Rose-Hulman Institute of Technology
Rose-Hulman Scholar

Graduate Theses - Electrical and Computer Engineering

Graduate Theses

8-2018

Accuracy Improvement of Pedestrian Trajectory Prediction by an Extended Kalman Filter and Pedestrian Behavior Classification

Jiayu Guo
Rose-Hulman Institute of Technology

Follow this and additional works at: https://scholar.rose-hulman.edu/electrical_grad_theses

Part of the Electrical and Electronics Commons

Recommended Citation
Guo, Jiayu, "Accuracy Improvement of Pedestrian Trajectory Prediction by an Extended Kalman Filter and Pedestrian Behavior Classification" (2018). *Graduate Theses - Electrical and Computer Engineering*. 13.
https://scholar.rose-hulman.edu/electrical_grad_theses/13

This Thesis is brought to you for free and open access by the Graduate Theses at Rose-Hulman Scholar. It has been accepted for inclusion in Graduate Theses - Electrical and Computer Engineering by an authorized administrator of Rose-Hulman Scholar. For more information, please contact weir1@rose-hulman.edu.

ACCURACY IMPROVEMENT OF PEDESTRIAN TRAJECTORY PREDICTION BY AN EXTENDED KALMAN FILTER AND PEDESTRIAN BEHAVIOR CLASSIFICATION

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Jiayu Guo

In Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Electrical Engineering

August 2018

©2018 Jiayu Guo

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Final Examination Report

Jiayu Guo
Name

Electrical Engineering
Graduate Major

Thesis Title Accuracy Improvement of Pedestrian Trajectory Prediction by Extended Kalman Filter and Pedestrian Behavior Classification

DATE OF EXAM: May 23, 2018

EXAMINATION COMMITTEE:

	Thesis Advisory Committee	Department
Thesis Advisor:	JianJain Song	ECE
	Mark Yoder	ECE
	Michael McInerney	PHOE

PASSED X **FAILED** ____

ABSTRACT

Guo, Jiayu

M.S.E.E.

Rose-Hulman Institute of Technology

August 2018

Accuracy Improvement of Pedestrian Trajectory Prediction by an Extended Kalman Filter and Pedestrian Behavior Classification

Thesis Advisor: Dr. Jianjian Song

The objective of this thesis is to improve the accuracy of predicting motion trajectory, i.e., speed and direction, of a pedestrian in front of an Ego Vehicle which has a Mobileye camera with an advanced driver assistance system (ADAS). The Ego Vehicle captures and records videos of pedestrians in front of it, and these videos are analyzed to predict a pedestrian trajectory from instantaneous, random actions of a pedestrian. Instant actions include, but are not limited to, walking at a constant speed, sudden accelerations/decelerations, sudden dodging from the Ego Vehicle, sudden advancements to the Ego Vehicle, sudden withdrawals or sudden stops at the road edge, etc. Pedestrian positions and motion data from the videos can be used to estimate pedestrian state parameters and predict pedestrian movement.

The pedestrian videos contain noises due to the nonlinear trajectory of a pedestrian and the Ego Vehicle. An extended Kalman filter (EKF) and pedestrian behavior classification are applied to these pedestrian videos to obtain a more accurate pedestrian trajectory. The EKF is used to suppress noises from the videos and aids in predicting the next state of pedestrian movement. The EKF can reduce noises in a nonlinear system. The EKF is an efficient and effective tool in creating more stable and smoother pedestrian positions from the Ego Vehicle videos, as we have demonstrated from analyzing pedestrian trajectories from real-world videos. These new position data inputs are used to calculate the new velocity of a pedestrian. This new velocity is averaged over 30 consecutive video frames to obtain a more accurate and stable velocity. After the new position and velocity are calculated, pedestrian behavior classification is applied to the data to calculate and group pedestrian behaviors into instant actions. The behavior classification is based on the estimation of the heading angle and acceleration of a pedestrian.

The combination of the extended Kalman filter and behavior classification forms a more accurate pedestrian trajectory prediction system. This approach is verified with 12 hours of ADAS camera Mobileye videos from an experimental car test site within a simulated urban area. Ten cases of pedestrian motion behaviors are analyzed. By calculating the Time to Collision (TTC) and comparing this result with the TTC directly from the ADAS camera, we have shown that our new TTC prediction is more stable and less noisy when contrasted with the older TTC predictions from an ADAS camera system.

ACKNOWLEDGMENTS

I would like to express my most humble and sincere gratitude to my advisor, Dr. Song, for all his support and guidance throughout my thesis study. He greatly contributed to aiding me in selecting my thesis, and it was under his guidance that I decided to carry out my study on ADAS camera research and development. This work would not be successful or possible even without Dr. Jianjian Song's amazing efforts. I would also like to thank my other fellow committee members: Dr. Mark A. Yoder and Dr. Michael F. McInerney. I learned the real-time OS knowledge from Dr. Yoder and image processing knowledge from Dr. McInerney. This knowledge shined much light upon this thesis study.

I am also grateful to all of my colleagues at APTIV; they helped me a lot with data collection in the cold winter of 2018. Finally, I would like to thank my family for supporting and loving me always.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ADAS	Advanced Driver Assistance System
AEB	Automatic Emergency Brake
AEB Pedestrian	Automatic Emergency Brake for Pedestrian
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
TTC	Time to Collision
EKF	Extended Kalman Filter

1 INTRODUCTION

An Automatic Emergency Brake (AEB) system is one of the most important and integral parts of an Advanced Driver Assistance System (ADAS). The AEB identifies an imminent collision and brakes without any driver intervention [1]. The European New Car Assessment Programme (EUNCAP) has included the AEB system in its test requirements since 2014[1]. Volvo, Audi, Mercedes-Benz and some OEMs already have AEB systems standardized in their production cars. Most ADAS suppliers and OEMs who have developed ADAS products by themselves have already developed complex AEB systems. Those AEB systems can detect front vehicles from cameras on the Ego Vehicle and calculate Vehicle-to-Vehicle (V2V) Time-to-Collision (TTC) quickly and accurately.

The Automatic Emergency Brake for Pedestrian (AEB Pedestrian) is a critical component of an AEB system. Recently, Vehicle-to-Pedestrian (V2P) accidents have become a major concern for road safety. The United States recorded 5,376 pedestrian deaths in 2015. On average, one pedestrian was killed every two hours and injured every seven minutes in traffic accidents [3]. EUNCAP has already included AEB Pedestrian into its test requirements in 2016 [2], and the US government's National Highway Traffic Safety Administration (NHTSA) has also required 99% of OEMs to include AEB Pedestrian by 2022 [4].

Initially, the thesis study was based on a paper entitled "Pedestrian-Vehicular Collision Avoidance Based on Vision System", which introduced a method that uses the C^4 algorithm to detect the contour of pedestrian movement [5]. The Kalman filter was implemented in that paper to track and record pedestrian movements, then the TTC and Time-to-Collision Range (TTCR) would be calculated to establish three levels of hazard. On the hardware side, a single-optical camera was installed on the experimental vehicle to detect pedestrians. A GPS was used to estimate the Ego Vehicle's motion. The motion of a pedestrian was considered as a linear motion or could be clustered as several predictable trajectories with a Kalman filter.

This thesis will show with experimental data that it is more difficult for an AEB function to estimate and track a pedestrian in front of the Ego Vehicle than to estimate and track a vehicle in front of the Ego Vehicle. An extended Kalman filter and a pedestrian behavior classification method are shown to predict pedestrian trajectories more accurately.

1.1 Overview of Mobileye's Automatic Emergency Brake for Pedestrians

Mobileye, an Intel company, is one of the largest computer vision and ADAS companies, which provides its products to global automakers including BMW, Ford, General Motors, Nissan, Volvo, Audi and Hyundai, etc. [6] In 2017, Intel acquired Mobileye [7] and the electric vehicle company Tesla changed its core ADAS technology acquisition from Nvidia to Mobileye [8]. The main product of Mobileye, the EyeQ series, is a system-on-chip (SoC) device with a monocular camera. ADAS on an EyeQ device implements both active and passive functions. The active functions will take real-time measurement, such as AEB, Adaptive Cruise Control, and Lane Keeping Assist. The passive functions such as Lane Departure Warning and Forward Collision Warning will alert drivers of potentially dangerous scenarios. There are multiple generations of the EyeQ series products ranged from the EyeQ1 to the EyeQ4. The EyeQ3 device used on a test car in this study can detect 14 different kinds of targets (e.g. a car, truck, bicycle, pedestrian, or traffic signs) and apply the active and passive functions on these targets.

One of the most important functions of the Mobileye's ADAS system, which will simply be called "Mobileye" through the rest of this thesis is the AEB function used for detecting and avoiding obstacles. A Mobileye device will automatically calculate the TTC and set brakes for all detected obstacles, including vehicles, motorcycles, bicycles, and pedestrians. A Mobileye device also has AEB Pedestrian functions that will detect pedestrians and set the TTC based on current pedestrian data to predict the pedestrian trajectory.

The pedestrian trajectory prediction of the AEB Pedestrian in the current Mobileye devices does not consider random pedestrian behaviors. For example, when a pedestrian is walking on a street, he/she could cross the street suddenly, or when a pedestrian crosses a street, he/she could suddenly change direction. Without prediction of these instant actions of pedestrians, the Mobileye devices could only predict the pedestrian trajectory based on the current constant speed. However, a pedestrian has different movement behaviors compared to vehicles. Therefore, the prediction of pedestrian trajectory will be inaccurate, and the AEB Pedestrian implementation in current Mobileye devices is not yet suitable for a real-world pedestrian movement situation.

1.2 Overview of Pedestrian Trajectory Prediction

The study of pedestrian trajectory prediction can be divided into two parts, pedestrian detection and pedestrian tracking. Pedestrian detection is the process of finding pedestrian information in videos or a series of images of pedestrians by analyzing pedestrian contours as well as histograms of oriented gradients (HOG). Pedestrian tracking is the process of finding and determining a pedestrian trajectory based on the information from pedestrian detection. Usually, pedestrian tracking can be done in two ways. One is to cluster a pedestrian trajectory by a number of training pedestrian trajectories, the other is to predict the next step of pedestrian movement by using a pedestrian history trajectory.

Pedestrian detection is based on 2D images or videos of the Front cameras of the Ego Vehicle to find the position of an obstacle and to determine whether the detected obstacle is a pedestrian. The Support Vector Machine (SVM) classifier is a popular trainable method for object detection [9], which takes a series of pedestrian data to calculate the histogram of gradient and develop a trained model. The trained model is then used to find the pedestrian in the test images or videos. Another traditional method is the Haar Wavelet [10], which can describe details of pedestrian patterns. Haar Wavelet image analysis consists of two steps. The first step is the determination of the Haar function, which defines the pattern features of the Haar Wavelet. The second step is the calculation of a scaling function which defines the scale filter. The higher the scale of the scaling function of the Haar wavelet, the higher the resolution of the pattern.

The first method for pedestrian tracking is to estimate the next step of the pedestrian trajectory based on the pedestrian's trajectory history. The Kalman filter is the most popular method for object tracking and predicting the next state of a linear system [11]. Since the pedestrian movement is non-linear, the Kalman filter is not good for pedestrian trajectory prediction. Therefore, an extended Kalman filter (EKF) was implemented for non-linear systems by Y.W. Xu, X.B. Cao and T. Li. The authors utilized the EKF to track a pedestrian trajectory by predicting each step of the pedestrian movement from a monocular camera system [12]. Three-dimensional coordinates and the velocity of the pedestrian were provided. The entire pedestrian trajectory was determined by applying the EKF regressively.

The second method of pedestrian tracking is to determine the pedestrian trajectory by a model trained through a number of pedestrian trajectories. Pedestrian trajectories are grouped into a number of clusters. Gaussian Process Regression (GPR) can be used to cluster the pedestrian trajectories [13] [14]. Y. Chen, M. Liu, S. Y. Liu, J. Miller, and J. P. How introduced a method that clusters the pedestrian trajectory patterns by applying GPR [15]. A number of pedestrian trajectories on a crossroad can be divided into several clusters. When a new pedestrian appears on this crossroad, the pedestrian will be predicted to belong to one of the clusters. This method requires the pedestrian to start from a fixed position and end at a fixed position; for example, the situation can be an airport shuttle station or a crossroad between buildings in a city.

2 COLLECTION AND SELECTION OF CAMERA DATA

In this thesis study, the Ego Vehicle was a production car equipped with an Intel Mobileye ADAS camera to take pedestrian videos. First, the Ego Vehicle performed a road test to collect testing data for three days with 600 video clips to simulate the real motion behaviors of a pedestrian. Each video had at least one pedestrian motion event. From these pedestrian movement videos, special cases of pedestrian behaviors were defined when a pedestrian walked on the street, such as crossing, walking along the street, or sudden, randomized movements.

Based on the pedestrian behavior cases observed from the road test data, six sets of training data for the behavior classification method were recorded from the experiments of pedestrian actions in front of the Ego Vehicle. Besides the vision information from the ADAS camera, the Ego Vehicle's velocity and yaw rate were available from the CAN bus of the Ego Vehicle for classification training.

Once the raw training data was collected, all instant actions of pedestrians had to be selected from the raw data. The selected data was divided into different cases, and the characteristics such as the position and the velocity of a pedestrian were observed for classification training.

2.1 Pedestrian Behavior Classification

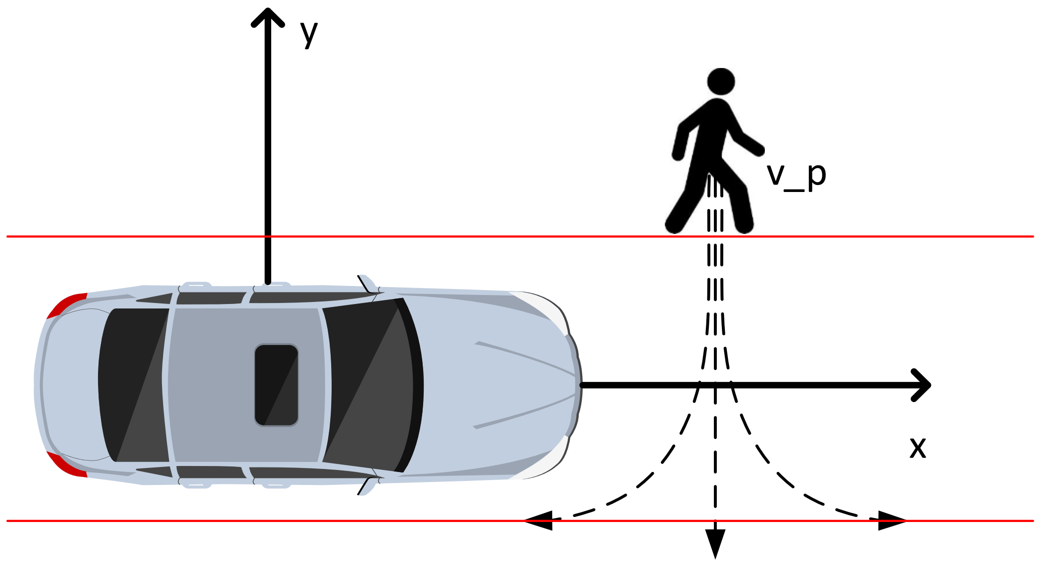
Pedestrian behaviors were classified into two case groups and ten subcases: Case A: Crossing the Street in Front of the Ego Vehicle and Case B: Walking on the Road alongside the Ego Vehicle.

There could be seven subcases in Case A: (A1) Constant Speed, (A2) Sudden Acceleration, (A3) Sudden Deceleration, (A4) Sudden Dodge from the Ego Vehicle, (A5) Sudden Advance to the Ego Vehicle, (A6) Sudden Withdrawal, and (A7) Sudden Stop at the Edge of the Road.

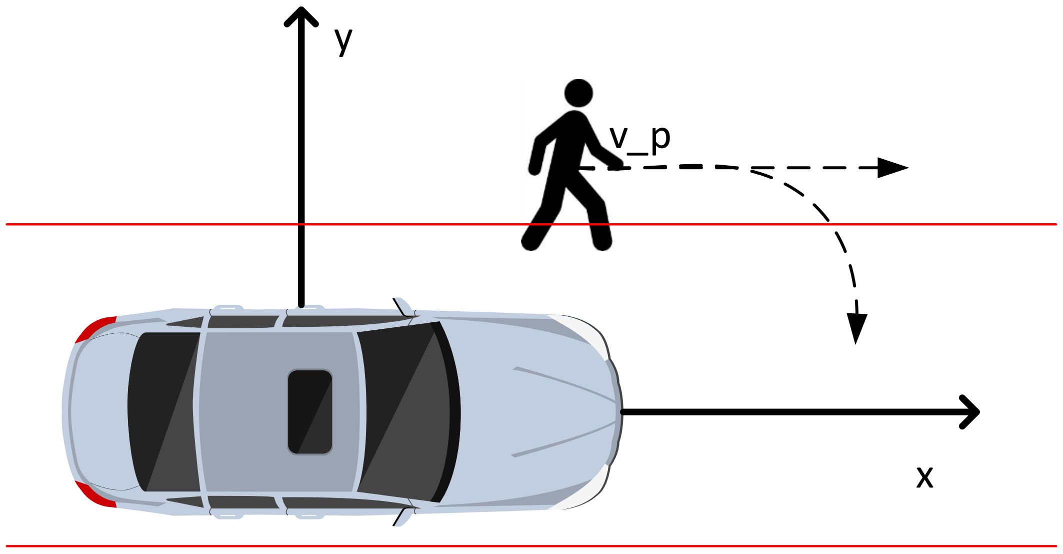
Case A4 and Case A5 could be further divided based on which side of the Ego Vehicle the pedestrian is passing, because it matters that the heading angles for both left and right actions have different features. Case A4 Left is defined as the pedestrian passing from the left side of the Ego Vehicle, while Case A4 Right is defined as the pedestrian passes from the right side of the Ego Vehicle. Case A5 Left and Case A5 Right are both defined accordingly.

There could be three subcases in Case B: (B1) Constant Speed, (B2) A Sudden Turn towards the Ego Vehicle and (B3) A Sudden Stop at the Road Edge.

In Case A1, the Ego Vehicle will go straight as in Figure 1, and the pedestrian will cross the street at the various speed states such as in Cases A1, A2, A3, or in different directions such as in Cases A4, A5, A6, or a stop by the side of the road such as in Case A7. For Case B, the Ego Vehicle will go straight, while the pedestrian will concurrently walk in the same direction as the Ego Vehicle. This situation normally takes place at a non-intersection street with no traffic signals. The pedestrian can cross the street from either side of the road with the above actions.



Case A: A Pedestrian Is Crossing the Street.



Case B: A Pedestrian Is Walking on the Road alongside the Ego Vehicle.

Figure 1. Case A and Case B of Pedestrian Behaviors.

2.2 ADAS Camera Data Collection

The data collection was performed in Shanghai, China. Some of the data collection was obtained at an experimental car test site in the Disneyland area while the rest was at Shanghai's

Waigaoqiao Free Trade Zone. It took five days to collect the data since the first day was spent setting up the ADAS camera, sensor devices, equipment, and vehicles, etc. The estimated time for each test was about 5 minutes for an estimated total of 12 hours. As a result, 12 different actions of data were recorded. All the data collection was held in good weather conditions to make sure that no external environmental effects such as rain and snow would influence the results. Two main groups were created for the pedestrian actions: Case A: crossing a street and Case B: walking along a street. These two scenarios are shown in Figure 1.

For each pedestrian action case, six sets of experiments were performed and recorded. At least one pedestrian was observed in each video clip at the nearest detected obstacle. The test plan is shown in Table 1 and Table 2.

Table 1. Case A Data Collection Experiments

Case / Data	A1	A2	A3	A4 right	A4 left	A5 right	A5 left	A6	A7
Experiments	6	6	6	6	6	6	6	6	6

Table 2. Case B Data Collection Experiments

Case / Data	B1	B2	B3
Experiments	6	6	6

2.3 ADAS Camera Data Selection

Most of the current ADAS cameras generate a lot of information pertaining to the observed obstacles, such as obstacle type, position, velocity, the angle between the Ego Vehicle and the obstacle, the rear light of the front vehicle, the height, and the width of the obstacle, the TTC, etc. These cameras also provide some information about the road, traffic signs, and barriers that can be found on the roadside. Only pedestrian data from the ADAS camera was used in this study. The data types that were used in this study are listed in Table 3.

