**Lane Detection using OpenCV**

Lane detection is a critical computer vision task with applications in various fields, including autonomous driving, advanced driver assistance systems (ADAS), and robotics. The primary objective of lane detection is to identify and track the lanes on a road or a similar environment, enabling vehicles or robots to navigate within their designated lanes. This task is essential for ensuring safe and reliable transportation. OpenCV, an open-source computer vision library, provides a rich set of tools and functions for performing this task.

OpenCV provides various functions and tools to implement each of these steps, making it a popular choice for lane detection tasks. However, the specific implementation details may vary depending on the exact requirements of your application and the environment in which it operates.

**Architecture:**

A diagram of a video production process

Description automatically generated

**Pre-requisites:**

**1. To complete this project you must have the following software versions and packages.**

* PyCharm ( Download: https://www.jetbrains.com/pycharm/ )
* Python 3.7.0 (Download: https://www.python.org/downloads/release/python-370/ )
* Numpy ● cv2

**2. To make a responsive python script you must require the following packages.**

**Flask:**

* Web framework used for building web applications.
* Flask Basics: [Click](https://www.youtube.com/watch?v=lj4I_CvBnt0) [here](https://www.youtube.com/watch?v=lj4I_CvBnt0)

If you are using anaconda navigator, follow below steps to download required packages:

* Open anaconda prompt.
* Type "pip install opencv-python” and click enter.
* Type "pip installopencv-python” and click enter. ● Type “pip install Flask” and click enter.

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

**Project objectives:**

By the end of this project you will:

* Know fundamental concepts and techniques of computer vision (openCV).
* Gain a broad understanding of image thresholding.

**Project Flow:**

* Data Collection o Download the dataset
* ROI (Region of interest) o Create python file o Import required libraries o Define ROI width and height o Select and deselect ROI o Denote ROI with BBOX
* Video Processing and object detection o Import required libraries o Reading input and loading ROI file o Checking for parking space o Looping the video o Frame processing and empty parking slot counters
* IBM Database Connection
  + Create IBM DB2 service and table
* Application building o Build HTML
  + Build python script for Flask

**Project Structure:**

Create a project folder which contains files as shown below:

A screenshot of a computer

Description automatically generated

* The Dataset folder contains the video and image file
* For building a Flask Application we needs HTML pages stored in the **templates** folder,CSS for styling the pages stored in the static folder and a python script **app.py** for server side scripting

# Milestone – 1: Data Collection Activity 1: Download the dataset

Click the below link to download dataset for lane detection using OpenCV. [Click](https://drive.google.com/file/d/1EWrzolzyulJ2P0GAvDQRuahnoF1i18Qn/view?usp=sharing) [here](https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr?usp=share_link).

# Milestone – 2: ROI (Region of interest)

ROI stands for Region of Interest, which refers to a specific rectangular lines of an image or video frame that is used for processing or analysis.

## Activity 1: Create a python file

Create a python file in the directory and name it as ‘lane.py’. This python file is used for creating ROI and deleting ROI.

## Activity 2: Importing the required packages:

* **Opencv:** OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including python, java C++.

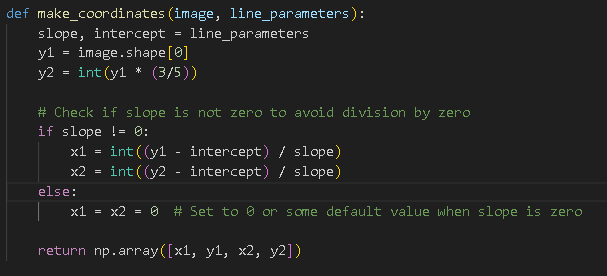
* **Numpy:** NumPy is a fundamental Python library used for numerical and array-based operations. It provides support for working with large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays



## Activity 3: Make Co-ordinates

## Calculating the coordinates of the two endpoints of a line based on the slope and intercept of the line (x1, y1, x2, y2 ).

