Finding Lane Lines on the Road

Reflection

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My image pipeline had the following stages. In the following, I am showing intermediate images of each output. The original image is the following.

1. Grayscale conversion

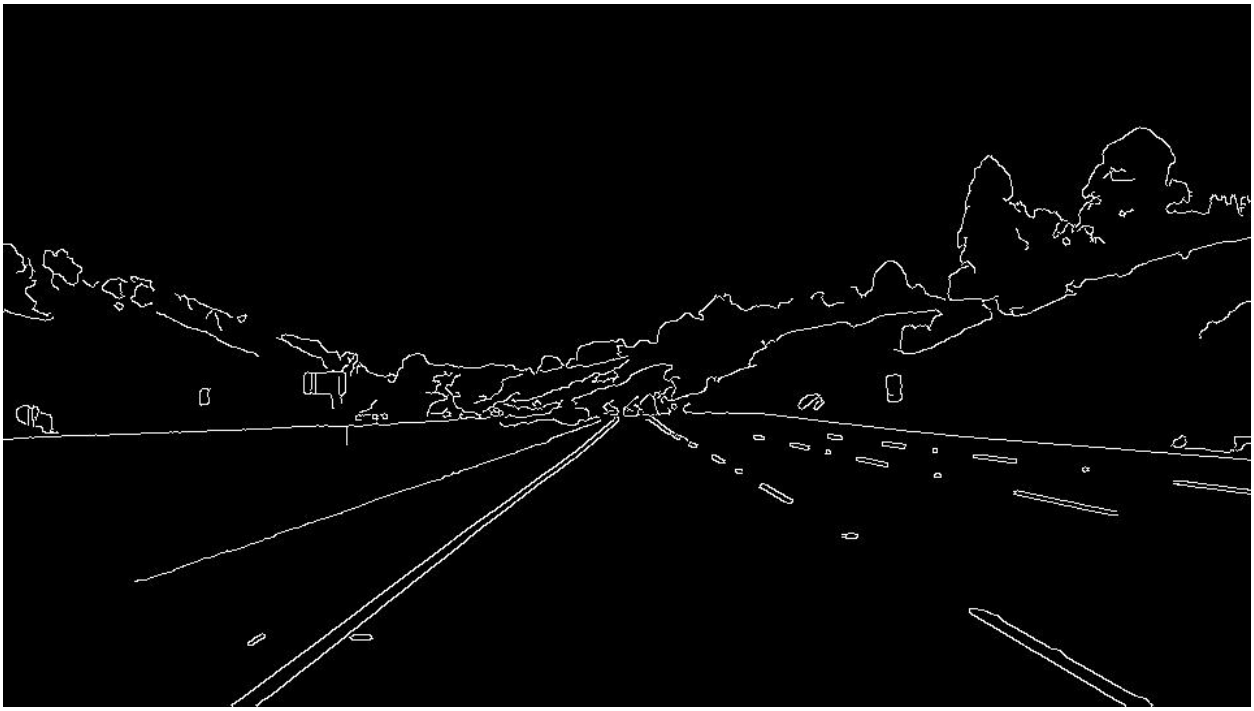
Grayscale conversion is a simple conversion of the image to grayscale. The edge detection works better on grayscale image. A typical grayscale image of a highway would look like the following.

2. Guassian smoothing

I did Gaussian smoothing using the gaussian_blur function. I didn't find any noticeable different in the final lane detection when I played around with values like 3, 5, 7, 9, etc. I chose to keep kernel_size = 5. After Gaussian smoothing the grayscale image would look like the following. Note that there are no more sharp edges in the image.

