Project 1 - Finding Lane Lines on the Road

Author: Carmelo Malara Complete Date: 16-Oct-2017
Course: Udacity's Self-Driving Car Nano-degree Program (Entry: 05-Oct-2017) Deadline: 12-Oct-2017

Objective:

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

fig 1. Example image to process (ref: image 2)

fig 2. Example processed image

Supporting Files:

The project's files have been uploaded to the following git repository:

https://github.com/cmalara/CarND-LaneLines-P1.git

They include the Jupyter Notebook, test images, test videos, the processed images and videos:

- notebook: ./P1.ipynb "P1.ipynb"
- test images: ./test_images "images to be processed
- test videos: ./test_videos "mp4 videos to be processed
- test_images_output ./test_images_output "processed images"
- test_videos_output ./test_videos_output "processed videos"

List of figures used in this write up:

- fig 1. & fig 3. ./test_images/solidWhiteRight.jpg "Original image"
- fig 2. ./test_images_output/laneLines_thirdPass.jpg "Ideal version of detected lanes"
- fig 4. & fig 5. ./test_images_output/GRAY_solidWhiteRight.jpg "Gray version of image"
- fig 6. & fig 7. ./test_images_output/Gaussian_solidWhiteRight.jpg "Blurred version of image"
- fig 8. & fig 9. ./test_images_output/Canny_solidWhiteRight.jpg "Canny edge version of image"
- fig 10. & fig 11. ./test_images_output/Mask_solidWhiteRight.jpg "Masked Canny edge version"
- fig 12. ./test_images_output/solidWhiteRight.jpg "Hough lines drawn over original image"
- fig 13. ./test_images_output/solidWhiteRight.jpg "Hough lines drawn over original image"

Project 1 - Finding Lane Lines on the Road

Author: Carmelo Malara Complete Date: 16-Oct-2017
Course: Udacity's Self-Driving Car Nano-degree Program (Entry: 05-Oct-2017) Deadline: 12-Oct-2017

Reflection

1. The Pipeline

In this project, I create and test a 5 step software pipeline to identify lane lines in images of roads. We make use of the Canny Edge Detector and the Probabilistic Hough Transform algorithms as encoded in the OpenCV library.

Step 1. Converting image to Gray Scale

We start by reading in our image, and converting the images to grayscale using the cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) function which returns an image with only one colour channel. The converted image looks like the one below.

fig 3. Original image to process

fig 4. Step 1. Convert image to gray scale

Step 2. Applying the Gaussian Blur

I then applied the Gaussian Blur function cv2.GaussianBlur(img, (kernel_size, kernel_size), 0). Theoretically speaking I am led to believe that this step is optional since the next step, step 3 - the Canny Edge detector function, also applies, by default, a 5 x 5 Gaussian blur. This Gaussian smoothing of the image is a way of suppressing noise and spurious gradients by averaging over a large area. The Kernel has to be an odd number. I am not sure why for now, but will read up on it. The processed image looks like the one below.

fig 5. Gray image of the original

fig 6. Step 2. Gaussian blurred image

Project 1 - Finding Lane Lines on the Road

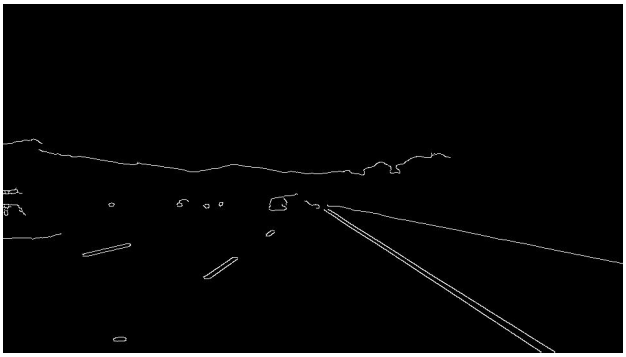
Author: Carmelo Malara Complete Date: 16-Oct-2017
Course: Udacity's Self-Driving Car Nano-degree Program (Entry: 05-Oct-2017) Deadline: 12-Oct-2017

Step 3. Running Canny Edge Detection

Once the amount of colour is normalised and variations between adjacent pixels evened out to keep only genuine transitions, it is time to apply the Canny Edge detector function cv2.Canny(img, low_threshold, high_threshold) to detect the strong edges, i.e. strong edges are where the pixel colour gradient is sharp (falling between the high and low thresholds). The processed image looks like the one below.

fig 7. The gaussian smoothed gray scale image

fig 8. Step 3. Canny Edge image showing edges



Step 4. Masking off the main region of interest (ROI)

Since the lane markings are most likely to consistently be in a particular region of the image from the point of view of the camera observer, there is no point keeping all the other information in the image. With this in mind it makes sense to apply a mask over a trapezoidal region of interest using both the cv2.fillPoly(mask, vertices, ignore_mask_color) function and the cv2.bitwise_and(img, mask). Where vertices describe the points (x1, y1), (x2, y2), … , (xn, yn) etc for the points that describe the trapezoidal form describing the region of interest. This second function keeps only the region of interest as can be seen in the processed image below by applying a bitwise AND only in the ROI..

