Lane Detection Algorithm

-Anirudh Narayanan

INTRODUCTION

Self-driving cars, also known as autonomous vehicles, are vehicles that can operate without a human driver. These vehicles are equipped with various sensors, cameras, and computer systems that enable them to sense and navigate their surroundings. Lane detection is an important part of autonomous driving systems that involves detecting and tracking the lane markings on the road. The purpose of lane detection is to enable the vehicle to stay within its lane and avoid collisions with other vehicles on the road.

Different Approaches

Traditional Lane Detection Techniques:

Traditional lane detection techniques involve the use of computer vision algorithms that analyze images or video frames captured by cameras mounted on the car. These algorithms detect lane markings by identifying edges and lines in the image and fitting them to a mathematical model of a lane. Some common techniques used for traditional lane detection include Hough Transform, Canny Edge Detection, and Sobel Edge Detection.

Advanced Lane Detection Techniques:

Advanced lane detection techniques involve the use of machine learning algorithms that can learn to detect lanes from training data. These algorithms are trained on large datasets of images and video frames with annotated lane markings. Some common techniques used for advanced lane detection include Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs).

STEPS IN PROCESSING INPUT FRAME

INPUT FRAME:

STEPS IN PROCESSING INPUT FRAME

GRAYSCALE:

STEPS IN PROCESSING INPUT FRAME

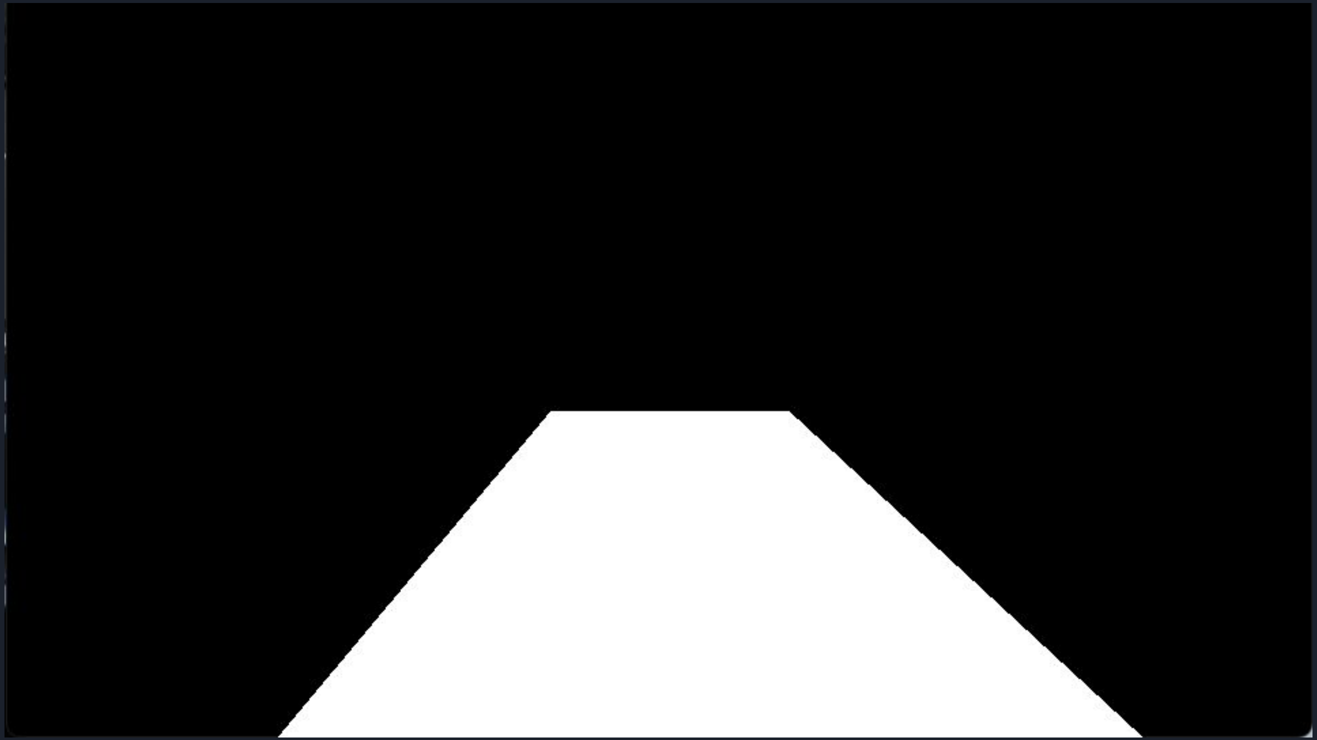
SMOOTHENED IMAGE: (Using Gaussian Blur)

