SIMULATION OF SELF-DRIVING VEHICLE AND LANE TRACKING FOR ROAD TRANSPORT USING COMPUTER VISION

CO8811 - PROJECT REPORT

Submitted by

JEFFERSON CLINTON B	**211420118029**
VIGNESH S	**211420118057**
YUGENDHARAN V	**211420118061**

in partial fulfillment for the award the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER AND COMMUNICATION ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report **"SIMULATION OF SELF-DRIVING VEHICLE AND LANE TRACKING FOR ROAD TRANSPORT USING COMPUTER VISION"** is the bonafide work of **JEFFERSON CLINTON B (211420118029), VIGNESH S (211420118057) AND YUGENDHARAN V (211420118061)** who carried out the project work under my supervision.

SIGNATURE
Dr.B.ANNI PRINCY M.E., Ph.D.,
HEAD OF THE DEPARTMENT
PROFESSOR
COMPUTER AND COMMUNICATION ENGINEERING,

PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI- 600123.

SIGNATURE
Mrs. J.BRINDHA M.E., (Ph.D.).,
SUPERVISOR
ASSISTANT PROFESSOR,
ELECTRONICS AND COMMUNICATION ENGINEERING,
PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI,
POONAMALLEE,
CHENNAI- 600123.

Certified that the above candidate(s) was/ were examined in the End Semester Project Viva-Voce Examination held on..........................

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors Tmt. **C.VIJAYARAJESWARI, Dr. C. SAKTHIKUMAR, M.E., Ph.D**., and **Dr.SARANYASREE SAKTHIKUMAR B.E., M.B.A., Ph.D**., for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.,** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CCE Department, **Dr. B. ANNI PRINCY, M.E.,Ph.D**., Professor, for the support extended throughout the project.

We would like to thank Our supervisor **Mrs. J.BRINDHA, M.E., (Ph.D.).,** Assistant Professor and all the faculty members of the Department of ECE for their advice and suggestions for the successful completion of the project.

JEFFERSON CLINTON B
VIGNESH S
YUGENDHARAN V

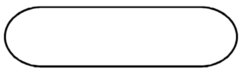
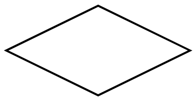
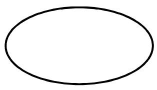
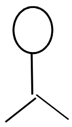
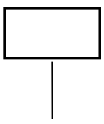
TABLE OF CONTENTS

ABSTRACT

Effective simulation and testing environment is a vital part in the research of self-driving vehicles. It is capable of testing self-driving software quickly in a variety of virtual environments at low cost. However, currently in mainstream simulation platforms, a considerable gap exists between the constructed virtual environment and the actual self-driving platform, which decreases the efficiency of development and makes it difficult. The goal of developing autonomous vehicle is to provide convenient and safe intelligent transportation service for human beings. In recent decades, governments are actively promoting the popularization and application of self-driving technologies. We use the Open CV (Computer Vision) Technique in the autonomous driving system, which is a real-time computer vision method. The algorithm that we use is Manhattan Algorithm and it is used for calculating the distance between two real valued vectors as dot product. Manhattan distance is calculated as the sum of the absolute difference between the two vectors. By this method the accuracy of traction can be increased from 65% to 97.5%.

LIST OF FIGURES

LIST OF SYMBOLS

S.NO.	SYMBOL NAME	NOTATION	DESCRIPTION
1	Initial Activity		The starting point or first activity of flow
2	Final Activity		The end of the activity diagram is shown by a Bull's eye symbol
3	Activity		Represented by a rectangle with rounded edges
4	Decision		A logic where a decision is to be made
5	Use Case		Describe the interaction between the user and a system
6	Actor		A role that a user plays with respect system
7	Object		A real time entity
8	Message		To send message between the life of an Object

LIST OF ABBREVIATIONS

S. NO.	ABBREVIATION	EXPANSION
1	CV	Computer Vision
2	CNN	Convolutional Neural Network
3	HOG	Histogram Object Gradient
4	HTTP	Hyper Text Transfer Protocol
5	HLS	HTTP Live Streaming
6	GPL	General Public License
7	IDLE	Integrated Development And Learning Environment
8	GUI	Graphical User Interface
9	PERL	Practical Extraction And Reporting Language
10	PHP	Hypertext Preprocessor
11	WWW	World Wide Web

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The aim of our project is to avoid number of crashes on our road. It constantly monitor the surrounding better and make informed decision. It prevents human error from happening as the system controls the vehicles. It avoids distraction and interruption. It reduces the chances of accident dramatically.

1.2 OVERVIEW

Computer vision is a field of study focused on the problem of helping computers to see. At an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world.

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Machines can accurately identify and locate objects then react to what they **"see"** using digital images from cameras, videos, and deep learning models.

As computer vision evolved, programming algorithms were created to solve individual challenges. Machines became better at doing the job of vision recognition with repetition. Over the years, there has been a huge improvement of deep learning techniques and technology.

We now have the ability to program supercomputers to train themselves, self-improve over time and provide capabilities to businesses as online applications.

Images are broken down into pixels, which are considered to be the elements of the picture or the smallest unit of information that make up the picture. Computer vision is not just about converting a picture into pixels and then trying to make sense of what's in the picture through those pixels. Neural networks and Deep Learning are Making Computer Vision More Capable of Replicating Human Vision.

Neural networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

1.2.1 COMPUTER VISION AND IMAGE PROCESSING

Computer vision is distinct from image processing. Image processing is the process of creating a new image from an existing image, typically simplifying or enhancing the content in some way. It is a type of digital signal processing and is not concerned with understanding the content of an image.

1.2.2 IMAGE CLASSIFICATION AND SEGMENTATION

A simple explanation of the classification of a picture is when a computer classifies an image in a certain category. In the picture below the classification of the first object would be sheep. The localization or location is identified by the box surrounding the object in the picture. Object detection detects instances of semantic objects of a certain class. Every pixel belongs to a particular class. Pixels within the class are represented by the same color. This describes semantic segmentation.

1.2.2.1 OBJECT CLASSIFICATION

The system parses visual content and classifies the object on a photo/video to the defined category.

1.2.2.2 OBJECT IDENTIFICATION

The system parses visual content and identifies a particular object on a photo/video.

1.2.2.3 OBJECT TRACKING

The system processes video finds the object (or objects) that match search criteria and track its movement.

1.3 PROBLEM STATEMENT

Every year, the number of road accidents keeps increasing around the world. To avoid road accidents, we could adopt the autonomous driving system which concentrates on autonomous driving as well as assisting people who have been in a car accident using a Computer Vision based approach. This computer vision algorithm needs to be trained with the required set of objects. As a result, when those objects are found, it correctly displays the image's situation. So that we can prevent accidents involving pedestrians and other vehicles.

Every year, the number of road accidents keeps increasing around the world. To avoid road accidents, we could adopt the autonomous driving system which concentrates on autonomous driving as well as assisting people who have been in a car accident using a Computer Vision based approach. This computer vision algorithm needs to be trained with the required set of objects. As a result, when those

objects are found, it correctly displays the image's situation. So that we can prevent accidents involving pedestrians and other vehicles.

To differentiate the path, Gabor channels with surface direction were used. They used a flexible delicate democratic approach that evaluated a vanishing point and divided the surface direction's certainty rank. A methodology based on vanishing point estimation has been presented to assess the presentation of path location for organized and unstructured streets. However, because of the difference in area, the evaporating point cannot be accurately calculated, which is one of the major drawbacks of these strategies. Currently, a computer vision-based procedure has been submitted that can proficiently differentiate paths in any surrounding condition. For path recognition, we have primarily used angle and HLS thresholding. For appropriate mapping, point of view change has been applied after thresholding.

CHAPTER 2

LITERTURE SURVEY

PAPER 1 : **Lane Detection in Autonomous Vehicles: A Systematic Review**

AUTHORS : **Noor Jannah Zakaria, Mohd Ibrahim Shapiai, Rasli Abd Ghani, Mohd Najib Mohd Yassin, Mohd Zamri Ibrahim, Nurbaiti Wahid**

YEAR : **2023**

ABSTRACT: This paper aims to present a review of recent as well as classic image registration methods. Image registration is the process of overlaying images (two or more) of the same scene taken at different times, from different viewpoints, and/or by different sensors. Problematic issues of image registration and outlook for the future research are discussed too. The registration geometrically align two images (the reference and sensed images). The reviewed approaches are classified according to their nature (area-based and feature-based) and according to four basic steps of image registration procedure feature detection, feature matching, mapping function design, and image transformation and resampling. Problematic issues of image registration and outlook for the future research are discussed too. The major goal of the paper is to provide a comprehensive reference source for the researchers involved in image registration, regardless of particular application areas.

LIMITATIONS

Elastic registration is in situations when image deformations are much localized.

PAPER 2 : **Detection of Lane and Speed Breaker Warning System for Autonomous Vehicles using Machine Learning Algorithm**

AUTHORS : **Fua. P and Lepetit Heltin. V, Genitha .C, Rajaji. P, Rahul. S**

YEAR : **2022**

ABSTRACT : In this paper, we introduce a local image descriptor that is inspired by earlier detectors such as SIFT and GLOH but can be computed much more efficiently for dense wide-baseline matching purposes. We will show that it retains their robustness to perspective distortion and light changes, can be made to handle occlusions correctly, and runs fast on large images. Our descriptor yields better wide-baseline performance than the commonly used correlation windows, which are hard to tune. Too small, they do not bring enough information. Therefore, recent methods tend to favor small correlation windows, or even individual pixel differencing and rely on global optimization techniques such as graph-cuts to enforce spatial consistency. They are restricted to very textured or high-resolution images, of which they typically need more than three. Our descriptor overcomes these limitations and is robust to rotation, perspective, scale, illumination changes, blur and sampling errors. We will show that it produces dense wide baseline reconstruction results that are comparable to the best current techniques using fewer lower-resolution images.

LIMITATIONS

The fusion of the image is performed at different stages of image matching.

PAPER 3 : **Machine Learning for Security in Vehicular Networks: A Comprehensive Survey Determination**

AUTHORS : **Duraiswami Anum Talpur. R, Mohan Gurusamy and Samet. H**

YEAR : **2022**

ABSTRACT: The problem of pose estimation arises in many areas of computer vision, including object recognition, object tracking, site inspection and updating, and autonomous navigation using scene models. We present a new algorithm, called Soft POSIT, for determining the poseof a 3D object from a single 2D image in the case that correspondences between model points and image points are unknown. The algorithm combines Gold's iterative Soft Assign algorithm for computing correspondences and DeMenthon's iterative POSIT algorithm for computing object pose under a full-perspective camera model. Our algorithm, unlike most previous algorithms for this problem. The performance of the algorithm is extensively evaluated in Monte Carlo simulations on synthetic data under a variety of levels of clutter, occlusion, and image noise. The algorithm is being applied to the practical problem of autonomous vehicle navigation in a city through registration of a 3D architectural models of buildings to images obtained from an on- board camera.

LIMITATIONS

It is an unusual classifier. (i.e.) Performance of classifier is depend on the type of dataset.

PAPER 4 : Robust Real-Time Object Detection using integral image

AUTHORS : Plabon Kumar Saha, Sinthia Ahmed, Tajbiul Ahmed, Hasidul Islam, Al Imran, Tahmidul Kabir A. Z. M , Al Mamun Mizans

YEAR : 2022

ABSTRACT: Frame work that is capable of processing images extremely rapidly while achieving high detection rates. The system yields face detection performance comparable to the best previous systems. Implemented on a conventional desktop, face detection proceeds at 15 frames per second. There are three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly. The second is a learning algorithm, based on Ada Boost, which selects a small number of critical visual features and yields extremely efficient classifiers. The third contribution is a method for combining classifiers in a “cascade” which blows background regions of the image to be quickly discarded while spending more computation on promising object-like regions. A set of experiments in the domain of face detection are presented. The system yields face detection performance comparable to the best previous systems. Implemented on a conventional desktop, face detection proceeds at 15 frames per second.

LIMITATIONS

It spending more computation on promising object-like regions.

