Aalto University
School of Engineering

Wencan Mao

Vision-Based Vehicle Detection and Tracking in Intelligent Transportation System

Master's Thesis
Aalto University
School of Engineering
Department of Mechanical Engineering

Thesis submitted as partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, 20.03.2019
Supervisor: Professor Kari Tammi
Advisor: Professor Kari Tammi

Aalto University
School of Engineering

Aalto University, P.O. BOX 11000, 00076
AALTO
www.aalto.fi

Abstract of master's thesis

Author Wencan Mao

Title of thesis Vision-Based Vehicle Detection and Tracking in Intelligent Transportation System

Master programme Master's Programme in Mechanical Engineering **Code** ENG25

Thesis supervisor Professor Kari Tammi

Thesis advisor(s) Professor Kari Tammi

Date 20.03.2019 **Number of pages** 64+13 **Language** English

Abstract

This thesis aims to realize vision-based vehicle detection and tracking in the Intelligent Transportation System. First, it introduces the methods for vehicle detection and tracking. Next, it establishes the sensor fusion framework of the system, including dynamic model and sensor model. Then, it simulates the traffic scene at a crossroad by a driving simulator, where the research target is one single car, and the traffic scene is ideal. YOLO Neural Network is applied to the image sequence for vehicle detection. Kalman filter method, extended Kalman filter method, and particle filter method are utilized and compared for vehicle tracking. The Following part is the practical experiment where there are multiple vehicles at the same time, and the traffic scene is in real life with various interference factors. YOLO Neural Network combined with OpenCV is adopted to realize real-time vehicle detection. Kalman filter and extended Kalman filter are applied for vehicle tracking; an identification algorithm is proposed to solve the occlusion of the vehicles. The effects of process noise as well as measurement noise are analysed using variable-controlling approach. Additionally, perspective transformation is illustrated and implemented to transfer the coordinates from the image plane to the ground plane. If the vision-based vehicle detection and tracking can be realized and popularized in daily lives, vehicle information can be shared among infrastructures, vehicles, and users, so as to build interactions inside the Intelligent Transportation System.

Keywords Intelligent Transportation System, Computer Vision, Vehicle Detection, Vehicle Tracking, YOLO Neural Network, Sensor Fusion, Kalman Filter, Extended Kalman Filter, Particle Filter, Perspective Transformation.

Table of Contents

1. Introduction

1.1 Background

During the past few decades, the automobile industry has experienced dramatic development. New technologies are emerging from time to time, which make our lives more convenient. Meanwhile, due to the growing number of vehicles, the transport problem arises. The major challenges for the transportation system are traffic congestion, environment effects, energy consumption, safety hazards, high maintenance cost, and land occupation (Lin et al. 2017).

The first problem is traffic congestion. The number of vehicles in China has reached 319 million. Among them, there are more than one million cars in 58 cities, and more than three million cars in 7 cities (Chinese Ministry of Public Security 2018). The gigantic number of vehicles led to ubiquitous congestions and transport delays, especially in the high-dense urban area. Secondly, air pollution and energy consumption brought by the enormous number of vehicles could not be neglected, and the traffic congestions had made the case even worse. Thirdly, under the pressure of fast-paced life, people were tired of being stuck in the traffic jams; and this may lead to chaotic driving behavior and potential accidents. From the perspective of transport construction, everyone anticipates a self-contained and efficient transportation system. However, adding more transport infrastructure means adding considerable expense (Andersen et al. 2000), let al.one the high maintenance costs and land consumption.

Under these circumstances, the Intelligent Transportation System (ITS) is conceived to solve the problems and gradually becomes a global phenomenon (Figueiredo et al. 2001). Intelligent Transportation System is an integrated transportation system, which aims at providing innovative services for all kinds of transportation and traffic management so that users can learn more about the transportation network and use it more safely, coordinately, and intelligently (Boonphoka et al. 2014). From the system perspective, the main components of the Intelligent Transportation System are transportation infrastructures, vehicles, and users. Many problems result from the lack of timely and accurate information as well as the lack of coordination of the users in the system (Andersen et al. 2000). Intelligent Transportation System combines the traditional transportation infrastructure with new technologies in the information system, communication system, sensor system, and control system, and adopts advanced mathematical methods for optimized planning. (Lin et al. 2017). These technologies provide the users in the system a better understanding of the traffic condition so that they can make synergetic decisions. In this way, transportation sustainability and mobility are improved, energy efficiency is increased, environmental impacts are reduced, and traveling safety is secured.

1.2 Research Problem

Vision-based Intelligent Transportation System is extensively utilized in daily life because of four reasons. Firstly, people are used to visual information. The information does not need to be transformed or derived, which would save a lot of time for the users to make judgments. Secondly, video sequences are able to detect the time-varying tendency since they contain an extensive range of information which can reflect the traffic condition directly. Thirdly, the installation, operation, as well as maintenance of video sensors are simple. Lastly, the vision-based device is cost-effective. (Zhang et al. 2011)

The detection, recognition, and tracking of objects in traffic system are widely used in vision-based Intelligent Transportation System. Vehicle detection and tracking, in particular, has broad applications such as over-speeding and red-light running identification, parking lot access control, automatic charging, and lost vehicle tracking (Zhang et al. 2011). Moreover, it can be used to construct a "vehicle-road-user" system. Real-time traffic image sequences are collected by cameras on road; traffic flow parameters are extracted by vehicle detection, recognition and tracking module; then the traffic information is processed in the control center and sent to the users (Liu et al. 2013), so as to build interactions among road, vehicle, and users.

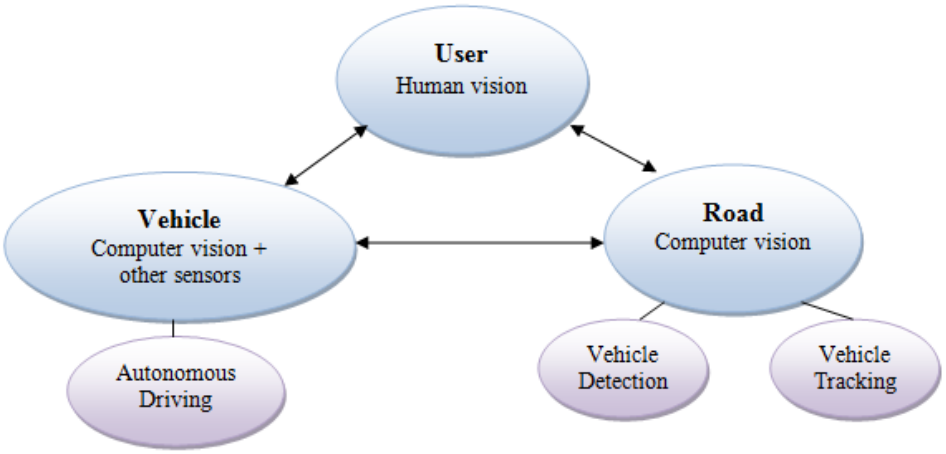


Figure 1.1: Components of Vision-based Intelligent Transportation System

There are some issues about vehicle detection and tracking in the Intelligent Transportation System. Firstly, the vehicles have different shapes, sizes, and colors, which increase the difficulty of detection. Secondly, the figure of vehicles will change with its pose and orientation. Thirdly, environmental factors could affect the result of detection and tracking. Fourth, the fast movement and the tracking algorithm require

high computing power. Lastly, the motion and drifts of vehicles require higher robustness. (Zhang et al. 2011) The issues will later be discussed and solved in the thesis.

1.3 Scope and Objectives

The objectives of the thesis are to achieve vision-based vehicle detection and tracking in the Intelligent Transportation System. Vehicle detection is to perceive the vehicles passing the region of detection (Liu et al. 2013). The vehicles need to be extracted from the video and located within the image. For computers, it manages to recognize a specific image such as QR codes, but it struggles to recognize things without "expectation". These require computer vision, which automates acquiring, processing, analyzing and understanding digital images or videos, and extracting information from them (Klette 2014). After recognition of the vehicle, it needs to be located within the region of detection; therefore, a bounding box is drawn around the vehicle to get its pixel location.

Beyond vehicle detection, vehicle tracking aims to estimate the motion parameters, calculate the corresponding trajectories, and predict the upcoming position of vehicles on the road (Sivaraman et al. 2013). Vehicle tracking could be unsmooth since the Visual-based vehicle detectors fluctuate between frames at pixel positions. For example, a pedestrian passing by may affect the detection result and lead to a drift in the trajectory. Besides, the loss of measurement data may occur when the vehicle is driving to the blind spot. The thesis aims to smooth out the tracking process.

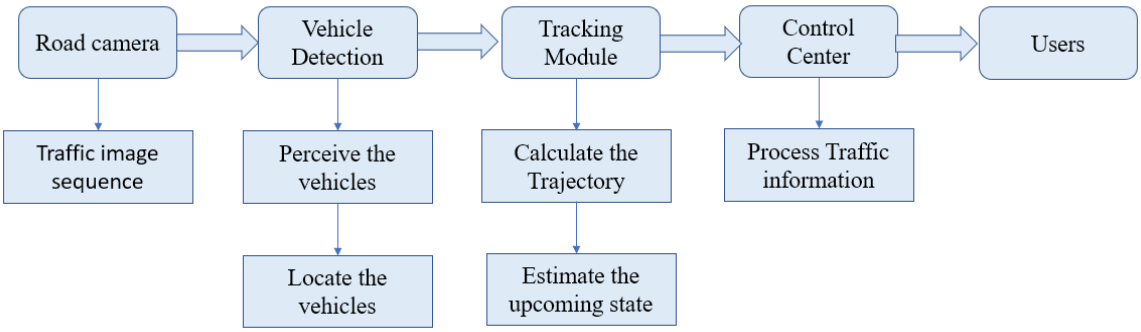


Figure 1.2: Flowchart of Vision-based Intelligent Transportation System

1.4 Structure of the Thesis

The thesis is organized as follows. In the second chapter, vehicle detection and tracking methods are discussed respectively, existing literature is introduced, and the methods of the thesis are selected. In the third chapter, a mathematical framework is established using the concept of sensor fusion. Dynamic model and sensor models are built, and the dynamic model is discretized correspond to the measurement sampling. In the fourth chapter, vehicle detection and tracking are simulated. The video simulates the traffic scene at a crossroad by a driving simulator. One vehicle is set as the target, and the traffic scene is an ideal one without inference factors such as pedestrians, bikes and other vehicles. The methods selected are tested by the simulation video to verify the feasibility and precision of each method. On this basis, in the fifth chapter, a practical experiment is carried out. The video is recorded by a camera installed at a crossroad at Aalto University. Various vehicles, pedestrians, bikes pass by from time to time, and there is also a construction site which blocks part of the view of the objects behind. The algorithm is optimized to meet more complicated requirements. And the effects of noise are analyzed using variable-controlling method. Then introduces the principle of perspective transformation to convert the coordinates from the image plane to the ground plane. And vehicle tracking with perspective transformation are conducted using Kalman filter method and extended Kalman filter method. The following chapter is the discussion of the findings, and the last chapter is the conclusion of the thesis.

2. Method

The methodology of the thesis is introduced, compared, and determined in this chapter. Various methods of vision-based vehicle detection and tracking are discussed; the current literature examples of realizing vision-based vehicle detection and tracking are taken for references; and finally, the methods to be adopted in the thesis are decided.

2.1 Vehicle Detection Method

Vehicle detection aims to estimate if there is any vehicle passing through the region of detection, and locate the vehicle within the region of detection. In order to distinguish the vehicle from the background, three steps are adopted in the detection algorithm: image acquisition, generation of candidates and verification of candidates (Bush et al. 2011, Cheng et al. 2011). There are two kinds of methods for vision-based vehicle detection: methods based on intrinsic properties and methods based on motion detection (Liu et al. 2013).

Methods based on intrinsic properties include representative methods, machine learning methods, and 3D modeling methods. Representative methods use intrinsic visual properties to detect the vehicles, such as symmetry, contour, edge, color, texture, shadow and parts of the vehicle. The methods are based upon the subjective hypothesis of the vehicle – they depend on the knowledge that human learns and describes before teaching the computers (Liu et al. 2013). Machine learning methods use a recognition classifier generated from training dataset to distribute category tags to the things it detects and shows the probability of each labeled category (Buch et al. 2011). They do not depend on the knowledge of human, thus have been widely used in computer vision. 3D modeling methods detect and track the vehicle by constructing 3D models of them. However, it is a challenge to build the 3D models which can sufficiently represent the features of vehicles, and suitable for all kinds of vehicles (Liebelt et al. 2008).

Methods based on motion detection aim to capture the most common characteristics of the vehicles – their “movement”. The methods include background subtraction method, optical flow method, frame differencing method, and virtual coil method. Background subtraction method compares the divergence between the present image and the background image by pixel and sets a threshold to determine whether it is background or not (Gupte et al. 2002). Optical flow method calculates the motion of every pixel by combing the temporal variation of the pixel with the relevance of the pixels in the image sequence (Beauchemin et al. 1995). Frame differencing method calculates the difference between two contiguous frames by pixel and set the threshold to obtain the moving foreground pixels and their region (Seki 2000). Virtual coil method detects vehicles by

the change of the image in the virtual coil area, assuming that there is a vehicle when the width of the covered image is larger than the threshold value.

For detection methods based on intrinsic properties, machine learning methods have a broad prospect. And for the motion detection methods, although the algorithm is much easier than machine learning methods, they could not always reflect the truth – the moving objects are not limited to vehicles, for example the pedestrians and bikes are also moving; and the vehicles can also be static, such as the cars in the parking lot (Liu et al. 2013). Therefore, detection methods based on machine learning will be further discussed in the thesis.

In recent years, Convolutional neural network (CNN) was developed, which does well in completing difficult tasks of object recognition. Convolutional Neural Network is capable of recognizing the visual patterns straightly from pixel images with a minimum amount of preprocessing. They can identify visual patterns with great variability and have the ability to resist distortion and simple geometric transformation.

On the basis of object recognition, object detection needs to draw a bounding box around the object of interest to locate it within the region of detection. It is not feasible to merely build a standard convolutional network with a fully connected layer, because the amount of the output is not a constant. The objects of interest can be located in different places of the image with its own aspect ratios. To avoid selecting a huge number of regions, four latest methods are introduced and compared here: R-CNN, fast R-CNN, faster R-CNN and You Only Look Once (YOLO) algorithm.

The first method is R-CNN proposed by Ross Girshick et al. In the method, selective search is used to extract 2000 regions from the image to be the region proposals. The candidate region proposals are warp into a square and then fed into a convolutional neural network that generates an output of a 4096-dimensional feature vector. The features extracted from the image are then fed into a Support Vector Machine to classify the objects within the region proposal, and predict four offset values to adjust the location of the bounding box. However, it takes around 47 seconds to classify 2000 region proposals per image.

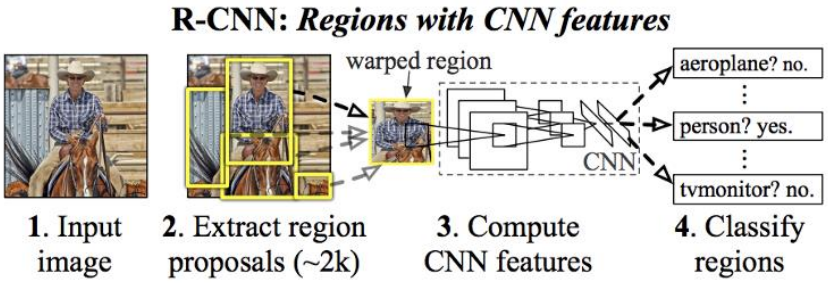


Figure 2.1: R-CNN system overview (Girshick et al. 2014)

The second method is fast R-CNN proposed by Ross Girshick. It is similar to R-CNN; except that the region proposals are not fed into the convolutional neural network, but the input image is directly fed into it to generate a convolutional feature map. From the map, the regions of proposals are identified and warp into squares. Region of interest (RoI) pooling layer is used to reshape the squares and feed them into the fully connected layer. Softmax layer is used to classify the region proposal from the RoI feature vector, and predict the offset values of the bounding box. It is much faster than R-CNN because the convolutional feature map is generated once per image instead of feeding 2000 region proposals in into the convolutional neural network every time.

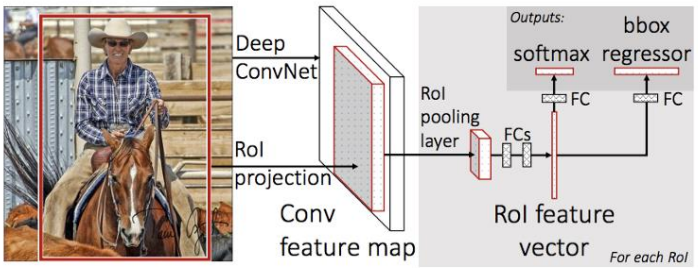


Figure 2.2: Fast R-CNN method overview (Girshick et al. 2015)

The third method is faster R-CNN. It is proposed by Shaoqing Ren et al. on the basis of fast R-CNN. Instead of using selective searching algorithm to predict the region proposals, it uses a region proposal network to do so. The method can significantly save time for detection because selective searching algorithm is a time-consuming process.

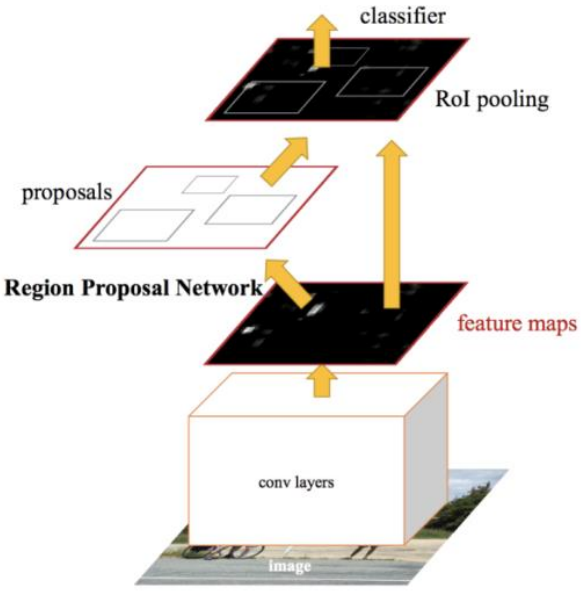


Figure 2.3: Faster R-CNN method overview (Ren et al. 2016)

The last method is You Only Look Once (YOLO) algorithm proposed by Joseph Redmon et al. All of the above methods use regions to localize the objects; the convolutional neural network does not look at the whole image, but the parts that have a high probability of containing the objects. YOLO algorithm, however, uses a single convolutional neural network to predict the bounding box together with the class probabilities. In the algorithm, the input image is split into an S×S grid, and within every grid, m bounding boxes are generated. The network outputs the class probability and offset values for each of the bounding boxes, and the bounding boxes with the probability larger than the threshold will represent the objects detected. Yolo neural network is very simple and fast. It processes images in real time at 45 frames per second.

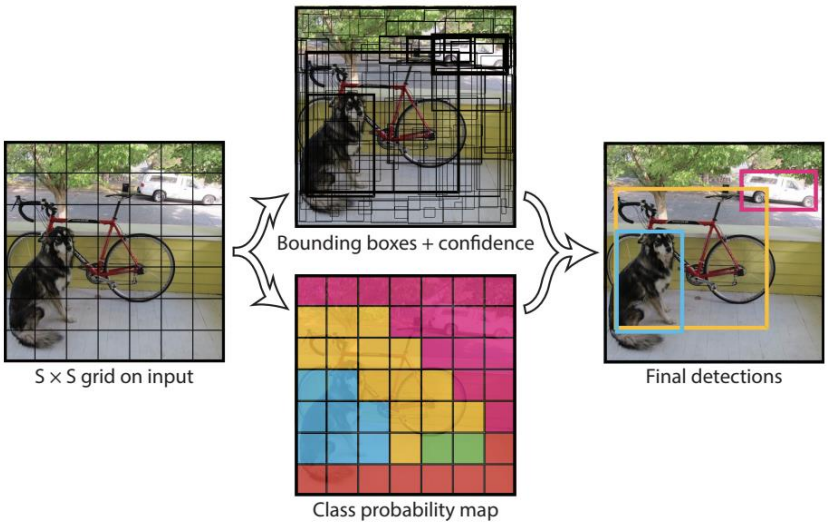


Figure 2.4: YOLO method overview (Redmon et al. 2016)

2.2 Vehicle Tracking Method

Vehicle tracking aims to estimate the motion parameters, calculate the corresponding trajectories, and predict the upcoming position of vehicles. In vehicle detection, the bounding box of the vehicles and the locations of the bounding box are already obtained. The bounding box can be the representation of the vehicles in the tracking process. There are two types of methods for vision-based vehicle tracking: methods with prior knowledge and methods without prior knowledge (Liu et al. 2013).

Methods with prior knowledge include template match method and mean shift method. Template match method looks for a specific template from vehicle image and uses certain correlation to match and recognize vehicle in both steady state and dynamic state images (Fieguth et al. 1997). It is simple and precise but takes a long implementing time. Mean shift method uses a non-parameter density gradient estimator based on a general kernel function to analyze the image. If the kernel function meets the requirement, the estimation would be asymptotic unbiased, continuous, and consistent with the true value (Comaniciu et al. 2003).

