*****Advanced Lane Finding Project*****

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a threshold binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

NOTE:

There are two jupyter notebooks to pay attention to.

Main_JNB2 ------ shows the breakdown of the functions with pictures.

Main ------ shows the Video Class used and its contained functions which are similar except for the fit_poly function.

Also, Most of the code is carried over from the class and I did gain some inspiration from Gallen Ballews' Git Hub Repo (https://github.com/galenballew/SDC-Lane-and-Vehicle-Detection-Tracking/tree/master/Part%20II%20-%20Adv%20Lane%20Detection%20and%20Road%20Features)

Explanation of Files Currently in Project Repo

The Output images has sorted images on the output of various functions and their images.

The Calib_mat.p file is the calibration data.

The Main.ipynb is the main code with a class defined for lines and its data used for videos.

The Main_JNB2 is the second Jupiter notebook that runs through each function individually.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

1. Provide a Writeup / README that includes all the rubric points. - Currently Reading It.

Camera Calibration

1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

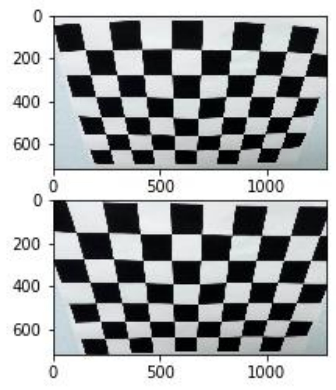
The code for this step is contained in the second cell of the Main_JNB2 jupyter notebook

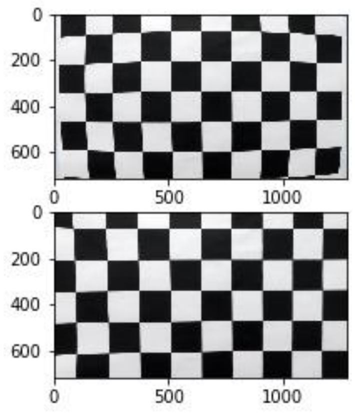
This is majorly the code from the class.

As I understand it, the code requires mapping known points on a chess board to the known points of an actual chess board without distortion, i.e., perfect squares.

It starts with creating an array of Object points that are the points to map the points of a chessboard found in the image.

Then this requires first converting the image to grayscale, using the cv2 function to find the corners in the image, and then do that for multiple images and use that as the input to the cv2 function to create calibration Matrices as output from the cv2 function. This Matrix can then be used to distort and undistort images





The above two are examples of said Clibration. The calibration matrix was then stored as a Pickle file to reduce time for future quicker calibrations.

Pipeline (single images)

Note: For Further explanations, I will be providing examples of the following image run through each function.