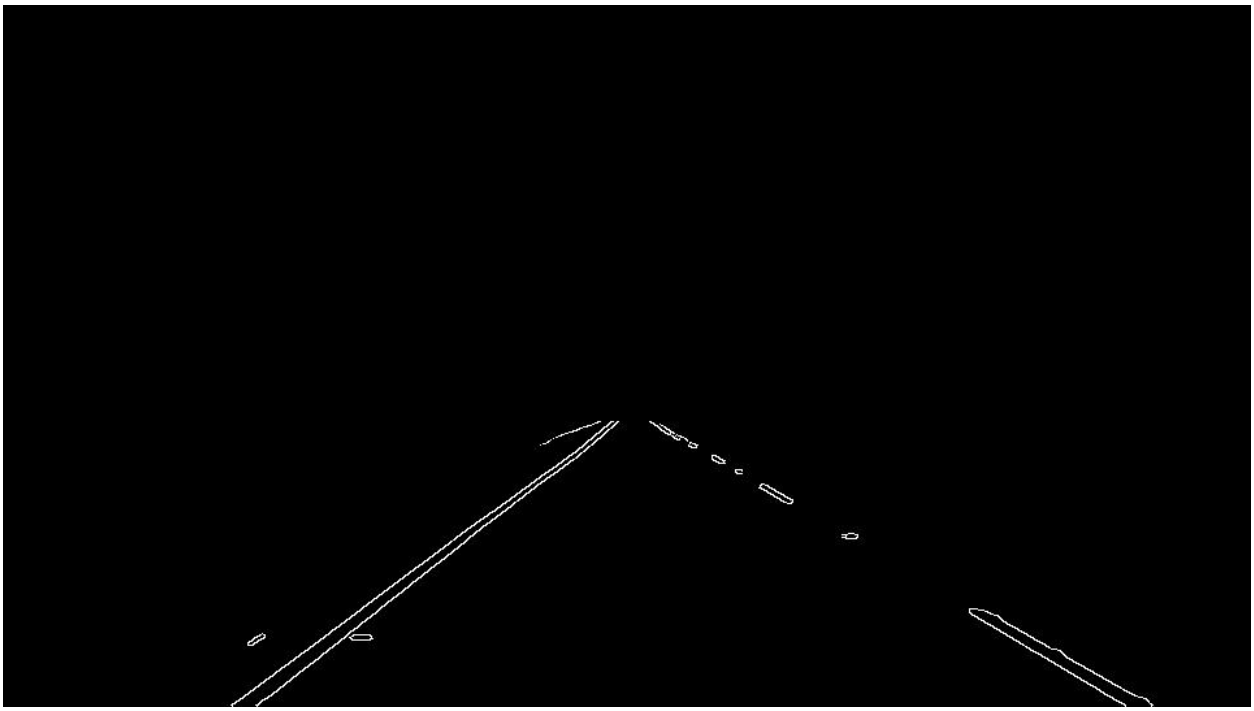
3. Canny Edge detection

For the Canny edge detection, I used low threshold of 50 and high threshold three times of that. Here also, I didn't find any real difference when I used 60 or 70 instead of 50. The edges in the image looks like the following. The edge detected image looks like a very plain outline drawing of the highway. A lot of details are gone which were not needed for the lane detection.



