**Path Tracking for AI-Lite**

**Objective:**

This project implements an autonomous Lane/Line following system that uses computer vision techniques to detect and follow lanes. The system consists of two main components: an ESP32-CAM providing a video stream and an ESP32-based robot that receives steering commands. The application processes real-time video input to identify lane boundaries, calculates appropriate steering angles, and sends control commands to the robot, enabling it to navigate autonomously along a designated path.

**Components Used:**

I) **Hardware:**

* ESP32-CAM
* DOIT ESP32 DEVKIT V1
* Motors & Motor Driver
* Power Supply (Lithium Battery)
* Chassis with Wheels

II) **Software:**

* OpenCV
* NumPy
* Socket
* Requests
* Time
* Arduino IDE
* Anaconda
* Python

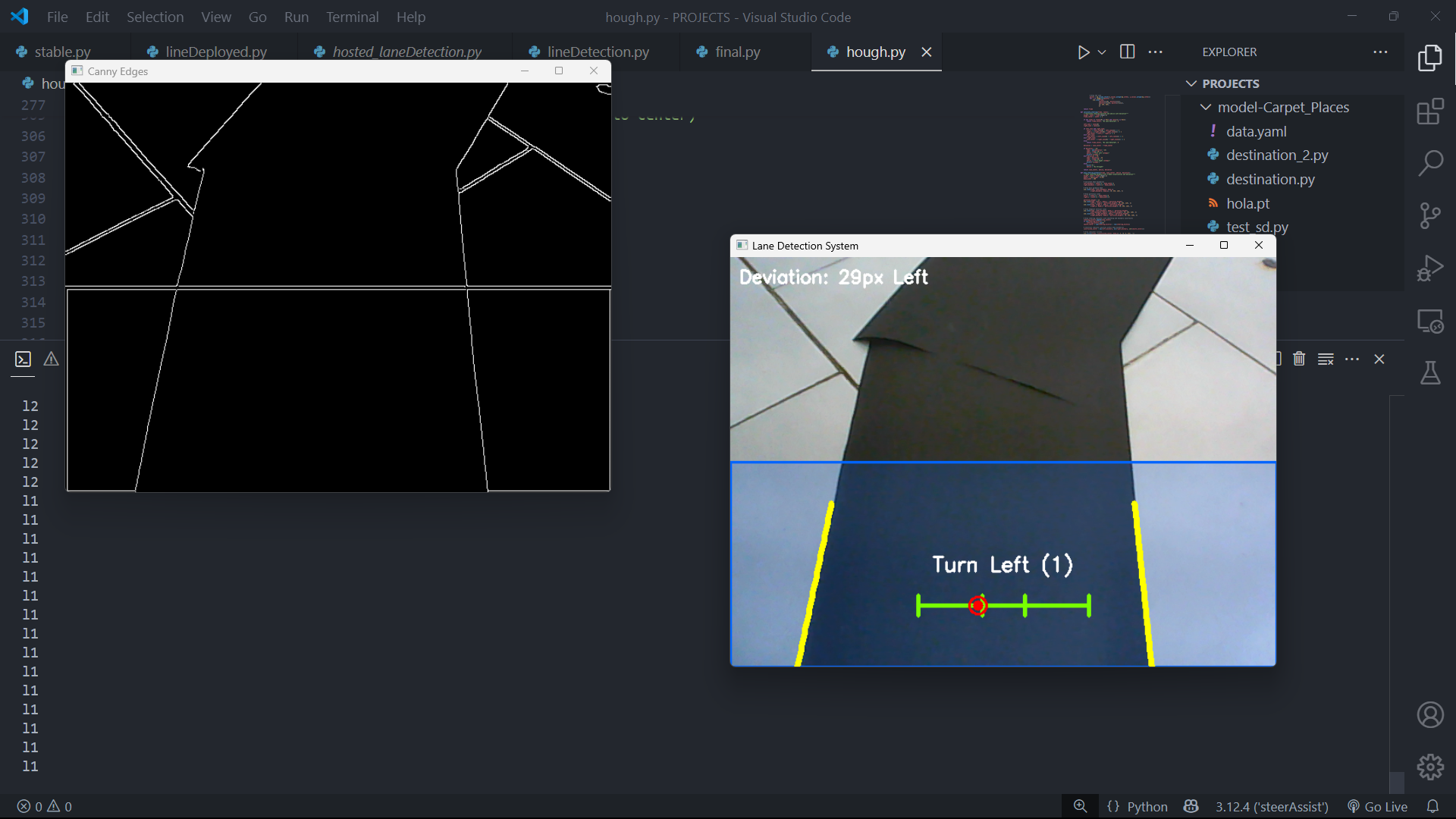
**Working:**

There are two esp32 chips one acts as a server and the other hosts the video feed in that server using a HTTP protocol.Then we can connect to the server through our laptop and send the commands by creating a socket through UDP protocol. There are three different approaches to accomplish this task