Methods without prior knowledge include Kalman filter method, extended Kalman filter method, unscented Kalman filter method, and particle filter method.

Kalman filter method is suitable for linear state-space models. It uses a series of measurements over time, including noise and uncertainty to estimate the upcoming state. The dynamic model is used to predict the upcoming state, and the upcoming state is estimated using the prediction as well as the new measurement. The information imposed by the dynamic model significantly improves the tracking performance. (Gustaffson 2018)

Extended Kalman method and unscented Kalman filter method are suitable for nonlinear state-space models. Extended Kalman filter method linearize the nonlinear models at a certain point, so the Jacobian matrix around the point needs to be calculated. Unscented Kalman filter collects a set of deterministic samples and propagates them on the nonlinear function to calculate the means and covariance. (Gustaffson 2018)

Particle filter is suitable for both linear and nonlinear state-space models. It uses a set of random samples with important weights to estimate the upcoming state. The important weights indicate the relevance of each sample. The dynamic model is used to propagate the samples; the measurement update evaluates the likelihood to assign an importance

weight to each sample; re-sampling is used to mitigate particle degeneracy. When the number of sampled particles is large enough, the samples can sufficiently describe the probability density distribution. (Gustaffson 2018)

2.3Method Selection

For vehicle detection method, methods based on machine learning are being considered. Compared to the classifier-based method, YOLO method has the following advantages. First, it looks at the entire image at once so that the prediction can be notified through the global context of the image. Secondly, it predicts through a single network evaluation, so it is very fast and suitable for real-time detection. Thirdly, YOLO has universality and extensive suitability. When it is extended from natural images to other fields such as works of art, it is superior to other methods. Fourthly, YOLO detects an object in each grid cell. It enhances spatial diversity in forecasting. A good example is Ojala et al. apply YOLO algorithm to localize pedestrians in real-time with high accuracy. Therefore, YOLO algorithm is selected as the vehicle detection method.

For vehicle tracking methods, methods without prior knowledge are being considered. Generally, Kalman filter method is suitable for linear model; extended Kalman filter method is suitable for weakly nonlinear model; unscented Kalman filter is suitable for highly nonlinear model; particle filter method is for both linear and nonlinear model (Zhao et al. 2015). Unscented Kalman filter is not selected due to its computational complexity. Therefore, Kalman filter method, extended Kalman filter method and particle filter method will be adopted and compared in the following context. The flowchart of vision-based vehicle detection and tracking is shown in Figure 2.5.

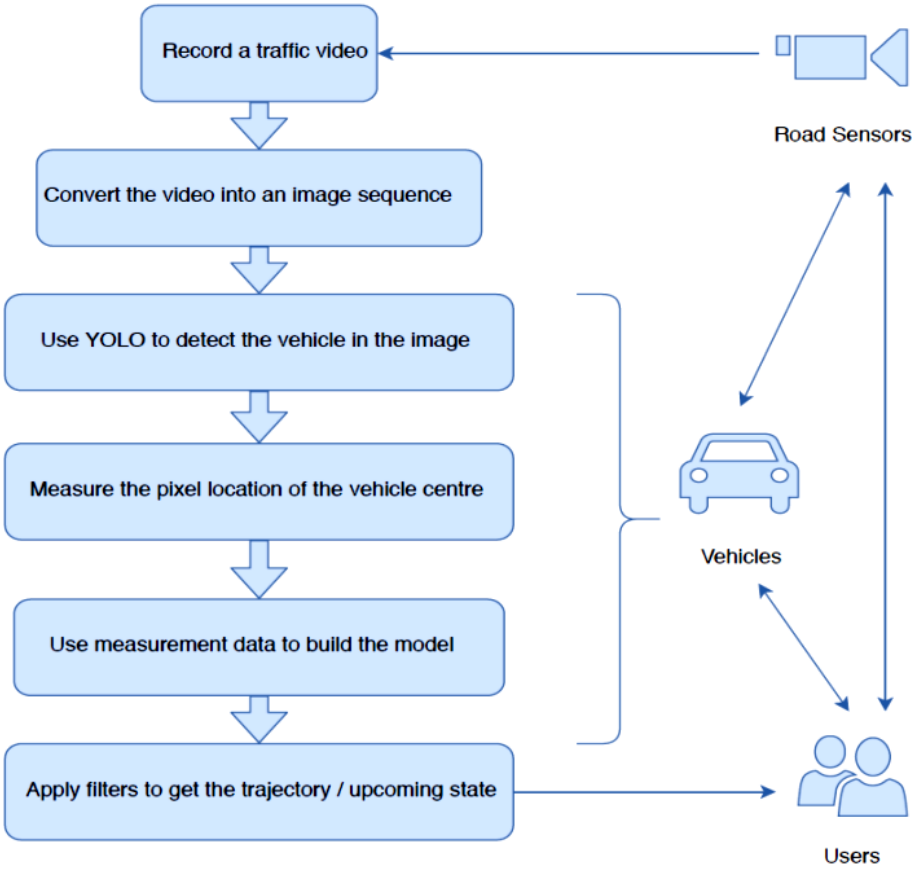


Figure 2.5 Flowchart of vision-based vehicle detection and tracking

2.4 Existing Literature

Having decided the thesis approaches, some literature relating to the topics are introduced here. The first topic is about using YOLO-based algorithm to detect the traffic participants; another topic is about utilizing the filtering algorithm to track the vehicles.

2.4.1 Literature about Vehicle Detection

Aleksa Corovic et al. (2018) use YOLO algorithm to detect real-time traffic participants. They train the algorithm for five categories: car, truck, pedestrians, traffic signs, and traffic lights. The result shows that YOLO algorithm is suitable for real-time detection, and all of the objects close to the camera are successfully detected. They point out that the failure of detection occurs in two situations: the first is in high dense traffic area when one cell of the detection grid contains more than three objects; the other is when some objects are blocked by others. They also compare the detection results in different weathers and declare that the accuracy of the algorithm could be improved by training from a larger and more diverse dataset covering various weather and lighting conditions.

Figure 2.6: Detection visualization examples in sunny (left) and snowy (right) weathers (Aleksa Corovic et al. 2018)

Zhi Xu et al. (2018) propose a vehicle detection system under UAV based on optimal dense YOLO method. For vehicle detection under aerial view angle of UAV platform, the background information is complex and the vehicle target is small. To solve the problem, they combine YOLO algorithm with dense topology and optimal pooling. The result shows the model is significantly improved in extracting features of the small target. It also presents better accuracy and robustness, while maintaining the fast speed of YOLO algorithm.

Figure 2.7: YOLO v2 detection result (left) and optimal dense YOLO method detection result (right) (Zhi Xu et al. 2018)

Jiaping Lin et al. (2018) develop a YOLO-based traffic counting system. The system consists of three modules, the Detector that generates the bounding boxes of the vehicles, the Buffer that stores the vehicle coordinates, and the Counter responsible for counting the vehicles. Video is input to the Detector where passing through YOLO and filter. Data access is built among frame number input and output, previous and current array in Buffer, and vehicle counting algorithm in Counter. They also add checkpoints to verify the validity of the detection. The correctness and overall efficiency of the system are evaluated by using video from different positions and angles. The results show that the system achieves high accuracy in the light-rich environment.

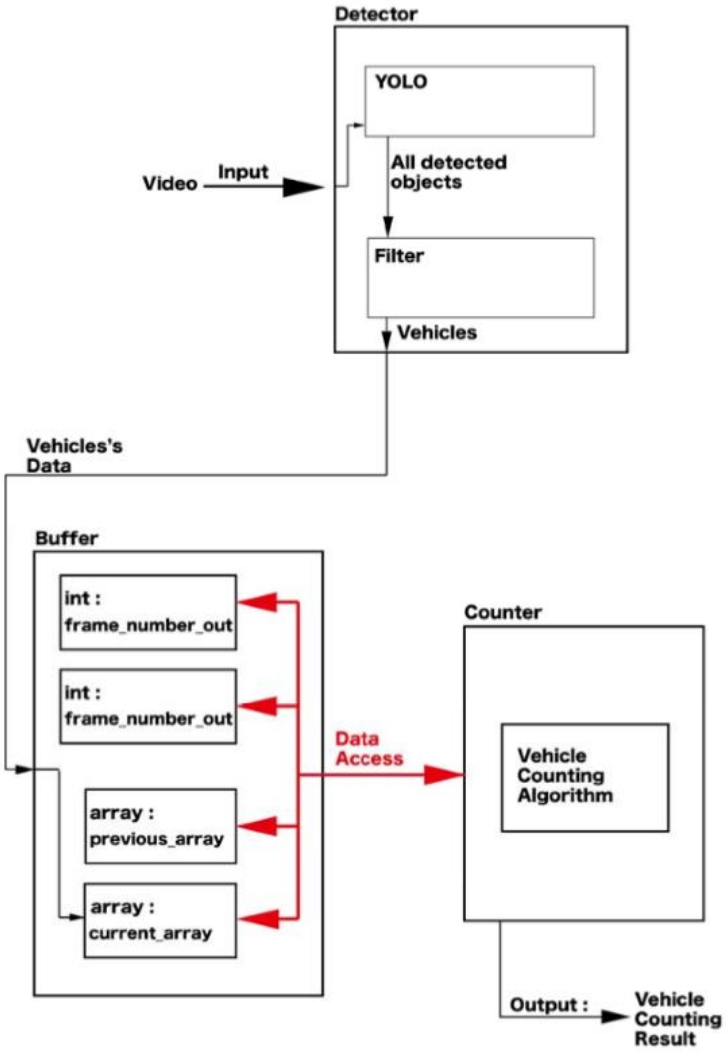
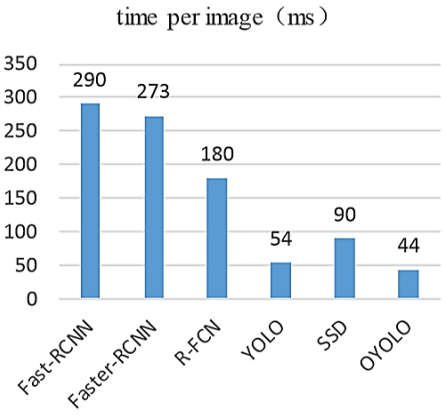


Figure 2.8: Architecture of traffic counting system (Jiaping Lin et al. 2018)

Jing Tao et al. (2017) build an object detection system for images in the traffic scene. They developed an optimized YOLO method called OYOLO by removing the last two fully connected layers and adding an average pool layer. The new network is faster than YOLO while exceeding other region-based methods in accuracy such as R-CNN. Combing OYOLO and R–FCN algorithm, the detection becomes more accurate. They also add a pre-processing procedure for night images by removing highlights, enhancing contrast and increasing brightness. The results show the new object detection system in the traffic scene is fast, accurate, and robust.



	mAP (trained on our training set)	mAP (trained on VOC07 + VOC12)
Fast-RCNN	-	60.9%
Faster-RCNN	-	62.4%
R-FCN	-	67.7%
YOLO	-	64.0%
SSD	-	64.5%
OYOLO	80.1%	66.0%
OYOLO + R-FCN	83.4%	69.3%

Figure 2.9: Speed (left) and accuracy (right) of OYOLO method compared with other object detection methods (Jing Tao et al. 2017)

2.4.2 Literature about Vehicle Tracking

Amir Salarpour et al. (2011) develop an algorithm to realize multiple vehicle tracking using Kalman filter and feature. First, they use background subtraction approach to detect the vehicles. Then they utilize both region-based and feature-based algorithms to track the vehicle. For region-based tracking, they use Kalman filter to predict the region of the vehicle in the next frame. And for feature-based tracking, they use color and size features to correct the results of the prediction. The result reflects that the method can solve the problem such as appearance, disappearance, and occlusion, thus it can distinguish and track all the vehicles individually in clutter scene with high accuracy.

Figure 2.10: Multi-vehicle tracking in clutter scene (Amir Salarpour et al. 2011)

Daniel Ponsa et al. (2005) present a multiple vehicle 3D tracking method using unscented Kalman filter. They use a monochrome camera platform installed on vehicles facing the road ahead with an inclination angle and determine the host coordinate system based on the camera position. Then a three-dimensional dynamic model of the system is established, in which the platform and the tracked vehicle are represented by state vectors. Combining the different observed values, a set of two-dimensional areas for vehicle

searching is obtained. The state vector is reevaluated by unscented Kalman filter from measurements obtained in every frame. After testing in long sequences, the results proved to be reliable even under poor acquisition conditions.

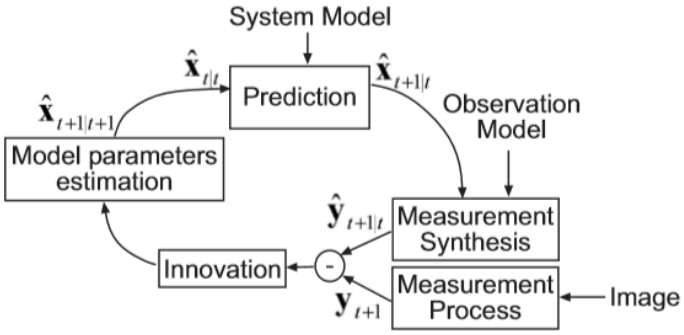


Figure 2.11: Model-based estimation cycle (Daniel Ponsa et al. 2005)

3. Mathematical Framework of the Vehicle

In order to detect and track the vehicle, the mathematical framework of the vehicle needs to be established. The concept of sensor fusion is employed here, which composed of sensor, variable of interest, models and estimation algorithm (Gustafsson 2018). Sensor, variable of interest, and models are established in this chapter, and estimation algorithm is discussed later in chapter 4.2.

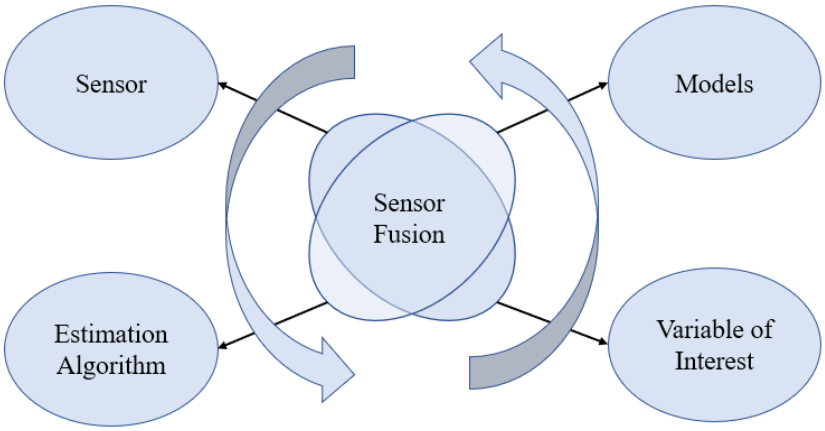


Figure 3.1: Basic concept of sensor fusion

The variable of interest is the time-varying state of a dynamic system that can be measured directly or indirectly. The state can be measured directly is the pixel location of the vehicle at each time step t_n, consisting of x-coordinate and y-coordinate. According to different indirect states, two models are established, Wiener velocity model and quasi-constant turn model. The former one is a linear model, while the latter one is a nonlinear model.

The derivation of equations takes reference of the work by Fredrik Gustafsson (2018).

3.1 Wiener Velocity Model

The movement of the vehicle can be determined by its acceleration, as shown in Figure 3.2.

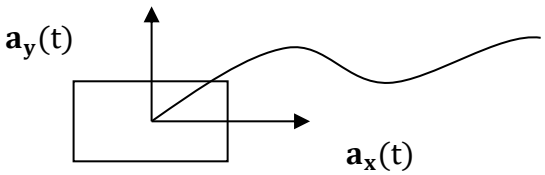


Figure 3.2: The movement of the vehicle determined by its acceleration

Assume the acceleration along x-axis and y-axis is random, it can be denoted as:

$$\dot{\boldsymbol{v}}(\mathrm{t}) = \mathbf{a}(\mathrm{t}) = \begin{bmatrix} \omega_1(t) \\ \omega_2(t) \end{bmatrix} \tag{1}$$

where $\omega_1(t)$ and $\omega_2(t)$ are stochastic quantity with variance δ_ω^2.

Set the variable of interest $\boldsymbol{x}(t)$ as:

$$\boldsymbol{x}(t) = [p^x(t) \quad p^y(t) \quad v^x(t) \quad v^y(t)]^T \tag{2}$$

The state-space model is:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}^x(t) \\ \dot{v}^y(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p^x(t) \\ p^y(t) \\ v^x(t) \\ v^y(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_1(t) \\ \omega_2(t) \end{bmatrix} \tag{3}$$

3.2 Quasi-Constant Turn Model

The movement of the vehicle can be determined by its speed φ(t) and heading direction φ(t), as shown in Figure 3.3.

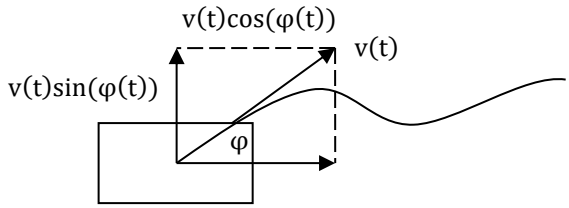


Figure 3.3: The movement of the vehicle determined by speed and heading direction

The speed along x-axis and y-axis is denoted as:

$$\dot{\boldsymbol{p}}(t) = \boldsymbol{v}(t) = \left[v(t)\, cos\big(\varphi(t)\big) \quad v(t)\, sin\big(\varphi(t)\big)\right]^T \tag{4}$$

Assume the speed acceleration, as well as the angular acceleration, is random, they can be denoted as:

$$\begin{bmatrix} \dot{v}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} \omega_1(t) \\ \omega_2(t) \end{bmatrix} \tag{5}$$

where $\omega_1(t)$ and $\omega_2(t)$ are stochastic quantity with variance δ_ω^2.

Set the variable of interest $\boldsymbol{x}(t)$ as:

$$\boldsymbol{x}(t) = [p^x(t) \quad p^y(t) \quad v(t) \quad \varphi(t)]^T \tag{6}$$

The state-space model is:

$$\begin{bmatrix} \dot{p}^x(t) \\ \dot{p}^y(t) \\ \dot{v}(t) \\ \dot{\varphi}(t) \end{bmatrix} = \begin{bmatrix} v(t)\,cos\big(\varphi(t)\big) \\ v(t)\,sin\big(\varphi(t)\big) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_1(t) \\ \omega_2(t) \end{bmatrix} \tag{7}$$

3.3 Sensor Model

A sensor is a device that provides a measurement of a variable of interest. The sensor, in this case, is the road camera that is used to record the traffic scene.

The sensor measurement is affected by measurement noise, and it represents the coordinates along x-axis and y-axis. So, for both of Wiener velocity model and quasi-constant turn model, the sensor model is:

$$\boldsymbol{y}_n = \boldsymbol{G}_n \boldsymbol{x}_n + \boldsymbol{r}_n \tag{8}$$

where

$$\boldsymbol{G}_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{9}$$

$\boldsymbol{r}_n$ is the measurement noise, which is a multivariate random variable with probability density function, which can be denoted as:

$$\boldsymbol{r}_n \sim p(\boldsymbol{r}_n) \tag{10}$$

Assume it is a zero-mean, independent random variable with covariance $\boldsymbol{R}_n$, it can be represented as:

$$E\{\boldsymbol{r}_n\} = 0, Cov\{\boldsymbol{r}_n\} = \boldsymbol{R}_n \tag{11}$$

3.4 Discretization of the State-Space Model

The measurement of the vehicle location is sampled at each time step t_n, so the continuous state-space models need to be discretized. Discretization of the models is equivalent to solving the stochastic differential equation between t_{n-1} and t_n.