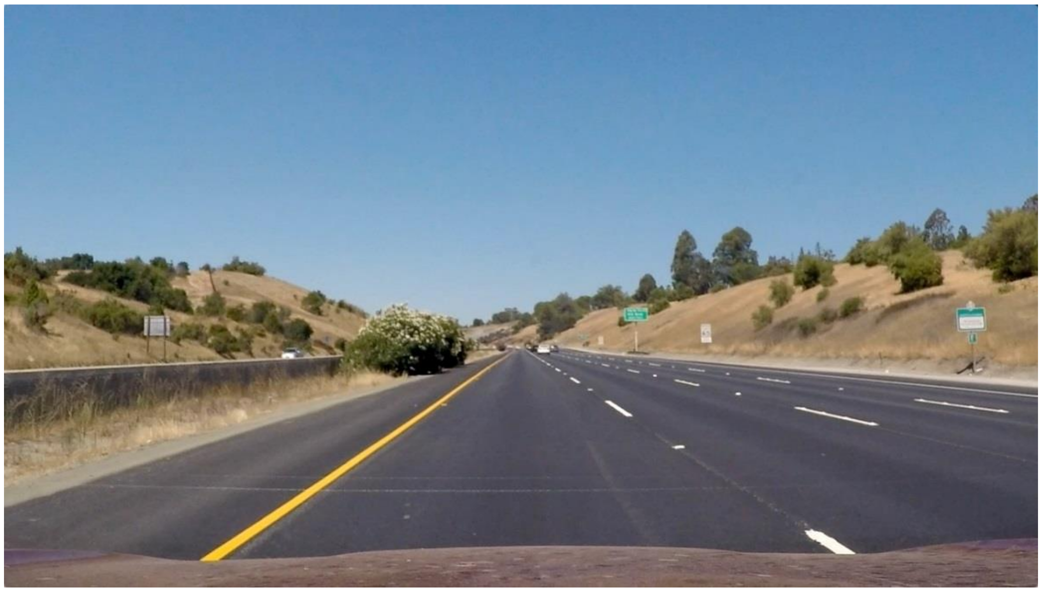


Figure 1 Original Image

1. Provide an example of a distortion-corrected image.

Using the steps mentioned above, I created the calibration matrix using the chessboard images and then pulled the data from the Calibration Pickle file and created the following undistorted image.

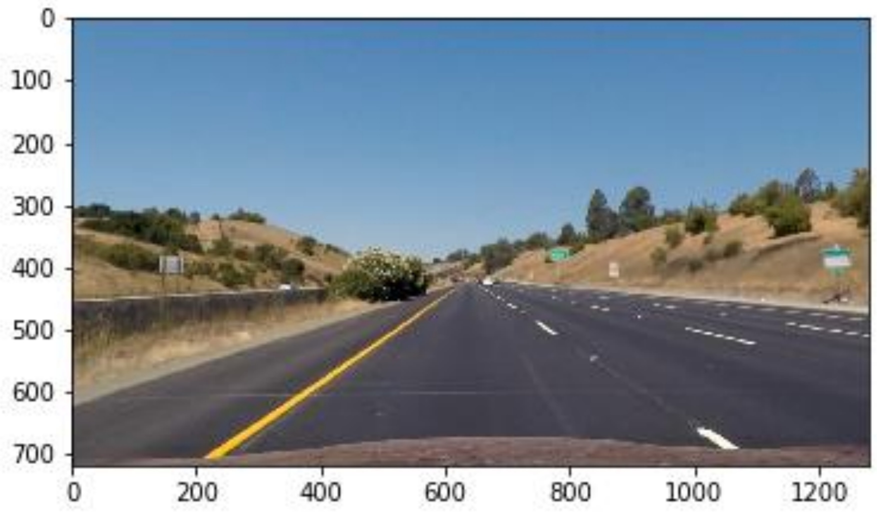


Figure 2 Undistorted Image

2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I attempted multiple different sets of thresholding and compared which sections could be used for providing the best combined thresholds. The code was taken from the class code, this is seen in the function definitions for Sobel Threshold, Directional Sobel, Magnitude threshold, Gray thresholding, and HLS thresholding. I decided to remove gray thresholding eventually because I didn't think it offered much to improve the quality of the combined thresholding without it. Also, I am highly against Directional thresholding more so as it is not a very accurate thresholding method and provides noise.

Below are examples of all thresholds.