STEPS IN PROCESSING INPUT FRAME

EDGES: (using canny edges)

STEPS IN PROCESSING INPUT FRAME

MASK:



STEPS IN PROCESSING INPUT FRAME

FINAL EDGES IMAGE:

STEPS IN PROCESSING INPUT FRAME

Hough Lines:

MATH INVOLVED

STEP 1: DIVIDE THE LINES INTO POSITIVE AND NEGATIVE LINES BASED ON SLOPE VALUE.

IF slope > 0 : edge is a part of the right lane.

IF slope < 0 : edge is a part of the left lane.

STEP 2: FIND median of positive and negative slopes and eliminate the slopes that are too far from the median.

STEP 3: FIND x-intercept where y value is zero.

1. Get x1, y1 and slope(i.e. m) information from the hough lines info for a particular line.
2. FIND y-intercept using y = mx + b in the form b = y1 - mx1.
3. FIND x-intercept using y = mx + b in the form -b/m = x (since y= 0).

STEP 4: FIND median of x-intercept for both positive and negative slopes and store lines that have x-Intercept within a certain range of the median.

MATH INVOLVED

STEP 5: Fit a line and plot the lane

For Positive Slope:

```
fitLine(points, FitedLine, DIST_L2, 0, 0.01, 0.01);                                  // Fit a line to the points
int t0 = (0-FitedLine[3])/FitedLine[1];                                              // Compute the intersection with the top edge of the image
int t1 = (frame1.rows -FitedLine[3])/FitedLine[1];                                   // Compute the intersection with the bottom edge of the image
Point p0 = Point(FitedLine[2], FitedLine[3]) + Point(t0 * FitedLine[0], t0 * FitedLine[1]);     // First endpoint of the line
Point p1 = Point(FitedLine[2], FitedLine[3]) + Point(t1 * FitedLine[0], (t1 * FitedLine[1])); // Second endpoint of the line
line(frame1, p0, p1, Scalar(0, 255, 0), 6);                                           // Draw the line
```