Table 3. Data Types Used for Pedestrian Prediction

Type	Symbol	Range	Unit
Obstacle classification	Obstacle_type	n/a	n/a
Lateral position	Lat_pos	-128-128	m
Lateral velocity	lat_vel	-128-128	m/s
Longitudinal position.	long_pos	0-256	m
Longitudinal velocity	long_vel	-128-128	m/s
Longitudinal acceleration	long_acc	-20-20	m/s^2
TTC based on constant velocity	ttc_vel	0-7	s
TTC based on constant acceleration	ttc_acc	0-7	s
pedestrian is on the pavement	ped_occ	0-1	n/a

To make it easier to extract video clips containing pedestrians for further analysis, a pedestrian clipping MATLAB program is written to detect videos containing pedestrians automatically. For example, the pedestrian and the white car are both detected in Figure 2, and the observed classes of pedestrians and vehicles are shown in the left column of Figure 3. The MATLAB program reads the obstacle type information and saves all pedestrian video clips. Then the pedestrian's actions in each video were manually classified and grouped. The frame number of the pedestrian actions fragmented in each video was recorded from the whole video. The subframe of the number of pedestrian actions fragmented in each video clip can be found from the frame number. The pedestrian actions fragment was cut out by this step. The right column in

Figure 3 shows the screenshot of one part of the look-up table that records the start and end frame numbers for the whole video and the start and end frame numbers for each video clip.

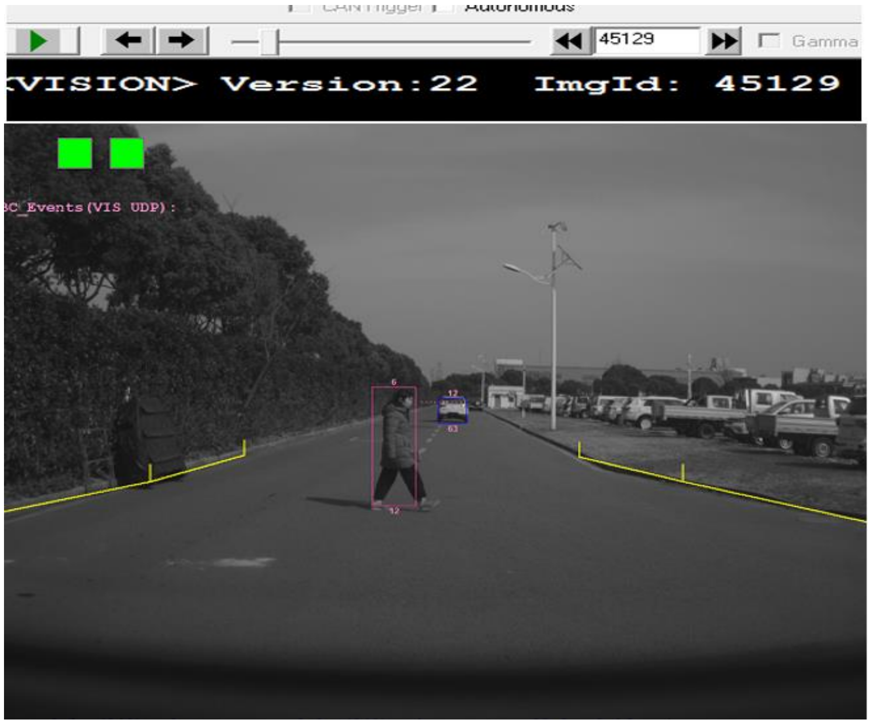
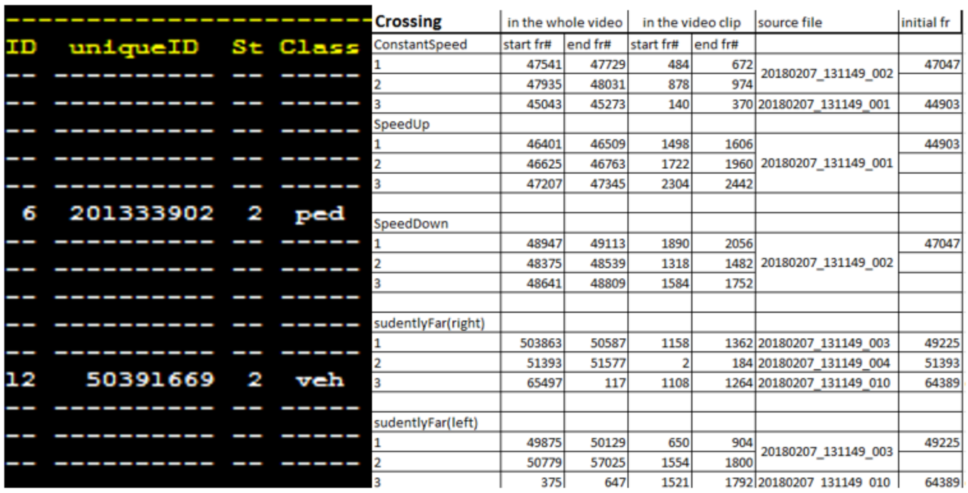


Figure 2. The Detection of a Pedestrian and a Vehicle by the ADAS Camera.



```
------------------------
ID  uniqueID   St Class
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
 6  201333902   2  ped
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
12   50391669   2  veh
--  ---------- -- -----
--  ---------- -- -----
--  ---------- -- -----
```

Crossing	in the whole video		in the video clip		source file	initial fr
ConstantSpeed	start fr#	end fr#	start fr#	end fr#		
1	47541	47729	484	672	20180207_131149_002	47047
2	47935	48031	878	974		
3	45043	45273	140	370	20180207_131149_001	44903
SpeedUp						
1	46401	46509	1498	1606	20180207_131149_001	44903
2	46625	46763	1722	1960		
3	47207	47345	2304	2442		
SpeedDown						
1	48947	49113	1890	2056	20180207_131149_002	47047
2	48375	48539	1318	1482		
3	48641	48809	1584	1752		
sudentlyFar(right)						
1	503863	50587	1158	1362	20180207_131149_003	49225
2	51393	51577	2	184	20180207_131149_004	51393
3	65497	117	1108	1264	20180207_131149_010	64389
sudentlyFar(left)						
1	49875	50129	650	904	20180207_131149_003	49225
2	50779	57025	1554	1800		
3	375	647	1521	1792	20180207 131149 010	64389

Figure 3. Recognized Classes of Observed Obstacles (left) and Recorded Frame Numbers of Pedestrian's Action Fragments to be Cut Out (right).

3 EXTENDED KALMAN FILTER FOR PEDESTRIAN TRAJECTORY PREDICTION

Kalman filter is a good noise reducing and data smoothing filter for a linear random noise process such as an airplane's flying or landing movements. However, a pedestrian's movement tends to be a non-linear random process, thus an extended Kalman filter should be used because it can filter noises of a non-linear process. An extended Kalman filter has two state parameters x and p; x is defined as the state, including the position and velocity information; p is defined as the covariance which will be updated with the previous x state.

The ADAS camera can convert the coordinates from the Ego Vehicle's view to a top view and the pedestrian prediction implements the EKF with the pedestrian position in the top view coordinates. In this EKF process, the new state is always predicted and updated from the observed pedestrian position when there is no instant action.

3.1 Extended Kalman Filter

Kalman filter is used to predict states recursively in a linear system, and it will take every measurement and previously predicted state into a new state estimation [16]. The extended Kalman filter is a non-linear extension of the Kalman filter

3.1.1 Review of Kalman Filter

The Kalman filter has two main phases: the prediction phase and the update phase. For the prediction phase, the filter will predict the current state using the previous state. Prediction equations are shown below [17]:

$$x_{k|prediction} = ax_{k-1|update} \quad \text{(Eq 3.1)}$$

$$p_{k|prediction} = ap_{k-1|update}a \quad \text{(Eq 3.2)}$$

The variable x is the estimated state, and k is the state number. $x_{k|prediction}$ is the predicted k^{th} state from the prediction phase, and $x_{k-1|update}$ is the updated $(\mathrm{k}-1)^{\mathrm{th}}$ state from the update phase. p is the covariance of state x. $p_{k|prediction}$ is the predicted p of the k^{th} state from the prediction phase and $p_{k-1|update}$ is the updated p of $(\mathrm{k}-1)^{\mathrm{th}}$ state from the update phase. a is a constant. For the update phase, the filter will update the current state using the new true measurement. The update equations are shown below[17]:

$$g_k = \frac{p_{k|prediction}}{p_{k|prediction} + r} \quad \text{(Eq 3.3)}$$

$$x_{k|update} = x_{k|prediction} + g_k(z_k - x_{k|prediction}) \quad \text{(Eq 3.4)}$$

$$p_{k|update} = (1 - g_k)p_{k|prediction} \quad \text{(Eq 3.5)}$$

g_k is the current gain. r is the average noise of the input data. z_k is the observation measurement.

3.1.2 Extended Kalman Filter for Pedestrian Trajectory Prediction

In this section, the detailed EKF implementation is discussed, and the initial states of the EKF state parameters are defined. The EKF is applied when there is no instant action of a pedestrian, while the observed pedestrian position data will be the input data to the EKF, and the output will be the prediction of the next state of the pedestrian position.

The EKF contains two phases: prediction and update. In the prediction phase, the new state and new covariance parameter are predicted. In the update phase, a system gain G is calculated from the new predicted covariance parameter which comes from the prediction phase, and the system gain is used to update the current state.

In the Prediction Phase of the EKF, the equations are listed below:

$$x_{k|prediction} = A * x_{k-1|update} + B * u_k \qquad \text{(Eq 3.6)}$$

$$p_{k|prediction} = A * p_{k-1|update} * A' + \mathrm{Q} \qquad \text{(Eq 3.7)}$$

The state x is a 4x1matrix, and the new state $x_{k|predicton}$ is predicted by the previous updated state$x_{k|update}$. B is a scale of the input control signal $\mathrm{u_k}$. For the initial state x_0, the value is set to $[lat_{pos0}\ long_{pos0}\ lat_{vel0}\ long_{vel0}]$ where lat_{pos0} is the initial latitude position, $long_{pos0}$ is the initial longitude position, lat_{vel0} is the latitude velocity, and $long_{vel0}$ is the longitude velocity. lat_{pos0} and $long_{pos0}$ will be initialized to the center position. lat_{vel0} and $long_{vel0}$ will be initialized to 0. For the scale B and the input control signal u_k in this study, there is no extra control signal applied, so B and u_k have been set to zero.

The covariance parameter p_k is predicted by a constant matrix A and the state error covariance matrix Q. The initial value of p_k needs to be an identity matrix times a nonzero parameter when k is 0. If the estimate of the initial state x_0 is accurate, p_0 can be initialized to be a small value. If the estimate of x_0 is far from the true value, p_0 should be initially set to be a large value.

The constant matrix A is used in the prediction step to calculate new state $x_{k|predicton}$from the previous state $x_{k-1|update}$and calculate the new covariance parameter $P_{k|predicton}$from the previous $p_{k-1|update}$. Matrix A should be a 4x4 Jacobian matrix for the 4x1 state x and the matrix A should be set to $[[1,0,0,0],[0,1,0,0],[dt,0,1,0],[0,dt,0,1]]$ by calculating the Jacobian matrix of state x.

The state error covariance Q should be set to a very small value according to the covariance matrix p. In this study, Q was set to $0.01 * I$, where I is the identity matrix.

Equations for the update phase of the EKF are listed below:

$$G_k = P_{k|prediction} * H' * (H * p_{k|prediction} * H' + R)' \qquad \text{(Eq 3.8)}$$

$$x_{k|update} = G_k * z_k + (I - H) * x_{k|prediction} \qquad \text{(Eq 3.9)}$$

$$p_{k|update} = (I - G_k * H) * p_{k|prediction} \qquad \text{(Eq 3.10)}$$

The system gain G_k can be calculated from the constant matrix H, the state covariance matrix p_k and the measurement error covariance R. The constant matrix H is used for mapping the state x_k to the observation z_k, which is a 2x1 Jacobian matrix of the position $[lat_{posk}\ long_{posk}]$ of x_k. The measurement error covariance R is calculated by the Mobileye camera based on the measurement noise.

State $x_{k|update}$ should be updated based on the system gain G_k observation z_k and the prediction state $x_{k|predicton}$. The state covariance $p_{k|update}$ should be updated based on the system gain G_k and the prediction state covariance $P_{k|predicton}$.

3.2 Application to Pedestrian Trajectory Prediction

The EKF takes pedestrian position data from the Mobileye ADAS camera as the input to predict the pedestrian trajectory. There are two types of pedestrian position data. Both are provided by the Mobileye in two different coordinates, the vehicle view and the top view. The vehicle view is based on the camera, and the pedestrian location data is based on the pixel information. The top view is based on the Ego Vehicle and a pedestrian. The pedestrian position data is the real distance between the Ego Vehicle and the surrounding obstacles. The top view is more accurate and used in the AEB Pedestrian function of the ADAS.

There were two situations in this study; one was the normal pedestrian movement without any instant actions, the other was the instant actions of a pedestrian. The EKF was used to predict the next step of the pedestrian trajectory without any instant actions, because the prediction was more accurate for the normal action and had less deviation for the instant actions.

3.2.1 Coordinate Conversion from Vehicle View to Top View

In Figure 4, there are two coordinate systems: the first one is the Vehicle View of the pedestrian detected in the video of the view of the Ego Vehicle, which is shown in Figure 4(a). The x-axis is measured in pixels along the top horizontal edge of the video, while the y-axis is measured in pixels along the right vertical edge of the video.

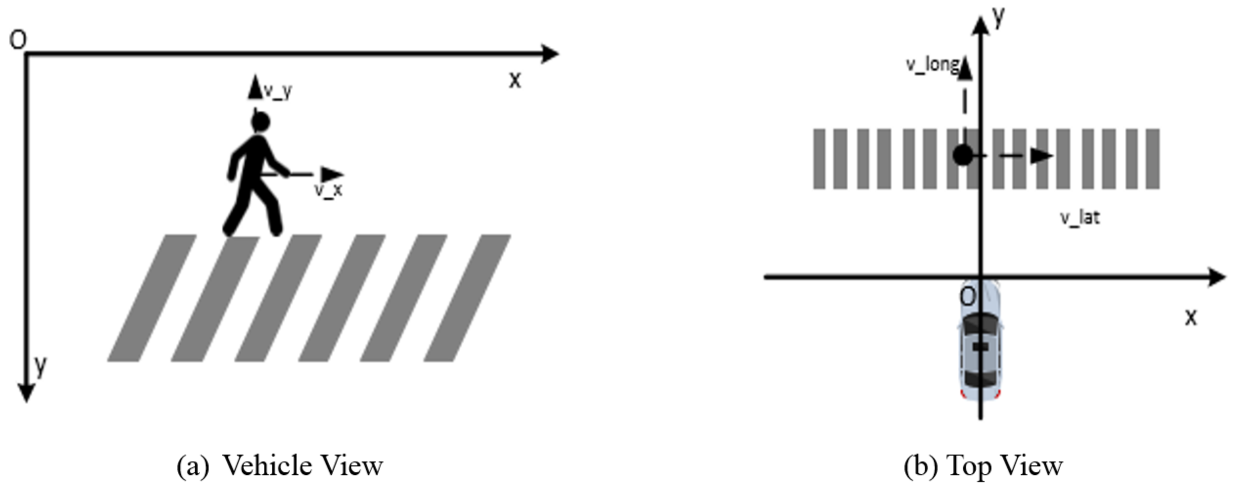


Figure 4. The Camera View and Top View of the Pedestrian.

The second one is shown in Figure 4(b), where the scene includes the pedestrian and the Ego Vehicle, which is the Top View used in this study. The x-axis is the latitude position in meters, and the y-axis is the longitude position in meters, where the coordinate system is based on the pedestrian and the Ego Vehicle.

The coordinate conversion was done because calculations of velocity, acceleration, heading angles, and TTC are based on the Top View Coordinate System. But to display the video from an ADAS camera, the Vehicle View Coordinate System was used, where the pedestrian pixel positions X and Y were used instead of the latitude and longitude positions.

3.2.2 Analysis of Extended Kalman Filter for Pedestrian Trajectory Prediction

The EKF can predict a new state of pedestrian trajectory quickly with high accuracy based on normal movement with no instant action on the part of the pedestrian. In this study, only when the instant actions were not detected by the classification method, was the EKF applied to the pedestrian trajectory prediction.

In Figure 5, the EKF is applied to predict the pedestrian trajectory. The dark rectangle block at the bottom represents the Ego Vehicle. The small dark red square represents the observation z_k, and the yellow circle represents the predicted state x_k. The Ego Vehicle is located at the center of the latitudinal position. The x-axis of Figure 5 is the latitudinal position in meters, while the y-axis is the longitudinal position in meters. Two graphs in Figure 5 are adjacent frames, and the real movement of the pedestrian is from left to right. Based on the observation position z_k of the current pedestrian, the EKF provides the pedestrian trajectory prediction for one step ahead to the right. The predicted position in frame 24 is [1.2277 7.4731] and the next observed position is [1.2304 7.4686] in frame 25. Therefore, the EKF can predict the pedestrian trajectory when the pedestrian walks on the street with no instant action.

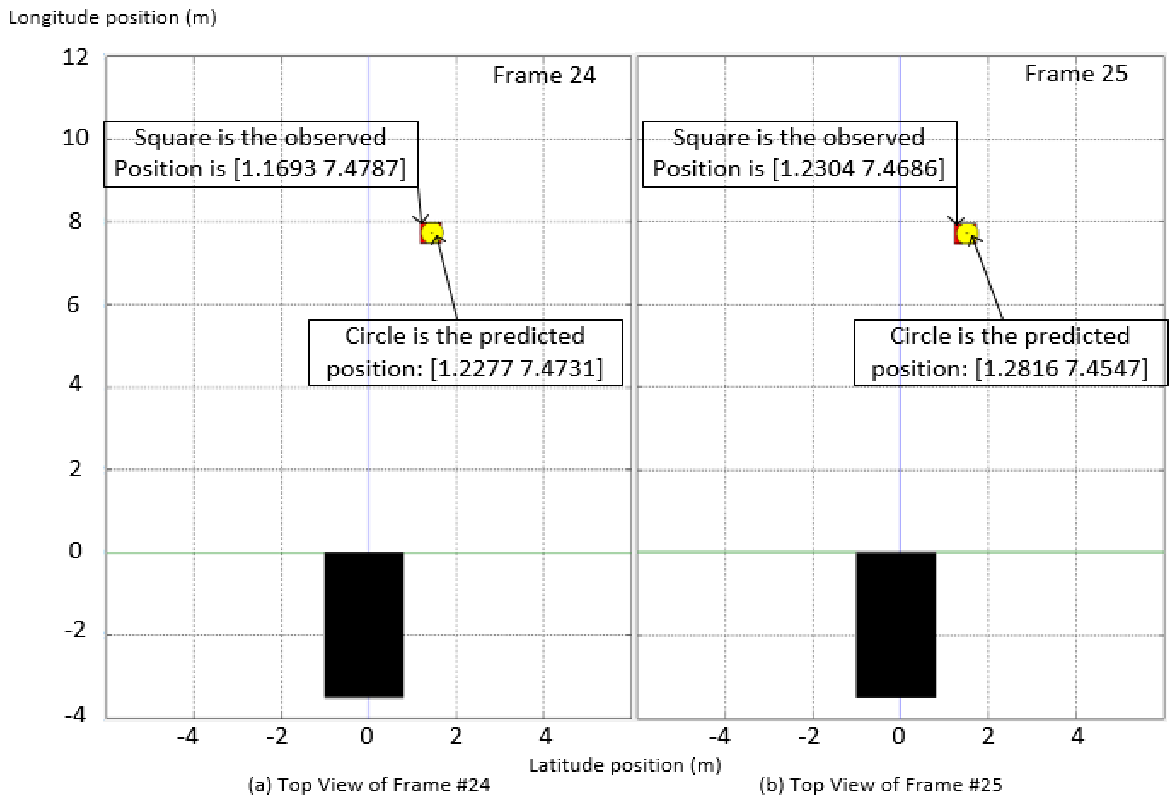


Figure 5. One Step Prediction of Pedestrian with EKF in the Top View Coordinates.

