Lane Detection

03.13.2019

Members :

1. Mrinali Vyas - 116189866
2. Srujan Panuganti - 116302319
3. Akshita Pothamshetty - 116399326

In Collaboration with

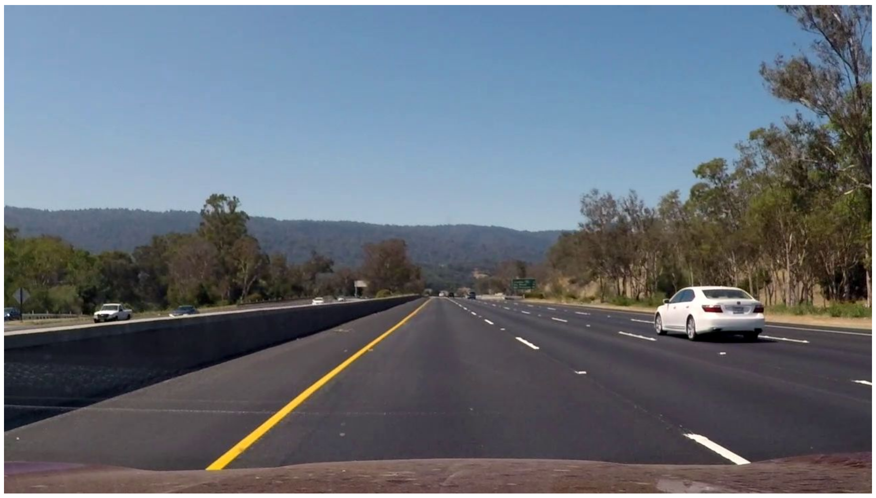
1. Sanchit Gupta
2. Aaradhana Kannan Subramanian

Overview

In this project we take a simple video as input data and process it with to detect the lane within which the vehicle is moving. We will fit a polynomial on the two lanes on both sides of the vehicle.

Instructions to run the code:

We need to make sure that the image **'project_video_frames16.png'** is in the folder from which the python file will be run.



We are using following python packages to do this project

1. Import cv2
2. Import numpy
3. Import the 2 videos and the image included in the folder.

Run “lane_detection.py” from the zip folder to run the code.

Pipeline

1. Applying distortion correction on the frames
2. Performing set of image processing techniques so that we can extract few specific features of the image.
3. Finding useful corners to calculate the Homograph.
4. Applying perspective transform to the image using the calculated homograph
5. Detecting lane pixels and fit a polynomial using a sliding window technique.
6. Warping back the lanes on to the original frame
7. Calculating the radius of curvature and predicting the turn.

1. Applying distortion correction on the frames :

In order to undistort the video captured by a monocular camera we use the opencv function cv2.undistort in order to compensate for the distortion.

Function : `cv2.undistort(initial_img, k, dist, None, k)`

Src : Initial image

cameraMatrix : K matrix provided with the files

distCoeffs: numpy array of the distortion matrix

2. Image processing pipeline:

We take each frame of the video and crop it into a frame of interest and pass through the following pipeline.

First, we have converted the cropped color image to grayscale using the function, `cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)`. This function takes the cropped image and turns it into a grayscale image.

The grayscale is thresholded using the cv2.threshold() function which converts the grayscale image to a binary image.

The thresholded image is then applied by a bilateral filter using the cv2.bilateral() function. Below image shows the bilateral filtered image which we have used to proceed with the later operation of the pipeline

3. Finding useful corners to calculate the Homograph

1. We have defined a function which uses the cv2.goodFeaturesToTrack() to give out four useful corners of the scene. These four corners are then used to calculate the Homograph matrix which is used in the further steps of the pipeline to detect lanes.
2. We have also defined a function to order the corners properly to get the correct homograph matrix. To calculate the homography we have used the cv2.getPerspectiveTransform(src,dst).

