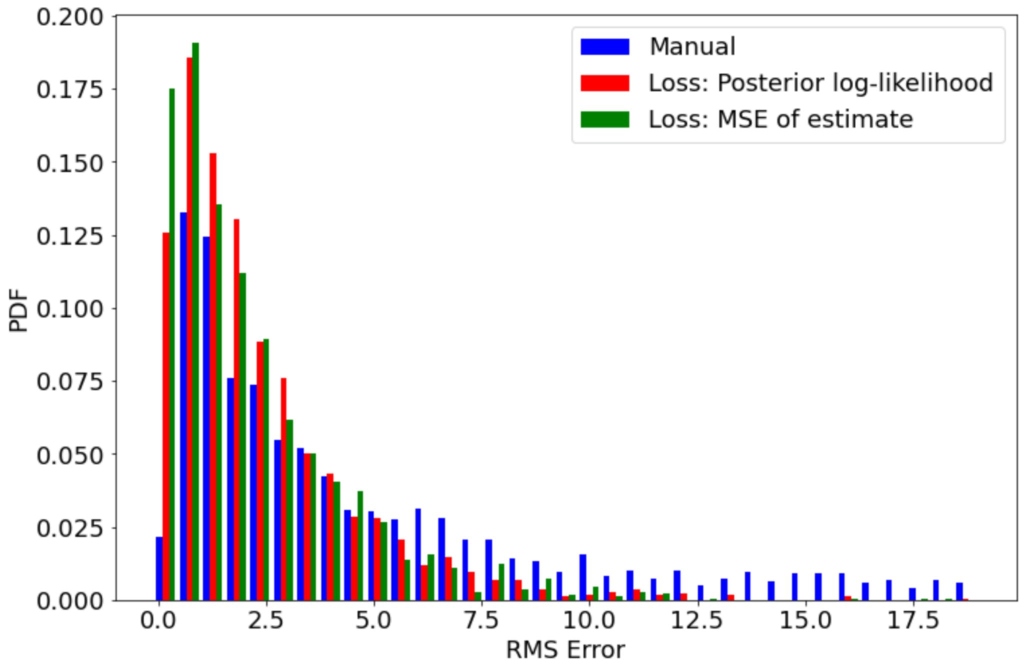


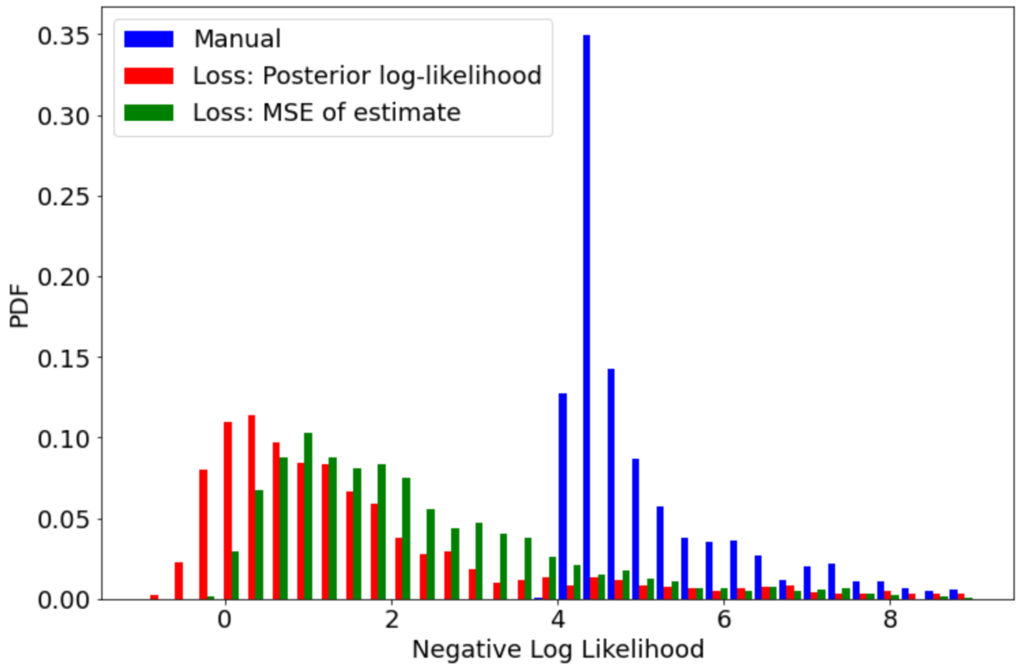
Fig. 27.: RMS error for location before and after training.

by considering the posterior state estimate covariance, namely L_L^G, the results are also shown in Fig. 28 in red. Compared with the manually tuned results in blue, the EKFs trained based on the posterior likelihood and the state estimation MSE improve the tracking performance significantly. After training, the long tails of the histograms (PDFs) of the state estimation MSE and the negative log-likelihood have diminished, and these two histograms/distributions have moved to the left with smaller losses on the average.

Further, from this figure, it is clear that a loss function based on likelihoods will lead to a better likelihood results, and a loss function based on MSEs will result in a better MSE performance. These results are consistent with their loss functions. Further, trained EKFs based on both criteria provide significantly better performance than the manually tuned EKF.



(a) Histogram of State Estimation RMS Errors



(b) Histogram of Posterior Negative Log Likelihood

Fig. 28.: Loss Histograms for EKFs trained using with L^G_{MSE}, L^G_L, and manually tuned EKF.

4.4.3.3 Numerical Results

In Table 5, the testing results are shown. Each row represents a different loss function, and each column stands for a different objective score. The loss functions are constructed as in (4.11) or (4.12). It is clear that the best score for each category is achieved by using that category as the loss function alone. If a total loss function is constructed by combining loss functions from two different categories, the scores in both categories are competitive but not the best. Further, the proposed EKFNet method always outperforms the EKF fine-tuned by hand and the Joint (sample covariance) method in [48].

Table 5.: Testing results. -LogL: negative log-likelihood.

Loss	RMSE (State)	RMSE (meas)	-LogL (State)	-LogL (meas)
$\mathbf{L}^G_{Res} + \mathbf{L}^M_{Res}$	1.235	2.434	3.843	5.673
$\mathbf{L}^G_{Res}$	**1.134**	2.563	3.745	5.996
$\mathbf{L}^M_{Res}$	1.922	**2.367**	4.745	5.856
$\mathbf{L}^G_L + \mathbf{L}^M_L$	2.367	3.412	3.562	5.432
$\mathbf{L}^G_L$	1.932	3.758	**2.341**	5.783
$\mathbf{L}^M_L$	2.745	3.227	3.735	**4.735**
Hand Tuned	4.246	6.342	8.649	9.354
Joint [48]	4.945	5.354	4.984	5.465

4.4.4 Imperfect Motion and Measurement Models

In this section, I will provide the training results using imperfect motion and measurement models. Imperfect models are common for model-based tracking due to the physical limitations. For example, in our case, the vehicle length l_{car} in Fig. 24 and the GPS sensor position in Fig. 25 are not precisely measured, which will cause estimation errors during vehicle tracking. As mentioned before, the process and measurement noise can compensate for these imperfections during tracking. Because the proposed method can be trained both with and without the ground truth data, I create two different experiment cases. The first one is with the ground truth and using the state estimation MSE as the loss function L^G_{MSE}. Another one is without the ground truth and using the residual error L^M_{Res} as the loss function. In order to run this experiment I change the l_{car} to $5.0m$, $l_{off} = 0.5$ and $\alpha = 1.6$, to make both the motion and measurement model parameters deviate from their true values.

In Fig. 29, the histograms for the state estimation MSE and the posterior negative log-likelihood are provided for two EFKs trained with different loss functions and a manually tuned EKF. The results for the manually tuned EKF are in blue. Compared with the results for the same configuration in Fig. 28, the distributions become fatter and have larger variance. The right tails have higher probabilities, due to the poorer tracking performance with the imperfect system models.

In the first EKF, the state estimation MSE L^G_{MSE} is used as a loss function. Its histograms are displayed in Fig. 28 in green. For both of the histograms, the distributions move to the left, which means the proposed training approach finds better system noise covariance matrices compared with the EKF before training.

In many cases, the state ground truth data are not available. So in the second EKF, the loss function is constructed by using the measurement residual L^M_{Res}. The histograms for the second EKF are showed in Fig. 28 in red. Compared with the original manually tuned EKF results, better tracking performance has been shown in terms of both the state estimation error and likelihood.

However, compared with the results for the first EKF trained with the ground truth data, it loses the competition in terms of both of the evaluation metrics. Especially in 29(b), its histogram moves not as far as the first EKF trained with ground truth data. This is due to the fact that during training, the loss function is constructed based on likelihood parameters that are far from their true values.

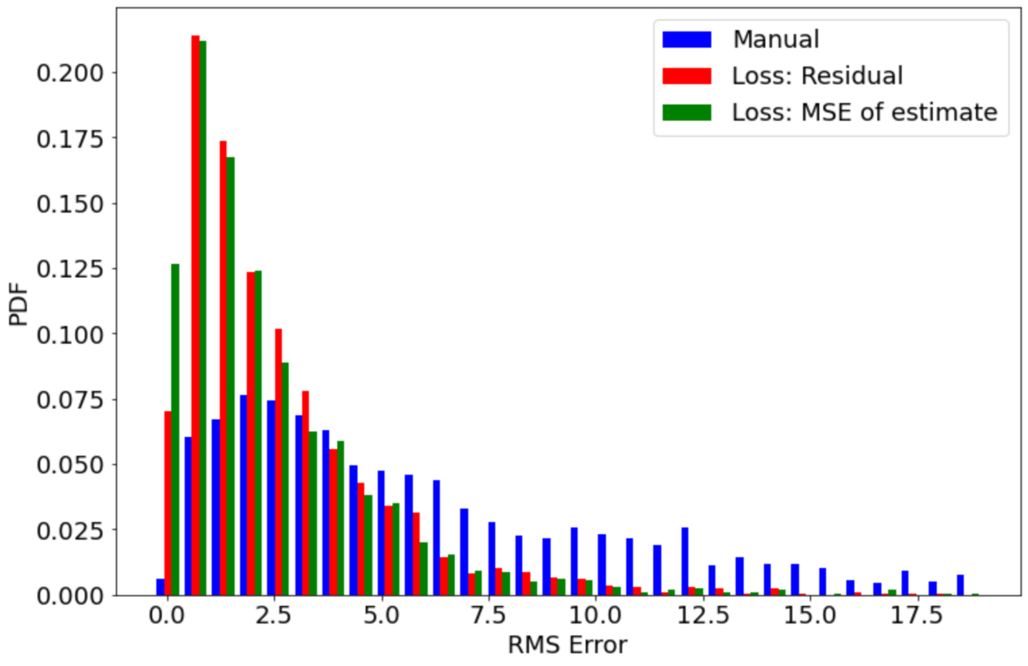
In Table 6, the tracking performance in terms of the RMSE and likelihood are provided for EKFs trained with different loss functions. From the results, the EKF trained using the ground truth outperforms the one without by all the different criteria. One of the reasons is that without using the ground truth data, the training relies on the correct model assumption. However, using imperfect models, it affects the training performance.

Table 6.: Testing results. -LogL: negative log-likelihood.

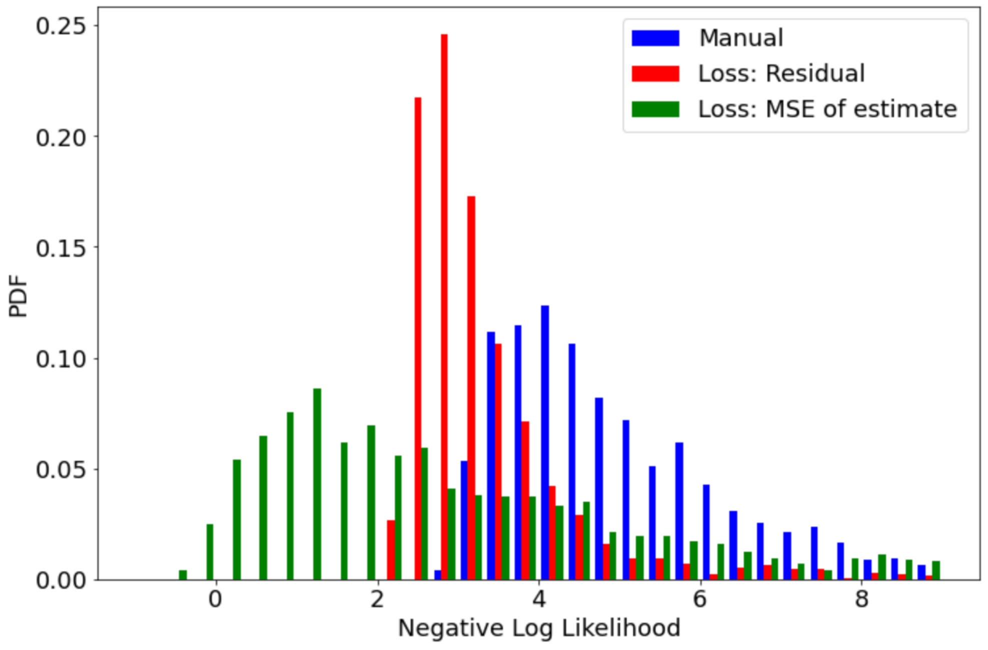
Loss	RMSE (State)	RMSE (meas)	-LogL (State)	-LogL (meas)
Manually Tuned	7.738	12.616	5.135	5.732
$\mathbf{L}_{MSE}^{G}$	**2.552**	**6.461**	**3.376**	**4.391**
$\mathbf{L}_{Res}^{M}$	2.605	6.473	3.412	4.632

4.5 Conclusion

In this chapter, a method called EKFNet was proposed, which can fine-tune the EKF automatically with or without the ground truth data. For learning the best process and measurement noise covariances offline, I presented four different objective functions. The whole framework is trained using gradient descent, and the gradients are calculated based on BPTT. In a tracking example using real GPS data,



(a) Histogram of State Estimation RMS Errors



(b) Histogram of Posterior Negative Log Likelihoods

Fig. 29.: Histogram Results for Imperfect Motion and Measurement Models.

the proposed method outperforms existing methods and a manually tuned EKF.

CHAPTER 5

TRAFFICEKF: A TRAFFIC AWARE KALMAN FILTER

Most vehicle tracking algorithms only consider the vehicle's kinematic state but ignore the information about the surrounding environment, which also plays an important role affecting how the driver controls the vehicle. In addition, how to represent the traffic information and its effect on the vesicle's state is a challenging problem. In this chapter , I propose a tracking method called traffic aware extended Kalman filter (TrafficEKF), which not only incorporates the vehicle's kinematic dynamics, but also the information from the surrounding environment. The traffic information has been represented by a bird-eye-view rasterized image, with the road shape, traffic light conditions, and other objects inside the field of view. The effect of the traffic information on vehicle driving is learned by TrafficEKF from the ground truth data. Through training, the algorithm learns to predict the control input to the vehicle and to optimize the process and measurement noise covariance matrices used by the EKF. Based on experiments with real data, I show that the TrafficEKF significantly outperforms both a manually tuned EKF, and a data trained EKF, which ignore the environment information.

5.1 Introduction

Nonlinear filtering is a critical part in an autonomous vehicle (AV)'s perception and tracking system, and it provides the ability to fuse prediction information with current measurements to enhance the tracking accuracy and mitigate the measurement error [9]. Popular nonlinear filters for state estimation include the extended

Kalman filter (EKF) [9, 45], unscented Kalman filter (UKF) [46], and the particle filter (PF) [9]. Compared with other methods the EKF is widely used due to its much lower computational load and robust performance. By using linearized dynamic and measurement models, the EKF can be used for estimating the current object's states [75].

A tracking algorithm typically uses results from a detection algorithm to estimate the object's state. Object detection and tracking could be based on LiDAR, radar, and cameras [78, 79, 80, 19]. Most of the tracking algorithms model the target as a single point object [81, 73, 82, 83], and some of them model the target as an extended object [84, 42, 85] based on LiDAR measurements. For single point assumption, the KF or EKF has been extensively used to process the output from the detection algorithm. For the extended object assumption, usually the target is modeled as a particular shape such as a rectangle or an ellipse, and the tracking algorithm can process multiple detection points originated from the tracked target [85, 42]. Also, many vehicle motion models only consider the kinematic state, and ignore the control input, which is usually unknown [75] to the AV. However, when a vehicle is being driven on the road, it is not only following the physical model, but also affected by driving rules and environment conditions [86, 87] such as the road shape, traffic lights, and other objects on the road. This information is usually missing in the EKF motion model when the algorithm makes the state estimate. How to combine the vehicle dynamics and the environment information to estimate vehicle state accurately, is still a challenging task for the EKF.

In recent years, several AV companies have made some progress on the vehicle motion prediction. They have not only published their approaches [86, 88, 89, 90, 91], but also provided openly available datasets [87, 92, 93]. To represent the driving environment by a known data structure, two popular methods have been used recently. One method uses the birds-eye-view (BEV) rasterized image [86, 88, 89] to learn the features through the Convolutional Neural Network (CNN). The rasterized image contains the road shape, traffic light, and other objects' locations on the road. It uses different colors to represent the objects' classes and their conditions. However, the objects' dynamic states like speeds and steering angles are usually missing in the rasterized image. Another method applies graphs to represent the relationships among different components and learn the features through the Graph Neural Network (GNN) [90, 91]. In this case, every object is represented as a node in the graph, which is easy to represent the dynamic states and relationship between the objects. But in GNN, it is difficult to represent the geometry of the road and other traffic information. Most of the vehicle trajectory prediction algorithms only predict the trajectory based on tracking results [87]. How to use the extracted features to improve vehicle tracking accuracy is still an open problem.