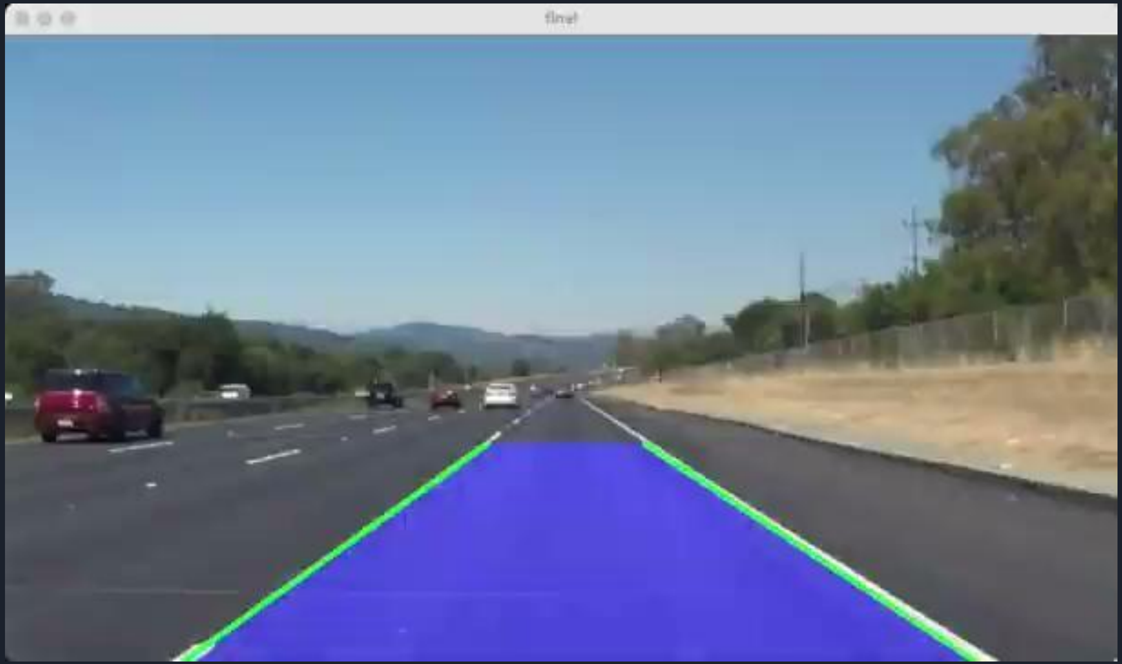
For Negative Slope:

```
fitLine(pointsN, FitedLineN, DIST_L2, 0, 0.01, 0.01)                                 // Fit a line to the points
int t0N = (0-FitedLineN[3])/FitedLineN[1];                                           // Compute the intersection with the top edge of the image
int t1N = (frame1.rows -FitedLineN[3])/FitedLineN[1];                                // Compute the intersection with the bottom edge of the image
Point p0N = Point(FitedLineN[2], FitedLineN[3]) + Point(t0N * FitedLineN[0], t0N * FitedLineN[1]);   // First endpoint of the line
Point p1N = Point(FitedLineN[2], FitedLineN[3]) + Point(t1N * FitedLineN[0], t1N * FitedLineN[1]);   // Second endpoint of the line
line(frame1, p0N, p1N, Scalar(0, 255, 0), 6);                                        // Draw the line
```

FINAL RESULT

FINAL RESULT: Learn from previous frame.

If the x-coordinates in the current frame jump over a threshold, then use the x-coordinate from the previous frame and only consider direction (change in x between current and previous frame) of the current x-coordinate and update the previous x-coordinate, in the direction of the new found x coordinate.



Second Implementation:

Here using homography, we convert the frame to a bird’s eye view and identify the lanes on it and project them back to the dashcam pov.

STEPS IN PROCESSING INPUT FRAME

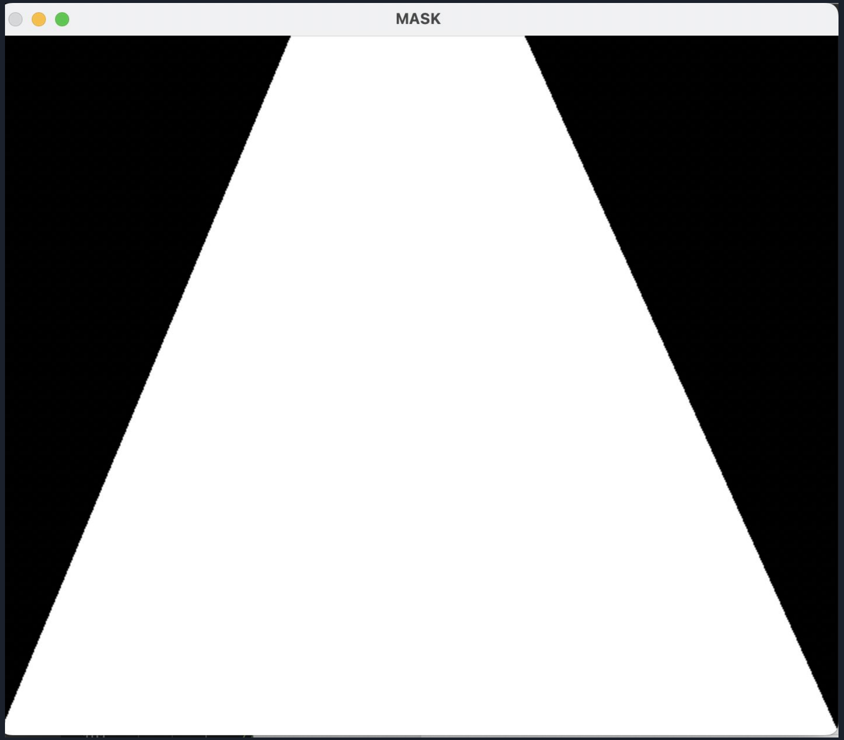
BIRD’s Eye POV:

STEPS IN PROCESSING INPUT FRAME

Edges:

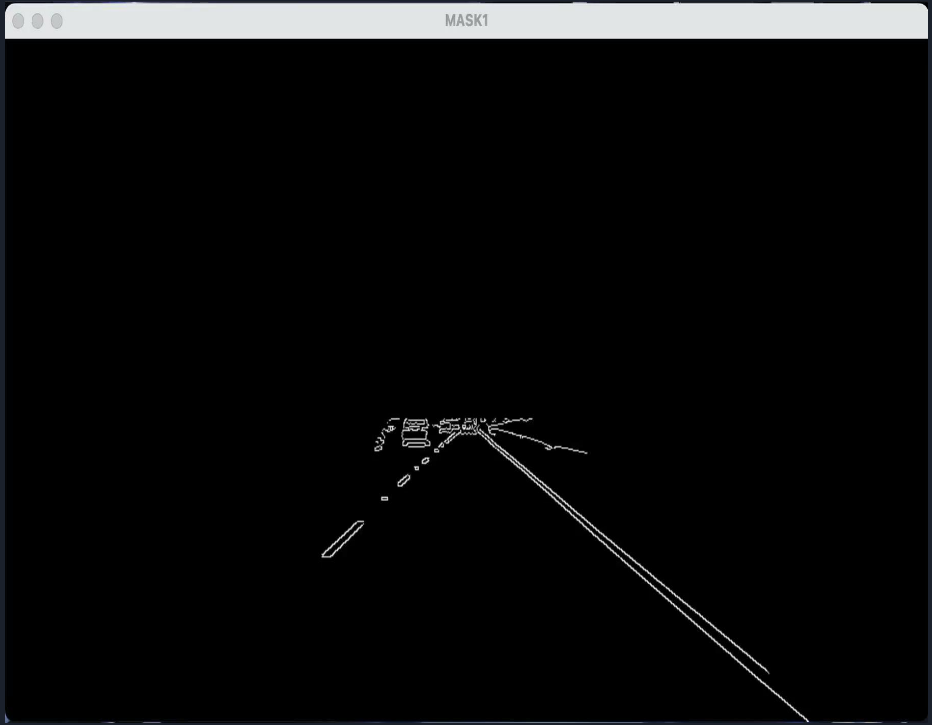
STEPS IN PROCESSING INPUT FRAME

Mask Used:

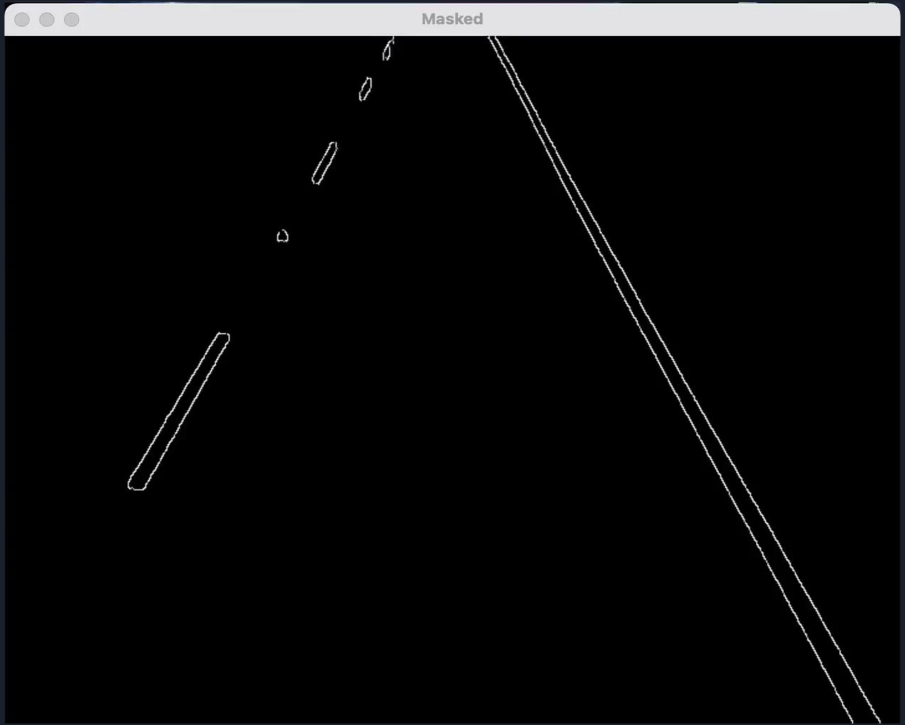


COMPARISON WITH PREVIOUS IMPLEMENTATION

IMPLEMENTATION 1

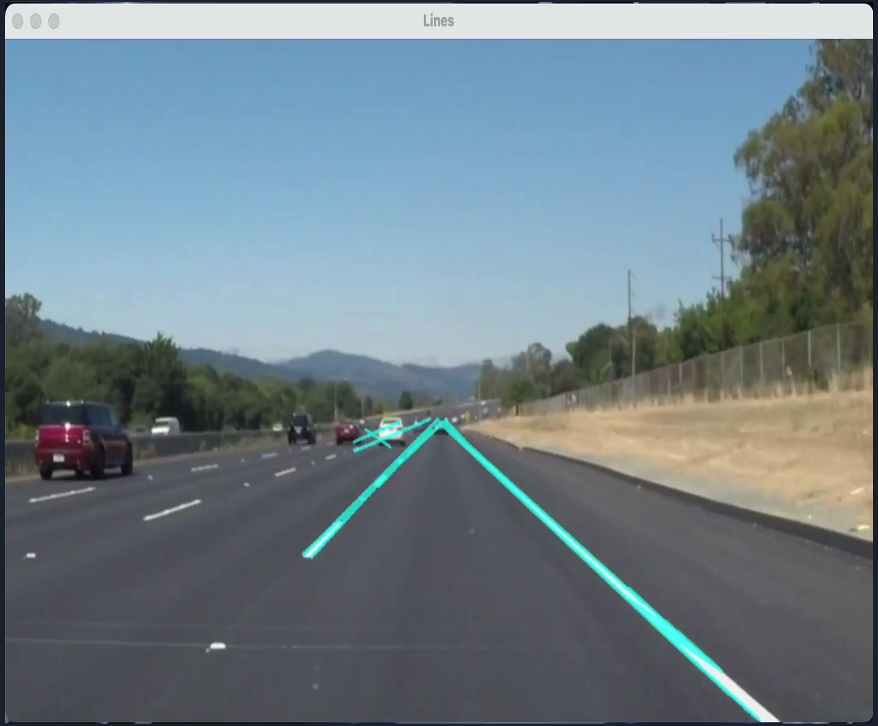


IMPLEMENTATION 2

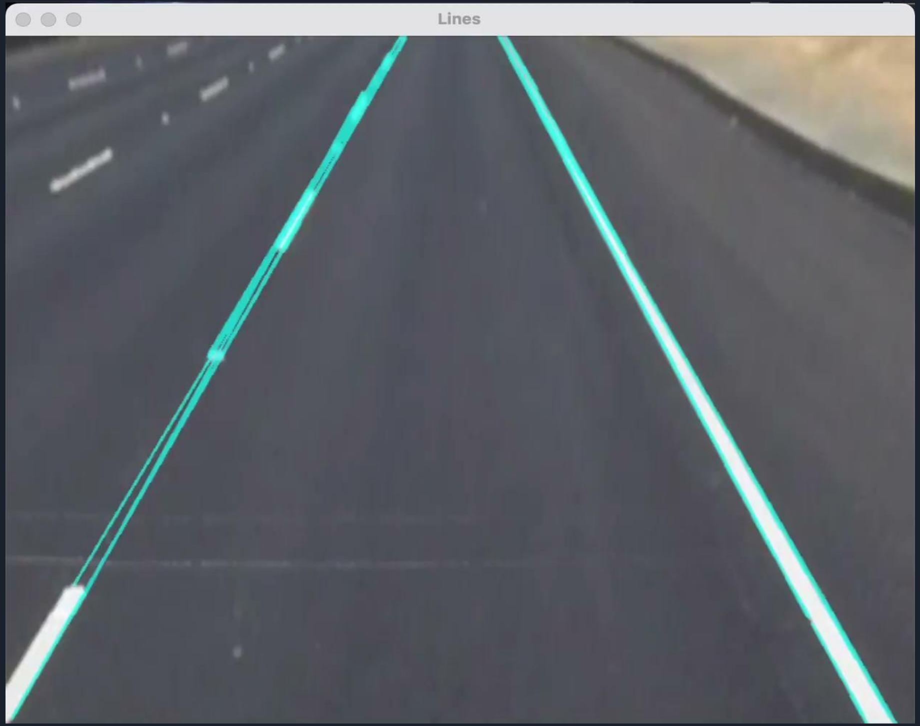


COMPARISON WITH PREVIOUS IMPLEMENTATION