In Figure 6, pedestrian positions are shown in two adjacent frames of Case A1. Case A is defined as a pedestrian crossing a street at a constant speed with no change of direction. The moving direction of the pedestrian is from left to right. The x-axis is the latitudinal position of the pedestrian, and the y-axis is the longitudinal position of the pedestrian. The diamond-shaped position is the original observed position of the pedestrian, and the circle-shaped position is the position prediction from the EKF. In this EKF prediction, the result is very accurate for Case A1.

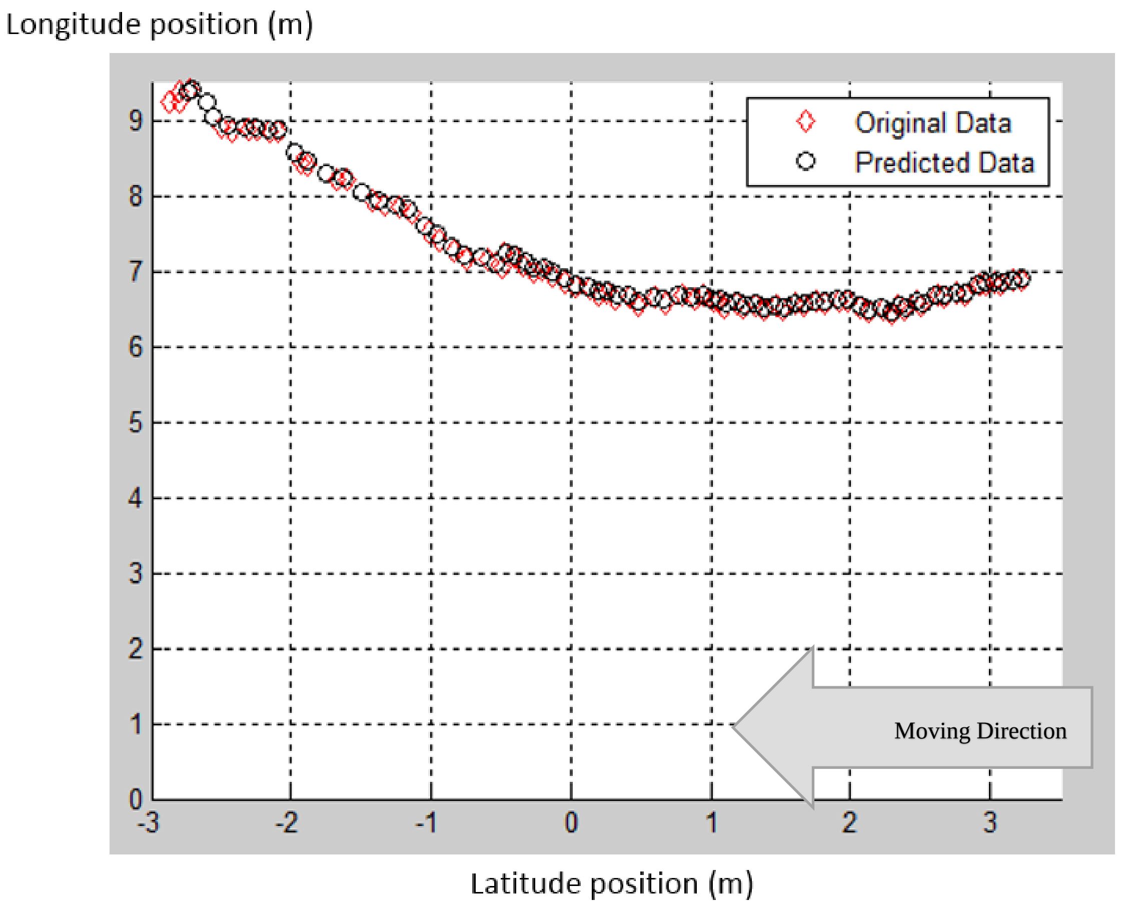


Figure 6. The EKF Prediction of Pedestrian Crossing a Street at Constant Speed.

In Figure 7, the pedestrian positions are shown in two adjacent frames of Case A4 Right. Case A4 Right is that a pedestrian suddenly changes direction when crossing the street. The moving direction is from right to left. The diamond-shaped position is the original observed position of the pedestrian, and the circle-shaped position is the prediction position from the EKF. In this situation, the pedestrian has made an instant action, so the prediction is not accurate when the pedestrian changes direction.

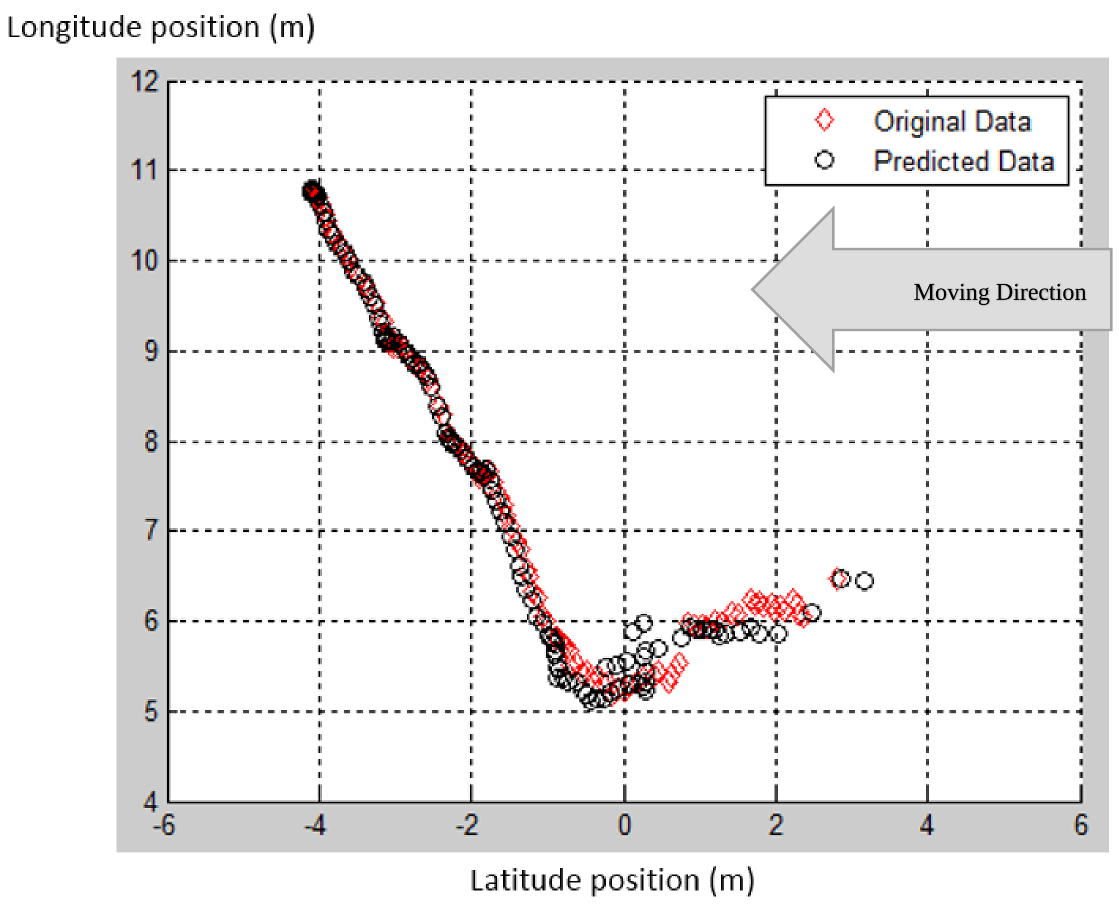


Figure 7. The EKF Prediction of Pedestrian Changing Direction Suddenly.

4 BEHAVIOR CLASSIFICATION FOR PEDESTRIAN TRAJECTORY PREDICTION

Most ADAS cameras mounted on the Ego Vehicle can provide information such as the class of each observed obstacle, distance between the Ego Vehicle and each obstacle, relative position of each obstacle, velocity of each obstacle and the angle between the Ego Vehicle and the obstacle, velocity and yaw-rate of the Ego Vehicle and the TTC and the brake warning of each obstacle. An ADAS camera, which is well-suited for a Vehicle-to-Vehicle (V2V) AEB system, can accurately calculate the TTC faster and set brake warning signals more correctly, because in the V2V system the front vehicle moves at either constant speed or constant acceleration.

However, TTC information about a pedestrian that comes from an ADAS camera is not suitable for a real-world situation and may causes considerable errors sometime, because the pedestrian trajectory prediction is not as accurate compared to the vehicle trajectory prediction, and predicted velocity and the acceleration of the pedestrian may have a lot of noises. A pedestrian does not usually move at constant speed along a straight line but moves suddenly and occasionally. The pedestrian’s instant velocity, acceleration and heading angle from an ADAS camera are less accurate than those of the V2V data are. These pedestrian measurement errors will be propagated into the pedestrian’s tracking and adversely affect the V2P AEB Pedestrian function.

Consider the case when the pedestrian walks along a street, and suddenly chooses to cross the street. Or the pedestrian could be crossing the street and suddenly decide to turn around. The ADAS camera would not be able to recognize these sudden behavioral changes of a pedestrian very well, and it could report an inaccurate TTC to the AEB Pedestrian system of the ADAS. Therefore, the current pedestrian trajectory prediction algorithm inside the ADAS camera needs to be improved by making it capable of tracking a pedestrian's instant action and removing video noises.

Z. Chen et al. in their paper on pedestrian action classification proposed a method that used the motion pattern to cluster the pedestrian trajectories [18]. The motion pattern includes location, velocity and heading angle of a pedestrian. However, their method requires the fixed starting and ending locations of a pedestrian. From another aspect of pedestrian trajectory prediction, a survey shows that pedestrians can be grouped into three safety levels by traffic lights [19]. Pedestrians can also be grouped by different environments and dates such as different cities and whether it is a weekday or a weekend [20]. More specific behaviors of pedestrians that influence pedestrian trajectories are listed in [21].

In this thesis, a new approach is proposed and has also been verified to improve prediction of the heading angle and the acceleration of a pedestrian in front of an Ego Vehicle from the video stream recorded by an ADAS camera on the Ego Vehicle. The main idea of this portion of this thesis is to calculate the pedestrian velocity more accurately from the position data of the pedestrian by averaging velocities over 30 frames to reducing noise due to velocity fluctuation. It will be shown that the heading angle and the acceleration of the pedestrian estimated from the new velocity are more stable and more realistic than those obtained from the velocity calculation using just the raw video data.

Besides increasing the prediction accuracy, time complexity is a significant factor for an AEB Pedestrian System. In this study, the frame rate is 36 frames per second and the camera will record odd and even frame numbers into two video channels separately. Each pedestrian action takes 1.66 seconds reaction time for the prediction, which is shorter than the average driver reaction time of 2.3 seconds [22].

4.1 Trajectory Prediction Based on Classification

In this thesis, behavior classification is implemented based on the heading angle and the acceleration of a pedestrian, which is used to classify instant actions of the pedestrian and to predict the next position of the pedestrian.

The position data of a pedestrian is used to calculate the velocity of the pedestrian. In order to remove the data fluctuation that causes velocity errors, the sampling points will be removed from the velocity calculation if the fluctuation is larger than the threshold. The velocity accuracy is further enhanced by taking a running average of velocities over 30 frames. The average velocity is then used to calculate the heading angle and the acceleration of the pedestrian.

Pedestrian behavior cases were defined in Section 2.1 on Page 7. For convenience, the definitions of behavior cases are repeated here: For Case A: (A1) Constant Speed, (A2) Sudden Acceleration, (A3) Sudden Deceleration, (A4) Sudden Dodge from the Ego Vehicle, (A5) Sudden Advance to the Ego Vehicle, (A6) Sudden Withdraw, and (A7) Sudden Stop at the Road Edge. For Case B: (B1) Constant Speed, (B2) Sudden Turn towards the Ego Vehicle and (B3) Sudden Stop at the Road Edge.

Several pedestrian's behavior cases were extracted from the videos recorded by a Mobileye ADAS camera. Raw instant velocities calculated from these videos were found to be unstable and noisy. Heading angles and accelerations for these cases were calculated using both raw instant velocity and the average velocity.

For Case A1 the heading angle and acceleration should not have any changes. For Case A2 and Case A3, the instant actions will be detected based on the acceleration of the pedestrian. If the change of acceleration is from low to high, the pedestrian is in Case A2; vice versa for Case A3. For Case A4 and Case A5, the instant actions will be detected based on the heading angle and the starting position of the pedestrian. The pedestrian's movement can be detected from the heading angle, while the instant action can be detected from both the pedestrian's movement and starting position. For Case A6 and A7, the instant actions will be detected based on the heading angle of the pedestrian. If the heading angle has a 180-degree change, the instant action will be in Case A6, and if the heading angle has a 180-degree change and no changes for 30 frames, the instant action will be in Case A7. For Case B2, instant actions will be detected based on the heading angle of the pedestrian. If the heading angle has a 90-degree change and stays that way for over 5 frames, the instant action is Case B2. Case B3 will not have a heading angle change and the velocity of the pedestrian will be zero.

More examples of pedestrian trajectory prediction by classification are shown in Section 4.2.3 and 4.3.3.

4.2 Average Velocity Calculation with Observed Position

Pedestrian velocity is critical in accurately calculating heading angle and acceleration. The pedestrian velocity data obtained directly from the ADAS camera is not very accurate. The range of velocity is ±128 meters per second, which is well-suited for vehicle tracking but is not suitable for measuring pedestrian velocity since a pedestrian's movement is a non-uniform motion. Pedestrian velocity can be calculated more accurately from the observed position information taken directly from the video stream.

4.2.1 Calculation of Average Velocity with Position Data

A pedestrian's position from the ADAS camera data contains both the latitude position and longitude position. The range of the latitude position is from -128 meters to 128 meters, and the center latitude position at the Ego Vehicle is 0 meters. The range of the longitude position is from 0 to 256 meters, and the closest position of longitude at the Ego Vehicle is 0 meters.

Position information taken from video frames can be used to calculate the velocity. The frame rate of the camera is 36 frames per second and the video is divided into two channels; therefore, the frame rate of each channel is 18 frames per second. Frame time interval dt is 0.055 seconds. The latitude velocity can be calculated as $v_{lat} = dx_{lat} / dt$, where dx_{lat} is the latitude position change between two adjacent frames. The longitude velocity can be calculated as $v_{long} = dx_{long} / dt$, where dx_{long} is the corresponding longitude position change.

More examples of new velocity predictions are shown in Figure 11 and Table 6 in Section 5 associated with the new acceleration calculation.

4.2.2 Noise Reduction from Velocity Calculation

Latitudinal velocity and longitudinal velocity calculated directly from the video position data can be very noisy. The instant velocities calculated from the observed position are shown in the first column of Figure 8 for Case A1 and Figure 9 for Case A4. The noises are caused by the errors in the observed position and characteristics of the pedestrian. A noise filter needs to be applied to remove the noises.

In order to remove these noises, the variance of the instant velocity is calculated and the upper bound of the variance is set. Any sample points that have higher variances than the upper bound are removed from the instant velocity points. The 30-frame moving average of the remaining instant velocity points is then used to obtain the average velocity. For example, if the variance of the latitude velocity is higher than 1.1, the corresponding velocity is removed from the velocity calculation. As a result, the average velocity becomes smoother and less noisy. Figure 8 and Figure 9 show the comparison of the raw velocity and average velocity. The first column is the latitude velocity with noise. The second column is the variance value. The third column is the noise-reduced velocity. The fourth column is the mean of the noise-reduced velocity.

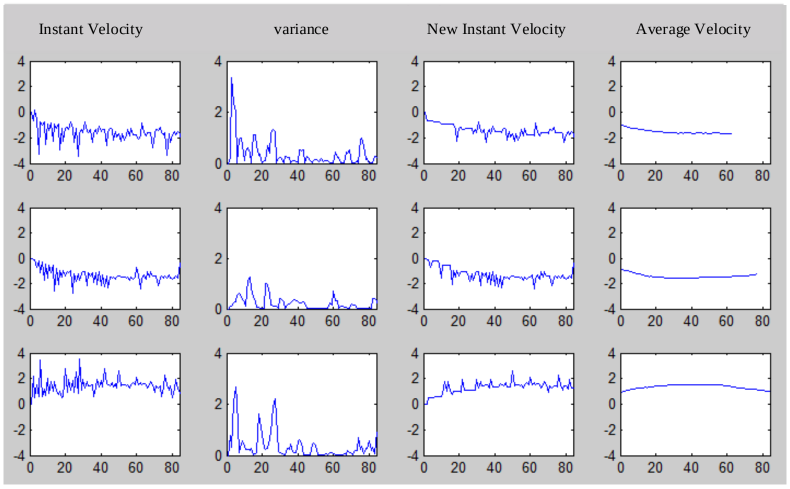


Figure 8. Average Velocity Calculation for Case A1.

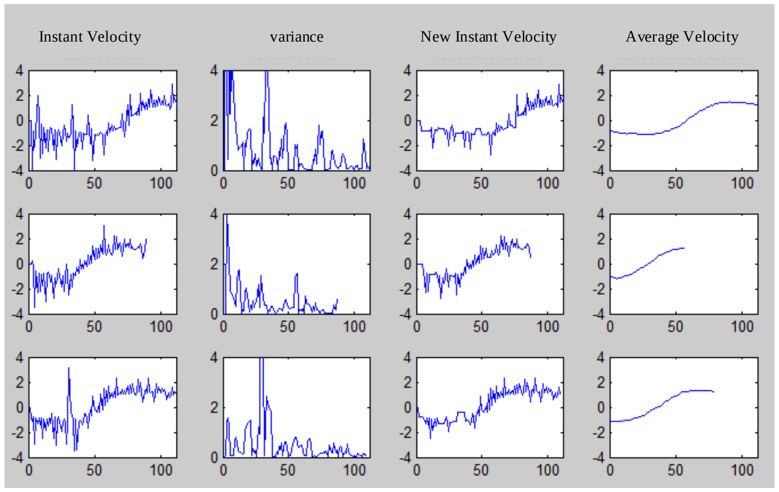


Figure 9. Average Velocity Calculation for Case A4.

Compared to the velocity from the ADAS camera, the filtered average velocity is more stable and smoother. The first column in Figure 10 shows the average velocity for Case A1 and the second column shows the instant velocity. It is clear that the average velocity is much smoother and more stable. Data for Case A4 in the two right columns in Figure 10 supports this conclusion as well.

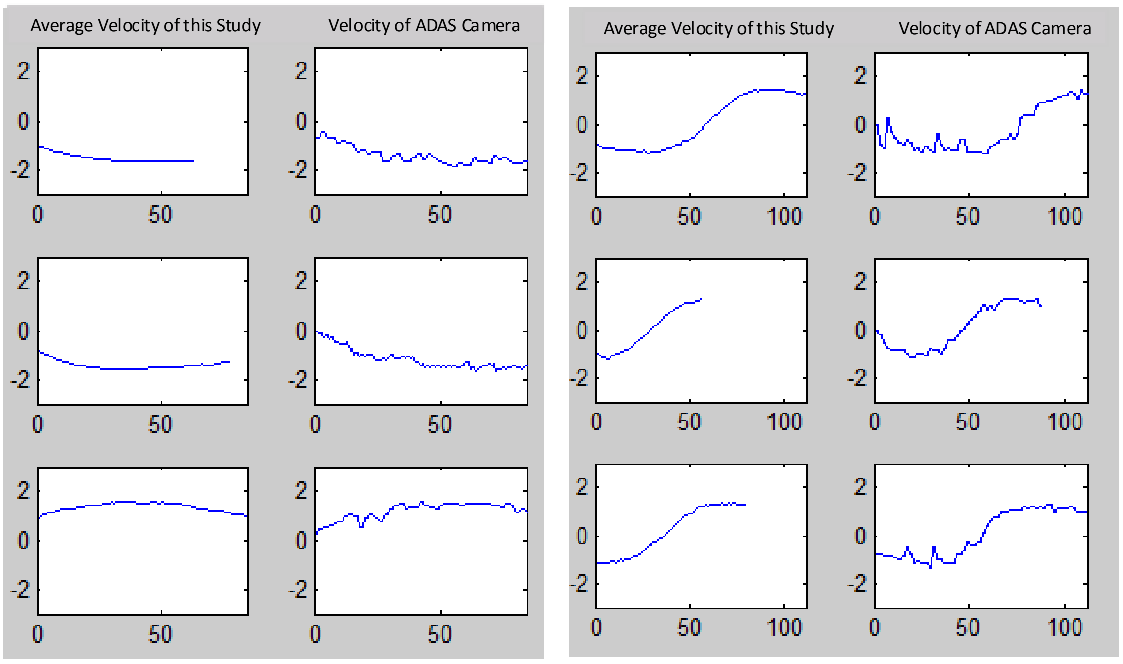


Figure 10. Comparison of Average Velocity and ADAS Velocity for Case A1 and Case A4.