In this chapter , I propose an EKF based tracking algorithm called TrafficEKF, which takes advantage of the information about a vehicle's surrounding environment. Different from existing trajectory prediction methods, the proposed TrafficEKF explicitly incorporates the vehicle's physical dynamic model and a state estimator (EKF), and predicts the control input of the dynamic model. The training of the TrafficEKF is based on the back propagation through time (BPTT) concept [68] proposed for the EKFNet in [7], by either minimizing the measurement residual error or maximizing the measurement likelihood. Note that the problem of road map aided vehicle tracking has been studied by researchers in the target tracking community [94, 95, 96]. In approaches proposed in [94, 95, 96] and references therein, the road map provides constraints on the vehicle's state, which enhances the tracking performance. The proposed TrafficEKF is different from these approaches, since it is a learning based approach that learns an environment feature extractor and the optimal noise covariance matrices for the EKF.

In summary, the new contributions in this chapter are:

- I construct a new framework for the EKF to incorporate both the vehicle dynamics and environment information, by combing deep learning techniques and physical dynamic models.

- I develop a method to train the proposed framework by using BPTT and construct loss functions for the TrafficEKF.

- The TrafficEKF can efficiently predict the control input for the vehicle from environment information, and automatically fine tune the process and measurement noise covariance matrices from measurement data.

The rest of the chapter is organized as follows. In Section 5.2, some background knowledge is introduced about the EKF and the vehicle motion and measurement models. In Section 5.3, the framework of TrafficEKF is proposed. Experiments results are provided in Section 5.4. Concluding remarks are given in Section 5.5. The majority material of this chapter is based on Traffic Aware EKF paper [97].

5.2 Preliminaries

5.2.1 Extended Kalman Filter

At time k, let us denote the vehicle's state as $\mathbf{x}_k$, and the measurement as $\mathbf{z}_k$. The state transition and measurement models are provided as follows:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{v}_k \tag{5.1a}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{w}_k \tag{5.1b}$$

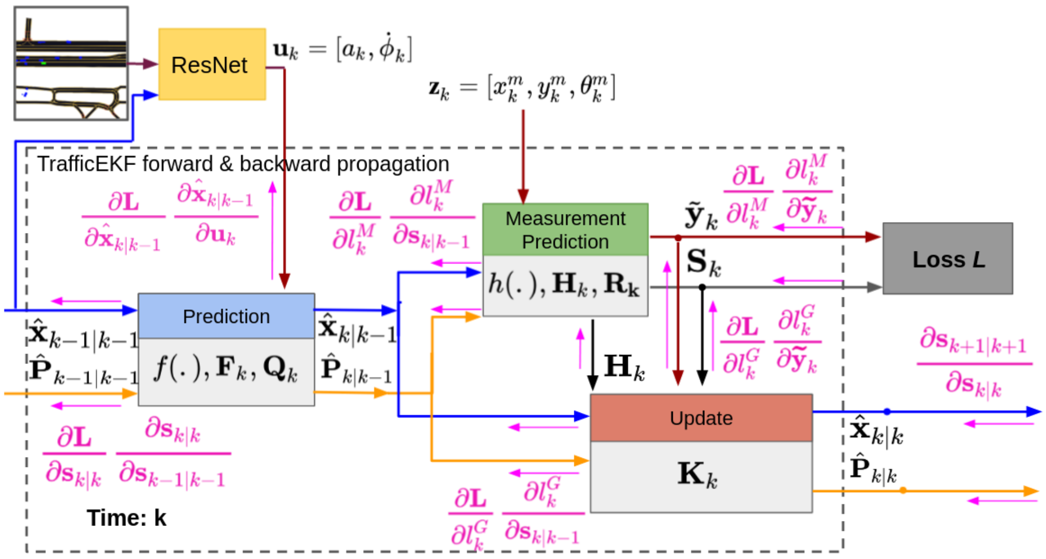

ResNet
$\mathbf{u}_k = [a_k, \dot{\phi}_k]$
$\mathbf{z}_k = [x_k^m, y_k^m, \theta_k^m]$
TrafficEKF forward & backward propagation
$\frac{\partial \mathbf{L}}{\partial \hat{\mathbf{x}}_{k|k-1}} \frac{\partial \hat{\mathbf{x}}_{k|k-1}}{\partial \mathbf{u}_k}$
$\frac{\partial \mathbf{L}}{\partial l_k^M} \frac{\partial l_k^M}{\partial \mathbf{s}_{k|k-1}}$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R}_k$
$\tilde{\mathbf{y}}_k$
$\frac{\partial \mathbf{L}}{\partial l_k^M} \frac{\partial l_k^M}{\partial \tilde{\mathbf{y}}_k}$
$\mathbf{S}_k$
Loss L
Prediction
$f(.), \mathbf{F}_k, \mathbf{Q}_k$
$\hat{\mathbf{x}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k-1|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\hat{\mathbf{P}}_{k|k-1}$
$\mathbf{H}_k$
$\frac{\partial \mathbf{L}}{\partial l_k^G} \frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial \mathbf{s}_{k+1|k+1}}{\partial \mathbf{s}_{k|k}}$
Update
$\mathbf{K}_k$
$\hat{\mathbf{x}}_{k|k}$
$\hat{\mathbf{P}}_{k|k}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{s}_{k|k}} \frac{\partial \mathbf{s}_{k|k}}{\partial \mathbf{s}_{k-1|k-1}}$
Time: k
$\frac{\partial \mathbf{L}}{\partial l_k^G} \frac{\partial l_k^G}{\partial \mathbf{s}_{k|k-1}}$

Fig. 30.: TrafficEKF diagram. $\mathbf{s} = \{\hat{\mathbf{x}}, \mathbf{P}\}$.

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions, and cannot be used directly to update the covariances. Instead, the Jacobian matrices $\mathbf{F}_k$ and $\mathbf{H}_k$ are used. $\mathbf{v}_k$ and $\mathbf{w}_k$ are the process and measurement noises, which are assumed to be zero mean Gaussian random vectors with covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively. $\mathbf{u}_k$ is the control input, which is usually unknown for the tracked vehicle. For the proposed method, $\mathbf{u}_k$ could be estimated from the traffic information according to the current state and the traffic information.

In Fig. 30 the proposed framework is shown at time k. The rightward information flows in green and orange are for the common EKF algorithm. The pink arrows pointing to the left are the gradient flows which are used for training the unknown system parameters. The proposed framework can be used as a common EKF by combining the traffic information for online updating, and it can be used to learn the unknown system parameters by BPTT [7]. The details on parameter training are provided in Section 5.3.

The prediction module in Fig. 30 is used for state prediction by using the previous updated state estimate and the control:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{5.2a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{5.2b}$$

The measurement prediction module is used to evaluate the measurement residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{5.3a}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{5.3b}$$

The update module in Fig. 30 provides the posterior (updated) state estimate:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{5.4a}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{5.4b}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{5.4c}$$

The updated state mean $\hat{\mathbf{x}}_{k|k}$ and covariance $\mathbf{P}_{k|k}$ will be used as prior information in the next time step $k + 1$.

5.2.2 Vehicle Motion and Measurement Models

I use the modified nonholonomic vehicle motion model with control inputs [73, 74, 75, 76]. The main feature of the kinematic model of a car-like vehicle is the presence of two nonholonomic constraints in (5.7) refrains the front/rear wheels from moving side way. Further, I assume that the vehicle is front wheel driving, and the rear wheel center represents the vehicle location state. As shown in Fig. 31, the state and control are:

$$\mathbf{x}_k = [x_k, y_k, \theta_k, v_k, \phi_k]^T \tag{5.5}$$

$$\mathbf{u}_k = [a_k, \dot{\phi}_k]^T \tag{5.6}$$

where x_k and y_k are the rear center coordinates, θ_k is the vehicle heading, v_k is the linear velocity, and ϕ_k is the steering angle, the angle between the vehicle heading direction and the steered wheel direction. The length and width of the vehicle are assumed to be constant during the tracking period, and known to the EKF. For control inputs, a_k is the acceleration, and $\dot{\phi}_k$ is the steering turn rate. The motion model $f(\cdot)$ governs how control inputs affect the state [76, 73], which is shown in Fig. 31 :

$$x_{k+1} = x_k + v_k \cos\theta_k \cos\phi_k \Delta t + 0.5\Delta t^2 \cos\phi_k a_k \tag{5.7a}$$

$$y_{k+1} = y_k + v_k \sin\theta_k \cos\phi_k \Delta t + 0.5\Delta t^2 \sin\phi_k a_k \tag{5.7b}$$

$$\theta_{k+1} = \theta_k + v_k \sin\phi_k \Delta t / l_{car} + 0.5\Delta t^2 \dot{\phi}_k \tag{5.7c}$$

$$v_{k+1} = v_k + a_k \Delta t \tag{5.7d}$$

$$\phi_{k+1} = \phi_k + \dot{\phi}_k \Delta t \tag{5.7e}$$

The measurements from Lyft prediction datasets [87] are the center position x_k^m, y_k^m and heading angle θ_k^m of the vehicle. I assume that the measurement vector is corrupted by the noise $\mathbf{w}_k$:

$$\mathbf{z}_k = [x_k^m, y_k^m, \theta_k^m]^T + \mathbf{w}_k \tag{5.8}$$

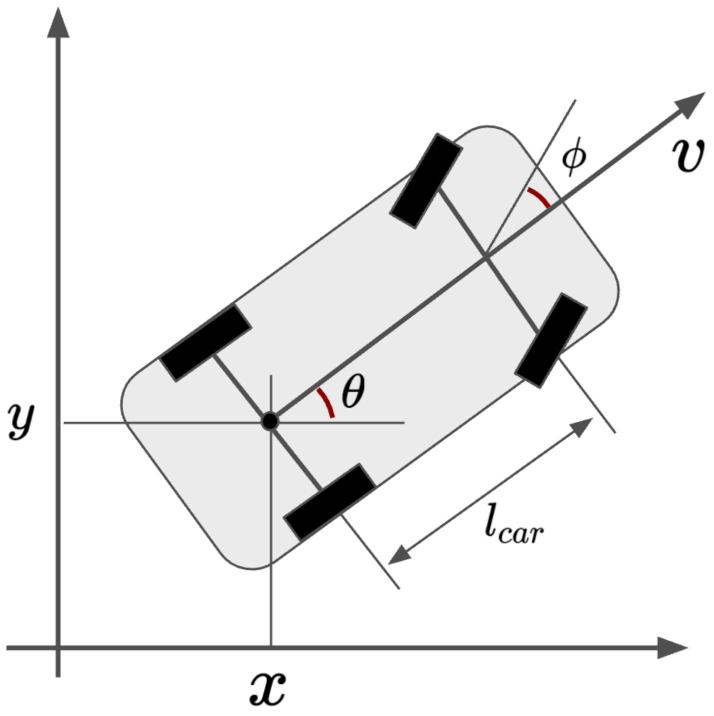


Fig. 31.: Motion model for Traffic Aware EKF.

The measurement model $h(\cdot)$ is illustrated in Fig. 32 and provided as follows

$$\begin{aligned} x_k^m &= x_k + 0.5 l_{car} \cos\theta_k \\ y_k^m &= y_k + 0.5 l_{car} \sin\theta_k \\ \theta_k^m &= \theta_k \end{aligned} \tag{5.9}$$

in which the measurement heading θ_k is the angle with respective to the x-axis, with a range of $(-\pi, \pi]$. Further, it is assumed that the rear center is 80% from the front, and $l_{car} = 0.6 l_l$. Clearly, the model in (5.9) is nonlinear, as I use the rear center as the reference point. The measurement model can be used to predict the current measurement based on the predicted state $\hat{\mathbf{x}}_{k|k-1}$.

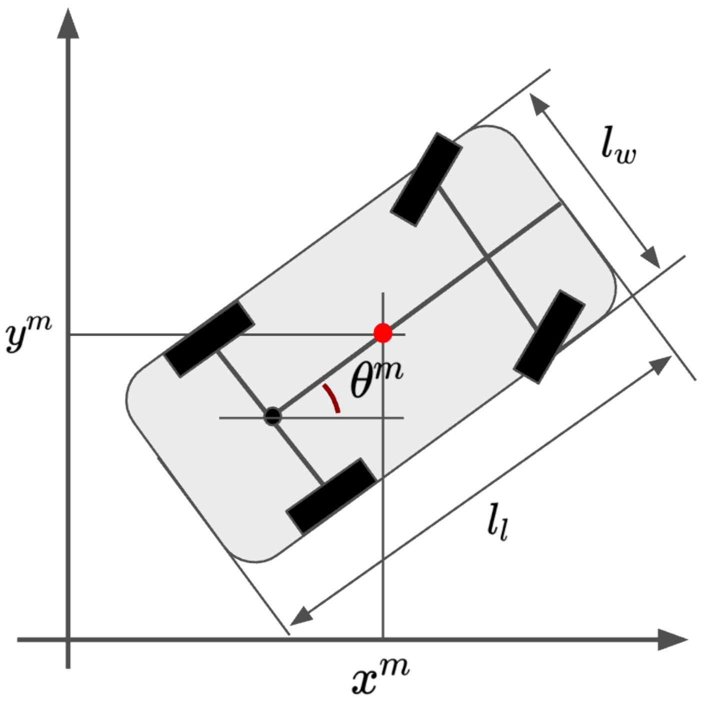


Fig. 32.: Measurement model for Traffic Aware EKF.

5.2.3 Process and Measurement Noise Covariances

For simplicity, I assume that the process noise is state dependent and the measurement noise is independent from the process noise. The state dependent process noise covariance matrix is given in (5.10), where σ_A is the maximum acceleration and σ_{SR} is the maximum steering rate due to physical limitations of the vehicle. The definition of $\mathbf{G}(\cdot)$ is provided in (5.11), which is the dynamic model for acceleration and steering rate [76].

$$\mathbf{Q}_k(\mathbf{x}_{k|k}) = \mathbf{G}(\mathbf{x}_{k|k})\mathbf{diag}(\sigma_A^2, \sigma_{SR}^2)\mathbf{G}(\mathbf{x}_{k|k})^T \tag{5.10}$$

$$\mathbf{G}(\mathbf{x}_{k|k}) = \begin{bmatrix} \frac{1}{2}(\Delta t)^2\cos(\phi_{k|k}) & \frac{1}{2}(\Delta t)^2\sin(\phi_{k|k}) & 0 & \Delta t & 0 \\ 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 & \Delta t \end{bmatrix}^T \tag{5.11}$$

The measurement noises are assumed to be independent over time, whose covariance matrices are provided below:

$$\mathbf{R}_k = \mathbf{diag}(\sigma_{z_x}^2, \sigma_{z_y}^2, \sigma_{z_\theta}^2) \tag{5.12}$$

All the σs in (5.10) and (5.12) are unknown variables, which are used to construct the covariance matrices. Totally there are 5 σs, which need to be learned during the training of the TrafficEKF.

5.2.4 Environment Conditions and Controls

The motion model in (5.7) involves both the vehicle's state and its control, and the control of the tracked vehicle is usually not known to the tracker used by the AV. A motion model without the control input will cause extra estimation errors and degrade the tracker's performance, as illustrated in Fig. 33 and 34. In Fig. 33, usually a vehicle changes its direction following a road. However, without predicting the control based on the road information, a tracker is not able to accurately predict the vehicle's future location which is affected by the road shape. Similarly, in Fig. 34 the effect of disregarding the traffic light information is illustrated. Also, other objects such as other vehicles and pedestrians all affect how the tracked vehicle is driven. In summary, without using the environment information, a tracker's performance will be degraded. However, how to estimate the control inputs in EKF is still an open problem.

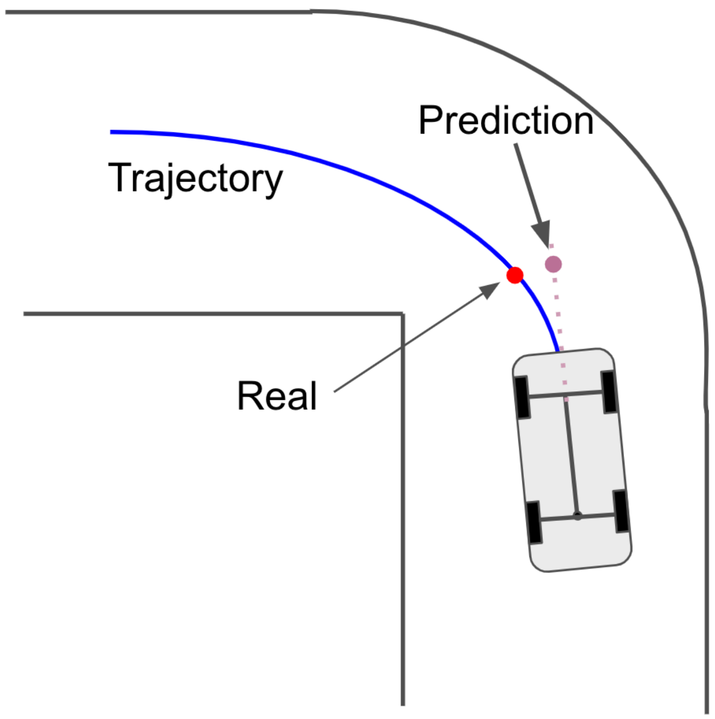


Fig. 33.: Road condition affecting steering state.

5.3 Methodology

In this section, I introduce the proposed method TrafficEKF to improve vehicle tracking performance using traffic information. I will present the method to represent the traffic information in an rasterized image, and provide discussions on how to process the rasterized image, and incorporate its information to the EKF.

