Advanced Lane Lines

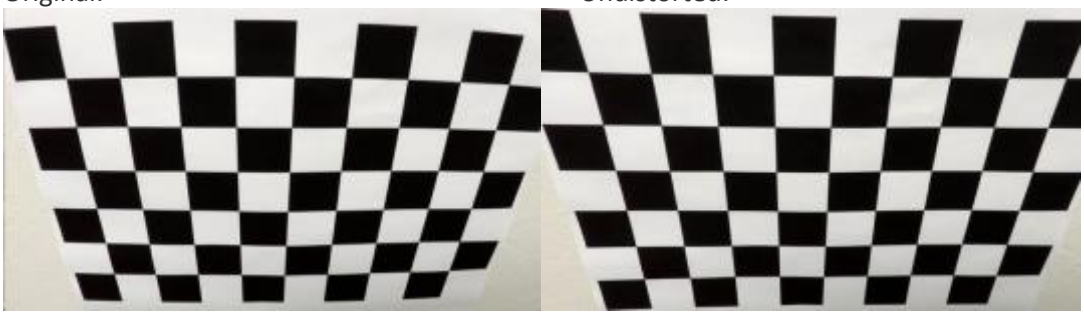
Files

- Image_generator.py – feed static images from file system into pipeline
- Video_generator.py – feed video into imaging pipeline
- Image_processing.py – defines image pipeline
- Line_tracker.py – class used to find lane lines on a binary image and keep track of lane line information from previous images
- Calibration.py – defines process for finding distortion matrix / coefficients based on 6x9 checkboard images

Camera Calibration

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I saved the distortion matrix and coefficients in a pickle file to be used in my image pipeline. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:

Original: Undistorted:



Pipeline:

- Distortion Correction
    - I used cv2.undistort, and my pickled mtx and dst from the calibration step to undistort the images

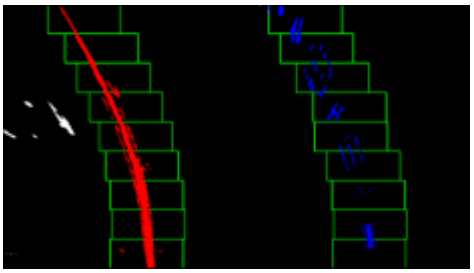
Original: Undistorted:

- Combing Thresholds
    - X & Y Gradient – used to detect steep edges that are likely to be lane lines
        - X_Grad – sobel kernel size = 3, threshold = (12,100).
        - Image_processing.py line 14-30, 87

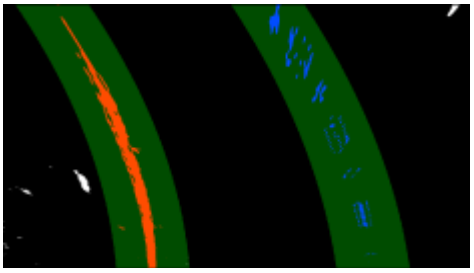
    - HLS Color – the HLS (Hue, Lightness, Saturation) color space, and specifically the S-channel, gives us a robust way to detect lane line pixels. Threshold = (100,255)
        - Image_processing.py line 66-71, 91

    - Combining Thresholds – (X_binary) | (Color_binary) – this combination of thresholded binary images allows me to detect lane lines in a variety of lighting conditions
        - Image_processing.py line 93

- Perspective Transform
    - Trapezoidal region of the image based on hand-selected percentage height / width
    - Used cv2.getPerspectiveTransform to get transform matrix and inverse matrix
    - See below for example images
- Lane Line Detection
    - When starting, or when we have not detected lane lines with “high confidence” (defined below) in the past 4 frames – use sliding window technique to identify non-zero pixels (possible lane line pixels) and adjust the sliding window accordingly.
    - 9 sliding windows, 100 pixel width, 50 pixel minimum to adjust sliding window



- If detected lane lines with “high-confidence” in one of the last 3 frames, then we do not perform the sliding window technique, and instead use the average of the last 3 “high confidence” lane line polynomial fits, to determine the region in which we should look for lane lines in this frame. This improves performance of the model by narrowing search to areas where we should expect to find the lane lines
- “high confidence” is defined by the detected lane lines having similar curvature: no more than a 400 meter curvature radius difference and the detected lane lines are no more than 750 pixels and no less than 500 pixels apart. If a high confidence frame is detected, then save the polynomial fit to the line_tracker class to use for detecting lane lines in the next frame. Line_tracker.py line 162-169
- Lane line pixels are identified as those non-zero values within the region of interest of the binary image perspective transform
- Lane line polynomial fit is generated by use np.polyfit on the non-zero values detected in the search



- Warp the fit from the rectified image back onto the original image and plot the lane lines
  - Calculate the radius of curvature of the left / right lane lines
    - Fit the lane line polynomials to x,y in world space – line_tracker.py line 154-155
    - Calculate the new radii of curvature in meters – line_tracker.py line 157-158
  - Calculate the distance to center of the lane – line_tracker.py line 197-201
  - Use cv2.warpPerspective and the inverse perspective matrix to warp back to original image space
  - Use cv2.addWeighted() to combine that warping with original image to produce the final output

Discussion

There is more to work to be done to improve this model for more challenging road conditions.

1. The trapezoidal region I have hand-selected for the perspective transform may “look out” too far down the road and be an issue on roads with sharper turns.
2. This transform may also be too narrow for sharp turns. The model could be more robust by widening the area to apply a perspective transform
3. My model also tends to be a little jittery and that could be improved by using an average polynomial fit (of the past few frames) instead of fitting the lane line pixels frame by frame. I am using an average lane line polynomial fit to search for likely lane line positions in future frames, but I’m not using that average to draw the current fit of the lane lines. This means that if my image pipeline incorrectly detects lane lines, the output video will draw a knowingly-incorrect lane line fit, but it won’t save that fit to search for lane lines in future frames.