PAPER 5 : **A Method of Multiple Lane Detection Based on Constraints of Lane Information**

AUTHORS : **Long Yang Ma, Hao Zhu, Hong Duan**

YEAR : **2021**

ABSTRACT : While feature point recognition is a key component of modern approaches to object detection, existing approaches require computationally expensive patch preprocessing to handle perspective distortion. In this paper, we show that formulating the problem in a Naive Bayesian classification framework makes such preprocessing unnecessary and produces an algorithm that is simple, efficient, and robust. Furthermore, it scales well to handle large number of classes. To recognize the patches surrounding key points, our classifier uses hundreds of simple binary features and models class posterior probabilities. To recognize the patches surrounding key points, our classifier uses hundreds of simple binary features and models class posterior probabilities. To recognize the patches surrounding key points, our classifier uses hundreds of simple binary features and models class posterior probabilities. Even though this is not strictly true, we demonstrate that our classifier nevertheless performs remarkably well on image datasets containing very significant perspective changes.

LIMITATIONS

It require computationally expensive patch preprocessing to handle perspective distortion.

PAPER 6 : Vision-Based Line Following Robot Using CNN

AUTHORS : Alfian Ma'arif, Aninditya Anggari Nuryono, Iswanto

YEAR : 2020

ABSTRACT : Object detection is important in car sharing services. Accuracy, efficiency, and low memory consumption are desirable for object detection in car sharing services. This paper presentsa network system that satisfies all these requirements. Our approach first divides the object detection task into multiple simpler local regression tasks. Then, we propose the generalized Haar filter-based convolutional neural network to reduce the consumption of memory and computing resource. To achieve real-time performance, we introduce a sparse window generation strategy to reduce the number of input image patches without sacrificing accuracy. To achieve real-time performance, we introduce a sparse window generation strategy to reduce the number of input image patches without sacrificing accuracy. We perform experiments on both vehicle and pedestrian data sets. Experimental results demonstrate that our approach can accurately detect objects under challenging conditions.

LIMITATIONS

CNN do not encode the position and orientation of object.

PAPER 7 : Lane Determination of Vehicles Based on a Novel Clustering Algorithm for Intelligent Traffic Monitoring

AUTHORS : Lin Cao, Tao Wang, Dongfeng Wang, Kangning Du, Yunxiao Liu, Chong Fu

YEAR : 2020

ABSTRACT: Autonomous vehicle (AV) is regarded as the ultimate solution to future automotive engineering; however, safety still remains the key challenge for the development and commercialization of the AVs. Therefore, a comprehensive understanding of the development status of AVs and reported accidents is becoming urgent. In this article, the levels of automation are reviewed according to the role of the automated system in the autonomous driving process, which will affect the frequency of the disengagements and accidents when driving in autonomous modes. Additionally, the public on-road AV accident reports are statistically analyzed. These safety risks identified during on-road testing, represented by disengagements and actual accidents, indicate that the passive accidents which are caused by other road users are the majority. The capability of AVs to alert and avoid safety risks caused by the other parties and to make safe decisions to prevent possible fatal accidents would significantly improve the safety of AVs.

LIMITATIONS

Firewalls can be difficult to configure correctly. Makes the system slower than before.

PAPER 8 : **Comprehensive and Practical Vision System for Self-Driving Vehicle Lane-Level Localization**

AUTHORS : **Kaifeng Gao, Bowen Wang, Xinxin Du, Kok Kiong Tan**

YEAR : **2016**

ABSTRACT: Autonomous driving is the main application of Internet of Things (IoT) technology in the field of intelligent transportation. In autonomous driving, self-driving cars will avoid changing lanes in a short distance. When theself-driving car executes the follow- up instruction, the road smoothness in front of the car will affect the driving safety and comfort of the car. The incomplete road point cloud data need be imputated to avoid potential misjudgments of the road conditions. Currently, little research work specifically focuses on imputating the incomplete road point cloud data that are caused by obstacle vehicles. In this paper, we propose a fast method to imputate the incomplete road point cloud data using a Graphics Processing Unit (GPU)-based parallel Inverse Distance Weighted (IDW) interpolation algorithm to enhance the safety of autonomous driving. To evaluate the performance of the proposed method, two groups of experiments are conducted.

LIMITATIONS

It does not have a potential difference in practical applications.

PAPER 9 : A Computer vision system for detection and avoidance for automotive vehicles

AUTHORS : Yuan-Ying Wang and Hung-Yu Wei

YEAR : 2016

ABSTRACT: Safe driving is a relatively new concept that focuses on solving the responsibility attribution problem for autonomous vehicles (AVs), claiming that once the AVs follow a series of pre-defined safe driving policies, it is free from liability even when accidents happen. In this work, we propose safe driving policies for traffic configurations, including straight road, intersection, and Manhattan-like city. Base on the defined safe driving policies, we offer the concepts of safe driving capacity (SDC) and safe driving throughput (SDT) to measure the safe efficiency of the traffic configurations. The former measures the maximum AVs a traffic configuration could accommodate, and the latter measures the maximum throughputs of safe AVs of a traffic configuration. The values obtained are the fundamental limits of the traffic efficiencies for liability-free AVs under the defined conditions. The theoretical performance bounds give people insight son the potential limitations of the safe traffic efficiencies. Finally, this work provides analytical results of SDC and SDT on all the traffic configurations mentioned with explanations, implications, and trade-offs on the issues that may have effects on them.

LIMITATIONS

Technology that Uses Moving Cars as Nodes in a Network to create a Mobile Network.

PAPER 10 : **Lane Detection System Based on Canny Method for Driving Assistance**

AUTHORS : **Nur Uddin, Hendi Hermawan, Fredy Jhon Philip Sitorus, Ida Nurhaida**

YEAR : **2023**

ABSTRACT: This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. With the rapid developments of wireless communication and increasing number of connected vehicles, Vehicular AdHoc Networks (VANETs) enable cyber interactions in the physical transportation system. Future networks require real-time control capability to support delay-sensitive application such as connected autonomous vehicles. In recent years, fog computing becomes an emerging technology to deal with the insufficiency in traditional cloud computing. In this paper, a fog-based distributed network control design is proposed toward connected and automated vehicle application. The proposed architecture combines a case study of connected cruise control (CCC) is introduced to demonstrate the efficiency of the proposed architecture and control design. Finally, we discuss some future research directions and open issues to be addressed.

LIMITATIONS

It is explained practically but presently it’s not implemented.

PAPER 11 : **A Vision-based driver assistance system using collaborative edge computing**

AUTHORS : **Arghavan Keivani, Farzad Ghayoor, Jules-Raymond Tapamo**

YEAR : **2017**

AUTHORS Reliable Detection and Recognition of planar objects Including traffic sign, street sign and road surface in dynamic cluttered Natural scenes are a big challenge for self-driving cars. In this paper, we propose a comprehensive method for planar object detection and recognition. First, the data association of LIDAR and camera is set up to acquire colorized laser scan, which simultaneously contain both color and geometrical information. Second, the combine three color spaces of RGB, HSV and CIE L*a*b* with laser reflectivity as an aggregation- based feature vector. Third, the 3-D geometrical characteristics of planar objects that contain planarity, size and aspect ratio are exploited to further reduce false alarm. Fourth, in order to increase robustness to any viewpoint variation, we present a new virtual camera-based rectification method to synthesize front-parallel views of refined object descriptors in 3-D space. Overall, the detection rate of our comprehensive method has up to 95.87% and the recognition rate even reaches 95.07% for traffic signs ranging within 100 m.

LIMITATIONS

Both cars required to slow down and negotiate the junction. As a result, the average delay would be greater the channeled intersection.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

CNN based object detection model failed to reveal clear conclusions due to its limited dataset size, insufficiently repeated experiments, and outdated baseline selection. The existing systems can be divided into proposal-based methods, such as CNN and R-CNN model in the object detection. Since many parameters needs to be learned, so training of CNN model is heavily dependent on computational power. In CNN model the accuracy of object detection is less in the range of value between 60-75%.

LIMITATIONS

The problem is that training of CNN model takes too much time and a large computational resources to get the expected output.

3.2 PROPOSED SYSTEM

In Autonomous driving system, we are using Open CV (Computer Vision) Technique that is used in real-time Computer Vision. It is used to process images, videos and even live streams. Open CV is a cross-platform library using which we can develop real time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection. So Computer should able to recognize the object such as face of human being, lamppost or even the statue that deals with how computers can gain high level understanding from digital images or videos. Autonomous driving is the most representative technology that can benefit people by providing driving environment information. The computer read any images such as range of value between 0-255.

3.3 SYSTEM ARCHITECTURE

System architecture is the conceptual mode that defines the structural behavior and more of a system description with a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

The architecture has three central components: monitoring, learning, detecting. In the first component, the camera that is fixed in the front of the vehicle monitors the surroundings thus checking the enclosing objects like vehicles , pedestrians and also monitors the lane in which the vehicle travels . The second component learns whether the vehicle moves in the destined path and learns the distance between the neighboring objects, thus maintaining a safe distance between them. The third component, detects the other objects moving along or opposite of the vehicle, thus avoiding unnecessary collisions. Lane detection system helps drivers to keep their vehicle in lane and helps in road collisions. This systems generate audio visual alerts if the vehicle starts deviating from its lane. It use small camera which is mounted near the rear. It triggers the alarm to react it. Vehicle detection system detects the vehicle in front of one owns by using computer vision technology. It also estimates the road curvature and car position from the centre point of vehicle detection. Pedestrian detection system is used to identify pedestrians in front view of the vehicle/car and provide warning to the driver avoiding fatalities. It help of computer vision or machine vision techniques because of the high performance algorithms as well as reduced cost of hardware.

The PDS should detect pedestrian ahead of the vehicle in various lighting conditions and should handle occlusions effectively. PDS uses object detector (in this case pedestrian) which recognizes pedestrians with the help of various feature sets such as pedestrian edges, color, shape etc.

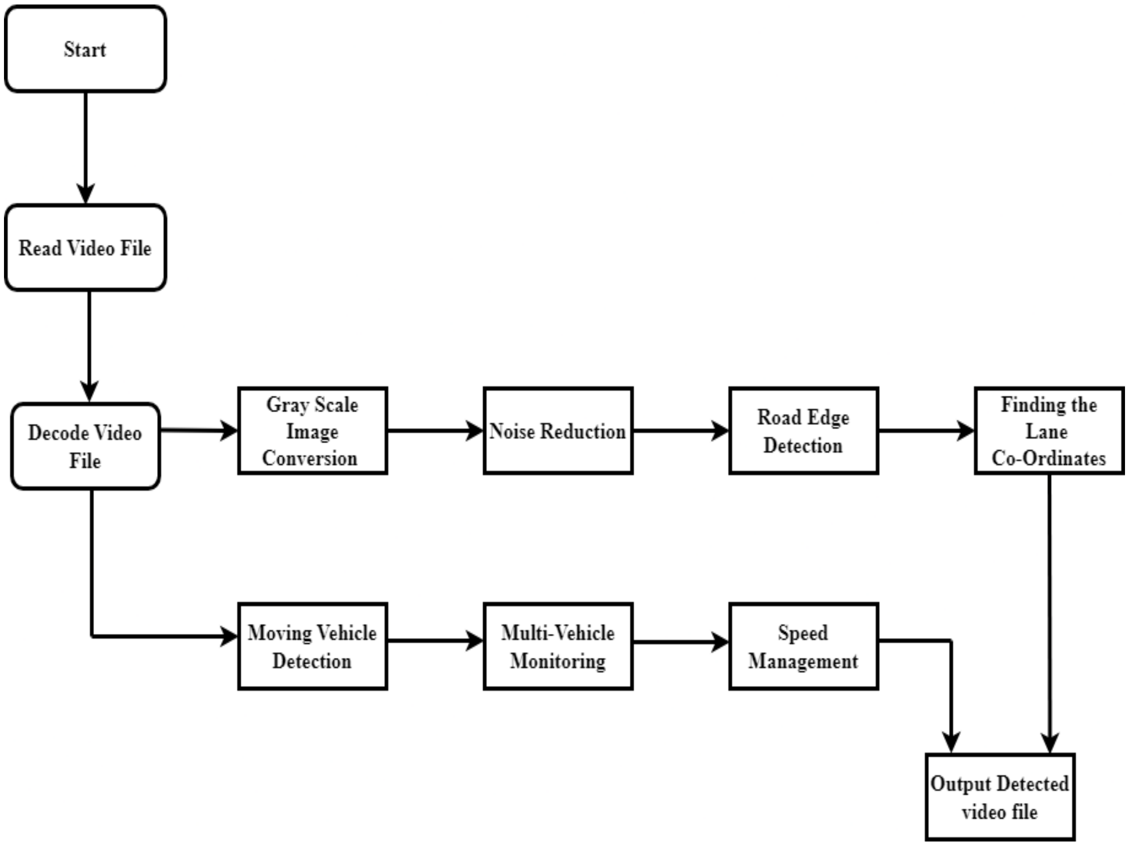


Fig 3.1 Architecture Diagram

CHAPTER 4

SYSTEM SPECIFICATION

4.1 SYSTEM SPECIFICATION

The requirements specification is a technical specification of requirements for the software products. It is the first step in the requirements analysis process it lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

4.1.1 HARDWARE ENVIRONMENT

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the systems do and not how it should be implemented.

