Reflection

Pipeline

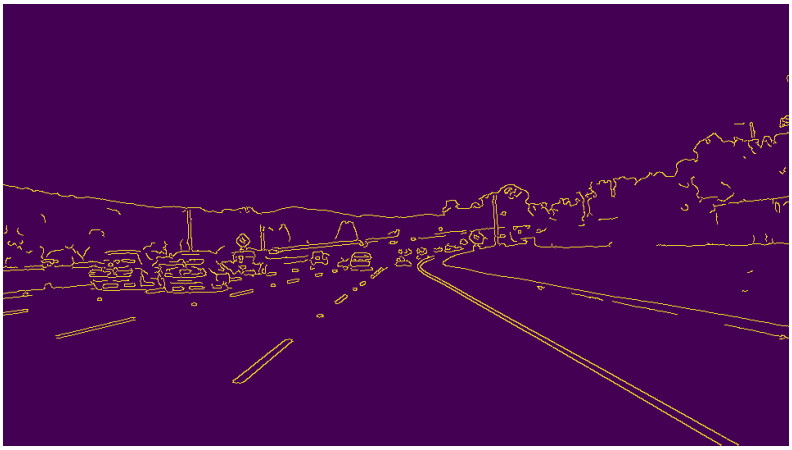
My pipeline consisted of the following steps:

1. Grayscale conversion
   Starting out with an image like the one below, we convert it to grayscale.
   
   Although the image below looks green, it has only a single color channel. It is a direct output from the CV function to convert to grayscale.

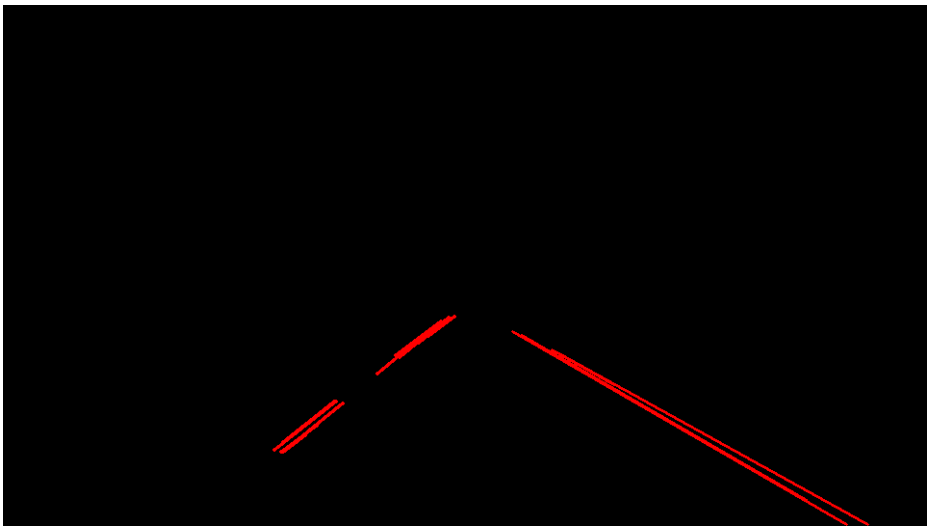
2. Gaussian Blur
   The second step in the pipeline is to apply a blur to the image to remove some of the noise. Care must be given here not to blur away the edges we are going to detect.
   

3. Canny edge Detection
   The next step is Canny edge detection. The output of this algorithm is a bitmap containing intensities related to computed gradient values. This algorithm as implement in the CV library has two parameters, low threshold and a high threshold. Setting these values allows you to "dial in" the right amount of detail. In my pipeline, these values where set through experimentation.



4. Masking a region of interested
   The next step was to mask off an area of interest. This is really an optimization to prevent the Hough Line Detection step from detecting lines where they would only be discarded.

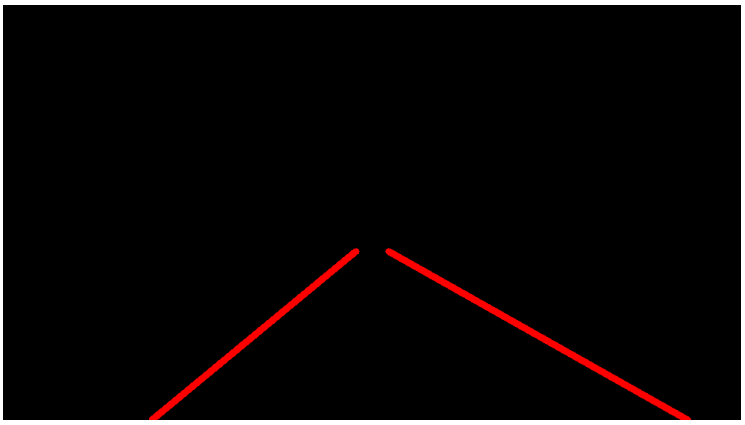
5. Hough Line Detection
   Hough Line Detection was used to detect the lines. The algorithm parameters, rho , theta, threshold, min ling length, and max line gap were all determined by trial and error. After many trials, I finally settled on the settings I felt gave me good results. The result including the area masking is below.



Below is the final composite of the original image with the detected lines.

Improving the draw_lines function

In addition to developing the pipeline, we were asked to make enhancements to the drawing routine so that it would draw full lines that would map out the full extent of the lanes instead. This is an improvement over the line segments produced by the Hough Line Detection. To do this, first I computed the slopes and y intercepts of the lines provided by the Hough line detection. I then computed averages for all these values. Next I needed to determine coordinates for the lines that would be my lane guides. For this I used the average slop and average y intercept. I then set y values fixed at the bottom of the image frame and a spot just below the center on the y axis. Using these locations, I computed corresponding X values. These provided coordinates I needed. See images below.



This looked reasonably good for the still pictures but appeared very jittery when applied against the video sequences. Finding way to smooth the changing slope and intercept values was an interesting problem.

There are many ways to make motion appear more smooth in video. One approach may have been to take into account previous line values and use them to help smooth out the averages. I saw talk of this approach on Slack. Nevertheless, I opted to look for a way to get better results strait out of the algorithms themselves.

I experimented with various values for Canny, Hough and even Gaussian blur. None of them really produced the smooth results I was hoping for. From there I realized some of the lines out of Hough were throwing my averages off. Maybe there was a distinct mark in the road, or some other artifact. There were also lane markings that were perpendicular to the lines we were tracking. Because of this, I filtered out lines with slopes that were not going to work. Anything less than 0.3 or so would be thrown out. This helped some but the output still didn't look like I wanted.

After a bit of experimentation, I noticed that the best line approximations seemed to be closer to the camera where the relative size is bigger. I then changed my average to a weighted average using the distance between the points of the line for the weights. This seemed to give me better results. While the results are still not as smooth as I would like, I decided this would be good enough for a first try.

Pipeline Shortcomings

I noticed the following shortcomings for this pipeline:

1. My pipeline as implemented does not display smooth slope or line transitions from frame to frame
2. The algorithm could easily be confused by other markings in the road that are not lane lines. I can see things like truck skid marks being a real problem
3. The algorithm does not handle curves.
4. I don't think it would handle obstacles like other cars very well

Potential Improvements

Potential improvements may include the following:

1. Add color consideration when detecting edges. This may be problematic however for reasons stated in the lesson.
2. Add ability to detect curves.
3. Add a way to smooth line transitions from frame to frame. Averaging with previously detected lines seems to be a good approach.