* Lane Detection using Hough Transformation
* Lane Detection using HSV Tuning
* Lane Detection using Binary Thresholding

**Lane Detection using Hough Transformation:**

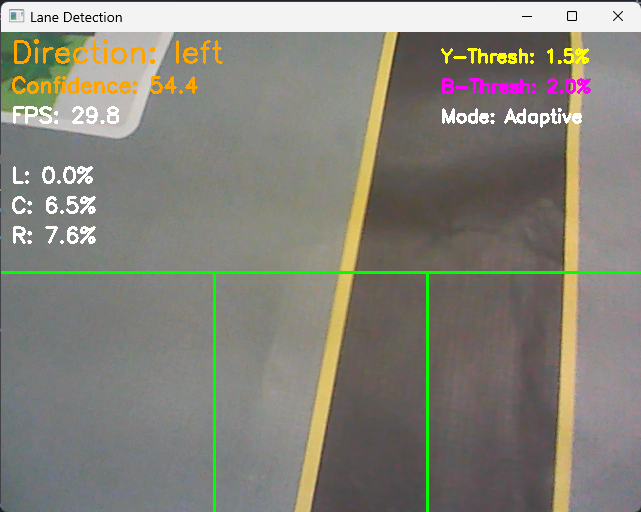
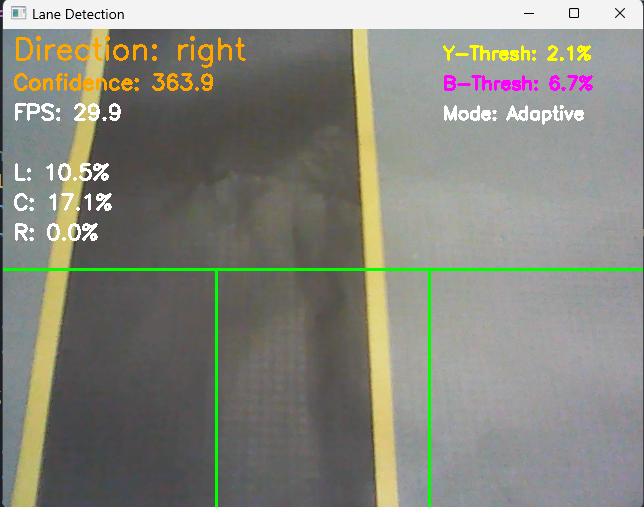
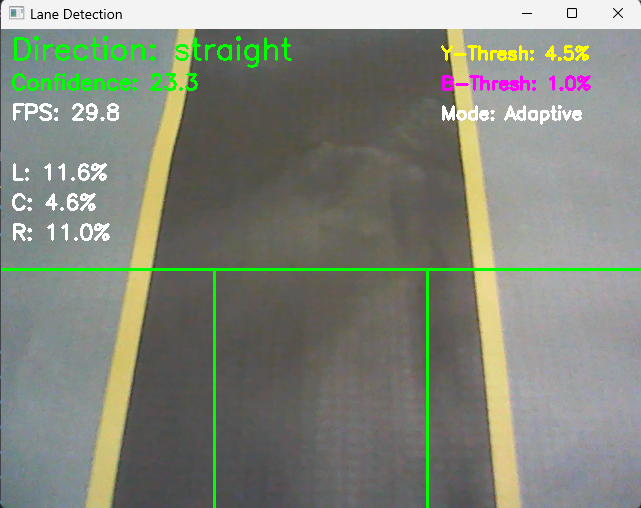
It begins by capturing frames from a webcam and applies preprocessing steps such as grayscale conversion, Gaussian blurring, and Canny edge detection to highlight lane boundaries. A region of interest (ROI) is defined, either as a rectangle or trapezoid, to focus on relevant lane markings while reducing noise. The Hough Transform is then used to detect vertical lines, filtering them based on slope to identify left and right lane lines separately. A history-based smoothing approach is applied to stabilize lane detection by averaging past lane positions. The program then fits either a linear or quadratic model to the detected lanes and selects the best fit based on R-squared values. The detected lanes are drawn on the frame, and the vehicle's lane center is estimated to calculate steering deviation. A dynamic guidance system visualizes steering recommendations using an I-beam overlay, providing real-time advice on turning left, right, or moving straight.



