FINDING LANE LINES ON THE ROAD

May 31, 2017
Bryan Rosales

Udacity Self-Driving Car Nanodegree

OVERVIEW

1. Project Description

To process images and videos to identify lanes lines using advanced techniques of computer vision.

2. Project Scope

The goals of this project are the following:

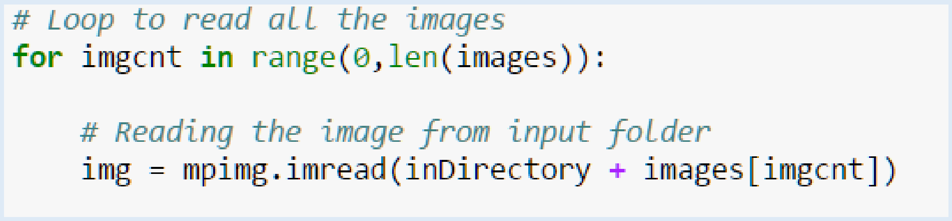
- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

3. Reflection

My Pipeline consisted of 3 steps that I am going to explain here:

I. Reading the image

I coded a *for* statement to read every image in the input folder using the library *matplotlib.image.* The image is read in full-color format.



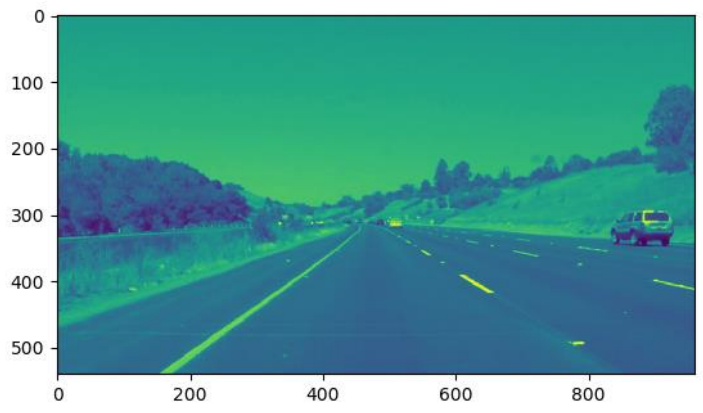
```python
# Loop to read all the images
for imgcnt in range(0,len(images)):

    # Reading the image from input folder
    img = mpimg.imread(inDirectory + images[imgcnt])
```

II. Processing the Image

Once the image is read, I called the function *process_image* which receives the image read as a parameter. Then, the next steps were executed:

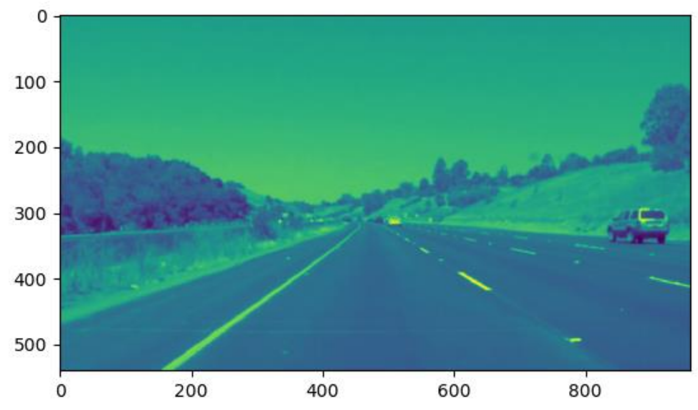
1. I converted the full-color image to grayscale to prepare for boundaries detection.

```
# Convert the input image to gray scale color
gray = grayscale(img)
```



