1. Describe your pipeline.

My pipeline consisted of 7 main steps. Firstly, I converted the images to grayscale using the cv2.cvtColor() function to work with one colour channel. One colour channel provides adequate data to detect lane lines in an image.

In order to use the canny line edge detection function, I firstly applied a filter to the grayscale image using the gaussian_blur() function. This reduces noise in the image. I selected a kernel size of 5. I then used the canny() function to detect the edges of the lines. I selected a low threshold of 50 to exclude pixels below this brightness value and a high threshold of 150 to find pixels above this brightness value. Any pixels in between these two values which are connected to pixels with a brightness of greater than 150 will also be included in my output image. The output of the canny() function can be seen in Figure 1. below.

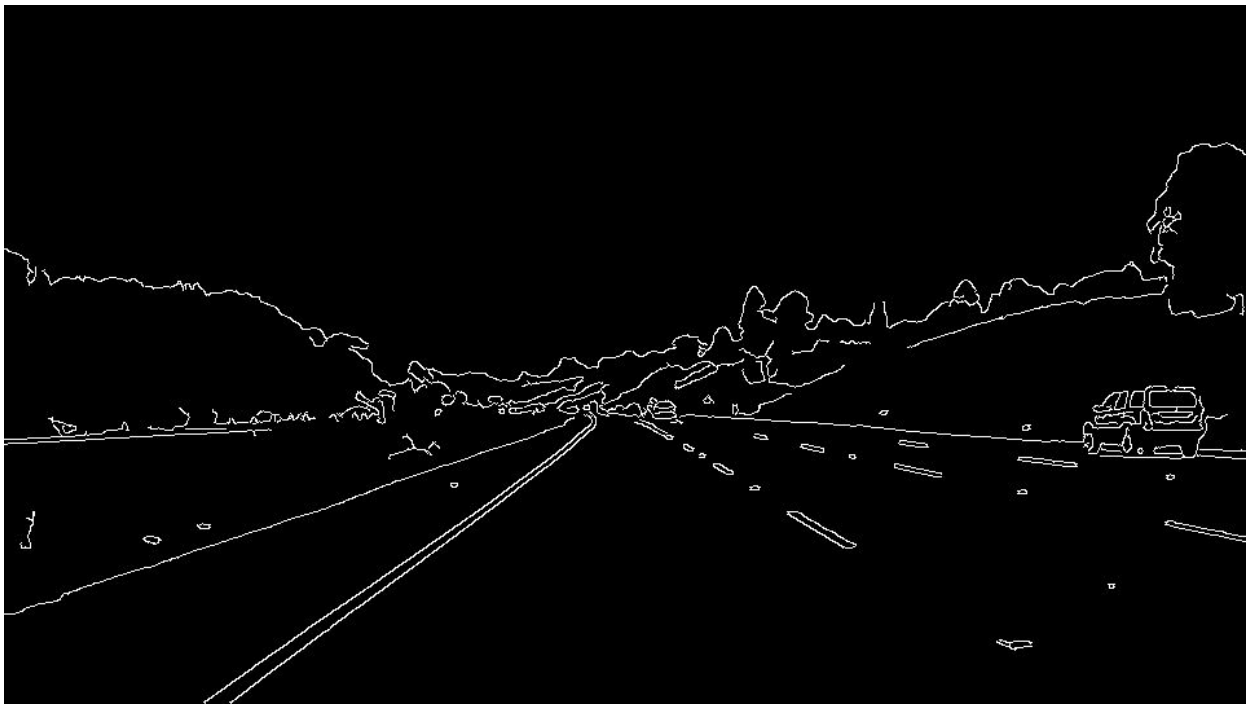

Figure 1. Canny Image

I then used the cv2.fillPoly function to define an area of the image where I wanted my algorithm to search for lane lines. I used a four sided polygon to apply a mask to each of the images as can be seen in Figure 2. Below.

