COMPUTER VISION BASED ROBUST LANE DETECTION VIA MULTIPLE MODEL ADAPTIVE ESTIMATION TECHNIQUE

by

Iman Fakhari

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering

Department of Mechanical and Energy Engineering

Indianapolis, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Sohel Anwar, Chair

Department of Mechanical and Energy Engineering

Dr. Mohammad Al Hasan

Department of Computer and Information Science

Dr. Lingxi Li

Department of Electrical and Computer Engineering

Approved by:

Dr. Likun Zhu

To the students who are always looking for a single optimum point in their life, but they don't know life is a multi-objective problem and there all hundreds of optimum solutions.

ACKNOWLEDGMENTS

I would like to use this opportunity to express my deep and sincere gratitude to my research supervisor, Professor Sohel Anwar, for giving me the chance to learn and research under his supervision. For trusting me and supporting me in this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

e	error
f	function
f	function
K_p	proportional gain
K_i	integral gain
K_d	derivative gain
p	probability
r	residual
s	steering
z	offset

ABBREVIATIONS

ADAS	advanced driving assistant system
CNN	convolutional neural network
EKF	extended Kalman filter
HSL	Hue saturation lightness
LKAS	lane keeping assistant system
MMAE	multiple model adaptive estimation
MPC	model predictive controller
PID	proportional integral derivative
RCNN	regional convolutional neural network
ROI	region of interest
RNN	recurrent neural network
SVM	support vector machine

ABSTRACT

The lane-keeping system in autonomous vehicles (AV) or even as a part of the advanced driving assistant system (ADAS) is known as one of the primary options of AVs and ADAS. The developed lane-keeping systems work on either computer vision or deep learning algorithms for their lane detection section. However, even the strongest image processing units or the robust deep learning algorithms for lane detection have inaccuracies during lane detection under certain conditions. The source of these inaccuracies could be rainy or foggy weather, high contrast shades of buildings and objects on-street, or faded lines. Since the lane detection unit of these systems is responsible for controlling the steering, even a momentary loss of lane detection accuracy could result in an accident or failure. As mentioned, different lane detection algorithms have been presented based on computer vision and deep learning during the last few years, and each one has pros and cons. Each model may have a better performance in some situations and fail in others. For example, deep learning-based methods are vulnerable to new samples. In this research, multiple models of lane detection are evaluated and used together to implement a robust lane detection algorithm. The purpose of this research is to develop an estimator-based Multiple Model Adaptive Estimation (MMAE) algorithm on the lane-keeping system to improve the robustness of the lane detection system. To verify the performance of the implemented algorithm, the AirSim simulation environment was used. The test simulation vehicle was equipped with one front camera and one back camera used to implement the proposed algorithm. The front camera images are used for detecting the lane and the offset of the vehicle and center point of the lane. The rear camera, which offered better performance in lane detection, was used as an estimator for calculating the uncertainty of each model. The simulation results showed that combining two implemented models with MMAE performed robustly even in those case studies where one of the models failed. The proposed algorithm was able to detect the failures of either of the models and then switch to another good working model to improve the robustness of the lane detection system. However, the proposed algorithm had some limitations; it can be improved by replacing PID controller with an MPC controller in future studies. In addition, in the presented algorithm, two computer vision-based algorithms were used; however, adding a

deep learning-based model could improve the performance of the proposed MMAE. To have a robust deep learning-based model, it is suggested to train the network based on AirSim output images. Otherwise, the network will not work accurately due to the differences in the camera's location, camera configuration, colors, and contrast.

1. INTRODUCTION

1.1 Overview

Lane-keeping systems are one of the necessary options in advanced driving assistant systems (ADAS) and autonomous vehicles (AV). There are two types of lane-keeping systems, active and passive. In the passive lane-keeping systems known as lane departure systems (LDS), the system does not have any control on the steering and only alerts the driver about crossing the lines of the current lane when there is no active turning signal. The active lane-keeping assist systems (LKAS) have relative control over the steering. However, all the current LKAS require continuous driver engagement since the system relies on the line markers, which could be faded or covered in some roads. The LKAS has two central units, a lane detection unit and a steering controller unit. Since steer by wire technology has been developed for many years and used in many modern vehicles, this unit is not the source of failure in LKASs. The origins of failure in LKAS are primarily in the lane detection unit. Years ago, a failure to distinguish the white side of a tractor-trailer from the sky resulted in the first death by AVs. Accordingly, lots of companies are now developing robust algorithms for LKAS to attain the highest possible performance. Most of the researches and developments are around the sources of failures and the solutions. The primary sources of failures in lane detection units are faint or covered lines with dirt or snow, rain and wet ground, other objects, and vehicles blocking the vision of the cameras. The first step to finding out the solutions to avoid these failures is how the lane detection unit works. The lane detection unit uses one or multiple cameras in front of the vehicle and captures the road ahead by implementing computer vision-based or machine learning-based algorithms on these images to locate the lines' position and curvature. There are lots of different lane detection algorithms, and each one has pros and cons in terms of complexity and accuracy. The purpose of this research is to combine multiple models using extended Kalman filter (EKF) and multiple model adaptive estimation (MMAE) to attain a robust algorithm that has the advantages of all models.

1.2 Contribution of This Thesis Work

In this thesis, firstly, two computer vision-based algorithms for lane detection are modified to be compatible with the AirSim simulation environment and developed to calculate the offset between the vehicle center and lane center. Due to the importance of reducing the uncertainty in the lane detection unit, a redundancy of lane detection models is used. MMAE is one of the robust methods for sensor fusion and combination of multiple models, and it has never been used for lane detection purposes. In this study, the MMAE is developed on the output of these two models, which have used front camera images as the input and generated the offset of the vehicle and the lane as the output. A rear camera is used to calculate the offset behind the car, and it is used as the observer of the proposed models. For controlling the steering, different methods are suggested. However, in this study, the focus is not on the controller; hence, a PID controller is developed to control the vehicle's steering and keep the car in the lane. The developed algorithms are tested on different roads in the simulation environment with varying conditions of weather. The results showed that combining the two models using MMAE works significantly better than a single model.

1.3 Organization of the Thesis

As the first step, in Chapter 2, a literature review is provided to illustrate the previous studies on the ADASs, LKAS, and challenges of these systems and show the research gap. In Chapter 3, a problem statement is presented to illustrate the challenges in lane detection by providing different examples in the simulation environment. In this chapter, the two models are tested on various roads, and the different sources of failures are detected and presented. In chapter 4, the methodology of the developed computer vision algorithms, as well as the controller algorithm, are presented. In chapter 5, the simulation results for the proposed MMAE are presented and discussed to illustrate the performance of this method. Finally, in chapter 6, the conclusion and the future works are discussed.

2. LITERATURE REVIEW

An autonomous vehicle is a self-driven vehicle that drives itself with necessary sensors, such as GPS, IMU, cameras, sensors, as can be seen in [1]. They have been in development for over 65 years—in fact, the first cruise control systems were introduced in 1948. With the current rate at which civilization and technological advancement are growing rapidly, especially in the automotive sector, especially in self-driven vehicles. Active vehicular safety has been one of the most researched topics. Among all the vehicles' active-safety options, the lane departure system (LDS) or lane departure warning have had the highest attention; since blind lane departures are known as the first reason for accidents [2]. LDS is also known as one of the base options in advanced driving assistant systems (ADAS), which is now offered by almost all the tire one car manufacture companies [3]. In the following sections, firstly, a review of the uncertainty of current systems for LDS or lane-keeping assistant systems (LKAS) is presented. Then a review of the developed lane detection technics and algorithms is conducted, divided into computer vision-based methods and deep learning-based methods. Finally, a review on multiple model adaptive estimation (MMAE) is presented with a concentration on autonomous vehicles or ADAS usage.

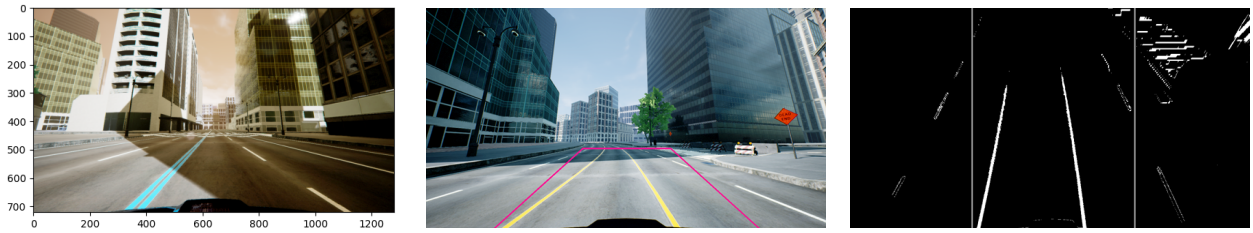
2.1 Lane Detection Methods

There are two methods for lane detection; feature-based approach and model-based approach. The feature-based approach uses low-level features such as edges, whereas the model-based approach uses geometric parameters for detecting lanes. In the feature-based approach, the features used are starting position, orientation, and intensity value. In the initial step, a Sobel operator is applied to get the edge information. The lane boundary is represented as a vector comprising of the three features. The current lane vector is calculated based on the input image and the previous lane model vector. Two windows, one for each, are used for left and right boundaries. Assuming N pixel in each horizontal line, N lane vector candidates are generated. Using a weighted distance metric, the best candidate is selected based on the minimum distance from the previous lane vector. For equalization, each feature is assigned a different weight. Then a lane inference system is used to predict the new lane

vector. If the road width changes abruptly, the current vector calculated is discarded, and the previous one is taken as the current vector [4]. Practically, most of the model-based approaches use the artificial neural network (ANN) as the model for detection. On the other hand, feature-based approaches use image processing and computer vision to detect features of lanes, and model-based use some annotated samples to train a model for lane detection. In the next section, a review on computer vision and deep learning is conducted to elaborate on these two methods.

2.2 Computer Vision Based Methods

In the computer vision-based methods, the overall idea is getting the input image, implementing some pre-processing for making a binary image, cropping the region of interest, implementing edge detection algorithms on the wrapped area, fitting polynomials to the lines and finally calculating the curvature and the center point of the lane. However, the idea of most computer vision-based methods is the same, but there are many different approaches for implementing each step [5]. As is mentioned, the first step of the lane detection process is image pre-processing. A color correction is necessary to reduce the effect of noises, shades, or any unwanted object on the street (before producing the binary image). A Hue Saturation Lightness (HSL) can be implemented on the input image to transform the color space [6]. Another idea to reduce the effect of shade is to filter the edges that are not in the vertical direction (or expected direction) [7]. However, this method cannot overcome and remove all the shades. The next step is removing irrelevant parts of the image which do not contain the lines of the lane. By defining a region of interest (ROI) in the area where we guess the lines are located, we can find a proper ROI [8]. An ROI is mostly a trapezoidal area that will be remapped to a rectangular area in a bird's eye view. The bird's eye view transforms the camera plane image to a 3D world perspective from a point on the top of the vehicle. The bird's eye view is useful since the lines are expected to be vertical in this view [9]. Figure 2.1 (a) shows a color corrected image, (b) shows a proper ROI in the front camera image, and (c) shows the binary bird's eye view image containing the lines of the lane relatively vertical.



(a) Color corrected image (b) ROI of front camera (c) Binary bird's eye view

Figure 2.1. The effect of different computer vision functions on front camera images

The next step is feature extraction. The purpose of this step is to extract the features in the image for the next step, the fitting step. Since the lane markings have an obvious appearance relative to the constant road texture, they can be extracted using gradient-based feature detection methods such as Canny, Sobel, or Hough transform [10]. Another method for detecting the markers is using a known color or brightness for the markers. In this method, the image will be classified according to the color or intensity probability in the image with the expected colors and intensity [6]. The final step of lane detection is fitting. In most algorithms, this process will be done on a bird's eye view image to obtain a better result since the geometry of the lines would be simpler. These methods are based on lane topology. One of the most common methods is sliding window lane detection. In this method, multiple rectangular windows are defined and will be moved all over the ROI to find out the window with the highest intensity of the detected edge. After locating all the windows, a polynomial will be fitted to the center of all these windows representing the detected lane. Borkar et al. [11] have used a linear piece-wise model for fitting and Hough transform for feature extraction. The developed algorithm showed a good performance for short-range detections on the highways. Huang et al. [12] used a parabolic model for fitting and a Random sample consensus (RANSAC) with least square optimization for feature extraction. Their developed algorithm showed a better performance for detecting curved lanes compared to the other common linear models.

2.3 Learning Based Methods

Despite computer vision methods for lane detection, learning methods do not need any assumptions about the geometry of the lines. These models extract the features of the image based on the numbers of extracted or annotated image samples. The performance of different machine learning methods for lane detection is evaluated by Kim et al. [13]. They have investigated methods: Intensity Bump Detection, ANN, Naïve Bayesian Classifier, and support vector machine (SVM). With the development of modern machine learning methods, deep learning became the most popular learning-based lane detection algorithm. A Convolutional neural network (CNN) with five hidden layers is used by Pazhayampallil et al. [14] to detect the lanes in a highway. In their study, the dataset was collected on the highway with ground truth labels which were made by combing camera, Lidar, Radar, and human annotators. The result of their study was a trained model which could detect up to 50 meters of the lane ahead in real-time. In another research, Li et al. [15] proposed two different deep learning models for lane detection. The first one was a CNN that detected the lane's markings and a Recurrent Neural Network (RNN), which had a memory to predict the boundaries of the lane from previously collected data. They could attain a better performance by combining these two models compared to the common available CNNs. Regional Convolutional Neural Network is one machine learning method for segmentation. Liu et al. developed an RCNN for lane detection based on 100,000 annotated images. The trained model showed a 97.9% accuracy. Figure 2.2 shows a schematic of RCNN in their study.