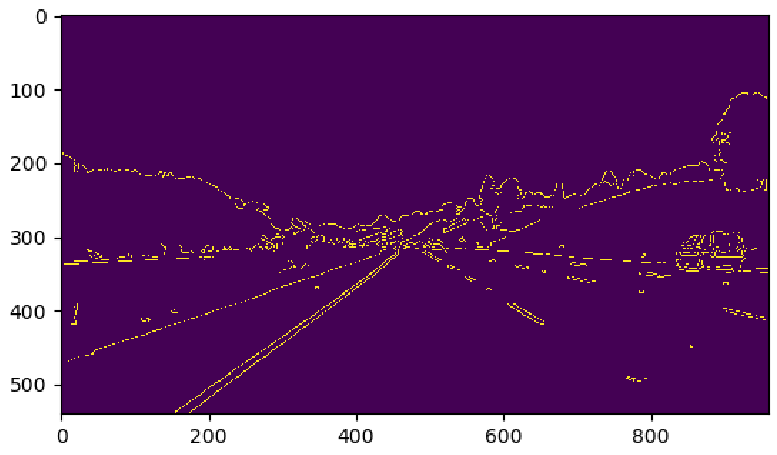
2. I applied a Gaussian filter to smooth the image. The kernel selected was a matrix 5x5.

```
# Define a kernel size of 5x5 and apply Gaussian smoothing
blurGray = gaussian_blur(gray, 5)
```



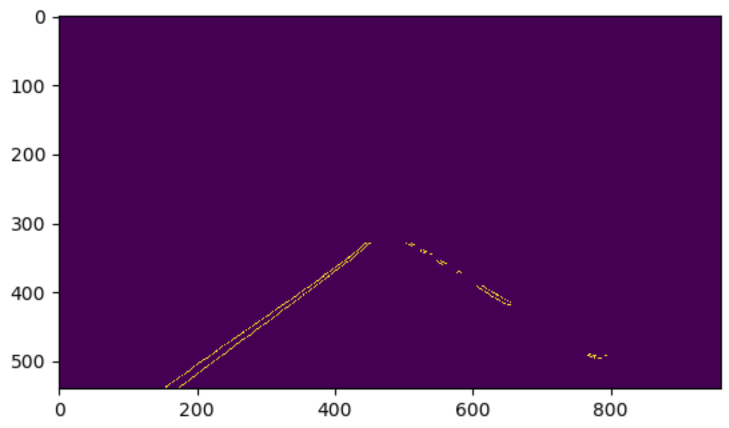
3. I called the Canny function for edges detection using a range threshold since 50 up to 150.

```
# Define parameters for Canny and apply to detect edges
low_threshold = 50
high_threshold = 150
edges = canny(blurGray, low_threshold, high_threshold)
```

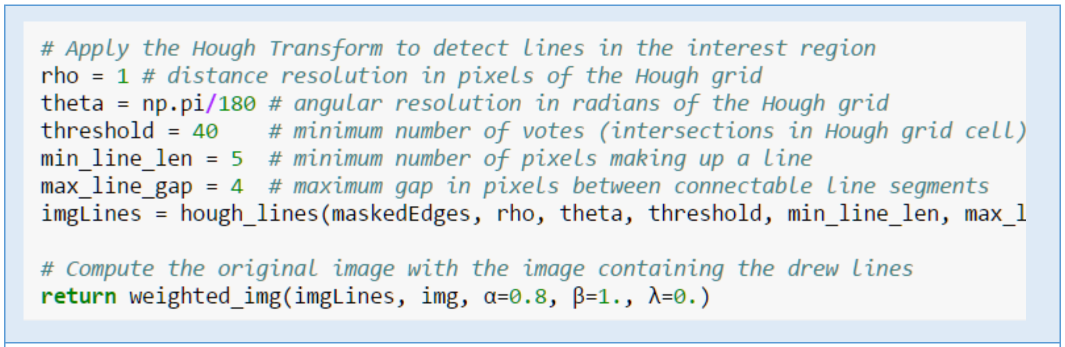


4. In this step, I applied a mask to hold the interest region and eliminate the rest of the image. It is done using a bitwise AND operation and a quadrilateral figure. The quadrilateral vertices were defined using proportions of the picture. This method allow to the algorithm can deal with different sizes of images.