4. Perspective Transform :

In order to draw the lines across the lanes we make certain assumptions like we assume that the two lines are parallel. With this assumption we try to get a birds eye view of the each frame so we can plot the parallel lines which is easier to plot than curved lines.

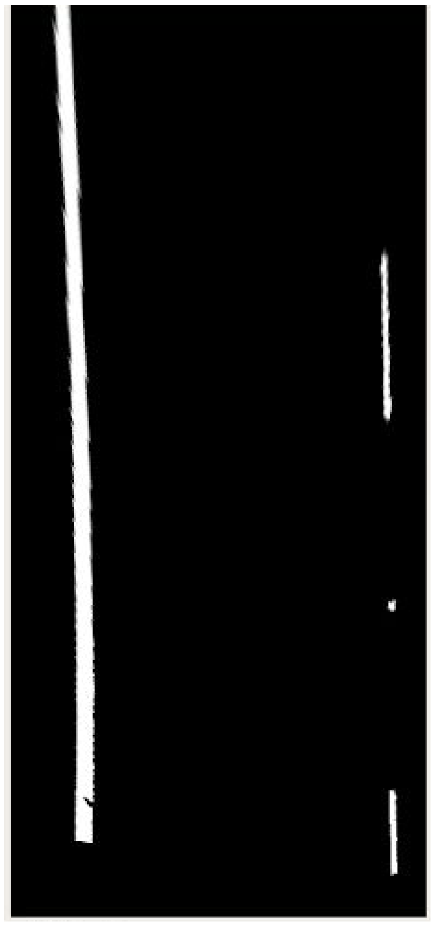
1. The road lane lines from the video stream are yellow and white so we can mask everything out except the yellows and whites of the image.
2. In order to neglect everything we might see yellow and while lines that are not lanes, or there might be lanes which are not that distinctly yellow and white, we use thresholding and apply bilateral filter to it.

   ```
   cv2.threshold(gray,180,255,cv2.THRESH_BINARY)
   ```

   ```
   cv2.bilateralFilter(threshold_image,1,100,100)
   ```

3. In order to calculate the perspective transform we first take cropped image and crop it vertically and horizontally to get the exact coordinates of the lanes visible in the frame using function good_feat.
4. We then give these coordinates in an ordered format to the getPersepectiveTransform function and get the birds eye view of the image.

   ```
   H_matrix = cv2.getPerspectiveTransform(src,dst)
   ```



5. Detecting lane pixels and fit a polynomial using a sliding window technique:

In order to fit a curve of a second degree polynomial like $x = y^2 + By + C$ we have used the function np.polyfit().

```
np.polyfit(pixels_y,pixels_x, 2)
```

In order to decide which values to consider for lanes we take the histogram of the image. The prominent peaks of the histogram are good indicators of the x position of the base of the lane. We then create a sliding window using that reference one above the other

Of window size (). The pixels that are enclosed inside the windows are the ones that are of interest.

```
active_pixels_x = np.array(active_pixels[1])
```

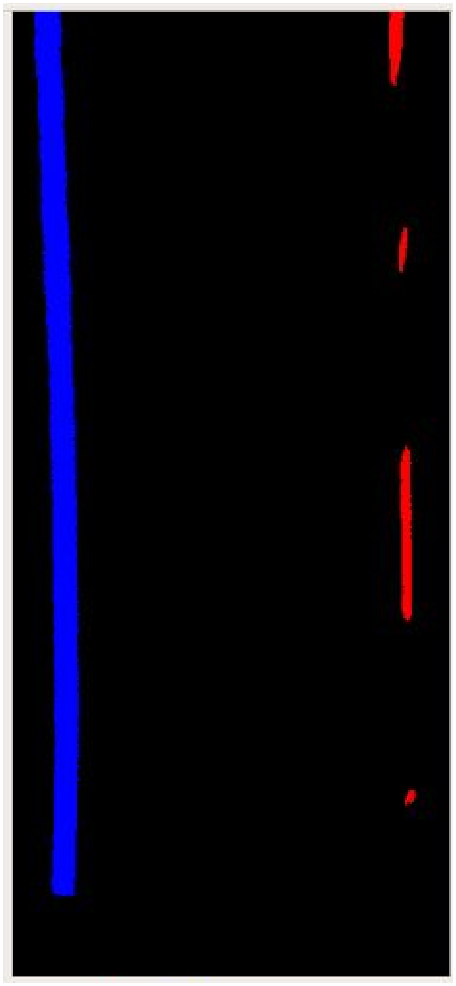
```
active_pixels_y = np.array(active_pixels[0])
```

