

# Artificial Intelligence

## Lab Project Submission

Submitted to: Ms. Palak Arora

Submitted by:

Name	Roll Number
102103327	Sumit Satija
102103321	Shivansh Gupta
102103345	Bhavya Bhalla

---

### Project Title:

Lane Line Detection System

---

### Description of problem:

This project is about lane line detection using computer vision and machine learning techniques. The goal is to develop a system that can accurately detect lane lines on a road in real-time using a camera mounted on a vehicle.

The project uses a combination of camera calibration, thresholding, perspective transformation, and machine learning techniques to achieve the goal. The camera calibration is used to correct for distortion in the camera's view, while

the thresholding is used to identify the lane lines in the image. The perspective transformation is used to change the viewpoint of the image to a bird's-eye view, which makes it easier to detect the lane lines. Finally, machine learning techniques are used to fit a model to the detected lane lines and predict their positions in the image.

The system can process both images and videos, and the output is a video or image with the detected lane lines overlaid on the original image. The output can also include additional information such as the lane curvature and the distance of the vehicle from the lane center. The system can handle different types of roads, such as curved roads and roads with different lighting conditions.

The project aims to improve road safety by providing an automated system that can assist drivers in maintaining their lane positions on the road.

---

## Approach:

The approach used in this project for lane line detection involves a combination of computer vision techniques, such as camera calibration, perspective transformation, and thresholding, along with machine learning algorithms, specifically the sliding window technique and polynomial curve fitting.

The first step involves calibrating the camera to remove any distortions in the image, followed by a perspective transformation to obtain a bird's eye view of the road. Then, thresholding techniques are applied to enhance the lane lines in the image.

After that, sliding windows are used to detect the lane lines and then a polynomial curve fitting algorithm is applied to determine the curvature of the lanes. The lane boundaries are then drawn onto the original image to visualize the results.

Finally, the algorithm is applied to a video, frame by frame, to track the lane lines

in real-time. Any deviations from the expected lane positions are flagged, and the driver can be alerted to take corrective actions if necessary.

## Code and Explanation:

### Importing Necessary Libraries and Classes:-

```
import numpy as np
import matplotlib.image as mpimg
import cv2
from docopt import docopt
from IPython.display import HTML
from IPython.core.display import Video
from moviepy.editor import VideoFileClip
from CameraCalibration import CameraCalibration
from Thresholding import *
from PerspectiveTransformation import *
from LaneLines import *
```

The project utilizes various Python libraries and classes to process images and videos and detect lane lines.

**Numpy** library is used to manipulate arrays and perform numerical operations on them.

**Matplotlib. Image** is used to read and display images.

**CV2** provides various computer vision and image processing functions, including image transformation and thresholding.

**Docopt** is used to parse command-line arguments and options.

**IPython.display** is used to display multimedia content, including images and videos.

**Moviepy.editor** is used to edit and process video files.

The code imports four classes from separate Python files, each with a specific purpose in the lane line detection process.

**CameraCalibration** performs camera calibration to correct for image distortion caused by the camera lens.

**Thresholding** performs thresholding operations to filter out noise and enhance the lane lines.

**PerspectiveTransformation** performs perspective transformation to obtain a bird's-eye view of the road and lane lines, making it easier to detect lane lines.

**LaneLines** performs lane line detection and visualization by fitting a polynomial curve to the lane lines and drawing them onto the original image.

---

```
class FindLaneLines:
    def __init__(self):
        """ Init Application """
        self.calibration = CameraCalibration('camera_cal', 9, 6)
        self.thresholding = Thresholding()
        self.transform = PerspectiveTransformation()
        self.lanelines = LaneLines()
```

The `__init__` function is the constructor for the `FindLaneLines` class. When an instance of this class is created, this function is automatically called to initialize its attributes.