As explained before it detects all the edges that are present in the current frame, even the minor details, but it is very difficult to differentiate between the curves and straight lines to fit a perfect line in that. It is very inconsistent in performance too. The Main problem with this method is that when it encounters the curved paths, lanes it fumbles and does not perform well.

**Lane Detection using HSV Tuning:**

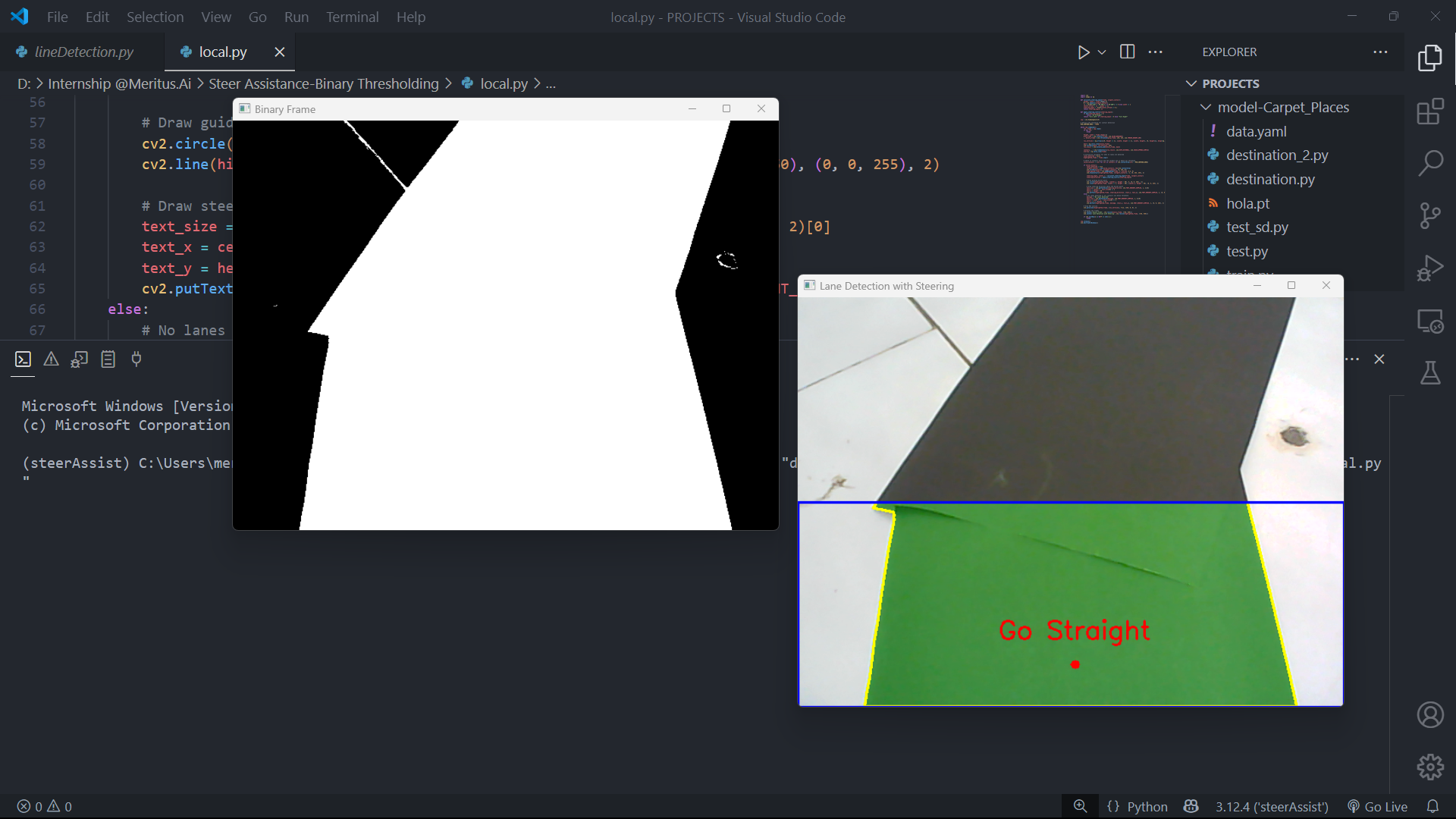
This method applies the color segmentation on the frames that the camera detects. The lower limit and the upper limit for the Hue, Saturation, Value are defined in the Region of interest to mask the other colors and to segment the color of the Lane. Once those features are applied to the Region of Interest, it is divided into three vertical sections. The system then calculates the percentage of color pixels in each section to determine lane positioning. It steers the bot by determining the dominant percentage of pixels in each region.



