Lane Tracking Robot using Computer vision

By. Rajeshwar N S

Siddharth Singi

Hardware:

- **Raspberry pi-3 Model B**
- **USB Webcam**
- **Servo-MG90s**
- **DC Motors-2**
- **L239D Motor IC**
- **LI-PO Battery**

Software:

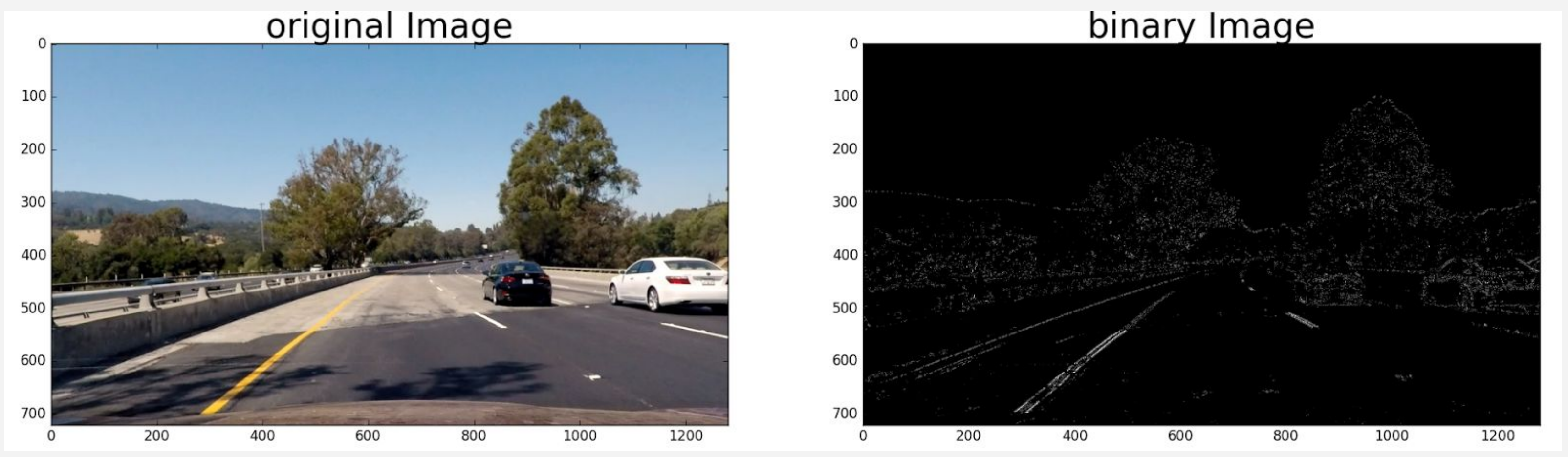
- **Python**
- **OpenCV**
- **SSH**

Synopsis:

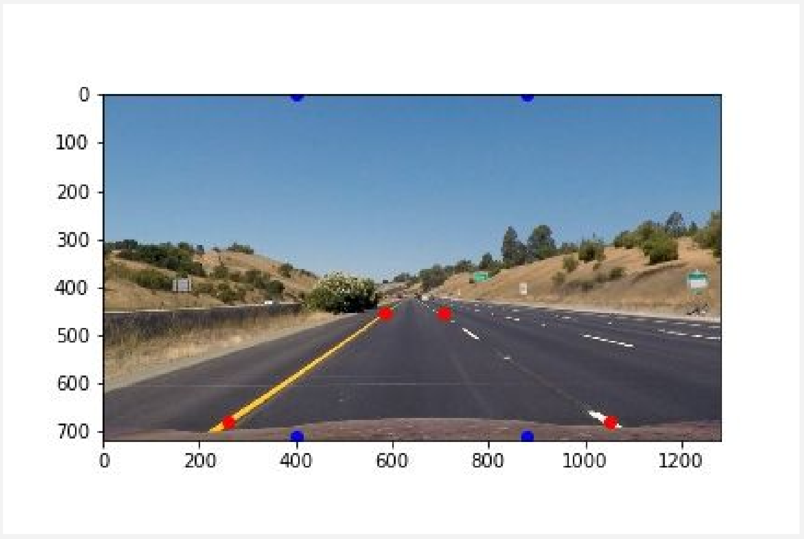
The project is divided into two parts:

