LANE DETECTION FOR SELF-DRIVING CARS USING PYTHON

A PROJECT

Submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF COMPUTER APPLICATIONS

By

NAME	ROLL NO.
Priya Ghosh	BCA/40043/21
Eshita Kumari	BCA/40048/21
Swati Tiwari	BCA/40049/21
Brishti Dey	BCA/40058/21
Mitali Kumari	BCA/40067/21

UNDER THE GUIDANCE OF

Dr. Partha Sarathi Bishnu

(Department of Computer Science)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BIRLA INSTITUTE OF TECHNOLOGY

(LALPUR), RANCHI

(2024)

BIRLA INSTITUTE OF TECHNOLOGY

MESRA,RANCHI- 835215

(Deemed university)

(LALPUR), RANCHI

CERTIFICATE

THIS IS TO CERTIFY THAT THE CONTENTS OF THIS PROJECT ENTITLED "LANE DETECTION FOR SELF- DRIVING CARS USING PYTHON" IS A BONAFIDE WORK CARRIED OUT BY PRIYA GHOSH, ESHITA KUMARI, SWATI KUMARI, BRISHTI DEY AND MITALI KUMARI IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF BACHELOR OF COMPUTER APPLICATIONS UNDER MY GUIDANCE.

THIS IS TO FURTHER CERTIFY THAT THE MATTER OF THE PROJECT HAS NOT BEEN SUBMITTED BY ANYONE ELSE FOR THE AWARD OF ANY DEGREE.

Dr. Partha Sarthi Bishnu

Department of Computer Science,

Birla Institute of Technology

Mesra (Lalpur),Ranchi.

BIRLA INSTITUTE OF TECHNOLOGY

MESRA, RANCHI-835215

(Deemed University)

(LALPUR), RANCHI

CERTIFICATE OF APPROVAL

The foregoing project is hearby approved as a creditable as a creditable work on “Lane detection for self-Driving cars using Python” , carried out and presented in the manner satisfactory to warrant its acceptance as prerequisite to the degree for which it has been entitled.

It is understood that by this approval the undersigned do not endorse any statement made or opinion expressed but approve work for the purpose for which it is submitted.

Committee for evaluation of the project

External Examiner

Date:

Internal Examiner

Date:

Director (In-charge)

BIRLA INSTITUE OF TECHNOLOGY

(LALPUR), RANCHI

ACKNOWLEDGEMENT

We are highly indebted to Prof. Partha Bishnu for his guidance and constant supervision as well as for providing us with necessary information regarding the project and also his support in completing the project.

We would especially like to express our gratitude to Dr. Amrita Priyam the Head of Department of Computer Science for providing us an opportunity to work in.

We are grateful to Dr. Vandana Bhattarcharjee, In- charge of Birla institute of Technology Lalpur Campus for extending all facilities that we required for carrying out this work.

We would like to express our gratitude towards our well-wisher and friends for their kind co-operation and encouragement which helped us in completion of this project.

NAME	ROLL NO.	SIGNATURE
Priya Ghosh	BCA/40043/21	
Eshita Kumari	BCA/40048/21	
Swati Kumari	BCA/40049/21	
Brishti Dey	BCA/40058/21	
Mitali Kumari	BCA/40067/21	

CONTENT

1. INTRODUCTION

In this we take a simple video as input data and process it to detect the lane within which the vehicle is moving. Then we will find a representative line for both the left and right lane lines and render those representations back out to the video as a red overlay. This post will be heavy on technical details about how to use the libraries available in the Python computer vision ecosystem to solve this problem.

Computer vision is an area of computer science devoted to the extraction and processing of structured information from mostly-unstructured image data. We are going to use OpenCV to process the input images to discover any lane lines held within and also for rendering out a representation of the lane. Additionally, images are really just dense matrix data, so we will use numpy and matplotlib to do transformations and rendering of image data. I've run all of this code within a Jupyter notebook, but you can run it in any environment that allows you to install the dependencies and execute python scripts.

First of all, one obvious way to make the problem easier is to work out our solution for a single image. A video is, after all, just a series of images. We can then move on to running our pipeline on an input video frame-by-frame as a final solution to the original problem of processing an entire video for lane detection.

For example, let's take the single image frame below.

Fig.1. Input

In the above image, the lane markers are obvious to any human observer. We perform processing of this image intuitively, and after being trained to drive a human can detect the lane in which the vehicle appears to be moving. Humans also effortlessly identify many other objects in the scene, such as the other vehicles, the embankment near the right shoulder, some road signs alongside the road, and even the mountains visible on the horizon. While many of these objects are complex in visual structure, it could be said that the lane markers are actually some of the simplest structures in the image!

Pre-existing knowledge of driving gives us certain assumptions about the properties and structure of a lane, further simplifying the problem. One obvious assumption is that the lane is oriented to be parallel with the direction of movement. This being the case, the lines denoting the lane will tend to extend from the foreground of an image into the background along paths that are angled slightly inwards. We can also assume that the lines will never quite reach the horizon, either disappearing with distance or being obscured by some other image feature along the way.