3.4.1 Discretization of Wiener Velocity Model

Wiener Velocity Model is a stochastic linear dynamic model:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}_{\omega}\boldsymbol{\omega}(\mathrm{t}) \tag{12}$$

Multiply the integrating factor $\mathrm{e}^{-\mathbf{A}\mathbf{t}}$ we get:

$$e^{-\boldsymbol{A}t}\dot{\boldsymbol{x}}(t) = e^{-\boldsymbol{A}t}\boldsymbol{A}\boldsymbol{x}(t) + \mathrm{e}^{-\mathbf{A}\mathbf{t}}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t) \tag{13}$$

Rearranging it we get:

$$e^{-\boldsymbol{A}t}\dot{\boldsymbol{x}}(t) - e^{-\boldsymbol{A}t}\boldsymbol{A}\boldsymbol{x}(t) = \mathrm{e}^{-\mathbf{A}\mathbf{t}}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t) \tag{14}$$

Substituting $\frac{\mathrm{d}}{\mathrm{dt}}e^{-\boldsymbol{A}t}\boldsymbol{x}(t) = e^{-\boldsymbol{A}t}\dot{\boldsymbol{x}}(t) - e^{-\boldsymbol{A}t}\boldsymbol{A}\boldsymbol{x}(t)$ we get:

$$\frac{\mathrm{d}}{\mathrm{dt}}e^{-\boldsymbol{A}t}\boldsymbol{x}(t) = \mathrm{e}^{-\mathbf{A}\mathbf{t}}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t) \tag{15}$$

Integrate it with respect to t we get:

$$\int_{t_{n-1}}^{t_n} \frac{d}{dt}e^{-\boldsymbol{A}t}\boldsymbol{x}(t)dt = \int_{t_{n-1}}^{t_n} e^{-\boldsymbol{A}t}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t)\,dt \tag{16}$$

$$e^{-\boldsymbol{A}t}\boldsymbol{x}(t_n) - e^{-\boldsymbol{A}t_{n-1}}\boldsymbol{x}(t_{n-1}) = \int_{t_{n-1}}^{t_n} e^{-\boldsymbol{A}t}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t)\,dt \tag{17}$$

The solution is:

$$\boldsymbol{x}(t_n) = e^{\boldsymbol{A}(t_n - t_{n-1})}\boldsymbol{x}(t_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - t)}\boldsymbol{B}_{\omega}\boldsymbol{\omega}(t)\,dt \tag{18}$$

It can be written as:

$$\boldsymbol{x}_n = \boldsymbol{F}_n\boldsymbol{x}_{n-1} + \boldsymbol{q}_n \tag{19}$$

where

$$\boldsymbol{F}_n \equiv e^{\boldsymbol{A}(t_n - t_{n-1})} = \boldsymbol{I} + \Delta t\boldsymbol{A} \tag{20}$$

$$\boldsymbol{q}_n \equiv \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - t)}\boldsymbol{B}_{\omega}\,\boldsymbol{\omega}(t)dt \tag{21}$$

$\boldsymbol{\omega}(t)$ is a white noise process and $R_{\omega\omega}(\tau) = \sigma_w^2\delta(\tau)$

The process noise $\boldsymbol{q}_n$ satisfies:

$$\boldsymbol{q}_n \sim N(0, \boldsymbol{Q}_n) \tag{22}$$

$$\boldsymbol{Q}_n = Cov\{\boldsymbol{q}_n\} = \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n-\tau)} \boldsymbol{B}_\omega \, \sigma_w^2 \boldsymbol{B}_\omega^T e^{\boldsymbol{A}^T(t_n-\tau)} d\tau \tag{23}$$

Substitute in the matrices we get:

$$\boldsymbol{F}_n = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

$$\boldsymbol{Q}_n = \sigma_\omega^2 \begin{bmatrix} \frac{(\Delta t)^3}{3} & 0 & \frac{(\Delta t)^2}{2} & 0 \\ 0 & \frac{(\Delta t)^3}{3} & 0 & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{(\Delta t)^2}{2} & 0 & \Delta t \end{bmatrix} \tag{25}$$

Therefore, the discrete time state-space model is:

$$\begin{cases} \begin{bmatrix} p_n^x \\ p_n^y \\ v_n^x \\ v_n^y \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{n-1}^x \\ p_{n-1}^y \\ v_{n-1}^x \\ v_{n-1}^y \end{bmatrix} + \boldsymbol{q}_n \\ \boldsymbol{y}_n = \boldsymbol{G}_n \boldsymbol{x}_n + \boldsymbol{r}_n \end{cases} \tag{26}$$

with

$$E\{\boldsymbol{q}_n\} = 0, Cov\{\boldsymbol{q}_n\} = \boldsymbol{Q}_n \tag{27}$$

$$E\{\boldsymbol{r}_n\} = 0, Cov\{\boldsymbol{r}_n\} = \boldsymbol{R}_n \tag{28}$$

3.4.2 Discretization of Quasi-Constant Turn Model

Quasi-Constant Turn Model is a stochastic nonlinear dynamic model:

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t)\big) + \boldsymbol{B}_\omega\big(\boldsymbol{x}(t)\big)\boldsymbol{\omega}(\mathrm{t}) \tag{29}$$

First, the model is linearized around $\boldsymbol{x}_{n-1}$ using first order Taylor series approximation:

$$f\big(\boldsymbol{x}(t)\big) \approx f(\boldsymbol{x}_{n-1}) + \boldsymbol{A}_x(\boldsymbol{x}(t) - \boldsymbol{x}_{n-1}) \tag{30}$$

$$\dot{\boldsymbol{x}}(t) \approx f(\boldsymbol{x}_{n-1}) + \boldsymbol{A}_x(\boldsymbol{x}(t) - \boldsymbol{x}_{n-1}) + \boldsymbol{B}_\omega\big(\boldsymbol{x}(t)\big)\boldsymbol{\omega}(\mathrm{t}) \tag{31}$$

where $\boldsymbol{A}_x$ is the Jacobian matrix of $f(\boldsymbol{x}(t))$ at $\boldsymbol{x}_{n-1}$.

Substitute in the matrices we get:

$$\boldsymbol{A}_x = \begin{bmatrix} 0 & 0 & \cos(\varphi_{n-1}) & -v_{n-1}\sin(\varphi_{n-1}) \\ 0 & 0 & \sin(\varphi_{n-1}) & v_{n-1}\cos(\varphi_{n-1}) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{32}$$

Then the model is discretized in the same way:

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - t)} dt\, f(\boldsymbol{x}_{n-1}) + \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - t)} \boldsymbol{B}_\omega \left(\boldsymbol{x}(t)\right)\boldsymbol{\omega}(t) dt \tag{33}$$

$$\begin{aligned} \boldsymbol{F}_x &\equiv e^{\boldsymbol{A}_x(t_n - t_{n-1})} = \boldsymbol{I} + \Delta t \boldsymbol{A}_x \\ &= \begin{bmatrix} 1 & 0 & \Delta\,\mathrm{tcos}(\varphi_{n-1}) & -\Delta t v_{n-1}\sin(\varphi_{n-1}) \\ 0 & 1 & \Delta\,\mathrm{tsin}(\varphi_{n-1}) & \Delta t v_{n-1}\cos(\varphi_{n-1}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{34}$$

$$\boldsymbol{q}_n \equiv \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - t)} \boldsymbol{B}_\omega \left(\boldsymbol{x}(t)\right)\boldsymbol{\omega}(t) dt \tag{35}$$

with

$$\boldsymbol{q}_n \sim N(0, \boldsymbol{Q}_n) \tag{36}$$

$$\boldsymbol{Q}_n = Cov\{\boldsymbol{q}_n\} = \int_{t_{n-1}}^{t_n} e^{\boldsymbol{A}(t_n - \tau)} \boldsymbol{B}_\omega\, \sigma_w^2 \boldsymbol{B}_\omega^T e^{\boldsymbol{A}^T(t_n - \tau)} d\tau \tag{37}$$

However, it is still hard to determine $\boldsymbol{Q}_n$. So, Euler-Maruyama method is used here.
The stochastic nonlinear dynamic model is:

$$\dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t)\big) + \boldsymbol{B}_\omega\big(\boldsymbol{x}(t)\big)\boldsymbol{\omega}(\mathrm{t}) \tag{38}$$

The integral representation is:

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \int_{t_{n-1}}^{t_n} f\big(\boldsymbol{x}(t)\big)\, dt + \int_{t_{n-1}}^{t_n} \boldsymbol{B}_\omega\big(\boldsymbol{x}(t)\big)\boldsymbol{\omega}(\mathrm{t})\, dt \tag{39}$$

By Euler-Maruyama discretization:

$$\boldsymbol{x}_n = \boldsymbol{x}_{n-1} + \Delta t f(\boldsymbol{x}_{n-1}) + \boldsymbol{q}_n \tag{40}$$

The process noise is:

$$\boldsymbol{q}_n \equiv \int_{t_{n-1}}^{t_n} \boldsymbol{B}_\omega\big(\boldsymbol{x}(t)\big)\boldsymbol{\omega}(\mathrm{t})\, dt \tag{41}$$

with

$$\boldsymbol{q}_n \sim N(0, \boldsymbol{Q}_n) \tag{42}$$

$$\boldsymbol{Q}_n = Cov\{\boldsymbol{q}_n\} = \int_{t_{n-1}}^{t_n} \boldsymbol{B}_\omega\left(\boldsymbol{x}(t)\right)\sigma_w^2 \boldsymbol{B}_\omega^T\big(\boldsymbol{x}(t)\big) d\tau \tag{43}$$

Using rectangle approximation, we get:

$$\boldsymbol{Q}_n = Cov\{\boldsymbol{q}_n\} \approx \Delta t \boldsymbol{B}_\omega(\boldsymbol{x}_{n-1})\sigma_w^2 \boldsymbol{B}_\omega(\boldsymbol{x}_{n-1})^T \tag{44}$$

The discrete time state-space model is:

$$\begin{cases} \begin{bmatrix} p_n^x \\ p_n^y \\ v_n \\ \varphi_n \end{bmatrix} = \begin{bmatrix} p_{n-1}^x \\ p_{n-1}^y \\ v_{n-1} \\ \varphi_{n-1} \end{bmatrix} + \begin{bmatrix} \Delta t v_{n-1} \cos(\varphi_{n-1}) \\ \Delta t v_{n-1} \sin(\varphi_{n-1}) \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{q}_n \\ \boldsymbol{y}_n = \boldsymbol{G}_n \boldsymbol{x}_n + \boldsymbol{r}_n \end{cases} \tag{45}$$

with

$$E\{\boldsymbol{q}_n\} = 0, Cov\{\boldsymbol{q}_n\} = \boldsymbol{Q}_n \tag{46}$$

$$E\{\boldsymbol{r}_n\} = 0, Cov\{\boldsymbol{r}_n\} = \boldsymbol{R}_n \tag{47}$$

4. Simulation of Vehicle Detection and Tracking

4.1 Simulation of Vehicle Detection

The aim of vehicle detection is to extract the vehicle from the video and locate it within the image. The traffic video is a simulation of the traffic scene at a crossroad recorded by a driving simulator called BeamNG.drive. In the 5-second video, the detection target is a single car, and it is making a turn from the top right side to the bottom left side. There are no pedestrians, bikes, or other vehicles passing through the region of detection.

4.1.1 Generation of Image Sequence

In order to conduct vehicle detection, the video is converted into an image sequence. Set the frame rate to be 10fps, 52 frames are generated. Figure 4.1 shows a subset of the frames.

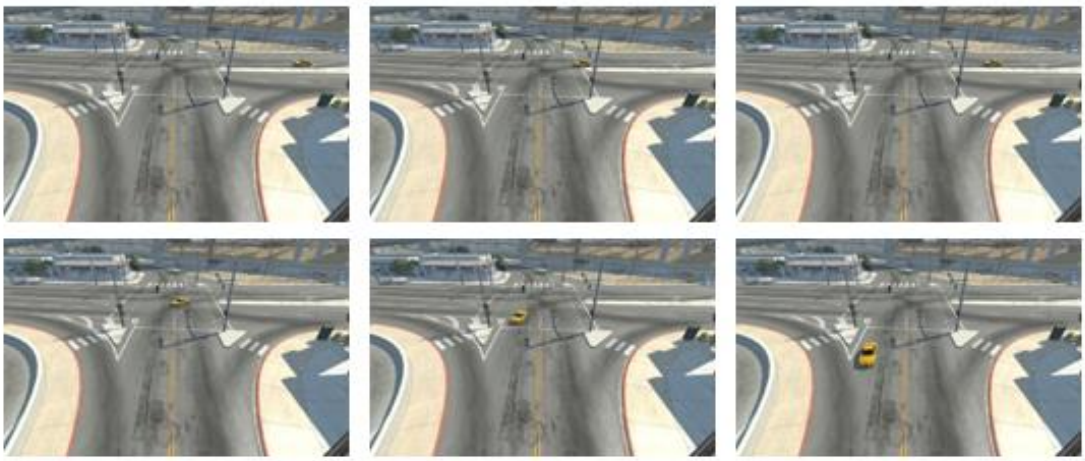


Figure 4.1: Frame 1, frame 10, frame 20, frame 30, frame 40 and frame 50 in the image sequence of the traffic scene

4.1.2 Adoption of YOLO Algorithm

YOLO is a network inspired by GoogleNet. It has twenty-four convolutional layers working as feature extractors and two dense layers to make predictions. The framework is called Darknet, which is also created by Redmon et al. Darknet uses mostly 3×3 filters to extract features, 1×1 filters to reduce output channels, and a global average pooling to make predictions (Hui 2018). The flowchart of YOLO network with Darknet is shown in Figure 4.2.

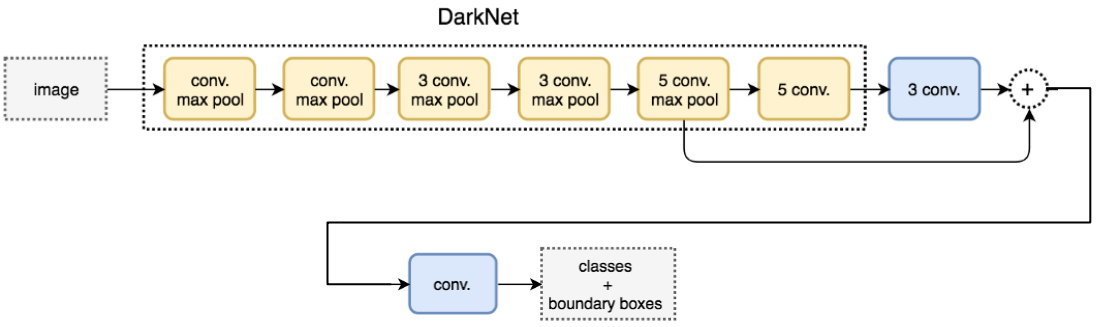


Figure 4.2: Flowchart of YOLO network with Darknet (Hui 2018)

The algorithm used in the simulation is YOLO v3 from open source. In YOLO v3, a new 53-layer Darknet-53 is used as the feature extractor, instead of the 19-layer Darknet-19 used in YOLO v2. Darknet-53 mainly consists of 3 × 3 and 1× 1 filters with skip connections similar to the residual network in ResNet. Darknet-53 has less billion floating point operations than ResNet-152, but it is two times faster with the same classification accuracy. (Redmon et al. 2018)

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4.3: Darknet-19 in YOLO v2 (left, Redmon et al. 2017)
and Darknet-53 in YOLO v3 (right, Redmon et al. 2018)

The simulation uses the pre-trained weights for object detection. The model is trained on COCO dataset, which includes 123287 images and 886284 instances. The results are divided into 80 categories shown in Figure 4.4.

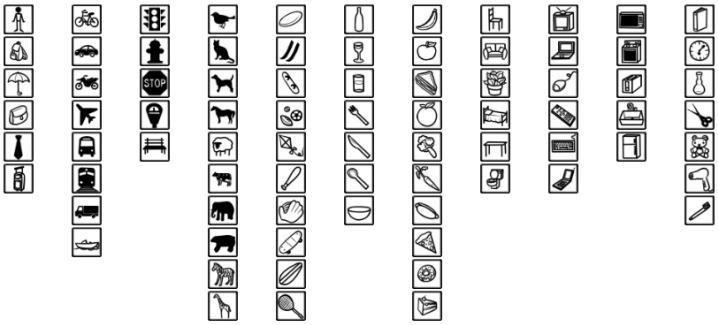


Figure 4.4: 80 catagories in COCO dataset (http://cocodataset.org)

Detect objects with YOLO v3 algorithm using the pre-trained weights in each frame by running the following command:
darknet.exe detect cfg/yolov3.cfg yolov3.weights

The default input image size of YOLO v3 is 416 × 416 pixels, which affects accuracy and computational load. Real-time detection can be achieved with higher input sizes, for example 608×608 pixels. The default detection threshold is 0.25, which affects the output number of class categories. The threshold can be adjusted according to the needs. Default values are used here for vehicle detection.

Enter the file path of every frame, and Darknet prints out the object categories, the confidence, and the time it takes to find them. Figure 4.5 shows the detection results corresponding to the frames in Figure 4.1.

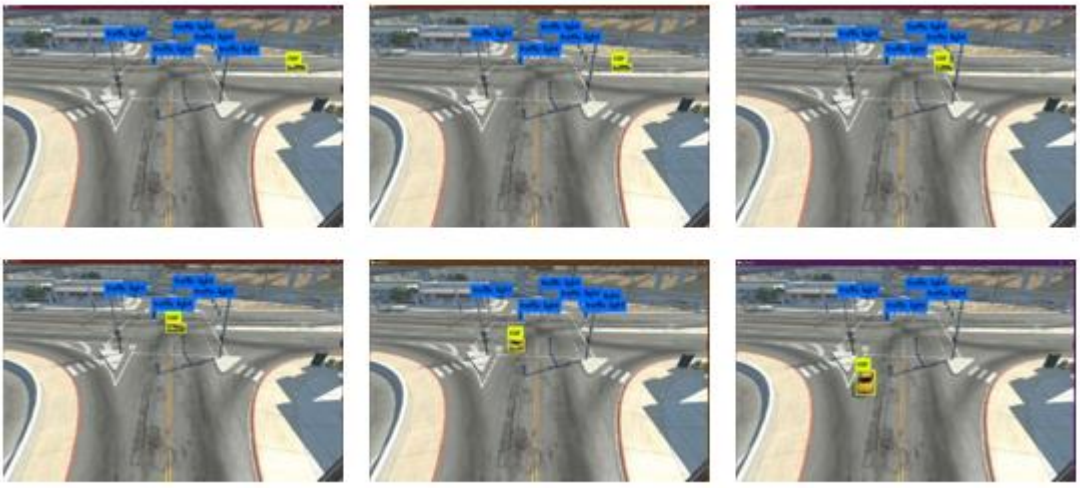


Figure 4.5: Object detection in frame 1, frame 10, frame 20, frame 30, frame 40 and frame 50 using YOLO algorithm

The vehicle can be detected in all the images, except that in the 18th image, it is blocked by the traffic lights.

4.1.3 Statistic Analysis

Select the bounding box of the vehicle in each frame and extract the left, top, right, bottom coordinates of the bounding box. The pixel location of the vehicle center can be obtained in each frame, as shown in Table 4.1. In the table, left, top, right, bottom are the offset values of the bounding box, c_x is the x-coordinate of the bounding box center, and c_y is the y-coordinate of the bounding box center.

Table 4.1: Characteristic coordinates of the vehicle in each frame

	Left	Top	Right	Bottom	c_x	c_y
1	1398	259	1503	296	1450	277
2	1375	259	1482	295	1428	277
3	1357	260	1460	295	1408	277
4	1335	261	1435	295	1385	278
5	1313	262	1410	296	1361	279
6	1286	261	1392	296	1339	278
7	1267	260	1367	295	1317	277
8	1243	260	1344	296	1293	278
9	1218	259	1318	296	1268	277
10	1197	259	1298	296	1247	277
11	1175	259	1271	296	1223	277
12	1150	256	1247	297	1198	276
13	1126	258	1225	295	1175	276
14	1110	257	1204	295	1157	276
15	1105	254	1182	294	1143	274
16	1063	256	1161	296	1112	276
17	1041	259	1144	294	1092	276
18	\	\	\	\	\	\
19	1011	261	1083	289	1047	275
20	981	258	1078	298	1029	278
21	963	259	1053	300	1008	279
22	945	256	1043	302	994	279
23	923	265	1026	305	974	285
24	903	265	1000	307	951	286
25	885	266	983	310	934	288
26	869	269	968	313	918	291

27	853	270	948	318	900	294
28	834	275	929	321	881	298
29	820	279	915	329	867	304
30	804	282	905	331	854	306
31	795	286	891	338	843	312
32	780	292	874	342	827	317
33	766	299	860	351	813	325
34	749	304	845	358	797	331
35	745	310	824	364	784	337
36	724	316	818	375	771	345
37	709	322	807	382	758	352
38	705	327	792	392	748	359
39	692	340	779	405	735	372
40	682	348	765	414	723	381
41	677	358	754	427	715	392
42	664	371	743	445	703	408
43	653	380	732	460	692	420
44	647	395	723	472	685	433
45	631	406	716	497	673	451
46	629	411	713	513	671	462
47	625	438	703	537	664	487
48	603	450	704	565	653	507
49	598	479	699	586	648	532
50	589	500	690	623	639	561
51	585	530	688	660	636	595
52	575	555	682	703	628	629

4.2 Simulation of Vehicle Tracking

Vehicle tracking aims to estimate the motion parameters, calculate the corresponding trajectories, and predict the upcoming position of vehicles in the image. Since the measurement points of the sensor are obtained in the "c_x" and "c_y" columns in Table 4.1, they can be put into the mathematical model of the vehicle to realize vehicle tracking. Three methods are compared here, Kalman filter method, extended Kalman filter method, and particle filter method.

The derivation of equations takes reference of the work by Fredrik Gustafsson (2018).

4.2.1 Kalman Filter Method

Kalman filter method iterates two steps for all measurement points in sequence. The first step is the prediction, which aims to predict the current state using the dynamic model. The second step is measurement update, which aims to estimate the current state using the prediction and the new measurement.

The objective is to estimate the current state $\boldsymbol{x}_n$ given the new measurement $\boldsymbol{y}_n$, taking the prediction into account.

$\widehat{\boldsymbol{x}}_{\mathrm{n|n}}$ denotes the estimated value at t_n, given the measurements $\boldsymbol{y}_{1:n} = \{y_1, y_2, \dots, y_n\}$,

$\mathbf{P}_{\mathrm{n|n}}$ denotes the covariance at t_n.

The initial condition is $\widehat{\boldsymbol{x}}_{0|0} = \boldsymbol{m}_0,\ \boldsymbol{P}_{0|0} = \boldsymbol{P}_0$.