- Processor - Intel i5 9^{th} Gen / M1 Chip / AMD Ryzen 5 and above
- Speed - 4 GHz
- RAM - 8 GB
- SSD - 500 GB

4.1.2 SOFTWARE ENVIRONMENT

The software requirements are the specification of the system. It should include both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide a basis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the teams and tracking the team's progress throughout the development activity.

- Operating System - Windows / Ubuntu
- Coding Language - Python
- Front End - Jupiter Notebook
- Back End - Anaconda 3.6

4.2 PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python Source code is also available under the GNU General Public License (GPL).This tutorial gives enough understanding on Python programming language. Python is a high- level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

4.2.1 PYTHON IDLE

IDLE is Python's Integrated Development and Learning Environment. IDLE has the following features: It is coded in 100% pure

Python, using the GUI tool kit. The cross-platform works mostly the same on Windows, UNIX, and mac OS. Python shell window (interactive interpreter) with colorizing of code input, output, and error messages. It contains multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features. It can be search within any window, replace within editor windows, and search through multiple files (grep). The debugger with persistent breakpoints, stepping, and viewing of global and local name spaces. It has configuration, browsers, and other dialogs.

4.2.1.1 CHARACTERISTICS OF PYTHON

It supports functional and structured programming methods as well as OOP. It can be used as a scripting language or can be compiled to byte-code for building large applications. It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic garbage collection. It can be easily integrated with C, C++, COM, Active X, CORBA, and Java.

4.2.1.2 APPLICATIONS OF PYTHON

Easy-to-learn: Python has few keywords, simple structure and a clearly defined syntax. This allows the student to pick up the language easily.

Easy-to-read: Python code is more clearly defined and visible to the eyes.

Easy-to-maintain: Python's source code is fairly Easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross-compatible on UNIX, Windows and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low-level modules to the python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Database: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI application that can be created and ported to many system calls, libraries and windows system, such as Windows MFC, Macintosh, and the Window system of UNIX.

Scalable: Python provides a better structure and support for large programs than shell scripting.

4.3 PACKAGE

4.3.1 NUMPY

NumPy is a Python package which stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object, provide tools for integrating C, C++ etc. It is also useful in linear algebra, random number capability etc. NumPy array can also be used as an efficient multi-dimensional container for generic data. NumPy array is a powerful N-dimensional array object which is in the form of rows and columns. We can initialize NumPy arrays from nested Python lists and access it elements. NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. We are using NumPy to process the multi-dimensional images which is in multiple videos and gives the results efficiently.

4.3.2 SCIPY

SciPy, a scientific library for Python is an open source, BSD-licensed library for mathematics, science and engineering. The SciPy library depends on NumPy, which provide convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays. It provides manyuser-friendly and efficient numerical practices such as routines for numerical integration and optimization. SciPy provides high-level commands and classes for data-manipulation and data-visualization, which increases the power of an interactive Python session by significant order. In addition to mathematical algorithms in SciPy, everything from classes and web and database subroutines to parallel programming is available to Python programmer, making it easier and faster to develop sophisticated and specialized applications. Since SciPy is open source, developers across the world can contribute to the development of additional modules which is much beneficial for scientific applications using SciPy.

4.3.3 OPEN CV

Python is a library of Python bindings designed to solve computer vision problems. cv2. In read () method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.

4.3.4 MATPLOTLIB

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays it provides an object-oriented API for embedding plots into applications using general-purpose GUI tool kits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine, designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

CHAPTER 5

SYSTEM DESIGN

5.1 OVERVIEW

System design is the process of defining the architecture components modules, interfaces and data for the system to satisfy specified requirements; system design could be seen as the application of system theory to product development.

5.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a diagrammatic representation (Fig 5.1, Fig 5.2, Fig 5.3) of the information (data) flow within a system. DFD illustrates how data is processed by a system in terms of inputs and outputs. Data flow diagrams are generally used as a system modelling tool and for structured system analysis and design. To represent a program or a software system, data flow diagram is implemented in the software designing phase.

5.2.1 DFD Level 0

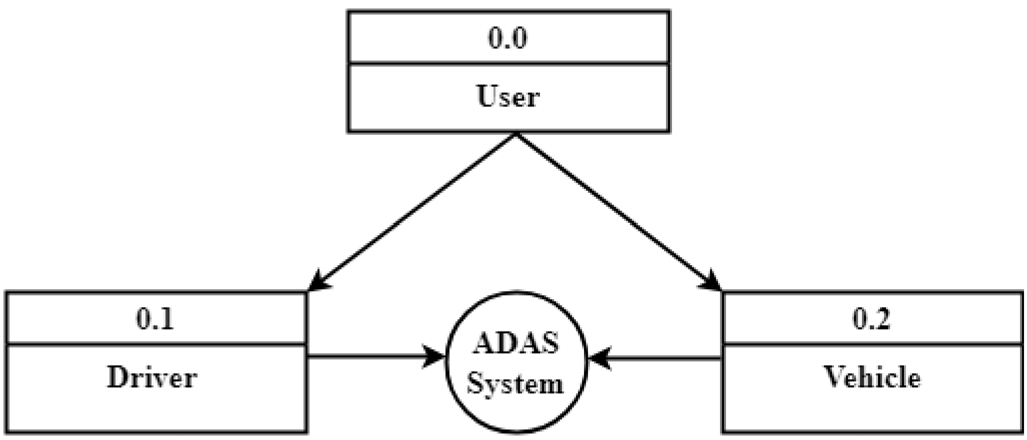


Fig 5.1 DFD Level 0

5.2.2 DFD Level 1

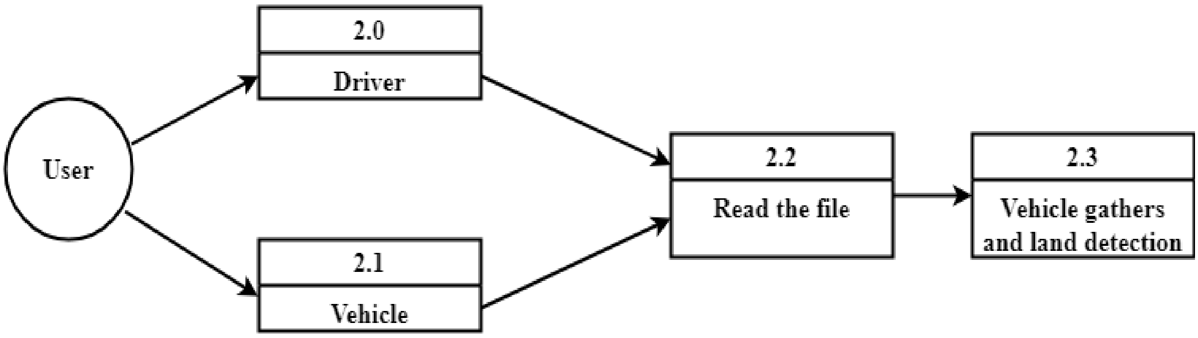


Fig 5.2 DFD Level 1

5.2.3 DFD Level 2

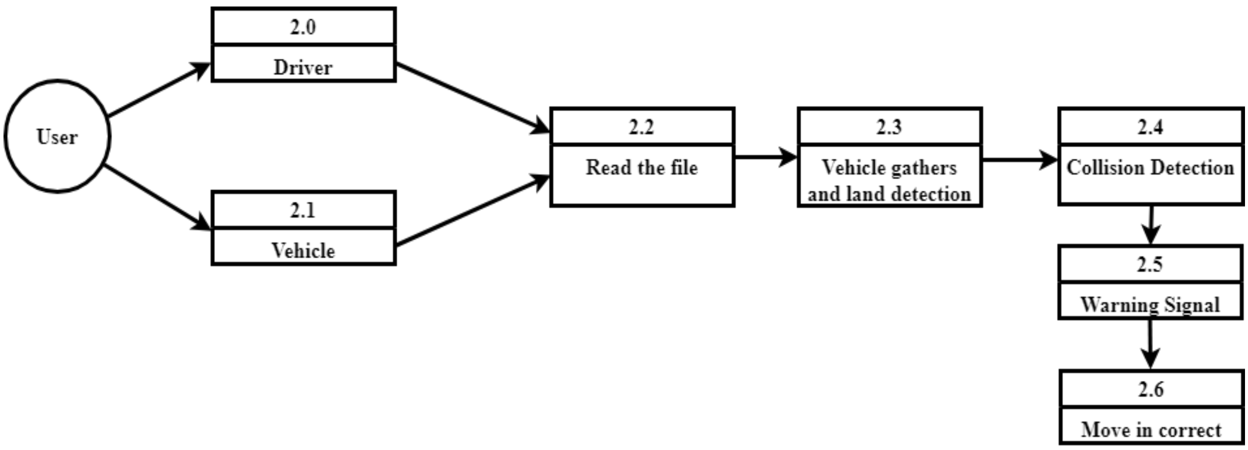


Fig 5.3 DFD Level 2

5.3 UML DIAGRAMS

The unified modelling language (UML) is a general purpose modelling language in field of software engineering. The basic level provides a set of graphical notation techniques to create visual models of object-oriented software- intensive systems.

5.3.1 USE CASE DIAGRAM

A use case diagram is a representation of user's interaction with the system and depicting the specifications of a use case (Fig 5.4). This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

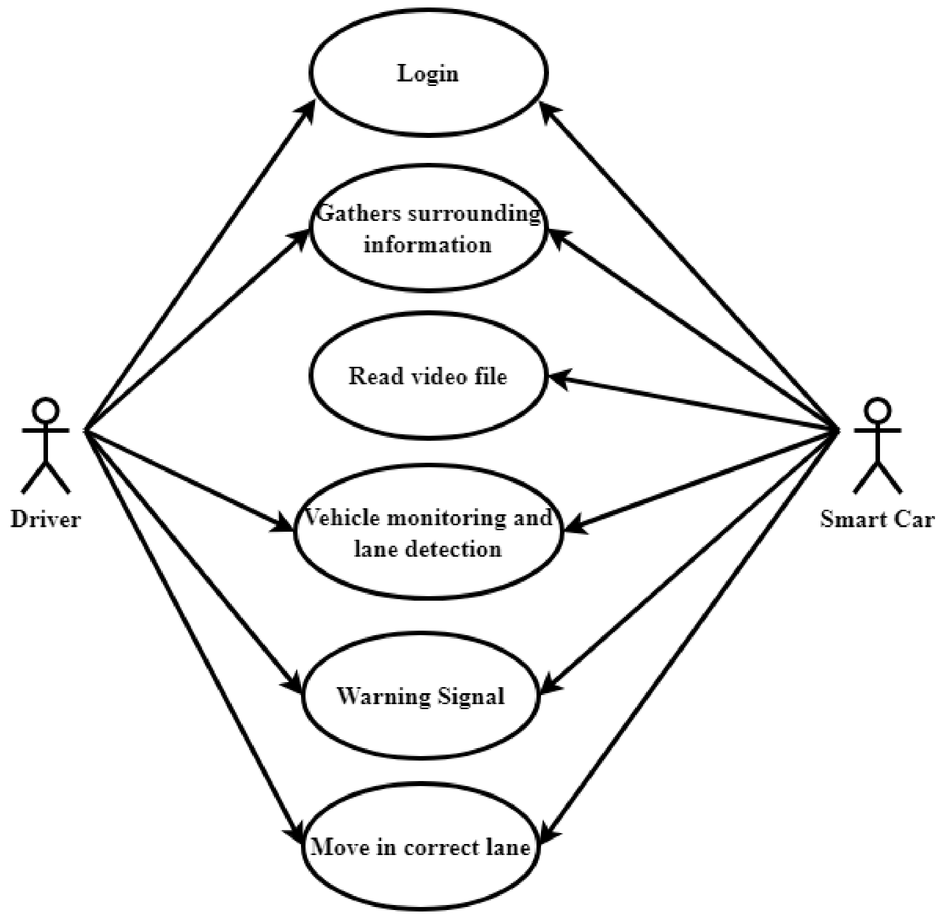


Fig 5.4 Use Case Diagram

5.3.2 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the object and classes involved in the scenario and the sequence of message exchanged between the object needed to carry out the functionality of the scenario (Fig 5.5). Sequence diagram are typically associated with use case realizations in the logical view of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