However when deploying this in the bot it is not very efficient with only having 6-12 frames per second. Also the lighting of the environment matters so much. This method is very computationally demanding and too complicated.

**Lane Detection using Binary Thresholding:**

The code first converts the video frame to grayscale. Then, it applies binary inverse thresholding. This converts the image into a black-and-white format where pixels with intensity values above the limit become white, others turn into black. This helps in segmenting the lane markings from the background very efficiently.



To focus only on the relevant portion of the image, a **Region of Interest (ROI)** is defined, where the bottom half of the frame is selected. The lane markings are extracted by applying a **bitwise AND operation** between the thresholded frame and the mask. Contours of the lane markings are detected. The largest contour is assumed to represent the lane, and its center position is used for steering calculations.

To avoid the miscalculations and detecting a small group of black pixels as a Lane/Line a certain area threshold is used. But it may vary according to the size of the Lane/Line. To overcome this problem an Adaptive/Dynamic method of Thresholding is used. It is programmed to store the recent values of the contour and set a fraction of the mean of those values as the threshold.

**Minimum Contour Area =**

Assuming the contour area history stores the last 10 detected contour areas, let the recorded values be [12000, 15000, 13000, 14000, 16000, 11000, 17000, 12500, 13500, 14500]. The adaptive minimum contour area is calculated by first determining the average of these values. The sum of all contour areas is 1,385,000, and dividing by the history size (n = 10) gives an average area of 13,850. This average is then multiplied by a fraction, which is set to 0.5, resulting in an adaptive threshold of 6,925. Any contour with an area smaller than this threshold is ignored, ensuring that only significant lane-like structures are considered for detection.

**Instructions:**

Open the speedControlled\_motor folder in Arduino IDE. Select the DOIT ESP32 DEVKIT V1 as the board and upload the code into the server esp32. Then by using the AI THINKER-ESP32 CAM as the board, upload the CameraWebServer code into the esp32 camera. Then make sure that you have the required libraries installed on your machine. If not, use the following command to install them in your environment.

pip install opencv-python numpy requests

There are two python scripts in the folder ‘local\_laneDetection.py’, ‘hosted\_laneDetection.py’. The local file runs the script in your web camera, you can use it to check the lighting and the ranges of the pixel values.

It is very important to adjust the lower limit of the threshold inside of the cv2.threshold function according to your lighting conditions to avoid detection of duplicate Lane/Line Contours.

These are a few examples of different threshold values.

● cv2.threshold(gray\_frame, 100, 255, cv2.THRESH\_BINARY\_INV)

● cv2.threshold(gray\_frame, 50, 255, cv2.THRESH\_BINARY\_INV)

● cv2.threshold(gray\_frame, 120, 255, cv2.THRESH\_BINARY\_INV)

● cv2.threshold(gray\_frame, 80, 255, cv2.THRESH\_BINARY\_INV)

Then place the bot on the Lane/Line to start the autonomous Lane Detection System.

**Results:**

The Path Tracking Method was finally decided to be the Binary Thresholding method due to its efficiency and performance. The outputs and the implementations have been tested and stored in a separate folder, please refer to that. The deployed system successfully navigated both straight paths and mild curves with minimal deviation. The combination of real-time video processing and UDP-based command transmission enabled smooth and responsive lane tracking. The lightweight computational requirements of the Binary Thresholding method ensured efficient execution, maintaining stable frame rates on the ESP32-based hardware.