5.3.1 Rasterized Images

In this chapter , I use the BEV rasterized image to represent the surrounding environment, which includes information about the roads, traffic elements, traffic lights conditions, and other detected traffic agents [87, 92, 93]. The rasterized image is used for AV motion forecasting, which is created around the target vehicle based

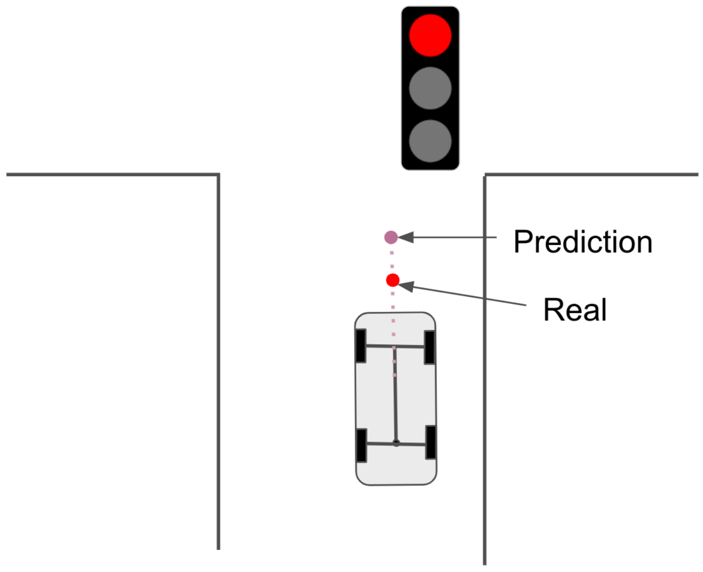


Fig. 34.: affic light condition affecting acceleration.

on the filtered results for detected traffic agents from previous updates and human annotated high definition (HD) semantic maps, as illustrated in Fig. 35. The semantic map usually contains all the labeled static elements along the route, such as positions of traffic lights, and lane boundaries and connectivity. The traffic agents include all the other detected vehicles, bikes, and pedestrians.

In the rasterized image, RGB colors are used to distinguish different items. At time k, the rasterized image is created for tracked vehicle i by using the HD map and tracking results from time $k-1$, and denoted as F_k^i. The tracked vehicle is created as green rectangular box centered at a specified location in the rasterized image with 0 heading by translating and rotating the whole scene. The traffic light and its condition are indicated by colored lane boundaries, e.g., red lane means that red light is on. The other traffic agents are represented by blue boxes, and the black region represents the road. Each pixel represents 0.5 meters. Details on the generation of rasterized images can be found in [87].

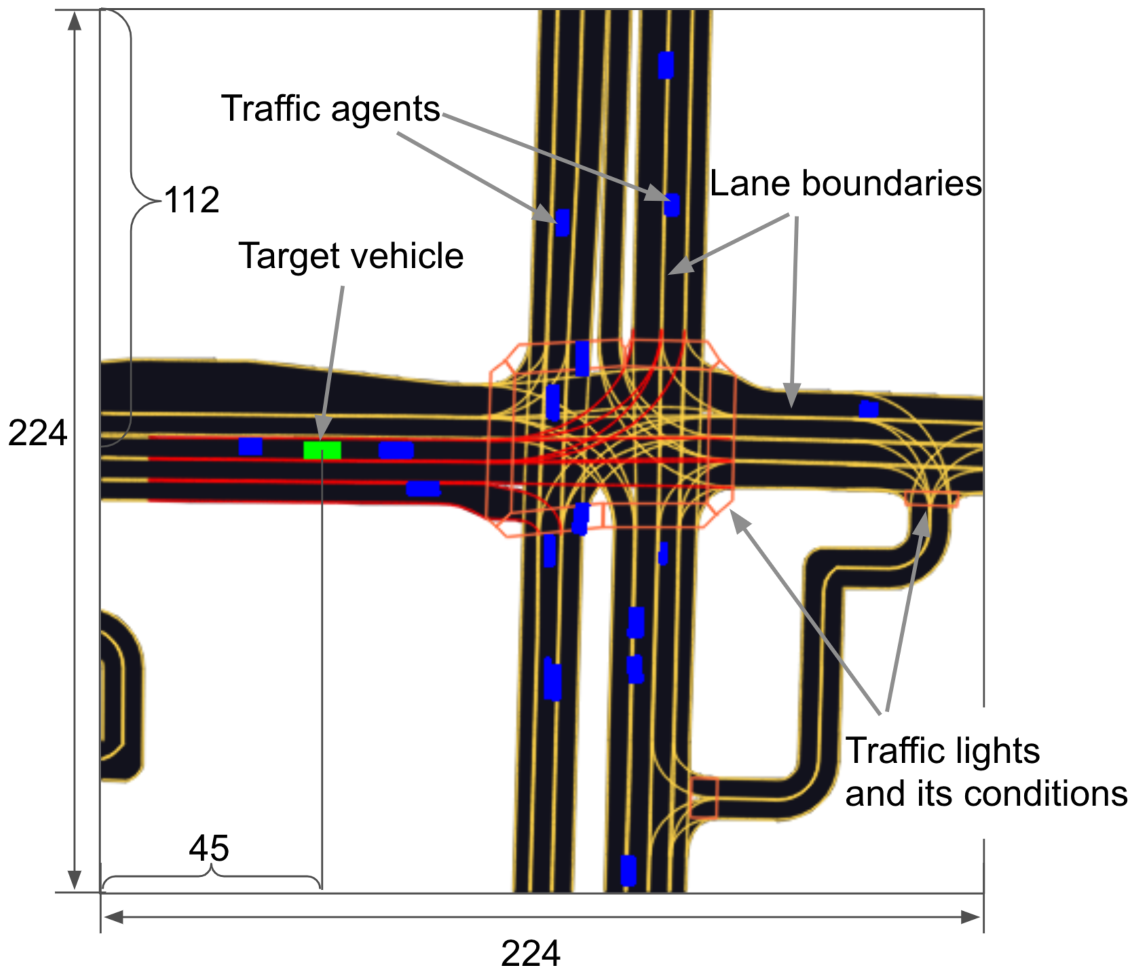


Fig. 35.: Rasterized image for target vehicle.

5.3.2 Network Structure

The network used for predicting the control $\mathbf{u}_k$ is illustrated in Fig. 36. The inputs are the past rasterized images and the past states. A ResNet [98] is used to extract a 1×1000 feature vector from a rasterized image. The image feature and the updated state estimate $\mathbf{C}\hat{\mathbf{x}}_{k-1|k-1}$ are concatenated together as the input to two fully connected (FC) layers [15], where $\mathbf{C}$ is a matrix to extract $\hat{\phi}_{k-1|k-1}$ and $\hat{v}_{k-1|k-1}$ from $\hat{\mathbf{x}}_{k-1|k-1}$. ReLU [15] is used as a nonlinear activation function between the two FC layers. The two FC layers fuse the image feature and the vehicle state estimate, and output the control prediction, which is activated by a scaled Tanh function. The scaled Tanh is adopted here because the acceleration can be either negative when the vehicle is braking, or positive when the vehicle is accelerating. The same reasoning also applies to the steering rate $\dot{\phi}_k$. Because the output range from Tanh is $[-1, 1]$, a scaled version of Tanh is needed to predict the control. In this chapter , I assume that the scale factors are 3.0 for the acceleration, and 0.5 for the steering rate, respectively.

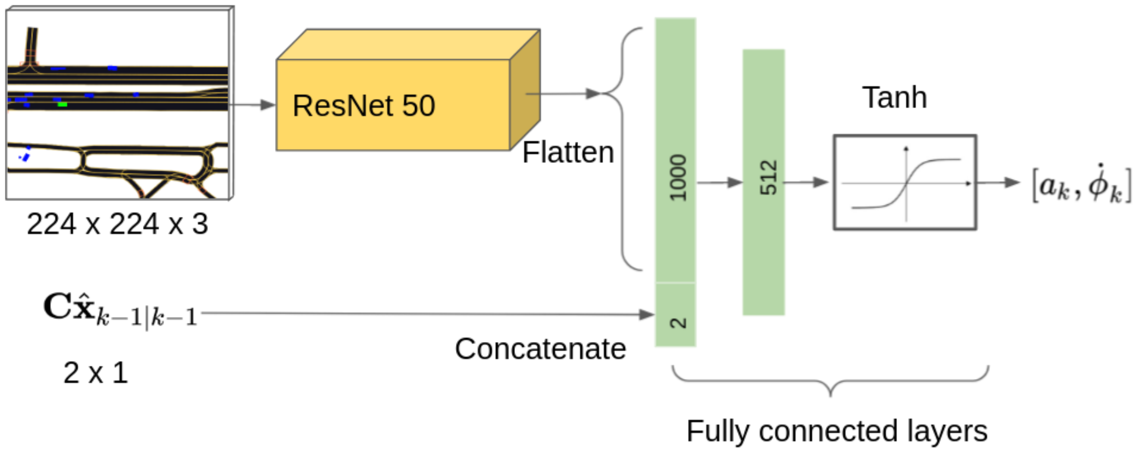


Fig. 36.: Predicting control inputs from rasterized images.

Similar network structures have been used for predicting the vehicle position recently without considering state estimation explicitly [86, 87, 99]. Different from these existing methods, the proposed network explicitly incorporates the vehicle's physical dynamic model and a state estimator (EKF), and predicts the control input of the dynamic model. The whole network is trained based on the EKF framework, which is an end-to-end learning method. In Table 7, the details of the control prediction network are provided.

Table 7.: Detailed Network Structure.

Name	Input Dimension	Output Dimension
ResNet50	$3 \times 244 \times 244$	1×1000
FC1	1×1002	1×512
ReLU	1×512	1×512
FC2	1×512	1×2
Tanh	1×2	1×2

5.3.3 Loss Functions for Training

In this subsection, I introduce the loss functions used for training the TrafficEKF.

5.3.3.1 Residual Error and Log-Likelihood

The measurement residual $\tilde{\mathbf{y}}_k$ is the output from the measurement prediction module. The loss function based on residual errors from time 1 to T can be defined as:

$$L_{Res}^{M} = \sum_{k=1}^{T} \|\tilde{\mathbf{y}}_k\|_2^2 \tag{5.13}$$

Note that the calculation of the residual error does not involve its covariance $\mathbf{S}_k$, which is also evaluated by the EKF at each time step.

Instead of minimizing the residual error, I can maximize the measurement likelihood, by considering not only the residual, but also its uncertainty. The negative log-likelihood for the measurement is provided as follows:

$$L_{L}^{M} = -\sum_{k=1}^{T} \log p(\mathbf{z}_k|\mathbf{z}_{0:k-1}) \tag{5.14}$$

5.3.3.2 Loss Functions and Regularization

I propose two different loss functions, based on either the residual error or the negative log-likelihood as follows:

$$L_{RMS} = L_{Res}^{M} + \lambda_1 r_1(\mathbf{Q}) + \lambda_2 r_2(\mathbf{R}) \tag{5.15}$$

$$L_{Like} = L_{L}^{M} + \lambda_1 r_1(\mathbf{Q}) + \lambda_2 r_2(\mathbf{R}) \tag{5.16}$$

Each loss function consists of two parts: the first term is either the residual error or the negative log-likelihood, and the last two terms are regularization functions ($r(\cdot)$ functions). The regularization functions are problem dependent. For example, they can encourage the covariance matrix to be close to the manually optimized result [47]. λ_1, and λ_2 are the weights for regularization terms. The loss function encourages the filter output to minimize the residual error or maximize the measurement likelihood.

5.3.4 TrafficEKF

The TrafficEKF forward and backward operations are illustrated in Fig. 30. The forward operation is an EKF used for tracking, and the backward operation is to learn the unknown parameters, which include the process noise covariance $\mathbf{Q}$, measurement noise covariance $\mathbf{R}$, and the parameters in the ResNet, collectively denoted as $\boldsymbol{\psi}$. Different from existing methods, the network predicts the control inputs according to the surrounding environment, and explicitly incorporates the physical dynamical model of the vehicle.

To train the TrafficEKF, I use the BPTT method to calculate gradient with respect to each training parameter. BPTT is a gradient-based method for training a directed computation graph based on time series [7, 68, 15, 70]. The training data are a sequence of input-output pairs $\{(\mathbf{z}_k, F_k), (\hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}, \tilde{\mathbf{y}}_k, \mathbf{S}_k)\}_{k=1}^{T}$, where the rasterized image F_k is constructed based on the perception results by combining the tracked agents' state estimates and the HD map. Note that the locations of all the other tracked agents and the the HD map are given in the datasets [87], meaning that the image can be easily constructed. When creating the rasterized image the location and heading of the target vehicle are based on the filtered results in the dataset for simplicity. Finally, the gradient descent optimization method is used to update the parameters [70, 15].

The back propagation flow is shown in Fig. 30. The pink equations and left arrows indicate the backward gradient flows. At each step k, the gradients are derived from: (1) measurement error L_k^M, which is the error made by measurement prediction L_{RMS}, or L_{Like}, (2) gradients from time step $k+1$. Detailed derivations for back propagation are skipped in this chapter for brevity. Details of back propagation for the EKF part can be found in our previous work [7]. The gradients for updating the noise covariances are:

$$\frac{\partial L}{\partial \mathbf{Q}} = \sum_{k=1}^{T} \frac{\partial L}{\partial \mathbf{P}_{k|k-1}} \frac{\partial \mathbf{P}_{k|k-1}}{\partial \mathbf{Q}} \tag{5.17a}$$

$$\frac{\partial L}{\partial \mathbf{R}} = \sum_{k=1}^{T} \frac{\partial L}{\partial \mathbf{S}_k} \frac{\partial \mathbf{S}_k}{\partial \mathbf{R}} \tag{5.17b}$$

The gradient updating the entire ResNet can be calculated as:

$$\frac{\partial L}{\partial \boldsymbol{\psi}} = \sum_{k=1}^{T} \frac{\partial L}{\partial \hat{\mathbf{x}}_{k|k-1}} \frac{\partial \hat{\mathbf{x}}_{k|k-1}}{\partial \mathbf{u}_k} \frac{\partial \mathbf{u}_k}{\partial \boldsymbol{\psi}} \tag{5.18}$$

5.4 Experiments

In order to test our algorithm, I use the Lyft prediction dataset [87] for the experiments. First, the framework is trained based on the filtered result which is provided by the Lyft dataset. After the framework has been trained, I test its tracking performance based on noisy detections, which are generated by adding noise to the original detections.

5.4.1 Implementation Details

The proposed algorithm is implemented in Pytorch [100] and Numpy [101] on a PC with an Intel Xeon, 32G RAM, and Nvidia RTX 2080Ti. The manually tuned EKF parameters are $\sigma_A^2 = 3.5$, $\sigma_{SR}^2 = 0.5$, $\sigma_{z_x}^2 = 0.1$, $\sigma_{z_y}^2 = 0.1$, and $\sigma_{z_\theta}^2 = 0.01$.

5.4.2 Dataset Generation

The Lyft prediction dataset [87] is used for predicting the vehicle trajectories, not for tracking the vehicle. Therefore, I modify the datasets to test our algorithm. Each scene is 25 seconds long with measurement data at 10Hz. During each scene, there is a different number of objects such as vehicles and pedestrians, which are marked in different colors in the rasterized image. For each vehicle at each different time step, I generate a corresponding rasterized image. Also, I select tracks that last at least 15 seconds, for training and testing the proposed algorithm. Longer tracks are preferred since they provide more information for training.

For training, I use the detections which are given in the Lyft dataset to learn the system noise covariance matrices and the parameters in the ResNet. After the training, the TrafficEKF can understand the effect by the surrounding environment on the vehicle's control. For this experiment, I simulate the measurements by adding some random noise to the original filtered results. Then I test the whole framework with the noisy measurements. The original results can be treated as ground truth for testing the algorithm. The dataset is divided into training and validation sets.

5.4.3 Training Results

In Table 8, the results for validation sets are provided. Each row represents a certain loss function with a certain set of parameters being trained in the learning process.The columns include the root mean square error (RMSE) and negative log-likelihood for the measurement residuals. Results in the first two rows are obtained by training the network parameters $\boldsymbol{\psi}$, and the covariance matrices $\mathbf{R}$ and $\mathbf{Q}$. Results in the third and fourth rows are based on the EKF with trained $\mathbf{R}$ and $\mathbf{Q}$ without using the predicted control inputs, which is equivalent to the EKFNet proposed in [7]. In the fifth and sixth rows, results are given by only training the network parameters $\boldsymbol{\psi}$ to predict the control inputs. In the last row are results for a manually tuned EKF. It is clear that by training all the parameters ($\boldsymbol{\psi}$, $\mathbf{R}$, and $\mathbf{Q}$), the algorithm can deliver the optimal result, in terms of either the minimum RMSE or the maximum likelihood. As expected, in general the loss function based on RMS leads to a smaller RMSE, and that based on the likelihood leads to a larger likelihood. I also test the result by only train the noise covariance or the ResNet. The results are all better than the manually tuned EKF, but not as good as those achieved via training all the system parameters. These results show that the TrafficEKF works the best when all the parameters are trained by the data.

In Fig. 38 the histogram of RMS values by using the loss function $L_{RMS}(\boldsymbol{\psi}, \mathbf{Q}, \mathbf{R})$ is shown. I can see the distribution has a long and heavy tail before the training in Fig. 37. After the training, the RMS errors become more concentrated to small values, and the long tail disappears.

The runtime for this algorithm is around 80 Hz without considering the rasterized image generation. The generation of rasterized images is much slower than the algorithm itself, it is around 5 Hz. The ResNet is fully optimized and running on

Table 8.: Validation results. -LogL: negative log-likelihood.

Objective Function	RMSE (meas)	-LogL (meas)
$L_{RMS}(\boldsymbol{\psi}, \mathbf{Q}, \mathbf{R})$	**0.0009**	-5.968
$L_{Like}(\boldsymbol{\psi}, \mathbf{Q}, \mathbf{R})$	0.0014	**-6.650**
$L_{RMS}(\mathbf{Q}, \mathbf{R})$	0.0049	-4.156
$L_{Like}(\mathbf{Q}, \mathbf{R})$	0.0063	-5.324
$L_{RMS}(\boldsymbol{\psi})$	0.0039	-3.805
$L_{Like}(\boldsymbol{\psi})$	0.0078	-4.382
Manually tuned EKF	0.0110	-2.522

the GPU, but the rasterized image is created on a single core CPU. The program is not fully optimized for creating the rasterized image, which can be optimized in our future work.

