ECE498 SM MP1 report

Mike Schaub (meschau2) and Shuijing Liu (sliu105)

Explanations of each block of the code

- warpImage
  We found 4 points on the both ends of two lanes by eyeballing, then passed them to cv2.getPerspectiveTransform function as source (src) and got the transformation matrix for the bird view image.

- Laplacian
  To apply Laplacian filter to an image, we first transformed the color image to grayscale by cv2.cvtColor function, then Gaussian blurred it with filter size = 5, and finally passed the result image to Laplacian filter of size 5.
  Then in order to convert the image to a binary one, we surpassed the pixels of the image which are lower than the minimum threshold or higher than the maximum threshold, and forced the rest of the pixels to 1. We returned the result binary image.

- Sobel
  To apply Sobel filter to an image, we converted the image to grayscale and blurred it in the same way as the Laplacian function above. Then we passed the result image to cv2.Sobel twice to get gradients in x direction and y direction. Then we used cv2.addWeighted to add x gradients and y gradients, and convert the result image to binary in the same way as the Laplacian function.
  We did not use the scaled_lap variable because the result image looked bad. Instead, we just added the return value of cv2.Sobel and converted the sum to binary image.

- colorThres
  To detect lanes with certain colors in an image, we first converted the image to HSV space. Then we camp up with two sets of thresholds for H, S, and V: one for the yellow lanes and one for the white lanes. We passed these thresholds to cv2.inRange to get a yellow filter and white filter. We ORed these two filters together to get a result image containing only lanes.

- combinedEdgeDetection
  To combine the above three edge detection methods, we passed the input image to Laplacian, Sobel, and colorThres functions, and got an output binary image

from each function. Since Laplacian filter does not work well with the provided input images, we combined the outputs of Sobel and colorThres by ORing their outputs together, and returned the combined image.

- findConvCenter
  To find the left centroids and right centroids which are key points on the lanes we detected, we divided the image into 10 layers in vertical direction and find one left centroid and one right centroids on each layer.
  We found the starting left and right points by summing up the bottom half of the image and getting an array of size (image width, 1), convoluting the sum with an array of 1s, and find the argmax of the left and right half of the array as the first left and right centroids.
  Then for each layer, we found the argmax point within a margin of the previous centroid as the new centroid. We iterated this loop for 9 times to get the left and right centroids of each layer, and returned the list of centroids.

- drawPoly
  To fit the set of centroids we found in findConvCenter above to a quadratic line, we used the np.polyfit function to solve a, b, and c of the quadratic equation y = a*x^2 + b*x + c. After generating 1080 numbers from 0 to 1080 in y direction, we used the a, b, and c to solve for the 1080 corresponding x values. Then we threw away (x, y) pairs that are out of the range of our warp image, used cv2.fillPoly to draw a quadrilateral on the warp image from the remaining points, and transformed the warp image back to the original perspective using the Minv matrix. Finally, we put the result image in original perspective on the top of the original image so that it displays the lane in red on a road.

Glitches and artifacts of our code

- For the Laplacian function, we never got good result. It's probably because the original image contains too many edges pointing to all directions, while if we keep increasing the filter size, the image will be over-blurred.
- For the combinedEdgeDetection function, our result image contains a lot of noise on the sidewalk, because the color of part of the sidewalk is too close to the white lane that we need to detect, so it is hard for the threshold to exclude all of them.
- In the final result of rh1.png, we only detected the left half of the lane. It is because in the findConvCenter function, the algorithm detected the letter P in the word "STOP" instead of the right lane marker. The vertical line of letter P is very

strong so that it becomes the index of argmax when we found the starting right centroid.
- The simulation freezes in the half way and the simulator gives a warning about bad condition number of polyfit.

Typos and room for improvement of the given code

- In the warpImage function, the position of (x1, y1) ~ (x4, y4) is not clear: for example, which is in the upper left or lower right. We had to adjust the order when passing the four points to cv2.getPersepctive function.
  Also, the resolution of the image displayed in jupyter notebook is very low, so that it's hard for us to eyeball good key points. I suggest we change to .py file for the next MP, so that it's more compatible with IDEs, which are very helpful in debugging.
- In the colorThres function, we need 12 thresholds for both yellow and white color, so it's better to pass 2 lists as arguments instead of 6 individual thresholds.
- The instruction of findConvCenter function is confusing. Also, the leftEmptyCount and rightEmptyCount variable are not used anywhere, but they were declared and initialized in the given code.

Photos of simulation and the scenarios where our code is not showing proper results

The simulator never showed up in the Righthook simulator, we can only see a grayscale image in the simulator. Tianqi suggested we will record the video later on piazza.