For n=1,2, …
The prediction is:

$$\widehat{\boldsymbol{x}}_{\mathrm{n|n-1}} = \boldsymbol{F}_n \widehat{\boldsymbol{x}}_{\mathrm{n-1|n-1}} \tag{48}$$

$$\mathbf{P}_{\mathrm{n|n-1}} = \boldsymbol{F}_n \mathbf{P}_{\mathrm{n-1|n-1}} \boldsymbol{F}_n^T + \boldsymbol{Q}_n \tag{49}$$

Using regularized squares as the estimation algorithm to estimate$\boldsymbol{x}_n$:

$$\mathrm{J_{RELS}}(\boldsymbol{x}_n) = (\boldsymbol{y}_n - \boldsymbol{G}_n \boldsymbol{x}_n)^T \boldsymbol{R}_n^{-1} (\boldsymbol{y}_n - \boldsymbol{G}_n \boldsymbol{x}_n) + \left(\boldsymbol{x}_n - \widehat{\boldsymbol{x}}_{\mathrm{n|n-1}}\right)^{\mathrm{T}} \mathbf{P}_{\mathrm{n|n-1}} \left(\boldsymbol{x}_n - \hat{\mathrm{x}}_{\mathrm{n|n-1}}\right) \tag{50}$$

and solve

$$\widehat{\boldsymbol{x}}_{n|n} = \underset{\boldsymbol{x}_n}{argmin}\, J_{RELS}(\boldsymbol{x}_n) \tag{51}$$

The measurement update is:

$$\mathbf{K}_{\mathrm{n}} = \mathbf{P}_{\mathrm{n|n-1}} \boldsymbol{G}_n^T \left(\boldsymbol{G}_n \mathbf{P}_{\mathrm{n|n-1}} \boldsymbol{G}_n^T + \boldsymbol{R}_n\right)^{-1} \tag{51}$$

$$\widehat{\boldsymbol{x}}_{n|n} = \widehat{\boldsymbol{x}}_{\mathrm{n|n-1}} + \mathbf{K}_{\mathrm{n}} \left(\boldsymbol{y}_n - \boldsymbol{G}_n \widehat{\boldsymbol{x}}_{\mathrm{n|n-1}}\right) \tag{52}$$

$$\mathbf{P}_{\mathrm{n|n}} = \mathbf{P}_{\mathrm{n|n-1}} - \mathbf{K}_{\mathrm{n}} \left(\boldsymbol{G}_n \mathbf{P}_{\mathrm{n|n-1}} \boldsymbol{G}_n^T + \boldsymbol{R}_n\right) \mathbf{K}_{\mathrm{n}}^{\mathrm{T}} \tag{53}$$

where $\mathbf{K}_{\mathrm{n}}$is the Kalman gain.

4.2.2 Extended Kalman Filter Method

Extended Kalman Filter first linearizes the dynamic model, then it also iterates the same two steps for all measurement points in sequence. The objective is also to estimate the current state $\boldsymbol{x}_n$ given the new measurement $\boldsymbol{y}_n$, taking the prediction into account.

The initial condition is $\hat{\boldsymbol{x}}_{0|0} = \boldsymbol{m}_0,\ \boldsymbol{P}_{0|0} = \boldsymbol{P}_0$.

First, the dynamic model is linearized around $\hat{\boldsymbol{x}}_{\mathrm{n-1|n-1}}$:

$$\boldsymbol{x}_n = f(\boldsymbol{x}_{n-1}) + \boldsymbol{q}_n \approx f\left(\hat{\boldsymbol{x}}_{n-1|n-1}\right) + \boldsymbol{F}_x\left(\boldsymbol{x}_{n-1} - \hat{\boldsymbol{x}}_{n-1|n-1}\right) + \boldsymbol{q}_n \tag{54}$$

For n=1,2, …
The prediction is:

$$\hat{\boldsymbol{x}}_{n|n-1} = E\{\boldsymbol{x}_n|\boldsymbol{y}_{1:n-1}\}$$

$$\approx E\left\{f\left(\hat{\boldsymbol{x}}_{n-1|n-1}\right) + \boldsymbol{F}_x\left(\boldsymbol{x}_{n-1} - \hat{\boldsymbol{x}}_{n-1|n-1}\right) + \boldsymbol{q}_n\middle|\boldsymbol{y}_{1:n-1}\right\} = f\left(\hat{\boldsymbol{x}}_{n-1|n-1}\right) \tag{55}$$

$$\boldsymbol{P}_{n|n-1} = E\left\{\left(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1}\right)\left(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1}\right)^T\middle|\boldsymbol{y}_{1:n-1}\right\}$$

$$= \boldsymbol{F}_x\boldsymbol{P}_{n-1|n-1}\boldsymbol{F}_x^T + \boldsymbol{Q}_n \tag{56}$$

Using regularized squares as the estimation algorithm to estimate$\boldsymbol{x}_n$:

$$\mathrm{J_{RELS}}(\boldsymbol{x}_n) = (\boldsymbol{y}_n - \boldsymbol{G}_n\boldsymbol{x}_n)^T\boldsymbol{R}_n^{-1}(\boldsymbol{y}_n - \boldsymbol{G}_n\boldsymbol{x}_n) + \left(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{\mathrm{n|n-1}}\right)^{\mathrm{T}}\mathbf{P}_{\mathrm{n|n-1}}\left(\boldsymbol{x}_n - \hat{\mathrm{x}}_{\mathrm{n|n-1}}\right) \tag{57}$$

and solve

$$\hat{\boldsymbol{x}}_{n|n} = \underset{\boldsymbol{x}_n}{argmin}\, J_{RELS}(\boldsymbol{x}_n) \tag{58}$$

The measurement update is:

$$\mathbf{K}_{\mathrm{n}} = \mathbf{P}_{\mathrm{n|n-1}}\boldsymbol{G}_n^T\left(\boldsymbol{G}_n\mathbf{P}_{\mathrm{n|n-1}}\boldsymbol{G}_n^T + \boldsymbol{R}_n\right)^{-1} \tag{59}$$

$$\hat{\boldsymbol{x}}_{n|n} = \hat{\boldsymbol{x}}_{\mathrm{n|n-1}} + \mathbf{K}_{\mathrm{n}}\left(\boldsymbol{y}_n - \boldsymbol{G}_n\hat{\boldsymbol{x}}_{\mathrm{n|n-1}}\right) \tag{60}$$

$$\mathbf{P}_{\mathrm{n|n}} = \mathbf{P}_{\mathrm{n|n-1}} - \mathbf{K}_{\mathrm{n}}\left(\boldsymbol{G}_n\mathbf{P}_{\mathrm{n|n-1}}\boldsymbol{G}_n^T + \boldsymbol{R}_n\right)\mathbf{K}_{\mathrm{n}}^{\mathrm{T}} \tag{61}$$

where $\mathbf{K}_{\mathrm{n}}$is the Kalman gain.

4.2.3 Particle Filter Method

The particle filter method also iterates the same two steps, prediction and measurement update. However, there are J sets of states sampled instead of one. And the particles in each set are evaluated respectively.

The initial condition is $\boldsymbol{x}_0^j \sim N(\boldsymbol{m}_0, \boldsymbol{P}_0)$.

For n=1,2, …

In prediction, the simulated states $\boldsymbol{x}_{n-1}^j \ (j = 1,2, \dots J)$ are given. And $\boldsymbol{x}_n^j \ (j = 1,2, \dots J)$ is obtained by simulating from t_{n-1} to t_n.

For $j = 1,2, \dots J$, Sample $\boldsymbol{q}_n^j \sim N(0, \boldsymbol{Q}_n)$ and propagate particles:

$$\boldsymbol{x}_n^j = \boldsymbol{F}_n \boldsymbol{x}_{n-1}^j + \boldsymbol{q}_n^j \quad (j = 1,2, \dots J) \tag{62}$$

In measurement update, $\boldsymbol{x}_n^j$ is evaluated by how well it explains $\boldsymbol{y}_n$ and assigned to a weight.

$$\widetilde{\omega}_n^j = p\left(\boldsymbol{y}_n \middle| \boldsymbol{x}_n^j\right) = N\left(\boldsymbol{y}_n; \boldsymbol{G}_n \boldsymbol{x}_n^j, \boldsymbol{R}_n\right)(j = 1,2, \dots J) \tag{63}$$

Normalize the particle weights:

$$\omega_n^j = \frac{\widetilde{\omega}_n^j}{\sum_{i=1}^J \widetilde{\omega}_n^j}(j = 1,2, \dots J) \tag{64}$$

And calculate $\hat{\boldsymbol{x}}_{n|n}$, $\mathbf{P}_{\mathrm{n|n}}$.

To prevent particle degeneracy after a few samples, the particles are resampled by removing samples with low weights and replicate samples with high weights so that:

$$\Pr\{\tilde{\boldsymbol{x}}_n^i = \boldsymbol{x}_n^j\} = \omega_n^j \tag{65}$$

4.3 Simulation Result

Using Kalman filter method, extended Kalman filter method and particle filter method respectively. For Kalman filter method and particle filter method, Wiener velocity model is adopted; and for extended Kalman filter method, quasi-constant turn model is adopted. The measurement noise covariance is set as $1 \times eye(2)$. The variance of stochastic vector is set as 500. For particle filter method, the number of particles J is set to be 1000.

The code of applying particle filter for tracking simulation is shown in appendix 4, and the code of applying Kalman filter and extended Kalman filter are later shown in chapter 5.2. The simulation results of vehicle detection and tracking of the three methods are shown in Figure 4.6-4.8.

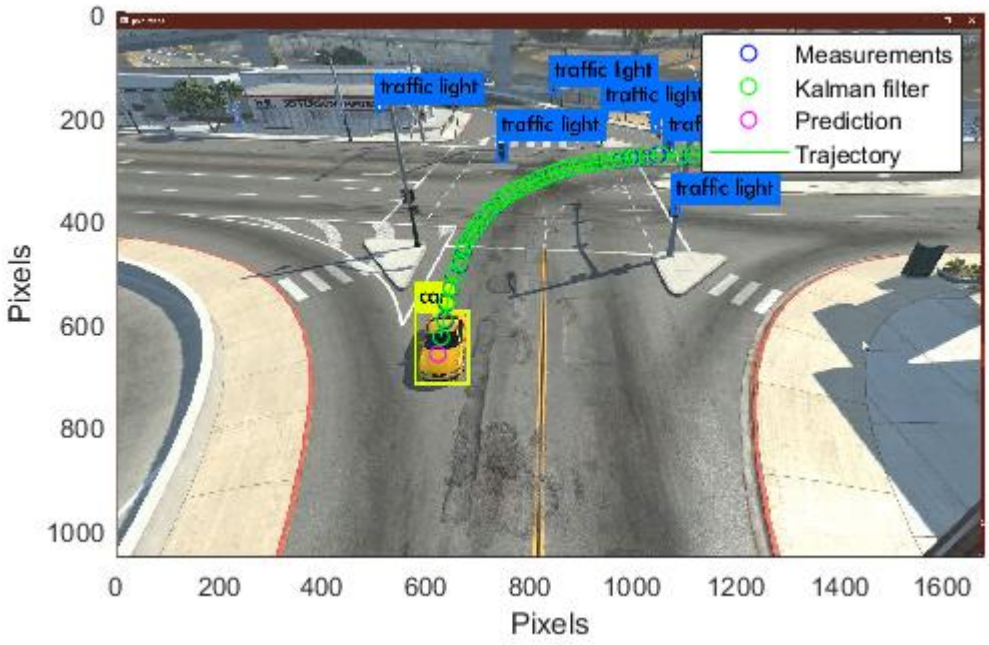


Figure 4.6: Detection and tracking result of Kalman filter method

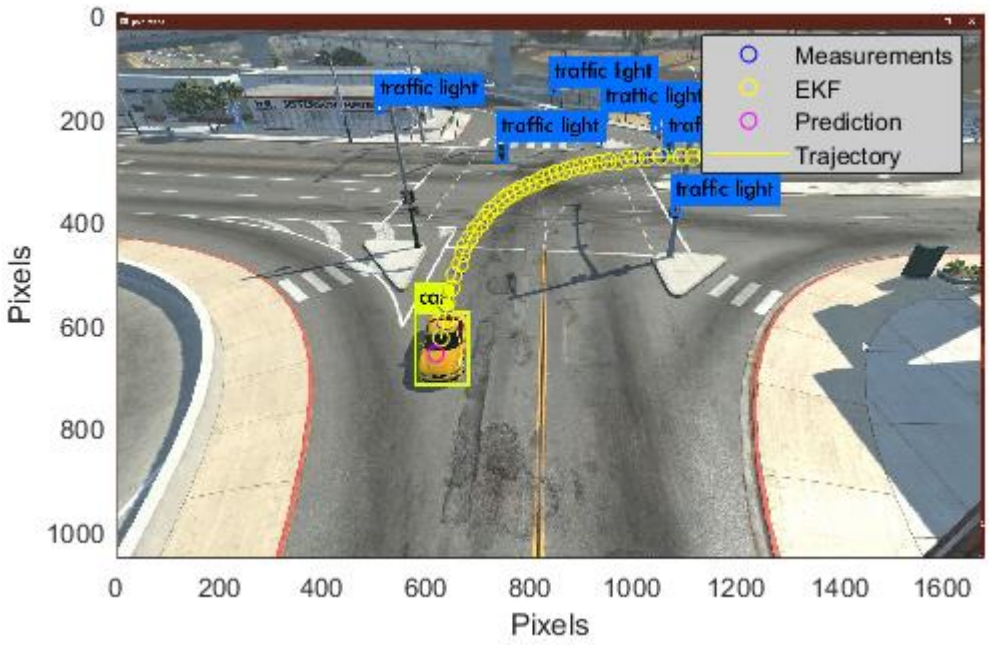


Figure 4.7: Detection and tracking result of extended Kalman filter method

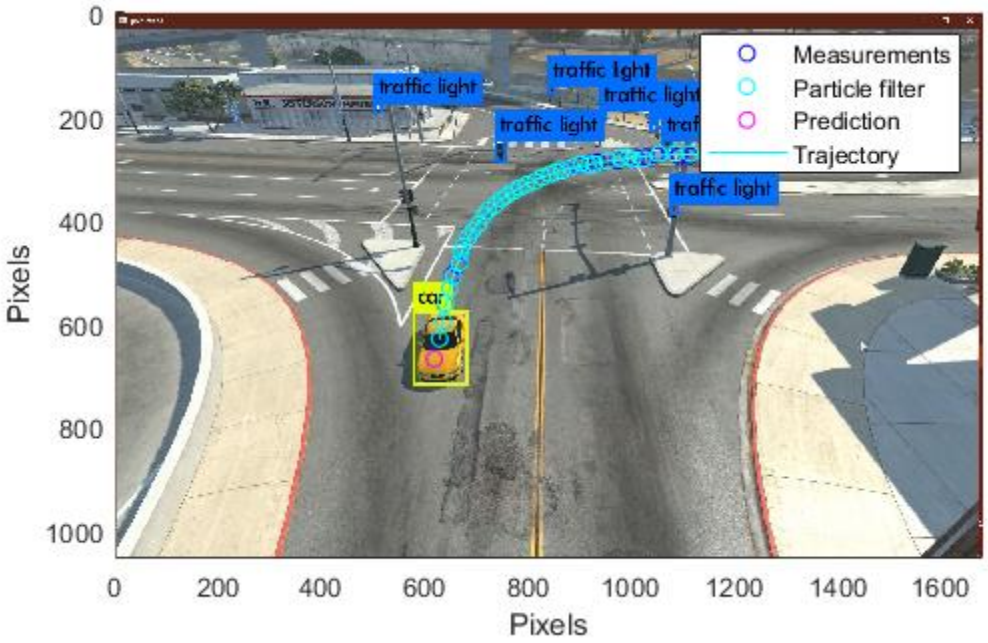


Figure 4.8: Detection and tracking result of Particle filter method

The performance of the two methods can be assessed in the following two ways. The first way is to measure the precision of the data. This can be realized by calculating the root mean square error (RMSE):

$$e_{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\hat{x}_{n|n} - x_n\right)^T\left(\hat{x}_{n|n} - x_n\right)} \tag{66}$$

where $\hat{x}_{n|n}$ is the estimated value and $x_n$is the measurement value.

Since the detection result of YOLO algorithm can be bouncing around the target due to the noise factors, the second way is to measure the fluctuation degree of the data. In order to do this, the points on the fitting trajectory are collected with the same x-coordinates of the estimated value, and the fluctuation error is calculated by:

$$e_{fluc} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\hat{x}_{n|n} - x_{n'}\right)^T\left(\hat{x}_{n|n} - x_{n'}\right)} \tag{67}$$

where $x_{n'}$is the point collected from the fitting trajectory.

The RMSE and fluctuation error of Kalman filter method (KF), extended Kalman filter method (EKF) and particle filter method (PF) are shown in Table 4.2, and translated into Figure 4.9 for analysis.

Table 4.2 RMSE and fluctuation error of different methods

	KF	EKF	PF
e_{RMSE} (pixels)	2.53	2.59	3.29
e_{fluc} (pixels)	1.61	3.01	2.58

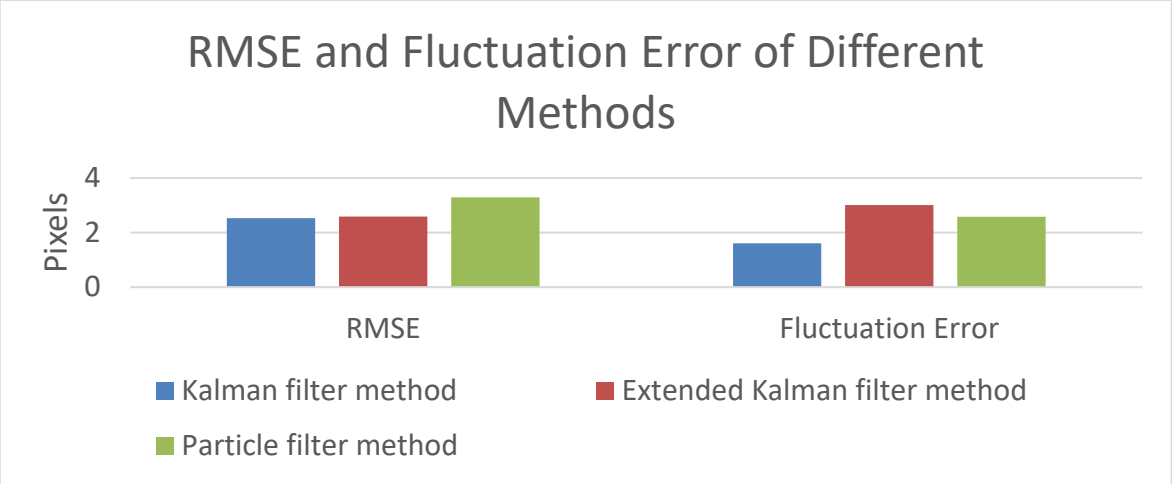


Figure 4.9: Chart of RMSE error and fluctuation error of different methods

It can be seen that the errors of the three methods are of the same order of magnitude. For RMSE error, Kalman filter method and extended Kalman filter method are more accurate than particle filter method. For fluctuation error, Kalman filter method surpasses the other two methods, which means it is less likely to be affected by the noise factors. Particle filter method, however, conducts numerical calculation on each step instead of the estimation algorithm, so it has lower computational complexity. Particle filter method also overcomes the constraint of a single Gaussian distribution of Kalman filter and extended Kalman filter methods (Liu et al. 2013). However, in the simulation, only one set of data is collected by the road camera, which could not sufficiently describe the probability density distribution. Therefore, Kalman filter method and extended Kalman filter method are more feasible than particle filter method, and they are selected as the vehicle tracking method in the practical experiment.

5. Practical Experiment

The simulation result is quite promising. However, in real life, the case is much more complicated. It is a challenge to deal with the occlusion problem of multiple targets in a scene and the data correlation among them (Liu et al. 2013). Besides, the traffic scenes in real life can be very complicated, many factors such as shadow, reflection and weather conditions could affect the result of vehicle detection and tracking. These require higher robustness and adaptive ability.

5.1Experiment of Vehicle Detection

The video is recorded by the test camera installed towards a crossroad at Aalto University. Cars, trucks, pedestrians, bikes pass through the region of detection from time to time, and there is a construction site which blocks part of the view of the objects behind. A video clip of 40 seconds is intercepted and set as the experiment target.

5.1.1 Real-Time Vehicle Detection

Real-time detection is achieved by combing Darknet and OpenCV. Run the following command:
darknet.exe detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <traffic video>

YOLO displays the detection categories, class probability, current FPS on the command window, together with the image with bounding boxes.

The detection frame rate depends on the computational capability of the computer. The computer specifications are listed in Table 5.1.

Table 5.1: Computer specifications of real-time detection

System	Specification
CPU	Intel Core i7-7700 3.6Ghz
RAM	32Gb DDR4 2666Mhz
GPU	Nvidia Geforce GTX 1080
Software	Nvidia CUDA Toolkit 10.0

As a result, 1227 frames are captured during the process, showing object categories such as cars, persons, trucks, and traffic lights. The average frame rate of the 1227 frames is 42.5 fps, which means real-time detection is achieved.

Though the detection frame rate is time-varying, it is fluctuating around the average within a narrow range. To simplify the calculation, the sampling rate is set as 42.5 fps. During the video clip, two trucks and seven cars are detected at the traffic scene.