5.4.4 Testing Results

The testing results are obtained by using the noisy measurements from the validation datasets. In Fig. 39, the state estimates over time (trajectories), obtained by the manually tuned EKF and the trained TrafficEKF respectively, are shown. The TrafficEKf, represented by the blue trajectory, takes advantage of the map information, especially during turns. The beginning is in the top right, and there is a straight lane followed by a turn. In the beginning, both the TrafficEKF and EKF behave the same. But during the turn, the EKF, represents by the green trajectory, overshoots and has a bias error during the turn.

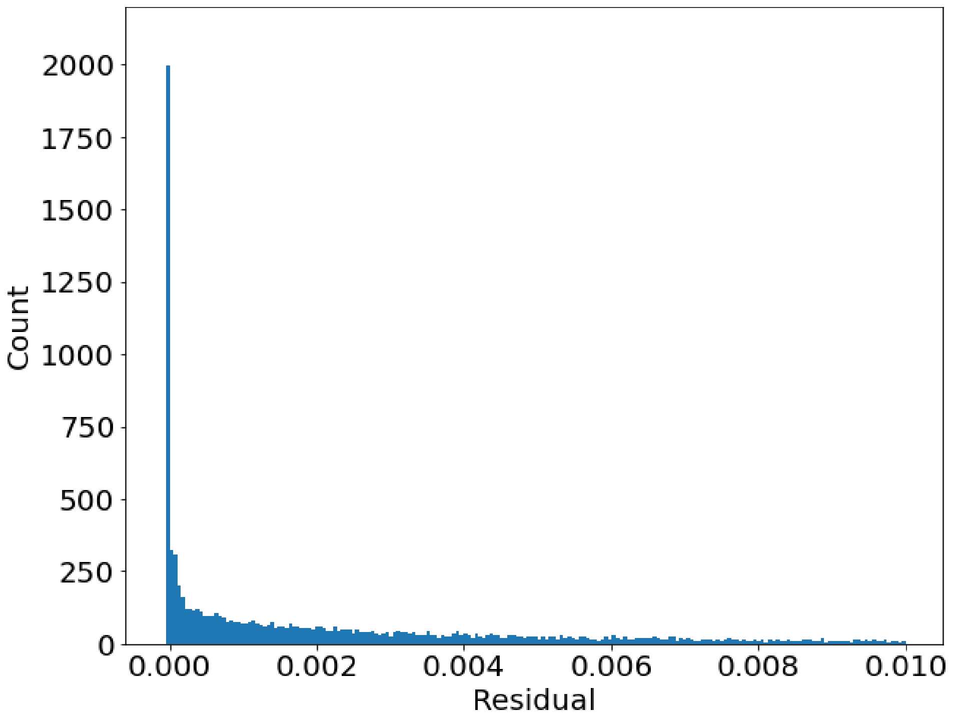


Fig. 37.: Residual histograms before Traffic Aware training.

In Fig. 40, an example is provided where the vehicle stops in front of a red traffic light, with detailed rasterized images illustrated in Figs. 41 and 42. The vehicle is decelerating in front of the red traffic light. In the TrafficEKF, the traffic light information is incorporated. It is clear from Fig. 40, the trajectory estimated by the TrafficEKF is closer to the ground truth, while the EKF result converges slowly without the traffic light information.

In Table 9, the testing results are compared with the ground truth. The performance measures used here are the RMSE and negative log likelihood. The RMSE is evaluated by taking the difference between the updated state estimate and the ground truth. The likelihood here means the vehicle state's posterior probability density function (PDF), evaluated at its true state. Usually these values, especially the RMSE, are used for evaluating the performance of a filter. For both performance measures,

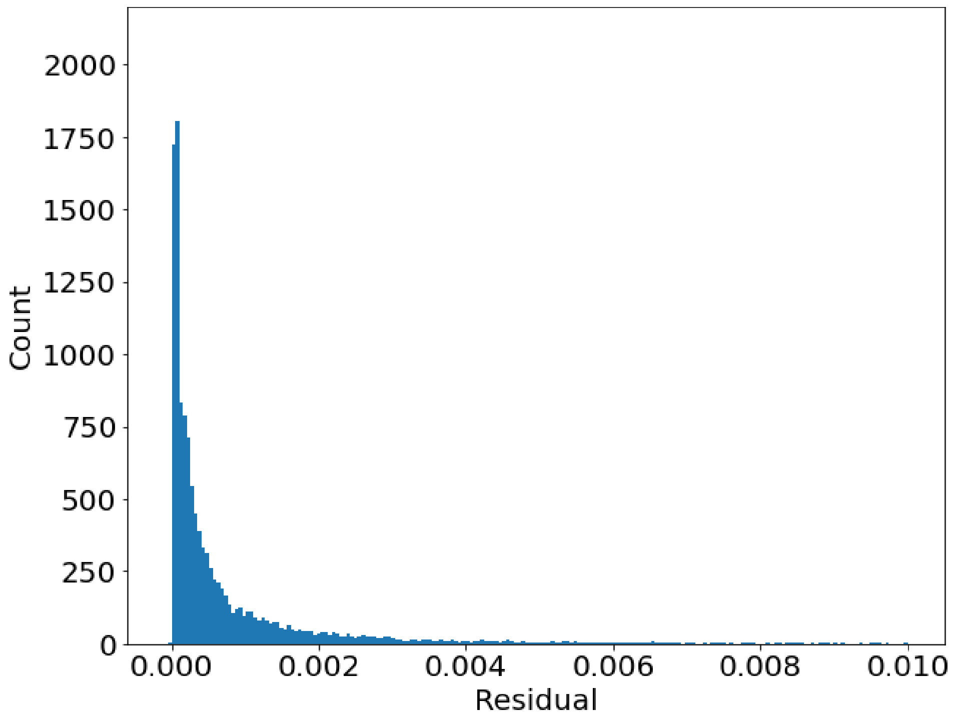


Fig. 38.: Residual histogram after Traffic Aware training training.

lower values indicate better performance. Compared with the EKFNet proposed in our previous work [7], which only trains Q, and R in the EKF, the TrafficEKKF leads to a better performance by using the extra traffic information. Further, it is clear that the TrafficEFK outperforms both the EKTNet and the manually tuned EKF no matter which loss function has been used for training.

5.5 Conclusion

In this chapter , I proposed an traffic aware EKF called TrafficEKF, which not only considers the vehicle's dynamic state but also its surrounding environment. By using deep learning and the EKFNet techniques, the TrafficEKF framework can be trained to learn from the rasterized image and the past object state estimates, for predicting the vehicle control inputs, and for optimizing the process and measurement

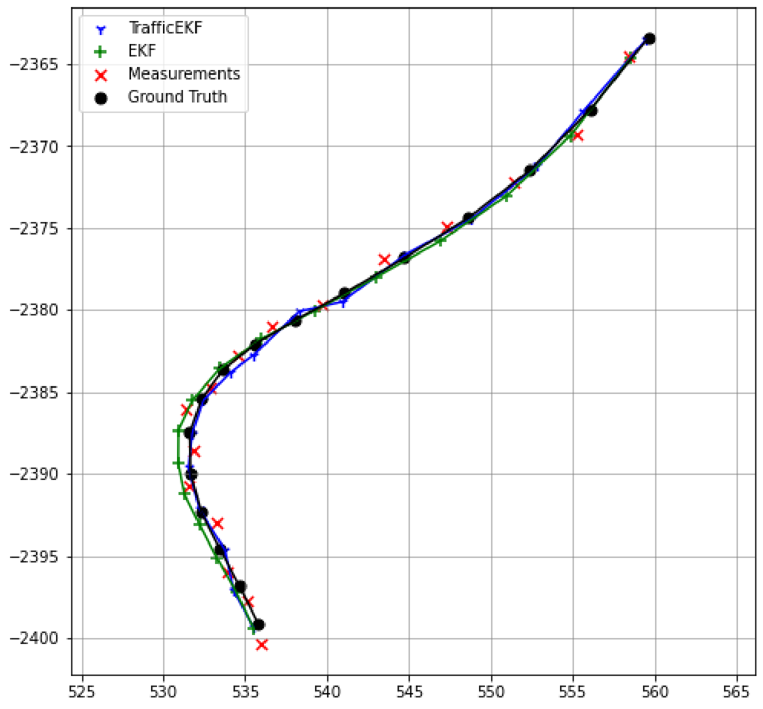


Fig. 39.: Road shape effect.

noise covariance matrices used by the EKF. Numerical results show that the proposed algorithm significantly improves the vehicle tracking performance, in comparison to both a manually tuned EKF and an EKFNet without environment information. For future work, I can explore the graph neural network to model the relationship among different vehicles, and different motion models for the tracked vehicles.

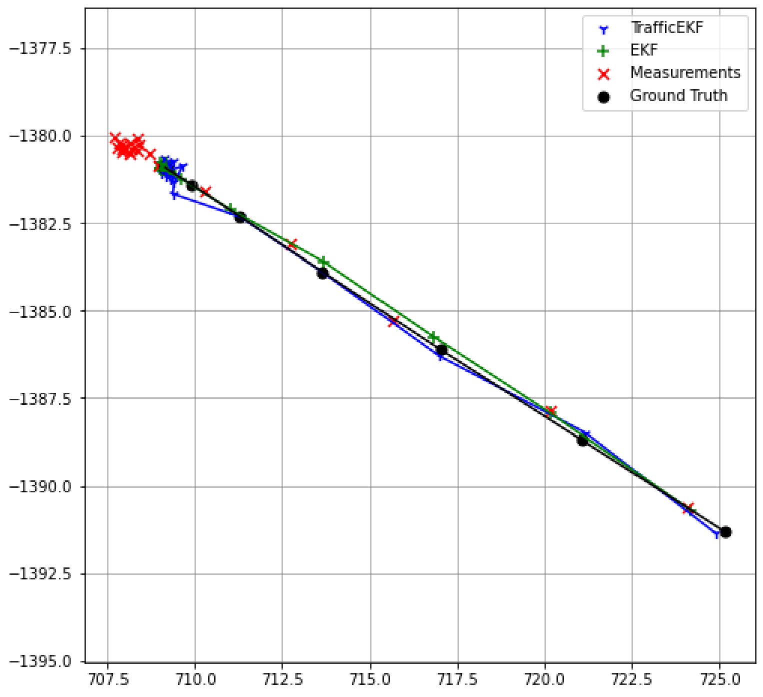


Fig. 40.: Red traffic light ahead.

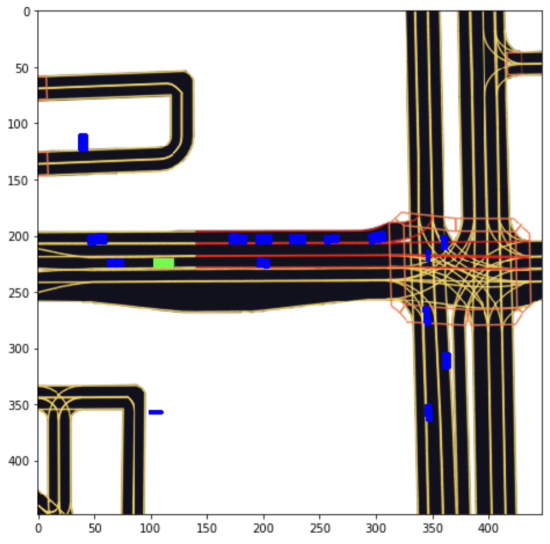


Fig. 41.: Vehicle slows down.

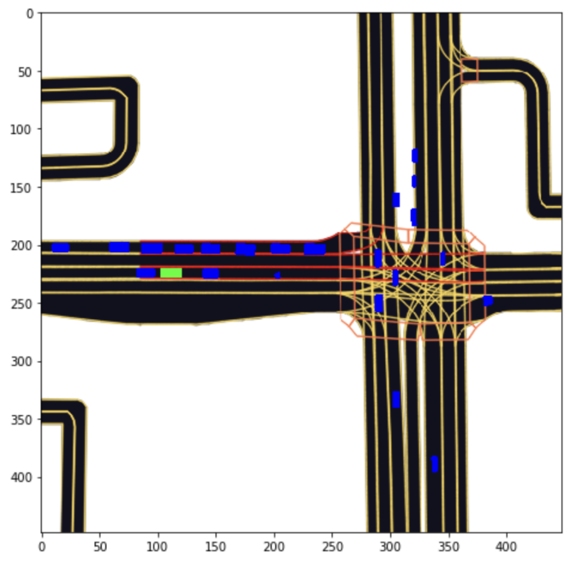


Fig. 42.: Vehicle stops, braking in front of red traffic light.

Table 9.: Testing results. -LogL: negative log-likelihood.

Objective Function	RMSE (GT)	-LogL (GT)
TrafficEKF L_{RMS}	**0.2320**	-0.3901
TrafficEKF L_{Like}	0.2843	**-1.5313**
EKFNet	0.3843	-0.3323
Manually tuned EKF	0.8231	1.6732

CHAPTER 6

UNCERTAINTY AWARE EKF: A TRACKING FILTER UNDERSTAND THE LIDAR MEASUREMENT UNCERTAINTY

The goal of this chapter is to develop an extended Kalman filter (EKF) framework called uncertainty aware EKF (UA-EKF) to improve the vehicle tracking accuracy for autonomous vehicles. The proposed framework can learn and estimate the uncertainty associated with the measurements provided by a LiDAR-based vehicle detection algorithm. The UA-EKF has two major parts: one has the ability to estimate the state-dependent measurement noise's statistics for LiDAR object detections, another is to create multiple-hypothesis measurements based on the detected vehicle's heading. The measurement uncertainties are learned based on the EKFNet, which is an algorithm that can learn the system noise covariance from measurement data. Both of the two major parts are used to compensate for the physical limitations of the LiDAR measurements. I also give a detailed analysis of the measurement uncertainty, and the methods to improve tracking performance during filtering. The obtained results on the nuScenes datasets show that estimating the measurement uncertainty is an efficient solution for tracking the vehicle based on LiDAR detections.

6.1 Introduction

Estimating the dynamic state and shape of a vehicle is a crucial aspect of autonomous vehicles (AV). The goal is to estimate the location, orientation, and shape of the vehicle in the surrounding environment. The results can be used for forecasting the other vehicle's motion and helping the AV to make future control decisions.

Bayesian based nonlinear filters are usually used as a vehicle state estimation algorithm. Popular nonlinear filters for state estimation include the extended Kalman filter (EKF) [9, 45], the unscented Kalman filter (UKF) [46], and the particle filter (PF) [102]. Extended Kalman Filter (EKF) is a popular nonlinear filter, which is often used as a tracking algorithm that can fuse an appropriate motion model and measurements [9]. For this chapter, I track the vehicle based on the measurements from the LiDAR vehicle detections algorithm [103].

6.1.1 LiDAR Detection

LiDAR points cloud is necessary but also very challenging for object detection [104, 2, 105, 106, 107, 108]. Compare with the visual objection detection and tracking algorithms [5, 109, 19, 32] it has its advantages, it can detect the object in 3D space by knowing not only the location but also the shape of the objects. For example, VoxelNet [107] uses a PointNet [107] for each voxel to generate feature representations with 3D sparse convolutions [110] then using 2D convolutions generates object detections. PointPillars [2] change the voxel with a pillar to improve backbone efficiency, which assumes only one object on each map location. SECOND [106] simplifies the VoxelNet and speeds up 3D convolutions. The detector I used in this chapter, CenterPoint [103] has two steps, the first step is to roughly locate the center of the object then the second step uses the regression head to get the refinement shape, location, and heading. Even though some of the detection claim they also can track the objects, still most of the methods only apply a 2D assignment and associate the closest detections from the last detection results. In this work, I am using the detection result for the CenterPoint [103] for vehicle tracking.

6.1.2 LiDAR Tracking

The LiDAR-based vehicle tracking has two categories. One is based on the LiDAR object detection algorithms and assumes the object is a single point target [111, 81, 82, 73], another is using the raw point cloud with extended object assumptions [83, 85, 42]. The difference between the single point target and the extended target is in the measurement model. One assumes only one detection point generated from the target, another uses multiple detection points as detections.

For the first category, the detection algorithm can use the point cloud to estimate the states either based on L-shape fitting algorithms [111, 84] or deep learning-based object detections algorithms [81, 82, 103].

In the second category, the object is assumed to be an extended target, which can generate multiple detections [83, 85, 42, 84, 33]. Usually, the object has been assumed to have a special shape, such as a rectangle [84], or an ellipse [42, 85]. Based on these assumptions, the object's state can be estimated by using multiple measurements points.

However, in most of the models, it is assumed that the LiDAR points follow the known Gaussian/Gaussian mixture model, which is not realistic in practice.

Some of the LiDAR-based tracking algorithms assume the measurement noise covariance is constant all the time [111, 81, 82]. From our analysis, the LiDAR measurement uncertainties can be estimated from the distance and bearing to the ego vehicle and the number of detections points. Also, in the literature, it was observed that the vehicle heading estimated by the detection algorithm is sometimes 180°off from the real heading. This problem was solved by simply rotating the heading towards the track. However, the rotated heading could be the first several measurements. In this case, the filter can easily lose the track of the object [81, 82]. In this chapter, I will provide an in-depth analysis on why this happens and how to handle it with multiple measurements hypotheses.

6.1.3 Contributions

In this chapter, I use outputs from a deep learning-based object detection algorithm, as the measurements for the tracker. The state-dependent measurement noise statistics will be learned by using the EKFNet [7], which is an algorithm that can learn the system noise covariance from data. For simplicity, I only use 2D tracking instead of 3D tracking by ignoring the distance from the ground to detection and the height of the vehicle. I more care about the 2D position, shape, and dynamic, because in most cases, there will only be 1 vehicle in a particular location.