First, the `CameraCalibration` object is created with the folder name 'camera\_cal' and the number of corners in each calibration chessboard image (9x6). This object is assigned to the `calibration` attribute of the `FindLaneLines` instance. The purpose of camera calibration is to correct for the distortion that occurs when images are captured by a camera lens. This is important for accurate lane line detection.

Next, a `Thresholding` object is created and assigned to the `thresholding` attribute of the `FindLaneLines` instance. The `Thresholding` class is responsible for converting the image into a binary format by applying color and gradient thresholds, which helps filter out noise and enhance the lane lines.

Then, a `PerspectiveTransformation` object is created and assigned to the `transform` attribute of the `FindLaneLines` instance. This class transforms the

image from the normal camera view to a bird's-eye view, allowing us to see the lane lines more clearly.

Finally, a LaneLines object is created and assigned to the lanelines attribute of the FindLaneLines instance. This class is responsible for detecting the lane lines in the transformed image and drawing them onto the original image.

Overall, the `__init__` function sets up the necessary components for lane line detection and visualization in the subsequent functions of the FindLaneLines class.

---

```
def forward(self, img):  
    out_img = np.copy(img)  
    img = self.calibration.undistort(img)  
    img = self.transform.forward(img)  
    img = self.thresholding.forward(img)  
    img = self.lanelines.forward(img)  
    img = self.transform.backward(img)  
    out_img = cv2.addWeighted(out_img, 1, img, 0.6, 0)  
    out_img = self.lanelines.plot(out_img)  
    return out_img
```

The "forward" method in the "FindLaneLines" class takes an input image and performs a series of operations on it to detect and visualize the lane lines.

First, it creates a copy of the original image "img" and saves it as "out\_img". Then, it undistorts the image using the camera calibration parameters obtained in the initialization step. Next, it applies perspective transformation to obtain a bird's-eye view of the road and lane lines. Then, it applies thresholding to filter out noise and enhance the lane lines. After that, it uses the "LaneLines" class to detect the lane lines in the thresholded image.

Once the lane lines are detected, the method performs a backward perspective transformation to bring the image back to the original perspective. It then overlays the detected lane lines on the original image using the

"cv2.addWeighted" function. Finally, it plots the lane lines and their curvature information using the "self.lanelines.plot" method.

The "forward" method is used in the "process\_image" and "process\_video" methods to detect and visualize the lane lines in a single image or a sequence of images (video).

---

```
def process_image(self, input_path, output_path):
```

```
    img = mpimg.imread(input_path)
```

```
    out_img = self.forward(img)
```

```
    mpimg.imsave(output_path, out_img)
```

The "process\_image" function of the "FindLaneLines" class is responsible for processing a single input image and generating the output image with the detected lane lines. It takes two input arguments - the path of the input image and the path for the output image where the generated image will be saved.

Initially, the method reads the input image using the "mpimg.imread" function and stores it in a variable. Then, it calls the "forward" method of the same class to detect and visualize the lane lines in the image. The "forward" method applies a series of image processing techniques, such as camera calibration, perspective transformation, thresholding, and lane line detection, to generate an image with the lane lines highlighted.

After obtaining the processed image with the lane lines, the "process\_image" method saves the image using the "mpimg.imsave" function. The output path for the image is provided as an argument to the method.

The "process\_image" method is useful for testing and validating the lane detection algorithm on individual images. It allows the user to process a single image and visualize the detected lane lines, which can be used to evaluate the performance of the algorithm. The method can be called multiple times on different images to generate outputs for each of them.