1. Vision part:

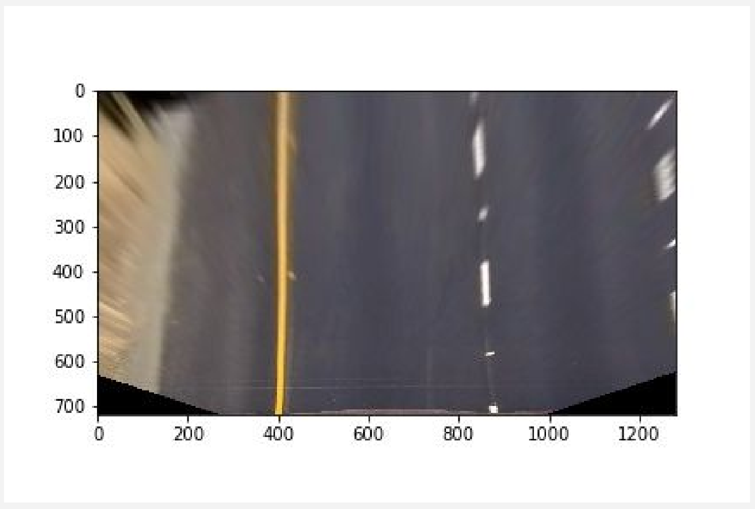
- The image is firstly converted to HLS space, and I use the S channel to filter the image, by which is more robust to locate yellow and white lanes even under different lighting conditions. Then a sobel filter (along x direction) and gradient direction filter is used to filter out most of the horizontal lines. Finally, to handle the cases where more than two candidate lanes are found, I assign the twice the weights to the S channel filter than the gradient filter. As a result, the yellow lane should be more visible than the edge of the road. Here's an example of my output for this step:



- Use perspective transform to see the image in bird-view: The image below shows the source and destination points on an image with straight lane lines: The red and blue points are source and destination points respectively:

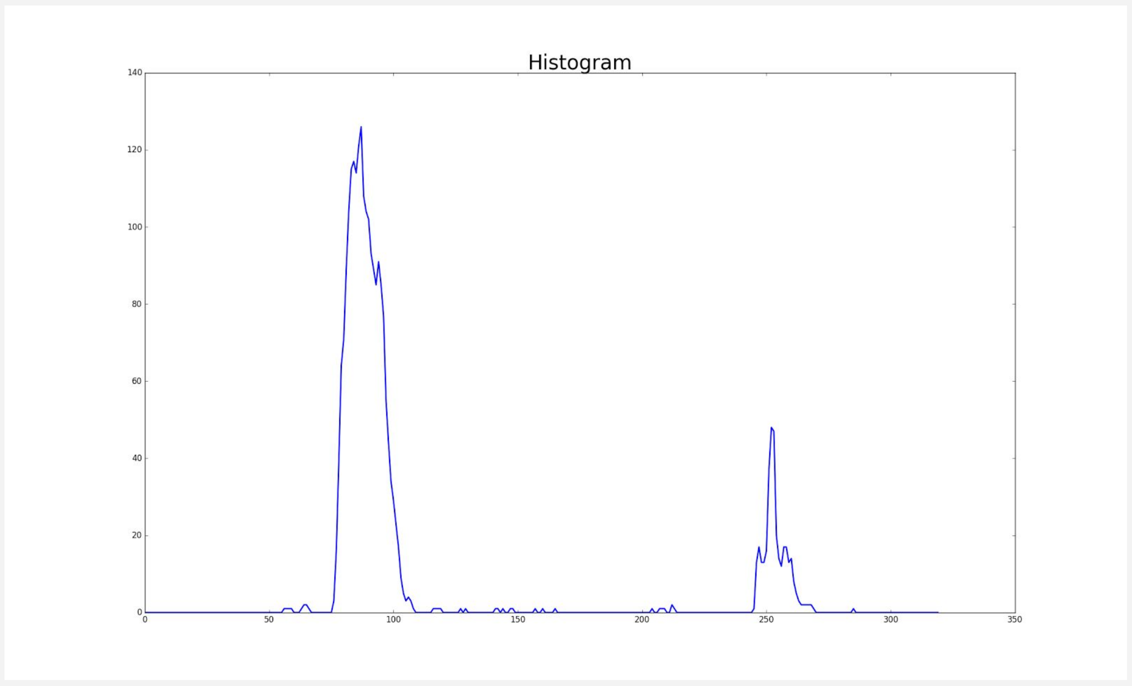


- Verify the perspective transform was working as expected by drawing the src and dst points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image by function cv2.getPerspectiveTransform



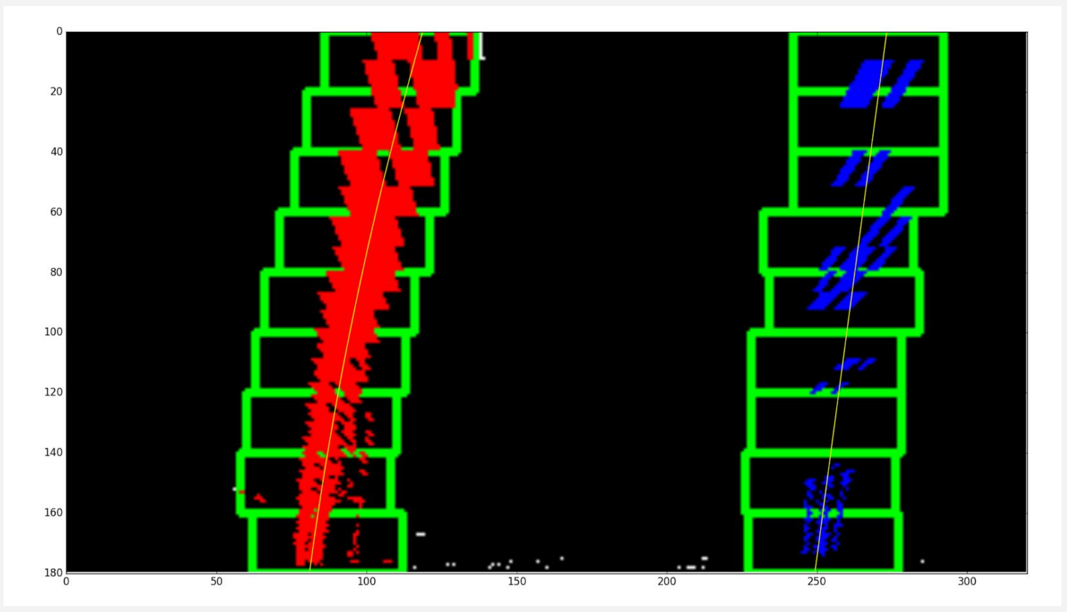
- Identify lane-line pixels and fit their positions with a polynomial