5.1.2 Statistic Analysis

By adding the following line to the function "draw_detections" in the file "image.c":
Printf ("%f %f %f %f \n", b.x, b.y, b.w, b.h);

The characteristic coordinates of different objects are displayed in the command window in each frame. For example, in the first frame (shown in Figure 5.5), the coordinates are shown in Table 5.2. In the table, c_x is the x-coordinate of the bounding box center, c_y is the y-coordinate of the bounding box center, w is width of the bounding box, and h is the height of the bounding box.

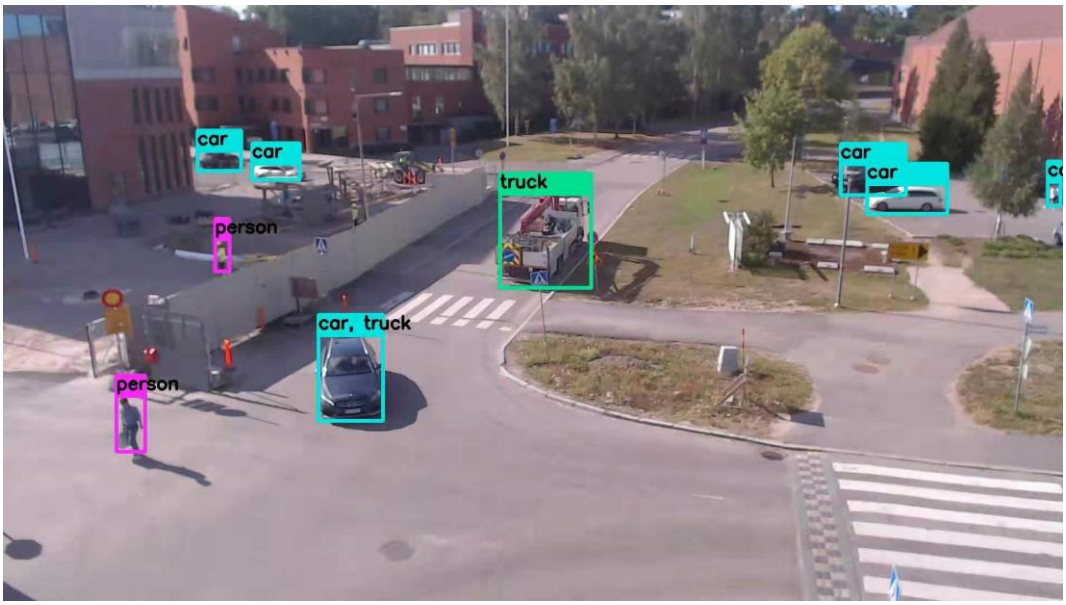


Figure 5.5: The first frame of the video clip

Table 5.2: Characteristic coordinates of frame 1

Category	Confidence	c_x	c_y	w	h
truck:	77%	0.512026	0.395793	0.086306	0.156872
car, truck:	65%, 57%	0.328256	0.625595	0.060951	0.143655
car:	85%	0.204036	0.261504	0.041046	0.032712
car:	93%	0.257539	0.279898	0.044714	0.027226
car:	44%	0.991706	0.316585	0.011802	0.040050
person:	75%	0.207041	0.419650	0.013232	0.053896
car:	88%	0.821291	0.292883	0.061517	0.051907
person:	94%	0.120525	0.702477	0.026776	0.093496
car:	99%	0.853219	0.325227	0.075767	0.050247

To get the coordinates of the vehicles, the first step is to keep the data of the vehicles including cars and trucks; and delete the data of other categories such as persons, bikes, and traffic lights. Then the static objects are removed, for example, the parking cars on the right side. Then the “cx” and “cy” columns collected in each frame composed of the measurement data.

5.2Experiment of Vehicle Tracking

Knowing the coordinates of all the vehicles in 1227 frames, the next step is to find out the trajectory of the vehicles by figuring out their correlations. The biggest challenge, as mentioned above, is to distinguish which coordinates belong to which vehicle, since the occlusion of multiple targets occurs at all times. This requires an identification algorithm, which assumes that the vehicle is the same one when the range between two measurement points is within the threshold.

5.2.1 Identification Algorithm

The idea of the algorithm is to put the initial points of different vehicles into a temporary matrix. Comparing the points with the new point in each frame, if the range of the two points is smaller than the threshold, the new point will be added below the old point in

the temporary matrix and removed from the frame matrix. Every matching string is stored as the trajectory of a car. The string is considered ended when no match could be founded in the new frame. Otherwise, the remaining points that could not find a match from previous strings in the frame will be added to the temporary matrix, made starts of new strings, and continue comparing. To get rid of accidental errors. The strings that are too short are eliminated. The flowchart of the identification algorithm is shown in Figure 5.6, and the code is shown in Appendix 1.

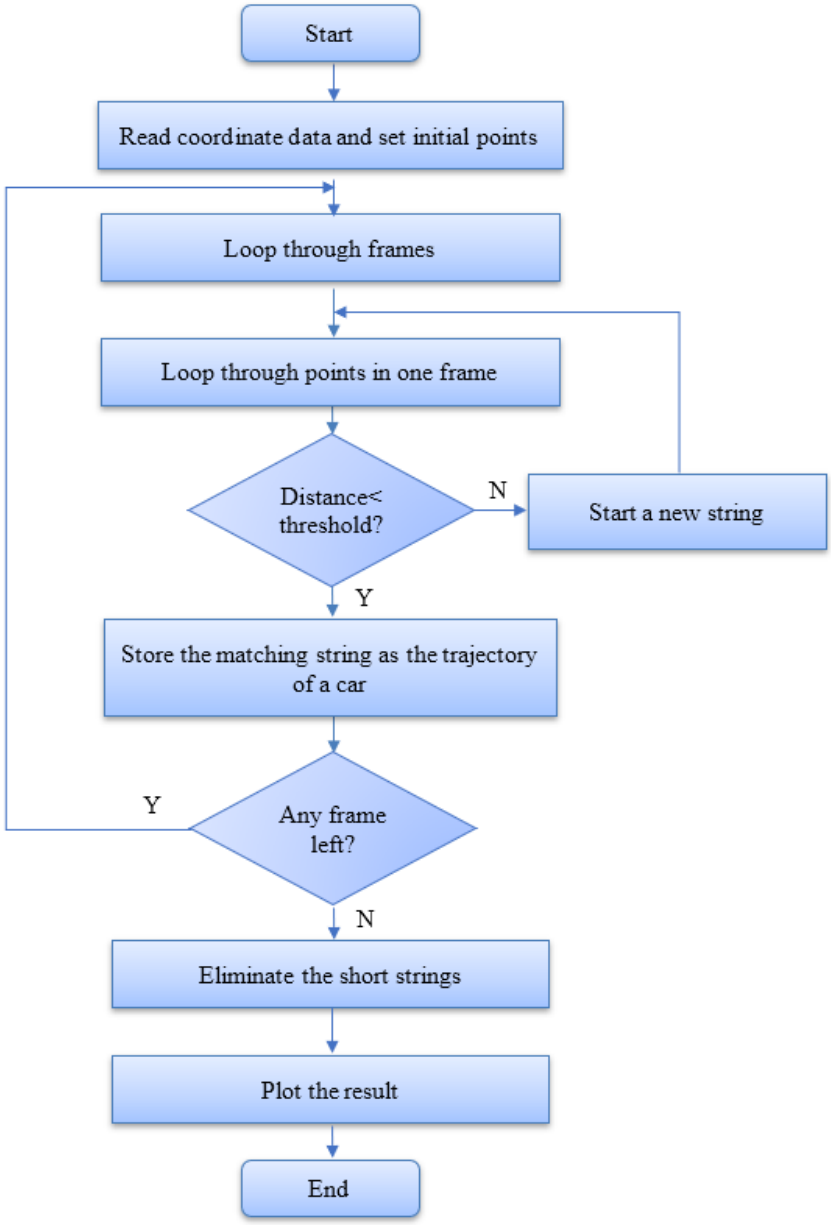


Figure 5.6: Flowchart of the identification algorithm

By running the code, point sequences are obtained in each cell of the result. In order to get the measurement points of each vehicle, Minimum length of the point sequences, as

well as the threshold of point matching, needs to be tuned. However, in some cases the point sequences are discontinuous since the detection is not always ideal; to solve the problem, the point sequences need to be stitched. The final result of the measurement points of each vehicle in saved in an Excel document.

5.2.2 Applying Filtering Algorithm

The next step is applying filters to the point sequence of each vehicle to realize tracking. Kalman Filter and extended Kalman filter are applied to calculate the trajectory and predict the upcoming position. The flowchart of applying Kalman filter and extended Kalman filter for tracking experiment is shown in Figure 5.7, and the codes are shown in Appendix 2-3. The measurement noise covariance is set as $0.1 \times eye(2)$. The variance of stochastic vector is set as 10^4.

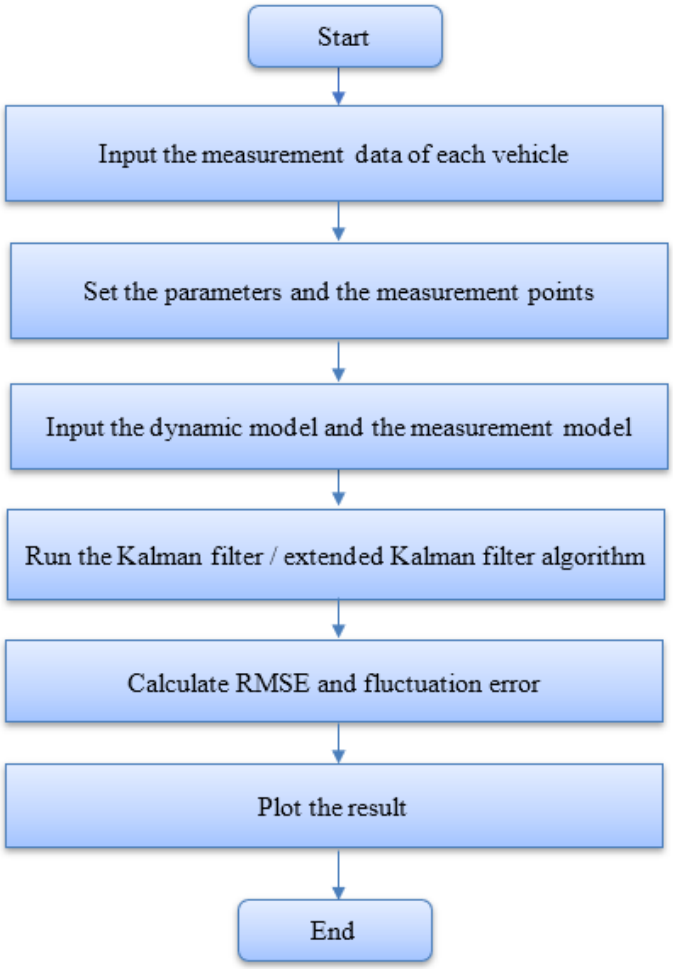
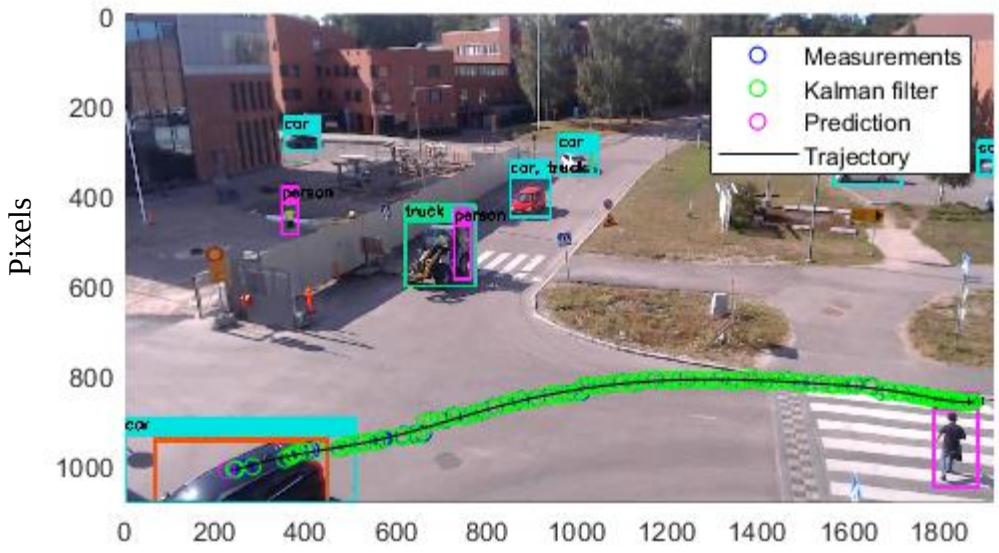


Figure 5.7: Flowchart of applying Kalman filter and extended Kalman filter for tracking experiment

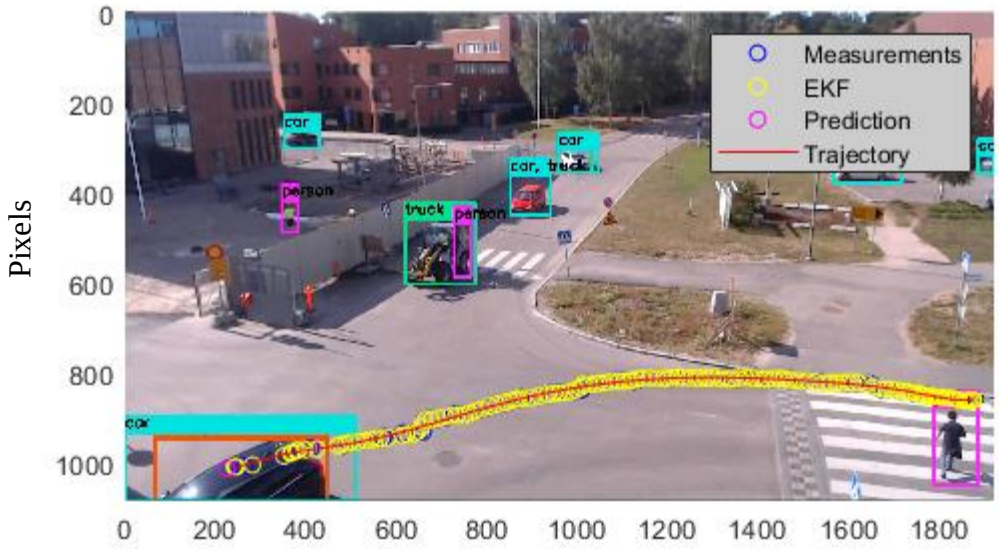
5.3 Experiment Result

5.3.1 Detection and Tracking Result

The detection and tracking results of part of the vehicles are shown in Figure 5.8-5.12.

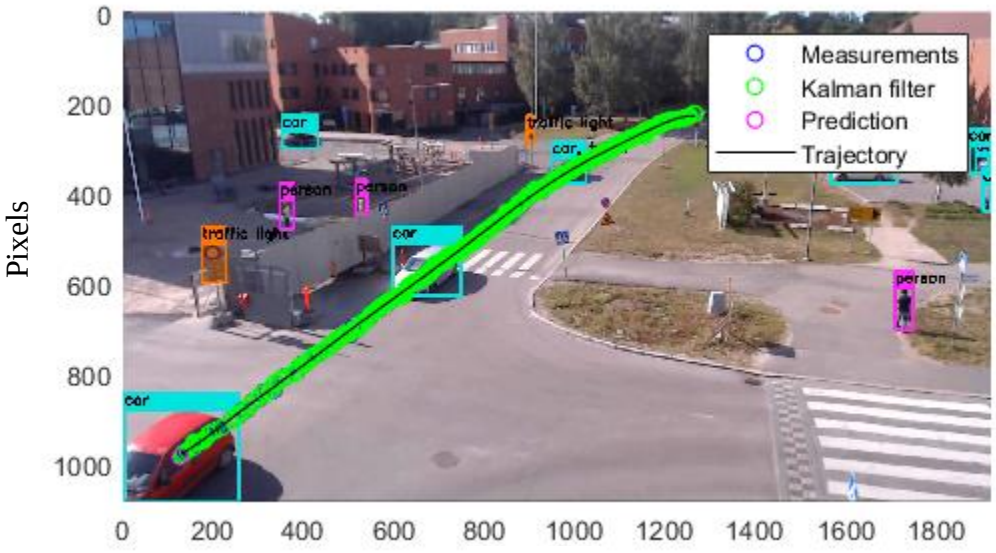


Kalman filter method

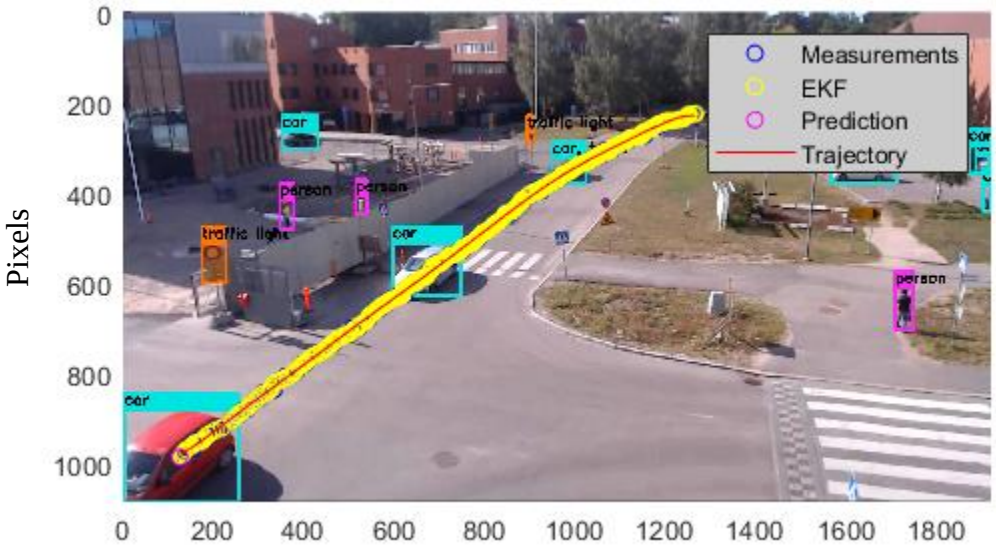


Extended Kalman filter method

Figure 5.8: detection and tracking result of Car 2

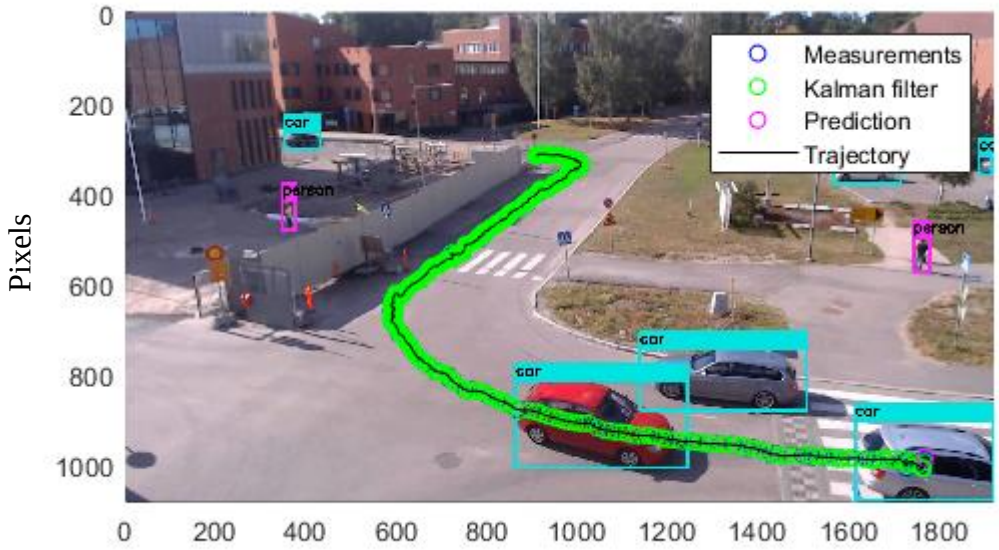


Kalman filter method

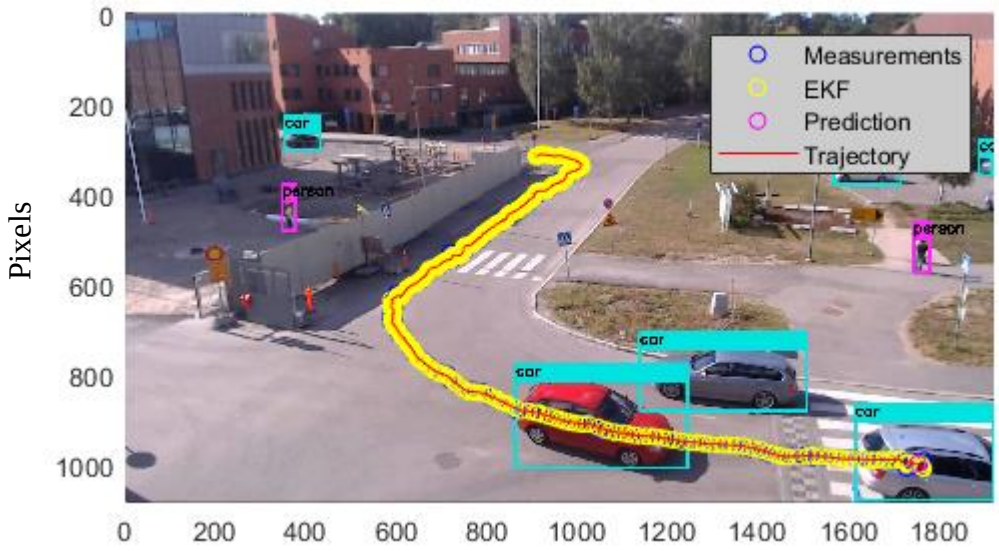


Extended Kalman filter method

Figure 5.9: Detection and tracking result of Car 3



Kalman filter method



Extended Kalman filter method

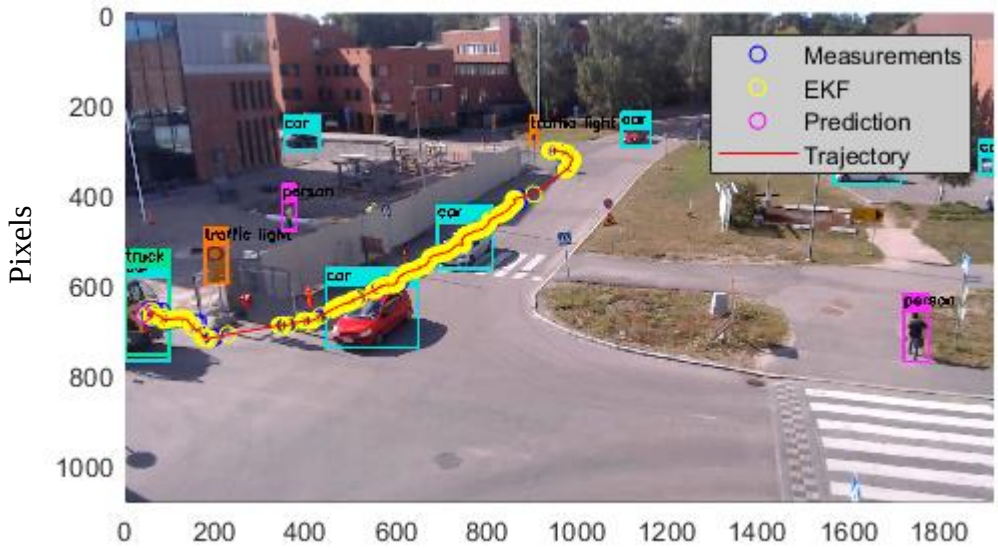
Figure 5.10: Detection and tracking result of Car 4

Kalman filter method

Extended Kalman filter method

Figure 5.11: Detection and tracking result of Car 5

Kalman filter method



Extended Kalman filter method

Figure 5.12: Detection and tracking result of Truck 2

The RMSE and fluctuation error of each vehicle are listed in Table 5.3, and translated into Figure 5.13 and Figure 5.14 for analysis.