fig 9. Canny Edge image

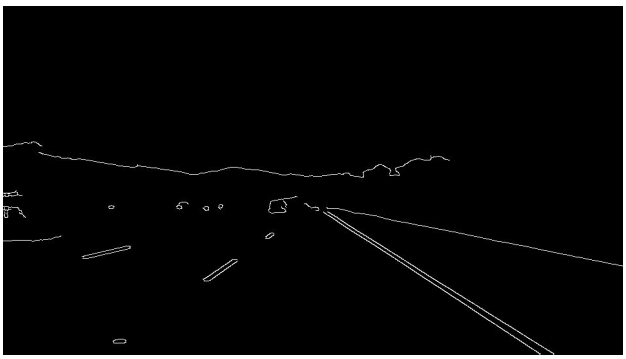
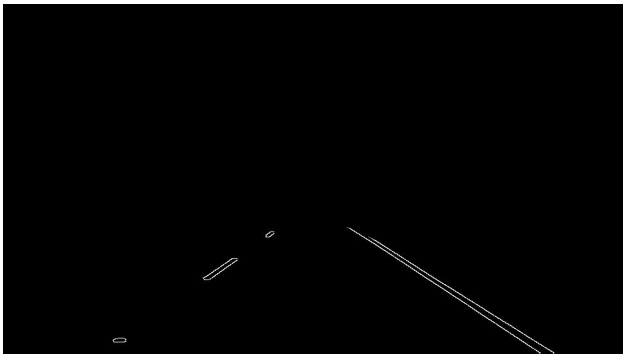


fig 10. Step 4. Masked ROI of Canny Edge image



Project 1 - Finding Lane Lines on the Road

Author: Carmelo Malara Complete Date: 16-Oct-2017
Course: Udacity's Self-Driving Car Nano-degree Program (Entry: 05-Oct-2017) Deadline: 12-Oct-2017

Step 5. Hough Transform to detect the lines then draw them

The final step is to identify the lines using the Hough transform function cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_line_len, maxLineGap=max_line_gap) and then after creating a canvas the same size as the original image using the function np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8), we draw the detected line segments using the function cv2.line(img, (x1, y1), (x2, y2), color, thickness) to obtain the image shown in fig 12.

fig 11. Masked ROI Canny Edge image

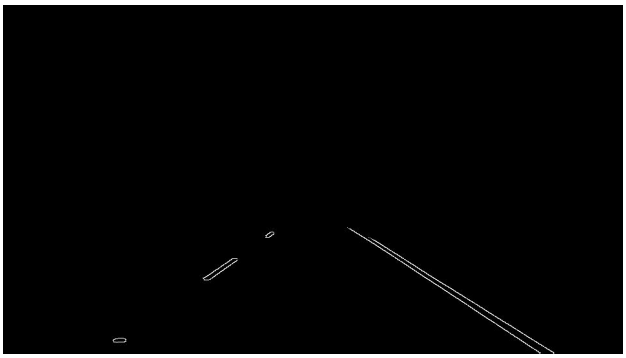
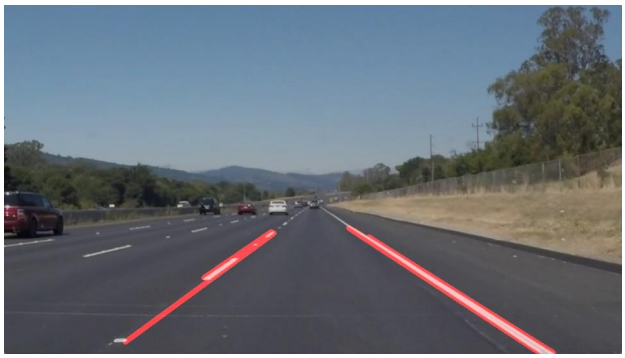


fig 12. Step 5. Lines obtained using Hough transform



The goal was to connect/average/extrapolate line segments. In order to draw single extrapolated lines on the left and right lanes, I modified the draw_lines() function to produce the output shown below.

fig 13. Final image with the extrapolated lane lines

Project 1 - Finding Lane Lines on the Road

Author: Carmelo Malara Complete Date: 16-Oct-2017
Course: Udacity's Self-Driving Car Nano-degree Program (Entry: 05-Oct-2017) Deadline: 12-Oct-2017

2. Identify potential shortcomings with your current pipeline

One potential shortcoming with my method of extrapolating lines is the potential to have several frames with lost lane lines. This happens when searching for a min x and min y value in the list of lines detected by the Hough transform, we are at the mercy of the Hough transform actually detecting some lines on the lane, otherwise we end up with frames where no detection took place and lost lane lines.

Another concern would be that the algorithm would not be able to cope with extraneous features interfering with the lane lines. Such features could be shadows cast across the image or changes in road surface paint as was seen in the challenge video.

3. Suggest possible improvements to your pipeline

Since it is more important to have an algorithm that is stable over a perceptible period of time (roughly similar to the speed of the vehicle) than to expect lanes to change as quickly as the video stream frame rate, and to cope with dropped frames and lost lane lines, a possible improvement could be the addition of an averaging mechanism to desensitise the algorithm's response.