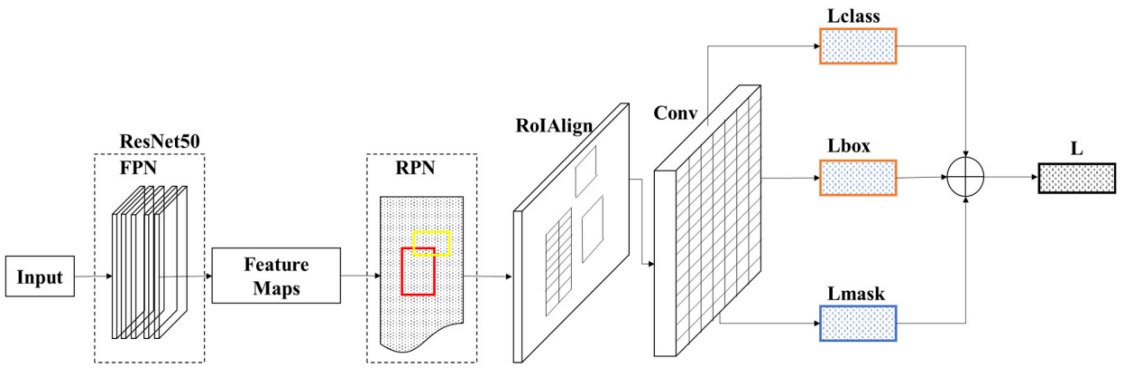


Figure 2.2. The schematic of a Mask-RCNN for lane detection [15]

2.4 Uncertainty of Current Methods

Computer vision and machine learning are two bases of self-driving technology. However, these tools are vulnerable to unseen or noisy situations, and such a misprediction in self-driving is catastrophic since it results in casualties. For example, machine vision failed to distinguish the white side of a turning tractor-trailer from the brightly lit sky, resulting in the first self-driving fatality in the world. Because of these limitations, most of the currently mounted self-driving technologies on vehicles require a human driver to be ready to take back control when the system requests. That is why an efficient perception unit is required in an AV, and uncertainty and fault detection should be considered in the development. Kim et al. [16] have developed the vision-based uncertainty-aware lane-keeping strategy where the high-level reinforcement learning policy hierarchically modulates the low-level lateral control and the reference longitudinal speed. The successfully trained deep reinforcement learning agent slows down the vehicle speed and minimizes the lateral error during high uncertainty situations, similar to what human drivers would do in such situations. The uncertainties in trained deep learning models can be defined in two categories. First, epistemic uncertainty accounts for the failures sourced from the model itself, and second, the aleatory uncertainty accounts for the failures sourced from noises in the observations and input data [17]. Since learning-based lane detection are vulnerable to new data, in this research [16], the authors calculated the uncertainty of learning-based lane detection using computer vision methods. They have proposed a convolutional mixture density network (CMDN) to calculate the vehicle's offset and the center point of the lane. Figure 2.3 shows the control schematic of their study. They have used the AirSim simulation environment for their study. According to this system, they have developed an uncertainty-aware lane detection unit that makes decisions according to the uncertainty of the model at each step. When the uncertainty exceeds a specific value, the speed of the vehicle reduces until the system gets to a step with a reasonable uncertainty.

The reviewed studies show that reducing the uncertainty of the proposed models for lane detection is one of the most important goals in researching and developing these systems.

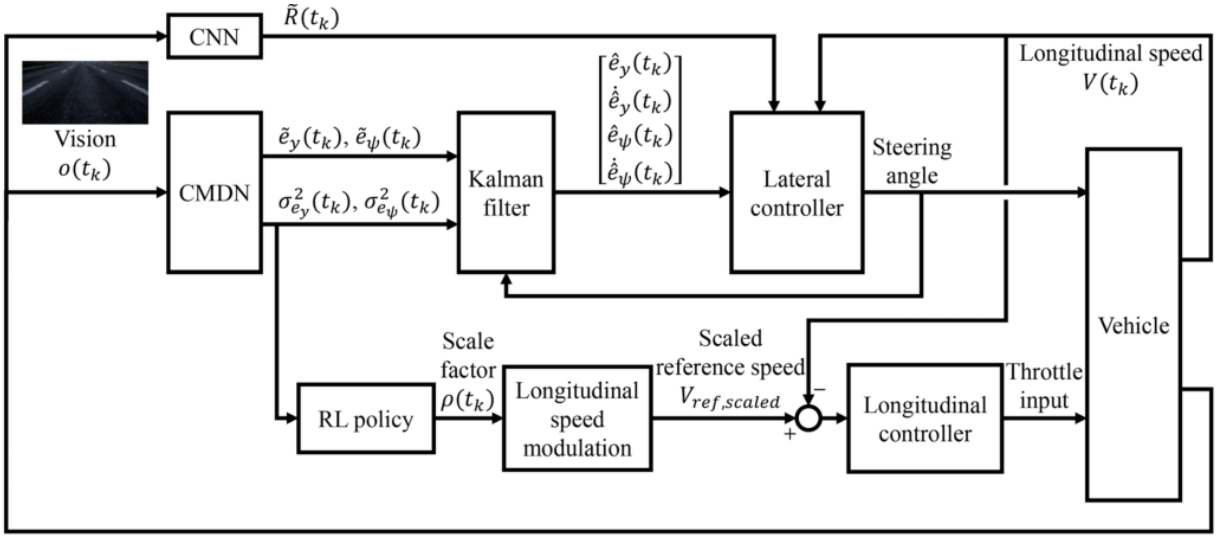


Figure 2.3. The control schematic for an uncertainty aware lane detection system

In the next section, multiple model adaptive estimation will be introduced and reviewed, as a method to reduce the uncertainty of the lane detection models by combining them.

2.5 Multiple Model Adaptive Estimation

Hamilton et al. [18] have merged two important lines of research, the Kalman filter and Takens method. A model-free filter is introduced based on the filtering equations of Kalman and the data-driven modeling of Takens. This procedure replaces the model with dynamics reconstructed from delay coordinates while using the Kalman update formulation to reconcile new observations. They find that this combination of approaches results in a comparable efficiency to parametric methods in identifying underlying dynamics and may be superior in cases of model error. Quinlan et al. [19] have discussed multiple-model Kalman filters that also are explicitly multi-modal. Motivated by the RoboCup SPL, they show how they can be used despite the highly multi-modal nature of sensed data and give a brief comparison with a particle filter-based approach to localization. The implemented MM-EKF was able to handle the ambiguous information directly, and therefore resultant multi-modal distributions common in the SPL. It showed substantially better performance than standard EKF implementations, and at least in a preliminary test, outperforms a particle filter applied

to the same problem. Barrios et al. [20] The Multiple Model Adaptive Estimation System (MMAE) algorithm is applied to the integration of GPS measurements to improve efficiency and performance. This paper evaluates the multiple-model system in different scenarios and compares it to other systems before discussing possible improvements by combining it with other systems for predicting vehicle location.

2.6 Control System

After the detection process, an offset between the vehicle center point and the lane center point is provided by the lane detection unit. In some more advanced methods, the curvature of the lane and the direction and velocity of the vehicle at the current time step are also considered. The final step is using these inputs to control the steering of the vehicle and drive it to the center point of the lane. There are multiple methods and controllers for implementing this part of the system. Pure pursuit controller is one of the simplest methods. Kuo et al. [21] developed a lane-keeping system using a pure pursuit controller and a lane detection algorithm using inverse perspective mapping. Their study used the developed algorithms on a robot and an onboard computer (ARM). Accordingly, the performance of the developed algorithms was their priority. The developed algorithm had a runtime of 11ms for lane detection, with up to 10% enhancement compared to similar studies. In addition, not only the developed algorithm was efficient it also performed effectively. The average lane detection error in their algorithm was within 5%. Kamat [22] developed and analyzed passenger cars' Lane Keep Assist Syscar in a simulated environment using Implicit and Explicit MPC. The authors examined their suggested approach's closed-loop performance in order to aid in the Model in Loop validation of control software development. This might simplify and accelerate its implementation of the integrated hardware necessary for the Advanced Driver Assistance System's LKA function. The study employs basic equations to construct a simpler model for a vehicle's lateral dynamics in the time domain using a Linear Parameter Variant state-space model for a regularly used speed range. The IMPC and EMPC controller actions were determined by building and solving optimization problems using a Quadratic Programming technique and a multi-parametric PWA solver approach.

Bujarbaruah et al. [23] suggested an Adaptive Robust Model Predictive Control technique for lateral control in lane maintaining challenges, in which the steering system's constant steering angle offset is learned. This research aimed to reduce the outputs, which are the distance from the lane center line and the steady-state heading angle error, while still adhering to the corresponding safety requirements. Using data, a rigorous Set Membership Method-based technique is used to determine the real-time maximum achievable limit for the steering angle offset. Their approach is well-suited for situations with abrupt curves at high speeds, where establishing a correct model bias for restricted control is challenging, but data-driven learning may be beneficial. The findings indicated that adopting a switching technique during lane curvature changes was persistently feasible. Samuel et al. [1] developed two MPC and PID controllers for a lane-keeping system using MATLAB and Simulink. They have compared the performance of both controllers and their results showed that even though the MPC had a better performance in terms of changing in the vehicle condition but the PID was able to achieve the objective in all the conditions.

2.7 Summary

LKAS is one of the main options of ADAS, which can increase the safety of vehicles substantially. As explained, there are multiple methods for implementing lane detection in LKAS, which can be computer vision-based or machine learning-based. However, all the methods have certain limitations. The primary purpose of this study is to reduce the uncertainties under road markings as well as weather conditions. In this research, a combination of two computer vision-based models using MMAE for lane detection and a PID controller is used to keep the vehicle in the lane.

3. PROBLEM STATEMENT

In this chapter, the sources of failure in lane detection are investigated. As it was mentioned, two different models are going to be used in this research. These two models are tested in different case studies that resulted in failure for either of models and presented in this chapter. Finally, these failures are divided to two groups. First, the unsolvable failures which are the ones that do not have any solution e.g. the covered lines by the snow. Second, the solvable failures that can be prevented by some adjustments in the models e.g. a missing part in a line or a sharp turn. This research has a focus on the second group of failures.

3.1 Unsolvable Failures

In this section the unsolvable failures such as snow or missing lines are investigated and illustrated.

- **Snow**

  When the snow covers the lines there is no way to detect the lines. Figure 3.1 shows the simulation environment in a snow situation and the detection result by the both models.

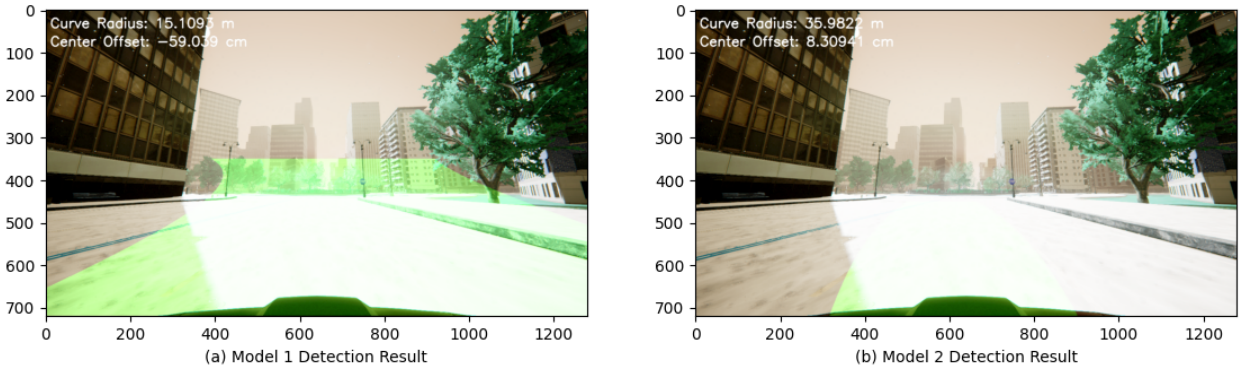


Figure 3.1. The failure of lane detection models in snow

- **Missing Lines**

  There are some streets that does not have the line markers! This is also another

unsolvable situation. Figure 3.2 shows the vehicle in a two way street that does not have the center line!