In summary, the new contributions in this chapter are:

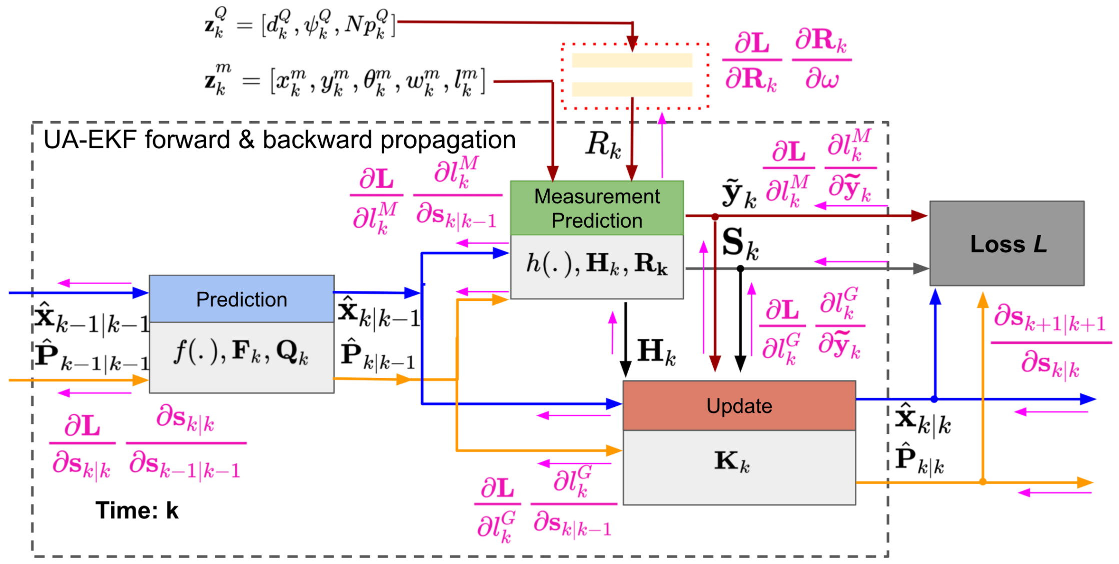
- I construct a new framework called Uncertainty aware EKF (UA-EKF) to track the vehicle's state based on LiDAR detection by incorporating its uncommon measurement uncertainties.
- I provide a detailed analysis of the physical limitation of LiDAR-based measurements.
- I develop a method that can learn the state-dependent measurement uncertainties with the ground truth.
- I provide different loss functions for different tracking objectives.

The rest of the chapter is organized as follows. Section 6.2, discussed some preliminaries knowledge. Section 6.3 included in detail what is the LiDAR physical limitation and its measurement uncertainties. Section 6.4 include the proposed method the UA-EKF framework. Section 6.5 is the experiment for the proposed framework with real LiDAR measurements.

6.2 Preliminaries

In this section, I discussed some preliminaries, which includes some background knowledge of EKF, state transition, and measurement model for vehicle tracking and output for LiDAR detection algorithm.

6.2.1 Extended Kalman Filter



$\mathbf{z}_k^Q = [d_k^Q, \psi_k^Q, Np_k^Q]$
$\mathbf{z}_k^m = [x_k^m, y_k^m, \theta_k^m, w_k^m, l_k^m]$
$\frac{\partial \mathbf{L}}{\partial \mathbf{R}_k} \frac{\partial \mathbf{R}_k}{\partial \omega}$
UA-EKF forward & backward propagation
R_k
$\frac{\partial \mathbf{L}}{\partial l_k^M} \frac{\partial l_k^M}{\partial \mathbf{s}_{k|k-1}}$
Measurement Prediction
$h(.), \mathbf{H}_k, \mathbf{R_k}$
$\tilde{\mathbf{y}}_k$
$\frac{\partial \mathbf{L}}{\partial l_k^M} \frac{\partial l_k^M}{\partial \tilde{\mathbf{y}}_k}$
$\mathbf{S}_k$
Loss *L*
Prediction
$f(.), \mathbf{F}_k, \mathbf{Q}_k$
$\hat{\mathbf{x}}_{k-1|k-1}$
$\hat{\mathbf{P}}_{k-1|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
$\hat{\mathbf{P}}_{k|k-1}$
$\mathbf{H}_k$
$\frac{\partial \mathbf{L}}{\partial l_k^G} \frac{\partial l_k^G}{\partial \tilde{\mathbf{y}}_k}$
$\frac{\partial \mathbf{s}_{k+1|k+1}}{\partial \mathbf{s}_{k|k}}$
Update
$\mathbf{K}_k$
$\hat{\mathbf{x}}_{k|k}$
$\hat{\mathbf{P}}_{k|k}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{s}_{k|k}} \frac{\partial \mathbf{s}_{k|k}}{\partial \mathbf{s}_{k-1|k-1}}$
Time: k
$\frac{\partial \mathbf{L}}{\partial l_k^G} \frac{\partial l_k^G}{\partial \mathbf{s}_{k|k-1}}$

Fig. 43.: EKFNet diagram [7]. $\mathbf{s} = \{\hat{\mathbf{x}}, \mathbf{P}\}$. Right arrows: forward flow; pink left arrows: back propagation.

At time k, let us denote the vehicle's state as $\mathbf{x}_k$, and the measurement as $\mathbf{z}_k^m$.

The state transition and measurement models are provided as follows:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}) + \mathbf{v}_k \tag{6.1a}$$

$$\mathbf{z}_k^m = h(\mathbf{x}_k) + \mathbf{w}_k \tag{6.1b}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions, and cannot be used directly to update the covariances. Instead, the Jacobian matrices $\mathbf{F}_k$ and $\mathbf{H}_k$ are used. $\mathbf{v}_k$ and $\mathbf{w}_k$ are the process and measurement noises, which are assumed to be zero mean Gaussian random vectors with covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

In Fig. 43 the EKFNet for learning the measurement noise is shown, and the data flow to the right is the common EKF algorithm. The prediction module in Fig. 43 is used for state prediction:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}) \tag{6.2a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{6.2b}$$

The measurement prediction module is used to evaluate the measurement residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1}) \tag{6.3a}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{6.3b}$$

The update module in Fig. 43 provides the posterior state estimate. The Kalman gain K_k is calculated in (6.4a):

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{6.4a}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{6.4b}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{6.4c}$$

6.2.2 LiDAR Detection

The outputs of the LiDAR object detection algorithm, obtained from order-less point clouds, are bounding boxes $B_k = \{b_0^k, b_2^k, ..., b_{N_k}^k\}$, where the nth box $b_n^k = \{x_n, y_n, z_n, w_n, l_n, h_n, \alpha_n\}$, for $n \in [0, N_k]$. $\{x_n, y_n, z_n\}$ usually is the center location of the object, $\{w_n, l_n, h_n\}$ stands for the width, length, and the height, and α_n is the vehicle heading. For simplicity, In this chapter I assume that the vehicle is driving on a 2D plane. So I ignore $\{z_n, h_n\}$ by assuming only one vehicle in each map location. Also, I focus on single object tracking, and only care about the detection which is close to the tracked vehicle like [81].

6.2.3 Motion and Measurement models

This section discussed the motion and measurement model used in the tracking filer.

6.2.3.1 Motion Model

Motion models have been used to improve the accuracy and stability of state estimation for vehicle applications such as vehicle tracking applications or navigation. The vehicle is assumed to comply with a certain motion model which describes its dynamic behavior; the choice of model complexity greatly influences the state estimation performance.

In this work, I use a motion model that closely represents the vehicle motion by the application of nonholonomic constraints as proposed in [73, 74, 75, 76]. As shown in Fig. 44 (1), the vehicle state is:

$$\mathbf{x}_k = [x_k, y_k, \theta_k, v_k, \phi_k, w_k, l_k]^T \tag{6.5}$$

where x_k and y_k are the rear center coordinates, θ_k is the global vehicle heading, v_k is the linear velocity, ϕ_k is the steering angle, and w_l and l_k are the widths and the length of the vehicle respectively. r is the ratio between the distance from the rear axle to the front of the vehicle and the vehicle length l_k. For simplicity, I assume $r = 0.7$ for all the vehicles. I assume that the vehicle is a front-wheel drive, and the motion model $f(\cdot)$ governing how the state transitions [76, 73] is provided below:

$$x_{k+1} = x_k + v_k \times \cos\theta_k \times \cos\phi_k \times \Delta t \tag{6.6a}$$

$$y_{k+1} = y_k + v_k \times \sin\theta_k \times \cos\phi_k \times \Delta t \tag{6.6b}$$

$$\theta_{k+1} = \theta_k + v_k \times \sin\phi_k \times \Delta t/(l_k \times r) \tag{6.6c}$$

$$v_{k+1} = v_k \tag{6.6d}$$

$$\phi_{k+1} = \phi_k \tag{6.6e}$$

$$w_{k+1} = w_k \tag{6.6f}$$

$$l_{k+1} = l_k \tag{6.6g}$$

In the motion model, I assume the steering angle ϕ_k and velocity v_k remain constant between each updating. This model is usually called Constant Turn Rate and Velocity (CSRV) model [77]. The main feature of the kinematic model of a car like a vehicle is the presence of two nonholonomic constraints 6.6 refraining the front/rear wheels from moving slide ways .

6.2.3.2 Measurement Model

By using the detection box b_n^k, the measurements are the center position x_k^m, y_k^m , heading angle θ_k^m and the shape w_k^m, l_k^m of the vehicle:

$$\mathbf{z}_k^m = [x_k^m, y_k^m, \theta_k^m, w_k^m, l_k^m]^T + \mathbf{w}_k \tag{6.7}$$

The measurement model $h(\cdot)$ shown in Fig. 45 is:

$$x_k^m = x_k + (r - 0.5) \times l_k \times \cos\theta_k \tag{6.8a}$$

$$y_k^m = y_k + (r - 0.5) \times l_k \times \sin\theta_k \tag{6.8b}$$

$$\theta_k^m = \theta_k \tag{6.8c}$$

$$w_k^m = w_k \tag{6.8d}$$

$$l_k^m = l_k \tag{6.8e}$$

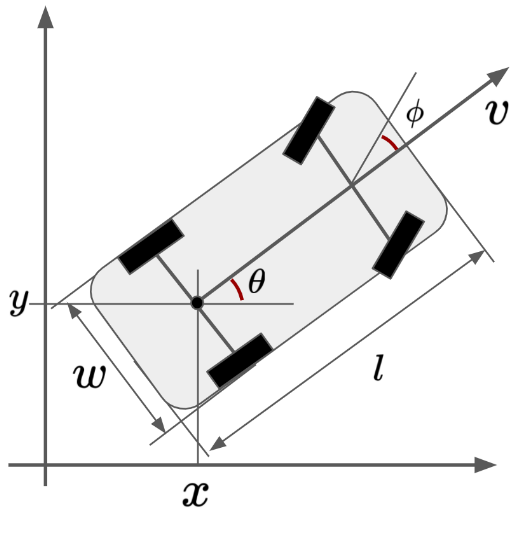


Fig. 44.: Vehicle Motion Model.

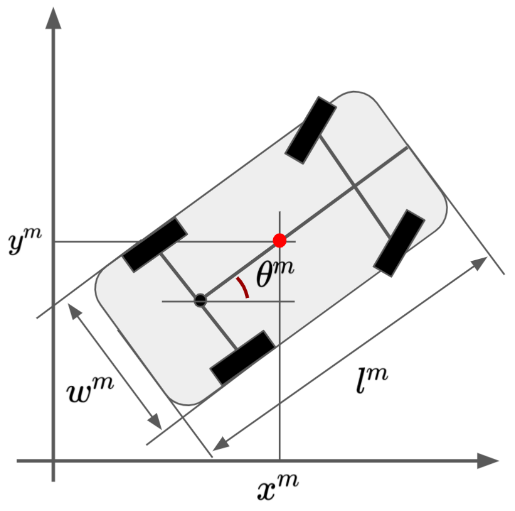


Fig. 45.: Vehicle Measurement Model.

$$\mathbf{G}(\mathbf{x}_{k|k}) = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 \cos(\phi_{k|k}) & \frac{1}{2}(\Delta t)^2 \sin(\phi_{k|k}) & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}(\Delta t)^2 & 0 & \Delta t & 0 & 0 \end{bmatrix}^T \tag{6.11}$$

6.2.3.3 Process and Measurement Noise

The state dependent process noise covariance matrix is given in (6.9), where σ_A is the maximum acceleration and σ_{SR} is the maximum steering rate. The definition of $\mathbf{G}(\cdot)$ is provided in (6.11), which is the dynamic model for acceleration and steering rate [7]. The measurement noise covariance is provided in (6.10).

$$\mathbf{Q}_k(\mathbf{x}_{k|k}) = \mathbf{G}(\mathbf{x}_{k|k})\mathbf{diag}(\sigma_A^2, \sigma_{SR}^2)\mathbf{G}(\mathbf{x}_{k|k})^T + \mathbf{diag}(\sigma_x^2, \sigma_y^2, \sigma_\theta^2, \sigma_v^2, \sigma_\phi^2, \sigma_w^2, \sigma_l^2) \tag{6.9}$$

$$\mathbf{R}_k = \mathbf{diag}(\sigma_{z_x}^2, \sigma_{z_y}^2, \sigma_{z_\theta}^2, \sigma_{z_l}^2, \sigma_{z_l}^2) \tag{6.10}$$

In most of the existing work, the measurement noise covariance $\mathbf{R}_k$ was assumed to be a constant matrix. However, from our analysis, $\mathbf{R}_k$ is state-dependent because of the physical limitation of the LiDAR measurements. In Section 6.3 I discussed in detail the dependency, and the method to estimate the measurement noise.

6.3 Uncertainty with Lidar Measurement

LiDAR is a measurement technique that uses light emitted from a sensor to measure the 3D location of a target. AVs may use LiDAR for obstacle detection and avoidance to navigate safely through environments. For most of the sensors, the measurement uncertainties can be assumed to be a constant value in EKF. However, due to the physical limitation of the LiDAR sensor, the measurement uncertainties are uncommon. This section is going to give a deep analysis of the physical limitation of the LiDAR measurement, both the 3D location and the heading. And how to estimate those uncertainties from measurements.

6.3.1 State Depended Uncertainty

The point cloud created by the LiDAR sensor measures the time-of-flight for each laser beam from a fixed number of stacked laser-detector by rotating the moving part 360°at a finite angular resolution [112]. A fixed number of measurement points have been created due to those limitations. The more stacked laser-detector and smaller the angular resolution the more measurement points could be created, which also leads to a high computation burden. In this chapter, I assume that the object is a vehicle in the surrounding environment, which can usually be represented as a bounding box for the detection algorithm. Because of the distance from the tracked vehicle to the ego vehicle and the tracked vehicle's heading, there are two unique uncertainties have been created.

6.3.1.1 Distance Dependent Uncertainties

I denote the finite LiDAR angle resolution as α_L, which makes the LiDAR measurement accuracy distance-dependent. For example in Fig. 46, which is a bird-eye-view (BEV) of the target, the point clouds are represented by red dots. Due to the finite angular resolution, when the target vehicle is far away, the distance between two adjacent measurement points becomes larger than that in a close-by target. In Fig. 47 , 48 and 49, the histograms of the measurement error for x, are provided for three scenarios with different ranges of distance between the tracked vehicle and the ego vehicle. It is can be seen that the measurement error is approximately Gaussian, and its variance increases as the distance become larger. The same effect can be observed for other measurements given by the detection algorithm. By checking the histogram of the Root Mean Square Error (EMSE) for detection x with the ground truth. I can easily find out the error distributed as a Gaussian and the variance increase with the distance. The same effect applied to other measurement states also.

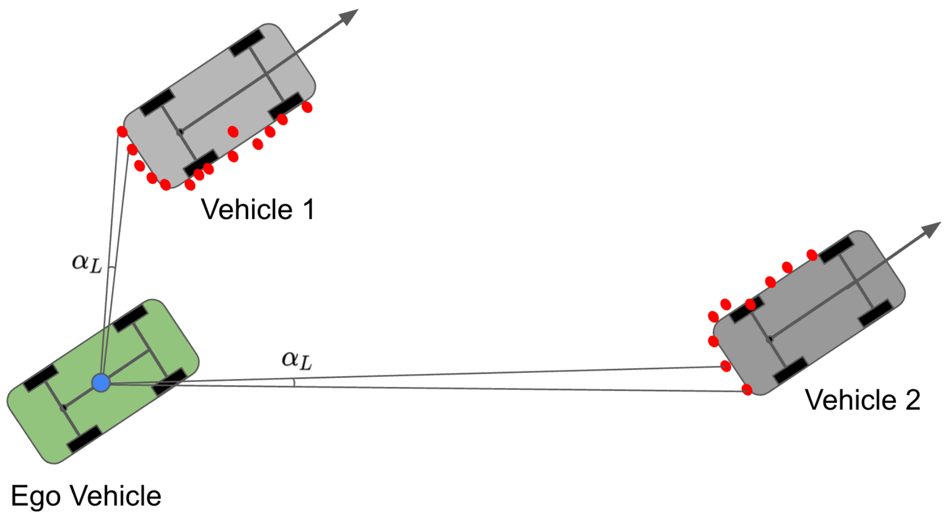


Fig. 46.: Finite angular resolution with different distance to ego vehicle.

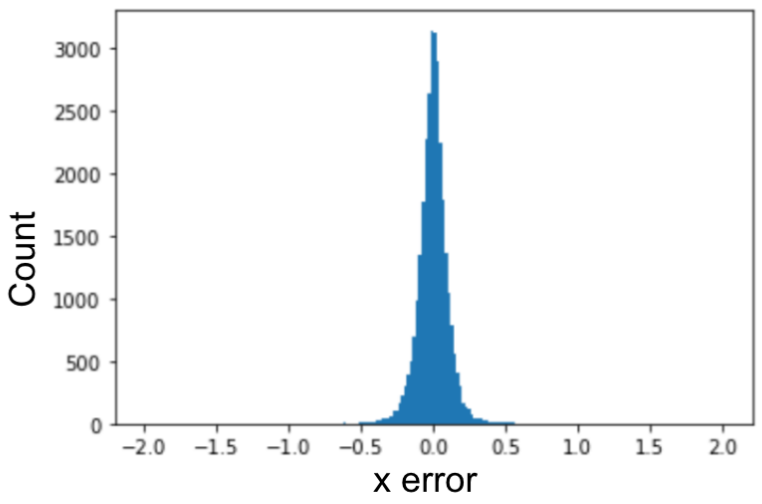


Fig. 47.: Distance $< 10m$, $\sigma_x^2 = 0.0115$.