1.1. Introduction to features

We need to figure out what differentiates the objects of interest from the rest ofthe image. We've already seen that things like colors and gradients can be good differentiators but let's give them an identity. All of these potential characteristics are features that we could use. What features are more important may depend upon the appearance of the track in question. In most applications, we'll end up using a combination of features that give us the best results.

- **Colour features**

The simplest features we can get from images consists of raw color values. For instance, here is an image of black road. Well, using our known car image as is, you can simply detect the track difference between the track difference between the car image and the test region, and see the observation. This basically means we are obtaining the frame or layouts from a video or image from a camera feed.

Alternatively, we could computer the correlations between the car image and test region and
Check if that is high. Our known image is the template or model and we try to match it with regions of the test image.

- **Color spaces**
  In the lane detection algorithms, color space plays a crucial role in isolating the lane markings from the rest of the image. The choice of color space depends on the characteristics of the lane markings and the environment in which the detection is performed. Here are some common color spaces used in lane detection and how they are applied:

  RCB(Red-Green-Blue):

  Usage: The RGB color space is often used as the default color representation in image**.**

  Lane Detection: Lane markings can be deleted by thresholding's specific range of RGB values. For examples, lane markings typically have high values in all RGB channels**.**

  HSV(Hue-Saturation-Values):

  Usage: HSV separates color information(hue), intensity(value), and saturation.

  Lane Detection: Lane markings can be isolated by setting appropriate thresholds on the hue and saturation channels. This can be particularly effective for handling varying lighting conditions.

  Grayscale:

  Usage: Converting an image to grayscale simplifies lane detection by focusing only on intensity.

  Lane Detection: Grayscale images are often used for edges detection and can be effective for detecting lane boundaries based on intensity changes.

1.2. When implementing lane detection:

Thresholding : After converting to the desired color spaces, apply thresholding techniques(line simple thresholding, adaptive thresholding, or using multiple thresholds) to isolate the lane markings.

Morphological Operations: Use operations like erosion and dilation to clean up the binary image obtained from thresholding.

Region of Interest(ROI): Focus the processing within a defined region of interest to improve efficiency and accuracy.

The optimal color space and techniques depend on factor such as lighting conditions, road surface variations, and the color of lane markings. Experimentation and tuning are often required to achieve robust lane detection performance across different environment.

1.3. Combining Features

Combining features of lane detection typically involves using various computer vision techniques and algorithms to identify and track lanes on roads or highways in images or video streams. Here are steps and methods commonly used for this:

1. **Image Preprocessing:**
- Convert the image to grayscale.
- Apply Gaussian blue to reduce noise.
- Use Canny edge detection to identify potential edges.

2. **Region of Interest (ROI) Selection:**
- Define the area of the image where lane lines are expected to appear(usually a trapezoidal shape covering the road ahead).

3. **Hough Transform:**
- Apply Hough transform on the edge-detection image within the ROI to detect lines.
- Convert detected lines from polar to Cartesian coordinates.

4. **Line Filtering:**
- Filter out lines based on slope and position to select potential lane lines.
- Separate detected lines into left and right lanes based on slope direction.

5. **Lane line estimation:**
- use technique like linear regression or polynomial fitting to estimate the lane lines based on the filtered line segments.
- Extrapolate the lane lines to cover the full extent from the bottom to the top of the ROI.

6. **Visualization:**
- Overlay the detected lane lines back onto the original image or video frames.

7. **Additional techniques (optional):**
- Implemented advanced methods like Kalman filtering or deep learning for robust lane detection under varying conditions(e.g., different lighting, road markings, etc.)

By combining these techniques effectively, you can create a lane detection system that can reliably identify and track lanes on roads, which is essential for applications like autonomous driving or driver assistance systems. Each step can be turned and optimized based on specific use cases and environmental conditions.

THE EVOLUTION

The self-driving car initiative was first realized decades before Google started doing technical research on the subject. The first recorded conceptof an autonomous car was introduced in the 1939 New York World's Fair in the Futurama section. General Motors created the Futurama exhibit as part of its vision of the future of America in 20 years time. Engineers and futurists included an automated highway system on which the self-drivingcars would depend on to get people from one place to another.

Of course, it took over 60 years before robotic vehicles started cruising our streets, but they're not as plentiful as General Motors imagined themto be and they didn't even need to create the automated highway system.However, the goal to develop full-fledged autonomous vehicles is gainingtraction with the goal of making driving efficient and safe.

By 1958 — almost two decades since they introduced the concept duringthe World's Fair in New York — the self-driving car Norman Bel Geddes' created for General Motors was finally realized. It relied on magnetized metal spikes embedded in the roadway and was remotely controlled by adevice that guided the car by changing the electromagnetic fields in the spikes to keep the car inside its designated lane.

In 1977 the Tsukuba Mechanical Engineering Lab made some improvements to GM's self-driving car by using cameras linked to a computer, which could guide the car through the road at 20 mph viaimage data processing.

Ten years later, two of Germany's leading car manufacturers, Daimler and Mercedes Benz, collaborated on a project called VaMoRs. The VaMoRs wasa 5-ton Mercedes Benz van equipped with cameras and other sensors, modified to share data with an on-board computer that allowed it to drivethe car unassisted. This technology was a big leap from Japan's self- driving car, because it could cruise at 56 mph on any road or highway and not collide or crash with other vehicles or objects.