4.3 Prediction of Heading Angle

The heading angle of a pedestrian crossing a street is the deviation angle from the straight line of crossing. Four pedestrian action cases have angle changes: Case A4, Case A5, Case A6 and Case B2. The first two cases will cause a slight angle change which may be less than a 90-degree change, Case A6 may cause an angle change of up to 180 degrees, and Case B2 may cause up to a 90-degree change.

Pedestrian velocity taken directly from the ADAS camera data along the x- and y-axes can be used to calculate the pedestrian heading angle [23], and the heading angle theta can be calculated using $arctan(vel_{long}/vel_{lat})$. The vel_{long} is the longitudinal velocity and vel_{lat} is the latitudinal velocity. The angle's range is between -90 and 90 degrees.

The heading angle is calculated with the ADAS camera velocity data using the following equation:

$$\theta_{old} = \arctan(\frac{vel_{long|cam}}{vel_{lat|cam}}) \quad \text{(Eq 4.1)}$$

The heading angle θ_{old} is directly calculated from the ADAS camera data, which is not accurate. The upper left six graphs for Case A1 in Figure 11 show how inaccurate this calculation is. Both the heading angle and the moving angle theta of these three examples of Case A1 data sets should ideally be 0 degree, but θ_{old} in these three datasets are always changing and far from 0 degree.

4.3.1 A New Method of Computing the Heading Angle

A new method to calculate the moving angle theta is proposed in this thesis by using the new average velocity calculated from the position information from the ADAS camera data as discussed in Section 3. The heading angle θ_{new} is calculated as follows:

$$\theta_{new} = \arctan(\frac{vel_{long|new}}{vel_{lat|new}}) \quad \text{(Eq 4.2)}$$

Figure 11 shows a number of comparisons of the old camera data and the new calculated heading angle for Case A. Figure 12 shows the heading angle comparisons for Case B2, including the case of a pedestrian crossing street at a constant velocity straightly and the pedestrian moving farther away from the Ego Vehicle while crossing the street with an obvious direction change.

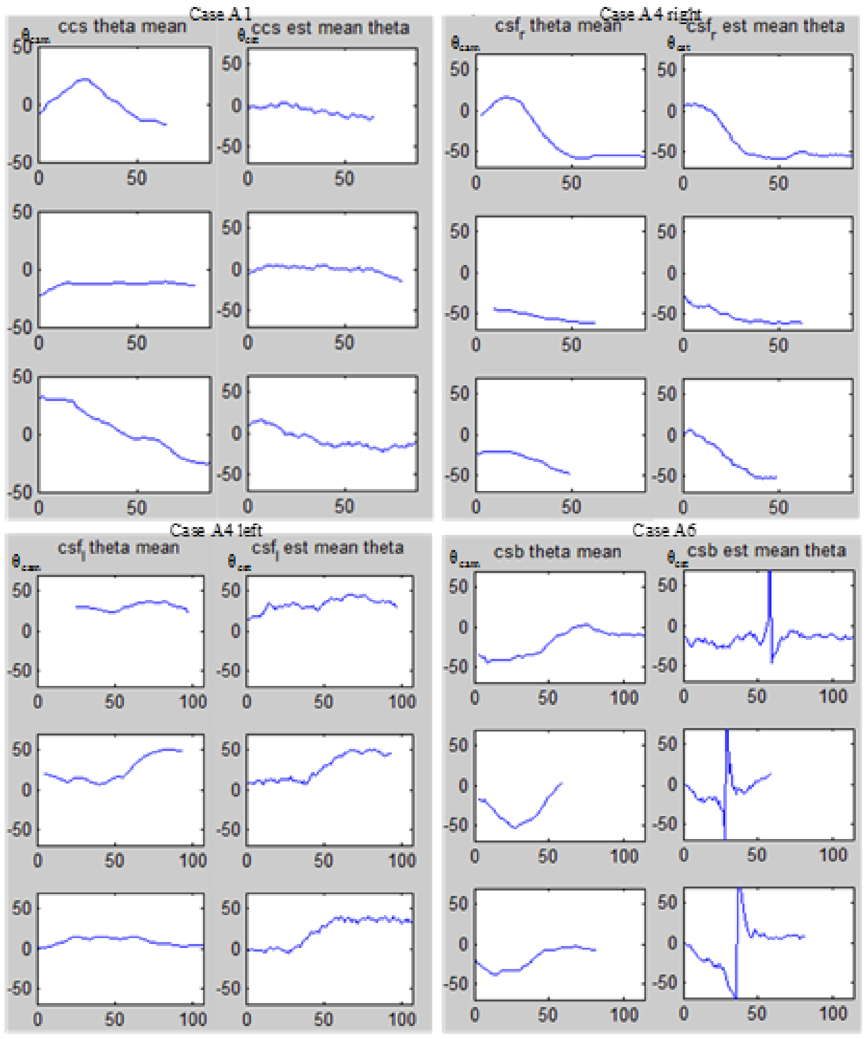


Figure 11. Comparison of Old Camera Data (right) and New (left) Heading Angles for Cases A1, A4 Left, A4 Right and A6.

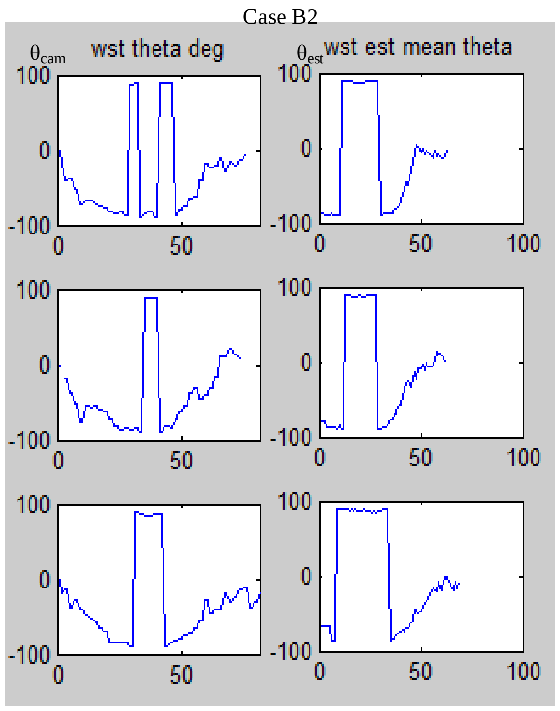


Figure 12. Comparison of Old Camera Data (right) and New (left) Heading Angles for Case B2.

4.3.2 Analysis of More Accurate Prediction of the Heading Angle

The heading angle for Case A1 is 0 degree, as the pedestrian walks in a straight line at constant speed. The heading angle for Case A4 shows a sudden change from 0 degree to ±50 degrees as the pedestrian dodged suddenly from the Ego Vehicle. The heading angle for Case A6 shows a sudden jump from -90 to 90 degrees or 90 to -90 degrees, because for both before and after the stopping point of the longitudinal, the velocity is ideally equal to 0. In addition, at the point where pedestrian stops and make a U-turn, the angle changes by 180 degrees.

The heading angle for Case B2 should be about 90 degrees when the pedestrian suddenly makes a 90-degree turn. The velocity should decrease to zero and the heading angle should change suddenly to -90 degrees, then return to 0 degree.

Table 4 shows the comparison of the ideal, old, and new heading angle predictions for Cases A1, A4, and A6. It is clearly shown that the new heading angle predictions for all cases are more stable and more accurate than the old ones.

Table 4. Comparison of Ideal, Old and New Heading Angle Predictions

Pedestrian actions	Ideal θ	θ from camera	θ from this study
Case A1	Constant at 0 degree	2 datasets fluctuated	All Stable
Case A4	Sudden from 0 to ± 50	2 datasets have no obvious angle change	All data sets are better than the camera data
Case A6	Sudden from -90 to +90 at stop point	None of them has obvious angle change	All data sets have obvious angle change
Case B2	Sudden from -90 to -90 then 0 with no bouncing	1 dataset has bouncing, this will cause system to recognize 2 turnings	None of them bouncing.

4.3.3 Pedestrian Trajectory Prediction by Heading Angle in Classification

Some cases of pedestrian's instant actions can be detected and classified based on the pedestrian heading angle. For example, in Case A6, the instant actions will be grouped into Class 1 corresponding to the case of a pedestrian suddenly turning back. And in Case A4 and Case A5, the change of heading angle is used to detect these actions. Class 2 for suddenly turning closer to the Ego Vehicle and Class 3 for suddenly turning even closer to the Ego Vehicle. Classes 4 and 5 are based on acceleration and will be discussed in Section 4.4.3. For Case B2, the instant action class is Class 6. The class definition is shown in Table 5 below.

Table 5. Action Classes and Corresponding Action Cases

Actions Classes	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Actions Cases	Case A6	Case A5	Case A4	Case A2	Case A3	Case B2

In Figure 13, two instant actions are detected separately. One is Case A6 on the left, the other is Case A4 on the right. In this classification process, the change rates of the heading angle over 30 frames can be observed and the class number will be signed for each action. When the instant action occurs, the class will be signed to the corresponding case.

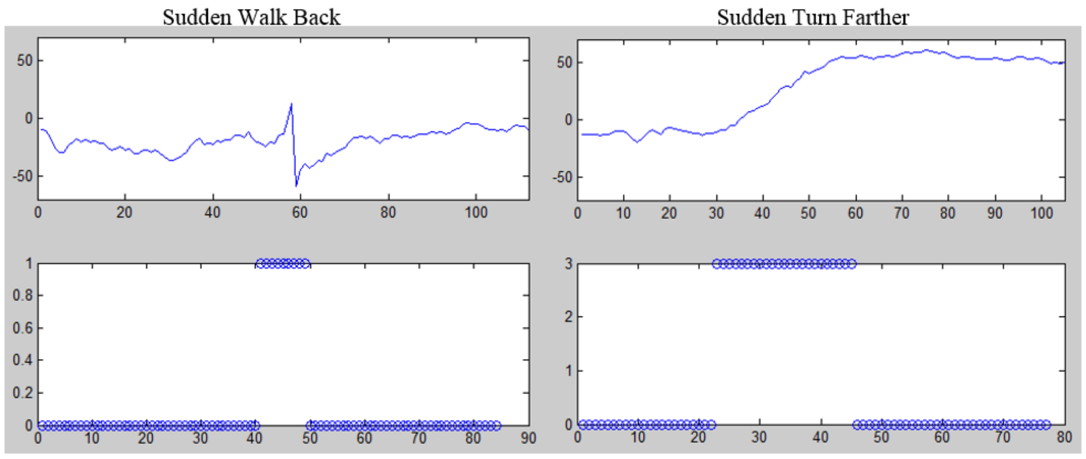


Figure 13. Detection of Sudden Withdrawal (Case A6) and Sudden Dodge (Case A4).

4.4 Prediction of Acceleration

Another important factor for predicting pedestrian trajectories is pedestrian acceleration. The velocity change $dv_{lat|cam}$ can be obtained from the latitude velocity change between two adjacent frames and the acceleration can be calculated as follows:

$$acc_{old|lat} = \frac{dv_{lat|cam}}{dt} \qquad \text{(Eq 4.3)}$$

Using the velocity information directly from the ADAS camera to calculate the pedestrian latitude velocity will produce an unstable acceleration prediction, as shown in Figure 14. The first two columns in Figure 14 are velocity and acceleration obtained from the camera data directly for three cases: Case A1 Constant Speed, Case A2 Acceleration and Case A3 Deceleration. Acceleration in all three cases shows unstable glitches.

4.4.1 A New Method of Computing the Acceleration

The way to get smoother and less noisy acceleration prediction is to use the new average velocity. The new acceleration $acc_{new|lat}$ is calculated as follows, where the velocity change is based on the average velocity:

$$acc_{new|lat} = \frac{dv_{lat|new}}{dt} \qquad \text{(Eq 4.4)}$$

The right two columns in Figure 14 are velocity and acceleration that have been calculated with the new average velocity prediction for Cases A1, A2, and A3. The accelerations are much more consistent than those based on the old velocity.

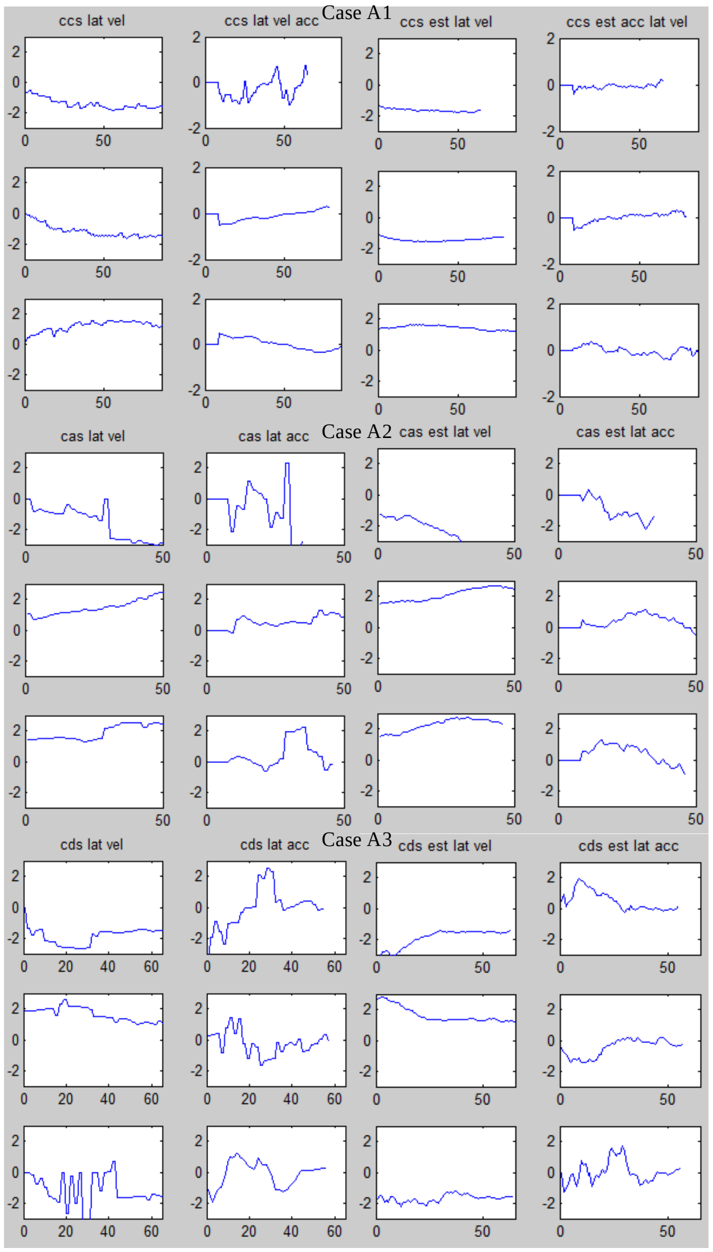


Figure 14. Old Velocity and Acceleration (Left Two Columns) and New Velocity and Acceleration (Right Two Columns).

4.4.2 Analysis of the More Stable Prediction of Acceleration

The velocity for Case A1 Constant Speed should be constant, and its acceleration should be zero. The velocity for Case A2 Acceleration should be increasing monotonically, and its acceleration should be positive. The velocity for Case A3 Deceleration should be decreasing monotonically, and its acceleration should be negative.

Table 6 shows the comparison of the ideal, old, and new velocity calculation for cases in Figure 14. Table 7 shows their acceleration comparison.

Figure 14, Table 6 and 7 have shown that the new methods of predicting velocity and acceleration proposed in this thesis are much more stable and accurate than the old methods of using the raw ADAS camera data.

Table 6. Comparison of Vel$_{idea}$ vs. Vel$_{cam|old}$ vs. Vel$_{est|new}$

Pedestrian Actions	vel_{idea}	$vel_{cam\|old}$	$vel_{est\|new}$
Case A1	A constant value	Speed is unstable and noisy	Stable
Case A2	Increasing	Two datasets have noise and unsmooth	Smoother increasing trend
Case A3	Decreasing	All datasets are noisy and unstable	Smoother and less noise

Table 7. Comparison of Acc$_{idea}$ vs. Acc$_{cam|old}$ vs. Acc$_{est|new}$

Pedestrian Actions	acc_{idea}	$acc_{cam\|old}$	$acc_{est\|new}$
Case A1	Constant at 0	One dataset is noisy and unstable	All datasets are stable
Case A2	Positive value	All datasets have a lot of noise	All datasets are more stable and smoother
Case A3	Negative value	All datasets have large noise	All datasets are better than the datasets from the camera

4.4.3 Pedestrian Trajectory Prediction by Acceleration in Classification

Some cases of pedestrian's instant actions can be detected by the classification based on pedestrian acceleration. For example, in Case A2, the instant actions class will be detected as Class 4 for a pedestrian suddenly speeding up. And in Case A3, the instant actions class will be detected as Class 5 for a pedestrian suddenly slowing down. The class definition is shown in Table 5 in Section 4.3.3.

In Figure 15, two instant actions are detected separately. One is Case A2 on the left, the other is Case A3 on the right. In this classification process, the change rates of the acceleration over 30 frames will be observed and the class number will be signed for each action. The left figure in Figure 15 is Case A2, and the class is set to Class 4 when the instant action happens. The right figure in Figure 15 is Case A3, and the class is set to Class 5 when the instant action occurs.

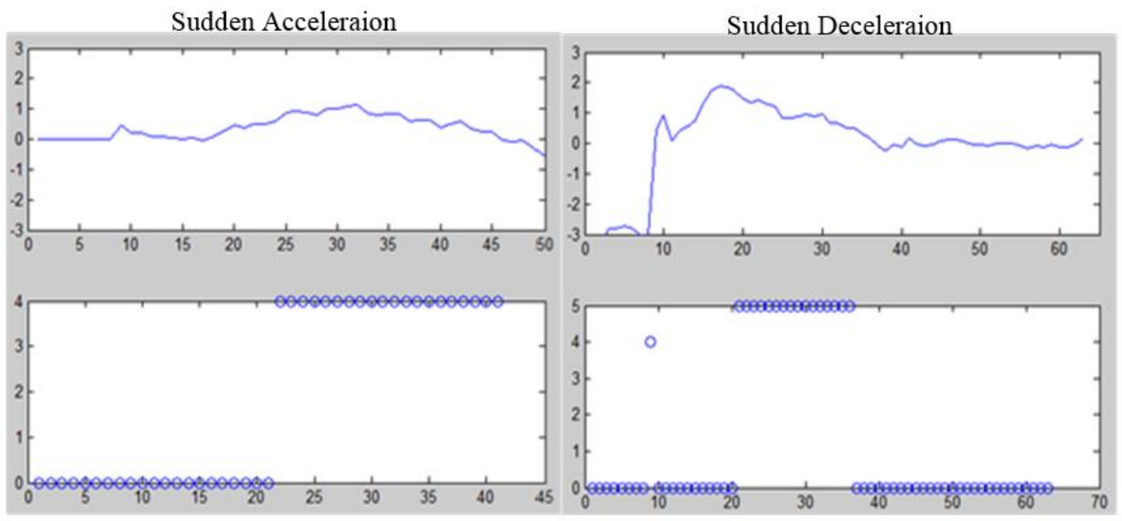


Figure 15. Detection of Sudden Acceleration (Case A2) and Sudden Deceleration (Case A3).