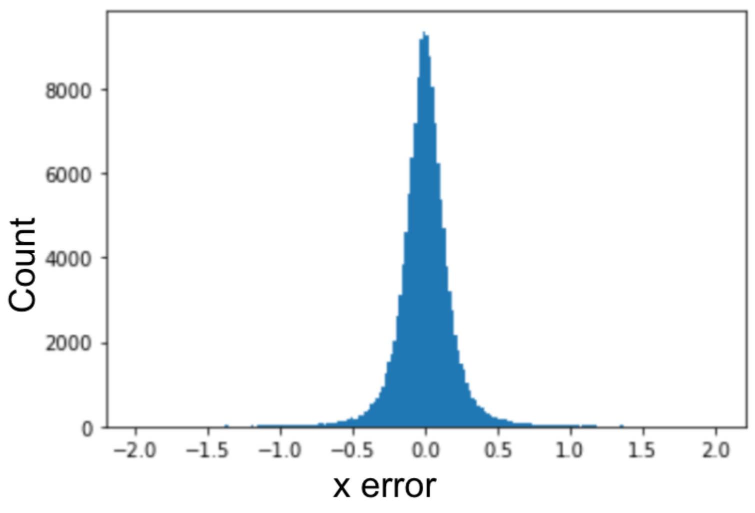


Fig. 48.: Distance $\in [10m, 30m]$, $\sigma_x^2 = 0.0466$.

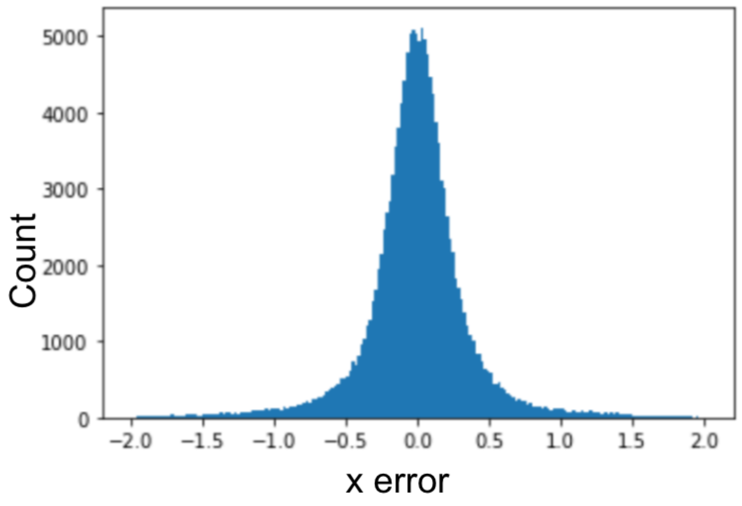


Fig. 49.: Distance $> 30m$, $\sigma_x^2 = 0.1343$.

6.3.1.2 Vehicle Heading Dependent Uncertainties

Because of the finite measurement angle resolution, the uncertainties are also dependent on the target's head with respect to the line of sight (LOS) direction between the target and ego vehicles. For example in Fig. 50, there are two different target vehicles at two different positions. Because all the measurement points are generated from the visible surface, not only the distance but also the relative heading creates the difference when measuring the length and width of the target vehicle. In this figure, I can define the angle of view for the width as β_W^1 and length as β_L^1 for the lth vehicle. Because of the finite measurement angle resolution, the measurement uncertainties are larger when this angle is small, because fewer points will be reflected from this side of the vehicle. In the extreme case, if a vehicle is just in front of the ego vehicle, it is impossible to measure the length which is not visible, but it is easy to measure the width.

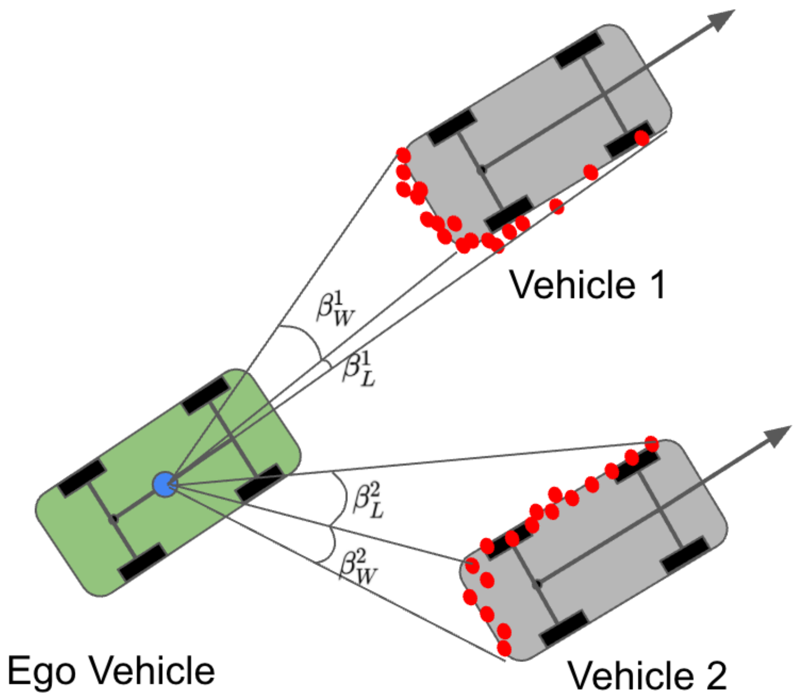


Fig. 50.: Finite angular resolution with different distance and angle to ego vehicle.

6.3.1.3 Measurement Noise Covariance

The measurement noise covariance for a target vehicle is a function of: 1. its distance to the ego vehicle d_k^Q. 2. its heading relative to the LOS direction ϕ_k^Q. 3. the number of LiDAR points on the vehicle Np_k^Q. In Fig. 52, all the inputs for estimating the state-dependent uncertainties (measurement noise covariance) are illustrated. By the independent noise assumption/approximation, I only consider the diagonal terms in $\mathbf{R}_k$, which are shown in (6.12). So, our goal is to construct a state-dependent function:

$$\mathbf{R}_k = \mathbf{R}^m + g(d_k^Q, \phi_k^Q, Np_k^Q) \tag{6.12}$$

where $\mathbf{R}^m$ is the common measurement noise which is measured from the detections and ground truths [81], and is a fixed matrix. The output from $g(\cdot)$ is the state-dependent component that can change $\mathbf{R}^m$. $g(\cdot)$ can be constructed by a fully connected neural network, and the network structure is shown in Fig. 51. The network has three layers and the first two layers are activated by ReLU, and the final layer is activated by a weighted Tanh function [15].

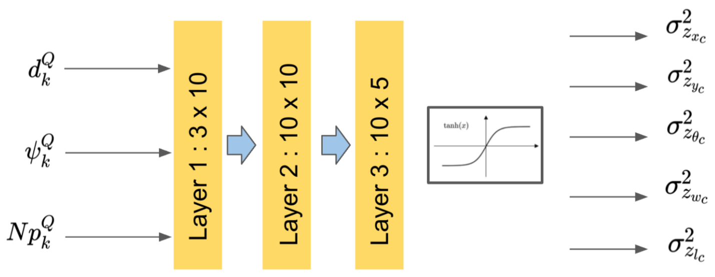


Fig. 51.: Uncertainty Net work. Inputs are the states between the ego vehicle and number of LiDAR points. Outputs are the correction for the measurements uncertainties.

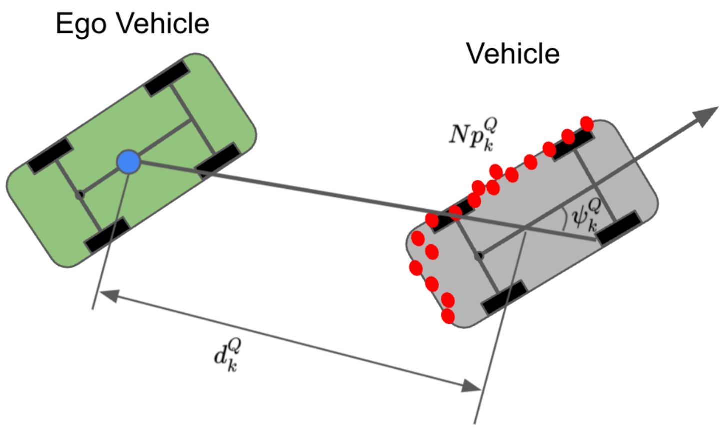


Fig. 52.: Inputs to estimate the state dependent measurement uncertainties.

6.3.2 Heading Measurement Uncertainties

The heading of the target is also estimated from a group of points. However, in many detection algorithms, the estimated heading may be in the wrong direction, e.g., 180°off from the ground truth. In Fig. 53, the histogram of the heading error is shown, where the y-axis represents the log-based distribution, and the x-axis is the heading error in radians. There are totally four peaks in range $[-\pi, \pi)$, which are at $[0°, 90°, -90°, 180°]$ respectively with different probabilities. Because the object is detected from a group of points, and the object is usually assumed to be a rectangle box, the algorithm could make mistakes by deciding the wrong heading direction as illustrated in Fig. 55, 56, 57,58. This is why the histogram of the heading error has four different modes. I can manually select the heading which is the closest to the ground truth from those four possible solutions, and the results are shown in Fig. 54. In this case, the error has a Gaussian-like unimodal distribution. By compensating for this unique uncertainty, our UA-EKF created 4 different hypothesis tracks initially. Then during the tracking, the measurement also generates 4 different pseudo measurements for each hypothesis. Each hypothesis will select the pseudo detections based on the measurement likelihood. Please check Section 6.4.1.4 for details.

6.4 Methodology

The goal of the proposed method is to find a method to compensate for the unique uncertainties mentioned in Section 6.3. This section begins with the UA-EKF framework, and the following is the loss function that is used for learning the uncertainty Net.

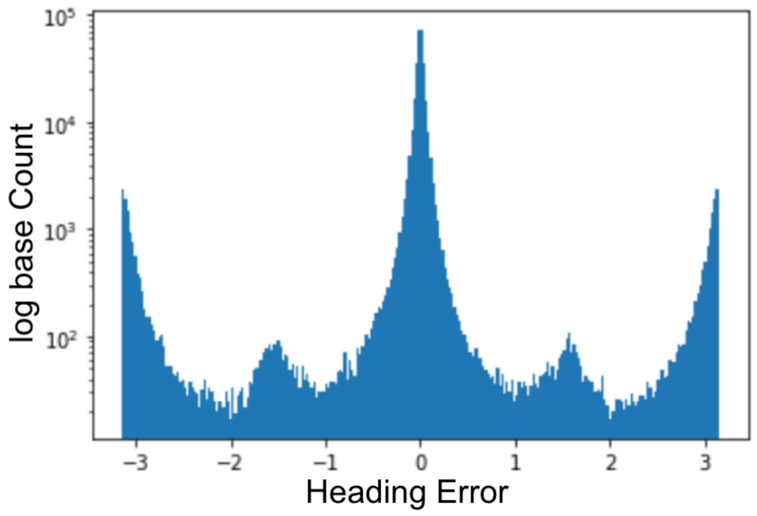


Fig. 53.: Heading Error Without Correction.

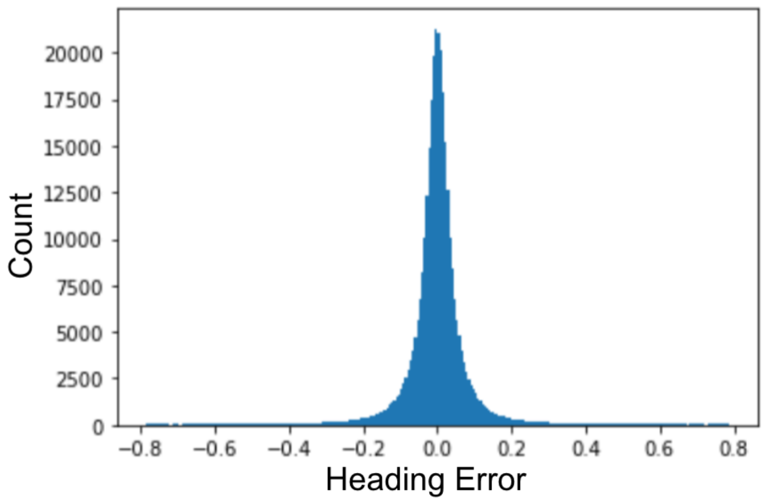


Fig. 54.: Heading Error With Correction.

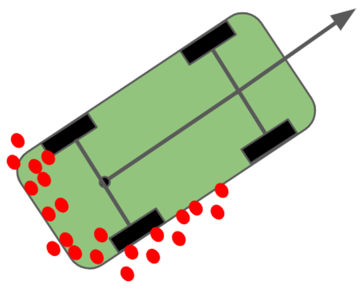


Fig. 55.: Heading 1.

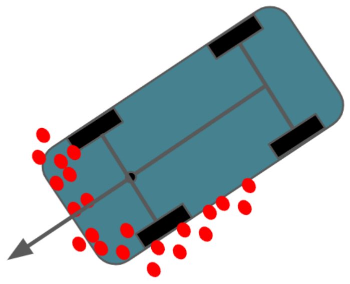


Fig. 56.: Heading 2.

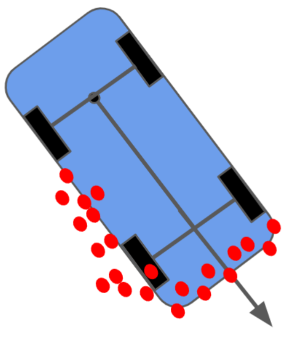


Fig. 57.: Heading 3.

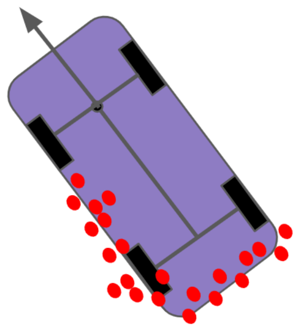


Fig. 58.: Heading 4.

6.4.1 Uncertainty Aware Extended Kalman Filter

The proposed method UA-EKF has two main parts: one has the ability to estimate the state-dependent measurement noise covariance based on LiDAR object detections; another is to create multiple hypotheses and pseudo measurements based on the detected vehicle heading. The estimated measurement uncertainties are learned based on the EKFNet.

6.4.1.1 Learning Measurement Uncertainties

The measurement noise covariance $\mathbf{R}_k$ is estimated by using (6.12) , and $g(\cdot)$ is learned by using the modified EKFNet. The EKFNet forward and backward operations are illustrated in Fig. 43. The forward operation for EKFNet is the same as the EKF, which is used for tracking. The backward operation is to learn the unknown parameters in the network which is shown in Fig. 51 and denoted as $\boldsymbol{\psi}$. Different from existing methods, the network $g(\cdot)$ predicts the measurement uncertainties by using $\mathbf{z}_k^Q$.

To train the network $f(\cdot)$, I use the backpropagation through time (BPTT) method to calculate the gradient for the loss function with respect to the training parameter. BPTT is a gradient-based method for training a directed computation graph based on time series [7, 68, 15, 70]. The backpropagation flow is shown in Fig. 43. The pink equations and left arrows indicate the backward gradient flows. At each step k, the gradients are derived from: (1) error made by loss function L, (2) gradients from time step $k+1$. Detailed derivations for backpropagation are skipped in this chapter for brevity. Details of backpropagation for the EKF part can be found in our previous work [7]. Finally, the gradient descent optimization method is used to update the parameters [70, 15].

6.4.1.2 Multiple-Hypothesis Tracking

Due to the heading estimation uncertainty discussed in Section 6.3.2, a modified Track Oriented Multiple-Hypothesis Tracking (TO-MHT) algorithm [9, 45], has been used for single vehicle tracking. The hypothesis space can be summarized as:

$$\left\{l^{h_k}, P^{h_k}_{k|k}(\mathbf{x}_k)\right\}^{\mathcal{H}_k}_{h_k=1} \tag{6.13}$$

where l^{h_k} is the log weight for Track Hypothesis h_k, $P^{h_k}_{k|k}(\mathbf{x}_k)$ is the posterior probability density function (PDF) of $\mathbf{x}_k$ under Track Hypothesis h_k given all the measurements up to time k ($\mathbf{z}_{1:k}$), and $\mathcal{H}_k$ is the total number of hypotheses at time k. In the beginning, four hypotheses have been created and their weights are initialized based the probabilities of the four heading estimates. The log weight for the h_kth Track Hypothesis is updated as:

$$\tilde{l}^{h_k} = l^{h_{k-1}} + max\left\{\log P\left(\mathbf{z}^{m_i}_k | P^{h_k}_{k|k-1}\left(\mathbf{x}_k\right)\right)\right\}^4_{i=1} \tag{6.14}$$

where $\tilde{l}^{h_k}$ is the un-normalized log weight, $\{\mathbf{z}^{m_i}_k\}^4_{i=1}$ are the pseudo measurements by rotating θ^m_k four times with a 90°increment, $P^{h_k}_{k|k-1}\left(\mathbf{x}_k\right)$ is the predicted state PDF for Track h_k. The physical meaning of the second term is to select the most likely pseudo measurement to update the vehicle state. After calculating all the un-normalized log weights for all the hypotheses, the weights need to be normalized.

During the tracking process, some low-weight hypotheses will be eliminated from the hypothesis space according to a predefined threshold. The track hypothesis with the highest weight is used as the tracker output.