The improvement of the technologies used in self-driving vehicles is directly proportional to how these types of vehicles perform on the road.Better tech means better autonomous cars, but developers were just starting to scratch the surface.

The first company that Elon Musk started was Tesla motors. He intended Tesla to be an all-Electric, clean and energy-efficient vehicle manufacturerto kick-start his dream of a green future. The company was

incorporated in 2003 and Musk immediately held a press conference stating that he would develop affordable AEVs (autonomous elective vehicles) for the US and the rest of the world in 3 – 5 years time.

The first AEV coupe that Tesla Motors developed was the Tesla Roadster which had a 7,376 ft-lb of torque, can do 0-60 mph in 1.9 seconds and hasa top speed of 250+ mph. It can go as far as 620 miles in a single charge and for an AEV released in 2012 with this specs was truly impressive!

However, the Roadster cost $200,000 and obviously wasn't feasible for the masses, but even the subsequent Model S still cost twice as much asthe $30,000 Tesla sedan Musk promised.

From 2014 onward Tesla started to include the hardware needed for full self-driving capabilities in all of their vehicles even before the software/data was available, and yet Tesla has the best safety record compared to other auto brands in its class. Plus, Tesla's Autopilot® feature in their vehicles has 360-degree coverage! Electronic sensors and cameras literally surround the car on every corner and can recognize cars and pedestrians on the road in various distances.

The narrow forward camera has a maximum viewing distance of 250 meters and it is backed up by the secondary camera/sensor, the mainforward camera, which can see up to 150 meters. The wide forward camera scans the peripheral vision of the car up to 200 degrees even though it can only see up to 60 meters out into the road. Finally, the rearward looking cameras cover over 180 degrees of the rear of the vehicle and can see up to 100 meters away.

All these cameras and sensors combine to form a fusion of sensors and eyes of the vehicle making Tesla cars almost military-grade, unmanned navigation systems. But as advanced as this technology may seem, Musksaid that it isn't at full self-driving capability yet.

In January of 2019 Musk tweeted that in just 3 – 6 months Tesla motors would slowly depart from the autopilot feature and introduce its first-everself-driving technology. As early as the year's end, Tesla car owners wouldbe able to safely drive from Los Angeles to New York City without ever touching the vehicle's steering wheel. That's almost 3,000 miles! I will be amazing to see if Musk can deliver on his promise.

2. MODULES AND THEIR DESCRIPTION

The system comprises of following modules:

- Open CV library
- Numpy- For working with arrays and linear algebra
- Canny edge detection- for detecting edges
- Region of interest- To define specific area

Description:

(i) Open CV

Computer vision is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data fromthem. Computer Vision is the base or mostly used for Artificial Intelligence.Computer-Vision is playing a major role in self-driving cars, robotics as well asin photo correction apps .

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identifyimage pattern and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license andhence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android.

When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

Applications of OpenCV : There are lots of applications which are solved usingOpenCV, some of them are listed below,

- Face recognition

- Automated inspection and surveillance
- Number of people-count (foot traffic in a mall,etc)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anamoly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robust and driver-less car navigation and control
- Object recognition
- Medical image analysis
- Movies- 3D structure from motion
- TV channels advertisement recognition

Open CV Functionality

- Image/video I/O, processing, display (core, img proc, high gui)
- Object/feature detection (objdetect, feature2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo video, superres)
- Machine learning & clustering (ml, flam)
- CUDA acceleration (gpu)

Training a classifier

The training phase essentially consists of extracting features for each sample inthe training set, and supplying these feature vectors to the training algorithm, along with corresponding labels. The training algorithm initializes a model, andthen tweaks its parameters using the feature vectors and labels. Typically, this involves an iterative procedure where one or more samples are presented to theclassifier at a time, which then predicts their labels. The error between these predicted labels and ground-truth is used as a signal to modify the parameters.

(ii) NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays .The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Num array into Numeric, with extensive modifications.

NumPy is open-source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project.

FEATURES:

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents due to the absence of compiler optimization.

NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays; using these requires rewriting some code, mostly inner loops, using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since theyare both interpreted,[21] and they both allow the user to write fast programsas long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicingor masking with other arrays are very efficient ways to access specific pixels ofan image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

(iii) Canny Fuction:

The Canny edge detection algorithm is a widely used technique for detecting edges in images. It's named after John Canny, who introduced it in 1986. The algorithm involves several key steps:

1.Noise Reduction:

Before detecting edges, it's common to reduce noise in the image using a Gaussian blur. This step smooths out the image to remove high-frequency noise, improving the quality of edge detection.

2.Gradient Calculation:

The next step is to calculate the gradient (rate of change) of intensity in the image. This is typically done using Sobel filters in both the horizontal and vertical directions. The gradients (Gx and Gy) can be computed as:

- Gx = Image * Sobel kernel for horizontal gradient

- Gy = Image * Sobel kernel for vertical gradient

where '*' denotes convolution.