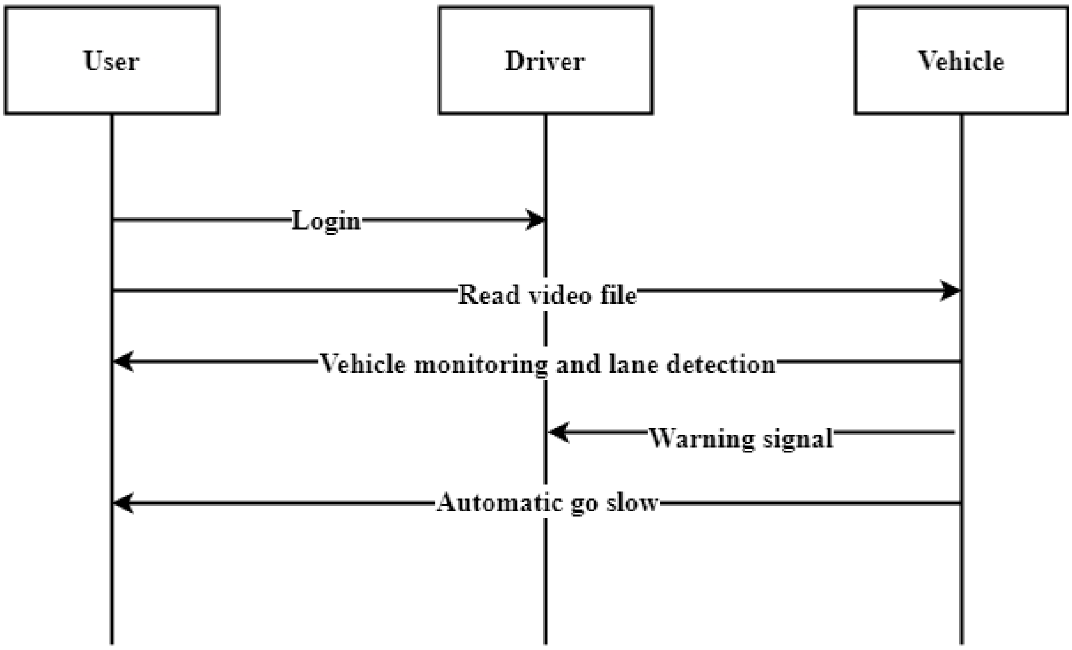


Fig 5.5 Sequence Diagram

5.3.3 ACTIVITY DIAGRAM

Activity diagram is a graphical representation (Fig 5.6) of work flows of stepwise activities and action with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

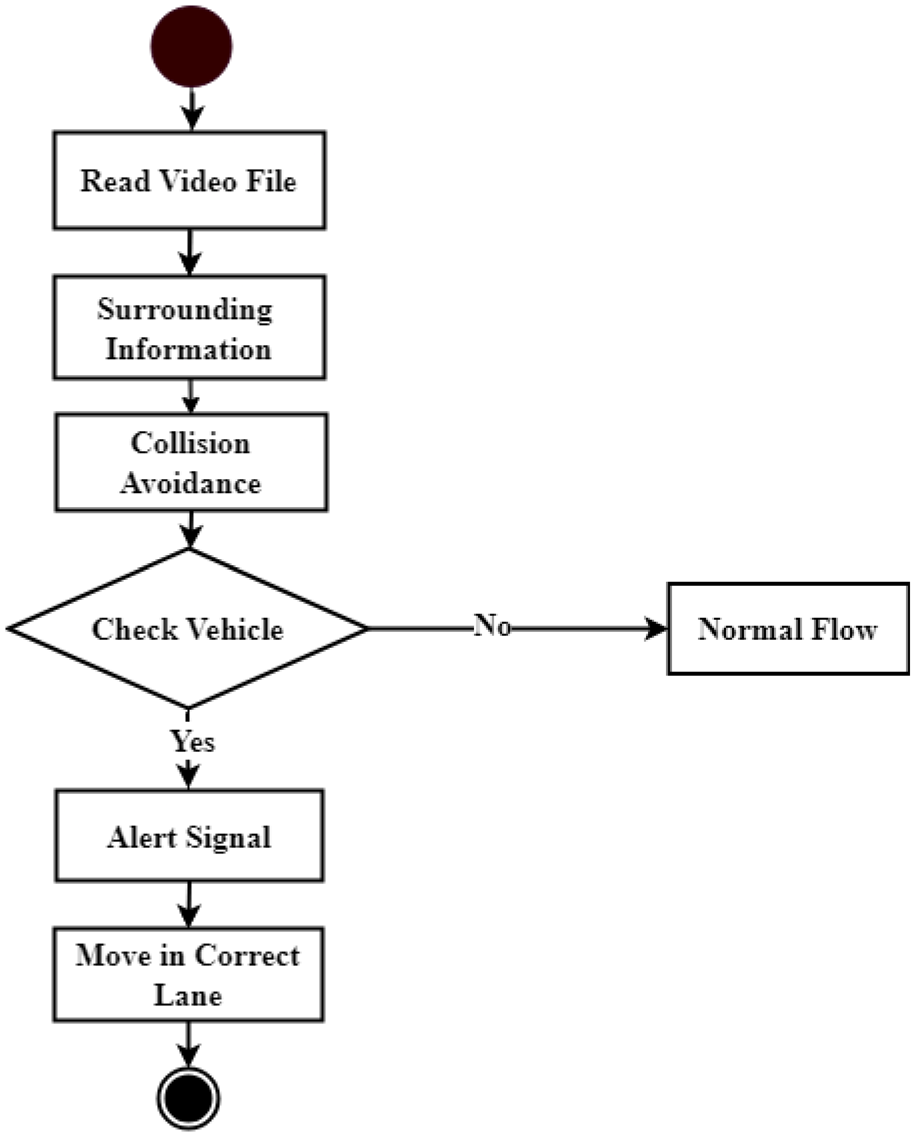


Fig 5.6 Activity Diagram

CHAPTER 6

MODULE DESCRIPTION

6.1 MODULES

6.1.1 LANE DETECTION SYSTEM

It helps drivers to keep their vehicle in lane and helps in avoiding road collisions. This systems generate audio visual alerts if the vehicle starts deviating from its lane. It use small camera which is mounted near the rear. It triggers the alarm to react it. The images from the camera are decoded into videos. The software requires less pixel, so Grey scale is used to reduce the RGB colors into two colors i.e., white and black. Thus reducing the time taken and increases the speed. Then noise reduction is used to get the clear image of the road especially for night journey, thus refining the images .This also helps in travelling along the destined path. It constantly checks whether the vehicle travels within the white lines.

6.1.2 VEHICLE DETECTION SYSTEM

It detect the vehicle in front of one owns by using computer vision technology. The cost of an optical sensor (such as CMOS and CCD ones) is much lower than that of an active sensor (such as laser or radar ones).It also estimates the road curvature and car position from the center point of vehicle detection. The vehicles moving along and opposite are continuously monitored and the distance between the vehicles are calculated such that the vehicle maintains a safer distance. The speed is managed thus it gives alert signals thus avoiding road collisions.

6.1.3 PEDESTRIAN DETECTION SYSTEM

It is used to identify pedestrians in front view of the vehicle/car and provide warning to the driver avoiding fatalities. It help of computer vision or machine vision techniques because of the high performance algorithms as well as reduced cost of hardware. The PDS should detect pedestrian ahead of the vehicle in various lighting conditions and should handle occlusions effectively. PDS uses object detector (in this case pedestrian) which recognizes pedestrians with the help of various feature sets such as pedestrian edges, color, shape etc.

6.2 MANHATTAN ALGORITHM

The Manhattan distance, also known as the taxicab distance or city block distance, is a metric used to measure the distance between two points in a grid-like environment, such as a city with a rectangular grid of streets. It is calculated as the sum of the absolute differences of their respective Cartesian coordinates. This distance measure is often used in applications where movement is restricted to horizontal and vertical directions, such as in integrated circuits, path planning, and data analysis. It is also commonly used in k-nearest neighbors algorithms as an alternative to the Euclidean distance.

FORMULA

ManhattanDistance = sum for i to N sum | v_1[i] – v_2[i] |

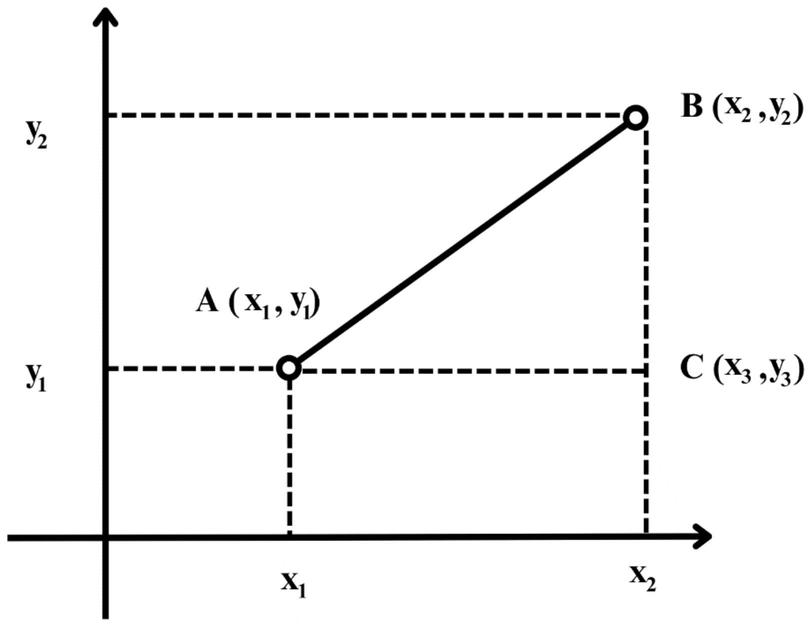


Fig 6.1 Manhattan Distance Graph

The Manhattan distance between point A, located at coordinates (x_1, y_1), and point B, located at coordinates (x_2, y_2), is (x_3, y_3) units (Fig.6.1). This indicates that you would need to travel a total of 8 units, either horizontally or vertically, to move from point A to point B in the city grid.

6.3 MANHATTAN ACCURACY

The Classified input data as coordinates of lane and edge of the road which is applied in the Manhattan algorithm. For accuracy, it is expressed as a percentage and is calculated by dividing the number of correctly classified input coordinate data by the total number of coordinate dataset, then multiplying the result by 100.

Accuracy = (Total number of coordinates dataset / Number of correctly classified input coordinates data) ×100%

CHAPTER 7

IMPLEMENTATION

7.1 OPEN CV (Computer Vision)

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. Image processing is mainly focused on processing raw input images to enhance them or preparing them to do other tasks. Computer vision is focused on extracting information from the input images and videos to have a proper understanding of them to predict the visual input like human brain. In less than a decade, today's systems have reached 99 percent accuracy from 50 percent making them more accurate than human at quickly reacting to visual inputs. Computer vision is the science of computers and software systems that can recognize and understand images and scenes. Object detection refers to the capability of computer and software systems to locate object is an image/scene and identify each object. Out of all the technologies used in Autonomous driving system, vision and image processing is predominant method used by Autonomous vehicle provides for understanding the on-road environment, detection of object, and taking corrective driving decisions.

7.1.1 WORKING

You install the library on your computer. You start writing your code that will make use of the many features in Open CV. You build your code and run it to perform the task you described. Open CV is a computer vision library with APIs that let you set up a pipeline for your computer vision project. Open CV is a free open source library used in real-time image processing. It’s used to process images, videos, and even live streams Open CV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focus on image processing, video capture and analysis including features like face detection and object detection.

SIMULATION PLATFORM	**TECHNIQUES**	**WORK**	**ALGORITHM**	**ACCURACY**
CARLA	EXISTING WORK	NEURAL NETWORK	CNN	70%
AIRSIM	EXISTING WORK	NEURAL NETWORK	CNN	73%
PRESCAN	EXISTING WORK	V2X COMM.	ADAS	86%
SLVD	PROPOSED WORK	OPEN CV	MANHATTAN	97.5%

TABLE 7.1 COMPARISION WITH EXISTING PLATFORMS

CHAPTER 8

CODING AND TESTING

8.1 CODING

Once the design aspect of the system is finalizes the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screwed into the system.

8.2 CODING STANDARDS

Coding standards are guidelines to programming that focuses on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

8.2.1 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be self-descriptive. One should even get the meaning and scope of the variable by its name. The conventions are adopted for easy understanding of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

8.2.2 CLASS NAMES

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

8.2.3 MEMBER FUNCTION AND DATA MEMBER NAME

Member function and data member name begins with a lower case letter with each subsequent letters of the new words in upper case and the rest of letters in lower case.