---

```
def process_video(self, input_path, output_path):
```

```
    clip = VideoFileClip(input_path)
```

```
out_clip = clip.fl_image(self.forward)

out_clip.write_videofile(output_path, audio=False)
```

The "process\_video" method in the "FindLaneLines" class takes the path of an input video and the desired path for the output video as input. It uses the "VideoFileClip" function from the "moviepy.editor" module to read the input video and obtain a Clip object. Then, it applies the "forward" method to each frame of the video using the "fl\_image" method, which processes the image frame-by-frame and outputs the frames with the detected lane lines. The resulting frames are then combined into an output video using the "write\_videofile" method, with the output path specified by the user. This method allows the user to analyze and test the performance of the lane detection algorithm on a continuous stream of video data. It can be used for various purposes such as evaluating the accuracy and stability of the algorithm under different lighting and weather conditions, testing the algorithm's ability to detect different lane configurations and obstacles, and comparing the performance of different lane detection algorithms. The "process\_video" method provides a powerful tool for lane detection and analysis in a real-world driving scenario.

---

```
def main():

    findLaneLines = FindLaneLines()

    findLaneLines.process_video("challenge_video.mp4","output.mp4")

if __name__ == "__main__":

    main()
```

The main function is the entry point of the Python script. It creates an instance of the FindLaneLines class and calls its process\_video method, passing the path to the input video file and the path to the output video file as arguments.

Inside the process\_video method, the VideoFileClip function from the moviepy.editor module is used to load the input video file.

Then, the fl\_image method is called on the clip object, with the forward method of the FindLaneLines class passed as an argument. This applies the lane detection algorithm to each frame of the input video and returns the processed image.

Finally, the processed video is written to a file using the `write_videofile` method of the `out_clip` object, with the path to the output video file and the argument `audio=False`. The audio argument is set to False because the input video does not contain an audio track.

Overall, the main function provides a simple interface for applying the lane detection algorithm to an input video file and saving the output to a new video file.

---

## Output:

The output of the lane detection project can be either a single image or a video file with the detected lane lines overlaid on the original input frames.

If the input is a single image, the output will be a new image with the detected lane lines drawn on it, along with additional information such as the lane curvature and vehicle position. The detected lane lines are represented by two polynomials that are fit to the left and right lane markings. These polynomials are drawn onto the image as solid lines, along with a shaded area between them to represent the lane.

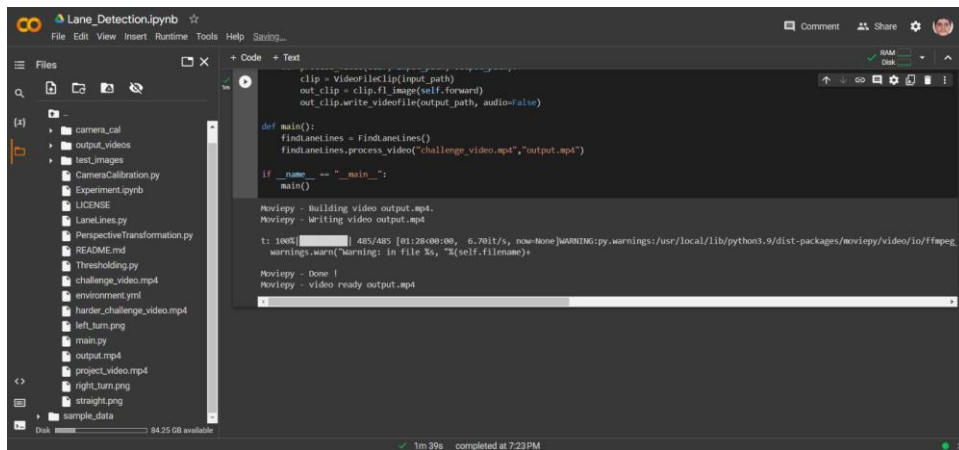
If the input is a video file, the output will be a new video file with the same frame rate and dimensions as the input, but with the detected lane lines overlaid on each frame. The detected lane lines are updated for each frame, based on the current lane markings and the position of the vehicle within the lane. The resulting video can be used to visualize the performance of the lane detection algorithm and analyze the behavior of the vehicle on the road.

Overall, the output of the lane detection project provides a visual representation of the detected lane lines, which can be used for a variety of purposes, such as improving the safety and efficiency of autonomous vehicles, or assisting human drivers in navigating complex roadways.