Table 5.3: RMSE and fluctuation error of each vehicle ($|R_n| = 0.1$, $\sigma_w^2 = 10^4$)

Vehicle	KF Method		EKF Method	
	RMSE (pixels)	Fluct. Error (pixels)	RMSE (pixels)	Fluct. Error (pixels)
Car 1	2.79	3.03	3.82	3.72
Car 2	2.67	3.41	3.08	3.14
Car 3	0.92	3.06	1.84	2.78
Car 4	1.03	/	1.98	/
Car 5	1.22	/	1.89	/
Car 6	3.53	1.89	3.74	1.71
Car 7	1.55	3.26	2.33	2.97
Truck 1	1.83	2.58	3.50	3.20
Truck 2	3.12	/	4.16	/
Average	2.07	2.87	2.92	2.92

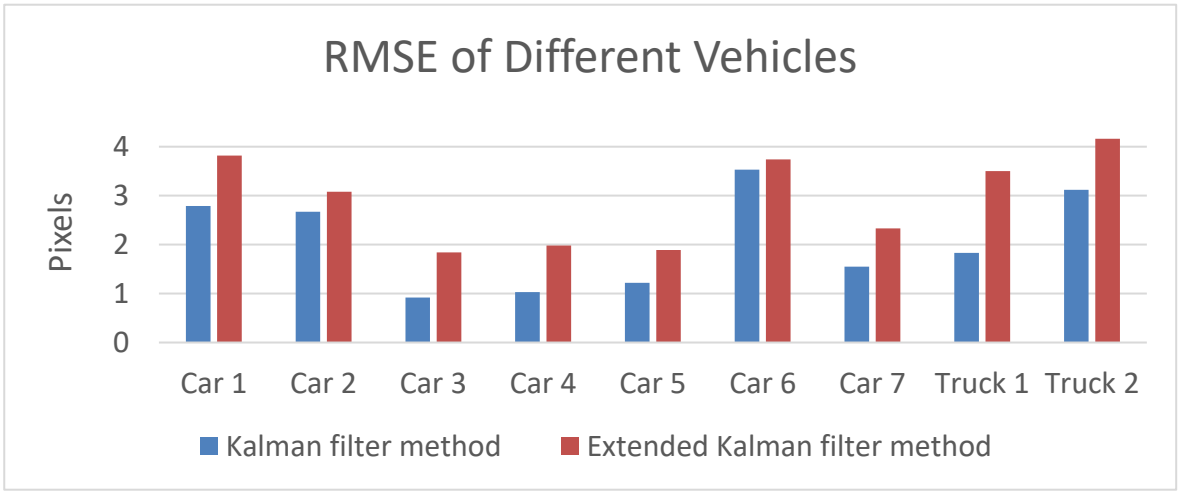


Figure 5.13: Chart of RMSE of different vehicles

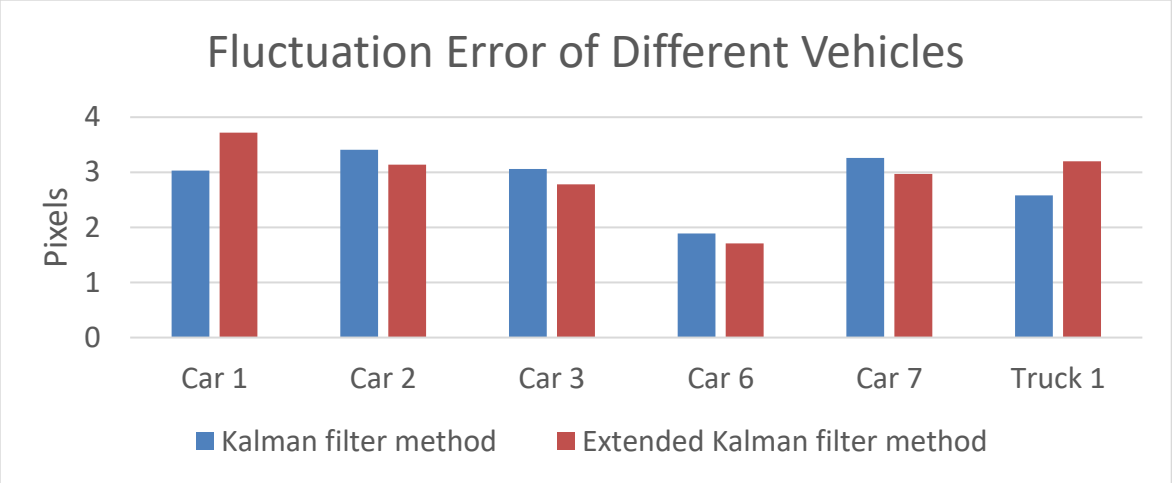
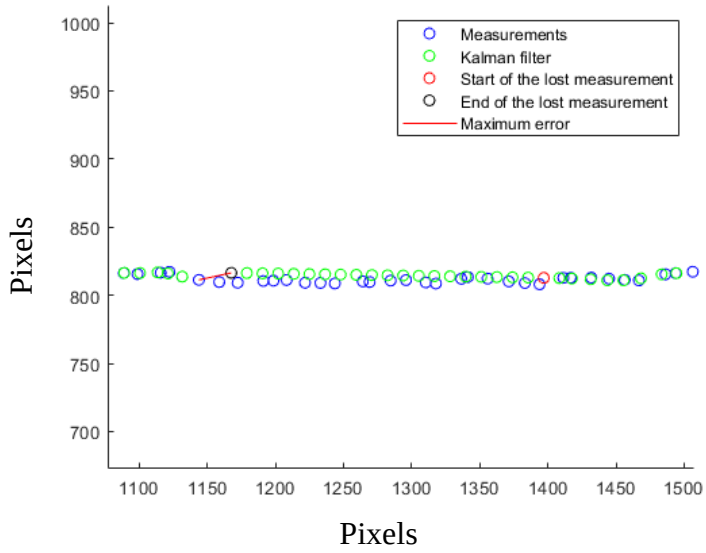


Figure 5.14: Chart of fluctuation error of different vehicles

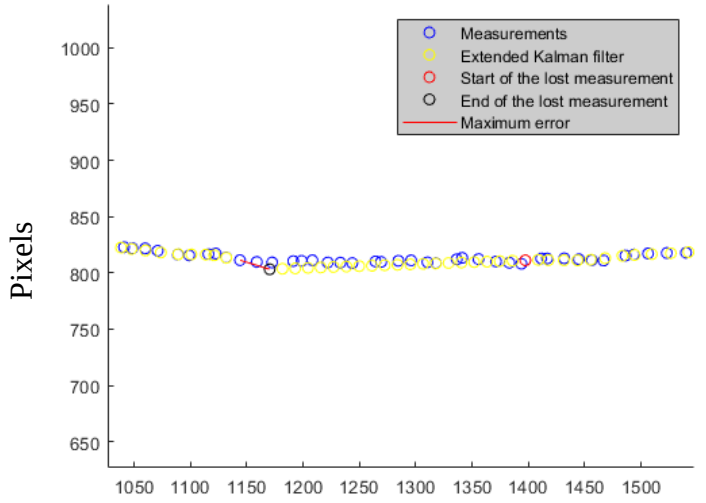
For both of the methods, the RMSE are within 4.5 pixels, and the fluctuation errors are within 4 pixels. So, the overall results achieve high accuracy and stableness. Kalman methods surpasses extended Kalman method in precision, and the two methods have similar performance in stableness.

There are a few points to note here. First, the interference factors could result in the loss of the measurement points where the machine vision is not detecting the vehicle for a while. For example, the measurement points of Truck 2 are not regular due to the occlusion by construction fence and shadows in the experiment.

This is the scenario where YOLO are not able to provide any measurements of the vehicle for a short period of time. Therefore, we would have to estimate the trajectory of the vehicle during that time. Assuming the 50^{th} to 70^{th} measurement points are lost, the tracking result for Car 2 during that period of time are shown in Figure 5.15.



Kalman filter method



Extended Kalman filter method

Figure 5.15: Tracking result for Car 2 when 50th to 70th measurement points are lost

For both Kalman filter method and extended Kalman filter method, during the period of lost measurement, the estimation points will follow the same gradient of the last measurement until it receives the new measurement. As a consequence, the fluctuation error of this period jumps to zero, but the RSME significantly increases because the trajectory becomes a line and deviates from the measurement points.

To estimate how long a vehicle can be undetected before the error becomes substantial, we set different time steps of the lost measuremt and compare the maximum error (shown in Figure 5.15) during the period of lost measurement.

Table 5.4: Maximum error during the period of lost measurement for Car 2 with different lost time steps ($|\boldsymbol{R_n}| = \mathbf{0.1}$, $\boldsymbol{\sigma_w^2} = \mathbf{10^4}$)

Lost time steps	KF Method	EKF Method
	Max error (pixels)	Max error (pixels)
50th - 55th	3.63	5.09
50th - 60th	13.96	14.79
50th - 70th	24.32	27.57
50th - 80th	30.49	41.08
50th - 90th	51.48	73.41

From the table, it can be seen when the lost measurement is around 30 time steps, the RMSE of that period exceeds 30 pixels, which is assumed to be a substantial error. The footage was recorded at 30 FPS, so it can be estimated that when the loss of measurement data achieves 1 second, the positioning error caused by the lack of measurement data could not be neglected. This prediction method is naturally very prone to errors when the vehicle is taking a turn at the intersection, so the fragment that we assume to be lost here is not at the intersection. More sophisticated prediction is required for those scenarios.

Second, polynomial fitting is used for the trajectory fitting in the experiment. In statistics, polynomial fitting is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. In the experiment, x and y denote the x-coordinate and y-coordinate of the vehicle estimated position, so that we can plot the fitting trajectory of the vehicles. We use 10 degrees of polynomial fitting here.

However, polynomial fitting does not work effectively in some complex curve. For example, polynomial fitting is not feasible for Car 4, Car 5, and Truck 2, so the trajectories for these vehicles are plotted by simply connecting the discontinuous estimation points. Since the points collected from fitting trajectory are used as the standards to calculate the fluctuation errors, the fluctuation errors of these vehicles cannot be calculated.

Third, the precision of the estimation depends on the precision of the measurement data, because they are regarded as the standard values to evaluate RMSE. However, if

fluctuation error was not considered, the filter could be tuned to perfectly copy the measurements, which would not add any value to the system. To avoid the filter mimicking the data, fluctuation error values are observed as well.

Fourth, although vehicle detection can be real-time, vehicle tracking takes time. The algorithm needs to be optimized to achieve real-time tracking with multiple targets.

5.3.2 The Effects of Noise

The noise in the experiment include the process noise $\boldsymbol{q}_n$ in the dynamic model and the measurement noise $\boldsymbol{r}_n$ in the sensor model. In order to analyze their effects, variable-controlling method is adopted here by change their covariance $\boldsymbol{Q}_n$ and $\boldsymbol{R}_n$ respectively.

First, fix the measurement noise covariance $\boldsymbol{R}_n$, change the process noise covariance $\boldsymbol{Q}_n$ and compare the tracking performance. For both of the dynamic models, the process noise covariance is proportional to σ_w^2, namely the variance of the stochastic vector. So, we change the value of σ_w^2 instead. Take Car 2 as an example, the RMSE and fluctuation errors with different σ_w^2 are shown in Table 5.5.

Table 5.5: RMSE and fluctuation error of Car 2 with different $\boldsymbol{\sigma_w^2}$ ($|\boldsymbol{R_n}| = \boldsymbol{0.1}$)

$\boldsymbol{\sigma_w^2}$	KF Method		EKF Method	
	RMSE (pixels)	Fluct. Error (pixels)	RMSE (pixels)	Fluct. Error (pixels)
100	10.03	1.92	11.88	1.18
1000	6.98	2.69	7.48	1.95
$\mathbf{10^4}$	4.81	3.18	5.09	2.62
$\mathbf{10^5}$	2.67	3.41	3.08	3.14
$\mathbf{10^6}$	0.97	3.47	1.56	3.38
$\mathbf{10^{12}}$	0	3.48	0	3.48

It can be concluded that when σ_w^2 is larger, the RMSE value becomes smaller, and the fluctuation error becomes bigger. When σ_w^2 is a large number, the RMSE value becomes zero, which means the estimation points and the measurement points coincide with each other. On the one hand, when $\boldsymbol{Q}_n$ is larger, the acceleration and heading direction can vary in a wider range, and the prediction points are closer to the measurement points, so

the estimation would be more precise assuming the measurement is precise. On the other hand, when $\boldsymbol{Q}_n$ is smaller, it would be more effective for smoothing the tracking process, so the estimation would be more stable.

Next, fix the process noise covariance $\boldsymbol{Q}_n$, change the measurement noise covariance $\boldsymbol{R}_n$ and compare the tracking performance. For the sensor model, $\boldsymbol{R}_n = |\boldsymbol{R}_n|\text{eye}(2)$. So, we change the value of $|\boldsymbol{R}_n|$ instead. Take Car 2 as an example, the RMSE and fluctuation errors with different $|\boldsymbol{R}_n|$ are shown in Table 5.6.

Table 5.6: RMSE and fluctuation error of Car 2 with different $|\boldsymbol{R}_n|$ ($\boldsymbol{\sigma_w^2 = 10^4}$)

$\|\boldsymbol{R}_n\|$	KF Method		EKF Method	
	RMSE (pixels)	Fluct. Error (pixels)	RMSE (pixels)	Fluct. Error (pixels)
$\mathbf{10^{-3}}$	0.18	3.48	0.89	3.41
$\mathbf{10^{-2}}$	0.97	3.47	1.56	3.39
0.1	2.67	3.41	3.08	3.14
1	4.81	3.18	5.09	2.62
10	7.03	2.69	7.56	1.95
$\mathbf{10^6}$	416	0	420	0

It can be concluded that when $|\boldsymbol{R}_n|$ is larger, the RMSE value becomes bigger, and the fluctuation error becomes smaller. When $|\boldsymbol{R}_n|$ is a large number, the fluctuation error value becomes zero, which means the estimation points can be joint into a perfectly smooth curve. On the one hand, when $\boldsymbol{R_n}$ is smaller, it means the measurement points are of high confidence, and the prediction points are closer to the measurement points, so the estimation would be more precise assuming the measurement is precise. On the other hand, when $\boldsymbol{R_n}$ is larger, it would be more effective for smoothing the tracking process, so the estimation would be more stable.

In conclusion, when the process noise covariance is larger, the estimation would be more precise but less stable; when the measurement noise covariance is larger, the estimation would be less precise but more stable. How to estimate the optimal process noise covariance and measurement noise covariance depend on the weights of precision and stableness.

5.4 Perspective Transformation

From the practical experiment, we have already got the pixel location of the vehicles in the image. In order to locate the vehicles on the road, the coordinates need to be transferred from the image plane to the ground plane.

5.4.1 Principle of Perspective Transformation

The principle of perspective transformation takes reference of the paper by Ojala et al. (2018).

The first step is to calculate x-coordinate from the image plane to the ground plane. The schematic diagram of calculating x-coordinate is shown in Figure 5.16. In the figure, x is the x-coordinate of the vehicle on the ground plane, h is the height of the camera, l is the distance between the lens and the sensor in the camera, s_x is the vehicle covering region of the sensor in x-direction, α is the angle between the camera and the horizontal direction, β and γ are the intersection angles shown in the figure.

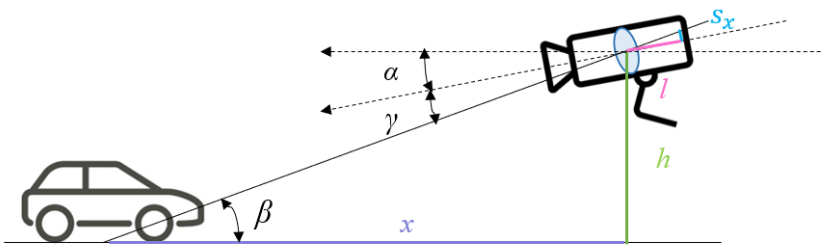


Figure 5.16: The schematic diagram of calculating x-coordinate

The derivation of x-coordinate is:

$$x = \frac{h}{tan(\beta)} = \frac{h}{tan(\alpha + \gamma)} = h\frac{1 - \tan(\alpha)\tan(\gamma)}{\tan(\alpha) + \tan(\gamma)} \tag{68}$$

$$\tan(\gamma) = \frac{s_x}{l} \tag{69}$$

The formation of image with YOLO detection is shown in Figure 5.17. In the figure, p_x is the ratio of the vehicle covering region on the sensor in x-direction and the maximum covering region of the sensor in x-direction, c_y is the y-coordinate of the center bounding box center displayed on YOLO interface, h_p is the height of the bounding box displayed on YOLO interface.

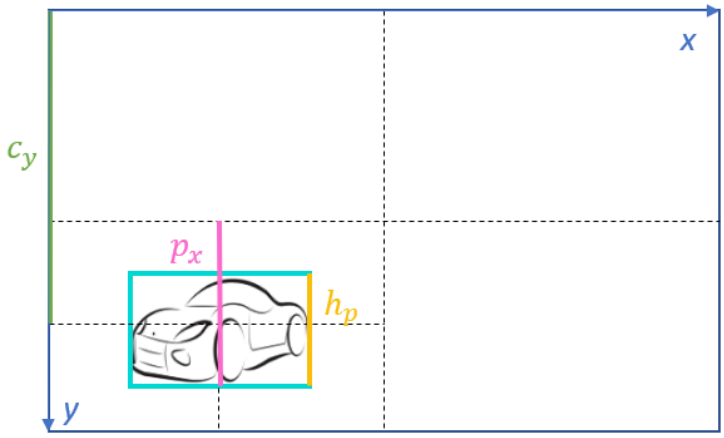


Figure 5.17: Formation of image with YOLO detection I

From the correlation in the figure we get:

$$p_x = \frac{s_x}{s_{x,max}} = c_y + \frac{1}{2}h_p - 0.5 \tag{70}$$

Measure the maximum covering region of the sensor in x-direction $s_{x,max}$ (which is equal to half the sensor height) and the distance between the lens and the sensor in the camera l respectively, denote their ratio as λ.

$$\lambda = \frac{s_{x,max}}{l} \tag{71}$$

$$\tan(\gamma) = \frac{s_x}{l} = \frac{s_{x,max}p_x}{l} = \lambda p_x \tag{72}$$

So, the x-coordinate of the vehicle on the ground plane is:

$$x = h\frac{1 - \lambda p_x \tan(\alpha)}{\tan(\alpha) + \lambda p_x} \tag{73}$$

The next step is to calculate y-coordinate from the image plane to the ground plane. The schematic diagram of calculating y-coordinate is shown in Figure 5.18. In the figure, x is the x-coordinate of the vehicle on the ground plane, y is the y-coordinate of the vehicle on the ground plane, l is the distance between the lens and the sensor in the camera, s_y is the maximum covering region of the sensor in y-direction, θ is the intersection angle shown in the figure.