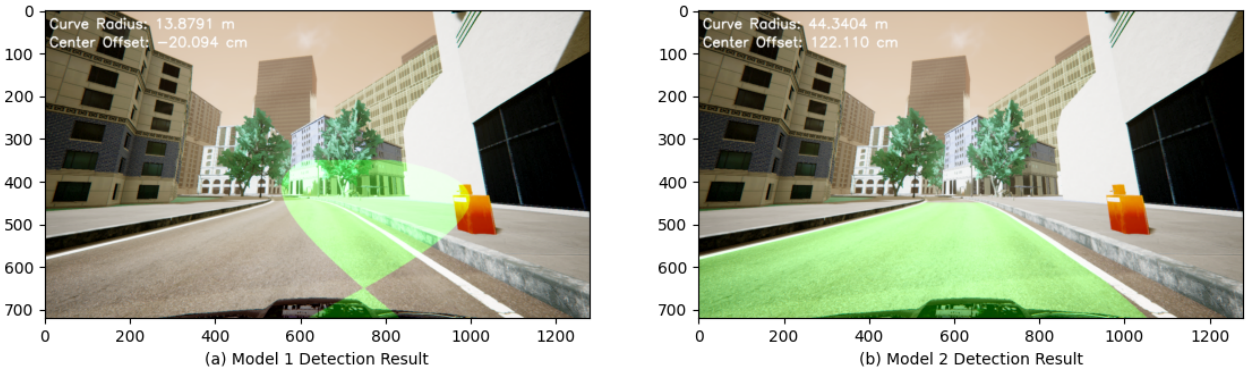


Figure 3.2. The failure of lane detection models in a missing marker situation

3.2 Solvable Failures

In this section the solvable failures are presented and illustrated. These failures can be caused by a wet ground or a weak vision by rain, a missing part of lines, fog, or sharp turns.

- **Wet Ground and Rain**

  Figure 3.3 shows the failure in lane detection in a wet ground situation. In fact, the reflex of lights in a wet ground will result in a confusion for the detection algorithms which results in a failure.

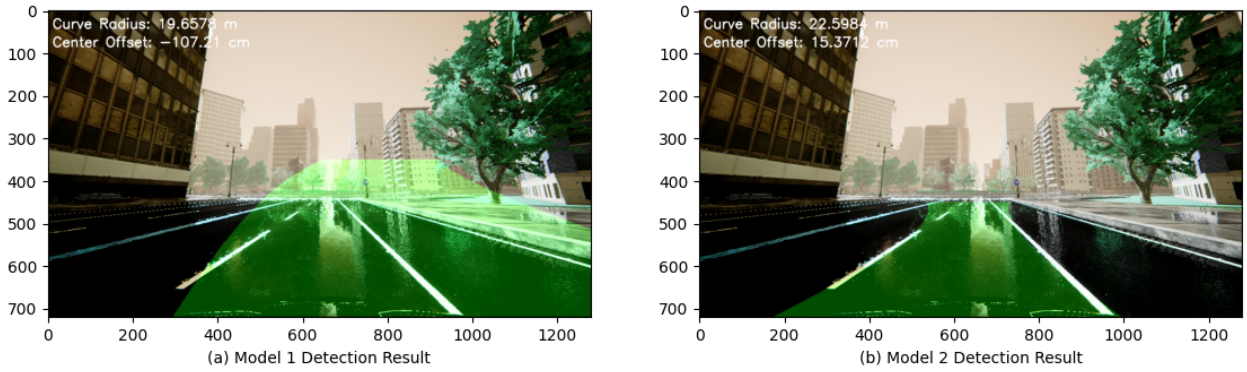


Figure 3.3. The failure of lane detection models in a wet ground and rainy situation

- **High Contrast Shade**

  Figure 3.4 shows an image that contains a high contrast shade on the street. In this situation the histogram of detected edges will have an extra noise because of the shade and that will result in a failure.

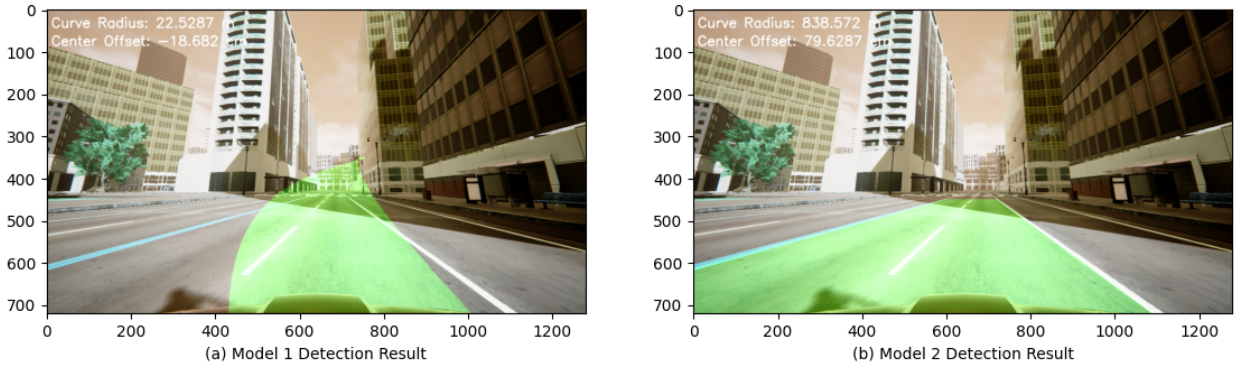


Figure 3.4. The failure of lane detection models in a street with a high contrast shade

- **Missing Parts of Lines**

  In Figure 3.5 the left line of the lane has a missed marker and there is a long distance between the camera and the next available marker which resulted in a failure.

- **Sharp Turns**

  In Figure 3.6 the vehicle is located in a street with a sharp turn. The second model which has a smaller ROI couldn't detect the curvature of the lane correctly however model 1 could detect the entire lane.

3.3 Summary

According to the stated problem, two different models of lane detection are implemented and each one has some faults under some of the solvable conditions. However, in most of the conditions one of the models could attain a reasonable value for offset. In the next chapter, these two models are going to be evaluated in details and their pros and cons will be presented. In addition, MMAE will be presented as the solution to cover the faults of the failed model in each time step. The proposed model is going to be validated in AirSim

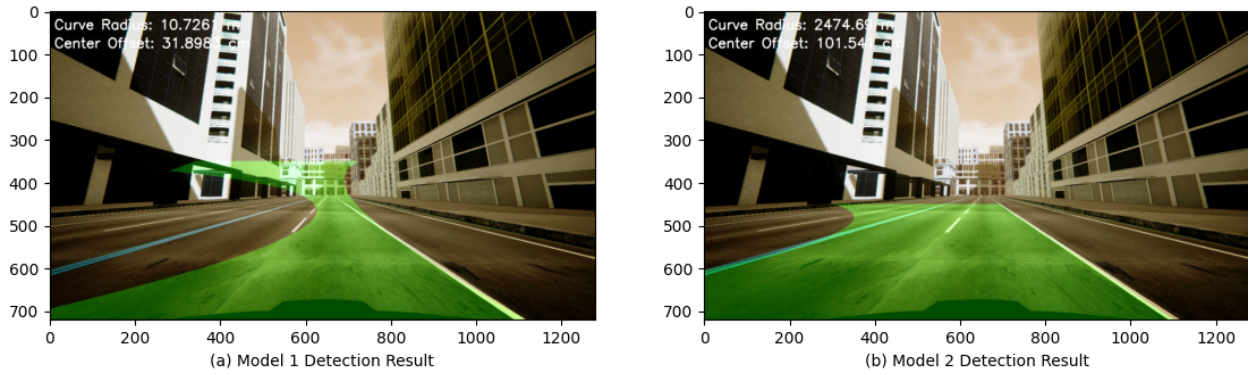


Figure 3.5. The failure of lane detection models in a street with a missed part of line marker

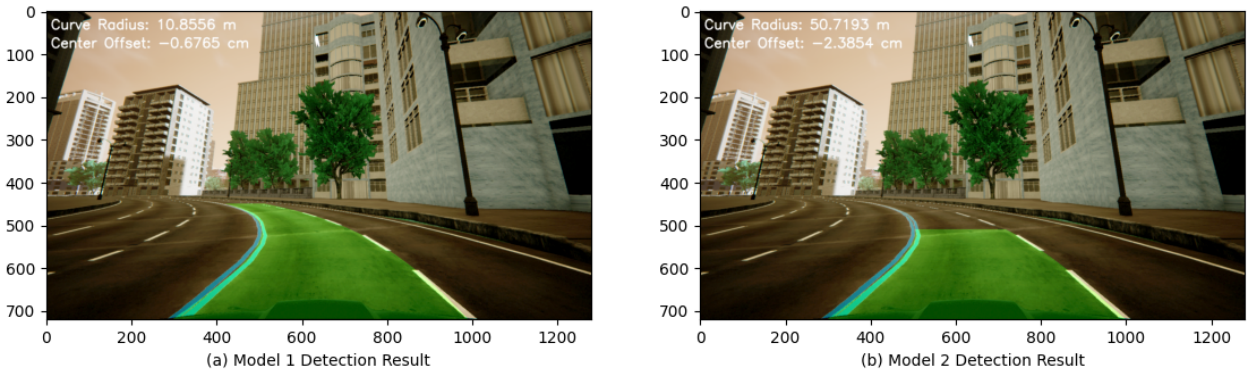


Figure 3.6. The failure of lane detection models in a street with a sharp turn

environment. To keep the vehicle in the lane a PID controller is used to control the steering based on the vehicle offset with the lane.

4. MMAE BASED LANE DETECTION

In this chapter, firstly, the location of the front and rear cameras and the configuration of cameras are illustrated. The front camera is used as the model's input image, and the rear camera is used for the estimator. Two different models for the lane detection of front camera images are presented. Both models are based on computer vision. The first model is more straightforward than the second model, but its ROI is larger and has a better performance in long-distance detection. The second model is more advanced; however, it has a smaller ROI and is robust for short-distance detection. In the next step, each model is evaluated, and the final result of their lane detection is presented. After that, the estimator and the Multiple Model Adaptive Estimation (MMAE) algorithms for combining these two models are presented. The formulation and the utilized equations for MMAE are also illustrated in this section. The PID controller for controlling the steering is explained in the next step, and tuned gains for the PID are presented. Finally, the proposed lane detection and lane-keeping system are validated against various case studies.

4.1 Simulation Environment

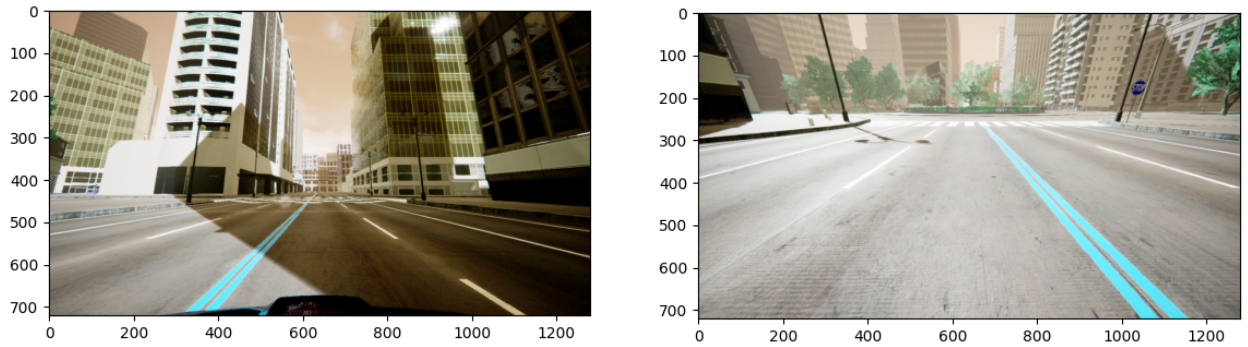
To evaluate models and implement lane detection in a software in loop; AirSim simulation environment is used. The City release of AirSim contains a limited number of streets and buildings of a simulated urban area. Two different cameras are installed on the vehicle for models and observer. The first camera is the front-camera and is located on the front tip of the roof of the vehicle. The second one is the back camera which is located in the back tip of the roof of the vehicle. The selected vehicle in the simulation is PhysXCar which is an SUV similar to BMW X4. Figure 4.1 shows the location of front and back cameras on the vehicle.

The setting.json file contains the configuration of AirSim environment. This file as well as the other codes are presented in a GitHub repo. The configuration of cameras can be found as below. The field of view for the front and back cameras are selected as 120 and 150 degrees, respectively. The pitch of the cameras are also fixed as 10 and -20 degrees, respectively. In AirSim the origin of coordinates is fixed on the center of the vehicle based

Figure 4.1. The locations of front and back cameras on the vehicle.

on NED coordinates where X is always pointing to the vehicle front. All these values found by try and error to get the best possible results in terms of lane detection.

Figure 4.2 (a) and (b) show the perspective view of front and back cameras, respectively. As it is clear from these images the pitch of front camera is positive and the pitch for the back camera is negative. The reason behind these values is the fact that the back camera is one step behind of the front camera and the immediate offset behind the car should be used for a robust observer in the controller system. Note that these images are output of pre-process (color correction) function.



(a) Front Camera View (b) Back Camera View

Figure 4.2. The perspectives of front and back cameras

Table 4.1. Camera configuration for front and rear cameras (Positions in meter and orientations in degree).

	Front Camera	Rear Camera
X	1	-2
Y	0	0
Z	-1.5	-1.5
$Pitch$	10	-20
$Roll$	0	0
Yaw	0	180
FOV	120	150
$Width$	1280	1280
$Height$	720	720

4.2 Lane Detection Models

In this research, two different lane detection algorithms have been used based on computer vision. However, it should be mentioned that another model based on CNN was considered to be a part of the MMAE. But due to the differences between the training dataset and the AirSim images (in terms of camera perspective, image colors and contrast) the CNN could not work robust on AirSim images. The schematic of these two models are presented in Figure 4.3. Accordingly, in model 1 the input image will be cropped using the ROI extraction, then the image will be transformed o binary and then the bird eye view of the image is extracted. By implementing Hough transform on the image the lines can be extracted. Finally after finding the lane the center point of the closest part of the lane to the vehicle (first row of the image) will be used as the offset. In model 2, firstly, a camera calibration is done the the image will be cropped and ROI will be extracted. For making the binary, the thresholding will be done using Sobel, Saturation and Hue thresholding. Then the binary image will be

passed to the sliding windows algorithm and using that algorithm the lines are detected. Similarly the offset of the vehicle and the lane will be calculated as the controller input.