3.Edge Strength and Direction:

From the gradients Gx and Gy, we can compute the magnitude (edge strength) and direction of the gradient at each pixel:

- Edge Strength (G) = sqrt(Gx^2 + Gy^2)

- Gradient Direction (θ) = arctan(Gy / Gx)

4.Non-maximum Suppression:

The purpose of non-maximum suppression is to thin out the edges so that only local maxima arpreserved. For each pixel, the algorithm checks if the edge strength is the maximum among its neighboring pixels along the gradient direction. If it's not a local maximum, the edge strength is suppressed (set to zero).

5.Double Thresholding:

Two thresholds (low and high) are used to categorize edge pixels into strong, weak, and non-edge pixels:

- Strong Edge: Edge pixels above the high threshold.

- Weak Edge: Edge pixels between the low and high thresholds.

- Non-edge: Edge pixels below the low threshold.

6.Edge Tracking by Hysteresis:

This final step aims to connect weak edges into strong edges by tracing along the edges. Weak edges that are connected to strong edges are considered part of the edge map. This helps in forming continuous lines or curves from the detected edges.

- Takes an image (img) as input.

- Checks if the image is not empty. If it's empty, it releases resources, closes windows, and exits.

- Converts the image to grayscale.

- Applies Gaussian blur to the grayscale image.

- Uses Canny edge detection on the blurred image with specific parameters (50, 150).

- Returns the resulting Canny edge-detected image.

The output of the Canny edge detection algorithm is a binary image where edges are represented as white pixels on a black background. It's a foundational step in many computer vision applications such as object detection, image segmentation, and of course, lane detection. The Canny edge detector is valued for its ability to accurately detect edges while suppressing noise and providing precise localization of edges.

(iv) Region of Interest Function:

In lane detection, the region of interest (ROI) function is used to define a specific area within the image where the lane lines are expected to appear. This helps to focus the lane detection algorithm on the most relevant part of the image, reducing computational load and improving accuracy. The ROI function typically involves the following steps:

1. ROI Selection:

Define the region within the image where lane lines are expected to be present. This is often a trapezoidal or rectangular area covering the lower portion of the image. The exact dimensions and position of the ROI can vary depending on the specific camera placement and the expected road geometry.

2.Masking:

Use a masking technique to zero out pixels outside the defined ROI. This involves creating a binary mask that retains pixels within the ROI and sets pixels outside the ROI to zero (black). The masked image focuses only on the selected region, ignoring irrelevant parts of the image.

3.Applying the Mask:

Multiply the original image by the binary mask to retain only the pixels within the ROI while setting all other pixels to zero. This operation effectively crops the image to the defined ROI.

4.ROI Parameters:

The parameters defining the ROI (e.g., top width, bottom width, height) need to be carefully chosen based on the expected perspective and position of the camera relative to the road. Adjusting these parameters may require empirical tuning to achieve optimal lane detection performance under different road conditions.

- Takes the Canny edge-detected image (canny) as input.

- Gets the height and width of the image.

- Creates a black mask with the same dimensions as the Canny image.

- Defines a triangular region of interest (ROI) in the mask.

- Fills this triangular ROI with white color (255).

- Combines the Canny image with the mask, keeping only the edges within the defined region.

- Returns the resulting masked image.

3. HARDWARE AND SOFTWARE REQUIREMENTS

The Project is developed using Python as a language. We used PyCharm forDesign and coding of project. Managed database using Python server.

Minimum Hardware Requirement:

- i5 Processor Based Computer
- 8GB-Ram
- 320GB Solid State Disk
- Monitor
- Keyboard (input device)

Minimum Software Requirement:

- Windows 11
- Python
- Spyder
- OpenCV

Programming Language:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supportsmultiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system with reference counting.

Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 codedoes not run unmodified on Python 3.

The Python 2 language was officially discontinued in 2020 (first planned for 2015), and "Python 2.7.18 is the last Python 2.7 release and therefore the last Python 2 release."[30] No more security patchesor other

improvements will be released for it.[31][32] With Python 2's end-oflife, only Python 3.5.x[33] and later are supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source [34] reference implementation. A non-profitorganization, the Python Software Foundation, manages and directs resources for Python and Python development.

IDE: Vs Code

Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package. Furthermore, Spyder offers built-in integration with many popular scientific packages, including NumPy, SciPy, Pandas, IPython, QtConsole, Matplotlib, SymPy and more.

Beyond its many built-in features, Spyder's abilities can be extended even further via its plugin system and API. Spyder can also be used as a PyQt5 extension library, allowing you to build upon its functionality and embed its components, such as the interactive console, in your own software.

Features:-

Some of the remarkable features of Spyder are:

- Customizable Syntax Highlighting
- Availability of breakpoints (debugging and conditional breakpoints)
- Interactive execution which allows you to run line, file, cell, etc.
- Run configurations for working directory selections, command-line options, current/ dedicated/ external console, etc
- Can clear variables automatically (or enter debugging)
- Navigation through cells, functions, blocks, etc can be achieved through the Outline Explorer
- It provides real-time code introspection (The ability to examine what functions, keywords, and classes are, what they are doing and what information they contain)
- Automatic colon insertion after if, while, etc
- Supports all the IPython magic commands
- Inline display for graphics produced using Matplotlib
- Also provides features such as help, file explorer, find files, etc

Writing code in Spyder becomes very easy with its multi-language code editor and a number of powerful tools. As mentioned earlier, the editor has features such as syntax highlighting, real-time analysis of code, style analysis, on-demand completion, etc. When you write your code, you will also notice that it gives a clear call stack for methods suggesting all the arguments that can be used along with that method.