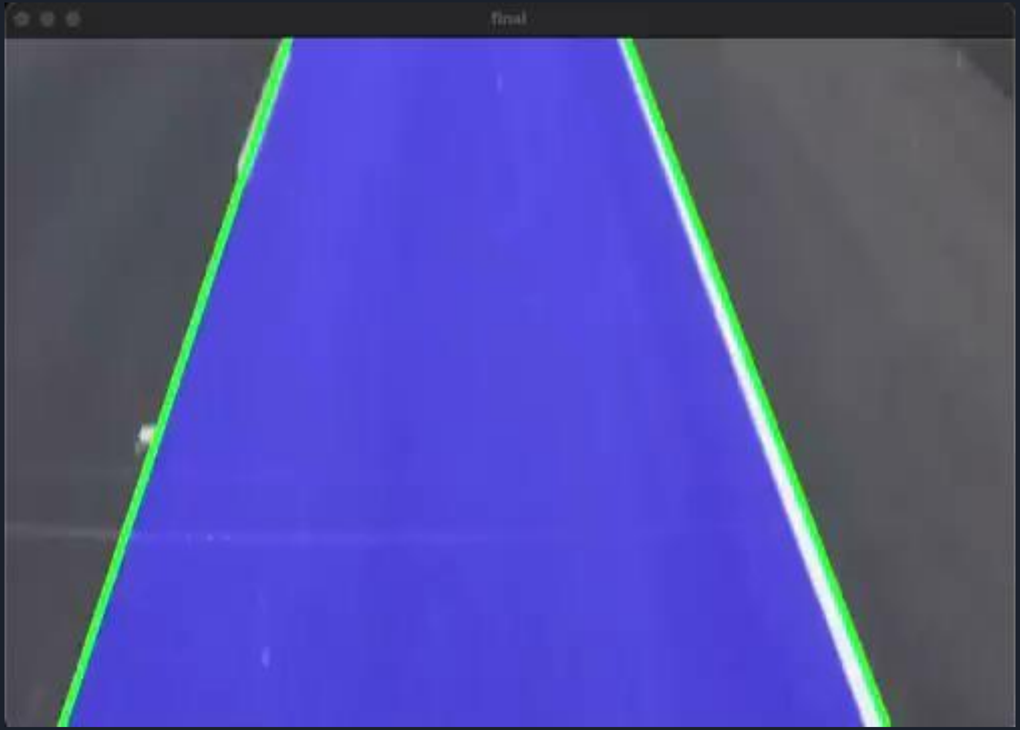
IMPLEMENTATION 1



IMPLEMENTATION 2



BIRD'S EYE VIEW FINAL OUTPUT



COMPARISON WITH PREVIOUS IMPLEMENTATION

IMPLEMENTATION 1

IMPLEMENTATION 2

OUTPUT ON A ROAD WITH CURVE