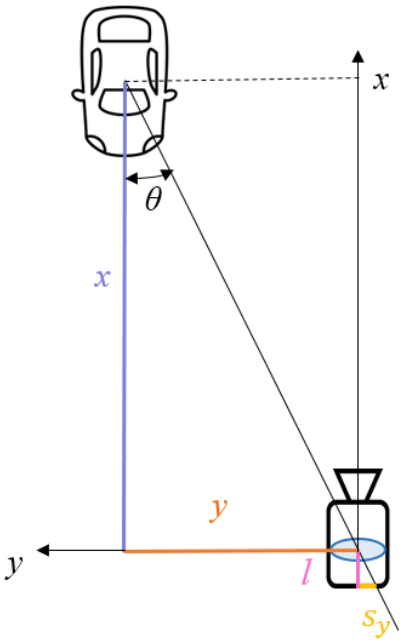


Figure 5.18: The schematic diagram of calculating y-coordinate

The derivation of y-coordinate is:

$$y = xtan(\theta) = \frac{xs_y}{l} \tag{74}$$

The formation of image with YOLO detection is shown in Figure 5.19. In the figure, p_y is the ratio of the vehicle covering region on the sensor in y-direction and the maximum covering region of the sensor in y-direction, c_x is the x-coordinate of the center bounding box center displayed on YOLO interface.

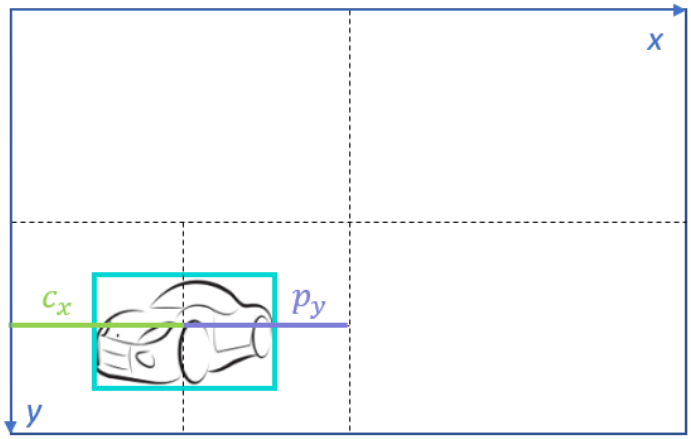


Figure 5.19: The formation of image with YOLO detection II

From the correlation in the figure we get:

$$p_y = \frac{s_y}{s_{y,max}} = -(0.5 - c_x) = c_x - 0.5 \tag{75}$$

The ratio of the maximum covering region in x-direction and y-direction is 9:16.

$$s_{y,max} = \frac{16}{9} s_{x,max} \tag{76}$$

$$s_y = p_y s_{y,max} = \frac{16}{9} p_y s_{x,max} = \frac{16}{9} p_y \lambda l \tag{77}$$

So, the y-coordinate of the vehicle on the ground plane is:

$$y = \frac{16}{9} p_y \lambda x \tag{78}$$

5.4.2 Implementation of Perspective Transformation

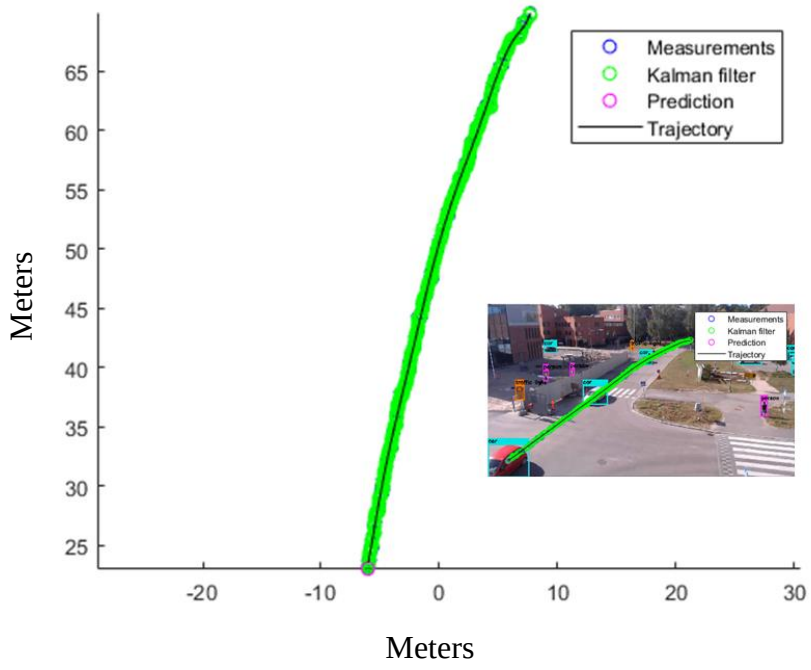
Measure the angle between the camera and the horizontal direction α, we get:

$$\alpha = 15°$$

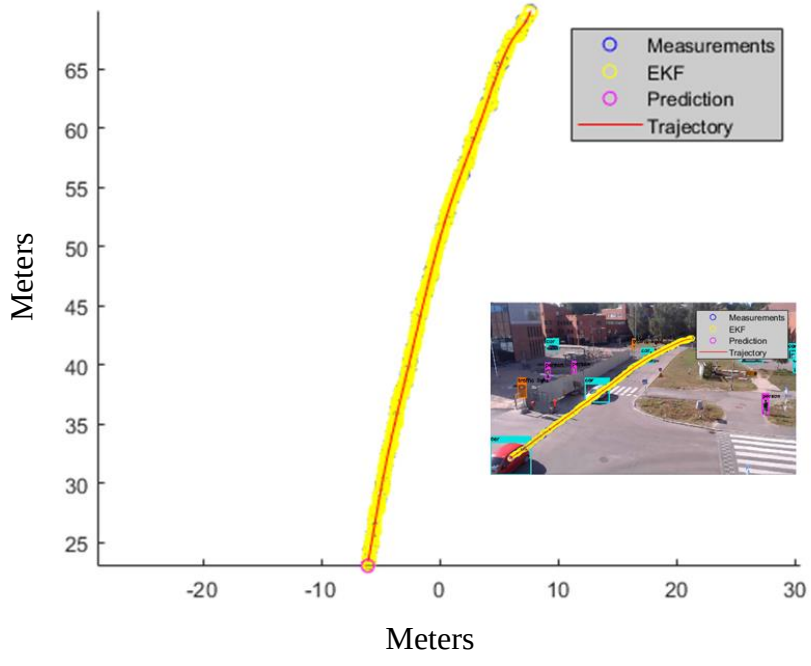
Measure the maximum covering region of the sensor in x-direction $s_{x,max}$ and the distance between the lens and the sensor in the camera l respectively, denote their ratio as λ, we get:

$$\lambda = 0.39$$

Set Car 3, Car 4, Car 5 as the targets. Substitute in α, λ values and the measurement points into equation (70), (73), (75), (78), and apply the Kalman filter and extended Kalman filter respectively. The code of perspective transformation is shown in Appendix 5. The tracking results with perspective transformation are shown in Figure 5.20-5.22.

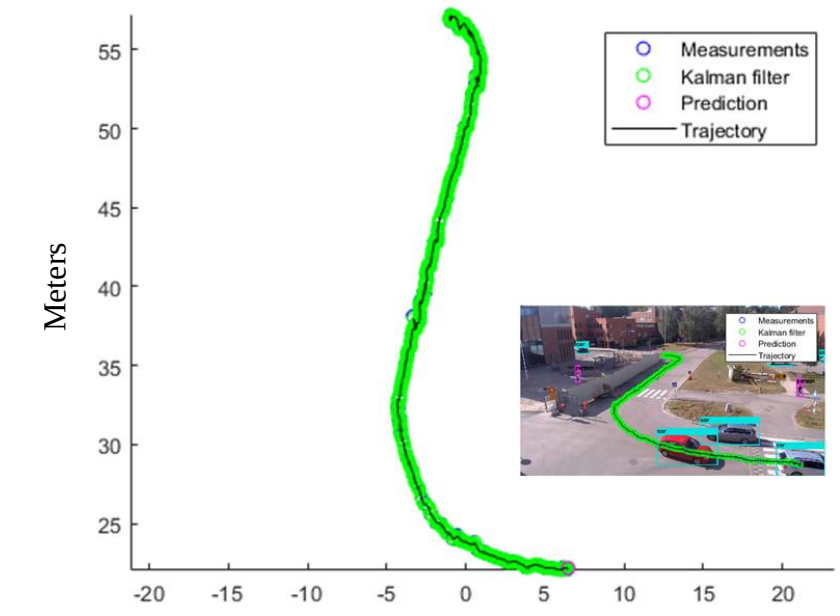


Kalman filter method



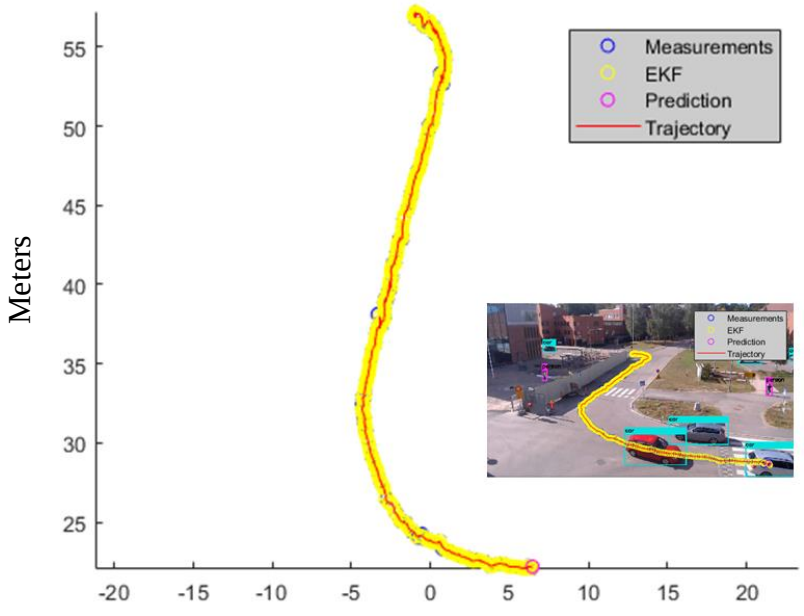
Extended Kalman filter method

Figure 5.20: Tracking Result of Car 3 with perspective transformation



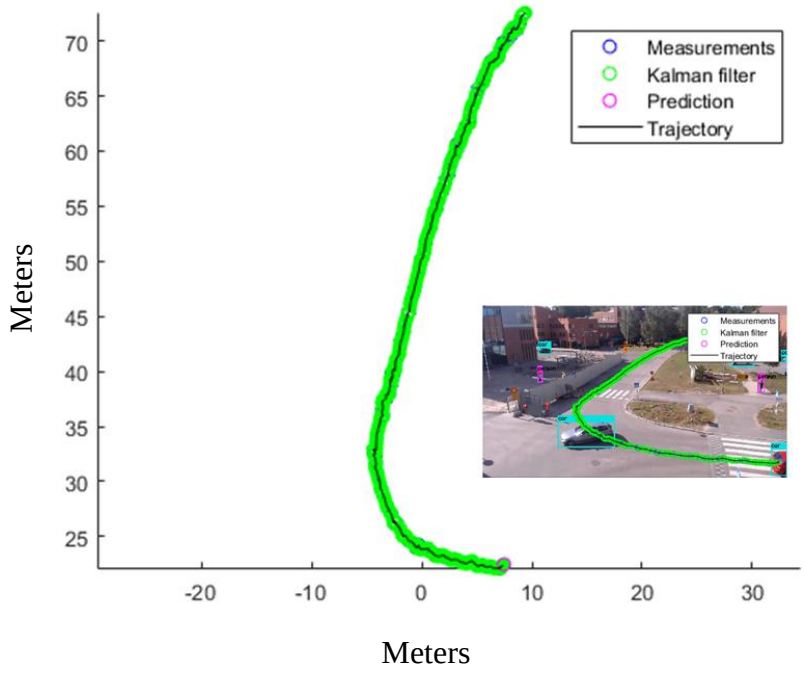
,

Kalman filter method

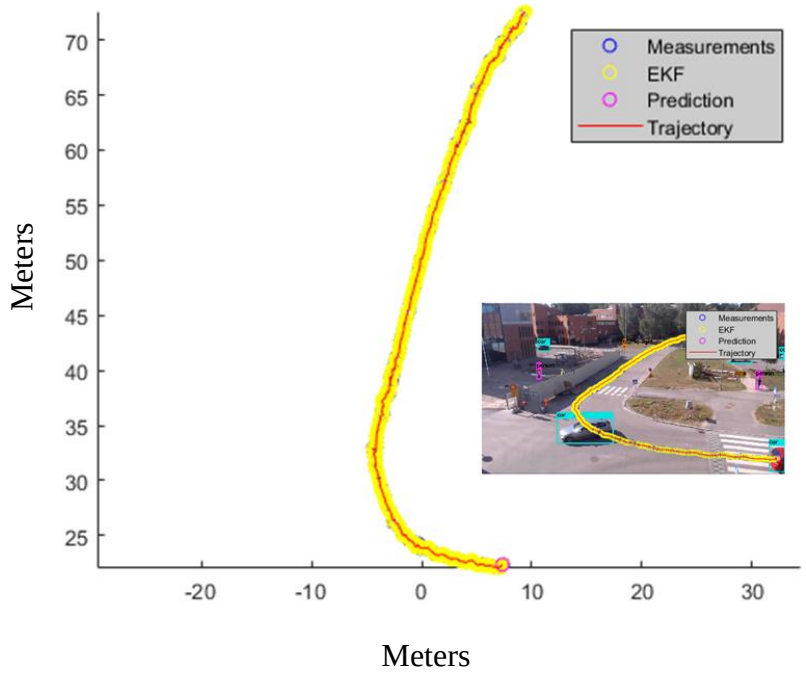


Extended Kalman filter method

Figure 5.21: Tracking Result of Car 4 with perspective transformation



Kalman filter method



Extended Kalman filter method

Figure 5.22: Tracking Result of Car 5 with perspective transformation

Before perspective transformation, it has been mentioned that polynomial fitting is not feasible for Car 4 and Car 5, so the trajectories for these vehicles are plotted by simply connecting the estimation points. After perspective transformation, the case is the same because perspective transformation would not change the basic shape of the trajectories. When calculating the fluctuation errors, we use the points collected from the fitting trajectories as standard. So, the fluctuation errors for Car 4 and Car 5 cannot be calculated, and only RMSE are compared in Figure 5.22.

The RMSE of the tracking result with perspective transformation are listed in Table 5.7.

Table 5.7: RMSE of the tracking result with perspective transformation ($|\boldsymbol{R_n}| = \boldsymbol{0.1}$, $\boldsymbol{\sigma_w^2 = 10^4}$)

Vehicle	KF Method		EKF Method	
	RMSE (m)	Fluct. Error (m)	RMSE (m)	Fluct. Error (m)
Car 3	0.0345	0.2856	0.0400	0.2929
Car 4	0.0327	/	0.0351	/
Car 5	0.0403	/	0.0417	/
Average	0.0358	0.2856	0.0389	0.2946

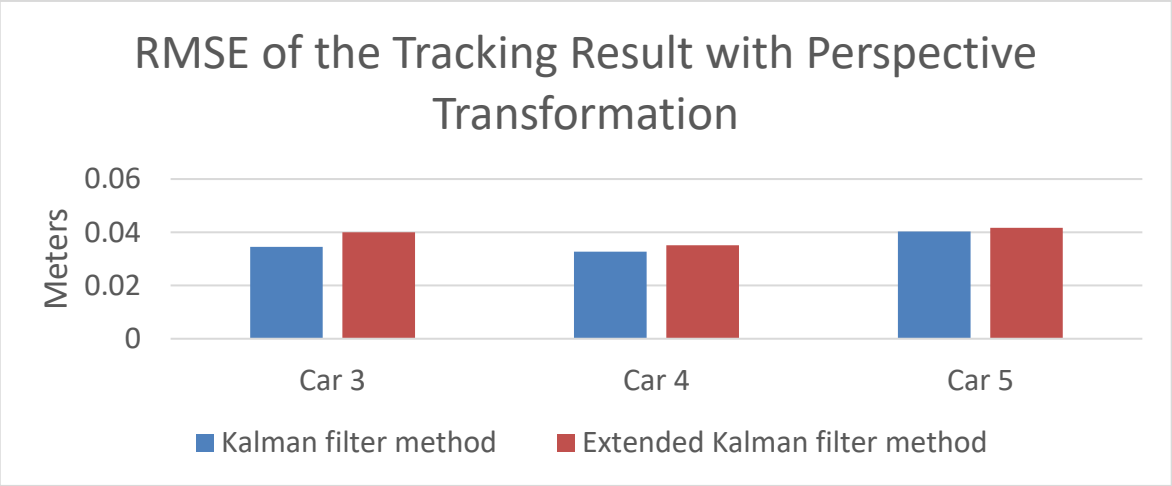


Figure 5.22: Chart of RMSE of the tracking result with perspective transformation (m) ($|R_n| = 0.1$, $\sigma_w^2 = 10^4$)

From the result, it can be seen that Kalam filter method and extended Kalman filter method are at an equivalent level in both precision and stableness after perspective transformation. Assuming the measurement points are precise, the RMSE of the two methods are within 0.05m. The fluctuation errors of the two methods are within 0.3 m. Considering the average driving range of the three vehicles are around 50m, the errors are within a reasonable range.

However, the validation of perspective transformation is under following conditions:
First, the measurement points collected by vehicle detection are reliable;
Second, there is no distortion in the camera image;
Third, the calibration of α and λ are precise;
Fourth, there is no elevation on the ground plane.

6. Discussion

In order to build the Intelligent Transportation System, computer vision is extensively utilized in life today. Compared to increasing mature on-board techniques such as pedestrian detection, lane departure warning, and traffic sign recognition, the techniques from the perspective of road and infrastructures are much less. The thesis has developed a vision-based vehicle detection and tracking system, where the vehicle images are collected by the video sensors on the road; traffic flow parameters are extracted by the detection and tracking module; then the traffic information is processed in the control center and shared inside the system.

In the simulation part, results show that YOLO algorithm is able to detect the vehicle continuously and precisely; all of Kalman filter method, extended Kalman filter method, and particle filter method can correctly reveal the vehicle trajectory and estimate the upcoming state. The Root Mean Square Errors, as well as the fluctuation errors of the three methods, are within 3.5 pixels. The practical experiment reflects that vehicle detection can be realized using YOLO algorithm in real-time; the trajectory, as well as the upcoming state, can be obtained individually for multi-vehicles. For both of the methods, The Root Mean Square Errors for all the vehicles are within 4.5 pixels, and the fluctuation errors are within 4 pixels. Therefore, the detection and tracking results of both the simulation and the practical experiment are of high accuracy and stableness. The effects of noise show that when the process noise covariance is larger, the estimation would be more precise but less stable; when the measurement noise covariance is larger, the estimation would be less precise but more stable.

It is noteworthy that besides the location of the vehicle, the state-space models also contain other states such as vehicle speed and heading direction. It can be inferred that the road sensors are not only able to get the pixel location the vehicles and estimate their next state, but also able to deduce their dynamic information such as vehicle speed, acceleration and heading direction. Using perspective transformation, the position as well as the dynamic information can be converted to the ground plane, which enables us to precisely locate and track the vehicle on road.

Admittedly, the vision-based vehicle detection and tracking system has its limitations. Firstly, YOLO algorithm is not effective for small objects within the image, due to the spatial constraints of the algorithm. Secondly, all of Kalman filter method, extended Kalman filter method and particle filter method rely on the measurement data of the sensor; if the measurements data are not precise, the results would be greatly influenced. Thirdly, the tracking algorithm requires great computational complexity, so vehicle tracking would take a relatively long time even when vehicle detection is a real-time process. Fourth, if the

interference factors such as shadows and objects blocking the view cover a big area or stay for a long time, its influences could not be neglected. Last but not least, a fatal flaw of the vision-based vehicle detection and tracking is that the video sensors only work during daytime. To construct an effective vision-based system for night vision, one way is to use an infrared camera, which is less demanding for lightening condition. Another way is combining YOLO algorithm with motion detection methods, since motion detection methods are less sensitive to the lighting conditions of the surrounding environment. The third way is to add the pre-processing procedures for night images such as removing highlights, enhancing contrast and increasing brightness.

Despite the shortcomings, vision-based vehicle detection and tracking are successfully accomplished in the thesis. Due to the limitation of the experimental samples, it could not deal with all kinds of complex circumstances in the traffic scene. Nevertheless, it does provide an initial instance for the systematic study of vision-based vehicle detection and tracking in the Intelligent Transportation System.