Plugins

Available plugins include:

- Spyder-Unittest, which integrates the popular unit testing frameworks Pytest, Unittest and Nose with Spyder
- Spyder-Notebook, allowing the viewing and editing of Jupyter Notebook within the IDE
  - Download Spyder Notebook
  - Using conda: *conda install spyder-notebook -c spyder-ide*
  - Using pip: *pip install spyder-notebook*
- Spyder-Reports, enabling use of literate programming techniques in Python
- Spyder-Terminal, adding the ability to open, control and manage cross-platform system shells within Spyder
  - Download Spyder Terminal
  - Using conda: *conda install spyder-terminal -c spyder-ide*
  - Using pip: *pip install spyder-terminal*
- Spyder-Vim, containing commands and shortcuts emulating the Vim text editor.
- Spyder-AutoPEP8, which can automatically conform code to the standard PEP 8 code style
- Spyder-Line-Profiler and Spyder-Memory-Profiler, extending the built-in profiling functionality to include testing an individual line, and measuring memory usage.

4. FLOW CONTROL

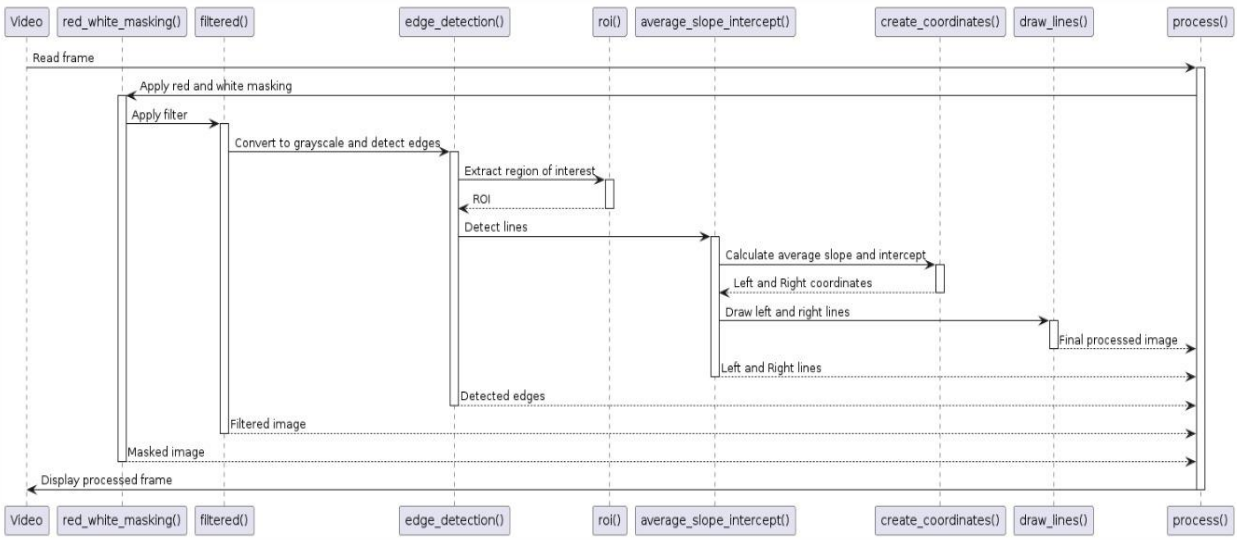

Fig.2. State diagram

Video -> Process: Read frame:

The Video participant (representing the video stream) sends a message to the Process participant (representing the processing function) to read a frame from the video.

Process -> RedWhiteMasking: Apply red and white masking:

The Process participant activates and then sends a message to the RedWhiteMasking participant (representing the function responsible for masking red and white colors) to apply the masking.

RedWhiteMasking -> Filtered: Apply filter:

The RedWhiteMasking participant activates and then sends a message to the Filtered participant (representing the function responsible for applying a filter) to apply the filter.

Filtered -> EdgeDetection: Convert to grayscale and detect edges:

The Filtered participant activates and then sends a message to the EdgeDetection participant (representing the function responsible for converting the image to grayscale and detecting edges) to perform edge detection.

EdgeDetection -> ROI: Extract region of interest:

The EdgeDetection participant activates and then sends a message to the ROI participant (representing the function responsible for extracting the region of interest) to extract the region of interest from the edge-detected image.

ROI --> EdgeDetection: ROI:

The ROI participant returns the region of interest back to the EdgeDetection participant.

EdgeDetection -> AverageSlopeIntercept: Detect lines:

The EdgeDetection participant sends a message to the AverageSlopeIntercept participant (representing the function responsible for detecting lines) to detect lines using the Hough Transform.