We then take the average of the x values to get the value of the window above the current one. And the fit a polynomial along all the collected pixels of interests.

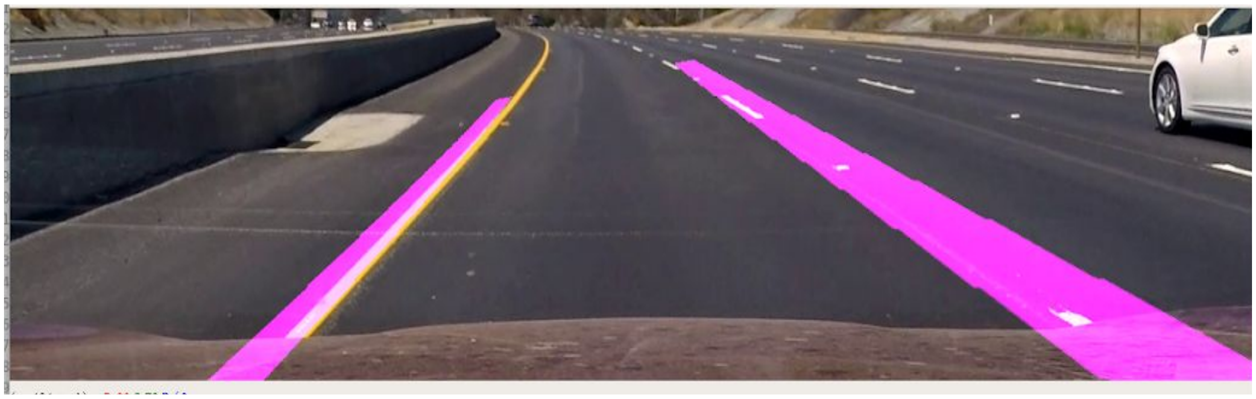
```
np.polyfit(final_left_pixels_y, final_left_pixels_x, 2)
```

We now fill the central area of the lanes using the following function -

```
cv2.fillPoly(window_image, np.int_([left_line_points]), (255,0,255))
```



Below image shows the filled polynomial on the lanes.



6. Warping back the lanes on to the original frame

We now project the detected lines back to the original video frame by using the following function -

```
cv2.getPerspectiveTransform(dst,src)
```

7. Turn Prediction :

The radius of curvature is given by the formula:

R = ((1+(2Ay+B)^2)^3/2)/|2A|

Where A and B are the x and y co-ordinates of the lanes.

```
left_curve_radius = ((1 + (2*left_polynomial_world[0]*y_max*ym_per_pix + left_polynomial_world[1])**2)**1.5) / np.absolute(2*left_polynomial_world[0])
```

Similarly we calculate the radius of curvature for the right curve.

