Reflection

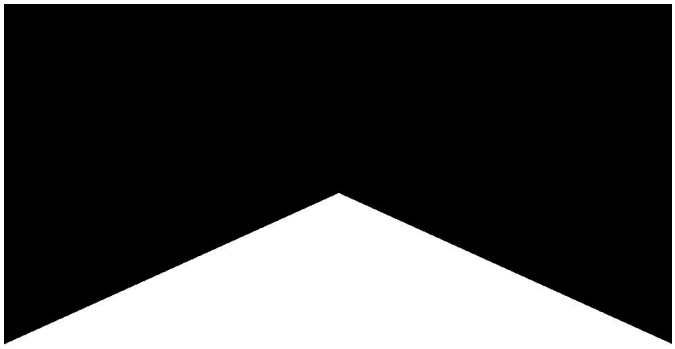
The pipeline consisted of 5 steps :

1. Convert the image to grayscale to deal with gradients on only one channel

2. Apply blur and then Canny edge detection
    a. We can play around with the blur kernel size or the canny thresholds to get similar results
    b. In my submission I have increased the blur threshold to 11 to filter out edges with relatively low gradients

    Result of Canny edge detection -

3. After canny I have applied a pentagon mask which looks like this



4. After masking out the Canny results I detect lines using the Hough Transform
   The detected lines look something like this

5. The lines are then filtered with regards to their slope and x intercept
   a. Candidate lines for left lane
      i. All lines between slopes 30° and 60° with the positive x axis
      ii. Lines with x intercept on the left 75% of the image
   b. Candidates for the right lane
      i. All lines between slopes 30° and 60° with the negative x axis
      ii. Lines with x intercept on the right 75% of the image

   The slopes and biases for the final two lines are found by averaging out the slopes and biases of the candidates for the left and right lane lines.

Shortcomings

1. Since the detection pipeline is heavily dependent on gradients on grayscale images, any high gradient in the region of interest affects the detection pipeline. Eg - gradients due shadows of other moving vehicles on the lane, or trees on the sides
2. The process of averaging out candidate lines to find the final lane line, is prone to errors from outliers. Outliers can affect the final line to a huge extent, if the number of candidate lines is relatively less.

Improvements

1. Instead of averaging out all the candidate lines, clustering techniques can be explored. We find the biggest cluster in which the candidate lines lie, and then average out lines in that cluster.
2. In some cases finding the median line instead of averaging them out could be helpful.