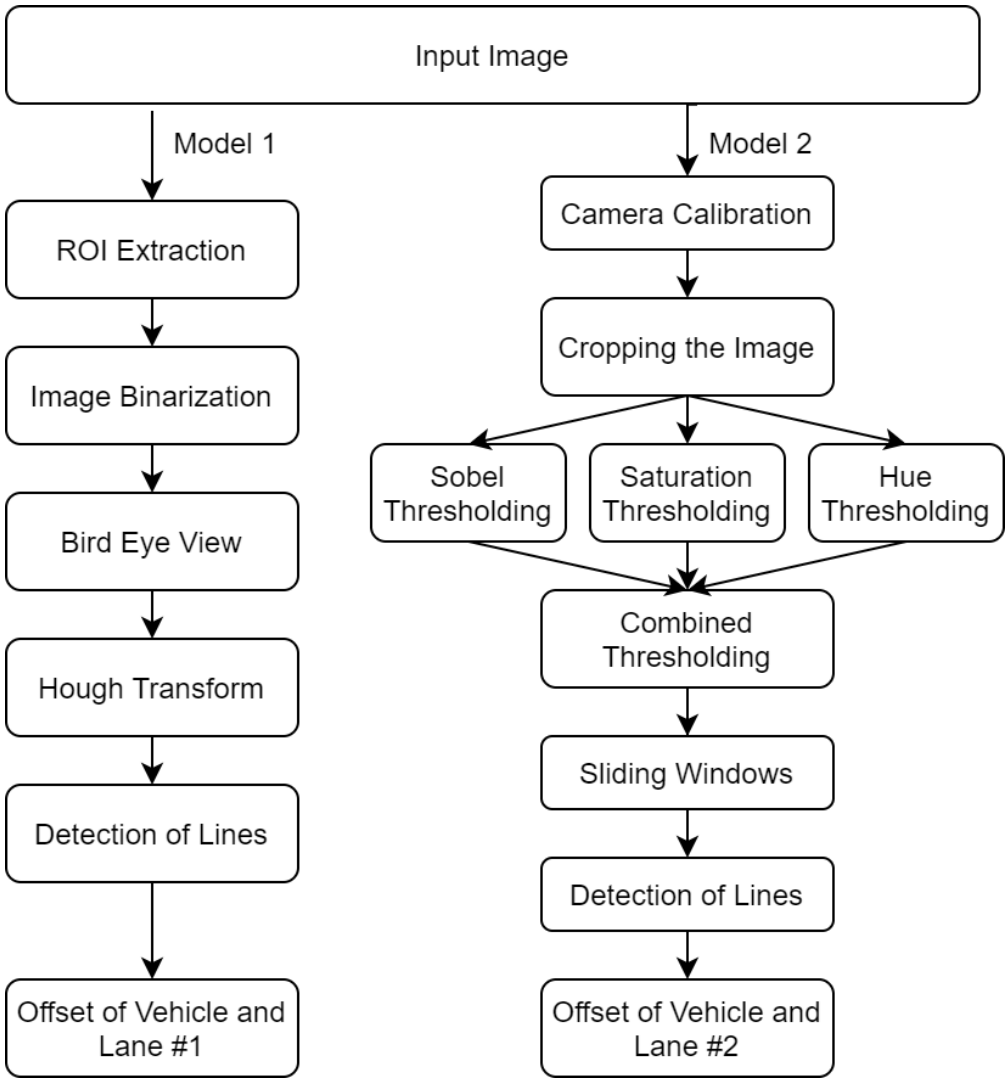


Figure 4.3. The schematic of two different models for lane detection and offset calculations.

4.3 Model 1 Evaluation

To evaluate the first model, the vehicle is located in the middle of a street and the received image from front camera is imported to the model 1 function. Figure 4.4 shows the region of interest (ROI) which is extracted for model 1 using the front camera data. The

trapezoidal area will be converted to a rectangle in the bird eye view image. The edges of the trapezoidal area are defined as below in a 1280*720 pixels image. Note that the origin of XY coordinates in Python matplotlib is top left the image. These values should be adjusted according to FOV and location of the camera in order to contain the lane inside it.

```
self.roi_points = np.float32([
  (500,450), # Top-left corner
  (200, 720), # Bottom-left corner
  (1080,720), # Bottom-right corner
  (780,450) # Top-right corner
])
```

FOV for the front camera is chosen to be 120 degrees. This value found using try and error to have the best view on the street on the current lane. To find bird's eye view image the intrinsic and extrinsic camera parameters are required. AirSim client returns an estimation of extrinsic camera parameters. However, it returns the transformation matrix and rotation matrix separately and the should be combined manually to get the extrinsic camera matrix. The focal length of camera for intrinsic matrix can be estimated using FOV as:

$$FoV = 2arctan(W/2f) \tag{4.1}$$

Where FoV is the field of view in radiant and W is the image width and f is the focal length.

As it was explained in the pipeline of model 1, in the next step the imported image will be converted a binary image and the to bird's eye view image. Figure 4.5 shows the black and white bird's eye view image of the front camera. Accordingly, the lines of the lane are located inside the ROI. In the next step of lane detection in model 1 the histogram of the binary image will be obtained. Figure 4.6 shows the histogram of the presented binary image. Accordingly the areas inside the ROI which have the highest peaks in the histogram will be nominated for line fitting. Figure 4.7 shows the result of detected lines and the lane area which is illustrated with a green overlay.

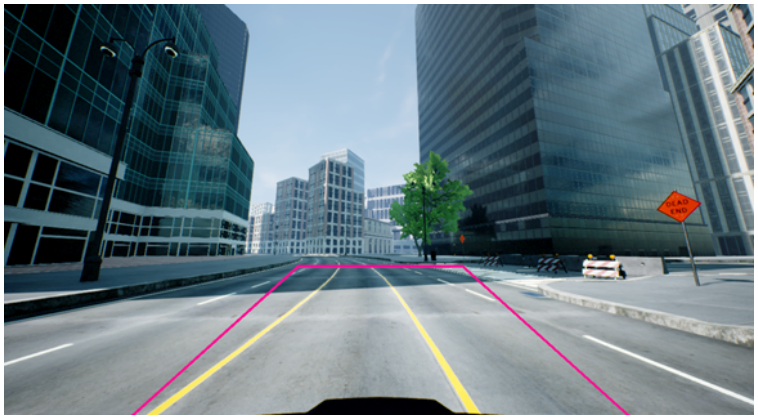


Figure 4.4. The region of interest of model 1 in a sample front camera date.

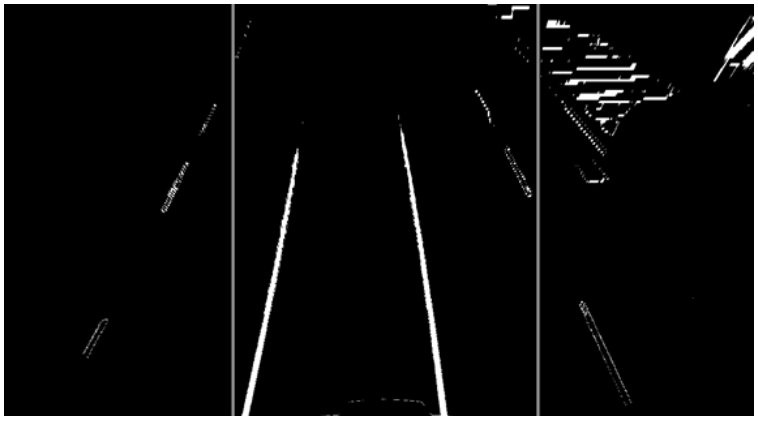


Figure 4.5. The binary image of front camera converted to bird's eye view in model 1.

4.4 Model 2 Evaluation

The pipeline of this model is a bit more complicated and advanced. Firstly the camera is calibrated in this method to find the intrinsic and extrinsic camera parameters more accurate. To calibrate the cameras in AirSim the common chess board experiment is designed by Marcelino Almeida for AirSim environment. With modifying the drone camera settings in this project we can find out our camera matrix. Figure 4.8 shows the chess board camera calibration environment in AirSim. In this environment 10 points on the chess board are selected and their position will be extracted from three different positions and this positions are going to be used to find the camera matrix. The undistorted image as a result of camera calibration will be imported to binary thresholding algorithm. The goal is to identify pixels

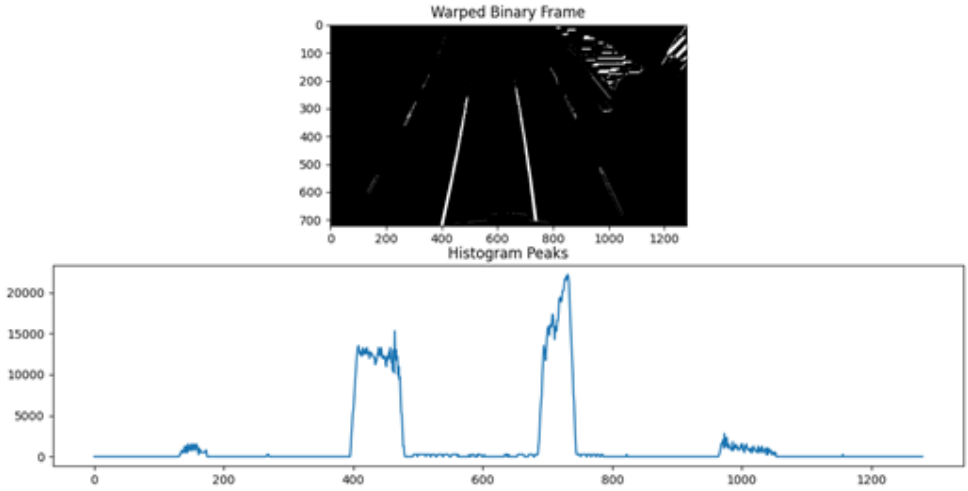


Figure 4.6. The histogram and corresponding binary image of front camera in model 1

that are likely to be part of the lane lines. In the following steps are conducted on the image. 1) Apply the following filters with thresholding, to create separate "binary images" corresponding to each individual filter. 2) Absolute horizontal Sobel operator on the image Sobel operator in both horizontal and vertical directions and calculate its magnitude. 3) Sobel operator to calculate the direction of the gradient. 4) Convert the image from RGB space to HLS space, and threshold the S channel. 5) Combine the above binary images to create the final binary image. Figure 4.9 shows the implemented thresholding algorithms on the input and the final combined result.

After finishing the thresholding the final binary image will be obtained and the ROI in this image will be extracted to transfered to bird's eye view. Figure - shows the ROI for second model which is smaller that ROI of the first model to reduce the time complexity and the final binary image as a result of combination of described methods.

In the next step the sliding windows approach will be implemented to find the windows containing the peaks in the image. Starting with these base positions on the bottom of the image, the sliding window method is applied going upwards searching for line pixels. Lane pixels are considered when the x and y coordinates are within the area defined by the window. When enough pixels are detected to be confident they are part of a line, their average position is computed and kept as starting point for the next upward window. The

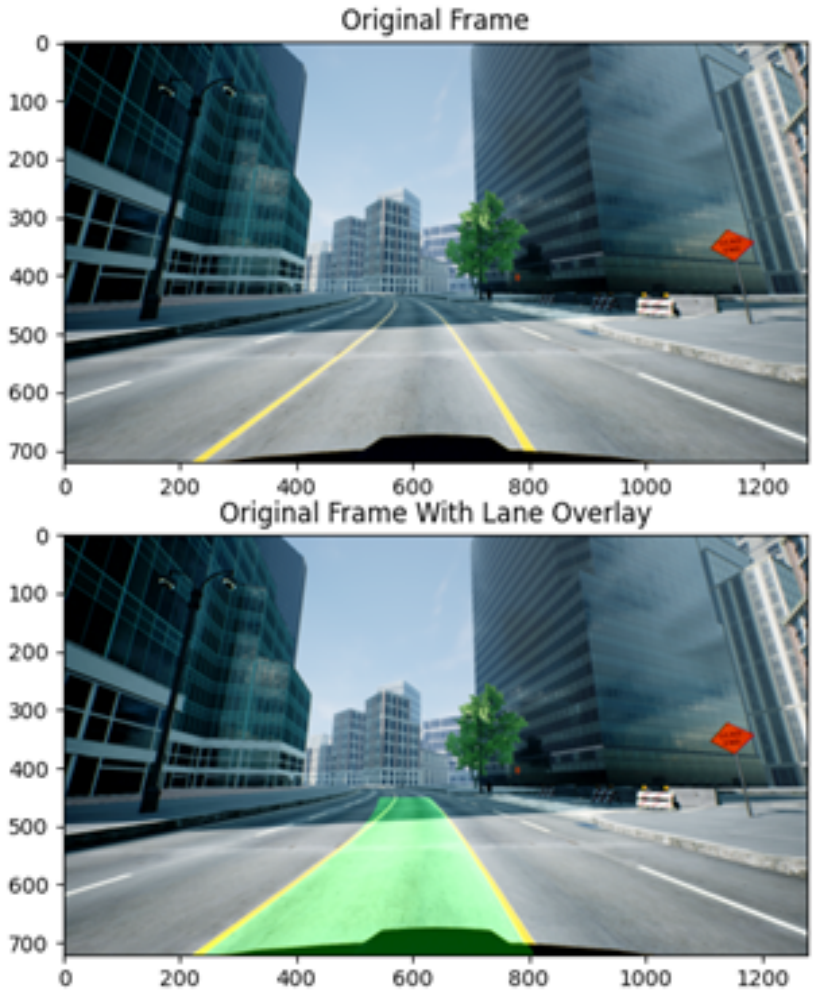


Figure 4.7. The original image and the result of lane detection algorithm by model 1.

number of sliding windows in model 2 is 9 windows which is found by try and error. The final output of sliding windows method is presented in Figure 4.11. The loop for implementation of this algorithm is attached in the appendix. Finally, a second order polynomial will be fitted to the center points of finalized windows and a green mask is implemented between these two polynomials to illustrate the lane area. Figure 4.12 shows the final result of lane detection using model 2. The center offset and curve radius are also calculated similar to model 1. Accordingly, the average of first point in the polynomial is the center point of the lane and the center of the image (here 1280/2) is the center point of the vehicle. The difference between these two values is the center offset.