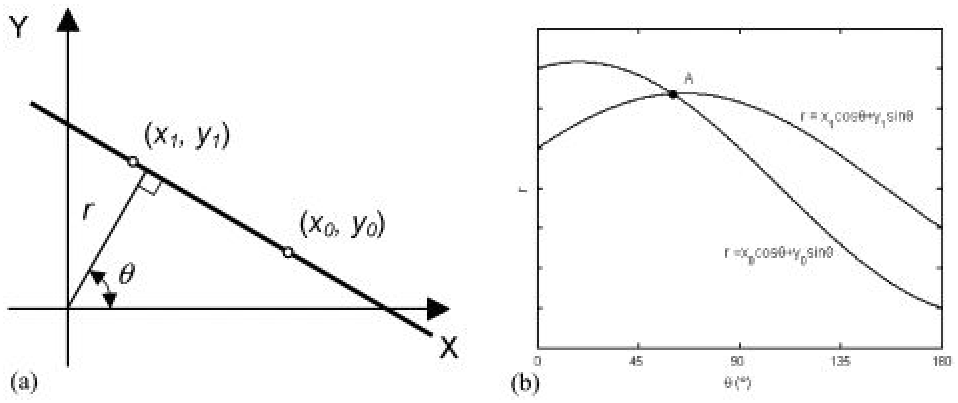
The turn of the road will be predicted by computing average of radius of curvatures of left and right lanes.

```
radius = int(left_curve_radius + right_curve_radius)/2
```

If the radius of curvature of the left lane is above 3500m the turn is right. If the ROC of the left lane is between the range 1500m to 3500m the prediction is straight else it is left if its below 1500m.

8. Challenges:

Hough lines is detected using using two parameters - an angle θ and a distance p.

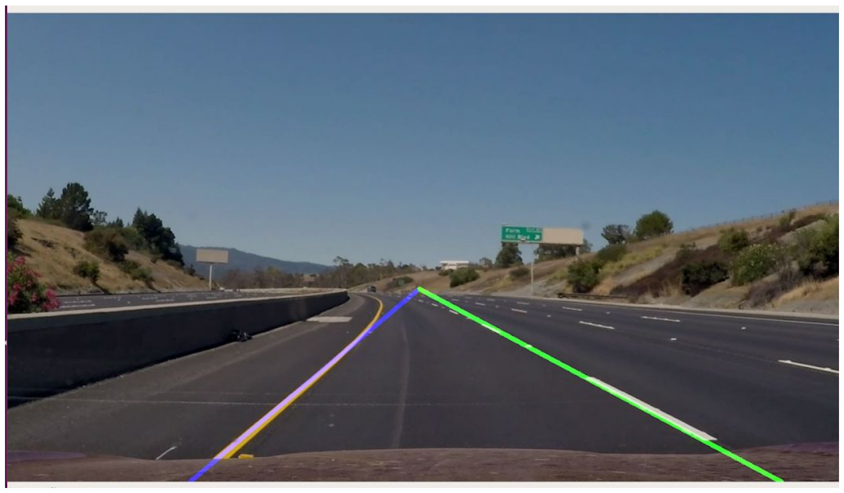

Y
(x_1, y_1)
r
(x_0, y_0)
θ
X
(a)
A
$r = x_1\cos\theta + y_1\sin\theta$
$r = x_0\cos\theta + y_0\sin\theta$
r
0
45
90
135
180
θ (°)
(b)

Where p is the length of the normal from the origin (0, 0) onto the line. and θ is the angle this normal makes with the x axis.

This method is applicable also for the vertical lanes. θ ranges from -90 to 90 and p can range from 0 to the diagonal length of the image. Hough uses the concept of vote, if a pixel is 0 its not an edge is the pixel is not 0 we takes its sine at that angle θ from -90 to 90 and its corresponding p, and then we vote. The pixels value with most number of votes determines the points of the line and we can use polyfit to plot a line for those points.

Implementation problems in hough :-

1. As the second lane is broken the hough transform doesn't always accurately detect the points hence the polyfill is very jumpy.
2. The hough transform is calculated using the vanishing points which when plotted doesn't detect the curves of the lane.



References used:

https://medium.com/pharos-production/road-lane-recognition-with-opencv-and-ios-a892a3ab635c

https://www.youtube.com/watch?v=VyLihutdsPk&t=649s

https://github.com/dmytronasyrov/CarND-LaneLines-P1/blob/master/P1.ipynb