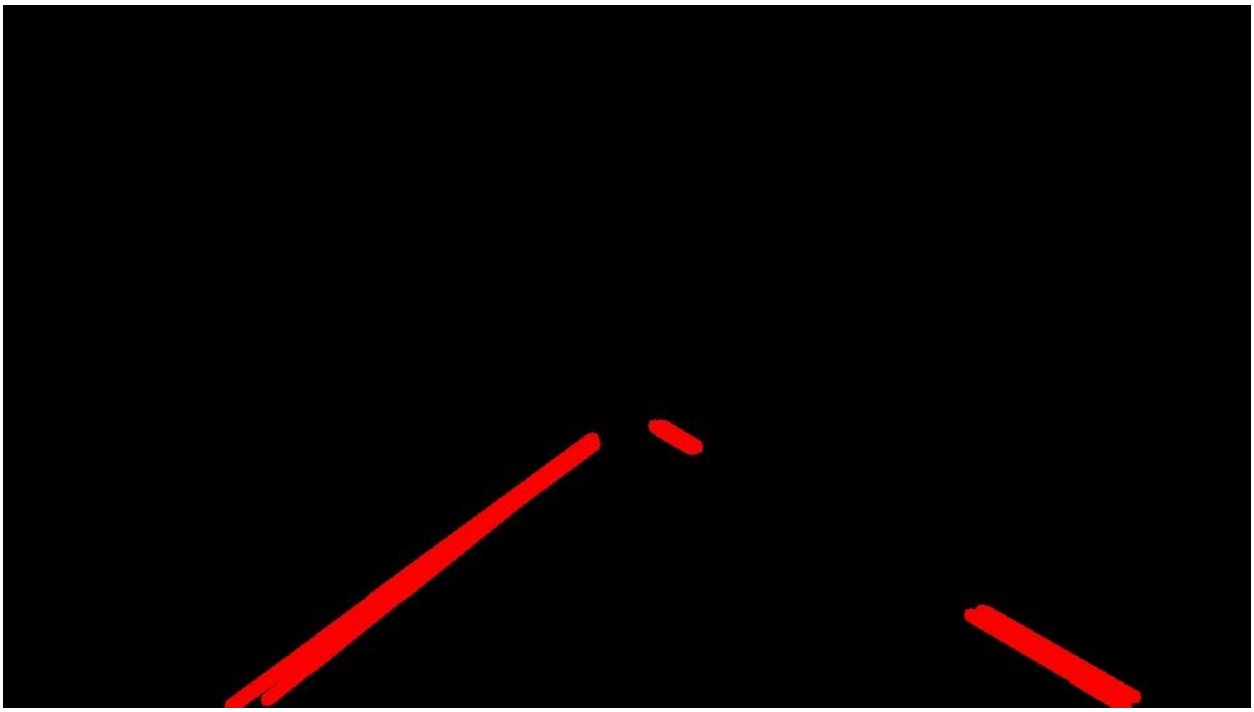
4. Creation of a masked edged image

For masked image, I used the fillPoly function to carve out an area where I would be looking for lane lines. The vertices I chose for an image of size 540x960 are: (0,540), (450,320), (500,320), (960,540). From the above image, I remove the landscape portion and keep on the highway / road.



5. Finding lines using Hough transform

Hough transform can be used to find lines in the above picture. The HoughLinesP function has many tunable parameters and changing each of the greatly varies the result. After applying Hough transform the raw image looks like this.

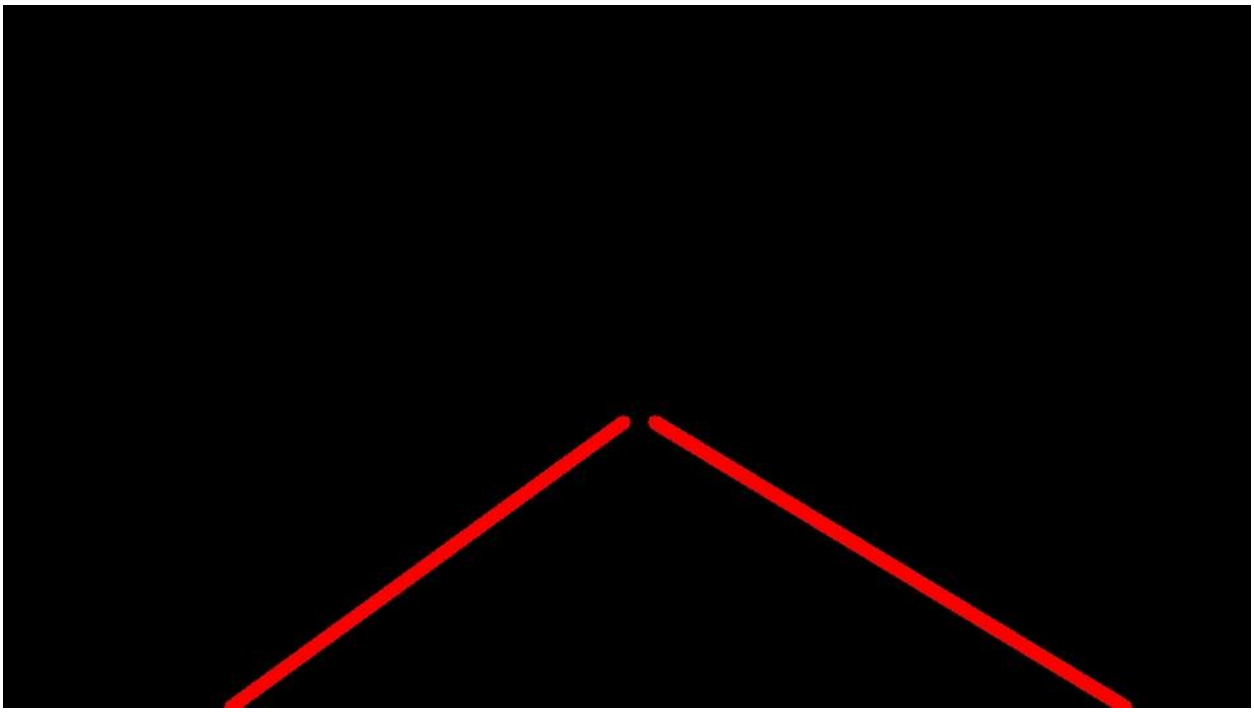


One can see that not all the lines are picked up in this image. This is because of the values, I have used in the HoughLinesP functions. By playing with the parameters, we can make it more or less sensitive to lines. There are two things to notice in the above picture.