AverageSlopeIntercept -> CreateCoordinates: Calculate average slope and intercept:

The AverageSlopeIntercept participant activates and then sends a message to the CreateCoordinates participant (representing the function responsible for calculating the average slope and intercept) to calculate the average slope and intercept of the detected lines.

CreateCoordinates --> AverageSlopeIntercept: Left and Right coordinates:

The CreateCoordinates participant returns the left and right coordinates back to the AverageSlopeIntercept participant.

AverageSlopeIntercept -> DrawLines: Draw left and right lines:

The AverageSlopeIntercept participant sends a message to the DrawLines participant (representing the function responsible for drawing lines) to draw the left and right lines on the image.

DrawLines --> Process: Final processed image:

The DrawLines participant returns the final processed image back to the Process participant.

AverageSlopeIntercept --> Process: Left and Right lines:

The AverageSlopeIntercept participant returns the left and right lines information back to the Process participant.

EdgeDetection --> Process: Detected edges:

The EdgeDetection participant returns the detected edges back to the Process participant.

Filtered --> Process: Filtered image:

The Filtered participant returns the filtered image back to the Process participant.

RedWhiteMasking --> Process: Masked image:

The RedWhiteMasking participant returns the masked image back to the Process participant.

Process -> Video: Display processed frame:

The Process participant sends a message to the Video participant to display the processed frame.

Linear equation (line equation)

1. Linear equation (Line equation):
y= mx + b
Where:

- y is the dependent variable.
- x is the independent variable.
- m is the slope of the line.
- b is the y-intercept.

2. Average slope calculation:

- Average_slope = $\frac{\sum_{i=1}^{n} slope\ i}{n}$

Where:

- n is the number of lines,
- $slope_i$ is the slope of the i^{th} line.

3. Average intercept Calculation:

- Average_intercept = $\frac{\sum_{i=1}^{n} intercept\ i}{n}$

Where:

- n is the number of lines,
- intercept_i is the intercept of the i^{th} line.

4. Line equations from slope and intercept:

- $x=\frac{y-b}{m}$

Where:

- X and y are coordinates,
- M is the slope,
- B is the y-intercept.

5. SOURCE CODE

```
Import cv2 as  cv #Importing OpenCV library
Import numpy as np #importing numpy library for numerical operations
#Function to mask red and white colours in the image
Def red white masking(image):
#convert image from BGR to HSV colour space
Hsv= cv.cvtColor(image, cv.COLOR_BGR2HSV
# Define lower and upper bounds for yellow colour in HSV
lower_y = np.array([10, 130, 120], np.uint8)
upper_y = np.array([40, 255, 255], np.uint8)
# Create a mask for the yellow color
mask_y = cv.inRange(hsv, lower_y, upper_y)
# Define lower and upper bounds for white colour in HSV
lower_w = np.array([0, 0, 212], np.uint8)
upper_w = np.array([170, 200, 255], np.uint8)
# Create a mask for white colour
mask_w = cv.inRange(hsv, lower_w, upper_w)
# Combine masks for yellow and white
mask = cv.bitwise_or(mask_w, mask_y)
# Apply the mask to the original image
masked_bgr = cv.bitwise_and(image, image, mask=mask)
return masked_bgr

# Function to apply a filter to the image
def filtered(image):
# Define a kernel for the filter
kernel = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
# Apply the filter to the image
filtered_image = cv.filter2D(image, -1, kernel)
return filtered_image
```

```
    # Function to define region of interest (ROI) in the image
    def roi(image, vert, color=[255, 255, 255]):
    # Create a mask of zeros with the same shape as the image. We use function np.zeros_like()
    mask = np.zeros_like(image)
    # Fill the convex polygon defined by 'vert' with the specified 'colour'
    cv.fillConvexPoly(mask, cv.convexHull(vert), color)
    # Apply the mask to the image
    masked_image = cv.bitwise_and(image, mask)
    return masked_image


# Function for edge detection using Canny edge detector
def edge_detection(image):
# Apply Canny edge detection
edges = cv.Canny(image, 80, 200)
return edges
# Function to calculate average slope and intercept of detected lines
def average_slope_intercept(image, lines):
left_fit = []  # Store parameters of left lane lines
right_fit = []  # Store parameters of right lane lines
for line in lines:
x1, y1, x2, y2 = line.reshape(4)
# Fit a line (y = mx + b) to the detected points
parameters = np.polyfit((x1, x2), (y1, y2), 1)
slope = parameters[0]
intercept = parameters[1]

    # Classify lines as left or right based on slope
    if slope < 0:
    left_fit.append((slope, intercept))
```

```
    else:
    right_fit.append((slope, intercept))

    # Calculate the average slope and intercept for left and right lines
    left_fit_average = np.average(left_fit, axis=0)
    right_fit_average = np.average(right_fit, axis=0)

   # Generate coordinates for left and right lines based on average slope and intercept
   left_line = create_coordinates(image, left_fit_average)
  right_line = create_coordinates(image, right_fit_average)

  return np.array([left_line, right_line])


  # Function to generate coordinates based on slope and intercept
  def create_coordinates(image, line_parameters):
  slope, intercept = line_parameters
  y1 = image.shape[0]  # Bottom of the image
  y2 = int(y1 * (2 / 3))  # 2/3rd height of the image
  x1 = int((y1 - intercept) / slope)
 x2 = int((y2 - intercept) / slope)
 return np.array([x1, y1, x2, y2])


 # Function to draw lines on the image
def draw_lines(left, right, image):
 # Draw left and right lines on the image
```

```
 cv.line(image, (left[0], left[1]), (left[2], left[3]), (0, 0, 255), 5)
 cv.line(image, (right[0], right[1]), (right[2], right[3]), (0, 0, 255), 5)

 # Define vertices for the region of interest
 vert = np.array([[(left[0], left[1]), (left[2], left[3]), (right[0], right[1]), (right[2], right[3])]])

 # Extract the region of interest
 cropped_lane = roi(image, vert, color=[0, 255, 255])

 # Overlay the region of interest on the original image
 detected_image = cv.addWeighted(image, 1, cropped_lane, 0.7, 0)

 return detected_image


# Function to process each frame of the video
def process(image):
h = image.shape[0]  # Height of the image
w = image.shape[1]  # Width of the image

# Apply red and white colour masking
masked = red_white_masking(image)

# Apply filter to the masked image
blurred = filtered(masked)

# Convert the filtered image to grayscale
```

```
gray = cv.cvtColor(blurred, cv.COLOR_BGR2GRAY)

# Define vertices for the region of interest
vertices = np.array([[(0, h), (w / 2, h / 2), (w, h)]], np.uint64)

# Extract region of interest
region_of_interest = roi(gray, vert=vertices)

# Detect edges in the region of interest
edges_detect = edge_detection(region_of_interest)

# Detect lines using Hough Transform
lines = cv.HoughLinesP(edges_detect, 1, np.pi / 180, 20,   miniLineLength=maxLineGap=200
# Calculate the average slope and intercept of detected lines
left_lane, right_lane = average_slope_intercept(image, lines)

# Draw lines on the image
final_image = draw_lines(left_lane, right_lane, image.copy())

return final_image


# Open video file
cap = cv.VideoCapture('nh33.mp4')

# Create a resizable window
cv.namedWindow('Video', cv.WINDOW_NORMAL)
```

```
 cv.resizeWindow('Video', 800, 600)


 # Process each frame of the video
 while(cap.isOpened()):
 ret, frame = cap.read()


 if ret == False:
cap.set(cv.CAP_PROP_POS_FRAMES, 0)
_, frame = cap.read()


 # Process the frame
 detected = process(frame)


 # Display the processed frame
 cv.imshow('Video', detected)


 # Break the loop if 'q' is pressed
 if cv.waitKey(1) & 0xFF == ord('q'):
 break


# now release the video capture and close all windows
cap.release()
cv.destroyAllW
```