we should have a binary image where the lane lines stand out clearly. However, we still need to decide explicitly which pixels are part of the lines and which belong to the left line and which belong to the right line. Convert the warped image to binary image. Perform a histogram and identify the lanes which have peak shoots.



- Use a sliding window, placed around the line centers, to find and follow the lines. It starts at the bottom, and search up to the top of the frame. The idea is visualized as the following figure.

Fit the lineswith a second order polynomial using np.polyfit function



- Calculate the radius of curvature of the lane and the position of the vehicle with respect to center.

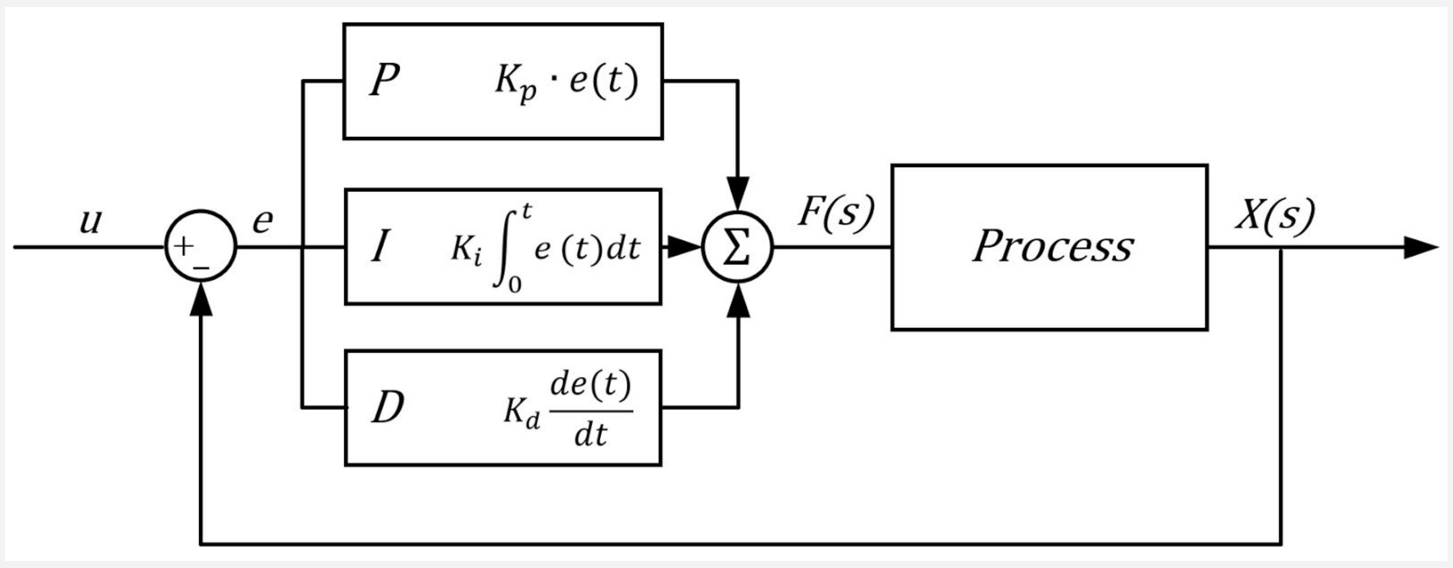
This function takes left_fitx, and right_fitx as inputs and returns mean distance between the lanes, centre of vehicle offset, left lane radius and right lane radius. This function is used in displaying the road values on the final output as well as to check the sanity of the new found lane lines.

$$\text{Radius of curvature} = \frac{\left[1 + \left(\frac{dy}{dx}\right)^2\right]^{3/2}}{\left|\frac{d^2y}{dx^2}\right|}$$

2. Motor part:

- The offset value (calculated from the radius of curvature function) which is the distance of the current position of the vehicle with respect to the center of the lane is fed from the camera to the pi.

- The returned offset value is fed to a PID controller which tries to minimize the steady state error.



Here u= current offset value
F(s) = corrected error
X(s) = motor angle fed to the servo motor which controls the front wheel.

- Appropriate values for the gains Kp,Ki,Kd are given by trial and error method to obtain a steady state response with less rise time, less settling time and no oscillations(damping)