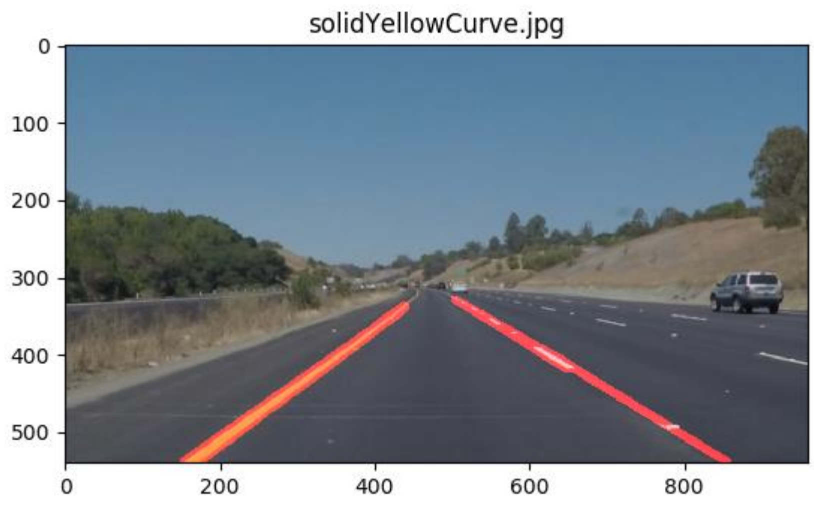
```
left_bottom = [0, ySize]
right_bottom = [xSize,ySize]
left_top = [xLeftThreshold, yThreshold ]
right_top = [xRightThreshold, yThreshold]
vertices = np.array( [[left_top,right_top,right_bottom,left_bottom]], dtype
maskedEdges = region_of_interest(edges, vertices)
```



5. Finally, the Hough Transform is used to detect the line segments in the interest region. The Hough parameters allow filtering lines might be noise because of the length or number of votes.



```
# Apply the Hough Transform to detect lines in the interest region
rho = 1 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 40    # minimum number of votes (intersections in Hough grid cell)
min_line_len = 5  # minimum number of pixels making up a line
max_line_gap = 4  # maximum gap in pixels between connectable line segments
imgLines = hough_lines(maskedEdges, rho, theta, threshold, min_line_len, max_l

# Compute the original image with the image containing the drew lines
return weighted_img(imgLines, img, α=0.8, β=1., λ=0.)
```

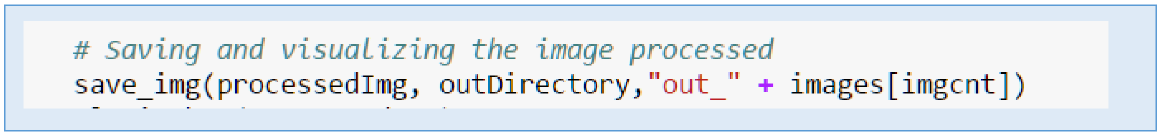


Inside Hough lines function, there is a call for a draw_lines function that draws the lines detected on the image processed at that time. The explanation about how was approach such function is described in point IV.

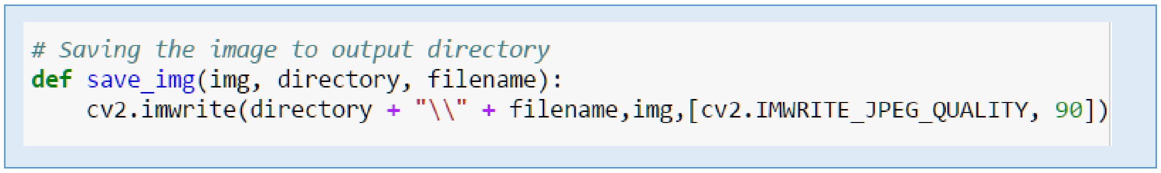
Also, the weighted function is used to compute the processed image with the original and obtain the final image.