8.2.4 SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

8.3 TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements ad user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

8.4 TYPES OF TESTING

8.4.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component label and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specification and contains clearly inputs and expected results.

8.4.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

8.4.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs are exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

8.4.4 SYSTEM TESTING

System testing ensures that the entire integrated software system meets requirements. It tests is configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows emphasizing pre-driven process links and integration points.

8.4.5 WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner working, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

8.4.6 BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box.

8.4.7 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

8.5 TESTING STRATEGY AND APPROACH

8.5.1 TEST OBJECTIVES

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

8.5.2 FEATURES TO BE USED

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

CHAPTER 9

EXPERIMENTS AND RESULT

9.1 IMAGE DETECTION

In input image consists of the color intensity of different color channels, i.e.(Fig. 9.1) the intensity and color information are mixed in RGB color space but in HSV color space the color and intensity information are separated from each other. This makes HSV color space more robust to lighting changes.

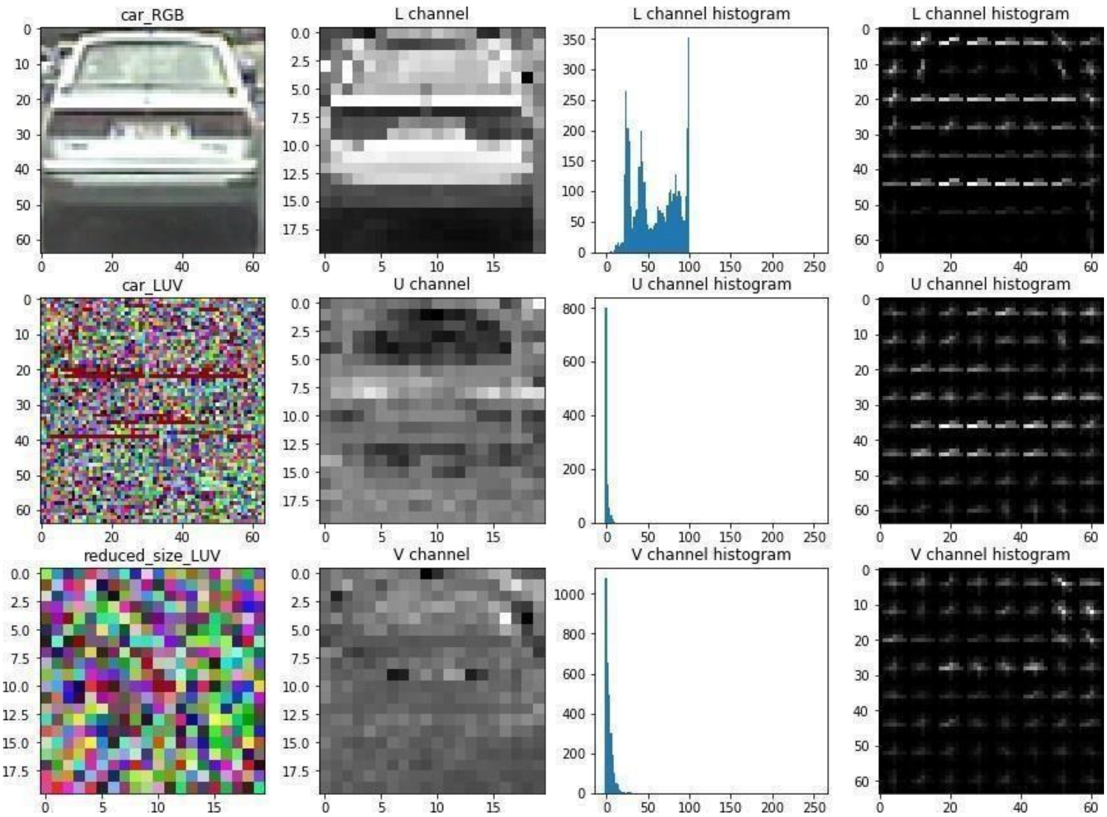


Fig. 9.1 Image Detection

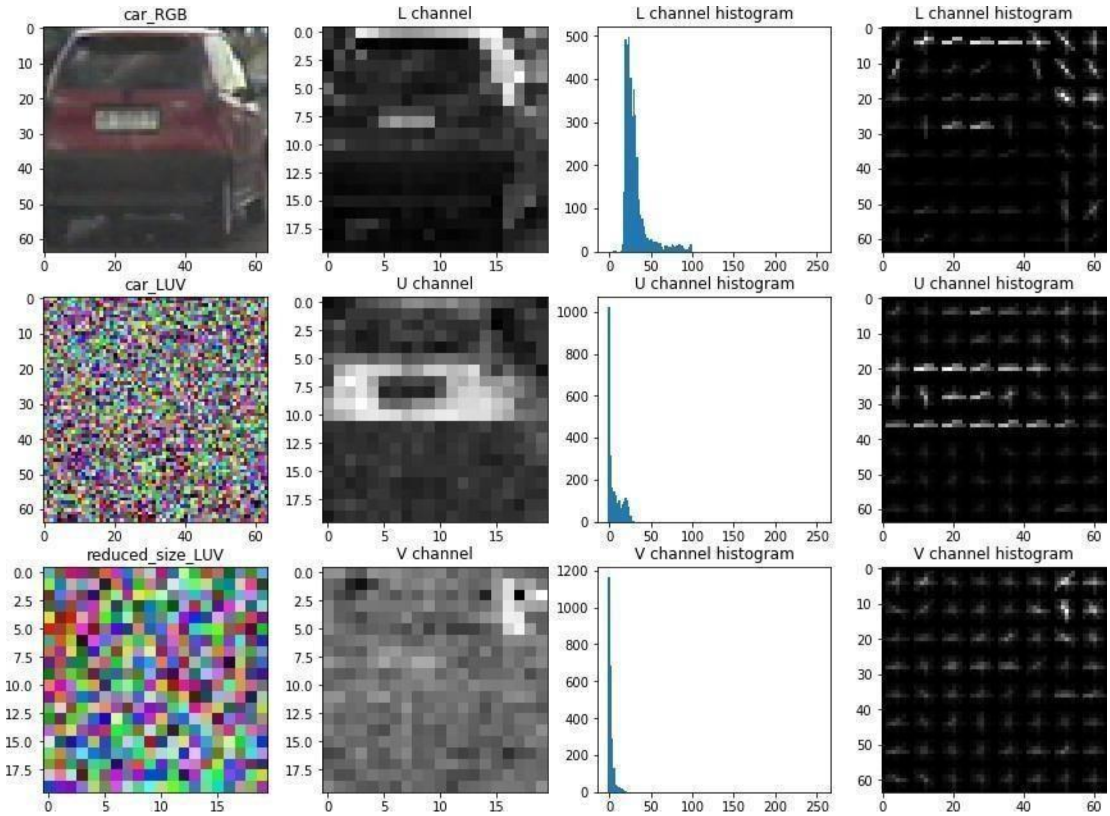


Fig. 9.2 Grey Scale Image

It takes the test input image ,suppose if the object is moving in lane to get the heat map produce .so it will get the object shapes in lane then ,it convert into grey scale conversion(Fig 9.2) Black and White (0 to 255).

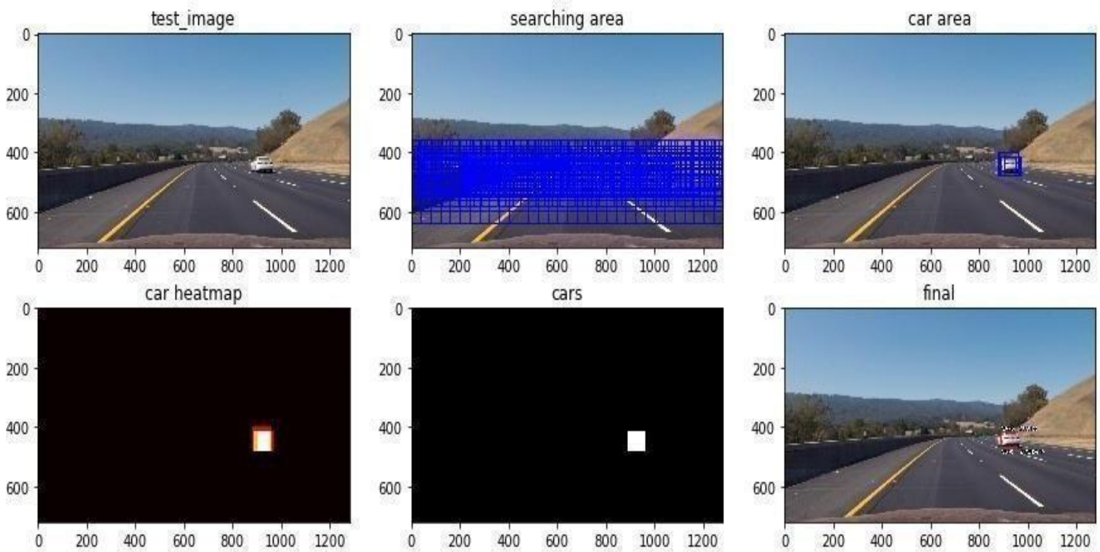


Fig.9.3 Road Curvature

Then apply on bounding boxes, it gives the prediction about bounding boxes (Fig 9.3 and Fig 9.4) along the probabilities and confidence accuracy gives us insight whether the box contain an object or not (in or case car).

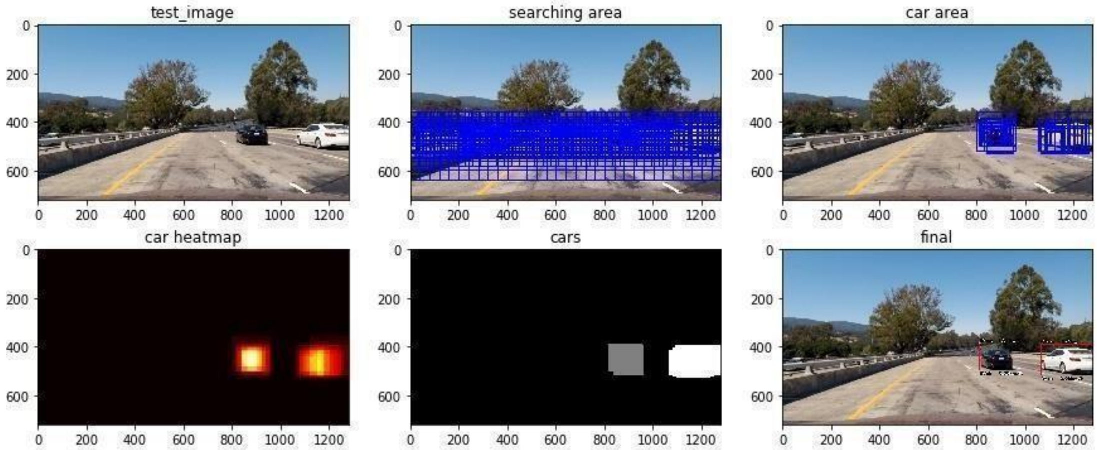


Fig.9.4 Identifying Obstacles

If there are more than one bounding boxes on a particular object that the box with maximum confidence score is retained.so, finally predict the image is an object (car). In (Fig 9.5) the lane edges are being detected and finds the path for the vehicle to move foward.

