Finding Lane Lines on the Road

Reflection

1. Pipeline:
   My pipeline has the following steps:
   1. Converting image to grayscale. This helps to reduce color variations (basically color variation translates to pixel intensities) and a number of downstream steps (specially edge detection) works well.
   2. Reducing details from the image using Gaussian blur. This is required to get rid of unwanted details from the picture to enable edge detection step to focus on bigger object boundaries. The amount of blurriness is controlled through specified kernel size.
   3. The next is to identify edges and Canny edge detection algorithm is used here. This works based on change in pixel intensity wrt its neighbors. The intensity difference is controlled via two parameters – low and high thresholds.
   4. Now is the time to focus on the particular region of the image where lane lines will practically occur – lower middle part. Rest of the image is masked at this point so that we can focus on edges belonging to this part only and these are most likely to be lane lines. Selecting the region of interest is tricky and figured out via iterative experiments.
   5. In the next step we focus on edges and try to find pattern in them – particularly line segments which can potentially be joined to build up the lane. The method used is called Hough transform. Internally this algorithm projects each edge from x-y plane to m-c (slope-intercept) plane to figure out segments of same line. The output of this step is a list of pair of points where pair of points signify a line.
   6. Next comes the most interesting part where we need to analyze these lines, figure out left and right lane lines and finally draw it on the initial image.
      a. In the beginning of this stage, the slope of input line segments (coming from Hough transform) are examined. +ve slope indicates potential right lane and -ve slope indicate potential left lane. A bunch of points can be thrown out at this stage which have slopes we don't care about to reduce noise.
      b. Now we have two set of candidate lines for right and left lane. The candidate lines are then averaged out and two lanes are drawn on the image (hypothetically).
      c. To ensure consistent display, the intersect is these two lines are computed and trimmed little bit at the top.
      d. Also the lines are extended till the bottom of the image.
      e. For video, lane line locations are averaged out across frames to reduce jittery change in lane lines.
      f. At the end, the lane lines are overlaid on the original image.

2. Potential Shortcomings:
   The main shortcoming of my approach is figuring out lanes on a road where lanes are not properly marked, faded away or not properly visible. If that is the case Canny won't be able to figure out edges and rest of the flow will break. This is also visible in Optional Challenge video where my output is not as stable as I like due to different lighting condition and almost invisible lane in few frames.

   Also my current implementation has been tuned wrt to the pictures and videos provided for testing. A bunch of parameters has been chosen and for a different test set these values might break or not perform well. If I think about real life scenario, the location of the camera capturing the images, lighting condition and road condition will impact test image or video.

3. Possible Improvements:
   A possible improvement to handle twisted roads would be to break down the pictured into smaller windows, cluster the detected lines and figure out lane segments in each of them. Later join them to create the curve.

   To handle different lighting situations (dull or overexposed), we can preprocess the image to enhance or reduce pixel intensity so that edge detection can work properly. Also I feel we need to be smart about reducing noise before and after Hough transform after studying many situations.

   Another approach to solve the same problem is supervised learning.