III. Saving and visualizing the Image

In this step, the resulting image is stored to disk using the function save_img that executes an 'imwrite' method from cv2.



```
# Saving and visualizing the image processed
save_img(processedImg, outDirectory,"out_" + images[imgcnt])
```

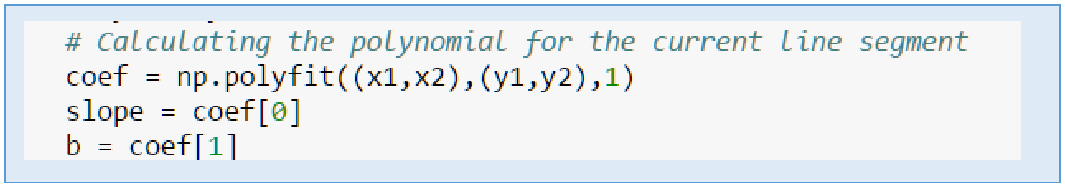


```
# Saving the image to output directory
def save_img(img, directory, filename):
    cv2.imwrite(directory + "\\" + filename,img,[cv2.IMWRITE_JPEG_QUALITY, 90])
```

IV. Drawing Lines with Polylines

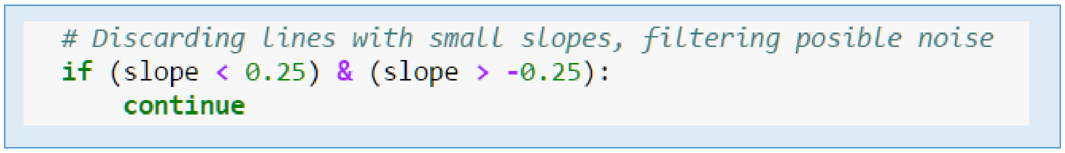
This function *draw_lines* processes every line detected by Hough Transform to be drawn on the original image. Then, the (Xs,Ys) points for each line segment are classified by the slope to determine if the line belongs to the right or left side of the lane. Here is the pipeline for the function.

1. The slope and interception of each line are calculated using polyfit by numpy.



```
# Calculating the polynomial for the current line segment
coef = np.polyfit((x1,x2),(y1,y2),1)
slope = coef[0]
b = coef[1]
```

2. The lines with slopes between -0.25 and 0.25 are discarded because do not match the angle line lane expected and introduce noise when drawing.



```
# Discarding lines with small slopes, filtering posible noise
if (slope < 0.25) & (slope > -0.25):
    continue
```

3. The points are separated by positives slopes that build the right lane line, and negatives slopes the left line.

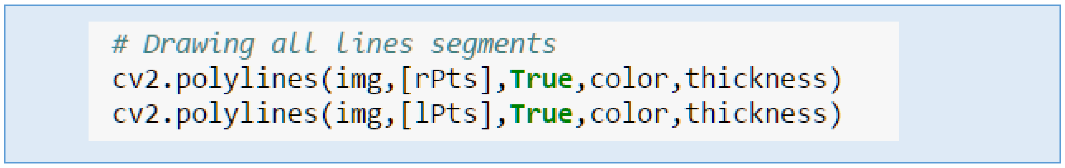
```python
# Separating points on the lines right and left using the slope
if (slope > 0) & (y1 > yThreshold):
    rXPoints = np.append(rXPoints , x1)
    rXPoints = np.append(rXPoints , x2)
    rYPoints = np.append(rYPoints , y1)
    rYPoints = np.append(rYPoints , y2)

elif (slope < 0) & (y1 > yThreshold):
    lXPoints = np.append(lXPoints , x1)
    lXPoints = np.append(lXPoints , x2)
    lYPoints = np.append(lYPoints , y1)
    lYPoints = np.append(lYPoints , y2)
```

4. Once all the lines have been processed, the resulting arrays are validated to ensure that they contain lines to display. Also, two polynomial degree 1 are built to estimate the value of Xs when Y is maximum to create a line segment that connects since the image bottom to the others lines.

```python
# Validating if there is lines detected on Right Side
if (len(rXPoints) > 0):
    # Fitting the Poly to estimate X in Ysize point
    p1R = np.polyfit(rXPoints,rYPoints,1)
    #Finding Xs point in YSize
    xR = int((ySize - p1R[1]) / p1R[0])
    #Finding Xs point in YSize
    xR1 = int((yRmax - p1R[1]) / p1R[0])
    rXPoints = np.insert(rXPoints,0 , xR)
    rYPoints = np.insert(rYPoints,0 , ySize)

# Validating if there is lines detected on Left Side
if (len(lXPoints) > 0):
    # Fitting the polynomial degree 1
    p1L = np.polyfit(lXPoints,lYPoints,1)
    # Predicting the bottom line segment
    xL = int((ySize - p1L[1]) / p1L[0])
    xL1 = int((yLmax - p1L[1]) / p1L[0])
    lXPoints = np.insert(lXPoints,0 , xL)
    lYPoints = np.insert(lYPoints,0 , ySize)
```