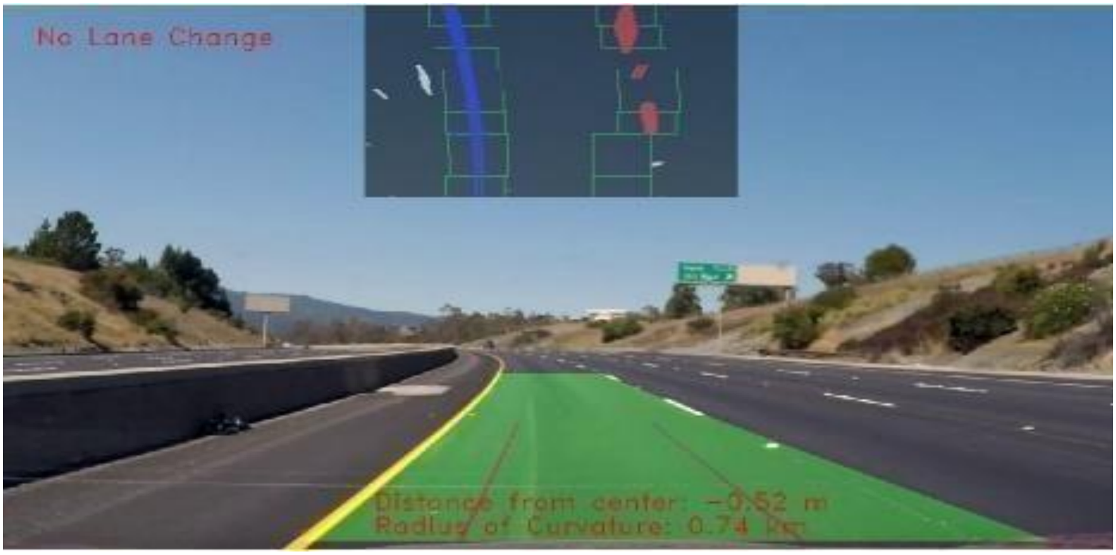


Fig. 9.5 Region of Interest

Fig 9.6 Detecting nearby vehicle

Fig 9.7 Detecting front vehicle

The camera detects the nearby vehicle then shows a warning signal to the driver (Fig 9.6 and Fig 9.7) and then it calculates the velocity and distance of the nearby vehicle from the car by using the Manhattan distance and changes the lane by adjusting the speed and direction according to the nearby vehicle, and vehicle which is behind our vehicle using rear camera.

CHAPTER 10

CONCLUSION AND FUTURE ENHANCEMENT

10.1 CONCLUSION

The project introduces a lane detection system that uses computer vision-based technologies to detect lanes on the road effectively. The proposed lane detection system combines various methods such as pre-processing, thresholding, and perspective transform. Gradient and HLS thresholding are effective at detecting the lane line in binary images. The left and right lanes on the road are identified using sliding window search. The cropping technique only worked in the area where the lane lines are located. Based on the results of the experiments, it can be concluded that the system detects lanes effectively in any environment. The system can be used on any road with well-marked lines and integrated into an embedded system to keep them on track.

10.2 FUTURE ENHANCEMENT

Autonomous driving is the most efficient way to avoid the road accident. It helps people to keep their vehicle in its lane and help in reducing on road collisions. By adding technologies like SOS it can send the SOS message to the nearby rescue center in case of accidents. So, road accident can be minimized to a greater extent by using Autonomous driving system.

ANNEXURE

Pipeline_helpers.py

```
import numpy as np
import cv2

# generate camera calibration parameters
def get_undist_params(fn_prefix='./camera_cal/calibration', nx=9, ny=6):
  """
  Compute parameters needed to undistort the distorted image

  Input
  -----
  fn_prefix : file name prefix which should be the path to the calibration chessboard images

  nx : number of corners in each row of a chessboard image

  ny : number of corners in eahc column of a chessboard image

  Output
  -----
  A 5-element tuple containing objects of different types
  """
  fnames = []
  objpoints = []
  imgpoints = []
  objp = np.zeros((nx*ny, 3), np.float32)
  objp[:,:2] = np.mgrid[0:nx,0:ny].T.reshape(-1,2)

  for i in range(20):
    fname = fn_prefix + str(i+1) + '.jpg'
    fnames.append(fname)

  for fn in fnames:
    img = cv2.imread(fn)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(img_gray, (nx, ny), None)
```

```
    if ret:
      objpoints.append(objp)
      imgpoints.append(corners)

  ret, camMat, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,   img.shape[:-1][::-1],
                                                             None, None)
  return ret, camMat, distCoeffs, rvecs, tvecs

# the following methods are for gradient and color thresholding
def abs_sobel_thresh(gray, orient='x', sobel_kernel=3, thresh=(0, 255)):
  """
  Compute binary grayscaled image that captures the lane lines

  Input
  -----
  gray : a gray image

  orient : the axis along which you compute your gradients

  sobel_kernel : the kernel size passed into Sobel()

  thresh : thresholds used to exclude noises

  Output
  -----
  A binary grayscaled image
  """
  if orient == 'x':
    sobel = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
  elif orient == 'y':
    sobel = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
  else:
    raise ValueError('orient must be either x or y')

  abs_sobel = np.absolute(sobel)
  scaled_sobel = np.uint8(255 * abs_sobel / np.max(abs_sobel))
  binary_output = np.zeros_like(scaled_sobel)
  binary_output[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1
```

```
  return binary_output

def mag_thresh(gray, sobel_kernel=3, thresh=(0, 255)):
  sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
  sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
  mag = np.sqrt(sobelx**2, sobely**2)
  scaled_sobel = np.uint8(255 * mag / np.max(mag))

  binary_output = np.zeros_like(scaled_sobel)
  binary_output[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1

  return binary_output

def dir_thresh(gray, sobel_kernel=3, thresh=(0, np.pi/2)):
  sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
  sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
  abs_sobelx = np.absolute(sobelx)
  abs_sobely = np.absolute(sobely)

  abs_grad_dir = np.arctan2(abs_sobely, abs_sobelx)

  binary_output = np.zeros_like(abs_grad_dir)
  binary_output[(abs_grad_dir >= thresh[0]) & (abs_grad_dir <= thresh[1])] = 1
  return binary_output

# I implemented the function used to do thresholding gradients but decided not to use it.
# Because a forum mentor said it was terrible with shadows.
# The final output video is the one generated without gradients. Looks fine.
def combine_grad(gray, ksize=3, grad_thresh=(20, 100), magnitude_thresh=(30, 100), direction_thresh=(0.7, 1.57)):
  """
  Compute binary grayscaled image that captures the lane lines

  Input
  -----
  gray : a gray image

  ksize : kernel size to pass into other methods called in this method
```

```
  grad_thresh : thresholds passed into abs_sobel_thresh()

  magnitude_thresh : threhsolds passed into mag_thresh()

  direction_thresh : thresholds passed into dir_thresh()

  Output
  -----
  A binary grayscaled image
  """
  gradx = abs_sobel_thresh(gray, orient='x', sobel_kernel=ksize, thresh=grad_thresh)
  grady = abs_sobel_thresh(gray, orient='y', sobel_kernel=ksize, thresh=grad_thresh)
  mag_binary = mag_thresh(gray, sobel_kernel=ksize, thresh=magnitude_thresh)
  dir_binary = dir_thresh(gray, sobel_kernel=15, thresh=direction_thresh)

  combined = np.zeros_like(dir_binary)
  combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))] = 1

  return combined

def combine_color(img_bgr, rgb_thresh=(220, 255), hls_thresh=(90, 255)):
  """
  Compute binary grayscaled image that captures the lane lines

  Input
  -----
  img_bgr : image in BGR form

  rgb_thresh : thresholds for R, G, B channel images to exclude noises

  hls_thresh : thresholds for H, L, S channel images to exclude noises

  Output
  -----
  A binary grayscaled image
  """
```

```
  img_r = img_bgr[:, :, 2]
  binary_r = np.zeros_like(img_r)
  binary_r[(img_r > rgb_thresh[0]) & (img_r <= rgb_thresh[1])] = 1

  hls = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2HLS)
  S = hls[:, :, 2]
  L = hls[:, :, 1]
  binary_s = np.zeros_like(S)
  binary_l = np.zeros_like(L)
  binary_s[(S > hls_thresh[0]) & (S <= hls_thresh[1])] = 1
  binary_l[(L > hls_thresh[0]) & (L <= hls_thresh[1])] = 1

  combined = np.zeros_like(img_r)
  combined[((binary_s == 1) & (binary_l == 1)) | (binary_r == 1)] = 1

  return combined

# I didn't use window_mask method in my pipeline, skip reading it if you
want
# window_mask is used to draw green windows on the lane lines in the
image
def window_mask(width, height, img_ref, center,level):
  output = np.zeros_like(img_ref)
  output[int(img_ref.shape[0]-(level+1)*height):int(img_ref.shape[0]-
level*height),
        max(0,int(center-width/2)):min(int(center+width/2),
                                img_ref.shape[1])] = 1
  return output

def find_lane_line_pixels(image, window_width, window_height,
margin):
  """
  Computes all the coordinates for the pixels that constitute the lane lines
  Use convolution to find window centroid. Then the lane line pixels

  Inputs
  -----
  image : presumably it is a binary grayscaled image, with elements of
only 0 or 1

  window_width : covolution window width of your choice
```

window_height : covolution widnow height of your choice

margin : the horizontal offset from the window centroids we use to draw a bounding box
at each level to find lane line pixels. Dont confuse it with covolution window width

Outputs
-----
a 4-element tuple containing x coordinates and y coordinates for left and right lane lines
"""
window_centroids = [] # Store the (left,right) window centroid positions per level
window = np.ones(window_width) # Create our window template that we will use for convolutions

First find the two starting positions for the left and right lane by using np.sum to get the vertical image slice
and then np.convolve the vertical image slice with the window template

# Sum quarter bottom of image to get slice, could use a different ratio
# image is a grayscale, looking at lower left quarter
l_sum = np.sum(image[int(image.shape[0]/2):,:int(image.shape[1]/2)], axis=0)
l_center = np.argmax(np.convolve(window,l_sum))-window_width/2
# the lower right quarter
r_sum = np.sum(image[int(image.shape[0]/2):,int(image.shape[1]/2):], axis=0)
# the convolution starts from index 0, so we shift it by half of the width
r_center = np.argmax(np.convolve(window,r_sum))-window_width/2+int(image.shape[1]/2)

note l_center and r_center are x coordinates in the image

# Add what we found for the first layer
# this is our first window
window_centroids.append((l_center, r_center))

still we need to collect all the nonzero pixels in this window
so later we can fit a polynomial

```
  # here the window width used to collect pixels is 2 * margin
  nonzero = image.nonzero()
  nonzeroy = nonzero[0]
  nonzerox = nonzero[1]

  left_lane_inds = []
  right_lane_inds = []

  win_y_low = image.shape[0] - window_height
  win_y_high = image.shape[0]

  win_x_l_low = l_center - margin
  win_x_l_high = l_center + margin
  win_x_r_low = r_center - margin
  win_x_r_high = r_center + margin

  good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy <
win_y_high) &
            (nonzerox >= win_x_l_low) & (nonzerox <
win_x_l_high)).nonzero()[0]
  good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy <
win_y_high) &
            (nonzerox >= win_x_r_low) & (nonzerox <
win_x_r_high)).nonzero()[0]

  left_lane_inds.append(good_left_inds)
  right_lane_inds.append(good_right_inds)

  # Go through each layer looking for max pixel locations
  for level in range(1,(int)(image.shape[0]/window_height)):
    # convolve the window into the vertical slice of the image
    # in the loop we go through the entire width
    image_layer = np.sum(image[int(image.shape[0]-
(level+1)*window_height):int(image.shape[0]-level*window_height),:],
axis=0)
    conv_signal = np.convolve(window, image_layer)
    # Find the best left centroid by using past left center as a reference
    # Use window_width/2 as offset because convolution signal reference
is at right side of window, not center of window
    offset = window_width/2
    # to avoid negative index, use max()
    # it is the index in conv_signal
```

```
    l_min_index = int(max(l_center+offset-margin,0))
    # to avoid index larger than width, use min()
    l_max_index = int(min(l_center+offset+margin,image.shape[1]))
    # get the index in original image
    l_center = np.argmax(conv_signal[l_min_index:l_max_index])+l_min_index-offset
    # Find the best right centroid by using past right center as a reference
    r_min_index = int(max(r_center+offset-margin,0))
    r_max_index = int(min(r_center+offset+margin,image.shape[1]))
    r_center = np.argmax(conv_signal[r_min_index:r_max_index])+r_min_index-offset
    # Add what we found for that layer
    window_centroids.append((l_center,r_center))

    win_y_low = image.shape[0] - (level + 1) * window_height
    win_y_high = image.shape[0] - level * window_height

    win_x_l_low = l_center - margin
    win_x_l_high = l_center + margin
    win_x_r_low = r_center - margin
    win_x_r_high = r_center + margin

    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                (nonzerox >= win_x_l_low) & (nonzerox < win_x_l_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                (nonzerox >= win_x_r_low) & (nonzerox < win_x_r_high)).nonzero()[0]

    left_lane_inds.append(good_left_inds)
    right_lane_inds.append(good_right_inds)

  left_lane_inds = np.concatenate(left_lane_inds)
  right_lane_inds = np.concatenate(right_lane_inds)
  # centroids we get is the x coordinates for n windows
  # Extract left and right line pixel positions
  leftx = nonzerox[left_lane_inds]
  lefty = nonzeroy[left_lane_inds]
  rightx = nonzerox[right_lane_inds]
  righty = nonzeroy[right_lane_inds]
```

```
  # Fit a second order polynomial to each
  left_fit = np.polyfit(lefty, leftx, 2)
  right_fit = np.polyfit(righty, rightx, 2)

  # Generate x and y values for plotting
  ploty = np.linspace(0, image.shape[0]-1, image.shape[0])
  left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
  right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]

  return leftx, lefty, rightx, righty
```