1. Not all lines are detected. As I mentioned that is because of the parameters, I have used.
2. The highway lines are shown as two overlapping lines. This is a bit annoying that it shows up as two lines. It is because, the Canny edge detector finds two edges on either side of the line. And HoughLinesP things those are two separate lines.

6. Process lines to get a more consolidated line

Much of this project's work goes on this item. To convert the above image, which is broken line segments, to a single line starting from the bottom of the picture. The image, in the end looks like this.



There are two approaches, I tried here.

1. Try to get the first and last points. The first point would be the top most of the line which should touch the top of my region-of-interest. And the last point would be the bottom point which touches the bottom edge of the image. Once we have both these points, simply draw a line joining them.
2. Another approach is to join the small pieces of line segments to form a bigger / longer line. The resultant line may not be straight. And to create such a line, we may need to fill intermediate gaps with lines.

I have done all my processing targeting the second approach. I create an array of points where I fill the line gaps also. The same approach can be used to create the straight line from approach 1 also. All I should do is to join the first and last points from that array of points.

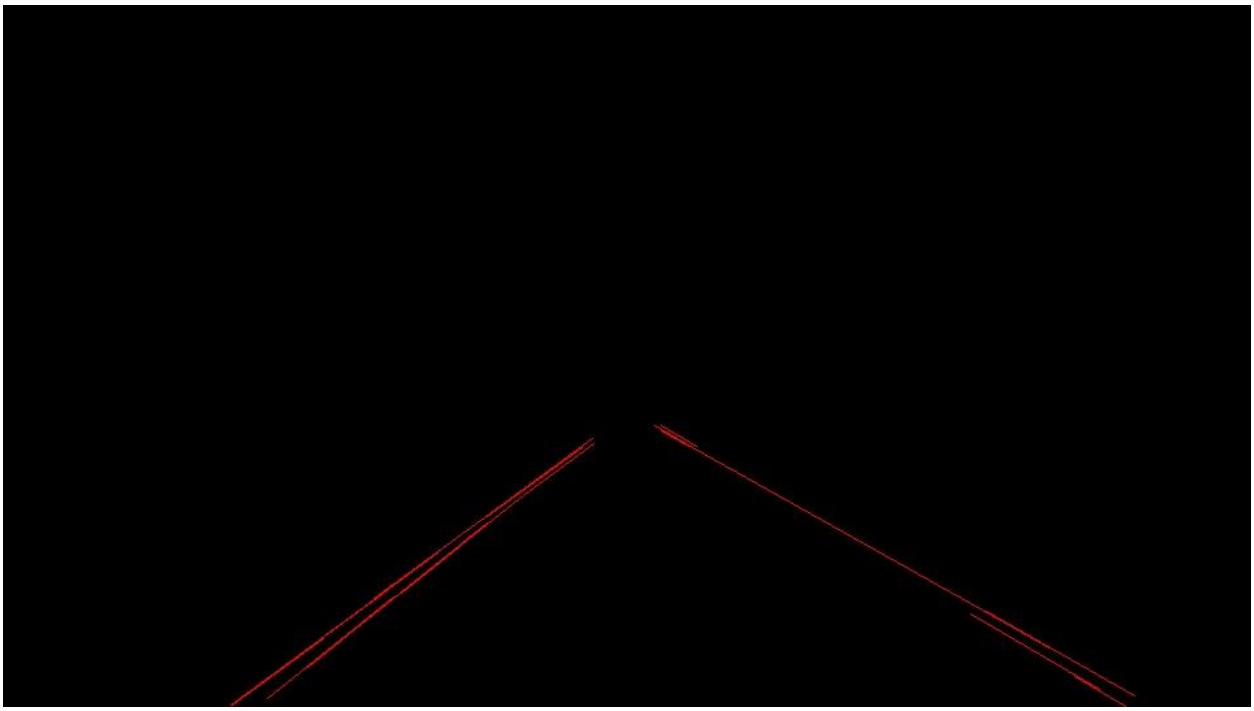
In the process of creating intermediate lines sometimes we get bogus lines, which we must get rid of. One good way is to look at the slope of all the lines obtained and then take a mean of those slopes. Then use standard deviation method to eliminate the unwanted lines. I took a lazy approach where, I decided to keep lines of the slope 0.5 to 0.85 and throw away anything else.