7. Conclusion

The thesis aims to realize vision-based vehicle detection and tracking in the Intelligent Transportation System. First, it introduced the methods for vehicle detection and tracking. Next, it established the mathematical framework of the system, including dynamic model and sensor model. Then, it simulated the traffic scene collected by road sensor where there is only one car, and the traffic scene is ideal. YOLO algorithm is applied to the image sequence for vehicle detection. Kalman filter method, extended Kalman filter method, and particle filter method are employed and compared for vehicle tracking. Following is the practical experiment where there are different vehicles at the same time, and the traffic scene is in real life with various interference factors. YOLO algorithm is utilized with OpenCV to realize real-time vehicle detection, and Kalman filter is applied to obtain the trajectories and upcoming positions of the vehicles. Finally, perspective transformation is utilized to transform the coordinates from the image plane to the ground plane, which makes it possible to track the vehicles on the ground plane. The results demonstrate that vision-based vehicle detection and tracking collected by road sensor are achieved.

Furthermore, if the amount of road sensors is big enough to cover a specific area, all the vehicles in the area can be unified and coordinated managed by the administrators of the system. Then the processed data can be shared inside of the whole system, either to vehicles, to infrastructures, or to users, and the interactions among them are built.

Back to the transportation system, the main challenges it needs to solve are traffic congestions, environmental impacts, energy consumption, safety hazards, and high maintenance costs. If vehicle detection and tracking can be realized and popularized in daily lives, the vehicles can be unified planned. For the drivers, congestion notification can be sent to prevent traffic rush; the optimal route can be suggested to save energy; risk warning can be sent to avoid accidents. Information about real-time public transportation can be shared with each individual to promote public transportation, so as to reduce environmental pollution. And infrastructures can be optimally distributed according to the feedbacks to save the construction expenses as well as the maintenance costs. Therefore, the system is sustainable and efficient, and transport problems are solved. And this will be a significant step towards the Intelligent Transportation System.

Reference List

Y. Lin, P. Wang and M. Ma (2017) *Intelligent Transportation System (ITS): Concept, Challenge and Opportunity*, 2017 IEEE 3rd international conference on big data security on cloud (big data security), IEEE international conference on high performance and smart computing (HPSC), and IEEE international conference on intelligent data and security (ids), Beijing, pp. 167-172.

Y. Liu, B. Tian, S. Chen, F. Zhu and K. Wang (2013) *A survey of vision-based vehicle detection and tracking techniques in ITS*, in Proceedings of 2013 IEEE International Conference on Vehicular Electronics and Safety, Dongguan, pp. 72-77.

S. Sivaraman and M. M. Trivedi (2013) *Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis*, in IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 4, pp. 1773-1795.

J. Zhang, F. Wang, K. Wang, W. Lin, X. Xu and C. Chen (2011) *Data-Driven Intelligent Transportation Systems: A Survey*, in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pp. 1624-1639.

L. Figueiredo, I. Jesus, J. A. T. Machado, J. R. Ferreira and J. L. Martins de Carvalho (2001) *Towards the development of intelligent transportation systems*, in Proceedings of 2001 IEEE Intelligent Transportation Systems, Oakland, CA, pp. 1206-1211.

T. Boonphoka, and P. Uthansakul (2014) *Remaining time improvement of V2V communication based GPS direction detection*, the 20th Asia-Pacific Conference on Communication (APCC2014), pp. 458-462.

R.Klette, (2014) *Concise Computer Vision*, Springer London,.

N. Buch, S. A. Velastin and J. Orwell (2011) *A review of computer vision techniques for the analysis of urban traffic*, IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 3, pp. 920-939.

J. Liebelt, C. Schmid and K. Schertler (2008) *Viewpoint-independent object class detection using 3d feature maps*, in Proceedings of 2008 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8.

S. Gupte, O. Masoud, R. F. K. Martin, and N. P. Papanikolopoulos (2002) *Detection and classification of vehicles*, IEEE Transactions on Intelligent Transportation Systems, vol. 3, no. 1, pp. 37-47.

S. S. Beauchemin and J. L. Barron (1995) *The computation of optical flow*, ACM Computing Surveys (CSUR), vol. 27, no. 3, pp. 433-466.

M. Seki, H. Fujiwara and K. Sumi (2000) *A robust background subtraction method for changing background*, in Proceedings of the 5th IEEE Workshop on Applications of Computer Vision, pp. 207-213.

R. Girshick, J.Donahue, T. Darrell, J. Malik (2014) *Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)*, in Proceedings of 2014 IEEE Conference on Computer Vision and Pattern Recognition.

R. Girshick (2015) *Fast R-CNN*, Proceedings of 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1440-1448.

S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis and Machine Intelligence. Jan. 2016.

J. Redmon, S. Divvala, R. Girshick and A. Farhadi (2016) *You Only Look Once: Unified, Real-Time Object Detection*, in Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779-788.

J. Redmon, A. Farhadi (2017) *YOLO9000: Better, Faster, Stronger*, in Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.6517-6525.

J. Redmon, A. Farhadi (2018) *YOLOv3: An Incremental Improvement*, arXiv preprint arXiv:1804.02767.

P. Fieguth, D. Terzopoulos (1997) *Color-based tracking of heads and other mobile objects at video frame rates*, in Proceedings of 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 21-27.

D. Comaniciu, V. Ramesh, and P. Meer (2003) *Kernel-based object tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, pp. 564-577.

W. Zhao and Y. Dong (2015) *Application research of Kalman filter in optical atomic magnetometer*, 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, pp. 1888-1892.

A. Corovic, V. Ilic, S. Duric, M. Marijan, and B. Pavkovic (2018) *The Real-Time Detection of Traffic Participants Using YOLO Algorithm*, the 26th Telecommunications Forum (TELFOR), pp.1-4.

J. Tao, H. Wang, X. Zhang, X. Li, H. Yang (2017) *An object detection system based on YOLO in traffic scene*, the 6th International Conference on Computer Science and Network Technology (ICCSNT), pp. 315-319.

J. Lin and M. Sun, (2018) "A YOLO-based Traffic Counting System", 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI).

Z. Xu, H. Shi, N. Li, C. Xiang, (2018) *Vehicle Detection Under UAV Based on Optimal Dense YOLO Method*, The 2018 5th International Conference on Systems and Informatics (ICSAI 2018).

A. Salarpour, A. Salarpour, M. Fathi, M.H. Dezfoulian, (2011) *Vehicle Tracking Using Kalman Filter and Features*, Signal & Image Processing: An International Journal (SIPIJ) Vol.2, No.2.

D. Ponsa, A. Lopez, J. Serrat, F. Lumbreras and T. Graf (2005) *Multiple vehicle 3D tracking using an unscented Kalman*, in Proceedings of 2005 IEEE Intelligent Transportation Systems, Vienna, pp. 1108-1113.

J. Hui (2018) *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3*, accessed 10 March 2019, <https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>.

F. Gustafsson (2018) *Statistical Sensor Fusion*, Studentlitteratur AB, ISBN: 978-9144127248.

R. Ojala, J. Vepsäläinen, J. Hanhirova, V. Hirvisalo, and K. Tammi (2018) *Novel Convolutional Neural Network-based Roadside Unit for Accurate Pedestrian Localisation*, Trans. Intell. Transp. Syst. [Submitted].

Appendix 1: Identification Algorithm

```
clear;

clc;
```

Input data

```
cd 'D:\Wencan';
num = xlsread('Book2.xlsx','Sheet2');
idx = all(isnan(num),2);
idy = 1+cumsum(idx);
idz = 1:size(num,1);
C = accumarray(idy(~idx),idz(~idx),[],@(r){num(r,:)});
temp = initialize_temp(C);
result = {};
```

Search for the matching string

```
for N = 2:size(C)
    Ct = C{N};                                          %Ct is frame matrix
    move = [];
    for i = 1:size(temp,2)                        %Loop through lines in temp
        [close_match, match_idx] = find_match(temp{i}, Ct);
                     %Find a match between one line in temp and frame matrix
        if close_match == 0
            result{end+1} = temp{i};
            move = [move, i];
        elseif close_match == 1
            temp{i} = [temp{i};Ct(match_idx,1:2)];
            Ct(match_idx,:) = [];                                %Delete row
        end
    end

    temp(move) = [];

    for j = 1:size(Ct)
```

```
        temp{end+1} = Ct(j,1:2);        %Put remaining rows in Ct into temp
    end
end

result = finalize_result(result,temp);
```

Delete short strings

```
result_new = {};

for p = 1:size(result,2)
    if size(result{p},1)>=100                                   %Minimum length
        result_new{end+1} = result{p};
    end
end
```

Plot the result

```
for p = 1:size(result_new,2)
    figure
    plot(result_new{p}(:,1),result_new{p}(:,2),'bo')
    set(gca,'ydir','reverse')
    xlim ([0 1])
    ylim ([0 1])
    axis tight;
    hold on
    I = imread('11088.jpg');
    h = image([0 1],[0 1],I);
    uistack(h,'bottom')
end
```

Define functions

```
function t = initialize_temp(M)
    t = {};
    for r = 1:size(M{1},1)
        t{end+1} = M{1}(r,1:2);
    end
end
```

```
function [a, b] = find_match(A,B)
    old_point = A(end,:);
    a = 0;
    b = 0;
    for k = 1:size(B,1)
        new_point = B(k,1:2);
        d = norm(new_point - old_point);
        if d<0.02                                   %Define threshold here
            a = 1;
            b = k;
            break
        end
    end
end

function A = finalize_result(A,B)
    for i = 1:size(B,2)
        A{end+1} = B{i};
    end
end
```

Appendix 2: Use Kalman Filter for Tracking Experiment

```
clear variables;
clc;
rng(48);
cd 'D:\Wencan';
num = xlsread('iden_result','sheet1');   %Choose vehicle
```

Parameters

```
T = size(num,1)/42.5;              % Simulation time
dt = 1/42.5;              % Sampling time (for measurements)
R = 0.1*eye(2);      % Measurement noise covariance
sigma2w = 10000;        % Measurement process covariance
m0 = [num(1,1); num(1,2);0;0];   % Initial guess of the state
P0 = 2*eye(4);       % Covariance of the initial guess
t = dt:dt:T;
N = length(t);
y(1,:) = num(:,1);
y(2,:) = num(:,2);
```

Model

```
% Dynamic model: Wiener velocity model in two dimensions
F = [1 dt;
     0  1;];

F = kron(F, eye(2));

Q = sigma2w*[
    dt^3/3, dt^2/2;
    dt^2/2, dt;
];

Q = kron(Q, eye(2));

% Measurement model
G = [eye(2), zeros(2)];
```

Kalman filter

```
% Preallocate
xps_kf = [num(1,1); num(1,2);0;0];
Pps_kf = zeros(4, 4, N);
xs_kf = [num(1,1); num(1,2);0;0];
Ps_kf = zeros(4, 4, N);

% Initialize the filter
x = m0;
P = P0;

% Process measurements
for n = 1:N
    % Prediction
    xp = F*x;
    Pp = F*P*F' + Q;

    % Measurement update
    K = Pp*G'/(G*Pp*G' + R);
    x = xp + K*(y(:, n) - G*xp);
    P = Pp - K*(G*Pp*G' + R)*K';

    % Store
    xps_kf(:, n) = xp;
    Pps_kf(:, :, n) = Pp;
    xs_kf(:, n) = x;
    Ps_kf(:, :, n) = P;

% prediction and trajectory
    xp = F*x;
    p = polyfit(xs_kf(1, :), xs_kf(2, :),10);
    x1 = xs_kf(1, :);
    y1 = polyval(p,x1);
    z1=cat(1,x1,y1);
end

% convert to pixels
y(1,:) = y(1,:)*1920;
y(2,:) = y(2,:)*1080;
xs_kf(1,:) = xs_kf(1,:)*1920;
xs_kf(2,:) = xs_kf(2,:)*1080;
x1 = x1*1920;
```

```
y1 = y1*1080;
z1(1,:) = z1(1,:)*1920;
z1(2,:) = z1(2,:)*1080;
xp(1,:) = xp(1,:)*1920;
xp(2,:) = xp(2,:)*1080;
```

RMSE and fluctuation error

```
fprintf('KF position RMSE: %.4f\n', sqrt(mean(sum((y(1:2, :)-
xs_kf(1:2, :)).^2))));
fprintf('KF fluctuation error: %.4f\n', sqrt(mean(sum((z1(1:2, :)-
xs_kf(1:2, :)).^2))));
```

Plot the result

```
figure(1); clf();
hold on;
plot(y(1, :), y(2, :), 'bo');
plot(xs_kf(1, :), xs_kf(2, :), 'go');
plot(xp(1),xp(2),'mo');
plot(x1,y1,'g-')
legend('Measurements', 'Kalman filter','Prediction','Trajectory');
axis equal;
set(gca,'ydir','reverse')
xlim ([0 1])
ylim ([0 1])
axis tight;
hold on
I = imread('10926.jpg');
h = image([0 1920],[0 1080],I);
uistack(h,'bottom')
```

Revised from Roland Hostettler, *GPS navigation example using a Kalman filter*, 2018.

Appendix 3: Use Extended Kalman Filter for Tracking Experiment

```
clear variables;
clc;
rng(48);
cd 'D:\Wencan';
num = xlsread('iden_result','sheet1');   %Choose vehicle
```

Parameters

```
T = size(num,1)/42.5;              % Simulation time
dt = 1/42.5;              % Sampling time (for measurements)
R = 0.1*eye(2);      % Measurement noise covariance
Sigma_w = 10000;        % Measurement process covariance
m0 = [num(1,1); num(1,2);0;0];   % Initial guess of the state
P0 = 2*eye(4);       % Covariance of the initial guess
t = dt:dt:T;
N = length(t);
y(1,:) = num(:,1);
y(2,:) = num(:,2);
```

Model

```
% Dynamic model: Quasi-constant turn model in two dimensions
f = @(x) [
    x(3)*cos(x(4));
    x(3)*sin(x(4));
    0;
    0;
];
B = [
    zeros(2);
    eye(2);
];

% Measurement model
G = [eye(2), zeros(2)];

% Euler-Maruyama discretization (+ Jacobian)
```

```
fd = @(x, ~) x + dt*f(x);
Q = dt*B*Sigma_w*B';
Fx = @(x) [
    1, 0, dt*cos(x(4)), -dt*x(3)*sin(x(4));
    0, 1, dt*sin(x(4)),  dt*x(3)*cos(x(4));
    0, 0,            1,                 0;
    0, 0,            0,                 1;
];
```

Extended Kalman filter

```
% Preallocate
xhats_ekf = zeros(4, N);
Ps_ekf = zeros(4, 4, N);

% Initialize the filter
x = m0;
P = P0;

% Process measurements
for n = 1:N
    % Prediction
    xp = fd(x);
    Pp = Fx(x)*P*Fx(x)' + Q;

    % Measurement update
    K = Pp*G'/(G*Pp*G' + R);
    x = xp + K*(y(:, n) - G*xp);
    P = Pp - K*(G*Pp*G' + R)*K';

    % Store
    xhats_ekf(:, n) = x;
    Ps_ekf(:, :, n) = P;

    % prediction and trajectory
    xp = fd(x);
    p = polyfit(xhats_ekf(1, :), xhats_ekf(2, :),10);
    x1 = xhats_ekf(1, :);
    y1 = polyval(p,x1);
    z1=cat(1,x1,y1);
end
```

```
% convert to pixels
y(1,:) = y(1,:)*1920;
y(2,:) = y(2,:)*1080;
xhats_ekf(1,:) = xhats_ekf(1,:)*1920;
xhats_ekf(2,:) = xhats_ekf(2,:)*1080;
x1 = x1*1920;
y1 = y1*1080;
z1(1,:) = z1(1,:)*1920;
z1(2,:) = z1(2,:)*1080;
xp(1,:) = xp(1,:)*1920;
xp(2,:) = xp(2,:)*1080;
```

RMSE and fluctuation error

```
fprintf('EKF position RMSE: %.4f\n', sqrt(mean(sum((y(1:2, :)-
xhats_ekf(1:2, :)).^2))));
fprintf('EKF fluctuation error: %.4f\n', sqrt(mean(sum((z1(1:2, :)-
xhats_ekf(1:2, :)).^2))));
```

Plot the result

```
figure(1); clf();
hold on;
plot(y(1, :), y(2, :), 'bo');
plot(xhats_ekf(1, :), xhats_ekf(2, :), 'yo');
plot(xp(1),xp(2),'mo');
plot(x1,y1,'y-')
legend('Measurements', 'Kalman filter','Prediction','Trajectory');
axis equal;
set(gca,'ydir','reverse')
xlim ([0 1])
ylim ([0 1])
axis tight;
hold on
I = imread('10926.jpg');
h = image([0 1920],[0 1080],I);
uistack(h,'bottom')
```

Revised from Roland Hostettler, *EKF/UKF/PF Example: Tracking a Manoeuvring Target*, 2018.

Appendix 4: Use Particle Filter for Tracking Simulation

```
clear variables;
rng(48);
```

Parameters

```
T = 5.1;              % Simulation time
dt = 0.1;              % Sampling time (for measurements)
R = 1*eye(2);      % Measurement noise covariance
sigma2w = 500;
m0 = [1450;277;0;0];    % Initial guess of the state
P0 = 2*eye(4);       % Covariance of the initial guess
t = dt:dt:T;
N = length(t);
J = 1000;        % J sets of particles

y(1,:) = [1450 1428 1408 1385 1361 1339 1317 1293 1268 1247 1223 1198 1175 ...
          1157 1143 1112 1092 1047 1029 1008 994 974 951 934 918 900 ...
          881 867 854 843 827 813 797 784 771 758 748 735 723 ...
          715 703 692 685 673 671 664 653 648 639 636 628];
y(2,:) = [277 277 277 278 279 278 277 278 277 277 277 276 276 276 ...
           274 276 276 275 278 279 279 285 286 288 291 294 298 304 ...
           306 312 317 325 331 337 345 352 359 372 381 392 408 420 ...
           433 451 462 487 507 532 561 595 629];
```

Model

```
% Dynamic model: Wiener velocity model in two dimensions
F = [1 dt;
     0  1;];
F = kron(F, eye(2));
Q = sigma2w*[
    dt^3/3, dt^2/2;
    dt^2/2, dt;
];
Q = kron(Q, eye(2));

% Measurement model
G = [eye(2), zeros(2)];
```

Particle filter

```
xs_bpf = zeros(4, J, N);     % Particles
ws_bpf = zeros(1, J, N);     % Weights
xhats_bpf = [1450;277;0;0];
Ps_bpf = zeros(4, 4, N);
wtilde = zeros(1, J);

% Initialize
x = m0*ones(1, J) + chol(P0).'*randn(4, J);
for n = 1:N
    for j = 1:J
        % Sample
        q = Q*randn(4, 1);
        x(:, j) = F*x(:, j) + q;

        % Calculate weights
        wtilde(:, j) = mvnpdf(y(:, n).', (G*x(:, j)).', R);
    end

    % Normalize the weights
    w = wtilde/sum(wtilde);

    % Calculate the mean and covariance
    xhats_bpf(:, n) = x*w.';
    P = zeros(4, 4);
    for j = 1:J
        P = P + w(j)*((x(:, j) - xhats_bpf(:, n))*(x(:, j) - xhats_bpf(:, n))');
    end
    Ps_bpf(:, :, n) = P;

    % Resample
    x = resample(x, w);

    % prediction and trajectory
    xp = F*x;
    p = polyfit(xhats_bpf(1, :), xhats_bpf(2, :),10);
    x1 = xhats_bpf(1, :);
    y1 = polyval(p,x1);
    z1=cat(1,x1,y1);
end
```

RMSE and fluctuation error

```
fprintf('PF position RMSE: %.2f\n', sqrt(mean(sum((y(1:2, :)-
xhats_bpf(1:2, :)).^2))));
fprintf('PF fluctuation error: %.2f\n', sqrt(mean(sum((z1(1:2, :)-
xhats_bpf(1:2, :)).^2))));
```

Plot the result

```
figure(1); clf();
hold on;
plot(y(1, :), y(2, :), 'bo');
plot(xhats_bpf(1, :), xhats_bpf(2, :), 'co');
plot(xp(1),xp(2),'mo');
plot(x1,y1,'c-')
legend('Measurements','Particle filter','Prediction','Trajectory');
axis equal;
set(gca,'ydir','reverse')
xlim ([0 1680])
ylim ([0 1050])
axis tight;
hold on
I = imread('52.jpg');
h = image([0 1680],[0 1050],I);
uistack(h,'bottom')
```

Define resample function

```
function x = resample(x, w)
    J = size(w, 2);
    u = ((0:J-1) + rand(1, J))/J;
    wc = cumsum(w);
    wc = wc/wc(J);
    [~, i1] = sort([u, wc]);
    i2 = find(i1 <= J);
    i = i2-(0:J-1);
    x = x(:, i);
end
```

Revised from Roland Hostettler, *EKF/UKF/PF Example: Tracking a Manoeuvring Target*, 2018.

Appendix 5: Perspective Transformation

```
clc;

clear;

load('match_string')
a = result_new{2}
lamda = 0.39;
h = 10.8;
cx = a(:,1); %horizontal
cy = a(:,2); %vertical(-)
wp = a(:,3); %h
hp = a(:,4); %v
px = cy + 0.5*hp - 0.5;
py = cx - 0.5;
for i = 1:size(a,1)
    xt(i) = h *(1 - lamda * px(i) * tand(15))/(tand(15) + lamda *px(i));
    yt(i) = 16/9 * py(i) * lamda *xt(i);
end
 figure;
 plot (yt,xt, 'o');
 axis equal;
 save('D:\Wencan\x_axis','xt');
 save('D:\Wencan\y_axis','yt');
```