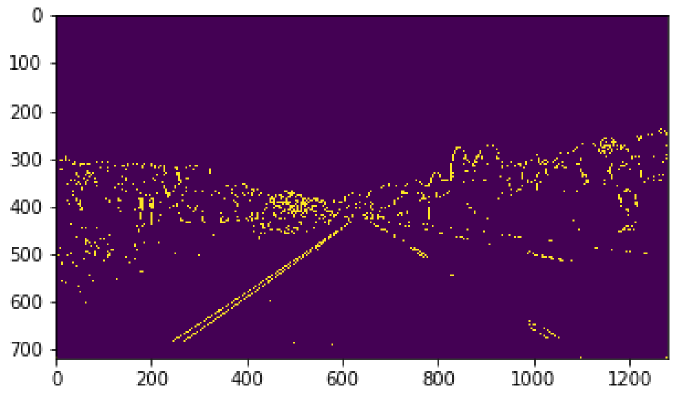


Figure 3   Sobel Thresholding

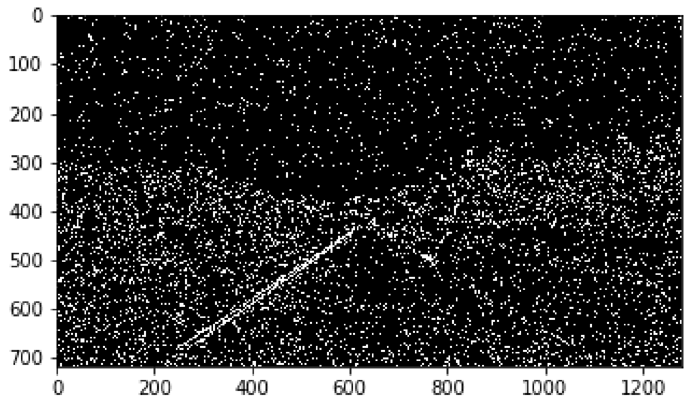


Figure 4 Directional Thresholding

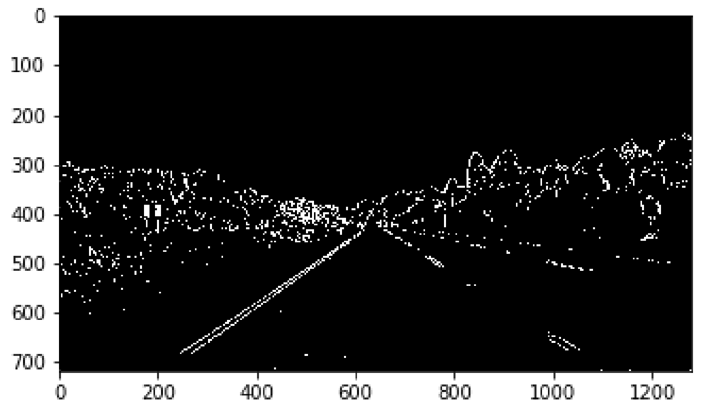


Figure 5   Magnitude Thresholding

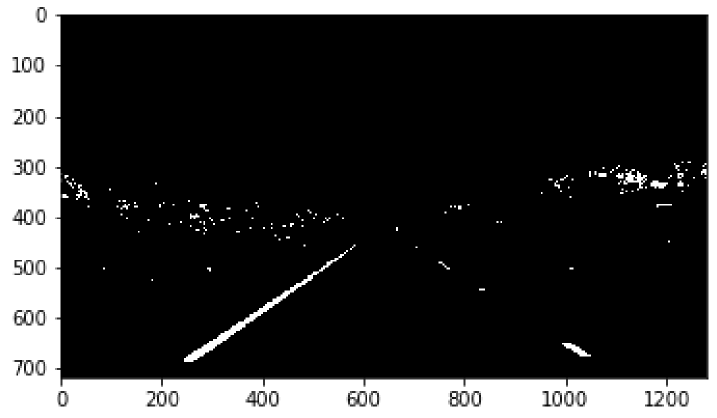


Figure 6   Grey Thresholding

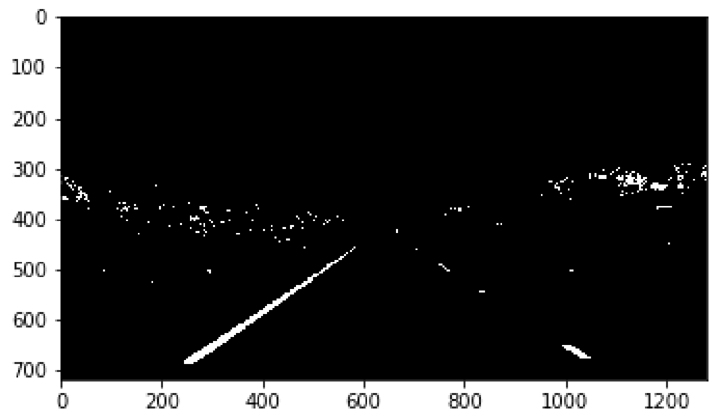


Figure 7   HLS Thresholding

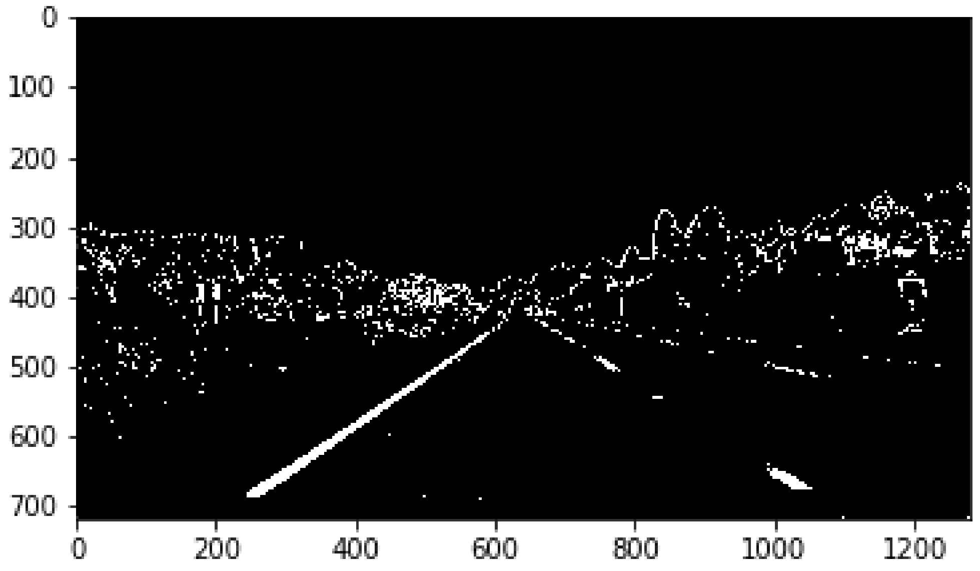


Figure 8   Combined Thresholding

Also, For HLS, I also decided to use the L part as well. I noticed that there was a certain section that changed as shown below.