I will show my entire approach in the following series of images.

The below image is obtained after I separate out the left and right lanes. I use the slope values to decide if it is left of right lane. Right lane has positive slope value and left lane has negative. In addition, I look at the position of the line also. If it on the right half of the image, then I assume it is right lane, else left lane. Here, I am assuming that the camera would be places in the middle of the car and the car is running in the middle of the lane. Also, note that I am using thinner lines to show the details of the process. Later, in the end I make the line thick.

I found that the list of lines given by the HoughLinesP is not in sorted form. By sorting, I mean, the top most line is the first in the list then the next, then next, like that. So, I go an sort the line based on y values. So, early entries of the array would represent top lines and then bottom ones. Once I arrange it in an order, inserting new lines in the gap becomes very easy.

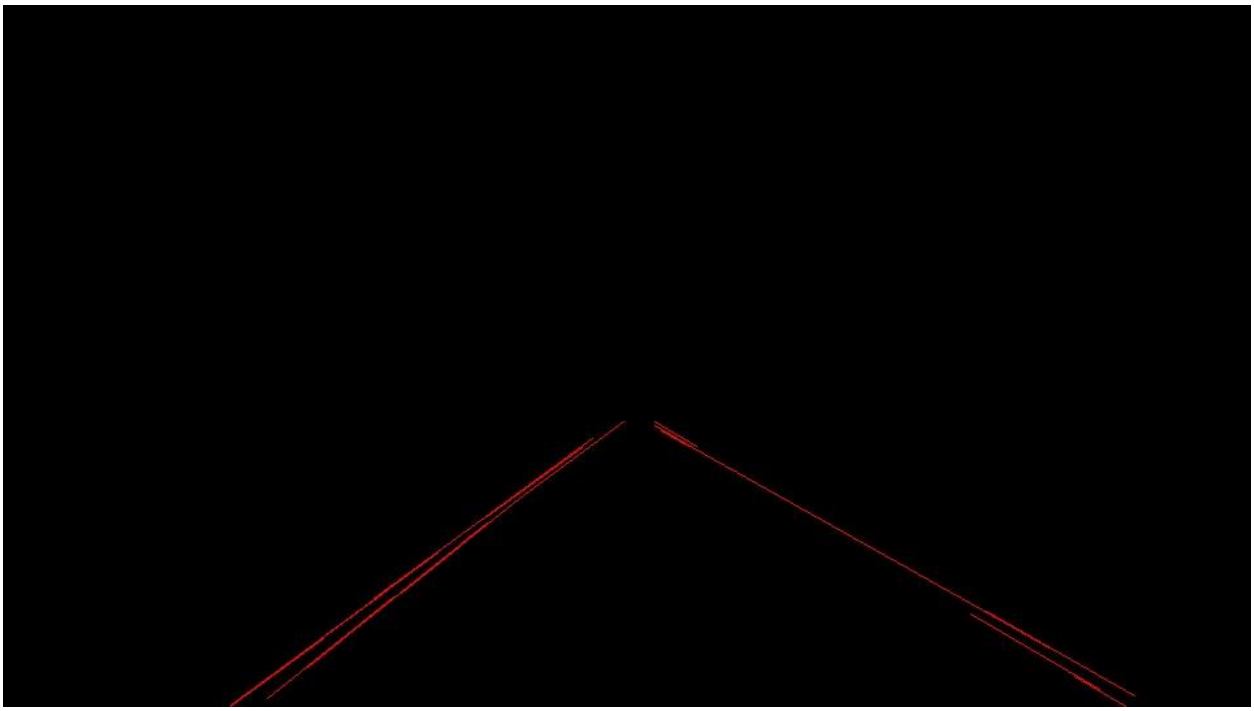
Once sorted, I add intermediate lines for filling the gaps. I do this by creating a new line with (x2,y2) of previous line and (x1,y1) of next line. The resultant image would look like this.



I look at the slope of the new line to make sure it is between 0.5 and 0.85 as I have previously decided.