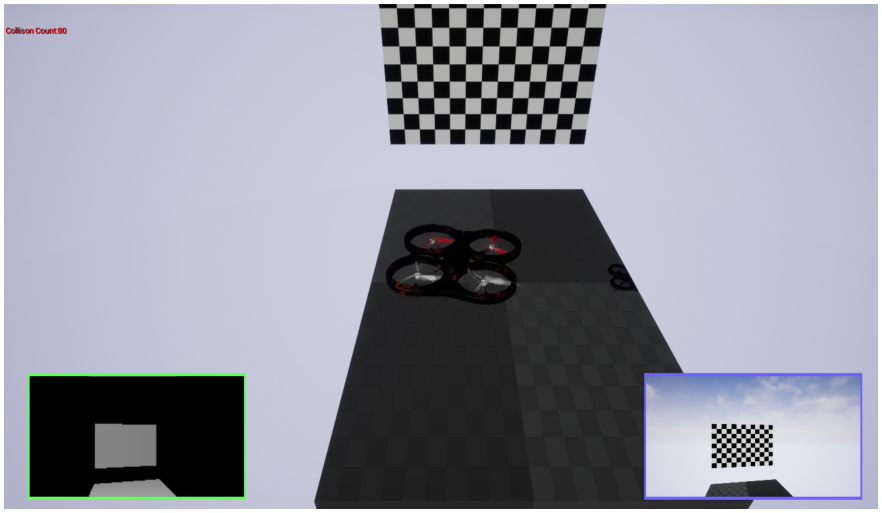


Figure 4.8. The camera calibration environment on AirSim for model 2.

4.5 Multiple Model Adaptive Estimation

So now two different models for calculating offset is derived. As it was mentioned in the previous sections, MMAE is going to be used for combining these two models. The calculated offset from model 1 and model 2 are the output that should be combined. The detected offset using the rear camera image will be used as the observer for calculating the probabilities. In fact, the residuals of each model will be the subtraction model offset and the observer model. However, it should be noted that the observer is one step behind. So the residual can be defined as the difference between the offset of the model at step $n-1$ with the offset of observer at step n. The reason behind this is the fact that the observer is implemented on the rear camera and it is always showing the past offset of the lane. Let's call the offset of the vehicle and the lane as z. Similar to state estimation, here the purpose is estimating the offset using multiple models. The offset can be calculated as:

$$z = f(inputImage) \tag{4.2}$$

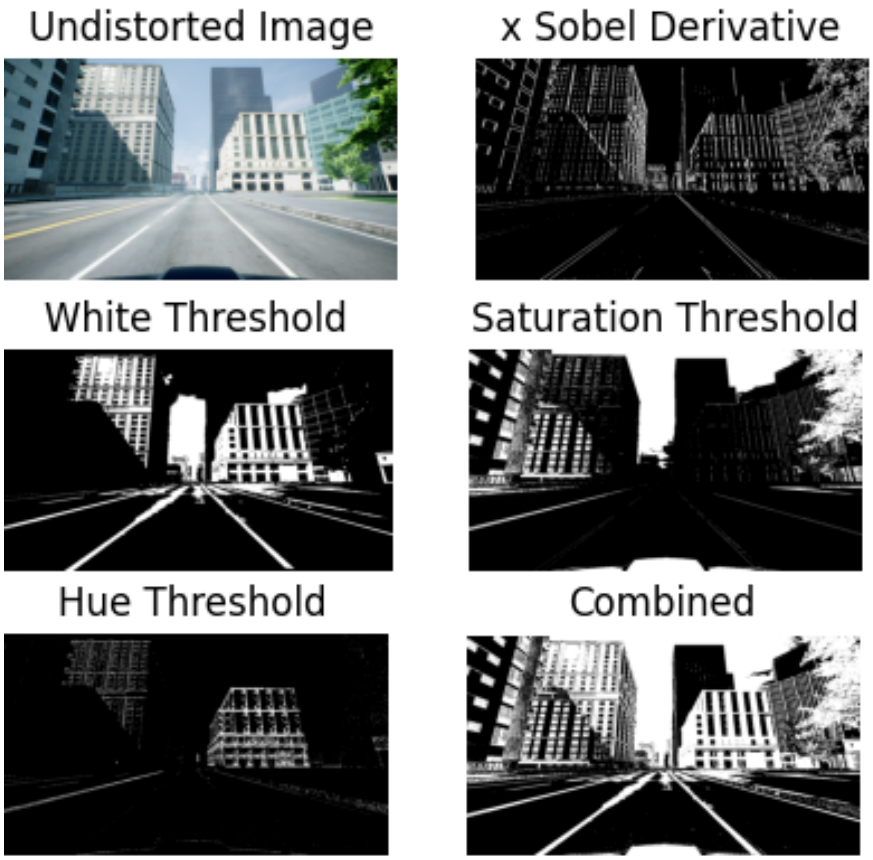


Figure 4.9. Finding the optimum binary image by combining different thresholding algorithms

Where f is the lane detection function which can be any of the proposed models or the observer model and z is the measured offset. The residual for calculation of likelihood can be defined as:

$$r_k = z_n^{est} - z_{n-1}^{model,k} \tag{4.3}$$

Where r_k is the residual for the kth model and z_n^{obs} is the estimator offset at nth step and $z_{n-1}^{model,k}$ is the offset for the kth model at step $n-1$. The probabilty of the two models can be defined as p_1 and p_2 and we know:

$$p_1 + p_2 = 1 \tag{4.4}$$

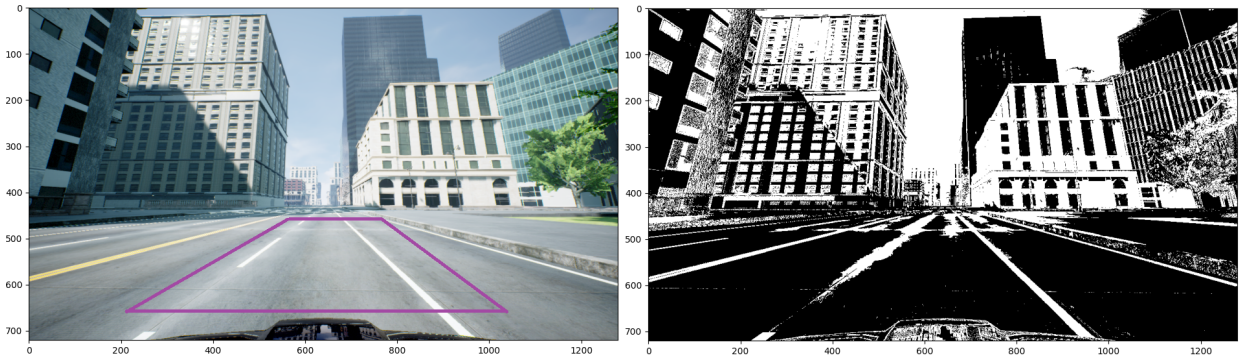


Figure 4.10. The ROI for model 2 and the final binary image of this model

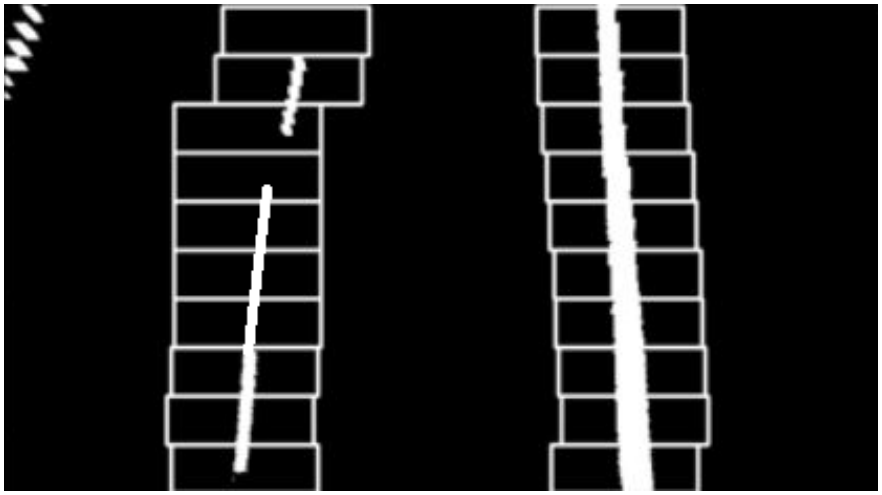


Figure 4.11. The result of sliding window algorithm on the binary bird's eye view image of model 2 on extracted ROI

Where the probabilities can be calculated using following equation for the n^{th} model at time sample k:

$$p_{n,k} = \frac{f_{z(k)\|a,z(k-1)}(z_k\|a_n, z_{k-1})p_n(k-1)}{\sum_{\mathrm{j}=1}^{n} f_{z(k)\|a,z(k-1)}(z_k\|a_{\mathrm{j}}, z_{k-1})p_{\mathrm{j}}(k-1)} \tag{4.5}$$

Where the nominator f is the conditional probability of n^{th} model at time step $k-1$, z is the output of models (the offset), and the denominator is the sum of all the conditional probabilities. Accordingly,

$$f_{z(k)\|a,z(k-1)}(z_k\|a_n, z_{k-1}) = \beta_n exp(o) \tag{4.6}$$

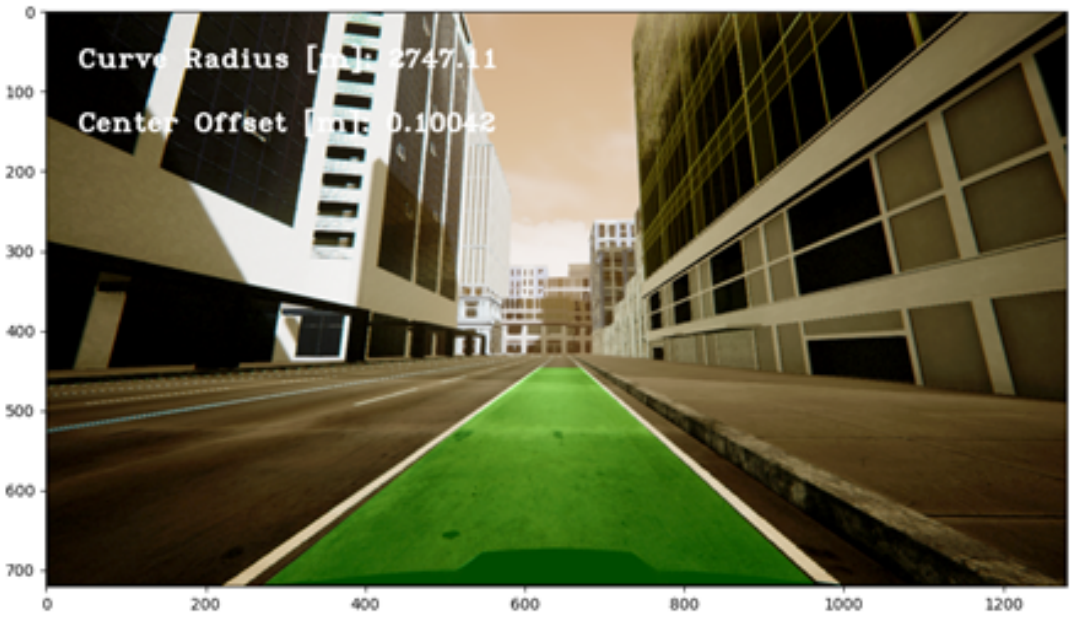


Figure 4.12. The final result of lane detection using model 2

Where,

$$\beta_n = \frac{1}{(2\pi)^{l/2}\|\psi_n(k)\|^{1/2}} \tag{4.7}$$

Where l is the measurement dimension which is 1 here and,

$$(\mathrm{o}) = \frac{1}{2} r_{n,k}^T \psi_{n,k}^{-1} r_{n,k} \tag{4.8}$$

Where $r_{n,k}$ is the residual of offset for the n^{th} model at time step of k, and $\psi_{n,k}$ is the covariance of the residuals.

$$\psi_{n,k} = C_{n,k} P_{n,k} C_{n,k}^T + R \tag{4.9}$$

Where C is output vector $\begin{pmatrix} 1 & 0 \end{pmatrix}$ and P is State covariance matrices of measurement noise is considered to be i*dentitymatrix* at initial step and the value of covariance of the measurement noise R is considered to be 10^{-6} by try and error. The final value of P for both models found as:

$$P_1 = \begin{pmatrix} 6.25\text{e}-13 & -4.16\text{e}-10 \\ -4.19\text{e}-10 & 1.98\text{e}-11 \end{pmatrix}$$
$$P_2 = \begin{pmatrix} 8.14\text{e}-13 & -2.53\text{e}-10 \\ -1.94\text{e}-10 & 1.67\text{e}-11 \end{pmatrix}$$

4.6 PID Controller

Considering e as the offset value in each time step and s as the steering value which is the output of the controller and the input of simulation environment the PID formulation for step n can be presented as:

$$s_n = K_P \mathrm{e}_n + K_I \sum_{\mathrm{i}=1}^{n} \mathrm{e}_\mathrm{i} + K_D(\mathrm{e}_n - \mathrm{e}_{n-1}) \tag{4.10}$$

Where n is the current time step. Note that the time step in unit in the controller model. The implementation of PID controller is straight forward and is presented below. $main_{offset}$ represents the final offset calculation from the MMAE section which will be the error itself; since the reference value is zero. The dt is the time of each time step and the $error_{dot}$ will be the discrete derivative of error. $error_{\mathrm{int}}$ is also the discrete integral of error. Having these three parameters and the tuned PID gains the steering angle can be calculated. Note that the conditions are defined to avoid index problems in the first two time steps.