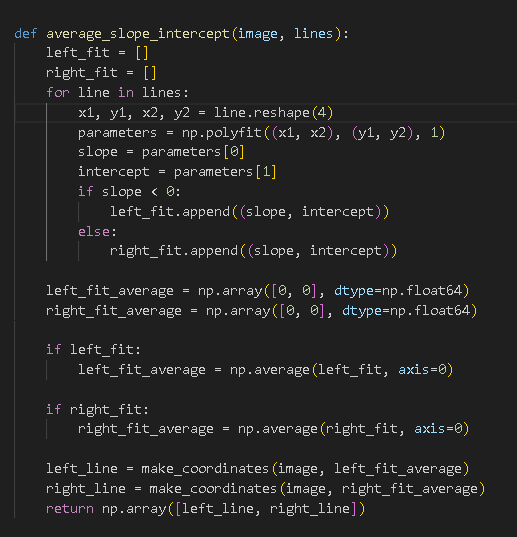
* Extract the slope and intercept from line\_parameters.
* Get the height (y1) of the image.
* Calculate the y2 value as 3/5 of the image height.
* Check if the slope is not zero to avoid division by zero.
* Calculate the x1 and x2 coordinates based on the slope and intercept.
* If the slope is zero, set x1 and x2 to a default value (in this case, both to 0).



## Activity 4: Define Average slope Intercept.

## This function averages and extrapolates the detected lane lines based on the slope and intercept of each line segment.

* Initialize empty lists left\_fit and right\_fit to store the slopes and intercepts of the left and right lane lines.
* Loop through each line segment in the lines list:
* Extract the coordinates of the line segment as x1, y1, x2, y2.
* Use np.polyfit to calculate the line parameters (slope and intercept) based on the line segment.
* Determine if the line corresponds to the left lane (if slope < 0) or the right lane (if slope >= 0).
* Append the line parameters to the respective lists: left\_fit for left lanes and right\_fit for right lanes.
* Initialize left\_fit\_average and right\_fit\_average as NumPy arrays with zeros.
* If there are elements in left\_fit and right\_fit, calculate the average slope and intercept for both lanes using np.average.
* Use the make\_coordinates function to calculate the coordinates of the two endpoints for both left and right lanes based on the averaged parameters.
* Return a NumPy array with two rows, each representing the coordinates of the left and right lane lines.



## Activity 5: Define Canny Edges

## This function applies the Canny edge detection algorithm to an input image to detect edges, which is a crucial step in many computer vision and image processing tasks, including lane detection.

* Convert the input image from the RGB color space to grayscale using cv2.cvtColor.
* Apply Gaussian smoothing to the grayscale image using cv2.GaussianBlur to reduce noise and make the edges more distinguishable.
* Use the Canny edge detection algorithm with specified low and high threshold values (50 and 150 in this case) to detect edges in the smoothed image.
* Return the resulting binary image where edges are represented by white pixels and non-edges are represented by black pixels.

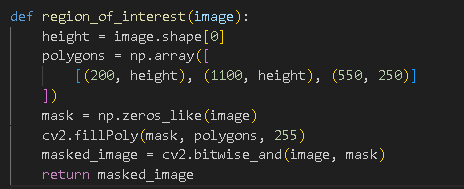
A screenshot of a computer screen

Description automatically generated

## Activity 6: Define Canny Edges

**This function creates a region of interest (ROI) mask on the input image to focus on a specific area within the image. It is commonly used in computer vision tasks, such as lane detection, to isolate the region where the lanes are expected to be.**

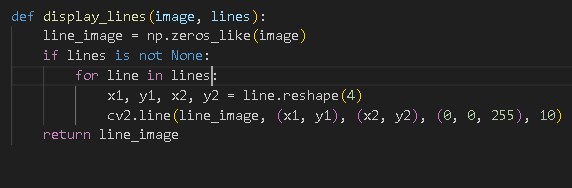
* Get the height of the input image using image.shape[0], which is typically the image's vertical dimension.
* Define a polygon representing the region of interest. In this case, it's a triangle with vertices at (200, height), (1100, height), and (550, 250).
* Create an empty binary mask (mask) with the same dimensions as the input image using np.zeros\_like(image). This mask is initialized with all black pixels.
* Fill the defined polygon (region of interest) with white (255) in the binary mask using cv2.fillPoly. This effectively creates a white area within the mask, corresponding to the region of interest.
* Use cv2.bitwise\_and to apply the mask to the input image. Pixels outside the region of interest become black (0), and pixels within the region of interest retain their original values.
* Return the masked image, which isolates the region of interest from the input image.



## Activity 7: Display Lines

## This function takes an input image and a set of line segments and draws the line segments on a new image to visualize them. It is commonly used in computer vision tasks, such as lane detection, to display detected lane lines on an imageCreate an empty image (line\_image) with the same dimensions as the input image using np.zeros\_like(image). This initializes the image with all black pixels.

