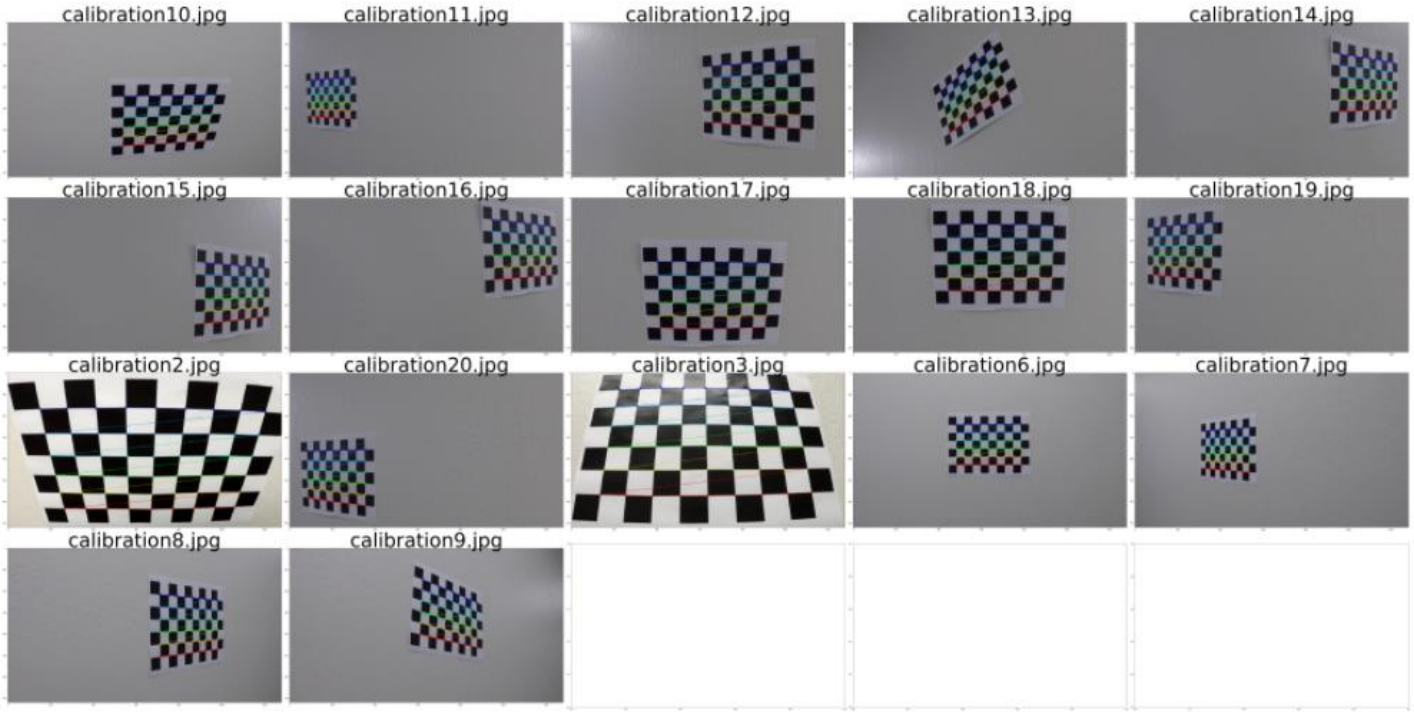
Project 4: Advanced Lane Finding Write Up