5 OPTIMIZED TRAJECTORY PREDICTION BY EKF AND BEHAVIOR CLASSIFICATION

In order to predict pedestrian trajectories accurately, extended Kalman filter (EKF) discussed in Section 3 and the behavior classification in Section 4 have been combined into one predictive system to perform the prediction. The EKF will reduce the observation noise of the detected pedestrian data, and the behavior classification will track instant actions of the pedestrian's sudden movement. In this prediction system integrated by the EKF and classification, the pedestrian data will initially be processed by the EKF process when the pedestrian is moving at constant speed. Once an instant action of one of the classification cases is detected by the system, the EKF process will stop calculation and reset its state and parameters. When the instant action detection is finished, the prediction system will restart the EKF process.

When this prediction system performs EKF filtering, the situation is defined as the EKF process. When this system stops the EKF and analyzes the pedestrian instant actions using behavior classification, this situation is defined as the Classification process. The main purpose of combining these two processes is to optimize the trajectory prediction by reducing the error rate, which is defined by equation: $(z - x_p)/x_p$, where z is the observed position of the pedestrian, and x_p is the predicted position.

The Time to Collision (TTC) was originally defined by Hayward as "the time required for two vehicles to collide if they continue at their present speed and on the same path. It is measured continuously with time." [24] The TTC is a key element in the AEB Pedestrian system of the ADAS, and the TCC calculation is based on the pedestrian trajectory prediction. In the

ADAS camera used in this study, the default TTC is set to 7 seconds, which is a relatively safe time range for collision avoidance. In some previous studies, the average TTC was observed to be 1.1 seconds, and the maximum TTC was observed to be 4.4 seconds for their experimental vehicles [25]. For this thesis, the braking decision was made based on the 7-second TTC.

5.1 Combination of Extended Kalman Filter and Behavior Classification

In the previous sections, the EKF and the behavior classification were implemented separately. A method of combining EKF and behavior classification will be discussed in this section. For this new method, the main real-time flowchart is shown in Figure 16 below:

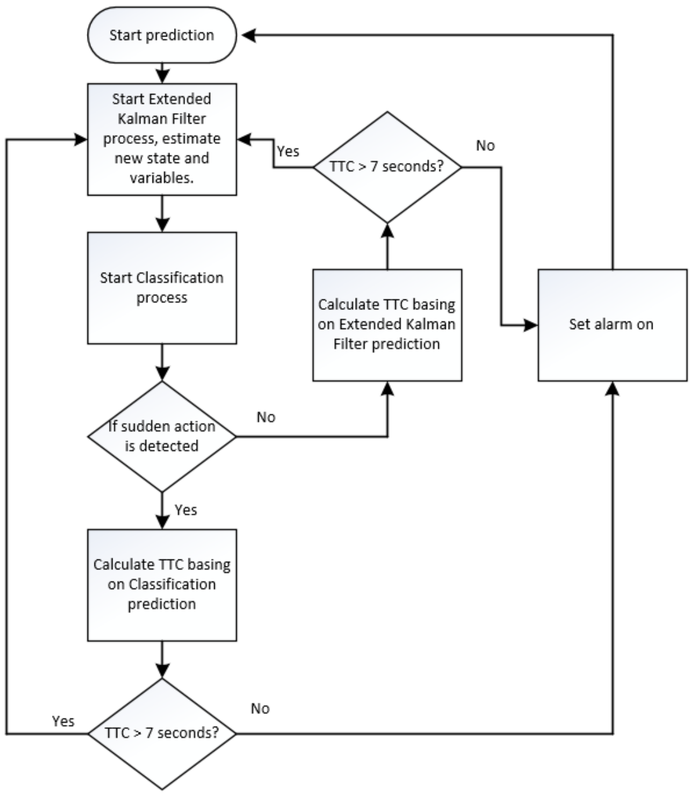

Start prediction
Start Extended Kalman Filter process, estimate new state and variables.
Yes
TTC > 7 seconds?
No
Start Classification process
Calculate TTC basing on Extended Kalman Filter prediction
Set alarm on
If sudden action is detected
No
Yes
Calculate TTC basing on Classification prediction
Yes
TTC > 7 seconds?
No

Figure 16. The State Flow of the Combination of EKF and Behavior Classification.

First, the system starts the EKF process to remove the observation noise and predicts the new state and update its parameters. Then, the system will process 30 frames of pedestrian video data by behavior classification. If an instant action is not detected, the system will calculate the TTC from the results of the EKF process. If the TTC exceeds the limit, the brake alarm will be turned on. Otherwise, the system will restart the EKF process. If an instant action is detected by the behavior classification, the system will make a new prediction based on the behavior classification, and it will then calculate the TTC from the results of the Classification process. If the TTC exceeds the limitation, the brake alarm will be turned on. Otherwise, the system will restart the EKF process.

This thesis has proposed two improvements in the state of the art of pedestrian trajectory determination in Ego Car ADAS systems. The first improvement is based on the EKF process where the “one state ahead” prediction of a pedestrian trajectory can save a 2-frame period for pedestrian collision avoidance. The EKF process will also remove the observation noise to obtain a smoother pedestrian trajectory.

The second improvement is in the classification of pedestrian behavior which can predict instant actions made by a pedestrian, where the ADAS camera cannot predict instant actions. The new method combined with the EKF and the behavior classification will provide a more advanced prediction time reduction and instant action classes to predict the pedestrian trajectory faster and more accurate.

The combination system will take these two improvements together to make the pedestrian trajectory prediction more accurate. When there is no instant action detected by the system, the EKF process will provide a noise-reduced prediction of the next pedestrian’s step, and when there is an instant action detected by this system, the behavior classification process will provide a prediction based on the classified actions.

5.1.1 Input Data for Extended Kalman Filter and Classification

The input data to the EKF process and the classification process are different: the raw position data from the ADAS camera is the input data to the EKF process. Since the EKF process is more focused on the fast computation of the next state in the real-time system, and it can also remove the observation noise by continually updating the covariance matrix p_k and the state x_k. The classification process is much more focused on measuring the changes of the pedestrian heading angle and acceleration data in a 30 frames period to detect when an instant action occurs. Hence the input data to the classification process is well-filtered average velocity data of the pedestrian. The average velocity data is used to calculate the pedestrian heading angle and acceleration.

In this overall system that combines the EKF and the classification processes, the input data need to be used in two different processes. First, the raw position data of a pedestrian from the ADAS camera will be used for the EKF process. Second, the raw velocity will be calculated from the raw position data for the classification calculation. The initial calculated velocity is noisy and unstable so that an extra noise filter will be applied to the raw velocity and an average will be taken by 30 frames. At this point, a well-filtered average velocity can be sent to the Classification process as the input data.

5.1.2 Implementation of Extended Kalman Filter and Classification in Real Time

The implementation of the EKF process is easy to do with only one pedestrian from the ADAS camera, but in real-life applications, multiple pedestrians need to be tracked at the same time. Although this study only considers examples of one single pedestrian, a Matlab program of the EKF process in this study has been implemented to support multiple pedestrian tracking.

When multiple pedestrians are tracked in real-time, all the required state data and position data need to be saved to the system buffer for one frame in order to be used in the next frame of the state calculation and prediction. Each system buffer will need to be assigned an index number.

To apply behavior classification in a real-time system, a sequence of 30 frames needs to be analyzed to perform data filtering and to obtain an average. At the same time, this 30-frame sequence must be used for the characteristics capture of an action detection. Therefore, in the classification process, a 30-frames data buffer is required for two purposes. One is to filter and average the velocity, and the other is to detect the changes in the heading angle and acceleration. In the real-time integrated system, the classification process will start after the first 30 frames, and the initial average velocity will be calculated. Before the classification process, the system will run the EKF process.

5.2 Analysis of Prediction of Pedestrian Trajectories

In this study, the EKF was implemented to provide a noise-reduced prediction for one state ahead based on the observed position of the pedestrian. This system will continuously process the whole video from the ADAS camera, and when a pedestrian is found to be first entering the video the EKF will be restarted and all of the covariance matrices and error parameters will be reset to their initial values.

For a single pedestrian being tracked by the EKF in this study, each calculation is performed in single frame increments, and the Mobileye ADAS camera saves one frame data in every two frames. The EKF will predict the next state for a pedestrian in the next frame and by this prediction, a two frames period can be saved, which takes 0.11 seconds. This is an improvement over a current pedestrian tracking system, since it offers faster calculation and a more accurate prediction by the EKF process.

Figure 17 shows the prediction error percentages of Case A1 and Case A4. Case A1 has no instant action. Hence the error rate is low and Case A4 has an instant action with a direction change, so the error rate is higher at that point.

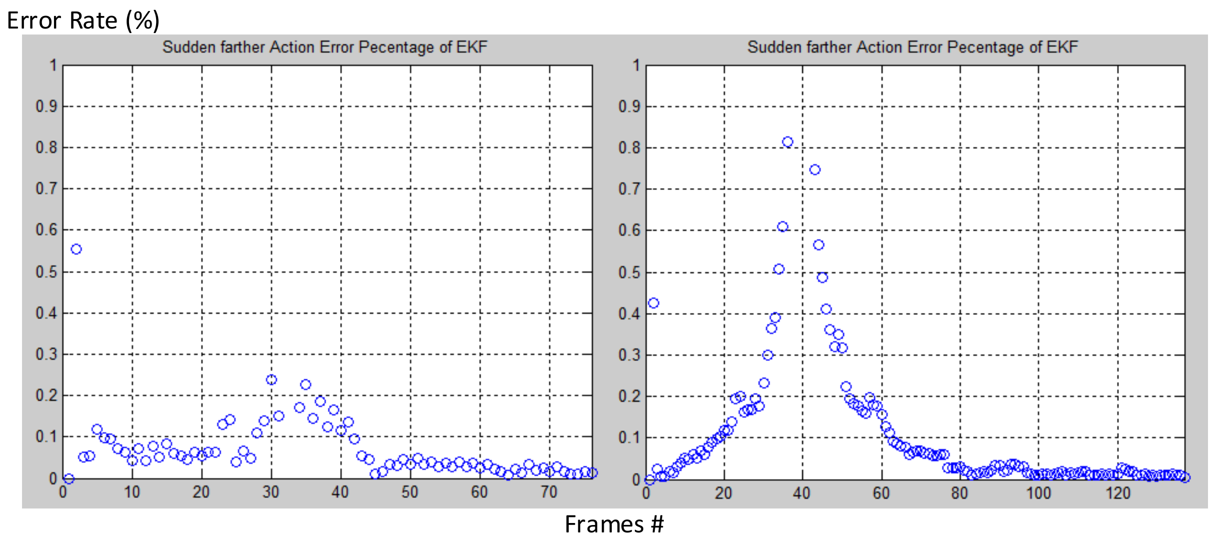


Figure 17. Comparing Error Rates of Case A1 and Case A4 in EKF.

Behavior classification will fix the high error rate problem during the pedestrian trajectory prediction by detecting instant actions and making a new prediction of the pedestrian's next step. Combining both the EKF and behavior classification will optimize the accuracy and reduce the error rate of the trajectory prediction. In Figure 18, the error rate of Case A4 is calculated based on the combination of the EKF and behavior classification. Compared to the error rate of Case A4 in Figure 17, the error range in Figure 18 is lower.

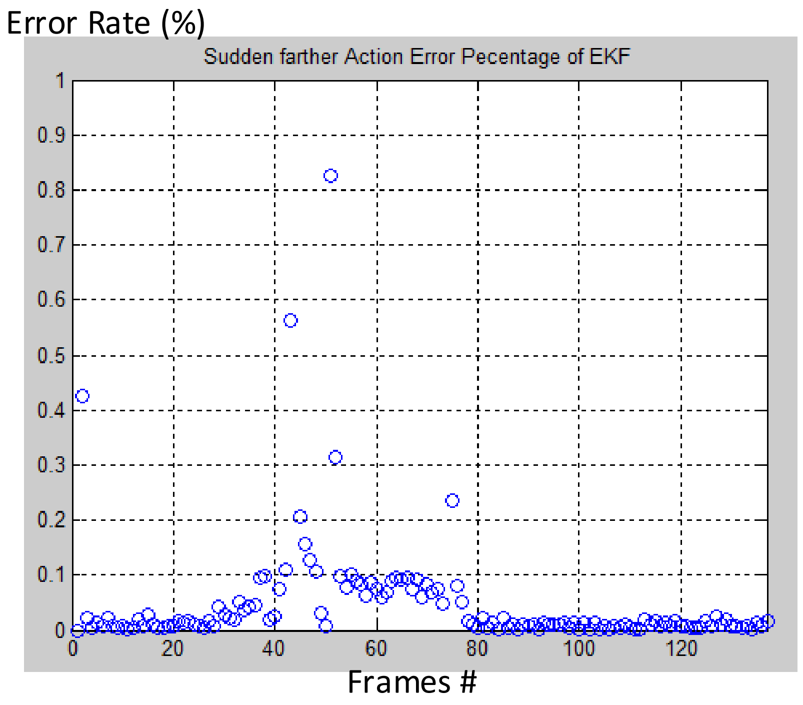


Figure 18. Error Rate of Case A4 in the Combined Method.

5.3 Time to Collision based on Pedestrian Trajectory Prediction

In the real-time system, the TTC should be continuously calculated for each frame of a video. Obtaining the TTC is a critical step of pedestrian trajectory prediction. For instant actions of the pedestrian, the safety time will be less due to the unsafe actions of the pedestrian. Once the TTC is above the limit, the system will set a brake alarm to warn the driver so that an impact on the pedestrian can be avoided. In this study, the TTC needed to be calculated for both the EKF and Classification processes.

5.3.1 Introduction to Computing Time to Collision

Based on the pedestrian movement cases in this study, calculations of the TTC are divided into three situations. The first situation is when the Ego Vehicle and the pedestrian are moving in the same direction and the TTC of Case B can be calculated for this situation. The second situation is when a pedestrian crosses a street and the Ego Vehicle moves in a perpendicular direction to the pedestrian's movement. The third situation begins with the second situation, but the pedestrian changes direction when crossing the street.

In the first situation shown in Figure 19, the Ego Vehicle and an observed pedestrian are moving in the same direction and along the same path, which is the basic situation of the TTC calculation. The TTC can be calculated by the following equation:

$$TTC = \frac{L_o}{V_f - V_l} \qquad \text{(Eq 5.1)}$$

V_f is the velocity of the Ego Vehicle. V_l is the velocity of an observed pedestrian. L_o is the distance between the Ego Vehicle and the pedestrian.

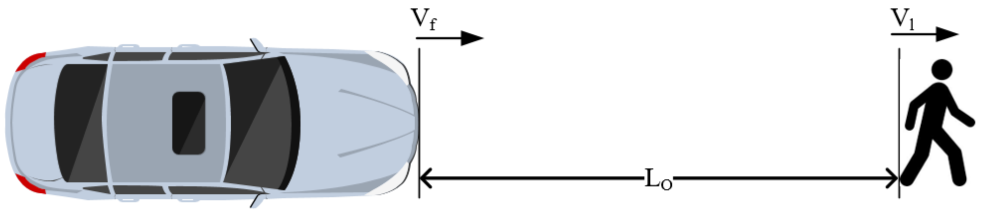


Figure 19. The Ego Vehicle and Pedestrian Move in the Same Direction.

In the second situation, the observed pedestrian is crossing a street with 0 degrees heading angle and the Ego Vehicle is moving on the street in a straight direction, shown in Figure 20. If the pedestrian's velocity is too slow or too fast, collision between the pedestrian and the Ego Vehicle won't occur. Hence time required for the pedestrian to move to the edge of the road needs to be calculated. For the near edge, time for the pedestrian to move to the edge can be calculated as $T_{o2\_1} = L_{o2\_1}/V_l$. Here L_{o2_1} is the distance between the pedestrian and the closer edge of the road. V_l is the velocity of the pedestrian arriving before the road. For the far edge, time of pedestrian reaches the edge can be calculated as $T_{o2\_2} = L_{o2\_1}/V_l$. Here L_{o2_2} is the distance between the pedestrian to the further edge of road. Time for the Ego Vehicle to reach the pedestrian latitudinal position can be calculated as $T_o = L_{o1}/V_f$. Here L_{o1} is the distance between the Ego Vehicle and the pedestrian. A collision will only occur when $T_{o2\_1} \leq T_o \leq T_{o2\_2}$. The TTC can be calculated by the following equation:

$$TTC = \frac{L_{O1}}{V_f} \tag{Eq 5.2}$$

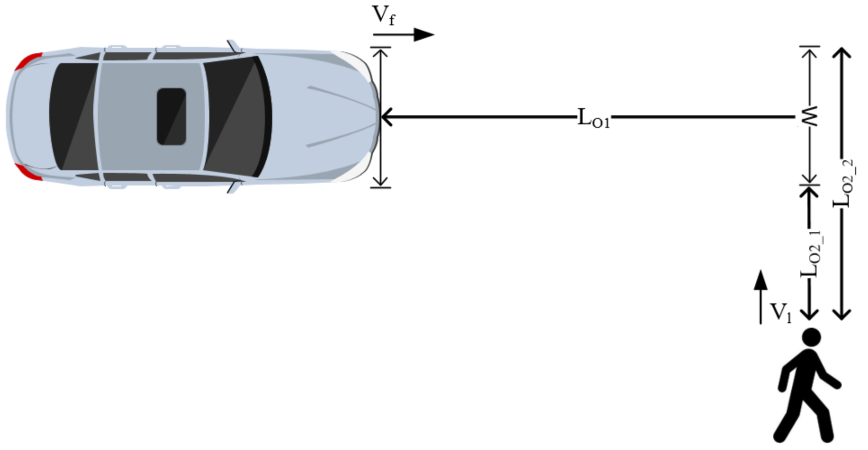


Figure 20. A Pedestrian Crosses a Street in Front of the Ego Vehicle.

In the third situation, the pedestrian will make a change in direction when crossing a street and two possible directions are shown in Figure 21. For the near edge, time for a pedestrian to reach the edge can be calculated as $T_{o2\_s1} = L_{o2\_s1} / V_{l\_1}$. Here L_{o2_s1} is the distance between the pedestrian and the closer edge of the road. Here V_{l_1} is the velocity of the pedestrian arriving at the road. For the far edge, time for a pedestrian to reach the edge can be calculated as $T_{o2\_1} = L_{o2\_1} / V_{l\_1} + L_{o2\_2} / V_{l\_2}$. Here L_{o2_1} is the distance of a pedestrian without direction change, and L_{o2_2} is the distance of pedestrian after the direction change. Here V_{l_2} is the velocity of the pedestrian on L_{o2_2} direction. Time the Ego Vehicle takes to reach the pedestrian's latitudinal position can be calculated as $T_{o1} = L_{o1} / V_f$. A collision will only happen when $T_{o2\_s1} \leq T_{o1} \leq T_{o2\_1}$.

The TTC can be calculated in two situations: A collision occurs before or after the pedestrian changes direction. When a collision occurs before the pedestrian changes direction, time when the vehicle reaches the pedestrian will be $T_o \leq L_{o2\_1}/V_l$, and the equation is:

$$TTC_1 = \frac{L_{O2}}{V_f - V_{1\_1}} \tag{Eq 5.3}$$

When a collision occurs after a pedestrian changes direction, time when the vehicle reaches the pedestrian will be $T_o \geq L_{o2_1}/V_l$, and the equation is:

$$TTC_2 = \frac{L_{O2}}{V_f - V_{1\_1} * \sin(\theta)} \tag{Eq 5.4}$$

L_{o2} is the distance between the Ego Vehicle and the pedestrian. θ is the heading angle of the pedestrian.