The PID gains are found by try and error. The tests showed that it would be better to use two different sets of gains for straight roads and curvy roads. Table 4.2 shows the tuned PID gains for this controller.

Table 4.2. Tuned PID gains for proposed controller system.

Gains	Straight Road	Curvy Road
K_p	0.001	0.002
K_i	0	0.0001
K_d	0.001	0.001

4.7 Overview of the Lane Detection System

Figure 4.13 shows the developed control system for the LKAS. According to this figure, the simulation environment at each step provides front and rear camera images. The front camera image will be used in model 1 and 2 for lane detection and offset calculation. Rear camera image will be used in the observer model for observing the offset. The output offset of both models as well as the observer output will be used in Likelihood Function to calculate the probability or weights of each output. With multiplying the weights to the outputs the final offset output will be derived. The subtract of this offset with the reference offset (which is zero) will be used as the input of the PID controller to obtain a proper steering value for the next step. The simulation environment will use the steering to drive the car for one time step. The throttle of the vehicle is always fixed on the max value until the vehicle gets to the defined max speed which is $5m/s$ here.

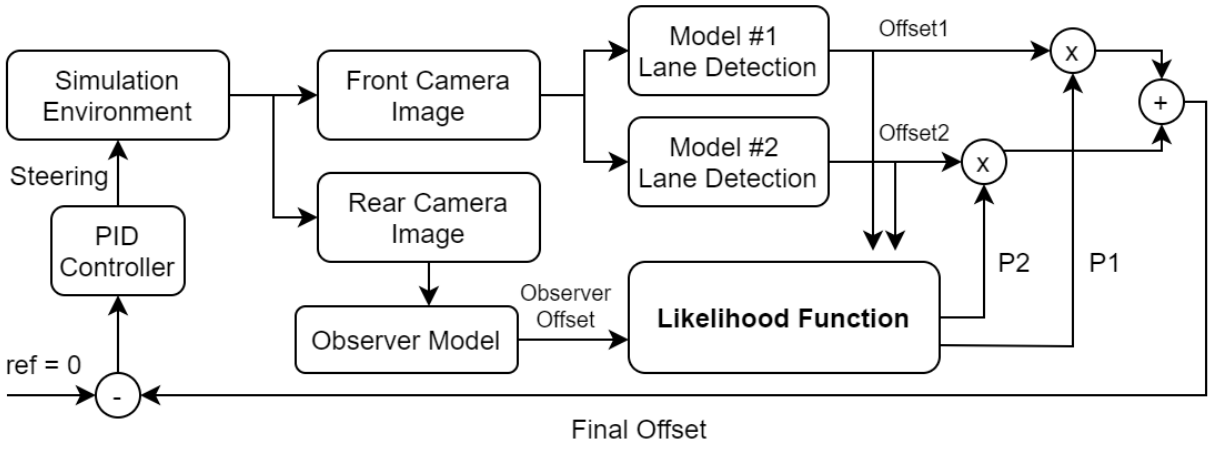


Figure 4.13. The schematic of proposed control system for LKAS

5. SIMULATION RESULTS

In this section first the results of lane detection for single model are presented and the performance of the control system with a single model is evaluated. In this step, model 1 and model 2 evaluated and a sample of their failures is presented as a case study. Then the results of the proposed algorithm is presented and the performance of the system with combination of the models is evaluated for the similar case studies. Finally, effect of some enhancements on the proposed algorithm and the estimator is presented and the final model is validated in 3 more case studies.

5.1 Single Model Performance

Figure 5.1 shows the test for a part of street that has a missing marker and also high contrast shades on the street (Case Study 1). Since the second model has a better performance individually, here the second model is used for single model evaluation.

Figure 5.1. The shade test view in the simulation environment for Case study 1

In the single model shade and missing marker test the vehicle moved out of the current lane in 15 time steps. However, it tried to stay in the left lane after passing the shades and failure sources. Figure 5.2 shows the result of calculated offset by the model and the position of the vehicle comparing to the desired trajectory.

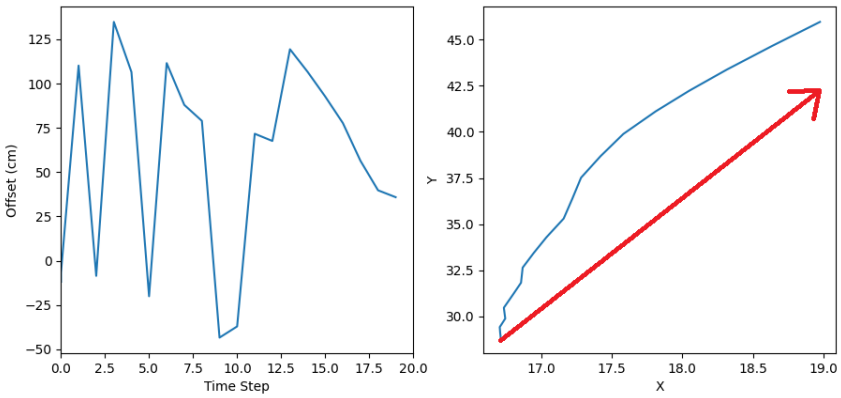


Figure 5.2. The failed results for Case study 1 using model 2. left) The failed offset calculation by model 2. right)blue: vehicle, red: desired trajectory

Figure 5.3 shows the test for a part of simulation environment with a sharp turn (Case Study 2). In the single model turn test, the vehicle moved out of the bounds in the first 10 time steps. Figure 5.4 shows the calculated offset by the second model and the position of the vehicle comparing to the desired trajectory.

Figure 5.3. The turn test view in the simulation environment (Case Study 2)

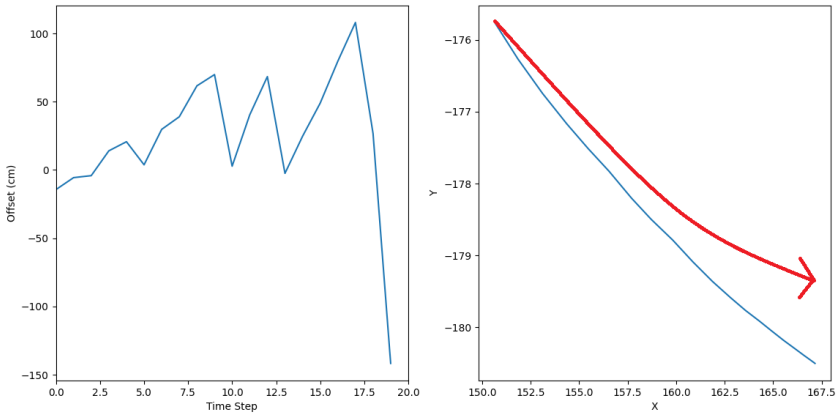


Figure 5.4. The failed turn test results using model 2. left) The failed offset calculation by model 2. right) blue: vehicle, red: desired trajectory

5.2 Multiple Model Adaptive Estimation Performance

In this section, both lane detection models are integrated using MMAE and EKF. The calculated offset by the back camera is used as the observer for evaluating the uncertainty and probability of each model. Figure 5.5 shows the result of MMAE for case study 1. Accordingly, both models has an acceptable performance until time-step 26. However, due to the low uncertainty of model 2 the probability of this model stayed up to 0.9. After time-step 26 some unstable calculations for both models are detected. At step 36 model 2 showed a high uncertainty comparing to the observer result and its probability fell down. In the further steps the probability of models switched multiple times. Note that each time-step is around 0.5 second in the simulation environment.

Figure 5.6 illustrates the MMAE result for another case study. In this case the vehicle is located in a sharp turn and the observer itself is not reliable. Despite the previous case study in this case model 1 had a better performance in most of the steps.

5.3 Improving the Performance of MMAE

The results of both case studies had some issues. The first issue is the fast and multiple transient between models. This issue can be fixed with changing the R value in the EKF.

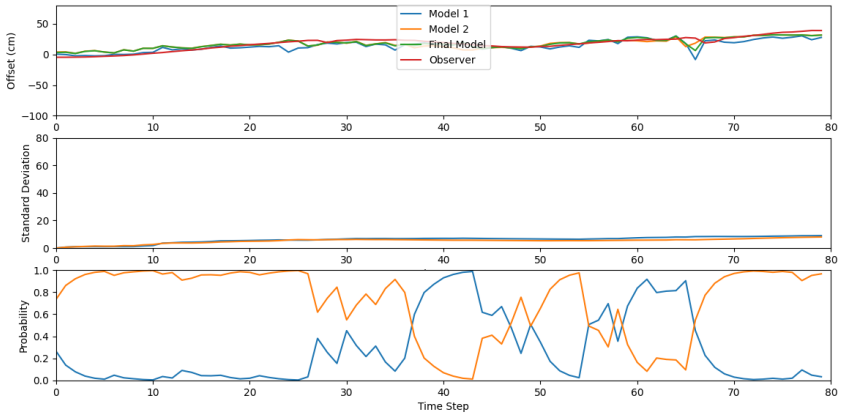


Figure 5.5. The final result of MMAE and the probability of each model at each time-step (Case study 1)

The enhanced result would have a more rectangular waveform shape. The second issue is the failure in the observer performance. Even though there is only few steps which the observer failed but the failure in observer calculation results in unreliable response from MMAE. To solve this issue [1] presented a simple method. Accordingly, the last few values of observer is collected and when the value of offset for observer changes significantly instead of using the measured offset an extrapolation estimated from previous measurements will be used for observing. For this we need to define a jumping threshold. Figure 5.7 shows the result of MMAE for a corrected observer with a jumping threshold of 40 cm for the straight street test. Accordingly, the failures by the observer is normalized by adding the effect of estimation.

Figure 5.8 shows the result of MMAE for a corrected observer with a jumping threshold of 20 cm for the straight street test. Accordingly, this threshold have shown a better performance comparing to the previous one and is going to be chosen as the optimum value.

Now that the MMAE is improved and finalized it is going to be validated in the next section.

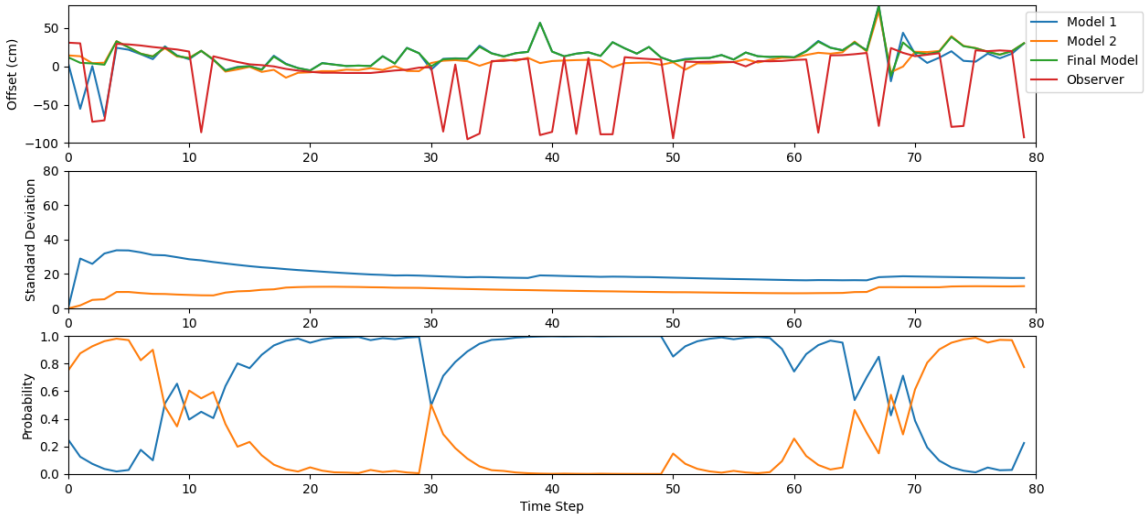


Figure 5.6. The final result of MMAE and the probability of each model at each time-step for Case study 2

5.4 Validation of the Proposed Algorithm

Now that the final MMAE algorithms is completed, it is going to be validated in three more case studies. The first one (Case study 3) is in a mild turn with high contrast shades and also speed bumpers. The controller could keep the vehicle inside the lane. Figure 5.9 shows a view from this test and Figure 5.10 shows the result of MMAE for this case study.