Camera Calibration

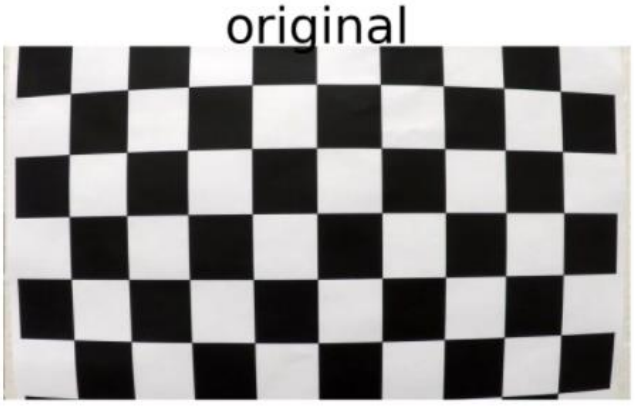
Camera Calibration Steps

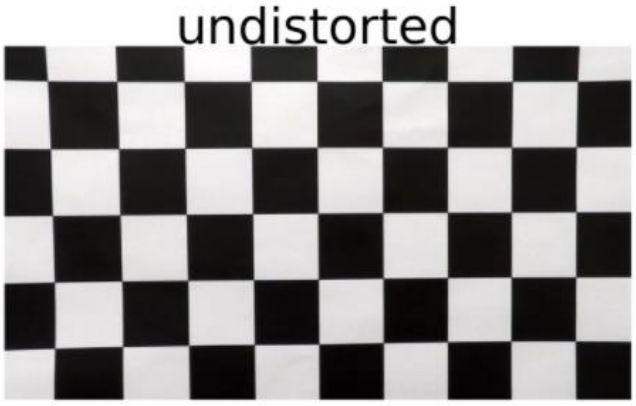
1. Read in calibration chessboard images (at least 20).
2. Map out image points and object points.
3. Do this for all 20 calibration images and append points to the image and object point array.
4. Calibrate camera (using cv2.calibrateCamera(objpoints, imgpoints, imageshape)).
5. Correct distortion.



Distortion was corrected via two steps:

1. Obtaining calibration matrix.
2. Undistorting using cv2.undistort





Binary Thresholding

Steps to Binary Thresholding

Primarily, three thresholding will be used. They are:

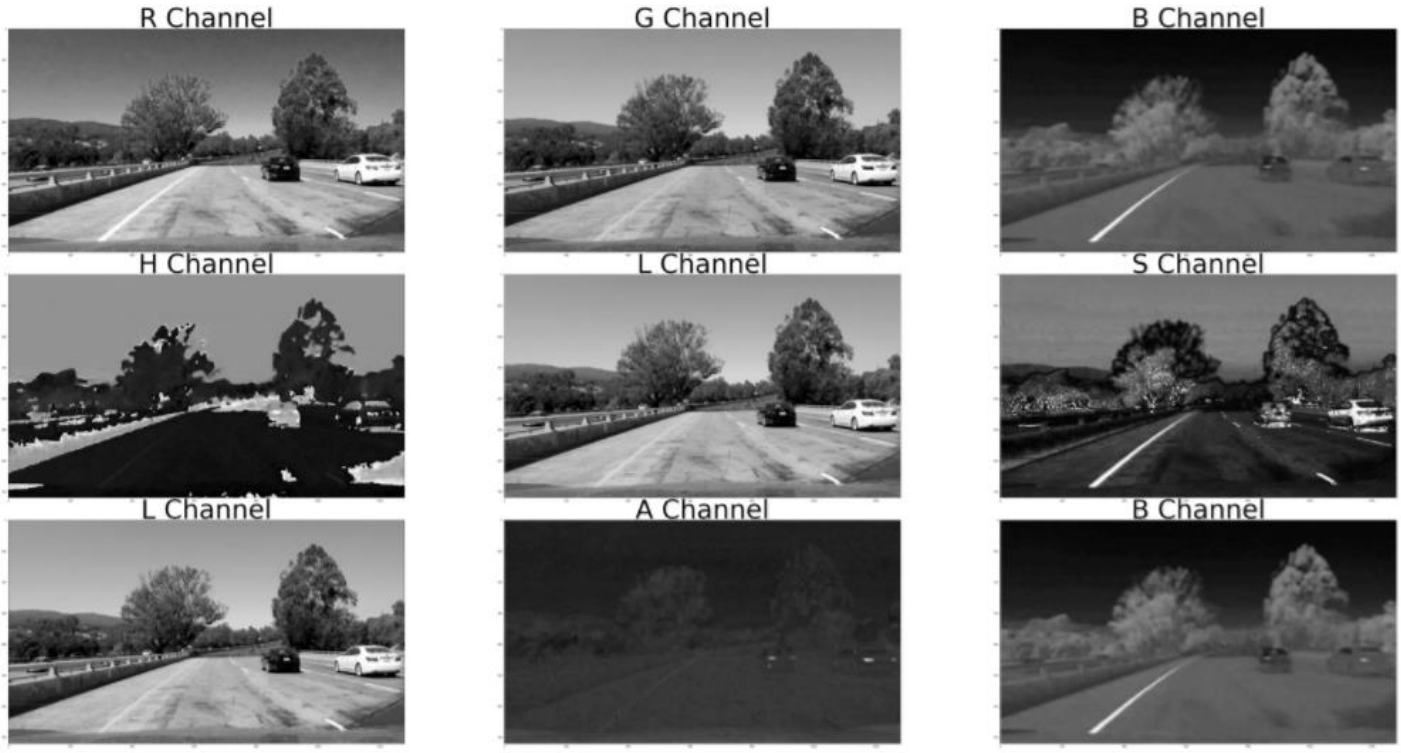
- absolute gradient thresholding (x or y).
- Gradient magnitude thresholding (by combining x and y).
- Gradient direction thresholding (by calculating the direction of the gradient)