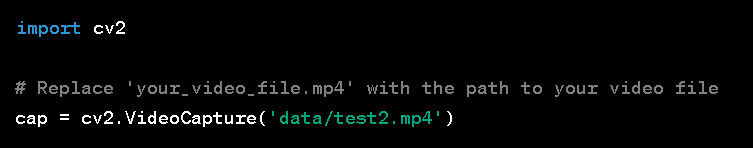
* Check if the lines parameter is not None.
* If there are line segments to be drawn, iterate through each line in the lines list:
* Extract the coordinates of the line segment as x1, y1, x2, y2.
* Use cv2.line to draw the line segment on the line\_image with a specified color (0, 0, 255) and line thickness 10. You can adjust the color and line thickness as needed.
* Return the line\_image with the drawn line segments.



# Milestone - 3: Video processing and Object detection

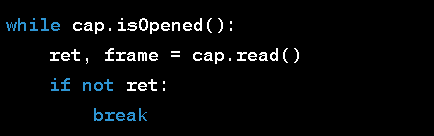
**Activity 1: Opening a Video File:**

The script uses OpenCV to open a video file named 'data/test2.mp4'. You should replace this file path with the path to your video file. The video file is opened using cv2.VideoCapture.



**Activity2: Video Processing Loop:**

The script enters a loop that processes each frame of the video as long as the video is open (cap.isOpened()).



**Activity 3: Reading Frames:**

Inside the loop, it reads the next frame from the video using cap.read(). The variable frame contains the current frame, and ret indicates whether the frame was successfully read.

**Activity 4: Lane Detection Pipeline:**

The following steps are performed for each frame:

* A copy of the current frame is created and stored in lane\_image.
* The Canny edge detection algorithm is applied to lane\_image to detect edges, resulting in the canny\_image.
* A region of interest (ROI) is defined and applied to the canny\_image to focus on a specific area of interest.
* The Hough Line Transform is used to detect lines in the cropped image (cropped\_image), and the detected lines are stored in the lines variable.
* The average\_slope\_intercept function is called to average and extrapolate the detected lane lines.
* The display\_lines function is used to draw the detected lane lines on the lane\_image, creating the line\_image.
* The combo\_image is created by overlaying line\_image on top of the original lane\_image to visualize the detected lane lines.

A screenshot of a computer program

Description automatically generated

**Activity 5: Displaying the Result:**

The combo\_image is displayed in a window with the title "result" using cv2.imshow.

A black screen with white text and pink letters

Description automatically generated

**Activity 6: Exiting the Loop:**

The script continues processing frames until you press the 'q' key. When 'q' is pressed (cv2.waitKey(1) & 0xFF == ord('q')), the loop exits.

After the loop, the video capture is released using cap.release() to free up system resources, and all OpenCV windows are closed using cv2.destroyAllWindows().

This script essentially processes a video file to detect and visualize lane lines, making it a common component of lane detection systems used in autonomous vehicles and advanced driver assistance systems (ADAS).