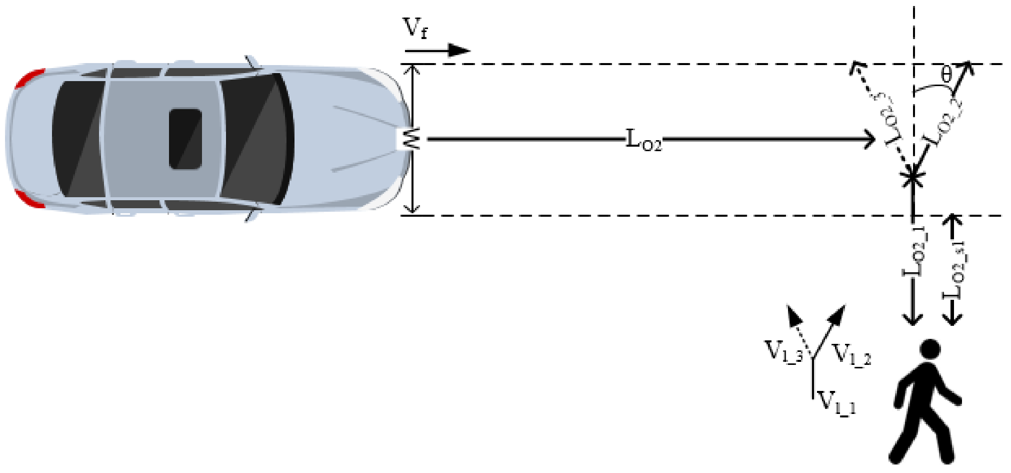


Figure 21. A Pedestrian Crosses a Street in Front of the Ego Vehicle with Direction Changes.

5.3.2 Computation of Time to Collision by the Extended Kalman Filter

When the instant action is not detected by the system, the TTC of the EKF process needs to be calculated for this situation. In the EKF process, every time the observed pedestrian data is taken into the EKF system, the new state will be predicted. Then the new position and velocity will be updated by the new state.

First, time for the Ego Vehicle to reach the pedestrian's longitudinal position, t_1, can be calculated as $t_1 = x_{long}/v_{car}$, where x_{long} is the longitudinal distance from the updated step of the EKF process between the Ego Vehicle and the pedestrian, and v_{car} is the velocity of the Ego Vehicle. Next, time when the pedestrian enters the driving path, t_2, can be calculated as $t_2 = (x_{lat} - l)/v_{lat}$, where x_{lat} is the pedestrian's latitudinal position from the updated step of the EKF process from the center line of the Ego Vehicle to the pedestrian in the latitudinal direction. Here $x_{lat} - l$ denotes the pedestrian's latitudinal position minus half of the width of the Ego Vehicle, where the width is 2 meters for a full-size car. Then time when the pedestrian leaves the driving path can be calculated as $t_3 = (x_{lat} + l)/v_{lat}$. Here $x_{lat} + l$ means the pedestrian latitudinal position plus half of the width of the Ego Vehicle.

When t_1 is greater than or equal to t_2, the pedestrian will collide with the Ego Vehicle. If $t_1 < t_2$, the Ego Vehicle will arrive at the pedestrian's latitudinal position and the pedestrian can see the vehicle moving in front of him/her and will not cross the street, which is a safe situation, needing no further consideration. If $t_1 > t_3$, the pedestrian has already crossed the street before the Ego Vehicle reaches the pedestrian's latitudinal position, which is also a safe situation.

5.3.3 Computation of Time to Collision by Behavior Classification

When the instant action is detected by the system, the TTC of the Classification process needs to be calculated. In the Classification process, each instant classification will predict a new position and calculate an average velocity for the pedestrian. Based on the new predicted position and the new average velocity, the TTC can be calculated.

First, elapse time when the Ego Vehicle reaches the pedestrian in longitudinal direction t_1 can be calculated as $t_1 = x_{long}/v_{car}$, where x_{long} is the longitudinal distance from the classification prediction process between the Ego Vehicle and the pedestrian, and v_{car} is the velocity of the Ego Vehicle. Next, elapse time when a pedestrian enters the driving path can be calculated $as\ t_2 = (x_{lat} - l)/v_{lat}$, where x_{lat} is the pedestrian's latitudinal position from the classification prediction process, measured from the centerline of the Ego Vehicle to the pedestrian in the latitudinal direction. Then, elapse time when the pedestrian leaves the driving path can be calculated as $t_2 = (x_{lat} + l)/v_{lat}$. Just as with the TTC of the EKF process, when t_1 is greater than or equal to t_2, the pedestrian will collide with the Ego Vehicle.

5.3.4 Analysis of TCC by Extended Kalman Filter and Classification

In the EKF process and Classification process, the TTC is calculated by the same method but with different positions and velocity inputs. In the EKF process, the position information used in the TTC calculation is from the EKF update step, which updates the velocity and position information using the predicted parameters. In the Classification process, the information comes from the prediction of the classified instant action that has a smoother and noise-reduced velocity and position data. In Figure 22, the EKF Process has a TTC based on the predicted next state. However, with little noise, the Classification process has a more stable trend, but it can only calculate the TTC of the past 30 frames.

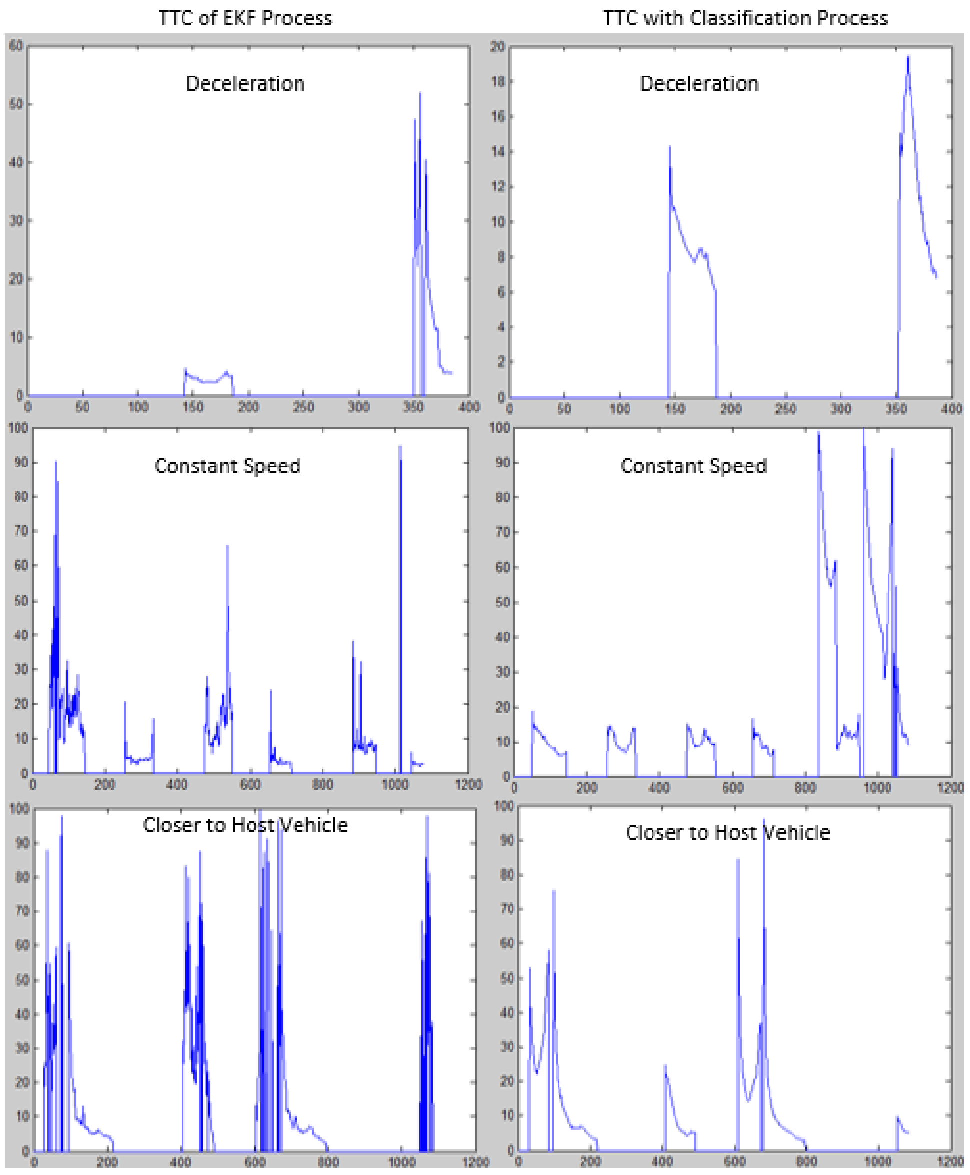


Figure 22. Comparison of TTC in the EKF Process and Classification Process.

There are four instant actions shown in Figure 23, the red markers are the original TTC from the ADAS Camera, and the blue markers are the predicted TTC from this study. The y-axis is the TTC in seconds, and the x-axis is the number of frames. These four instant actions are two for Case A6 and two for Case A4. The original TTC does not include the instant action classification and will not calculate an accurate TTC for the pedestrian. The predicted TTC is calculated based on the prediction of both the EKF and the pedestrian behavior classification, and this yields more accurate TTC for AEB Pedestrian to set the brake warning.

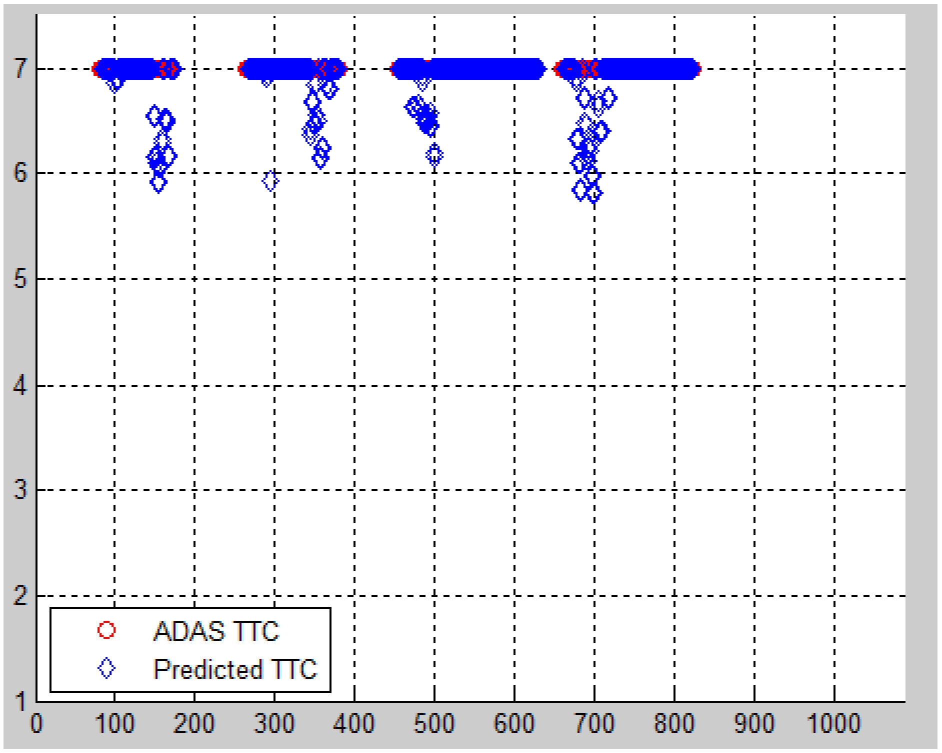


Figure 23. Comparison of TTC from the ADAS Camera and TTC Calculated from the Predicted Trajectory.

6 CONCLUSION AND FUTURE DEVELOPMENT

The observation noise of the pedestrian position data was removed by the EKF, and a new state was predicted by the previous state. The EKF is a dynamic model capable of fast calculations, of removing noise, of self-correction and of predicting a new state from a previous state, which does not require saving the data in its history database.

For behavior classification, the pedestrian’s instant actions were detected based on the pedestrian heading angle and acceleration. The heading angle and the acceleration were calculated from the new average velocity. The main idea is to calculate the average velocity from the position data of the pedestrian by reducing noise due to velocity fluctuation and by taking the average of velocities over 30 frames.

This study proposes an improved method of predicting pedestrian trajectory by combining the EKF and behavior classification to predict pedestrian trajectory from raw pedestrian position data from the ADAS camera. The EKF was used to smooth the original position data to remove the observation noise and to predict a new position. The EKF provides a fast calculation of the next state prediction and saves 0.11 seconds over two frames for the TTC calculation. The resulting heading angle and acceleration were then calculated and used as inputs to the behavior classification process in order to identify each behavior class based on the new average velocity estimated from the previous filtering step. Behavior classification provides a prediction with more stability and less noise.

Future work might entail using more training data and attempting to classify a larger variety of pedestrian actions into more detailed classes. Weather conditions and night time conditions could be added as variables for the prediction. The location of the Ego Vehicle could be provided by GPS and this information might be added to the prediction. Regarding the hardware of the ADAS camera, a newer generation of the Mobileye camera will be released in 2020 [26], which will have a face detection function. With this function, behavior classification will be more accurate with respect to heading angle detection.

LIST OF REFERENCES

[1] "ADAS technology," 2018. [Online]. Available: https://www.mobileye.com/our-technology/adas/.

[2] "Euro NCAP | Timeline." 2017. [Online]. Available: https://www.euroncap.com/en/about-euro-ncap/timeline/.

[3] "Pedestrian Safety | NHTSA." 2017 [Online]. Available: https://www.nhtsa.gov/road-safety/pedestrian-safety.

[4] "U.S. DOT and IIHS announce historic commitment of 20 automakers| NHTSA." 2017 [Online]. Available: http://www.iihs.org/iihs/news/desktopnews/u-s-dot-and-iihs-announce-historic-commitment-of-20-automakers-to-make-automatic-emergency-braking-standard-on-new-vehicles.

[5] Zhijun Chen, Chaozhong Wu, Nengchao Lyu, Gang Liu, and Yi He, "Pedestrian-vehicular collision avoidance based on vision system," 7th IEEE International Conference Intelligent Transportation System, October 3-6, 2004, Washington, WA. 2004.

[6] "ABOUT MOBILEYE N.V.," 2018. [Online]. Available: https://www.mobileye.com/en-us/about-mobileye/.

[7] Kirsten Korosec, "Mobileye, Why Intel Bought," 2017. [Online]. Available: http://fortune.com/2017/03/13/why-intel-bought-mobileye/.

[8] "Intel: 2M Nissan, BMW, VW Cars Will Use Mobileye," 2018. [Online]. Available: https://www.investopedia.com/news/intel-2m-nissan-bmw-vw-cars-will-use-mobileye/

[9] Pablo Negri and Damian Garayalde, "Concatenating multiple trajectories using Kalman filter for pedestrian tracking," 2014 IEEE Biennial Congress of Argentina, June 11-13, 2014, Bariloche, Argentina, 2014.

[10] Fuliang Li, Ronghui Zhang, and Feng You, "Fast pedestrian detection and dynamic

tracking for intelligent vehicles within V2V cooperative environment," *IET Image Processing*, vol. 11, no. 10, pp. 833-840, 2017.

[11] Kang Chen and Wangchen Ge, "Pedestrian tracking algorithm based on Kalman filter and partial mean-shift tracking," 2nd International Conference System Informatics, November 15-17, 2014, Shanghai, China, 2014.

[12] Yanwu Xu, Xiaobin Cao, and Tingxia. Li, "Extended Kalman filter based pedestrian localization for collision avoidance," 2009 International Conference Mechatronics Automation, August 9-12, 2009, Changchun, China, 2009.

[13] David Ellis, Eric Sommerlade, and Ian Reid, "Modelling pedestrian trajectory patterns with Gaussian processes," IEEE 12th International Conference Computer Vision, September 27 - October 4, 2009, Kyoto, Japan, 2009.

[14] Kihwan Kim, Dongryeol Lee, and Iifan Essa, "Gaussian process regression flow for analysis of motion trajectories," 2011 IEEE International Conference Computer Vision, November 6-13, Barcelona, Spain, 2011.

[15] Yufan Chen, Miao Liu, Shih-Yuan Liu, Justin Miller, and Jonathan P. How, "Predictive Modeling of Pedestrian Motion Patterns with Bayesian Nonparametrics," 2016 AIAA Guidance Navigation Control Conference, January 4-8, 2016, San Diego, California, 2016.

[16] Neil Thacker, Tony Lacey, "Tutorial: The kalman filter," *Imaging Science and Biomedical Engineering Division - Tina Memo* no. 100*6-002.* pp. 133–140, 1998.

[17] Simon. D. Levy, "The Extended Kalman Filter: An Interactive Tutorial." 2016 [Online], Available at: https://home.wlu.edu/~levys/kalman_tutorial/

[18] Zhuo. Chen, Daniel Chi Kit Ngai, Nelson Hon Ching Yung "Pedestrian Behavior Prediction based on Motion Patterns for Vehicle-to-Pedestrian Collision Avoidance," 11th Intelligent Transportation System International Conference IEEE, October 12-15, 2008, Beijing, China, 2008.

[19] Ying Ni, Yingying Cao, and Keping Li, "Pedestrians' Safety Perception at Signalized Intersections in Shanghai," *Transportation Research Procedia*, vol. 25, no. June, pp. 1960–1968, 2017.

[20] Steven Bennett, Adam Felton, and Rahmi Akçelik, “Pedestrian movement characteristics at signalized intersections,” 23rd Conference Australia Institutes Transportation Research, December 10-12, 2001, Melbourne, Australia, 2001.

[21] Mohammed M. Hamed, “Analysis of pedestrians’ behavior at pedestrian crossings,” *Safety Science*, vol. 38, no. 1, pp. 63–82, 2001.

[22] Daniel V. McGehee, Elizabeth N. Mazzae, and GH Scott Baldwin, “Driver Reaction Time in Crash Avoidance Research: Validation of a Driving Simulator Study on a Test Track,” *Processing Human Factors Ergonomics Society Annual Meeting*, vol. 44, no. 20, pp. 3-320-3–323, 2000.

[23] Ross Grote Stirling, “Development of a Pedestrian Navigation System Using Shoe Mounted Sensors,” University of Alberta Master Thesis of 2004 Canada.

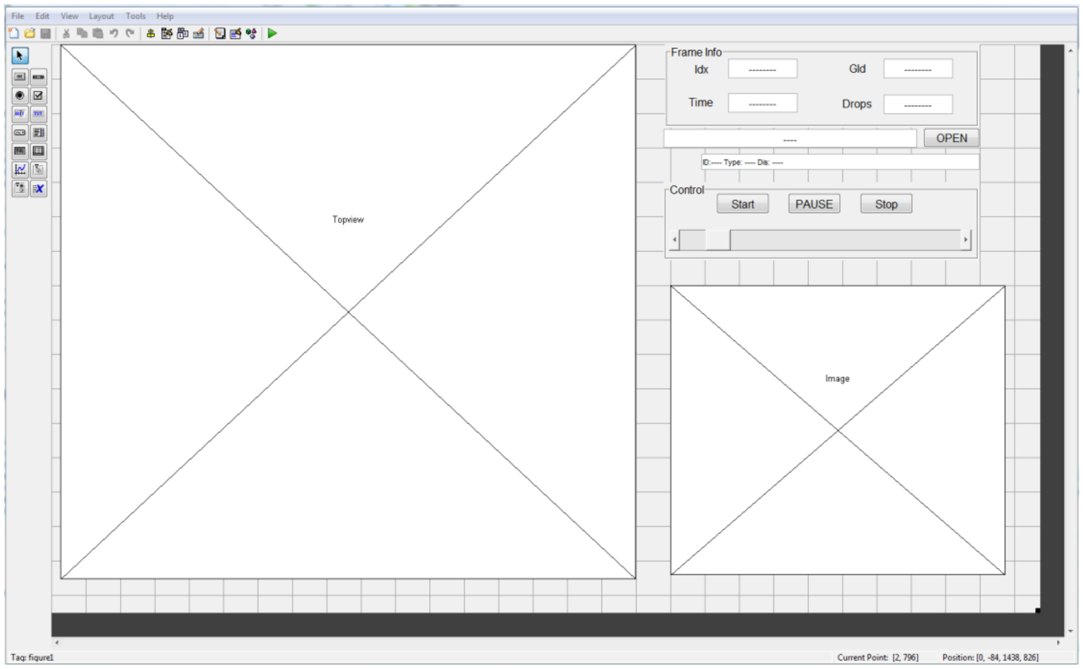
[24] John C. Hayward, “Near-Miss Determination Through,” *Highway Research Board*, vol. 384, no. 385-1972, pp. 24–35, 1971.