6.4.1.3 Multiple hypothesis tracking

6.4.1.4 Multiple-Hypothesis Tracking

Due to the heading estimation uncertainty discussed in Section 6.3.2, a modified Track Oriented Multiple-Hypothesis Tracking (TO-MHT) algorithm [9, 45], has been used for single vehicle tracking. The hypothesis space can be summarized as:

$$\left\{ l^{h_k}, P^{h_k}_{k|k}(\mathbf{x}_k) \right\}_{h_k=1}^{\mathcal{H}_k} \tag{6.15}$$

where l^{h_k} is the log weight for Track Hypothesis h_k, $P^{h_k}_{k|k}(\mathbf{x}_k)$ is the posterior probability density function (PDF) of $\mathbf{x}_k$ under Track Hypothesis h_k given all the measurements up to time k ($\mathbf{z}_{1:k}$), and $\mathcal{H}_k$ is the total number of hypotheses at time k. In the beginning, four hypotheses have been created and their weights are initialized based on the probabilities of the four heading estimates. The log weight for the h_kth Track Hypothesis is updated as:

$$\tilde{l}^{h_k} = l^{h_{k-1}} + max \left\{ \log P \left(\mathbf{z}^{m_i}_k | P^{h_k}_{k|k-1} \left(\mathbf{x}_k \right) \right) \right\}_{i=1}^{4} \tag{6.16}$$

where $\tilde{l}^{h_k}$ is the un-normalized log weight, $\{\mathbf{z}^{m_i}_k\}_{i=1}^4$ are the pseudo measurements by rotating θ^m_k four times with a 90°increment, $P^{h_k}_{k|k-1}\left(\mathbf{x}_k\right)$ is the predicted state PDF for Track h_k. The physical meaning of the second term is to select the most likely pseudo measurement to update the vehicle state. After calculating all the un-normalized log weights for all the hypotheses, the weights need to be normalized.

During the tracking process, some low-weight hypotheses will be eliminated from the hypothesis space according to a predefined threshold. The track hypothesis with the highest weight is used as the tracker output.

6.4.2 Loss Function

In this subsection, I introduce the loss function used for training the EKFNet. First, let us consider the l_2 norm of the estimation error, which is the difference between the ground truth $\mathbf{x}_k^G$ and $\hat{\mathbf{x}}_{k|k}$. The corresponding loss function over T time steps is:

$$L^G = \sum_{k=1}^{T} \|\mathbf{x}_k^G - \hat{\mathbf{x}}_{k|k}\|_2^2 \tag{6.17}$$

The measurement log likelihood can be used to improve the measurement association. At each time, the measurement residual $\tilde{\mathbf{y}}_k$ and its covariance $\mathbf{S}_k$ is calculated. The negative log-likelihood for measurement residuals is provided as:

$$\begin{aligned} L_L^M &= -\log\left(\prod_{k=1}^{T} p(\mathbf{z}_k^m|\mathbf{z}_{0:k-1}^m)\right) \\ &= -\sum_{k=1}^{T} \log p(\mathbf{z}_k^m|\mathbf{z}_{0:k-1}^m) \end{aligned} \tag{6.18}$$

The total loss function is a combination of the two loss functions introduced above:

$$L = \lambda_1 L_L^M + \lambda_2 L^G \tag{6.19}$$

where λ_1 and λ_2 are hyper parameters, which are set to 1 in this chapter.

6.5 Experiment

6.5.1 Datasets Creation and Implement Details

As no benchmark for this task is available so far, I construct a new dataset from the nuScenes dataset [93] and use the CenterPoint [103] as the detection algorithm. The CenterPoint gives multiple detections for multiple objects at each time frame. I select the closest detection to a particular target as its detection as proposed in [81]. Even though the vehicle is not detected on every frame, ™I only selects the tracks with gaps less than two seconds. This dataset is created for tracking.

The learning dataset is created based on the tracking dataset with corrected heading measurements, by selecting the closest heading from the pseudo measurements with ground truth. The ground truth state $\mathbf{x}_k^G$ is created by running the EKF smoother on the human-annotated ground truth data. The proposed method implemented with PyTorch [100] on a PC with an Intel Xeon, 32G RAM, Nvidia RTX 2080Ti.

6.5.2 Results

6.5.2.1 Training Results

The training is accomplished by the EKFNet, with the proposed loss function in Section 6.4.2. The training data sets have a total of 600 tracks, and the testing data sets have 200 tracks for testing the EKF's tracking performance by using the learned measurement covariance matrix. The testing metrics include the RMSE between the estimated state and the ground truth, the error between the predicted measurements and real measurements, and negative log-likelihoods for the state and measurements.

Figure 59 60 display the histogram for EKF without using the network, and Fig-

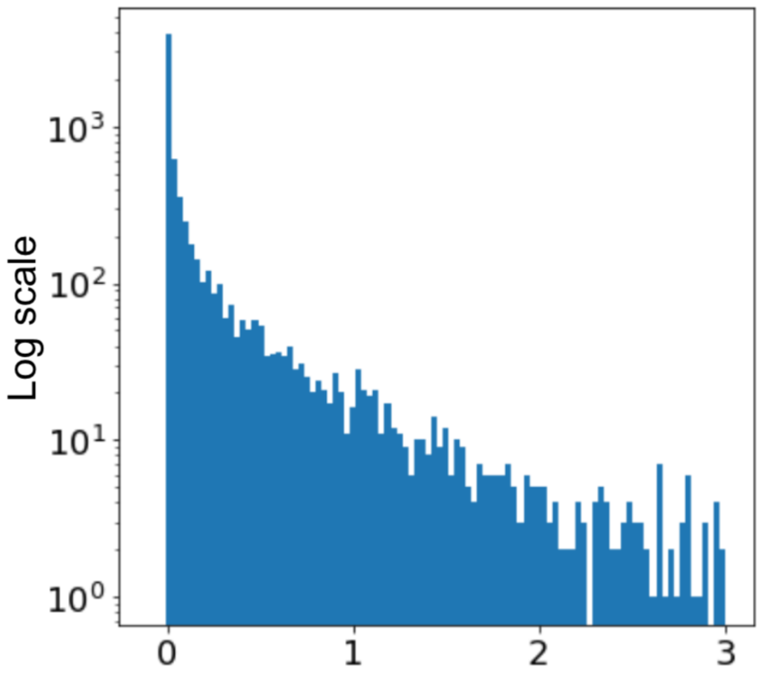


Fig. 59.: RMSE for states, without using the Uncertainty Net.

ure 59 60 are the results for using the learned state-dependent covariance matrix are shown respectively. From these figures, it is clear that by learning the measurement uncertainty, the proposed method significantly improves the tracking performance in terms of both the RMSE and measurement likelihood. In Table 6, the performances of the traditional EKF and EKF using the uncertainty Net are compared, in terms of all the different testing metrics. It is clear that in both the training and testing stages, the tracking performance has been significantly improved after learning the measurement uncertainty parameters.

6.5.2.2 Tracking Results

The tracking results are obtained by using the tracking dataset.Different from the training dataset, heading measurements are not corrected in the tracking dataset. One example is shown in Fig. 63 64, where the red x represents the first detection, and

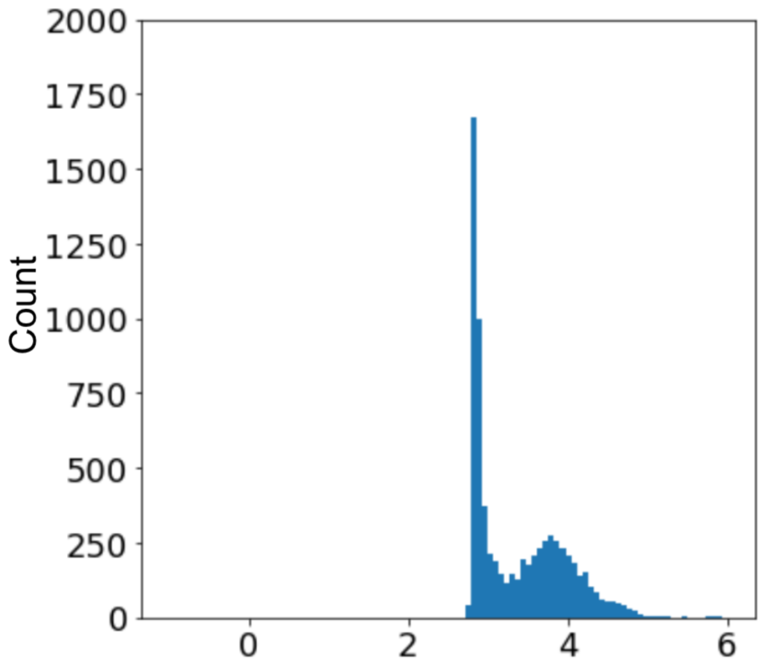


Fig. 60.: -LogL for measurements, without using the Uncertainty Net.

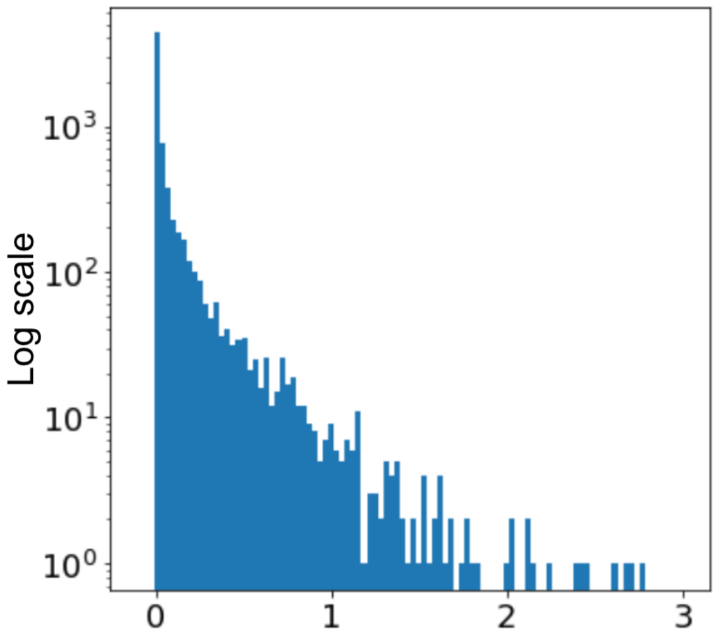


Fig. 61.: RMSE for states, using the Uncertainty Net.

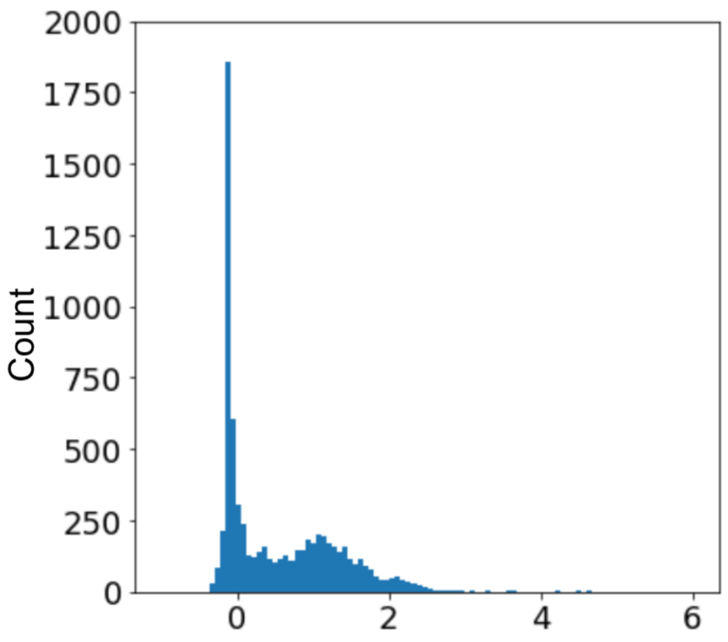


Fig. 62.: -LogL for measurements, using the Uncertainty Net.

green arrow represents the heading state. In this example, the detection algorithm makes a mistake for the first measurement. It takes the traditional EKF several time steps to converge to the right heading estimate. It also has large estimation errors in the mid of the track, when heading measurements provided by the detection algorithm are wrong. In comparison, the proposed method can keep the track of the vehicle after the second time steps by filtering out the unlikely hypotheses and correcting the wrong heading measurements, which leads to a much more accurate trajectory estimate. In Table 11, the performances of four tracking approaches, EKF, EKFNet, and UA-EKF without uncertainty Net, and UA-EKF. Compared with the manually tuned EKF, both the EKFNet and UA-EKF without UN improve the tracking performance incrementally. The UA-EKF, with the uncertainty Net's help, outperforms all the other tracking methods in terms of all the evaluation metrics.

Table 10.: Training results. -LogL: negative log-likelihood.

Metrics	RMSE (State)	RMSE (meas)	-LogL (State)	-LogL (meas)
Training (EKF)	0.2498	0.2681	-0.9254	3.4593
Testing (EKF)	0.2245	0.2277	-1.0154	3.3841
Training (EKF-Net)	0.1134	0.1626	-7.3652	0.6627
Testing (EKF-Net)	0.1009	0.1309	-7.5732	0.5587

6.6 Conclusion

In this research work, I proposed a framework called Uncertainty Aware EKF, which is used for vehicle tracking by understanding the uncommon measurement uncertainties from the LiDAR-based vehicle detections. The proposed method has two major components, one is estimating the LiDAR measurement noise, another is creating multiple hypotheses for the measurement heading. The obtained results on the nuScenes datasets show that understanding the measurement uncertainty is an efficient solution for tracking the vehicle based on the LiDAR detection.

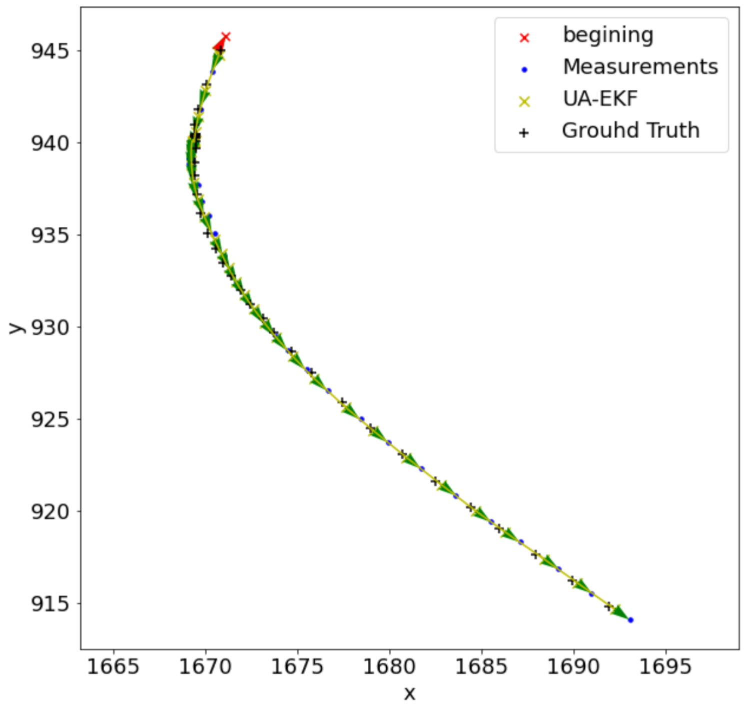


Fig. 63.: UA-EKF tracking result.

Table 11.: UA-EKF tracking result

Metrics	RMSE (State)	RMSE (meas)	-LogL (State)	-LogL (meas)
EKF	0.5932	0.6987	1.4424	4.9746
EKFNet	0.4424	0.4676	-0.4876	3.3876
UA-EKF(No Uncertainty Net)	0.2645	0.2887	-1.7864	1.6576
UA-EKF	0.1498	0.1743	-5.8476	0.7987

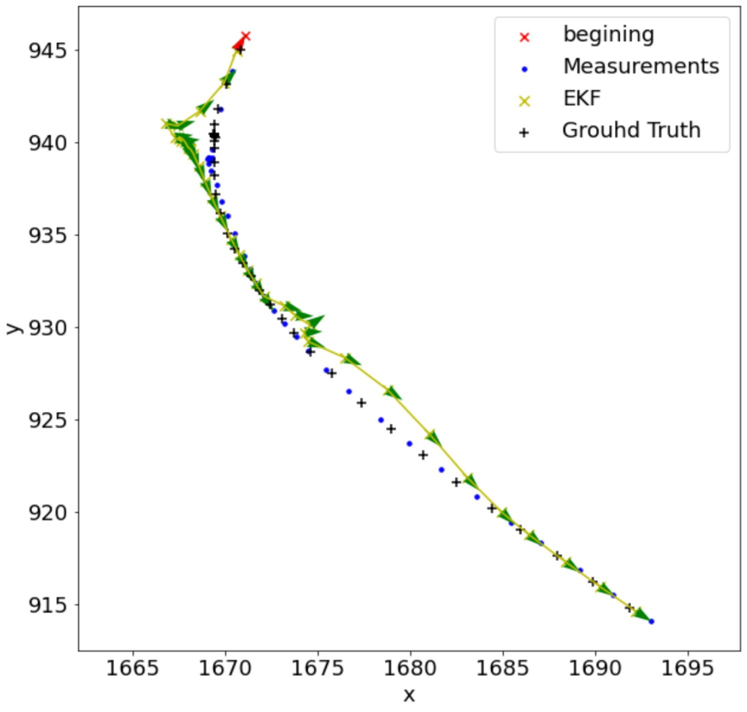


Fig. 64.: EKF tracking result.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this dissertation, I made two contributions to improve the performance of learning assisted tracking systems. One is to use deep learning techniques to track an object with unstructured measurement data. Another one is to use machine learning techniques to improve the performance of a tracking filter.