If the above proves to be insufficient, we shall also explore color thresholding. The color thresholding we will be looking at are:

- RGB
- HSV
- LAB

Steps to carry these out are as follows:

1. Create functions for the three thresholding method
2. depending on feedback from visualization, add/combine color thresholding accordingly
3. Combine

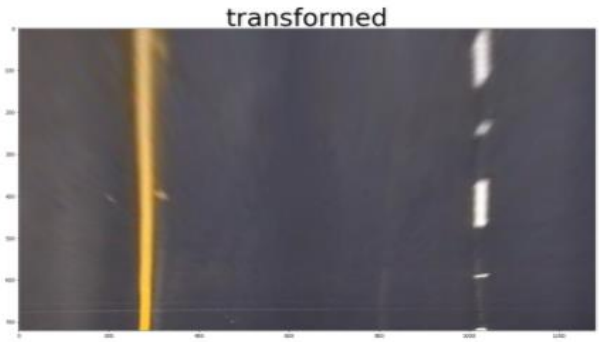


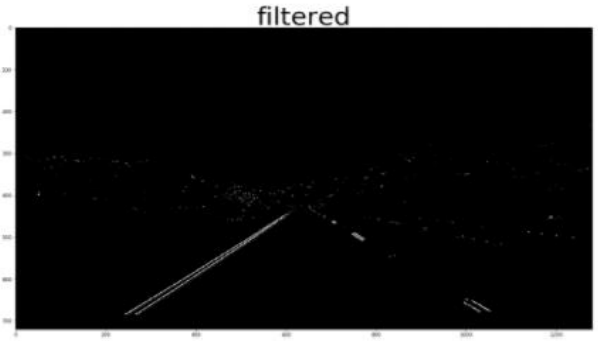
Perspective Transform

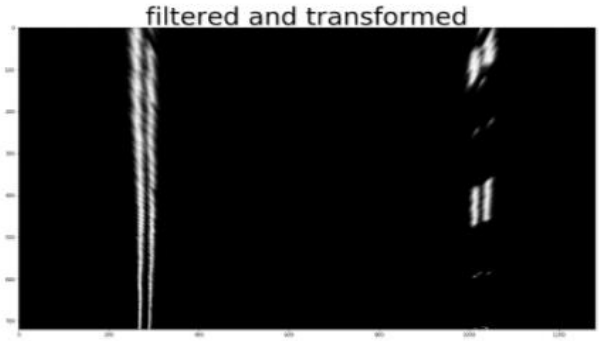
Steps to Performing a Perspective Transform

1. Obtain transformation matrix
2. Warp image using said transformation matrix

A picture with straight lane was used. Points were chosen based on histogram to find the beginning and end points of the left lane line (see project file for detail). Here is the result:





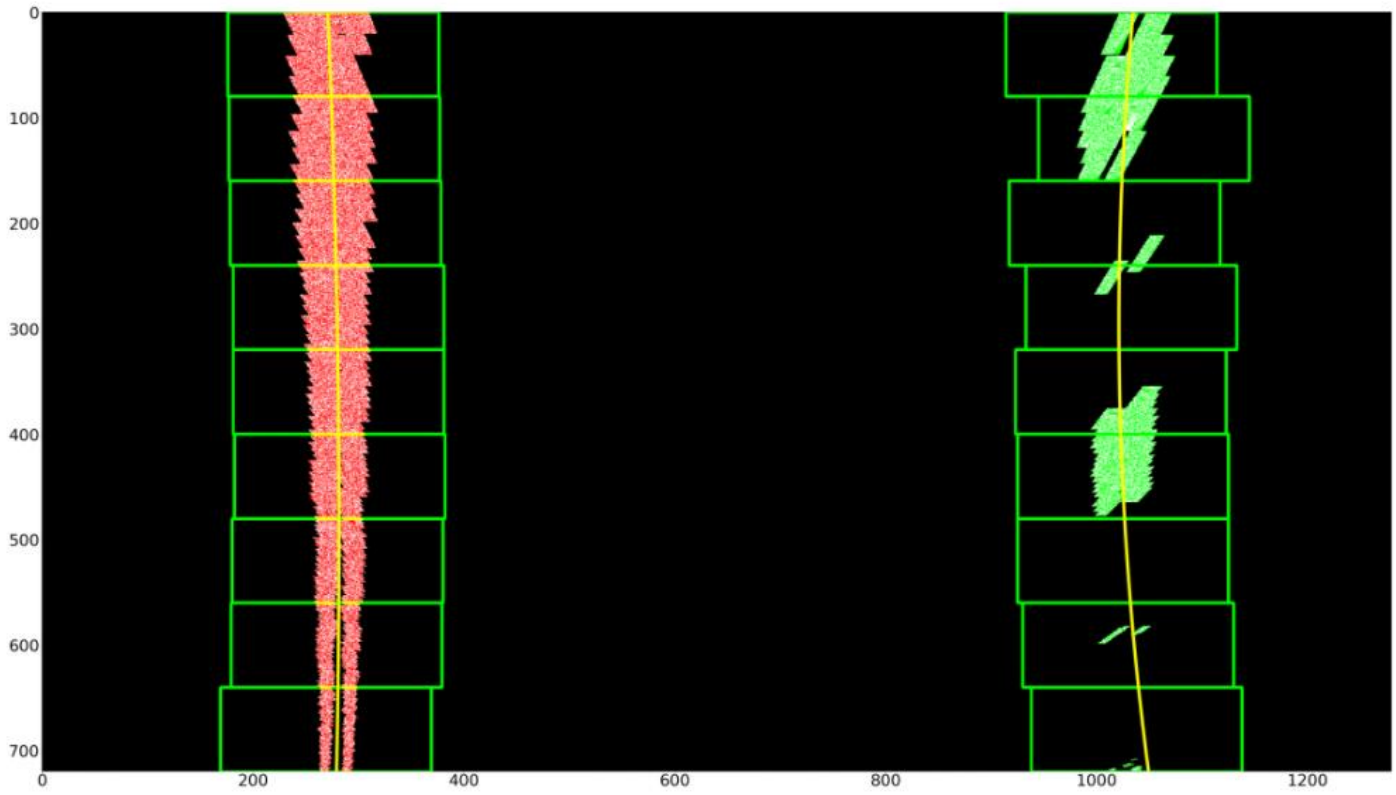


Line Detection and Lane Boundary

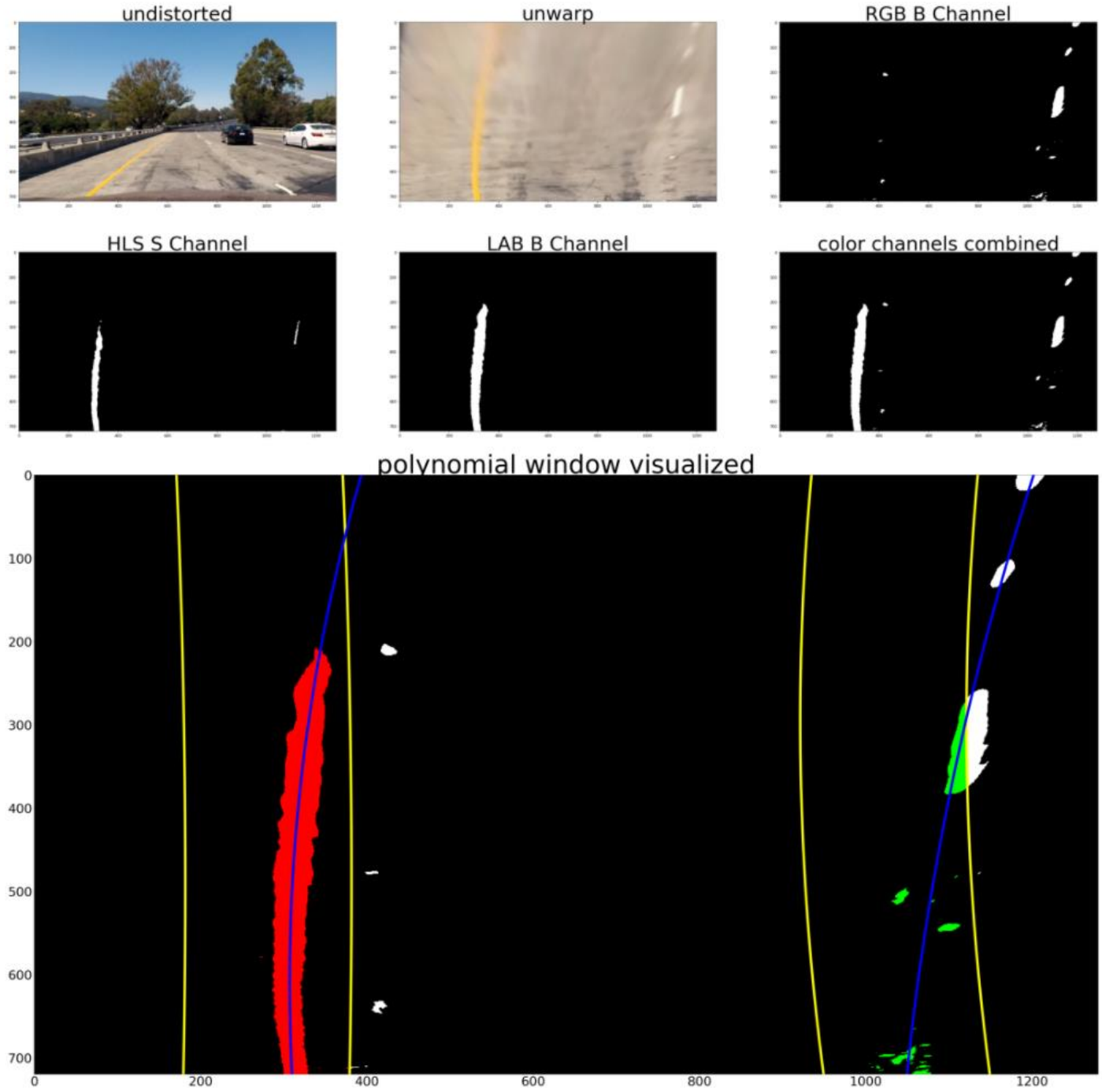
Steps to detecting pixels as part of the lane line:

1. Segregate the warped picture and create a histogram.
2. Use the peaks on the histogram to find the lane.
3. Set hyperparameters for sliding window (number of windows, window width).
4. Find and append/store the coordinates of pixels within each window.
5. Poly fit these points and obtain a polynomial.
6. Using the polynomial to locate pixels in the subsequent frames.
7. Refit new found pixels
8. Repeat until last frame.