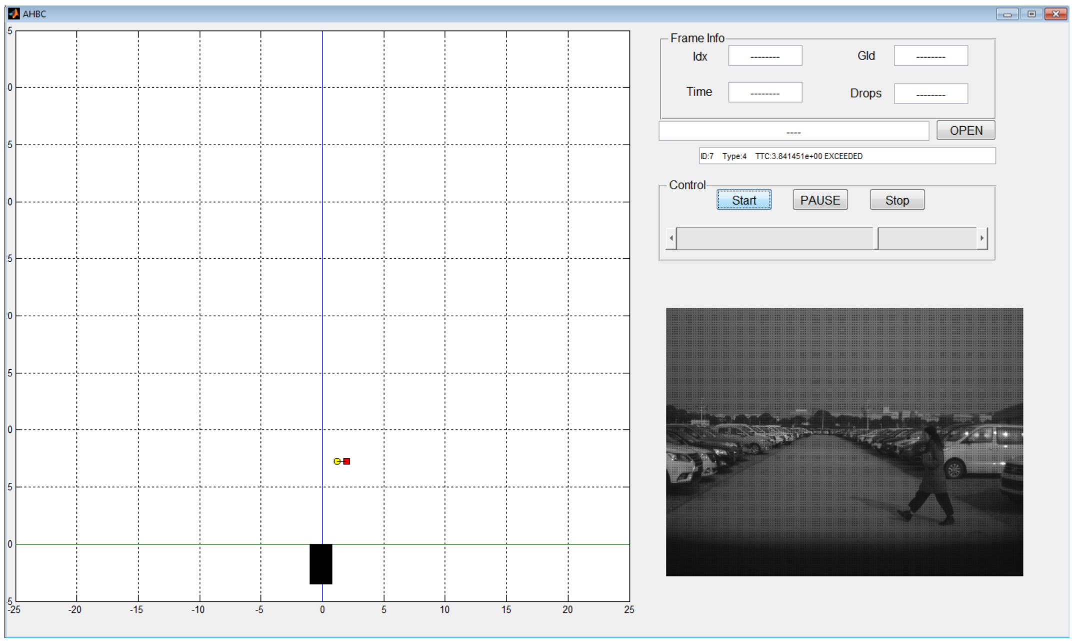
[25] Kristofer D. Kusano, Hamptom Gabler, “Method for Estimating Time to Collision at Braking in Real-World, Lead Vehicle Stopped Rear-End Crashes for Use in Pre-Crash System Design,” *SAE International Journal Passenger Cars - Machines Systems*, vol. 4, no. 01-0576, pp. 435-443, 2011.

[26] “The Evolution of EyeQ,” 2018. [Online]. Available: https://www.mobileye.com/our-technology/evolution-eyeq-chip/.

APPENDICES

Appendix A. ADAS Camera User Interface Matlab GUI Design





ID:7 Type:4 TTC:3.841451e+00 EXCEEDED

Appendix B. Matlab Program of Callback functions with GUI

```
function varargout = AHBC(varargin)
% AHBC MATLAB code for AHBC.fig
%      AHBC, by itself, creates a new AHBC or raises the existing
%      singleton*.
%
%      H = AHBC returns the handle to a new AHBC or the handle to
%      the existing singleton*.
%
%      AHBC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in AHBC.M with the given input arguments.
%
%      AHBC('Property','Value',...) creates a new AHBC or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before AHBC_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to AHBC_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help AHBC

% Last Modified by GUIDE v2.5 08-Feb-2016 16:01:00

% Begin initialization code - DO NOT EDIT

  gui_Singleton = 1;

  gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AHBC_OpeningFcn, ...
                   'gui_OutputFcn',  @AHBC_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
  if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
  end

  global v;
  global p_frame_buf;
  global p_frame_data;

  % Init value for image buffer
  v = zeros(1280,960,'uint8');
  % Pointer to image buffer
  p_frame_buf = libpointer('uint8Ptr',v);
  % For sync data
  struct frame_data ('idx',0,'gid',0,'time',0,'drop',0);
  loadlibrary('vidsframe',@mHeader);
  p_frame_data = libpointer('FRAME_DATAPtr');



  if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
  else
    gui_mainfcn(gui_State, varargin{:});
  end
% End initialization code - DO NOT EDIT

% --- Executes just before AHBC is made visible.
function AHBC_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to AHBC (see VARARGIN)

% Choose default command line output for AHBC
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = AHBC_OutputFcn(hObject, eventdata, handles)
```

```
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in OpenButton. open button
function OpenButton_Callback(hObject, eventdata, handles)
% hObject    handle to OpenButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



global filename;
global pathname;
global filter;
global currentFrame;

global avi_path;
global mudp_path;
global mudp_data;
global nFrame;
global startStopPauseFlag;

currentFrame =1;
startStopPauseFlag = 0;
[filename pathname filter] = uigetfile('*.avi','Select AVI File');
avi_path = [pathname,filename]
calllib('vidsframe', 'open', avi_path);
ind = findstr(avi_path, '.');
mudp_path = [avi_path(1:ind),'mudp']

mudp_data = read_mudp_data(mudp_path, [5]);



nFrame = calllib('vidsframe','get_nFrame');
set(handles.path,'String',avi_path);
set(handles.slider1,'Sliderstep',[1/nFrame,1/nFrame]);
set(handles.slider1,'Max',nFrame);
```

```
% --- Executes on button press in Start.
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

  global p_frame_buf;
  global frame_data;
  global p_frame_data;
  global mudp_data;
  global nFrame;
  global currentFrame;
  global startStopPauseFlag;
  global lastFlag;
  global Type;

  % Buttom state

  lastFlag = startStopPauseFlag;
  startStopPauseFlag =0;% start 0; stop 1; pause 2;

  if(lastFlag ~= 2)
    currentFrame =1;
    lastFlag=0;
  end

  if(lastFlag == 2)
   currentFrame
  end
  % Kalman filter initialization
  pre_ID = 0;
  MR = 5; % the inital state value
  MC = 5;
  r1=0.2;
  r2=0.02;
  R=[[r1,0]',[0,r2]']; % Measurement error covariance
  H=[[1,0]',[0,1]',[0,0]',[0,0]'];
  Q=0.01*eye(4); % State error covariance
  dt=1; % Step
  A=[[1,0,0,0]',[0,1,0,0]',[dt,0,1,0]',[0,dt,0,1]']; % Jacobian matrix
  Bu = [0,0,0,0]'; % Control Signal is none
```

```
xyv=zeros(100,4);
count = 1;

est_lat_vel_ped=zeros(nFrame,1);
est_long_vel_ped=zeros(nFrame,1);
lat_pos_ped = zeros(nFrame,1);
long_pos_ped = zeros(nFrame,1);
lat_pos_ped_pre = zeros(nFrame,1);
long_pos_ped_pre = zeros(nFrame,1);
lat_ped_vel = zeros(nFrame,1);
ctrf = 1;
nextStepClassifiy =0;

set(handles.slider1,'Sliderstep',[1/nFrame,1/nFrame]);
for i = currentFrame:nFrame
  set(handles.slider1,'Value',currentFrame);
  if(startStopPauseFlag ~= 0)
    break;
  end
  ctrf = ctrf + 1;

  axes(handles.Topview);
  if(i == nFrame)
    i = nFrame-1;
  end
  plainviewplayer(i,handles);



  % Pedestrian selection and movement
  if(mudp_data.vision_obstacles_info.visObs.obstacle_class(i,1) ~= 4)
    buf = sprintf('ID: %s Type: %s TTC: %s ', '----', '----', '----');
    set(handles.spot1,'String', buf);
  else
    lat_pos_ped(i) = mudp_data.vision_obstacles_info.visObs.lat_pos(i,1);
    long_pos_ped(i) = mudp_data.vision_obstacles_info.visObs.long_pos(i,1);
    lat_ped_vel(i) = mudp_data.vision_obstacles_info.visObs.lat_vel(i,1);
    % long_ped_vel(i) = mudp_data.vision_obstacles_info.visObs.long_vel(i,1);
    if(i > 1)
      lat_pos_ped_pre(i) = mudp_data.vision_obstacles_info.visObs.lat_pos(i-1,1);
      long_pos_ped_pre(i) = mudp_data.vision_obstacles_info.visObs.long_pos(i-1,1);
    end
```

```
      ID =  mudp_data.vision_obstacles_info.visObs.id(i,1);
      Type =  mudp_data.vision_obstacles_info.visObs.obstacle_class(i,1);
      ad_ttc(i) = mudp_data.vision_obstacles_info.visObs.ttc_const_vel(i,1);
      car_vel = mudp_data.vision_vehicle_info.vehicleVelocity(i);

      %EKF
      x_po = lat_pos_ped(i);
      y_po = long_pos_ped(i);

      if(ID ~= pre_ID)
        kfinit=0;
        m = 1;
        count = count +1;
      else
        m =m +1;
      end

      x(m) = x_po; % Obervation data
      y(m) = y_po; % zk = [x[i] y[i]]
      hold on;
      % rectangle('Position', [x(m), y(m), 15, 15],'EdgeColor', 'r' );
      processFlag=0;
      % Kalman Predict if kfinit is not 0, otherwise initialize parameters
      if kfinit==0
        xp = [MC/2,MR/2,0,0]';
        P = 1*eye(4); % Process error covariance
        PP= P;
      else
        xp=A*xyv(m-1,:)' + Bu;
        PP = A*P*A' + Q;
      end

      kfinit=1;
      % Update
      K = PP*H'*inv(H*PP*H'+R);
      xyv(m,:) = (xp + K*([x(m),y(m)]' - H*xp))';
      P = (eye(4)-K*H)*PP;
      xyv(m,3) = lat_ped_vel(i);
      rectangle('Position',[xyv(m,1),xyv(m,2),0.5,0.5],'Curvature',1,'facecolor','yellow');
      plot(handles.Topview, [xyv(m,1)+0.25 lat_pos_ped(i)+0.25],[xyv(m,2)+0.25
long_pos_ped(i)+0.25], 'color', 'black');
      hold off;
      pre_ID =ID;
      % Classification of theta
      er_vel = 3; %
      dd=4;
      dt = 0.44;
      if(ctrf > 1 && i > 2 && i< 1086)
        if( abs(lat_pos_ped(i) - lat_pos_ped(i-1))/0.055 < er_vel )
          est_lat_vel_ped(ctrf) = (lat_pos_ped(i) - lat_pos_ped_pre(i))/0.055;
        else
          est_lat_vel_ped(ctrf) = est_lat_vel_ped(ctrf-1);
        end

        if(abs(long_pos_ped(i) - long_pos_ped(i-1))/0.055 < 1)
          est_long_vel_ped(ctrf) = (long_pos_ped(i) - long_pos_ped_pre(i))/0.055;
        else
          est_long_vel_ped(ctrf) = est_long_vel_ped(ctrf-1);
        end

        if(ctrf > 30)
          mean_est_long_vel_ped(ctrf) = mean(est_long_vel_ped(ctrf-30:ctrf));
          mean_est_lat_vel_ped(ctrf) = mean(est_lat_vel_ped(ctrf-30:ctrf));
          mean_est_theta_ped(ctrf) =
rad2deg(atan(mean_est_long_vel_ped(ctrf)/mean_est_lat_vel_ped(ctrf)));
          est_lat_acc_ped(ctrf) = (mean_est_lat_vel_ped(ctrf)-mean_est_lat_vel_ped(ctrf-dd))/dt;

          % Check sudden actions based on 30 frames
          g_arr = zeros(1,30);
          a_arr = zeros(1,30);
          d_arr = zeros(1,30);
          gCheckPre30 = zeros(1,30);
          aCheckPre30 = zeros(1,30);
          dCheckPre30 = zeros(1,30);

          for h = 1:30
            g_arr(h) = mean_est_theta_ped(ctrf+1-h)-mean_est_theta_ped(ctrf-h);
            a_arr(h) = est_lat_acc_ped(ctrf+1-h) - est_lat_acc_ped(ctrf-h);
            d_arr(h) = -a_arr(h);

            if(h>1 && g_arr(h) -g_arr(h-1) < 20)
              gCheckPre30(h) = 1;
            end
            if(h>1 && a_arr(h) -a_arr(h-1) < 0.15 && a_arr(h) ~= 0)
              aCheckPre30(h) = 1;
            end

            if(h>1 && d_arr(h) - d_arr(h-1) > -0.15 && a_arr(h) ~= 0)
              dCheckPre30(h) = 1;
            end
          end

          cp = sum(gCheckPre30);
          ap = sum(aCheckPre30);
          dp = sum(dCheckPre30);
          if(cp == 29 && mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-1) >150)% Check Sudden
back
            nextStepClassifiy = 1;
          elseif(cp == 29 && mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-29)> 30 &&
mean_est_theta_ped(ctrf)>0 && mean_est_theta_ped(ctrf-29)<0)% sudden far
            nextStepClassifiy = 2;
          elseif(cp == 29 && abs(mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-29))> 30 &&
mean_est_theta_ped(ctrf)<0 && mean_est_theta_ped(ctrf-29)>0)% sudden close
            nextStepClassifiy = 3;
          end
          if(ap == 29)% Check acceleration
            nextStepClassifiy = 4;
          elseif(dp == 29)% Check dcceleration
            nextStepClassifiy = 5;
          end

          if(nextStepClassifiy ==1)% Check Sudden back
            mean_est_lat_vel_ped(ctrf+1)=mean_est_lat_vel_ped(ctrf-1);
          elseif (nextStepClassifiy ==2)% sudden far
            mean_est_long_vel_ped(ctrf) = 1.02*mean_est_long_vel_ped(ctrf);
          elseif (nextStepClassifiy ==3)% sudden close
            mean_est_long_vel_ped(ctrf) = 0.98*mean_est_long_vel_ped(ctrf);
          elseif (nextStepClassifiy ==4)
            mean_est_lat_vel_ped(ctrf)=1.02*mean_est_lat_vel_ped(ctrf);
          elseif (nextStepClassifiy ==5)
            mean_est_lat_vel_ped(ctrf)=0.98*mean_est_lat_vel_ped(ctrf);
          else
            % Deflaut: no changes
          end

          if(mean_est_lat_vel_ped(ctrf) ~= 0)
            if(processFlag == 0) %ekf pro
              t1 = abs(xyv(m,2) / car_vel);
              t2 = abs((abs(xyv(m,1))-1)/xyv(m,3));
              t3 = abs((abs(xyv(m,1))+1)/xyv(m,3));
              ttc(ctrf) = abs(xyv(m,2) / (car_vel-xyv(m,3)));
            else
              t1 = abs(xyv(m,2) / car_vel);
              t2 = abs((abs(xyv(m,1))-1)/mean_est_lat_vel_ped(ctrf));
              t3 = abs((abs(xyv(m,1))+1)/mean_est_lat_vel_ped(ctrf));
              ttc(ctrf) = abs(xyv(m,2) / (car_vel-mean_est_long_vel_ped(ctrf)));
            end

            if(abs(ttc(ctrf)) > 8)
              ttc(ctrf) = 7;
            end


            if(t1 >= t2)
              if(ttc(ctrf)>10)
                buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d', ID, Type, ttc(ctrf));
              else
                 buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d EXCEEDED', ID, Type, ttc(ctrf));
              end
              set(handles.spot1,'String', buf);
            end
          end
        end
    end
    end



    % Call function to play video
    calllib('vidsframe','seek',i);
    calllib('vidsframe','get_frame',p_frame_buf);
    p_frame_data =  calllib('vidsframe', 'get_frame_info');
    axes(handles.Image);
    imshow(get(p_frame_buf,'Value')');

  % In Camera View
  % Mark the pedetrian position
%   if( top>0 && bottom>0 && right-left>0 && top-bottom>0)
%     %axes(handles.Image);
%     rectangle('Position', [left, (960-top), (right-left), (top-bottom)],'EdgeColor', 'g' );
%     text(left, (960-top),txt_str,'color',[1,1,1]);
%   end

%   buf = sprintf('ID:%-4d Type:%-4d', ID, Type);
%   set(handles.spot1,'String', buf);
%   x_po = (left+right)/2;
%   y_po = 960-(top+bottom)/2;
%   rectangle('Position', [x_po, y_po, 15, 15],'EdgeColor', 'r' );
   set(handles.slider1,'Value',currentFrame);
   currentFrame = currentFrame + 1;
  end
  nsize = size(mean_est_lat_vel_ped);
  if(startStopPauseFlag == 0)
    figure;
    subplot(4,1,1);
    plot(mean_est_lat_vel_ped(30:nsize(2)-1)); %nFrame
    axis([0 nsize(2) -6 6]);
    subplot(4,1,2);
    plot(mean_est_long_vel_ped(30:nsize(2)-1));
    axis([0 nsize(2) -6 6]);
    subplot(4,1,3);
    plot(mean_est_theta_ped(30:nsize(2)-1),'o');
    title('theta deg mean');
    axis([0 nsize(2) -100 100]);
    subplot(4,1,4);
    plot(est_lat_acc_ped(30:nsize(2)-1));
    title('Accel lat');
    axis([0 nsize(2) -3 3]);
    grid;
    figure;
    plot(ttc,'o');
    figure;
    plot(ad_ttc,'o');
  end
  if(startStopPauseFlag == 1 || startStopPauseFlag == 2)
    % Call function to play video
    calllib('vidsframe','seek',i);
    calllib('vidsframe','get_frame',p_frame_buf);
    p_frame_data =  calllib('vidsframe', 'get_frame_info');
    axes(handles.Image);
    imshow(get(p_frame_buf,'Value')');

    ID =  mudp_data.vision_obstacles_info.visObs.id(currentFrame,1);
    Type =  mudp_data.vision_obstacles_info.visObs.obstacle_class(currentFrame,1);
    if(mudp_data.vision_obstacles_info.visObs.obstacle_class(currentFrame,1) ~= 4)
      buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d', ID, Type, ttc(ctrf));
      set(handles.spot1,'String', buf);
    else
      axes(handles.Topview);
      plainviewplayer(currentFrame,handles);
    end
  end

unloadlibrary('vidsframe');

function plainviewplayer(idx,exdls)

global mudp_data;
global pos_veh;
global Type;



    vox=[0 0];
    voy=[-5 45];
    hox=[-25 25];
    hoy=[0 0];
    plot(vox,voy,hox,hoy);
    axis([-25 25 -5 45]);
    if(Type == 4)
        lat_ped_pos = mudp_data.vision_obstacles_info.visObs.lat_pos(idx,1);
        long_ped_pos = mudp_data.vision_obstacles_info.visObs.long_pos(idx,1);

        % Vehicle
        pos_veh = 0;%pos_veh + lat_vel_veh*0.055;
        rectangle('Position',[-1,-3.5+pos_veh,1.8,3.5],'facecolor','black');

        % Ped
        rectangle('Position',[lat_ped_pos,long_ped_pos+pos_veh,0.5,0.5],'facecolor','red');
        axis([-25 25 -5 45]);
        grid;
    end

% --- Executes on button press in Pause.
function Pause_Callback(hObject, eventdata, handles)
% hObject    handle to Pause (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global startStopPauseFlag;
global lastFlag;
lastFlag = startStopPauseFlag;
startStopPauseFlag =2;

% --- Executes on button press in Stop.
function Stop_Callback(hObject, eventdata, handles)
% hObject    handle to Stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global currentFrame;
global startStopPauseFlag;
global lastFlag;

global p_frame_buf;
global frame_data;
global p_frame_data;

lastFlag = startStopPauseFlag;
startStopPauseFlag =1;
currentFrame =1;

   videoplayer(currentFrame, handles);
   updateInfo(currentFrame, handles);

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider




global nFrame;
global currentFrame;
global startStopPauseFlag;
global lastFlag;
```

```
lastFlag = startStopPauseFlag;
startStopPauseFlag =2;




click = get(hObject, 'Value');
currentFrame = int16(click);
    videoplayer(currentFrame, handles);

updateInfo(currentFrame, handles);

function drawLightArea(left,right, top, bottom, objExist)
    c_x = 684;
c_y = 340;
edge_h = 500.0;
k2=-0.269;
b2=804;
k1=0.234;
b1=460;
    %imshow(img);
    hold on;

    y1 = 960-(left*k1+b1);
    y2 = 960-(right*k2+b2);

if(objExist == 1)
   if(bottom >c_y)
       if(left <= c_x && right >= c_x)% in the middle
            if(y1<bottom && y2<bottom)
                c = [1 1 left left right right 1279 1279];
                r = [959 edge_h y1 bottom bottom y2 edge_h 959];
                plot(c,r,'y');
            elseif(y1<bottom && y2>bottom)
                x2 = (960-bottom-b2)/k2;
                c = [1 1 left left x2 1279 1279];
                r = [959 edge_h y1 bottom bottom edge_h 959];
                plot(c,r,'y');
           elseif(y1>bottom && y2<bottom)
                x1 = (960-bottom-b1)/k1
                c = [1 1 x1 right right 1279 1279];
                r = [959 edge_h bottom bottom y2 edge_h 959];
                plot(c,r,'y');
            elseif(y1>=bottom && y2>=bottom)
                 x1 = (960-bottom-b1)/k1;
                 x2 = (960-bottom-b2)/k2;
                 c = [1 1 x1 x2 1279 1279];
                 r = [959 edge_h bottom bottom edge_h 959];
                 plot(c,r,'y');
            end
       else
          if(left>c_x)%on right side
                y1 = 960-(left*k2+b2);
                if(y1<=bottom && y2<=bottom)
                    c = [1 1 c_x left left right right 1279 1279];
                    r = [959 edge_h c_y y1 bottom bottom y2 edge_h 959];
                    plot(c,r,'y');
                 elseif(y1<bottom && y2>bottom)
                    x2 = ((960-bottom)-b2)/k2;
                    c = [1 1 c_x left left x2 1279 1279];
                    r = [959 edge_h c_y y1 bottom bottom edge_h 959];
                    plot(c,r,'y');
                else
                    c = [1 1 c_x 1279 1279];
                    r = [959 edge_h c_y edge_h 959];
                    plot(c,r,'y');
                end
         elseif(right<c_x)
                y2 = 960-(right*k1+b1);
                if(y1<=bottom && y2<=bottom)
                    c = [1 1 left left right right c_x 1279 1279];
                    r = [959 edge_h y1 bottom bottom y2 c_y edge_h 959];
                    plot(c,r,'y');
                elseif(y1>bottom && y2<bottom)
                    x1 = (960-bottom-b1)/k1;
                    c = [1 1 x1  right right c_x 1279 1279];
                    r = [959 edge_h bottom  bottom y2 c_y edge_h 959];
                    plot(c,r,'y');
                else
                    c = [1 1 c_x 1279 1279];
                    r = [959 edge_h c_y edge_h 959];
                    plot(c,r,'y');
                end
         end
       end
```

```
    else
             c = [1 1 c_x 1279 1279];
             r = [959 edge_h c_y edge_h 959];
             plot(c,r,'y');
    end

else
     c = [1 1 c_x 1279 1279];
             r = [959 edge_h c_y edge_h 959];
             plot(c,r,'y');
end
    hold off;
```