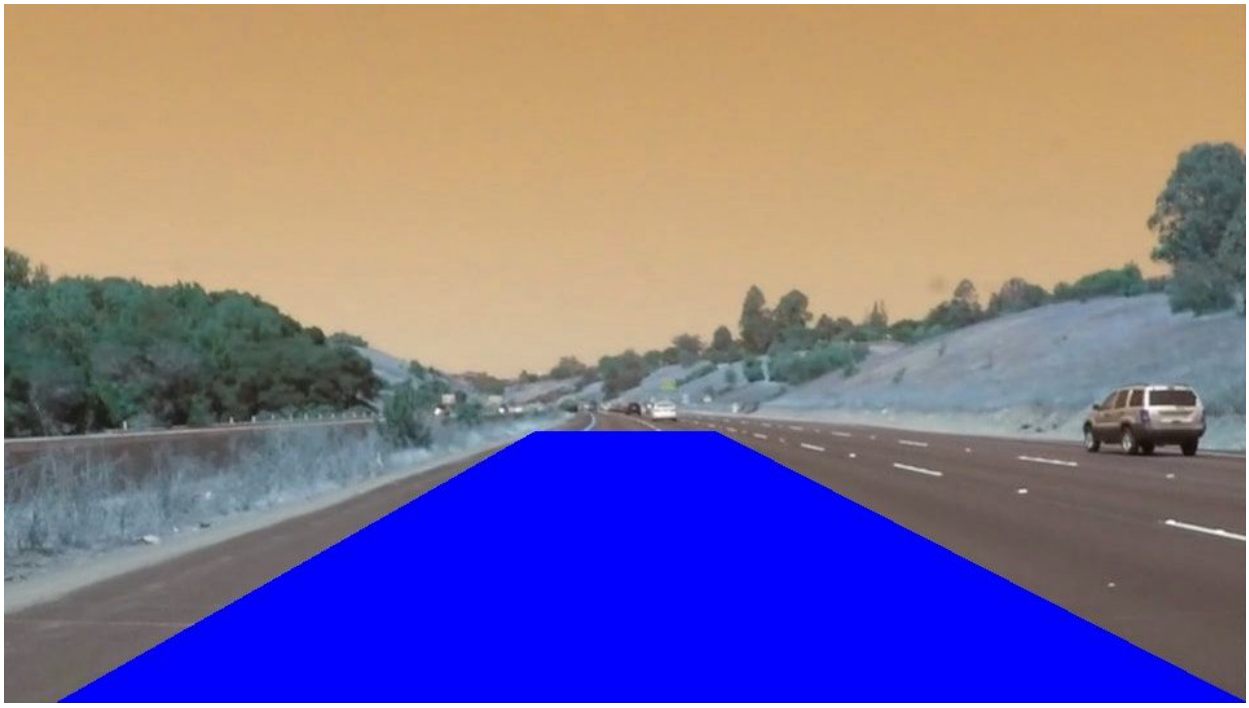

Figure 2. Area of Interest

I then used the hough_lines() function to identify lines in the canny image of the same slope and intercept. I selected the following values for the hough function parameters.

- ϱ = 2 pixels
- θ = 1°
- Threshold = 15
- Min line length = 40 pixels
- Max line gap = 40 pixels

This gave me the result shown in Figure 3. below.

Figure 3. Output of Hough function.

The next step was to the extrapolate the identified lane markers in order to have a continuous line identifying the right and left lane markers. In order to separate the left and right lane marker I used the slope of the lines to identify which lines belonged to the left and right side. The left line has a negative slope and the right line has a positive slope. I also set a minimum slope of 0.4 to disregard any lines which were close to horizontal. I then combined all the endpoints of the lines which had a positive and negative slope. I then took the minimum and maximum values to find the endpoints of the lines which made a continuous line for the right and left line.

The lane lines must be drawn from the bottom of the image. In order to do this I found the point where the line intersected with the bottom of the image. I found the intercept and slope of the line. I then used the equation of a line and set the y coordinate to the height of the image and found the x coordinate where the line intersected with the bottom of the image. I then used these coordinates to draw the lines on both sides of the lane. These lines were drawn on a blank image of the same size as the original image. In order to make the lines semi-transparent I used the weighted_img() function to combine the lines drawn on the blank image with the real image. This gave me a final output image shown in Figure 4. below.

Figure 4. Final Output

I then applied this pipeline to the video output which annotated lines to both videos (insert videos here). This gave me a result similar to (reference example here).

2. Identify potential shortcomings with your current pipeline

This current pipeline works well on long straight roads. In order to detect lane lines on curvy roads with the current pipeline I would have to reduce the search area. This may not give adequate information in time for a controller to smoothly control the vehicle.

The lane detection in the video is is not very smooth. It would preferable to reduce the effects of disturbances in order to get a smooth output. This may also not be suitable to use as data to give to a controller.

3. Suggest possible improvements to your pipeline

One possible improvement would be to improve the drawing of lane lines to accurately draw lanes on a curved piece of road. This could be done by fitting a polynomial line to the images.

Another improvement would be to reduce the effect of disturbances on the output. This could possibly be done by doing some more filtering on the output of the grayscale function. Further tuning of the canny and hough function parameters could also improve this.