Shown below is a lane identified by sliding window method:



Shown below is lane finding through polynomial established through previous frames:



Yellow lines mark the pixel detection window made by the polynomial derived from first frame (which was made by sliding window). The colored pixels (red and green for left and right lane line respectively) denotes the pixels detected by said window. We can see that although a significant portion of the right lane line lies outside of the right window (denoted by the white pixels). However, because these pixels lie on the same line, the polynomial still goes through those points that have been excluded.

Find Line Curvatures

Steps:

1. using information given on 15.35:ym_per_pix = 30/720. xm_per_pix = 3.7/700
2. Multiply it to x and y data to convert pixel data to distance data.
3. Find curvature

In [17]:
```python
def find_curvature_vehicle_pos(binary, left_fitx, right_fitx):
    import numpy as np
    #this is given by on 15.35
    ym = 30/720
    xm = 3.7/700
    #generating independent input values:
    ploty = np.linspace(0, binary.shape[0]-1, binary.shape[0])
    y_eval = np.max(ploty)
    #convert polynomial to real world size
    left_fit_cr = np.polyfit(ploty*ym, left_fitx*xm, 2)
    right_fit_cr = np.polyfit(ploty*ym, right_fitx*xm, 2)
    #calculate the new radii
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym + right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])
    #now to find vehicle position in x
    #assuming camera is mounted in vehicle center
    left_pt = left_fit_cr[0]*(y_eval*ym)**2 + left_fit_cr[1]*(y_eval*ym) + left_fit_cr[2]
    right_pt = right_fit_cr[0]*(y_eval*ym)**2 + right_fit_cr[1]*(y_eval*ym) + right_fit_cr[2]
    lane_mid_point = np.mean([left_pt, right_pt])
    vehicle_pos = (binary.shape[1]//2)*xm - lane_mid_point
    left_curverad = round(left_curverad, 2)
    right_curverad = round(right_curverad, 2)
    vehicle_pos = round(vehicle_pos, 2)
    if vehicle_pos <= 0:
        vehicle_pos = str(np.absolute(vehicle_pos)) + ' m left of center'
    elif vehicle_pos >= 0:
        vehicle_pos = str(np.absolute(vehicle_pos)) + ' m right of center'

    return left_curverad, right_curverad, vehicle_pos
```

Warping Onto Original Image

Steps to warping detected lane lines back onto original image:

1. Using the equation obtained from polyfit, generate x and y points for left and right line to plot
2. Draw line onto warped blank image
3. Warp drawn lines back onto original image

Shown below is an example output done on a test image:

Main Pipeline for Video

Summary for pipeline:

1. Undistort
2. Perspective Transform
3. Thresholding
4. Depending on whether this is the first frame or second frame, do either sliding window or 'line window' to detect line
5. Polyfit
6. Unwarp and overlay information onto original image/frame

Video is stored in the main directory under the repo, named “project video processed2”.

Note that “project video processed” is what is obtained without “sanity check”.

Without “sanity check”:

With “sanity check” and averaging out frames:

It is still not perfect, but it is much better.

Discussion

Issues

- The pipeline has no trouble picking up information from the road. However, it has trouble separating noise (e.g. shadows, road textures) from useful information such as lane lines.
- It would be more useful to average out the curvature for the last n frames, as opposed to updating it every single frame.

Potential troublesome scenarios

- Where lane curves around quickly, escaping the sliding window altogether. It would likely run into the same issue on roads with great elevation changes.
- Roads under conditions where there is a lot of shadows, or high contrast silhouettes that is interfering with the lane lines.

Potential Improvements

- A more robust thresholding to filter out useful info from noise. Perhaps do this via exclusivity logic between different combination of gradient and color spacing.
- A more robust video process pipeline to average out the frames better. This can potentially be done via monitoring more things besides curvature, average x coordinates.