A black background with white text

Description automatically generated

# Milestone - 5: Application building

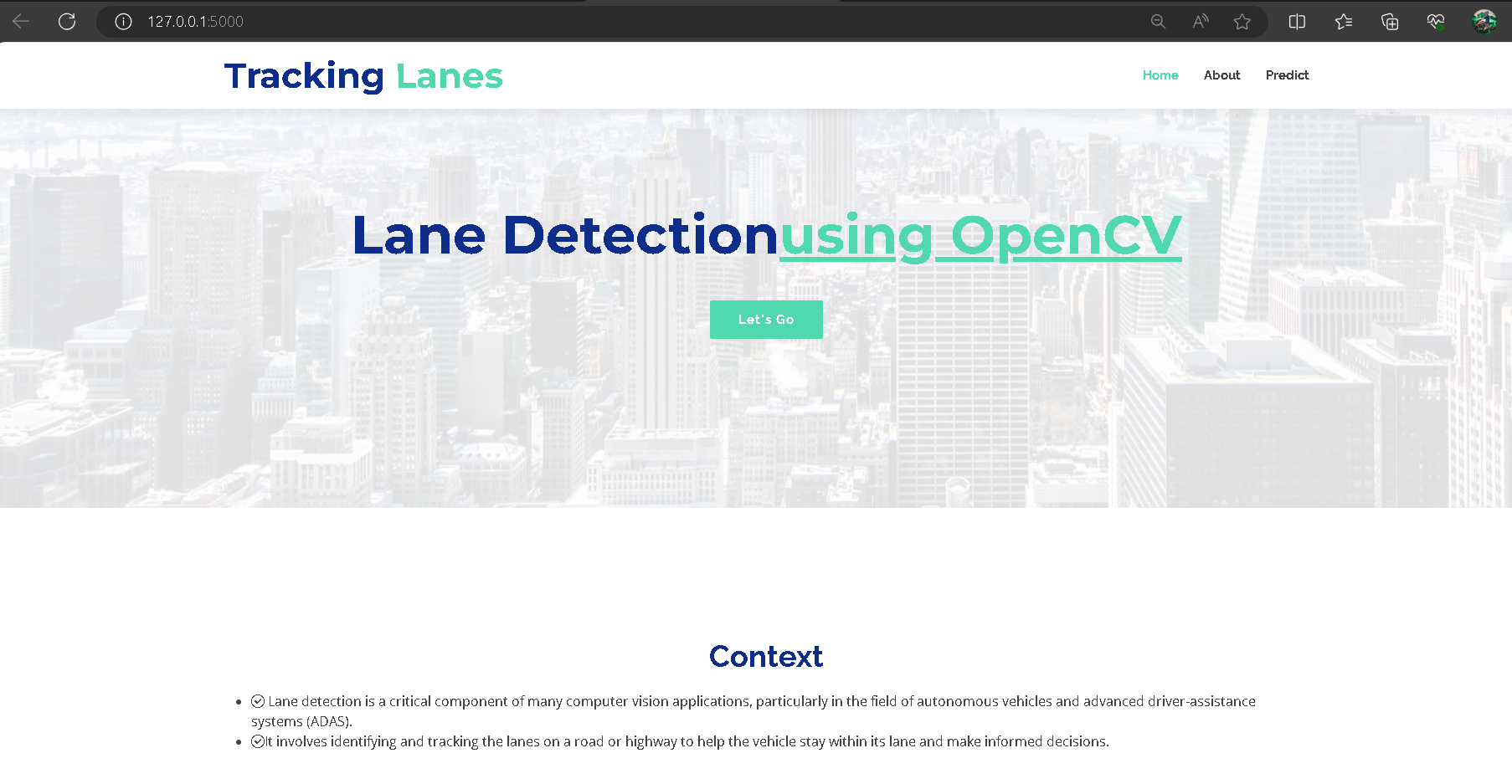
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he/she has to navigate to detect button. Then the video will be showcased on the UI.

This section has the following tasks

* Building HTML Pages
* Building server-side script

**Activity1: Building Html Pages:**

For this project we have created 1 HTML files and saved them in the templates folder. Let’s see how those html pages looks like:



A screenshot of a computer screen

Description automatically generated

**Activity 2: Build Python code:**

* Import the libraries
* Importing the flask module into the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (\_\_name\_\_) as an argument.

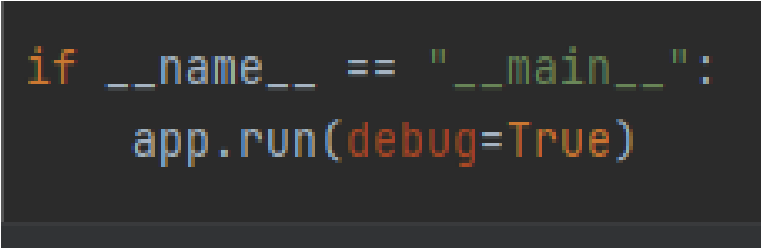


Render HTML page:

* Here we will be using the declared constructor to route to the HTML page that we have created earlier. In the above example, the ‘/’ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered.

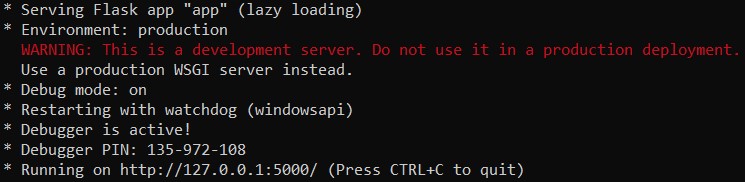


* Create a decorator to route to ‘/process\_video’. Go to the milestone 3 and copy the entire code and paste it inside the function processvideo.
* Main Function: Used to run the current module.



## Activity 3: Run the application

* Open the anaconda prompt from the start menu
* Navigate to the folder where your python script is.
* Now type the “python app.py” command
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.



**Output:**

Click on the ‘Click here’ button.



A road with green lines and mountains in the background

Description automatically generated