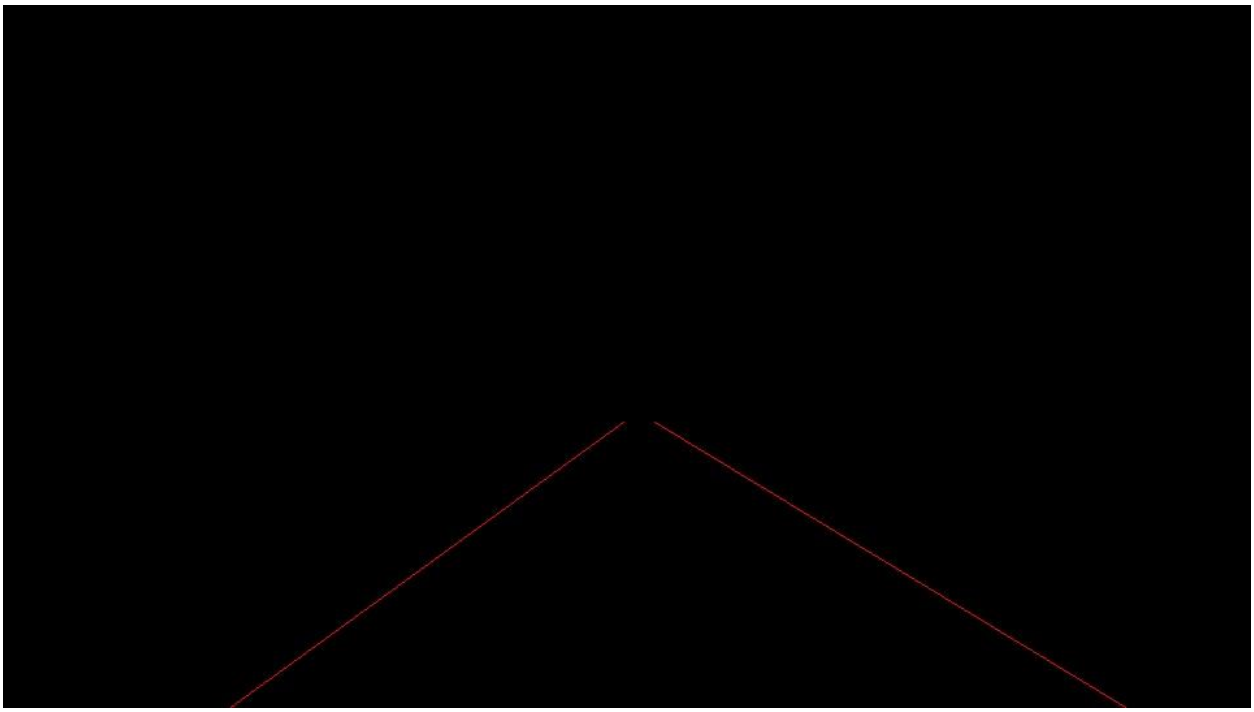
At this point, if the line doesn't touch the bottom or the top of the region of interest, I extrapolate the line. In this example, the lines are touching top and bottom of the region of interest. So, it would be hard to see any difference.

Now, finally, I join all the lines. In this example, it would look like the above image. I am showing the image again below.



If you see, there are multiple lines. Not only the final line is a joined line of all the lines, there are parallel lines running together. This is because, I didn't find a good method to eliminate one of them.

At this point, I give up and go for the first approach, I mentioned earlier. Have only one line. And since, I have a sorted array of lines, I can simply create the final line by selecting one point from the first array entry and one point from the last array entry. It would look like this.



7. Draw lines on top of the image

Now, the job is almost done. Just add this line on top of the real image. And change the thickness also.

2. Identify potential shortcomings with your current pipeline

I have identified the following issues with this pipeline.

1. Doesn't work properly when the road turns. The video output I generate for extra-credit problem looks disastrous.
2. My pipeline doesn't eliminate one of the line edges. The edge detector, finds two lines for left and right edges of the lane markers. And thus, I get two lines. This pipeline doesn't eliminate it. Thus in the video, when the frames start changing sometimes the algorithm picks left edge and sometimes right edge. The overall edge line looks very jittery in the video.
3. My pipeline is compute intensive as I was trying to get smaller line segments to join and create a lone line. Finding two end points directly would have been easier.

3. Suggest possible improvements to your pipeline

Obviously, the above two points I suggested in the shortcomings should be improved.

1. Overall to take care of turns, lot of improvements should be done.
    a. Adaptive region of interest, which is not a static 4-side polygon.
2. Better smoothening algorithm to take care of gray color road (which is there in the challenge question)