5. Finally, the lines are stacking and drawing using ‘polylines’ function.

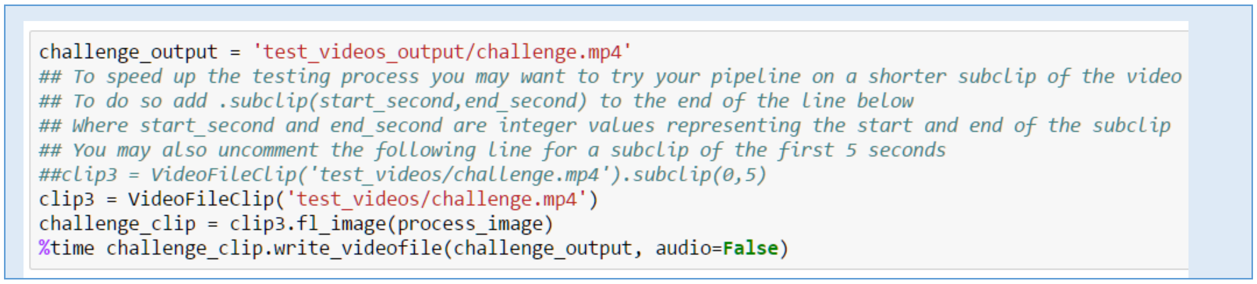
```python
# Stacking Xs,Ys points in one array by side (Left and Right)
rPts = np.array(np.dstack((rXPoints,rYPoints)).reshape(len(rXPoints),2), dtype= np.int32)
lPts = np.array(np.dstack((lXPoints,lYPoints)).reshape(len(lXPoints),2), dtype= np.int32)
```



```
# Drawing all lines segments
cv2.polylines(img,[rPts],True,color,thickness)
cv2.polylines(img,[lPts],True,color,thickness)
```

V. **Processing and Saving the modified Video**

For video processing, I am using ‘moviepy’ library to load and read frame by frame the video. Then, every frame is processed by ‘process_image’ function. Basically, it is the same process described above for each image. In the final, the video is rebuilt and saved.



```
challenge_output = 'test_videos_output/challenge.mp4'
## To speed up the testing process you may want to try your pipeline on a shorter subclip of the video
## To do so add .subclip(start_second,end_second) to the end of the line below
## Where start_second and end_second are integer values representing the start and end of the subclip
## You may also uncomment the following line for a subclip of the first 5 seconds
##clip3 = VideoFileClip('test_videos/challenge.mp4').subclip(0,5)
clip3 = VideoFileClip('test_videos/challenge.mp4')
challenge_clip = clip3.fl_image(process_image)
%time challenge_clip.write_videofile(challenge_output, audio=False)
```

4. Potential shortcomings and Improvements

- ✓ The function draw_lines requires being adjusted to draw more smoothed lines on videos.
- ✓ The parameters for Hough Transform function might be improved to avoid lines that are not part of the lanes lines. For example, the ‘challenge’ video contains some features that introduce noise to the image process. The pavement has ruptures which are detected as lines. The camera is taking part of the front side of the vehicle which generates others boundaries. The analysis requires taking in mind more dynamic conditions to robust the detection and drawn of lines.
- ✓ The mask quadrilateral should be calculated automatically through of the features and dimensions of the image to ensure the independence of the image size and location of the lane lines.
- ✓ The points to draw with polylines function can be sort by Y value which might improve the shape of the line forcing polylines to draw in only one direction.

5. Conclusions

- ✓ The project was successful because it meets the requirements. However, it is necessary to improve and to include more validations in the processing of the images to create a more robust algorithm.
- ✓ The correct chosen and tuning of the parameter in functions like Canny and Hough Transform are critical to determining the lines of the lanes correctly avoiding image contamination.