6. CODE EXPLANATION

 # Libraries Imported

- cv2 (OpenCV): Used for image and video processing.

- numpy (np): Used for numerical operations.

 # Functions Defined

1. *red_white_masking(image)*:

- Converts the input BGR image to HSV color space.

- Masks out red and white colors using predefined HSV ranges.

- Returns the masked BGR image.

2. *filtered(image)*:

- Applies a specific kernel ([-1, 0, 1]) for filtering along the horizontal direction.

- Returns the filtered image.

3. *roi(image, vert, color=[255, 255, 255])*:

- Defines a region of interest (ROI) in the image using a convex polygon.

- Masks out the region outside the defined polygon.

- Returns the masked image.

4. *edge_detection(image)*:

- Applies Canny edge detection to the input image.

- Returns the edge-detected image.

5. *average_slope_intercept(image, lines)*:

- Calculates the average slope and intercept of detected lines.

- Classifies lines as left or right based on slope.

- Returns coordinates for left and right lane lines.

6. *create_coordinates(image, line_parameters)*:

- Generates coordinates (x1, y1, x2, y2) based on slope and intercept.

7. *draw_lines(left, right, image)*:

- Draws left and right lane lines on the image.

- Defines vertices for the region of interest and extracts it.

- Overlays the region of interest on the original image.

8. *process(image)*:

- Main function to process each frame of the video.

- Applies color masking, filtering, grayscale conversion, and region of interest extraction.

- Detects edges and lines using Canny edge detection and Hough transform.

- Calculates and draws lane lines on the image.

 # Video Processing Loop

- *Opening Video File*:

- Reads the input video file ('nh33.mp4') using cv.VideoCapture.

- *Frame Processing*:

- Iterates over each frame of the video using cap.read().

- If end of video (ret == False), resets to the beginning of the video.

- Calls process(frame) to process each frame.

- *Displaying Results*:

- Shows the processed frame in a resizable window named 'Video' using cv.imshow.

- Breaks the loop if 'q' is pressed (cv.waitKey(1) & 0xFF == ord('q')).