Appendix C. EKF Process and Classification Process Program

```
function Start_Callback(hObject, eventdata, handles)
% hObject    handle to Start (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

  global p_frame_buf;
  global frame_data;
  global p_frame_data;
  global mudp_data;
  global nFrame;
  global currentFrame;
  global startStopPauseFlag;
  global lastFlag;
  global Type;

  % Buttom state

  lastFlag = startStopPauseFlag;
  startStopPauseFlag =0;% start 0; stop 1; pause 2;

  if(lastFlag ~= 2)
    currentFrame =1;
    lastFlag=0;
  end

  if(lastFlag == 2)
   currentFrame
  end
  % Kalman filter initialization
  pre_ID = 0;
  MR = 5; % the inital state value
  MC = 5;
  r1=0.2;
  r2=0.02;
  R=[[r1,0]',[0,r2]']; % Measurement error covariance
```

```
H=[[1,0]',[0,1]',[0,0]',[0,0]'];
Q=0.01*eye(4); % State error covariance
dt=1; % Step
A=[[1,0,0,0]',[0,1,0,0]',[dt,0,1,0]',[0,dt,0,1]']; % Jacobian matrix
Bu = [0,0,0,0]'; % Control Signal is none
xyv=zeros(100,4);
count = 1;

est_lat_vel_ped=zeros(nFrame,1);
est_long_vel_ped=zeros(nFrame,1);
lat_pos_ped = zeros(nFrame,1);
long_pos_ped = zeros(nFrame,1);
lat_pos_ped_pre = zeros(nFrame,1);
long_pos_ped_pre = zeros(nFrame,1);
lat_ped_vel = zeros(nFrame,1);
ctrf = 1;
nextStepClassifiy =0;

set(handles.slider1,'Sliderstep',[1/nFrame,1/nFrame]);
for i = currentFrame:nFrame
  set(handles.slider1,'Value',currentFrame);
  if(startStopPauseFlag ~= 0)
    break;
  end
  ctrf = ctrf + 1;

  axes(handles.Topview);
  if(i == nFrame)
    i = nFrame-1;
  end
  plainviewplayer(i,handles);



  % Pedestrian selection and movement
  if(mudp_data.vision_obstacles_info.visObs.obstacle_class(i,1) ~= 4)
    buf = sprintf('ID: %s Type: %s TTC: %s ', '----', '----', '----');
    set(handles.spot1,'String', buf);
  else
    lat_pos_ped(i) = mudp_data.vision_obstacles_info.visObs.lat_pos(i,1);
    long_pos_ped(i) = mudp_data.vision_obstacles_info.visObs.long_pos(i,1);
    lat_ped_vel(i) = mudp_data.vision_obstacles_info.visObs.lat_vel(i,1);
    % long_ped_vel(i) = mudp_data.vision_obstacles_info.visObs.long_vel(i,1);
```

```
if(i > 1)
  lat_pos_ped_pre(i) = mudp_data.vision_obstacles_info.visObs.lat_pos(i-1,1);
  long_pos_ped_pre(i) = mudp_data.vision_obstacles_info.visObs.long_pos(i-1,1);
end

ID =  mudp_data.vision_obstacles_info.visObs.id(i,1);
Type =  mudp_data.vision_obstacles_info.visObs.obstacle_class(i,1);
ad_ttc(i) = mudp_data.vision_obstacles_info.visObs.ttc_const_vel(i,1);
car_vel = mudp_data.vision_vehicle_info.vehicleVelocity(i);

%EKF
x_po = lat_pos_ped(i);
y_po = long_pos_ped(i);

if(ID ~= pre_ID)
  kfinit=0;
  m = 1;
  count = count +1;
else
  m =m +1;
end

x(m) = x_po; % Obervation data
y(m) = y_po; % zk = [x[i] y[i]]
hold on;
% rectangle('Position', [x(m), y(m), 15, 15],'EdgeColor', 'r' );
processFlag=0;
% Kalman Predict if kfinit is not 0, otherwise initialize parameters
if kfinit==0
  xp = [MC/2,MR/2,0,0]';
  P = 1*eye(4); % Process error covariance
  PP= P;
else
  xp=A*xyv(m-1,:)' + Bu;
  PP = A*P*A' + Q;
end

kfinit=1;
% Update
K = PP*H'*inv(H*PP*H'+R);
xyv(m,:) = (xp + K*([x(m),y(m)]' - H*xp))';
P = (eye(4)-K*H)*PP;
```

```
      xyv(m,3) = lat_ped_vel(i);

      rectangle('Position',[xyv(m,1),xyv(m,2),0.5,0.5],'Curvature',1,'facecolor','yellow');

      plot(handles.Topview, [xyv(m,1)+0.25 lat_pos_ped(i)+0.25],[xyv(m,2)+0.25 
long_pos_ped(i)+0.25], 'color', 'black');

      hold off;

      pre_ID =ID;


      % Classification of theta

      er_vel = 3; %

      dd=4;

      dt = 0.44;

      if(ctrf > 1 && i > 2 && i< 1086)

        if( abs(lat_pos_ped(i) - lat_pos_ped(i-1))/0.055 < er_vel )

          est_lat_vel_ped(ctrf) = (lat_pos_ped(i) - lat_pos_ped_pre(i))/0.055;

        else

          est_lat_vel_ped(ctrf) = est_lat_vel_ped(ctrf-1);

        end


        if(abs(long_pos_ped(i) - long_pos_ped(i-1))/0.055 < 1)

          est_long_vel_ped(ctrf) = (long_pos_ped(i) - long_pos_ped_pre(i))/0.055;

        else

          est_long_vel_ped(ctrf) = est_long_vel_ped(ctrf-1);

        end


        if(ctrf > 30)

          mean_est_long_vel_ped(ctrf) = mean(est_long_vel_ped(ctrf-30:ctrf));

          mean_est_lat_vel_ped(ctrf) = mean(est_lat_vel_ped(ctrf-30:ctrf));

          mean_est_theta_ped(ctrf) = 
rad2deg(atan(mean_est_long_vel_ped(ctrf)/mean_est_lat_vel_ped(ctrf)));

          est_lat_acc_ped(ctrf) = (mean_est_lat_vel_ped(ctrf)-mean_est_lat_vel_ped(ctrf-dd))/dt;


          % Check sudden actions based on 30 frames

          g_arr = zeros(1,30);

          a_arr = zeros(1,30);

          d_arr = zeros(1,30);

          gCheckPre30 = zeros(1,30);

          aCheckPre30 = zeros(1,30);

          dCheckPre30 = zeros(1,30);


          for h = 1:30

            g_arr(h) = mean_est_theta_ped(ctrf+1-h)-mean_est_theta_ped(ctrf-h);

            a_arr(h) = est_lat_acc_ped(ctrf+1-h) - est_lat_acc_ped(ctrf-h);

            d_arr(h) = -a_arr(h);
            if(h>1 && g_arr(h) -g_arr(h-1) < 20)
              gCheckPre30(h) = 1;
            end

            if(h>1 && a_arr(h) -a_arr(h-1) < 0.15 && a_arr(h) ~= 0)
              aCheckPre30(h) = 1;
            end

            if(h>1 && d_arr(h) - d_arr(h-1) > -0.15 && a_arr(h) ~= 0)
              dCheckPre30(h) = 1;
            end
          end

          cp = sum(gCheckPre30);
          ap = sum(aCheckPre30);
          dp = sum(dCheckPre30);
          if(cp == 29 && mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-1) >150)% Check Sudden
back
            nextStepClassifiy = 1;
          elseif(cp == 29 && mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-29)> 30 &&
mean_est_theta_ped(ctrf)>0 && mean_est_theta_ped(ctrf-29)<0)% sudden far
            nextStepClassifiy = 2;
          elseif(cp == 29 && abs(mean_est_theta_ped(ctrf)-mean_est_theta_ped(ctrf-29))> 30 &&
mean_est_theta_ped(ctrf)<0 && mean_est_theta_ped(ctrf-29)>0)% sudden close
            nextStepClassifiy = 3;
          end
          if(ap == 29)% Check acceleration
            nextStepClassifiy = 4;
          elseif(dp == 29)% Check dcceleration
            nextStepClassifiy = 5;
          end

          if(nextStepClassifiy ==1)% Check Sudden back
            mean_est_lat_vel_ped(ctrf+1)=mean_est_lat_vel_ped(ctrf-1);
          elseif (nextStepClassifiy ==2)% sudden far
            mean_est_long_vel_ped(ctrf) = 1.02*mean_est_long_vel_ped(ctrf);
          elseif (nextStepClassifiy ==3)% sudden close
            mean_est_long_vel_ped(ctrf) = 0.98*mean_est_long_vel_ped(ctrf);
          elseif (nextStepClassifiy ==4)
            mean_est_lat_vel_ped(ctrf)=1.02*mean_est_lat_vel_ped(ctrf);
          elseif (nextStepClassifiy ==5)
            mean_est_lat_vel_ped(ctrf)=0.98*mean_est_lat_vel_ped(ctrf);
```

```
      else
        % Deflaut: no changes
      end

      if(mean_est_lat_vel_ped(ctrf) ~= 0)
        if(processFlag == 0) %ekf pro
          t1 = abs(xyv(m,2) / car_vel);
          t2 = abs((abs(xyv(m,1))-1)/xyv(m,3));
          t3 = abs((abs(xyv(m,1))+1)/xyv(m,3));
          ttc(ctrf) = abs(xyv(m,2) / (car_vel-xyv(m,3)));
        else
          t1 = abs(xyv(m,2) / car_vel);
          t2 = abs((abs(xyv(m,1))-1)/mean_est_lat_vel_ped(ctrf));
          t3 = abs((abs(xyv(m,1))+1)/mean_est_lat_vel_ped(ctrf));
          ttc(ctrf) = abs(xyv(m,2) / (car_vel-mean_est_long_vel_ped(ctrf)));
        end

        if(abs(ttc(ctrf)) > 8)
          ttc(ctrf) = 7;
        end

        if(t1 >= t2)
          if(ttc(ctrf)>10)
            buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d', ID, Type, ttc(ctrf));
          else
             buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d EXCEEDED', ID, Type, ttc(ctrf));
          end
          set(handles.spot1,'String', buf);
        end
      end
    end
end
end



% Call function to play video
calllib('vidsframe','seek',i);
calllib('vidsframe','get_frame',p_frame_buf);
p_frame_data =  calllib('vidsframe', 'get_frame_info');
axes(handles.Image);
imshow(get(p_frame_buf,'Value')');
```

```
  % In Camera View
  % Mark the pedetrian position
%   if( top>0 && bottom>0 && right-left>0 && top-bottom>0)
%     %axes(handles.Image);
%     rectangle('Position', [left, (960-top), (right-left), (top-bottom)],'EdgeColor', 'g' );
%     text(left, (960-top),txt_str,'color',[1,1,1]);
%   end

%   buf = sprintf('ID:%-4d Type:%-4d', ID, Type);
%   set(handles.spot1,'String', buf);
%   x_po = (left+right)/2;
%   y_po = 960-(top+bottom)/2;
%   rectangle('Position', [x_po, y_po, 15, 15],'EdgeColor', 'r' );
   set(handles.slider1,'Value',currentFrame);
   currentFrame = currentFrame + 1;
  end
  nsize = size(mean_est_lat_vel_ped);
  if(startStopPauseFlag == 0)
    figure;
    subplot(4,1,1);
    plot(mean_est_lat_vel_ped(30:nsize(2)-1)); %nFrame
    axis([0 nsize(2) -6 6]);
    subplot(4,1,2);
    plot(mean_est_long_vel_ped(30:nsize(2)-1));
    axis([0 nsize(2) -6 6]);
    subplot(4,1,3);
    plot(mean_est_theta_ped(30:nsize(2)-1),'o');
    title('theta deg mean');
    axis([0 nsize(2) -100 100]);
    subplot(4,1,4);
    plot(est_lat_acc_ped(30:nsize(2)-1));
    title('Accel lat');
    axis([0 nsize(2) -3 3]);
    grid;
    figure;
    plot(ttc,'o');
    figure;
    plot(ad_ttc,'o');
  end
  if(startStopPauseFlag == 1 || startStopPauseFlag == 2)
    % Call function to play video
    calllib('vidsframe','seek',i);
    calllib('vidsframe','get_frame',p_frame_buf);
    p_frame_data =  calllib('vidsframe', 'get_frame_info');
    axes(handles.Image);
    imshow(get(p_frame_buf,'Value')');
    ID =  mudp_data.vision_obstacles_info.visObs.id(currentFrame,1);
    Type =  mudp_data.vision_obstacles_info.visObs.obstacle_class(currentFrame,1);
    if(mudp_data.vision_obstacles_info.visObs.obstacle_class(currentFrame,1) ~= 4)
      buf = sprintf('ID:%-4d Type:%-4d TTC:%-4d', ID, Type, ttc(ctrf));
      set(handles.spot1,'String', buf);
    else
      axes(handles.Topview);
      plainviewplayer(currentFrame,handles);
    end
  end

unloadlibrary('vidsframe');
```

Appendix D. Matlab Program for Data Selection

```
[filename pathname filter] = uigetfile('*.avi','Select AVI File');

avi_path = [pathname,filename]
ind = findstr(avi_path, '.');
mudp_path = [avi_path(1:ind),'mudp']
mudp_data = read_mudp_data(mudp_path, [5]);



pathname1 = mudp_data.vision_obstacles_info.visObs;
%% edit here
st =n1/2;
en =m1/2;
num = 1;
%%
top1 = pathname1.pixel_top(st:en,num);
bottom1 = pathname1.pixel_bottom(st:en,num);
left1 = pathname1.pixel_left(st:en,num);
right1 = pathname1.pixel_right(st:en,num);
lat_pos1 = pathname1.lat_pos(st:en,num);
long_pos1 = pathname1.long_pos(st:en,num);
lat_vel = pathname1.lat_vel(st:en,num);
long_vel = pathname1.long_vel(st:en,num);
veh_vel = mudp_data.vision_vehicle_info.vehicleVelocity(st:en,num);
veh_yr = mudp_data.vision_vehicle_info.vehicleYawRate(st:en,num);

 x1 = left1+(right1 - left1)/2;
 y1 = bottom1+(top1 - bottom1)/2;

% x1 = lat_pos1;
% y1 = long_pos1;
```

```
c1 = linspace(1,10,length(x1));
figure;
scatter(x1,y1,[],c1);
axis([0, 1280, 0, 960]);
data1 = [x1';y1';lat_pos1';long_pos1';lat_vel';long_vel';veh_vel';veh_yr']';
%% edit here
save('sel_s_wst_data1','data1');
%save('ccs1','mudp_data');
```

Appendix E. Matlab Program for Dynamic Plain View

```
csf_data1 = load('sel_s_wst_data1.mat');
csf_data2 = load('sel_s_wst_data2.mat');
csf_data3 = load('sel_s_wst_data3.mat');
x_ped1 = csf_data1.data1(:,3);
y_ped1 = csf_data1.data1(:,4);
x_ped2 = csf_data2.data1(:,3);
y_ped2 = csf_data2.data1(:,4);
x_ped3 = csf_data3.data1(:,3);
y_ped3 = csf_data3.data1(:,4);

m1 = size(x_ped1);
m2 = size(x_ped2);
m3 = size(x_ped3);
n1 = min([m1(1) m2(1) m3(1)]);
p=0;
vox=[0 0];
voy=[-5 25];
hox=[-15 15];
hoy=[0 0];
hold on;
for i=2:n1
title(Case A /Case B set 1');
    subplot(3,1,1);
    hold on;
    plot(vox,voy,hox,hoy,'b');

rectangle('Position',[x_ped1(i),y_ped1(i)+p,0.02,0.1],'Curvature',1,'facecolo
r','red','edgecolor','red');
    axis([-10 10 -5 15]);
    grid;
    p=0;
```

```
   rectangle('Position',[-1,-3.5+p,1.8,3.5],'facecolor','black');
   axis([-10 10 -5 15]);
   title(Case A /Case B set 2');
    subplot(3,1,2);
    hold on;
    plot(vox,voy,hox,hoy,'b');

rectangle('Position',[x_ped2(i),y_ped2(i)+p,0.02,0.1],'Curvature',1,'facecolo
r','red','edgecolor','red');
    axis([-10 10 -5 15]);
    grid;
     p=0;
   rectangle('Position',[-1,-3.5+p,1.8,3.5],'facecolor','black');
   axis([-10 10 -5 15]);
    title('Case A /Case B set 3');
    subplot(3,1,3);
    hold on;
    plot(vox,voy,hox,hoy,'b');

rectangle('Position',[x_ped3(i),y_ped3(i)+p,0.02,0.1],'Curvature',1,'facecolo
r','red','edgecolor','red');
    axis([-10 10 -5 15]);
    grid;
     p=0;
   rectangle('Position',[-1,-3.5+p,1.8,3.5],'facecolor','black');
   axis([-10 10 -5 15]);


    pause(0.055);
end


    axis([-10 10 -5 15]);


grid
```