Pipeline.py

```
import pickle
import matplotlib.pyplot as plt
import PIL
import pipeline_helpers as ph
import numpy as np
import cv2
import imageio
from moviepy.editor import VideoFileClip


# Define a class to receive the characteristics of each line detection
class Line():
  def __init__(self):
    # was the line detected in the last iteration?
    self.detected = False
    # x values of the last n fits of the line
    self.recent_xfitted = []
    # average x values of the fitted line over the last n iterations
    self.bestx = None
    # polynomial coefficients averaged over the last n iterations
    self.best_fit = None
    # polynomial coefficients for the most recent fit
    self.current_fit = [np.array([False])]
    # radius of curvature of the line in some units
    self.radius_of_curvature = None
```

```
    # distance in meters of vehicle center from the line
    self.line_base_pos = None
    # difference in fit coefficients between last and new fits
    self.diffs = np.array([0, 0, 0], dtype='float')
    # x values for detected line pixels
    self.allx = None
    # y values for detected line pixels
    self.ally = None


# load the undistort parameters for future use, or run the method to generate
# and save them
def get_undist_params():
  """
  Compute parameters needed to undistort the distorted image

  Output
  -----
  A 5-element tuple containing objects of different types
  """
  try:
    with open('undist_params.p', mode='rb') as f:
      undist_params = pickle.load(f)
      ret, camMat, distCoeffs, rvecs, tvecs = undist_params['ret'], \
                                              undist_params['camMat'], \
                                              undist_params['distCoeffs'], \
                                              undist_params['rvecs'], \
                                              undist_params['tvecs']
  except FileNotFoundError:
    undist_params = {}
    ret, camMat, distCoeffs, rvecs, tvecs = ph.get_undist_params()
    undist_params['ret'], undist_params['camMat'], undist_params['distCoeffs'], \
      undist_params['rvecs'], undist_params['tvecs'] = ret, camMat, distCoeffs, rvecs, tvecs

    with open('undist_params.p', mode='wb') as f:
      pickle.dump(undist_params, f)

  return ret, camMat, distCoeffs, rvecs, tvecs
```

```
# load and preprocess the image
def get_all_imgs(img, is_bgr):
  """
  Return the same images in 3 different color form

  Input
  -----
  img : an image that is either in RGB or BGR form

  is_bgr : Boolean value indicating if 'img' is in BGR form

  Output
  -----
  img : image in RGB form

  img_bgr : image in BGR form

  img_gray : image in grayscale form
  """
  if is_bgr:
    img_bgr = img
    img = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
  else:
    img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
  img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)

  return img, img_bgr, img_gray


def get_perspective_mat(camMat, distCoeffs, img_size):
  """
  Return matrices used to transform images between different perspectives

  Input
  -----
  camMat : calibration matrix use to undistort images

  distCoeffs : distortion coefficients used to undistort images

  Output
  -----
```

```
  matP : matrix used to transform images from original to bird eye's view

  matP_inv : matrix used to transform images from bird eye's view to
original
  """
  # either we load the matrices that can be used to transform images to
bird eye's
  # view perspective or we generate such matrices from a straight line
image
  try:
    with open('perspective_params.p', mode='rb') as f:
      perspective_params = pickle.load(f)
      matP, matP_inv = perspective_params['matP'],
perspective_params['matP_inv']
  except FileNotFoundError:
    straight_line_bgr = cv2.imread('./test_images/straight_lines2.jpg')
    undist_straight_line = cv2.undistort(straight_line_bgr, camMat,
distCoeffs, None, camMat)

    # uncomment the following line to save undistorted straight line image
to your local disk
    # after storing the undistorted image we can eyeball the 4 rectangle
corners in the image
    # cv2.imwrite('./output_images/undist_straight_line.jpg',
undist_straight_line)
    # coordinates are in (width, hieght) or (x, y)
    # following are the 4 corners of a rectangle in the unwarped image by
eyeballing
    # with some software help

    upper_left = [578, 462]
    upper_right = [704, 462]
    lower_left = [212, 719]
    lower_right = [1093, 719]

    offset_x = 300
    offset_y = 0

    src = np.float32([upper_left, upper_right, lower_right, lower_left])
    dst = np.float32([[offset_x, offset_y], [img_size[0] - offset_x - 1,
offset_y],
               [img_size[0] - offset_x - 1, img_size[1] - offset_y - 1],
```

```
                 [offset_x, img_size[1] - offset_y - 1]])

    matP = cv2.getPerspectiveTransform(src, dst)
    matP_inv = cv2.getPerspectiveTransform(dst, src)

    perspective_params = {}
    perspective_params['matP'], perspective_params['matP_inv'] = matP,
matP_inv
    with open('perspective_params.p', mode='wb') as f:
      pickle.dump(perspective_params, f)

  return matP, matP_inv


def get_bin_lane_line_img(img_gray, img_bgr):
  """
  Get the best lane lines captured image we can with color space

  Input
  -----
  img_gray : an image in grayscale form

  img_bgr : an image in BGR form

  Output
  -----
  combined_color : lane lines captured binary grayscaled image by
merging images from different color channels
  """
  combined_color = ph.combine_color(img_bgr)

  return combined_color


def color_warped_lane_lines(warped, leftx, lefty, rightx, righty):
  """
  For the warped image, set pixels that form the left line in red, pixels that
  form the right line in blue
  """
  warped[lefty, leftx] = [255, 0, 0]  # left line in red
  warped[righty, rightx] = [0, 0, 255]  # right line in blue
```

```
  return warped


def get_lane_line_bounded_image(warped, left_fitx, right_fitx, ploty, margin):
  """
  Return a warped image that within a margin along the lane lines colored in green

  Input
  -----
  warped : original warped image

  left_fitx : x coordinates for the left fitted polynomial line

  right_fitx : x coordinates for the right fitted polynomial line

  ploty : y coordinates for left and right polynomial lines

  margin : offset from the polynomial lines that determines the width of the colored area
  """
  window_img = np.zeros_like(warped)
  # Generate a polygon to illustrate the search window area
  # And recast the x and y points into usable format for cv2.fillPoly()
  # vstack is vertical stack, more rows
  # transpose gives you pairs of pixel coordinates, (x, y), it gives you left lane line
  # left boundary from top to bottom
  left_line_window1 = np.array([np.transpose(np.vstack([left_fitx - margin, ploty]))])
  # it gives you left lane line right boundary from bottom to top, by flipping upside down
  left_line_window2 = np.array([np.flipud(np.transpose(np.vstack([left_fitx + margin,
                              ploty])))])

  # (left_x, y, right_x, y), y is the same, so it gives you the smooth boundaries of lane lines
  # left_line_window1 and left_line_window2 are of the shape(1, 720, 2)
  # horizontal stack is stacking the 2nd dimension, (1, 1440, 2)
  left_line_pts = np.hstack((left_line_window1, left_line_window2))
```

```
  right_line_window1 = np.array([np.transpose(np.vstack([right_fitx -
margin, ploty]))])
  right_line_window2 =
np.array([np.flipud(np.transpose(np.vstack([right_fitx + margin,
                              ploty])))])
  right_line_pts = np.hstack((right_line_window1, right_line_window2))

  # Draw the lane onto the warped blank image
  cv2.fillPoly(window_img, np.int_([left_line_pts]), (0, 255, 0))
  cv2.fillPoly(window_img, np.int_([right_line_pts]), (0, 255, 0))

  return window_img


def get_cuvature(leftx, lefty, rightx, righty, ploty):
  """
  Return radiuses of curvature for left and right lane lines

  Input
  -----
  leftx : x coordinates for pixels that form the left lane line

  lefty : y coordinates for pixels that form the left lane line

  rightx : x coordinates for pixels that form the right lane line

  righty : y coordinates for pixels that form the right lane line

  ploty : y coordinates for left and right polynomial lines

  Output
  -----
  left_curverad : Radius of curvature measured in meters for left lane line.
                It is measured at the bottom of the image
  right_curverad : Radius of curvature measured in meters for right lane
line.
                It is measured at the bottom of the image
  """
  ym_per_pix = 30 / 720  # meters per pixel in y dimension
  xm_per_pix = 3.7 / 700  # meters per pixel in x dimension

  y_eval = np.max(ploty)
```

```
  # Fit new polynomials to x,y in world space
  left_fit_cr = np.polyfit(lefty * ym_per_pix, leftx * xm_per_pix, 2)
  right_fit_cr = np.polyfit(righty * ym_per_pix, rightx * xm_per_pix, 2)
  # Calculate the new radii of curvature
  left_curverad = ((1 + (2 * left_fit_cr[0] * y_eval * ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2 * left_fit_cr[0])
  right_curverad = ((1 + (2 * right_fit_cr[0] * y_eval * ym_per_pix + right_fit_cr[1])**2)**1.5) / np.absolute(2 * right_fit_cr[0])

  return left_curverad, right_curverad


def get_car_offset(combined, matP, img_size, bottom_left_fitx, bottom_right_fitx, xm_per_pix):
  """
  Return the measurement of how much car center is off the lane center, in meters

  Input
  -----
  combined : the undistorted, binary grayscaled image in original perspective

  matP : matrix used to transform images from original to bird eye's view

  img_size : a tuple containing (image_width, image_height)

  bottom_left_fitx : x coordinate for the left line polynomial fit when y = 719

  bottom_right_fitx : x coordinate for the right line polynomial fit when y = 719

  xm_per_pix : measurement on how many meters changed by increasing or decreasing a pixel horizontally
          in the image

  Output
  -----
  car_offset_meters : measurement in meters on how much car center is off the lane center
```

```
  """
  # also I want to know where the car center is in the warped image
  # car center is assumed to be horizontally in the center of the unwarped image
  # I mark the center at the bottom of the unwarped image, warp the image, find
  # the marked point in the warped image then I get the car center in the warped image
  car_center = np.zeros_like(combined)
  car_center[car_center.shape[0] - 1, car_center.shape[1] // 2 - 1] = 1
  car_center_warp = cv2.warpPerspective(car_center, matP, (img_size[0], img_size[1]))
  car_centerx = np.argmax(car_center_warp[car_center_warp.shape[0] - 1, :])
  lane_centerx = ((bottom_right_fitx + bottom_left_fitx) // 2)

  car_offset_meters = (car_centerx - lane_centerx) * xm_per_pix

  return car_offset_meters


def color_unwarped_lane(warped, img_size, left_fitx, right_fitx, ploty, matP_inv):
  """
  Return an image in the unwarped perspective with lane colored in green

  Input
  -----
  warped : warped image in bird eye's perspective

  img_size : a tuple containing (image_width, image_height)

  left_fitx : x coordinates for the left fitted polynomial line

  right_fitx : x coordinates for the right fitted polynomial line

  ploty : y coordinates for left and right polynomial lines

  matP_inv : matrix used to transform images from bird eye's view to original
  """
  # Let us try to draw the lane in green and warp it back to the original
```

```
perspective
  # Create an image to draw the lines on
  warp_zero = np.zeros_like(warped).astype(np.uint8)
  color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

  # Recast the x and y points into usable format for cv2.fillPoly()
  pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
  pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx,
ploty])))])
  pts = np.hstack((pts_left, pts_right))

  # Draw the lane onto the warped blank image
  cv2.fillPoly(color_warp, np.int_([pts]), (0, 255, 0))

  # Warp the blank back to original image space using inverse perspective
matrix (Minv)
  newwarp = cv2.warpPerspective(color_warp, matP_inv, (img_size[0],
img_size[1]))

  return newwarp


def paste_curvature_and_offset(image, curverad, offset):
  """
  Return image with curvature and car offset information embedded

  Input
  -----
  image : image to be modified

  curverad : Radius of curvature in meters

  offset : measurement in meters on how much car center is off the lane
center
  """
  font = cv2.FONT_HERSHEY_SIMPLEX
  image = cv2.putText(image, "lane curvature: " + str(curverad) + "
meters", (20, 40), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
  image = cv2.putText(image, "car offset: " + str(offset) + " meters", (20,
120), font, 1, (255, 255, 255), 2, cv2.LINE_AA)

  return image
```

```
def update_line(line, fitx, fit):
  """
  Update Line() instance variable

  Input
  -----
  line : the Line() instance to be updated

  fitx : x coordinates for the fitted polynomial line

  fit : 2nd order polynomial coefficients
  """
  line.detected = True
  num_tracked_lines = len(line.recent_xfitted)

  # I choose to track up to 10 latest frames
  if num_tracked_lines == 10:
    line.recent_xfitted.pop(0)
  line.recent_xfitted.append(fitx)

  line.bestx = np.mean(line.recent_xfitted, axis=0)

  if line.best_fit is None:
    line.best_fit = fit
  else:
    if num_tracked_lines == 10:
      line.best_fit = (line.best_fit * num_tracked_lines + fit) /
num_tracked_lines
    else:
      line.best_fit = (line.best_fit * num_tracked_lines + fit) /
(num_tracked_lines + 1)

  line.diffs = fit - line.current_fit
  line.current_fit = fit


# In order to make my pipeline function compatible with `moviepy`
functions
# I wrapped up image processing pipeline in another function.
# This is called closure. It is also how currying is done in Python.
```

```
def process_frames(is_bgr=True, left_line=None, right_line=None):
  """
  Return a pipeline function that can process a single image

  Input
  -----
  is_bgr : parameter that decide if the returned function takes BGR images
or RGB images

  left_line : the Line() instance used to keep track of the information of the
            detected left lines in the last n frames
  right_line : the Line() instance used to keep track of the information of
the
            detected right lines in the last n frames

  Output
  -----
  process_image : a function that takes an BGR or RGB image as its
argument
  """
  if left_line is None:
    left_line = Line()
  if right_line is None:
    right_line = Line()

  def process_image(img):
    """
    Return the original image with the lane colored in green

    Input
    -----
    img : an RGB or BGR image
    """
    # STEP1: Camera Calibration
    # we have many chessboard images from the same camera
    # we use all of them to calibrate the camera
    img, img_bgr, img_gray = get_all_imgs(img, is_bgr)
    img_size = (img.shape[1], img.shape[0])
    ret, camMat, distCoeffs, rvecs, tvecs = get_undist_params()
    matP, matP_inv = get_perspective_mat(camMat, distCoeffs, img_size)
    undist = cv2.undistort(img, camMat, distCoeffs, None, camMat)
    # STEP2: retrieve a grayscale image only contains lane lines
```

```
    combined = get_bin_lane_line_img(img_gray, img_bgr)

    # STEP3: let us warp the image to bird's eyes view perspective
    warped = cv2.warpPerspective(combined, matP, (img_size[0],
img_size[1]))

    # window settings
    # this is the window_width used to do convolution
    window_width = 50
    window_height = 180  # Break image into 9 vertical layers since image
height is 720
    margin = 100  # How much to slide left and right for searching
    xm_per_pix = 3.7 / 700

    # find the lane lines centers in the bird eye's perspective
    leftx, lefty, rightx, righty = ph.find_lane_line_pixels(warped,
window_width, window_height, margin)

    ploty = np.linspace(0, warped.shape[0] - 1, warped.shape[0])
    # compute curvature of the lane
    left_curverad, right_curverad = get_cuvature(leftx, lefty, rightx, righty,
ploty)
    curverad = (left_curverad + right_curverad) / 2

    # Fit a second order polynomial to each
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    # Generate x and y values for plotting
    left_fitx = left_fit[0] * ploty**2 + left_fit[1] * ploty + left_fit[2]
    right_fitx = right_fit[0] * ploty**2 + right_fit[1] * ploty + right_fit[2]

    update_line(left_line, left_fitx, left_fit)
    update_line(right_line, right_fitx, right_fit)

    # uncommet the following code to generate warped image with lane
line pixels colored
    # red and blue. Also each line will be covered by a green polygon with
certain width
    # out_img = np.dstack((warped, warped, warped))*255
    # out_img = color_warped_lane_lines(out_img, leftx, lefty, rightx,
righty)
```

```
    # lane_line_bounded = get_lane_line_bounded_image(out_img, left_fitx, right_fitx, ploty, margin)

    # colored_lane_line_bounded = cv2.addWeighted(out_img, 1, lane_line_bounded, 0.3, 0)

    # compute car offset
    car_offset = get_car_offset(combined, matP, img_size, left_fitx[-1], right_fitx[-1], xm_per_pix)

    colored_lane = color_unwarped_lane(warped, img_size, left_line.bestx, right_line.bestx, ploty, matP_inv)
    colored_lane_img = cv2.addWeighted(undist, 1, colored_lane, 0.3, 0)
    colored_lane_img = paste_curvature_and_offset(colored_lane_img, curverad, car_offset)

    return colored_lane_img

  return process_image


if __name__ == '__main__':
  clip1 = VideoFileClip("./project_video.mp4")

  left_line = Line()
  right_line = Line()
  result = clip1.fl_image(process_frames(is_bgr=False, left_line=left_line, right_line=right_line))
  result.write_videofile('./detected_project_video.mp4', audio=False)
```