For the first aspect, visual object tracking was used as an example to demonstrate that the tracking system can track an object with unstructured measurements. For the semi-supervised visual object tracking method, I departed from the traditional fully convolutional Siamese network and developed a variational Siamese network that trains feature embedding through both supervised and unsupervised learning. The embedded features are represented by multivariate Gaussian distributions in a feature space, and the distance between two objects' features is measured by an information metric (Wasserstein distance). For the extended visual object tracking work, I modeled the visual object as an extended target. The estimates of both the kinematic states and shape parameters are updated based on multiple detection points. I also provided an analysis to explain why the single point target assumption is not sufficient. To the best of our knowledge, this is the first work where the visual object is modeled as an extended target and a closed-loop detection-tracking framework is proposed with the convolutional neural network as a detector.

The second aspect is how to use the data to improve the tracking algorithm. In this part, I used the EKF as the desired algorithm to be optimized. For EKFNet, I proposed a method that can fine-tune the EKF automatically with or without the ground truth data. For learning the best process and measurement noise covariances offline, I presented four different objective functions. The whole framework is trained using gradient descent, and the gradients are calculated based on BPTT. For the TrafficEKF, the proposed method not only considers the vehicle's dynamic state but also its surrounding environment. By using deep learning and the EKFNet techniques, the TrafficEKF framework can be trained to learn from the rasterized image and the past object state estimates, for predicting the vehicle control inputs, and for optimizing the process and measurement noise covariance matrices used by the EKF. For the uncertainty-aware EKF, I proposed a method that can estimate the measurement uncertainty, which is used for vehicle tracking by understanding the uncommon measurement uncertainties from the LiDAR-based vehicle detections. The proposed method has two major components: one is estimating the LiDAR measurement noise statistics, and another is creating multiple hypotheses for the heading measurements.

All the methods proposed in this dissertation are aimed to improve the tracking performance with the assistance of deep learning. Instead of complicated mathematical modeling of the system, I used a data-driven approach to "learn" the best parameters which provide better performance.

7.2 Future Work

Sensor fusion and tracking play an important role in autonomous systems, especially in the perception sub-systems. With deep learning techniques becoming mature and more and more sensors being used to sense the surrounding environment, the perception system faces new challenges. In this dissertation, several topics were discussed to help the traditional tracking system cope with these challenges. There is a lot of future work which can be done after this dissertation.

The first area the current work can be extended is multi-object tracking (MOT), which has not been touched in this dissertation. However, there is potential work that needs to be explored. Since the single tracker is the building block of an MOT tracker, what has been proposed in this dissertation can be used to develop an MOT tracker to improve its data association and tracking performances.

The second area is tracking aided detection. Currently, object detection algorithms only treat each measurement individually, and the previous detection results do not contribute to the current detection process. However, for most of the applications, for example, autonomous vehicles, the measurement data correlates from one frame to the next. Also, due to sensing limitations, a sensor may not "see" the object which is hidden or missed by a detector. The tracking system, which provides predictions for the future state and sensor measurements, can help improve a detector's performance.

The intelligent tracking system will shine its light on autonomous systems with the information provided by data. There could be great potentials for researchers to continue working on it.

REFERENCES

[1] Erik Bohnsack and Adam Lilja. “Multi-object tracking using either end-to-end deep learning or PMBM filtering”. MA thesis. 2019.

[2] Alex H Lang et al. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12697–12705.

[3] *VOLUNTARY SAFETY SELF-ASSESSMENT(VSSA)*. `https://motional.com/safety-the-driver-at-motional/`. Accessed: 2021-01-31.

[4] Michel Deudon. “Learning semantic similarity in a continuous space”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 986–997.

[5] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[6] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[7] Liang Xu and Ruixin Niu. “EKFNet: Learning System Noise Statistics from Measurement Data”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 4560–4564.

[8] Anton J Haug. “A tutorial on Bayesian estimation and tracking techniques applicable to nonlinear and non-Gaussian processes”. In: (2005).

[9] Yaakov Bar-Shalom, X Rong Li, and Thiagalingam Kirubarajan. *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[10] *Deep Learning.* `https://www.ibm.com/cloud/learn/deep-learning`. Accessed: 2021-07-10.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[12] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[13] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 652–660.

[14] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[15] Ian Goodfellow et al. *Deep learning.* Vol. 1. MIT press Cambridge, 2016.

[16] Bo Li et al. "High performance visual tracking with siamese region proposal network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2018, pp. 8971–8980.

[17] Martin Danelljan et al. "Eco: Efficient convolution operators for tracking". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pp. 6638–6646.

[18] Qiang Wang et al. "Fast online object tracking and segmentation: A unifying approach". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2019, pp. 1328–1338.

[19] Liang Xu and Ruixin Niu. "Semi-supervised Visual Tracking Based on Variational Siamese Network". In: *International Conference on Dynamic Data Driven Application Systems*. Springer. 2020, pp. 328–336.

[20] Luca Bertinetto et al. "Fully-convolutional siamese networks for object tracking". In: *European conference on computer vision*. Springer. 2016, pp. 850–865.

[21] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[22] Xiao Wang et al. "Sint++: Robust visual tracking via adversarial positive instance generation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4864–4873.

[23] Chung-Ching Lin et al. "Video Instance Segmentation Tracking With a Modified VAE Architecture". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13147–13157.

[24] Li Ding and Lex Fridman. "Object as Distribution". In: *arXiv preprint arXiv:1907.12929* (2019).

[25] Matej Kristan et al. "The sixth visual object tracking vot2018 challenge results". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 0–0.

[26] SJ Hadfield, R Bowden, and K Lebeda. "The visual object tracking VOT2016 challenge results". In: *Lecture Notes in Computer Science* 9914 (2016), pp. 777–823.

[27] Lianghua Huang, Xin Zhao, and Kaiqi Huang. "Got-10k: A large high-diversity benchmark for generic object tracking in the wild". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[29] Alan Lukezic et al. "Discriminative correlation filter with channel and spatial reliability". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6309–6318.

[30] Feng Li et al. "Learning spatial-temporal regularized correlation filters for visual tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4904–4913.

[31] Chong Sun et al. "Learning spatial-aware regressions for visual tracking". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8962–8970.

[32] Martin Danelljan et al. "Atom: Accurate tracking by overlap maximization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4660–4669.

[33] Liang Xu and Ruixin Niu. "Tracking Visual Object As An Extended Target". In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2021, pp. 664–668.

[34] Matthias Mueller, Neil Smith, and Bernard Ghanem. "A benchmark and simulator for uav tracking". In: *European conference on computer vision*. Springer. 2016, pp. 445–461.

[35] Bo Li et al. “Siamrpn++: Evolution of siamese visual tracking with very deep networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4282–4291.

[36] Martin Danelljan, Luc Van Gool, and Radu Timofte. “Probabilistic regression for visual tracking”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7183–7192.

[37] Goutam Bhat et al. “Learning discriminative model prediction for tracking”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6182–6191.

[38] David S Bolme et al. “Visual object tracking using adaptive correlation filters”. In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2544–2550.

[39] Kaiming He et al. “Identity mappings in deep residual networks”. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[40] Marcus Baum, Florian Faion, and Uwe D Hanebeck. “Modeling the target extent with multiplicative noise”. In: *2012 15th International Conference on Information Fusion*. IEEE. 2012, pp. 2406–2412.

[41] Shishan Yang and Marcus Baum. “Tracking the orientation and axes lengths of an elliptical extended object”. In: *IEEE Transactions on Signal Processing* 67.18 (2019), pp. 4720–4729.

[42] Shishan Yang and Marcus Baum. “Extended Kalman filter for extended object tracking”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 4386–4390.

[43] Kolja Thormann, Shishan Yang, and Marcus Baum. "A Comparison of Kalman Filter-based Approaches for Elliptic Extended Object Tracking". In: *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*. IEEE. 2020, pp. 1–8.

[44] Martin Danelljan and Goutam Bhat. *PyTracking: Visual tracking library based on PyTorch*. 2019.

[45] Yaakov Bar-Shalom and Xiao-Rong Li. *Multitarget-multisensor tracking: principles and techniques*. Vol. 19. YBs Storrs, CT, 1995.

[46] Simon J Julier and Jeffrey K Uhlmann. "Unscented filtering and nonlinear estimation". In: *Proceedings of the IEEE* 92.3 (2004), pp. 401–422.

[47] Shane T Barratt and Stephen P Boyd. "Fitting a kalman smoother to data". In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 1526–1531.

[48] Pieter Abbeel et al. "Discriminative Training of Kalman Filters." In: *Robotics: Science and systems*. Vol. 2. 2005, p. 1.

[49] Dylan M Asmar and Greg J Eslinger. *Nonlinear programming approach to filter tuning*. 2012.

[50] Zhaozhong Chen et al. "Weak in the NEES?: Auto-tuning Kalman filters with Bayesian optimization". In: *2018 21st International Conference on Information Fusion (FUSION)*. IEEE. 2018, pp. 1072–1079.

[51] Lingyi Zhang et al. "On the Identification of Noise Covariances and Adaptive Kalman Filtering: A New Look at a 50 Year-old Problem". In: *IEEE Access* 8 (2020), pp. 59362–59388.

[52] Peter Ondruska and Ingmar Posner. "Deep tracking: Seeing beyond seeing using recurrent neural networks". In: *arXiv preprint arXiv:1602.00991* (2016).

[53] Robert Wilson and Leif Finkel. “A neural implementation of the Kalman filter”. In: *Advances in neural information processing systems*. 2009, pp. 2062–2070.

[54] Tuomas Haarnoja et al. “Backprop kf: Learning discriminative deterministic state estimators”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4376–4384.

[55] Huseyin Coskun et al. “Long short-term memory kalman filters: Recurrent neural estimators for pose regularization”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5524–5532.

[56] Youji Iiguni, Hideaki Sakai, and Hidekatsu Tokumaru. “A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter”. In: *IEEE Transactions on Signal processing* 40.4 (1992), pp. 959–966.

[57] Simon Haykin. *Kalman filtering and neural networks*. Vol. 47. John Wiley & Sons, 2004.

[58] Roberto Togneri and Li Deng. “Joint state and parameter estimation for a target-directed nonlinear dynamic system model”. In: *IEEE transactions on signal processing* 51.12 (2003), pp. 3061–3070.

[59] N Mert Vural, Salih Ergüt, and Suleyman S Kozat. “An efficient and effective second-order training algorithm for lstm-based adaptive learning”. In: *IEEE Transactions on Signal Processing* 69 (2021), pp. 2541–2554.

[60] Alina Kloss, Georg Martius, and Jeannette Bohg. “How to train your differentiable filter”. In: *Autonomous Robots* (2021), pp. 1–18.

[61] Zhenhua Zhang et al. “Multi-objective optimization of mechanisms with clearances in revolute joints”. In: *New Trends in Mechanism and Machine Science.* Springer, 2015, pp. 423–433.

[62] Guy Revach et al. “Kalmannet: Data-Driven Kalman Filtering”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* IEEE. 2021, pp. 3905–3909.

[63] Rico Jonschkowski, Divyam Rastogi, and Oliver Brock. “Differentiable particle filters: End-to-end learning with algorithmic priors”. In: *arXiv preprint arXiv:1805.11122* (2018).

[64] Peter Karkus, David Hsu, and Wee Sun Lee. “Particle filter networks with application to visual localization”. In: *Conference on robot learning.* PMLR. 2018, pp. 169–178.

[65] Xiao Ma et al. “Particle filter recurrent neural networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 34. 04. 2020, pp. 5101–5108.

[66] Adrien Corenflos et al. “Differentiable particle filtering via entropy-regularized optimal transport”. In: *International Conference on Machine Learning.* PMLR. 2021, pp. 2100–2111.

[67] Hao Wen et al. “End-To-End Semi-supervised Learning for Differentiable Particle Filters”. In: *arXiv preprint arXiv:2011.05748* (2020).

[68] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[69] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[70] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[71] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. "The matrix cookbook". In: *Technical University of Denmark* 7.15 (2008), p. 510.

[72] Andreas Geiger et al. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[73] Jorge Almeida and Vitor M Santos. "Real time egomotion of a nonholonomic vehicle using LIDAR measurements". In: *Journal of Field Robotics* 30.1 (2013), pp. 129–141.

[74] X Rong Li and Vesselin P Jilkov. "Survey of maneuvering target tracking. Part I. Dynamic models". In: *IEEE Transactions on aerospace and electronic systems* 39.4 (2003), pp. 1333–1364.

[75] Robin Schubert et al. "Empirical evaluation of vehicular models for ego motion estimation". In: *2011 IEEE intelligent vehicles symposium (IV)*. IEEE. 2011, pp. 534–539.

[76] Jean-Paul Laumond et al. *Robot motion planning and control*. Vol. 229. Springer, 1998.

[77] Robin Schubert, Eric Richter, and Gerd Wanielik. "Comparison and evaluation of advanced motion models for vehicle tracking". In: *2008 11th international conference on information fusion*. IEEE. 2008, pp. 1–6.

[78] Jianqing Wu. "An automatic procedure for vehicle tracking with a roadside LiDAR sensor". In: *Institute of Transportation Engineers. ITE Journal* 88.11 (2018), pp. 32–37.

[79] Axel Gern, Uwe Franke, and Paul Levi. "Robust vehicle tracking fusing radar and vision". In: *Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001 (Cat. No. 01TH8590)*. IEEE. 2001, pp. 323–328.

[80] Zheng Tang et al. "Single-camera and inter-camera vehicle tracking and 3D speed estimation based on fusion of visual and semantic features". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 108–115.

[81] Hsu-kuang Chiu et al. "Probabilistic 3d multi-object tracking for autonomous driving". In: *arXiv preprint arXiv:2001.05673* (2020).

[82] Xinshuo Weng et al. "AB3DMOT: A Baseline for 3D Multi-Object Tracking and New Evaluation Metrics". In: *arXiv preprint arXiv:2008.08063* (2020).

[83] Kristian Amundsen Ruud, Edmund Førland Brekke, and Jo Eidsvik. "LIDAR Extended Object Tracking of a Maritime Vessel Using an Ellipsoidal Contour Model". In: *2018 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*. IEEE. 2018, pp. 1–6.

[84] Karl Granström, Christian Lundquist, and Umut Orguner. "Tracking rectangular and elliptical extended targets using laser measurements". In: *14th International Conference on Information Fusion*. IEEE. 2011, pp. 1–8.

[85] Karl Granstrom, Marcus Baum, and Stephan Reuter. "Extended object tracking: Introduction, overview and applications". In: *arXiv preprint arXiv:1604.00970* (2016).

[86] Nemanja Djuric et al. “Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving”. In: *The IEEE Winter Conference on Applications of Computer Vision*. 2020, pp. 2095–2104.

[87] John Houston et al. “One Thousand and One Hours: Self-driving Motion Prediction Dataset”. In: *arXiv preprint arXiv:2006.14480* (2020).

[88] Yuning Chai et al. “Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction”. In: *arXiv preprint arXiv:1910.05449* (2019).

[89] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. “Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst”. In: *arXiv preprint arXiv:1812.03079* (2018).

[90] Jiyang Gao et al. “VectorNet: Encoding HD Maps and Agent Dynamics from Vectorized Representation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11525–11533.

[91] Sergio Casas et al. “Spatially-aware graph neural networks for relational behavior forecasting from sensor data”. In: *arXiv preprint arXiv:1910.08233* (2019).

[92] Ming-Fang Chang et al. “Argoverse: 3d tracking and forecasting with rich maps”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8748–8757.

[93] Holger Caesar et al. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11621–11631.

[94] M. Ulmke and W. Koch. “Road-Map Assisted Ground Moving Target Tracking”. In: *IEEE Transactions on Aerospace and Electronic Systems* 42.4 (Oct. 2006), pp. 1264–1274.

[95] C. Yang and E. Blasch. “Fusion of Tracks with Road Constraints”. In: *Journal of Advances in Information Fusion* 3.1 (June 2008), pp. 14–32.

[96] L. Snidaro et al., eds. *Context-Enhanced Information Fusion*. Springer, 2016.

[97] Liang Xu and Ruixin Niu. “TrafficEKF: a Learning Based Traffic Aware Extended Kalman Filter”. In: *2021 IEEE 24th International Conference on Information Fusion (FUSION)*. IEEE. 2021, pp. 1–8.

[98] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[99] Fang-Chieh Chou et al. “Predicting motion of vulnerable road users using high-definition maps and efficient convnets”. In: *arXiv preprint arXiv:1906.08469* (2019).

[100] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[101] Charles R Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (2020), pp. 357–362.

[102] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications (Artech House Radar Library)*. Artech Print on Demand, 2004.

[103] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. “Center-based 3d object detection and tracking”. In: *arXiv preprint arXiv:2006.11275* (2020).

[104] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3354–3361.

[105] Ming Liang et al. "Multi-task multi-sensor fusion for 3d object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7345–7353.

[106] Yan Yan, Yuxing Mao, and Bo Li. "Second: Sparsely embedded convolutional detection". In: *Sensors* 18.10 (2018), p. 3337.

[107] Yin Zhou and Oncel Tuzel. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499.

[108] Bin Yang, Wenjie Luo, and Raquel Urtasun. "Pixor: Real-time 3d object detection from point clouds". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7652–7660.

[109] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *arXiv preprint arXiv:1506.01497* (2015).

[110] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. "3d semantic segmentation with submanifold sparse convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9224–9232.

[111] Xiao Zhang et al. "Efficient L-shape fitting for vehicle detection using laser scanners". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 54–59.

[112] Corneliu Rablau. “LIDAR–A new (self-driving) vehicle for introducing optics to broader engineering and non-engineering audiences”. In: *Education and Training in Optics and Photonics*. Optical Society of America. 2019, 11143_138.

VITA

Liang Xu was born on May 10, 1987, in Tangshan City, Hebei Province, China, and is a Chinese citizen at the time when this dissertation written. He received his Bachelor of Science degree in Mechanical Engineering from Tianjin University of Technology, Tianjin, China, Master of Science in Mechanical Engineering from Wichita State University Wichita, Kansas, USA, and Master of Science in Computer Science from Georgia Institute of Technology, Atlanta, Georgia, USA. Liang was the winner of the 2019 ECE Outstanding Graduate Teaching Assistant Award from the School of Engineering at VCU.