- *Cleanup*:

- Releases the video capture (cap.release()) and closes all windows (cv.destroyAllWindows)

7. OUTPUT

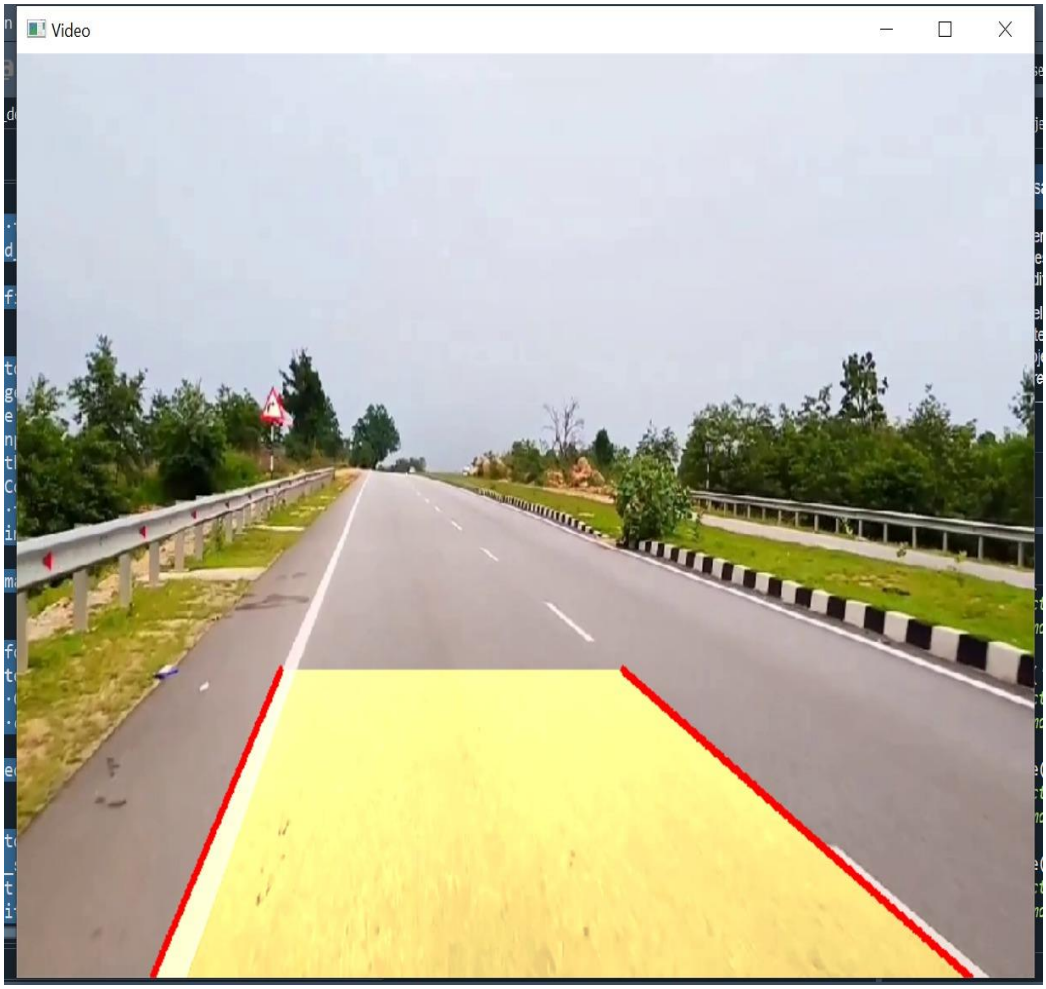

Fig.3. Output

SUMMARY

This code processes a video frame by frame for lane detection:

- It performs color masking to isolate yellow and white lane markings.

- Filters and extracts edges using Canny edge detection.

- Identifies and averages lane lines using the Hough transform.

- Draws lane lines on the original frame and displays the processed video.

Make sure to have the necessary video file ('nh33.mp4') in the same directory as the script or provide the correct path to the video file for this code to run successfully. Adjust parameters and functions as needed based on specific lane detection requirements and video characteristic

8. CONCLUSION

First, we need to decide what features to use. We'll want to try some combination of colour and gradient based features. But keepin mind that this might require some experimentation to decide what works best. Next, we'll need to choose and train a classifier. A linear SVM is probably our best bet for an ideal combination of speed and accuracy. We've chosen features and trained a classifier. Next, we'll implement a sliding window technique to search for vehicles in some test images. We can try multiscale search or different tiling schemes to see what works best. But keep in mind, we'd like to minimize the number of search windows. So, for example, we probably don't need to search for cars in the sky and thetreetops. Once we've got a working detection pipeline, we'll try iton a video stream. And implement tracking to follow, detect vehicles. And reject spurious detections.

9. BIBLIOGRAPHY

For books:

Driverless: Intelligent Cars and the Road Ahead (MIT Press) 1St Ed

For Website:

- www.geeksforgeeks.com
- https://medium.com/driveupply/selfdrive-101-everythingyou-ever-wanted-to-know-about-selfdrive-478c0b825dd0
- www.youtube.com
- https://pyimagesearch.com/2019/12/02/opencv-vehicle-detection-tracking-and-speed-estimation/ https://pypi.org/project/pyttsx3/
- https://en.wikipedia.org/wiki/Selfdriving
- https://pypi.org/project/opencv-python/