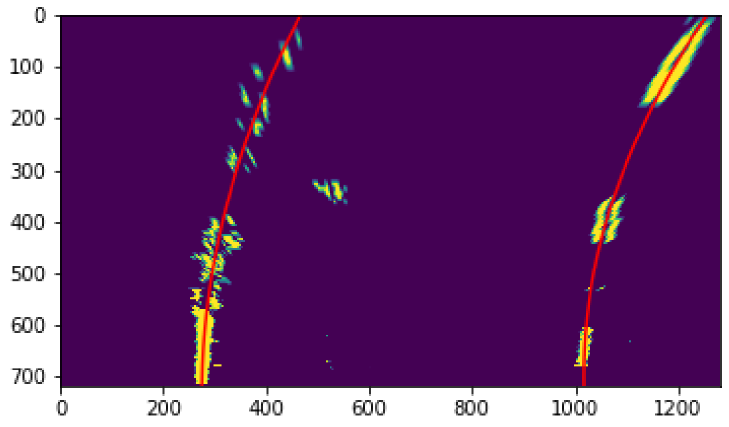


Figure 9 Test Image 7 with L thresholding

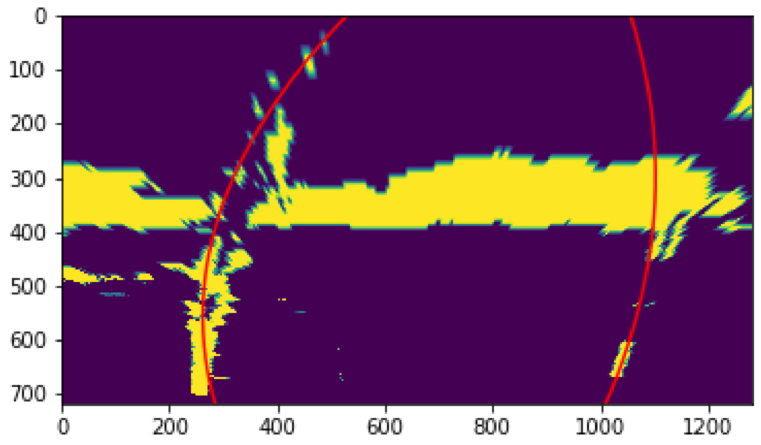


Figure 10 Test Image 7 without L Thresholding

3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

In the Function Definition Section of the Jupyter notebook, using the p_transform Function defined, I set up src points as follows.

```
src = np.float32(
        [[200, 720],
        [1100, 720],
        [595, 450],
        [685, 450]])
```