Snapshot of our code's output:-

```
MoviePy - Building video output.mp4.  
MoviePy - Writing audio in outputTEMP_MPV_wvf_snd.mp3  
MoviePy - Done.  
MoviePy - Writing video output.mp4  
t: 100% [██████████] 485/485 [01:28:00:00, 3.32it/s, now=None]WARNING:py.warnings:/usr/local/lib/python3.9/dist-packages/moviepy/video/io/ffmpeg_reader.py:123:  
warnings.warn("warning: in file %s, %s"%(self.filename))  
MoviePy - Done !  
MoviePy - video ready output.mp4
```





```
clip = VideoFileClip(input_path)
out_clip = clip.fl_image(self.forward)
out_clip.write_videofile(output_path, audio=False)

def main():
    find_lane_lines = find_lane_lines()
    find_lane_lines.process_video("challenge_video.mp4", "output.mp4")

if __name__ == "__main__":
    main()

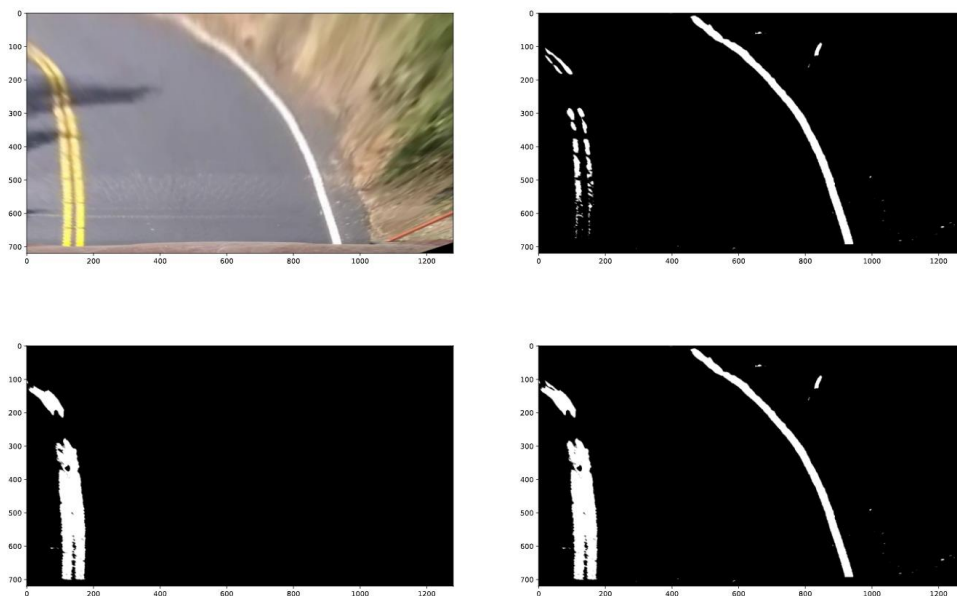
MoviePy - Building video output.mp4.
MoviePy - Writing video output.mp4
t: 100% [ 485/485 [01:28:00:00, 6.70it/s, now=None] WARNING:py.warnings:/usr/local/lib/python3.9/dist-packages/moviepy/video/io/ffmpeg_w
warnings.warn("warning: in file %s, %s(self.filename)"

MoviePy - Done !
MoviePy - video ready output.mp4
```

Link to the output video:-

<https://drive.google.com/file/d/1hWmoWAdLRQxy4GFS2OjjKhkf1OEAVBuA/view?usp=sharing>

Images representing the working of our project:-



[https://colab.research.google.com/drive/1Gi-6Plu4Q8n\\_cGOPjcQuWnAbuHwHsWr6#scrollTo=EP73rGIBlexM](https://colab.research.google.com/drive/1Gi-6Plu4Q8n_cGOPjcQuWnAbuHwHsWr6#scrollTo=EP73rGIBlexM)