Case study 4 is in a straight street with high contrast shades. The controller could keep the vehicle inside the lane in this test and performed robust. Figure 5.11 shows a view from this test and Figure 5.12 shows the result of MMAE for this case study which had a stable result.

Case study 5 is in a turn with a wet ground and rainy weather. The controller performed robust and the vehicle kept in the lane. Figure 5.13 shows a view from this test and Figure 5.14 shows the result of MMAE for this case study which had a stable result. Accordingly, for this case the observer had failure in time step 5 but it is resolved fast and the vehicle remained in the lane. The video of this test case is available here.

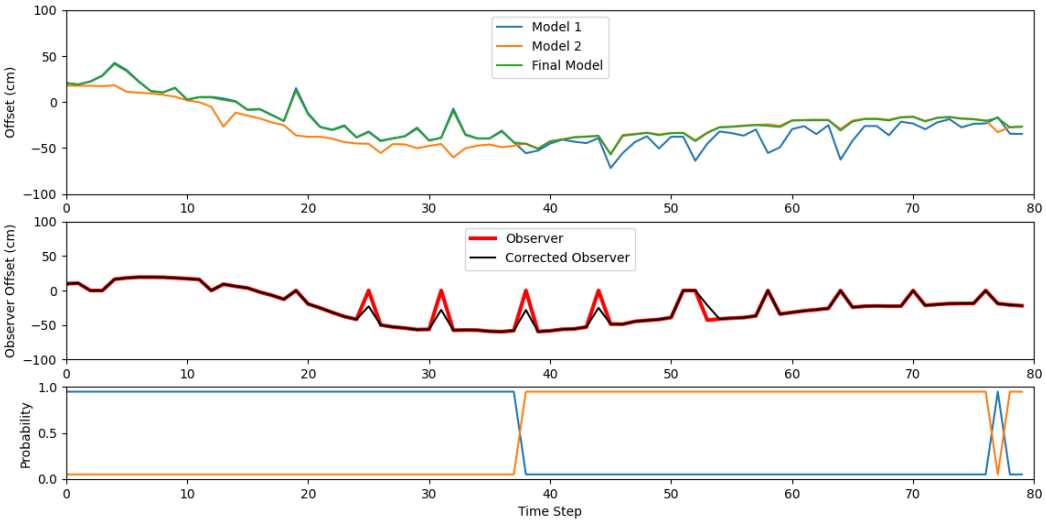


Figure 5.7. The final result of MMAE, the probabilities, and corrected observer with 40 cm threshold for Case study 2

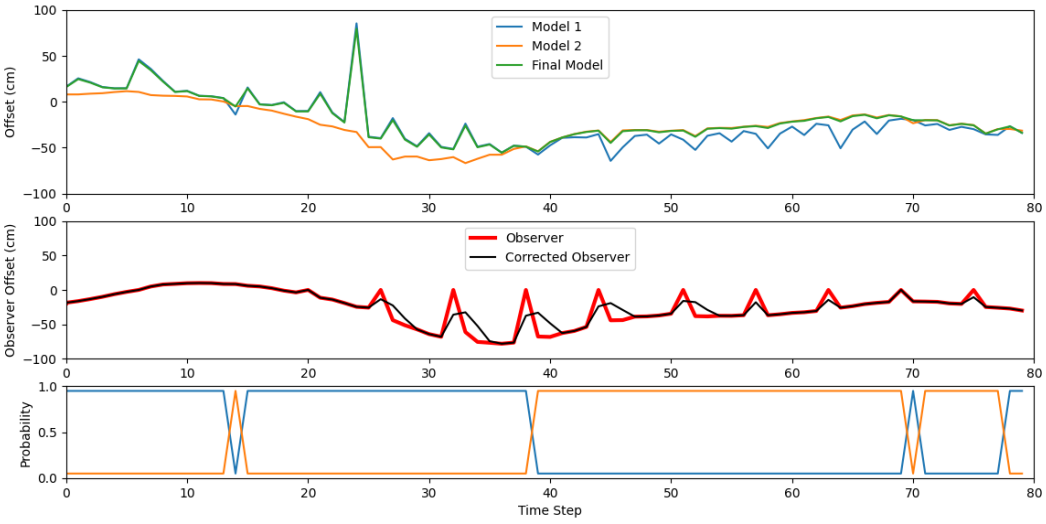


Figure 5.8. The final result of MMAE and the probability and corrected observer with 20 cm threshold for Case study 2

Figure 5.9. The view of Case Study 3 for validation of the proposed algorithm

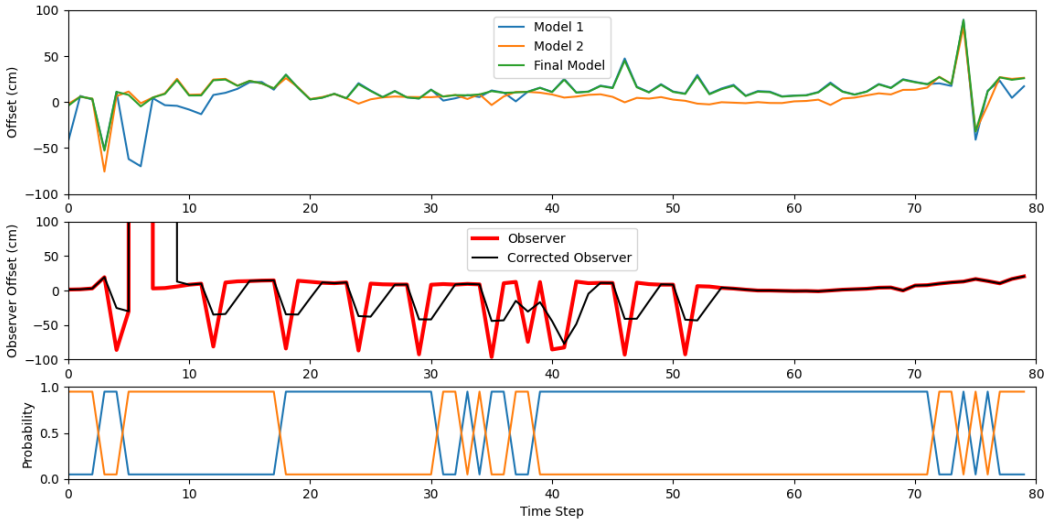


Figure 5.10. The MMAE result for Case Study 3

Figure 5.11. The view of Case Study 4 for validation of the proposed algorithm

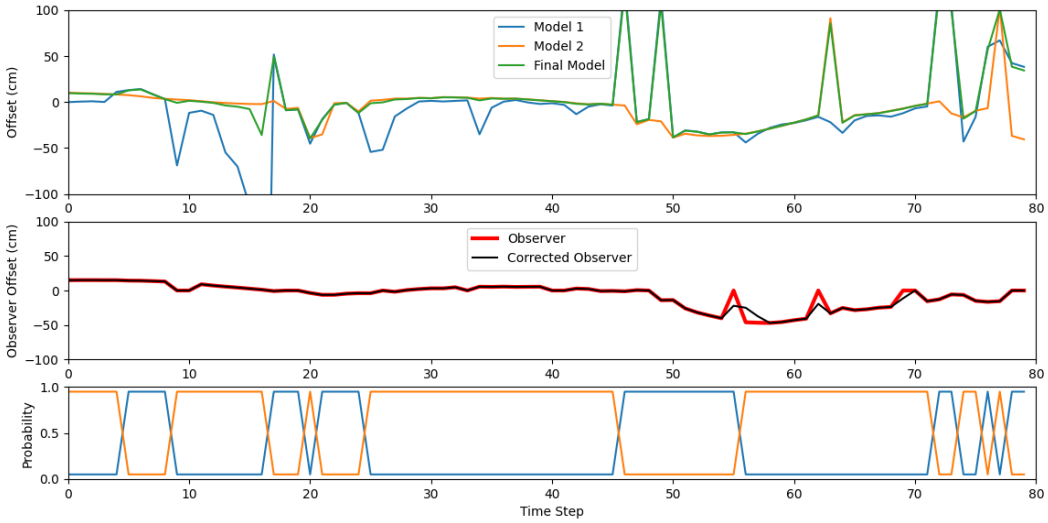


Figure 5.12. The MMAE result for Case Study 4

Figure 5.13. The view of Case Study 5 for validation of the proposed algorithm

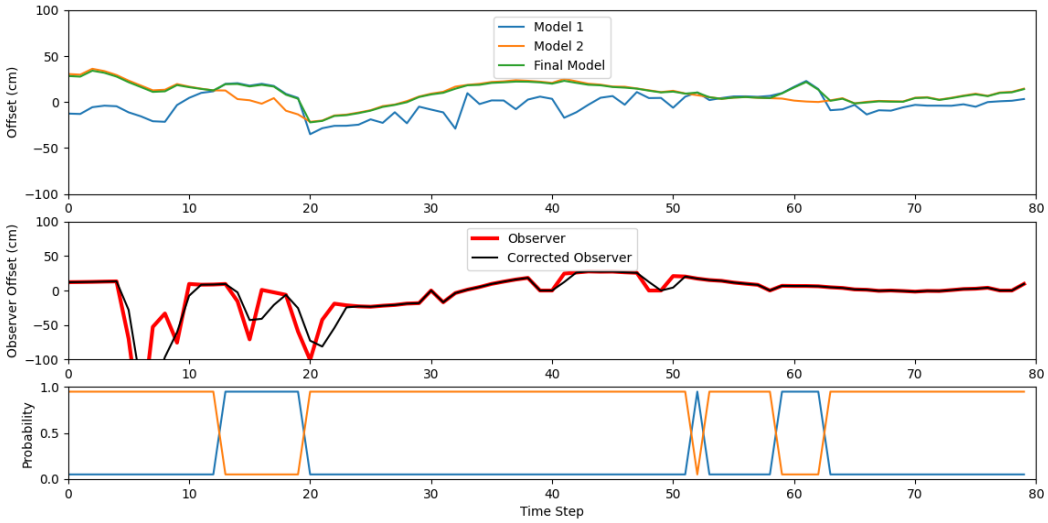


Figure 5.14. The MMAE result for Case Study 5

6. CONCLUSION AND FUTURE WORK

In this section firstly, the an overall description of the study and the conclusion is presented and then some suggestions for future works are presented.

6.1 Conclusion

This research develops a lane-keeping assistant system containing two central units: the lane detection unit and the control unit. This research aimed to enhance the lane detection unit, and for the control unit, a simple PID controller has been used to control the steering. In the lane detection unit, multiple available lane detection algorithms based on computer vision and deep learning are first investigated. The deep learning-based lane detection algorithms did not have a robust performance in the simulation environment since they were trained using front camera data on real-world images with a different view and could not predict the lane markers in the simulation environment. Hence, two computer vision-based algorithms with different pros and cons are considered as lane detection units. These models are investigated in various case studies, and the different sources of failures are detected. In the next step, a third model is used based on the rear camera as the first two models' estimator. The final purpose of all the models was to calculate the offset between the lane and the vehicle's center point. Using the estimator result, the proposed algorithm utilized MMAE to estimate the probability or weight of each model in each step and the final offset value. The final offset value is used as the input error of the PID controller, and the output of the PID is the steering angle used in the simulation environment. The results showed that the proposed algorithm performed robustly and addressed the issues of each of the models by combining the two models, which was able to improve lane detection in sharp turns, lanes with missing markers, wet ground, and streets with high contrast shadows.

6.2 Future Work

In this project, due to the time limitation making a dataset based on AirSim images was not possible. In the other hand, the available trained datasets could not work robust since

they had different perspective, lighting and contrast. Hence, adding another model which is based on a deep learning can enhance the MMAE significantly. In this study the focus was on the MMAE algorithm and a PID is used for controlling the steering. Utilizing an MPC controller would be interesting in a similar project. However, it should be noted that for implementing an MPC controller the output offset of the models is not sufficient. MPC controller requires the input polynomial of the center line at each step. It would be a good idea to use a RNN to have a prediction-based model for detecting the center line in case of failure in the models.

REFERENCES

[1] M. Samuel, M. Mohamad, M. Hussein, and S. M. Saad, "Development of Lane keeping Controller Using Image processing," *International Journal of Computing & Network Technology*, vol. 06, no. 02, pp. 94–97, 2018, ISSN: 2210-1519. DOI: 10.12785/ijcnt/060303.

[2] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine Vision and Applications 2012 25:3*, vol. 25, no. 3, pp. 727–745, Feb. 2012, ISSN: 1432-1769. DOI: 10.1007/S00138-011-0404-2. [Online]. Available: https://link.springer.com/article/10.1007/s00138-011-0404-2.