I tried playing around with src points by setting the upper line of the y axis by moving it between the range of 400 – 550. I found that the anything between 500 – 550 seemed to be too small to give enough points for the polynoimial f it, unless I started changing the margin as well which would then reduce the window size.

I then tried setting the destination points and I moved it between 720 to 1200 but It distorted the image too much for the lines that were supposed to be straight.

```
dst = np.float32(
        [[300, 720],
        [980, 720],
        [300, 0],
        [980, 0]])
```

Then using the CV2 function, I was able to get the perspective transform matrix which was then used as an input to the warpPerspective function.

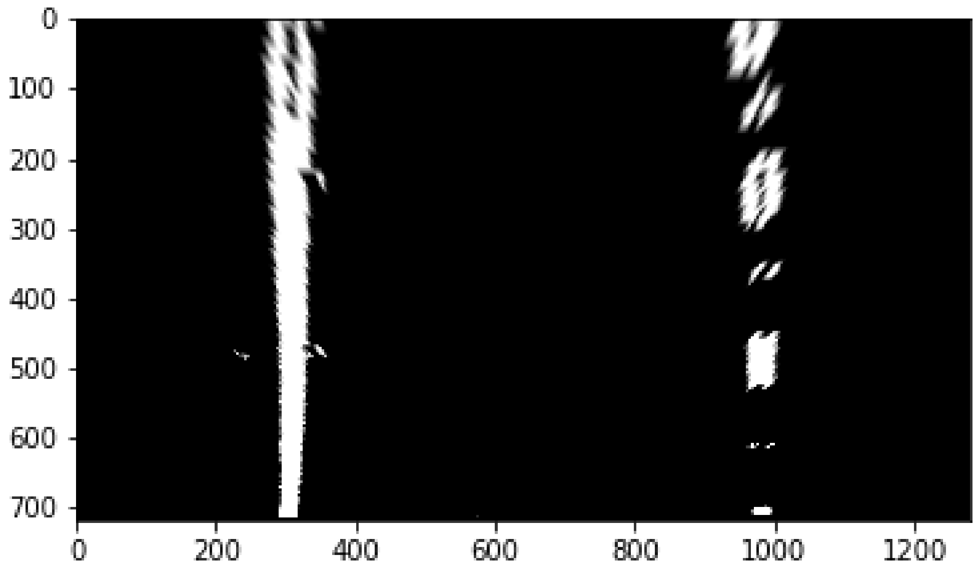


Figure 11 Warped perspective

4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

I utilized the method mentioned in the Class as I understood as follows:

1) First identify how many sections the Y Axis has to be split into, and define the margin. Here set to 100.
2) Find the Histogram of the bottom half of the image.
3) Split the image into two halves.
4) Find the Max of the histogram in each half
5) Set it as the Base of the left Line.
6) Iterate through each window
    a. Set the corners of the window
    b. Within the window Identify the pixels which are nonzero (i.e., Any pixels which seem are not blacked out after thresholding
    c. Set those as the indices for Good lanes in the left and right segment.

    d. Uptil the above point, I had taken inspiration from the Class. Now comes implementation for the Video.
    e. From here, Identify if there are sections where there were no pixels found ,
        i. Try to re-center the image at the Histogram Max location
    f. But I found this to be incorrect as between dashed lines, it would find inconsistent points and make wrong polynomials to fit those points.
    g. So I removed that section and then Set Leftx_Current to be the mean of the last points instead.

    h. Then add this to the list and create the indices matrix for those lines.

    i. Then Use the polyfit function to fit a second order polynomial. I had also attempted to fit a third order polynomial instead but it did not work. And I got inconsistent lines.