REFERENCES

[1] Andrei. M.-A, Boiangiu. C.-A, Tarbă. N, and Voncilă. M.L, ‘‘Robust Lane detection and tracking algorithm for steering assist systems,’’ Machines, vol. 10, no. 1, p. 10, Dec. 2021.

[2] Biswas. S, Tatchikou. R, and Dion. F, “Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety,” IEEE communications magazine, vol. 44, no. 1, pp.74–82, 2022.

[3] Chen. S, Huang. L, Chen. H, and Bai. J, ‘‘Multi- Lane detection and tracking using temporal-spatial model and particle filtering,’’ IEEE Trans. Intell.Transp. Syst., vol. 23, no. 3, pp. 2227–2245, Mar. 2022.

[4] Dong. Y, Patil. S, Van Arem. B, and Farah. H, ‘‘a hybrid spatial–temporal deep learning architecture for lane detection,’’ Comput. -Aided Civil Infrastruct. Eng., vol. 38, no. 1, pp. 67–86, Jan. 2023.

[5] Fu. Z, Xiong. L, Qian. Z, Leng. B, Zeng. D, and Huang. Y, ‘‘Model Predictive trajectory optimization and tracking in highly constrained environments,’’ Int. J. Automot. Technol., vol. 23, no. 4, pp. 927–938, Aug. 2022.

[6] Hanuman. A.S and Kumar. G.P, ‘‘Robust and real-time multi-lane and single lane detection in Indian highway scenarios,’’ in Proc. E3S Web Conf., vol. 309, 2021, pp. 839–846.

[7] Jeong. Y and Yi. K, ‘‘Target vehicle motion prediction-based motion planning framework for autonomous driving in uncontrolled intersections,’’ IEEE Trans. Intell. Transp. Syst., vol. 22, no. 1, pp. 168–177, Dec. 2021.

[8] Lana. I, Sanchez-Medina. J.J, Vlahogianni E.I, and Ser. J.D, “From data to actions in intelligent transportation systems: A prescription of functional requirements for model actionability,” 2020.

[9] Liu. T, Chen. Z, Yang. Y, Wu. Z, and Li. H, ‘‘Lane detection in lowlight conditions using an efficient data enhancement: Light conditions style transfer,’’ in Proc. IEEE Intell. Vehicles Symp. (IV), Oct. 2020, pp. 1394–1399.

[10] Li. L, Lin. Y.L, Zheng. N.N, Wang. F.Y, Liu. Y, Cao. D, Wang. K, and Huang. W.L, “Artificial intelligence test: a case study of intelligent vehicles,” Artificial Intelligence Review.

[11] Li. X, Zhang. S, Chen. X, Wang. Y, Fan. Z, Pang. X, and Hu. J, ‘‘Robustness of visual perception system in progressive challenging weather scenarios,’’Eng. Appl. Artif. Intell., vol. 119, Mar. 2023.

[12] Liu. Y, Wang. J, Li. Y, Li. C, and Zhang. W, ‘‘Lane-GAN: A robust lane detection network for driver assistance system in high speed and complex road conditions,’’ Micromachines, vol. 13, no. 5, p. 716, Apr. 2022.

[13] Liu. T, Chen. Z, Yang. Y, Wu. Z, and Li. H, ‘‘Lane detection in lowlight conditions using an efficient data enhancement: Light conditions style transfer,’’ in Proc. IEEE Intell. Vehicles Symp. (IV), Oct. 2020, pp. 1394–1399.

[14] Luo. S, Zhang. X, Hu. J, and Xu. J, ‘‘Multiple Lane detection via combining complementary structural constraints,’’ IEEE Trans. Intell. Transp. Syst., vol. 22, no. 12, pp. 7597–7606, Dec. 2021.

[15] Nie. X, Xu. Z, Zhang. W, Dong. X, Liu. N, and Chen. Y, ‘‘Foggy Lane dataset synthesized from monocular images for lane detection algorithms,’’ Sensors, vol. 22, no. 14, p. 5210, Jul. 2022.

[16] Olofsson. B and Nielsen. L, ‘‘Using crash databases to predict effectiveness of new autonomous vehicle maneuvers for lane-departure injury reduction,’’ IEEE Trans. Intell. Transp. Syst., vol. 22, no. 6, pp. 3479–3490, Jun. 2021

[17] Rahaman. M.N, Biswas. M.S, Chaki. S, Hossain. M, Ahmed. S, and Biswas. M, ‘‘Lane detection for autonomous vehicle management: PHT approach,’’ in Proc. 24th Int. Conf. Comput. Inf. Technol. (ICCIT), Dec. 2021.

[18] Ren. K, Hou. H, Li. S, and Yue. T, ‘‘Lane Draw: Cascaded Lane and its bifurcation detection with nested fusion,’’ Sci. China Technol. Sci., vol. 64, no. 6, pp. 1238–1249, Jun. 2021.

[19] Sultana. S and Ahmed. B, ‘‘Lane detection and tracking under rainy weather challenges,’’ in Proc. IEEE Region 10 Symp. (TENSYMP), Aug. 2021, pp. 1–6.

[20] Sang. I.-C and Norris. W.R, ‘‘An autonomous underwater vehicle simulation with fuzzy sensor fusion for pipeline inspection,’’ IEEE Sensors J., vol. 23, no. 8, pp. 8941–8951, Apr. 2023.

[21] Tang. C, Liu. Y, Xiao. H, and Xiong. L, ‘‘Integrated decision making and planning framework for autonomous vehicle considering uncertain prediction of surrounding vehicles,’’ in Proc. IEEE 25th Int. Conf. Intell. Transp. Syst. (ITSC), Oct. 2022, pp. 3867–3872.

[22] Wang. S, Wang. Y, and Li. Y, ‘‘Robust Lane detection method based on dynamic region of interest,’’ IEEE Sensors J., early access, May 2023.

[23] Xu. Z, Wang. M, Zhang. F, Jin. S, Zhang. J, and Zhao. X, “Patavtt: A hardware-in-the-loop scaled platform for testing autonomous vehicle trajectory tracking,” Journal of Advanced Transportation, vol. 2021.

[24] Yoon. Y, Kim. D, Kim. C, and Yi. K, ‘‘Driving data-based motion planning of lane change maneuver for autonomous vehicles,’’ presented at the 15th Int. Symp. Adv. Vehicle Control, 2022.

[25] Yusuf. M. M, Karim. T, and Saif. A.F.M.S, ‘‘A robust method for lane detection under adverse weather and illumination conditions using convolutional neural network,’’ in Proc. Int. Conf. Comput. Advancements, Jan. 2020, pp. 1–8

[26] Wu. Y, Gou. W, and Chen. J, ‘‘A robust lane detection method based on CNN and self-attention traffic situations using adas,’’ J. Adv. Transp., vol. 2021, pp. 1–12, Aug. 2021.

[27] Zhang. R, Wu. Y, Gou. W, and Chen. J, ‘‘RS-lane: A robust lane detection method based on ResNeSt and self-attention distillation for challenging traffic situations,’’ J. Adv. Transp., vol. 2021, pp. 1–12, Aug. 2021.

[28] Zhang. C, Liu. Y, Zhang. Q, and Wang. L, “A graded offline evaluation framework for intelligent vehicle’s cognitive ability”.

[29] Zheng. T, Huang. Y, Liu. Y, Tang. W, Yang. Z, Cai. D, and He. X,‘‘CLRNet: Cross layer refinement network for lane detection,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2022.

[30] Zhou. H and Song. X, ‘‘Lane detection algorithm based on Haar feature based coupled cascade classifier,’’ in Proc. IEEE Asia–Pacific Conf. Image Process., Electron. Comput. (IPEC), Apr. 2021, pp. 286–91.