[3] R. Okuda, Y. Kajiwara, and K. Terashima, "A survey of technical trend of ADAS and autonomous driving," *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test, VLSI-DAT 2014*, 2014. DOI: 10.1109/VLSI-DAT.2014.6834940.

[4] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: A survey," *Machine Vision and Applications*, vol. 25, no. 3, pp. 727–745, 2014, ISSN: 14321769. DOI: 10.1007/s00138-011-0404-2.

[5] H. Zhou and H. Wang, "Vision-based lane detection and tracking for driver assistance systems: A survey," *undefined*, vol. 2018-Janua, pp. 660–665, Jan. 2017. DOI: 10.1109/ICCIS.2017.8274856.

[6] H. Y. Cheng, B. S. Jeng, P. T. Tseng, and K. C. Fan, "Lane detection with moving vehicles in the traffic scenes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 571–582, Dec. 2006, ISSN: 15249050. DOI: 10.1109/TITS.2006.883940. [Online]. Available: https://www.researchgate.net/publication/3427990_Lane_Detection_With_Moving_Vehicles_in_the_Traffic_Scenes.

[7] J. Ruyi, K. Reinhard, V. Tobi, and W. Shigang, "Lane detection and tracking using a new lane model and distance transform," *Machine Vision and Applications 2011 22:4*, vol. 22, no. 4, pp. 721–737, Jan. 2011, ISSN: 1432-1769. DOI: 10.1007/S00138-010-0307-7. [Online]. Available: https://link.springer.com/article/10.1007/s00138-010-0307-7.

[8] G. Zhang, N. Zheng, C. Cui, Y. Yan, and Z. Yuan, "An efficient road detection method in noisy urban environment," *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 556–561, 2009. DOI: 10.1109/IVS.2009.5164338.

[9] M. Aly, "Real time Detection of Lane Markers in Urban Streets," *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 7–12, Nov. 2014. DOI: 10.1109/ivs.2008.4621152. arXiv: 1411.7113. [Online]. Available: https://arxiv.org/abs/1411.7113v1.

[10] J. C. Trivedi, M. Manubhai, J. C. McCall, and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 20–37, Mar. 2006, ISSN: 1524-9050. DOI: 10.1109/TITS.2006.869595. [Online]. Available: https://escholarship.org/uc/item/1bg5f8qd.

[11] A. Borkar, M. Hayes, and M. T. Smith, "Robust lane detection and tracking with Ransac and Kalman filter," *Proceedings - International Conference on Image Processing, ICIP*, pp. 3261–3264, 2009, ISSN: 15224880. DOI: 10.1109/ICIP.2009.5413980.

[12] A. S. Huang, D. Moore, M. Antone, E. Olson, and S. Teller, "Finding multiple lanes in urban road networks with vision and lidar," *Autonomous Robots 2009 26:2*, vol. 26, no. 2, pp. 103–122, Mar. 2009, ISSN: 1573-7527. DOI: 10.1007/S10514-009-9113-3. [Online]. Available: https://link.springer.com/article/10.1007/s10514-009-9113-3.

[13] Z. W. Kim, "Robust lane detection and tracking in challenging scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, Mar. 2008, ISSN: 15249050. DOI: 10.1109/TITS.2007.908582.

[14] J. Pazhayampallil and K. Y. Kuan, "Deep Learning Lane Detection for Autonomous Vehicle Localization," 2013.

[15] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 690–703, Mar. 2017, ISSN: 21622388. DOI: 10.1109/TNNLS.2016.2522428.

[16] M. Kim, J. Seo, M. Lee, and J. Choi, "Vision-Based Uncertainty-Aware Lane Keeping Strategy Using Deep Reinforcement Learning," *Journal of Dynamic Systems, Measurement, and Control*, vol. 143, no. 8, Aug. 2021, ISSN: 0022-0434. DOI: 10.1115/1.4050396. [Online]. Available: http://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/143/8/084503/6685908/ds_143_08_084503.pdf.

[17] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 5575–5585, Mar. 2017, ISSN: 10495258. arXiv: 1703.04977. [Online]. Available: https://arxiv.org/abs/1703.04977v2.

[18] F. Hamilton, T. Berry, and T. Sauer, "Ensemble Kalman filtering without a model," *Physical Review X*, vol. 6, no. 1, pp. 1–12, 2016, ISSN: 21603308. DOI: 10.1103/PhysRevX.6.011021.

[19] M. J. Quinlan and R. H. Middleton, "Multiple model Kalman filters: A localization technique for RoboCup soccer," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5949 LNAI, pp. 276–287, 2010, ISSN: 03029743. DOI: 10.1007/978-3-642-11876-0_24.

[20] C. Barrios, H. Himberg, Y. Motai, and A. Sadek, "Multiple model framework of adaptive extended Kalman filtering for predicting vehicle location," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, no. October, pp. 1053–1059, 2006. DOI: 10.1109/itsc.2006.1707361.

[21] C. Y. Kuo, Y. R. Lu, and S. M. Yang, "On the image sensor processing for lane detection and control in vehicle lane keeping systems," *Sensors (Switzerland)*, vol. 19, no. 7, 2019, ISSN: 14248220. DOI: 10.3390/s19071665.

[22] S. Kamat, "Lane Keeping of Vehicle Using Model Predictive Control," *2019 IEEE 5th International Conference for Convergence in Technology, I2CT 2019*, Mar. 2019. DOI: 10.1109/I2CT45611.2019.9033958.

[23] M. Bujarbaruah, X. Zhang, H. E. Tseng, and F. Borrelli, "Adaptive MPC for Autonomous Lane Keeping," Jun. 2018. arXiv: 1806.04335. [Online]. Available: https://arxiv.org/abs/1806.04335v2.

A. PYTHON CODE

In this section the implemented code is attached. However, the source code of this project is also avaialbe in this GitHub repo. The main code is inside RandomMoving.py which has the implementation of software in loop and controller system. Then in lane.py and lane2.py the offset for the models are implemented. In the edge_detection.py the utlitiy for edge detection is presented. All the codes are annotated with comments and they are based on Advanced Lane Detection and Automatic Addison Lane Detection projects.

A.1 RandomMoving.py

```
import os
import airsim
import time
import numpy as np
from numpy.core.fromnumeric import size
from scipy.interpolate import InterpolatedUnivariateSpline

import re
import cv2
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
import lane
import lane2
import math
import adv_lane

def std(data, ddof=0):
    n = len(data)
    mean = sum(data) / n
    return math.sqrt(sum((x - mean) ** 2 for x in data) / (n - ddof))
```

```
# connect to the AirSim simulator
client = airsim.CarClient()
client.confirmConnection()
client.enableApiControl(True)
car_controls = airsim.CarControls()
i = 0
Kp = 0.001 #0.002 for curvy roads -- 0.001 for straight lane
Ki = 0
Kd = 0.001
offset = []
offset2 = []
main_offs = []
t = []
timestep = []
error = []
error_dot = []
error_int = []
controlledSteering = []
var = []
var2 = []
var_obs = []
prob = []
prob2 = []
observer = []
cor_obs = []

end_bool = True
while end_bool:
```

```
client.simPause(False)
# get state of the car
car_state = client.getCarState()
#print("Speed %d, Gear %d" % (car_state.speed, car_state.gear))

# set the controls for car
#client.simPause(False)
car_controls.throttle = 0.5
if(i==0):
    car_controls.steering = 0
else:
    car_controls.steering = controlledSteering[i-1]

client.setCarControls(car_controls)

# let car drive a bit
time.sleep(0.5)
#client.simContinueForTime(1)

client.simPause(True)
responses = client.simGetImages([airsim.ImageRequest("CAM0", airsim.
    ImageType.Scene, False, False),airsim.ImageRequest("CAM1", airsim.
    ImageType.Scene, False, False)])
#back_resps = client.simGetImages([airsim.ImageRequest("CAM1",
    airsim.ImageType.Scene, False, False)])
#back_data = back_resps[0]

# processing the image to find lanes and offset
offset.append(0)
```

```
offset2.append(0)
main_offs.append(0)
t.append(0)
timestep.append(0)
error.append(0)
error_dot.append(0)
error_int.append(0)
controlledSteering.append(0)
var.append(0)
var2.append(0)
var_obs.append(0)
prob.append(0)
prob2.append(0)
observer.append(0)

#responses = client.simGetImages([airsim.ImageRequest("0"
, airsim.ImageType.DepthPlanner, True)])
for response in responses:
    img1d = np.fromstring(response.image_data_uint8, dtype=np.uint8)
    #back1d = np.fromstring(back_data.image_data_uint8, dtype=np.
        uint8)

    # reshape array to 4 channel image array H X W X 4
    img_rgb = img1d.reshape(response.height, response.width, 3)
    #back_rgb = back1d.reshape(back_data.height, back_data.width, 3)

    # saving the image in the storage
    fileName = time.strftime()
    dir_path = os.path.dirname(os.path.realpath(__file__))
```

```
    airsim.write_png(dir_path + '\\Images\\' + fileName + '.png',
       img_rgb)
    plt.imshow(img_rgb)
    plt.show()



    try:
        if(response.camera_name=='CAM0'): #Front Camera
            offset2[i] = lane.main(img_rgb) #offset (cm-not scaled)
            offset[i] = lane2.main(img_rgb)
            #offset[i] = adv_lane.lane_finding_pipeline(img_rgb)
        elif(response.camera_name=='CAM1'): #Back Camera
            observer[i] = -lane2.main(img_rgb)
    except:
        print("error in image processing")

# Probability Calc
if (i>=1):
     r1[i] = abs(var1[i-1]-abs(offset[i])**2)+epsilon #residuals
        r2[i] = abs(var2[i-1]-abs(offset2[i])**2)+epsilon


        s1=P1+R
        beta1=(((2*math.pi)**1)*(s1))**-0.5
        s2=P2+R
        beta2=(((2*math.pi)**1)*(s2))**-0.5
        #prob[i] = c2/(c1+c2)
        #prob2[i] = c1/(c1+c2)
        prob1[i]=((beta1*(math.exp(-0.5*(r1[i]))*s1**-1*(r1[i])))*
           prob1[i-1])/((beta1*(math.exp(-0.5*(r1[i]))*s1**-1*(r1[i]))
```

```
            )*prob1[i-1]+(beta2*(math.exp(-0.5*(r2[i]))*s2**-1*(r2[i]))
            )*prob2[i-1])
        prob2[i]=((beta2*(math.exp(-0.5*(r2[i]))*s2**-1*(r2[i])))*
            prob2[i-1])/((beta2*(math.exp(-0.5*(r2[i]))*s2**-1*(r2[i]))
            )*prob2[i-1]+(beta1*(math.exp(-0.5*(r1[i]))*s1**-1*(r1[i]))
            )*prob1[i-1])



else:
    prob[i] = 0.5
    prob2[i] = 0.5

main_offs[i] = prob[i]*offset[i] + prob2[i]*offset2[i]

#derivative and integral calculations:
t[i] = round(time.time()*1000) #time in millisecond
dt = 1



if(i>1):
    error[i] = main_offs[i]
    error_dot[i] = (main_offs[i]-main_offs[i-1])/dt
    error_int[i] = error_int[i-1]+(error[i]+error[i-1])/2*dt
    timestep[i] = timestep[i-1] + dt
    var[i] = std(offset)
    var2[i] = std(offset2)
    var_obs = std(observer)
elif(i==1):
```

```
        error[i] = main_offs[i]
        error_dot[i] = (main_offs[i]-main_offs[i-1])/dt
        error_int[i] = error_int[i-1]+(error[i]+error[i-1])/2*dt
    elif(i==0):
        error[i] = main_offs[i]
        error_dot[i] = 0
        error_int[i] = 0

    controlledSteering[i] = -(Kp*error[i]+Kd*error_dot[i]+Ki*error_int[i
        ])
    print(error[i],error_dot[i],error_int[i])
    print(controlledSteering[i])
    print()

    if (i==80):
        end_bool = False
    i += 1
# Plotting the results
client.simPause(True)

# here we are creating sub plots
figure, axes = plt.subplots(3)
line1 = axes[0].plot(timestep, offset, label = "Model 1")
line2 = axes[0].plot(timestep, offset2, label = "Model 2")
line3 = axes[0].plot(timestep, main_offs, label = "Final Model")
line04 = axes[0].plot(timestep, observer, label = "Observer")
line05 = axes[0].plot(timestep, observer, label = "Corrected Observer")
axes[0].legend(bbox_to_anchor=(0.5, 1.05), loc="upper center")
axes[0].set(ylabel='Offset (cm)')
```

```
axes[0].set_xlim([0, 80])
axes[0].set_ylim([-100, 80])
axes[0].set(size=[10,2])

line4 = axes[1].plot(timestep, var, label = "Model 1")
line5 = axes[1].plot(timestep, var2, label = "Model 2")
#plt.title("Offset from center line", fontsize=20)
plt.xlabel("Time Step")
plt.ylabel("Standard Deviation")
axes[1].set(ylabel='Standard Deviation', xlabel="Time Step")
axes[1].set_xlim([0, 80])
axes[1].set_ylim([0, 80])

line6 = axes[2].plot(timestep, prob, label = "Model 1")
line7 = axes[2].plot(timestep, prob2, label = "Model 2")
axes[2].set(ylabel='Probability', xlabel="Time Step")
axes[2].set_xlim([0, 80])
axes[2].set_ylim([0, 1])

plt.show()
```