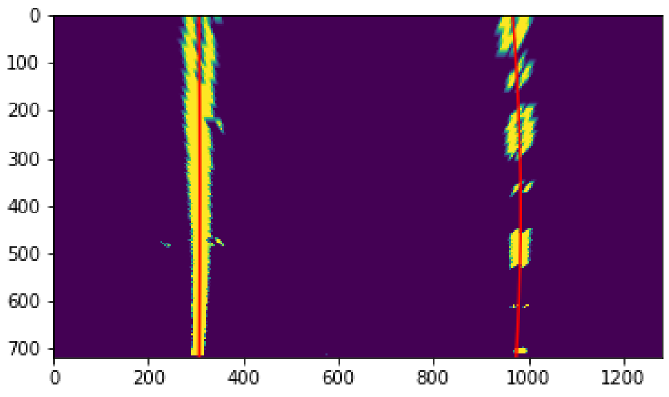


Figure 12 Lines found.

5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in get_lane_curvature function defined in the MAIN.IPYNB. Here I used the code from the class as well. I identified that the first few lines were defining how many meters each pixel represented in the image.

Then Getting the indicies for each lane, I fit a cucle to those points using the polyfit function and then calculated the curvature of the circle using the equation provided in the class and the additinaol Line provided during the class.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in the Section defined as Second Pipeline where it run the image through the calibration function and then each of the thresholding factors and finally through the perspective transform, the line detection algorithm, Curvature calculation and finally Unwarping it using the reverse perspective matrix calculated earlier.

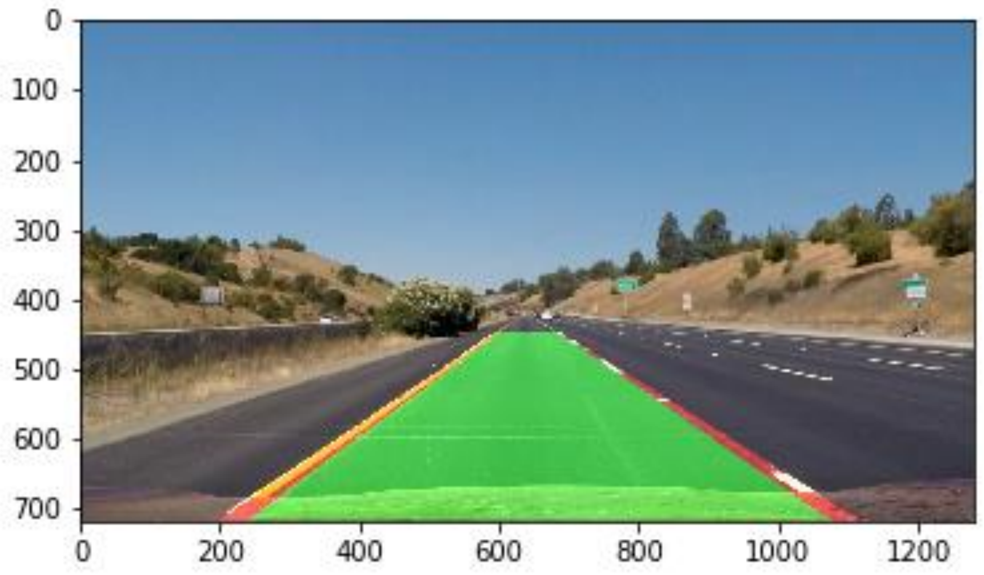


Figure 13 Final Output

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

This is in the Main.ipynb and defined in Answer 4

In Addition to running the image through the above mentioned funcitons, I utilized **a region of interest mask** before running it through thresholding to remove outlying features.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

I failed in certain sections during the video where I am unsure why at a section of the video the right lane went to the midpoint

Also, The thresholding needs to be tuned and I need to apply better smooting to the polynomial fit over multiple frames.

Also, I tried and tried multiple times to setup smoothing over frames but I kept failing. I also tried to keep setting it with the Lines.detected value but It would not work.
I tried for a week and finally Asked my mentor who pointed me to other peoples code which I am still trying to understand.