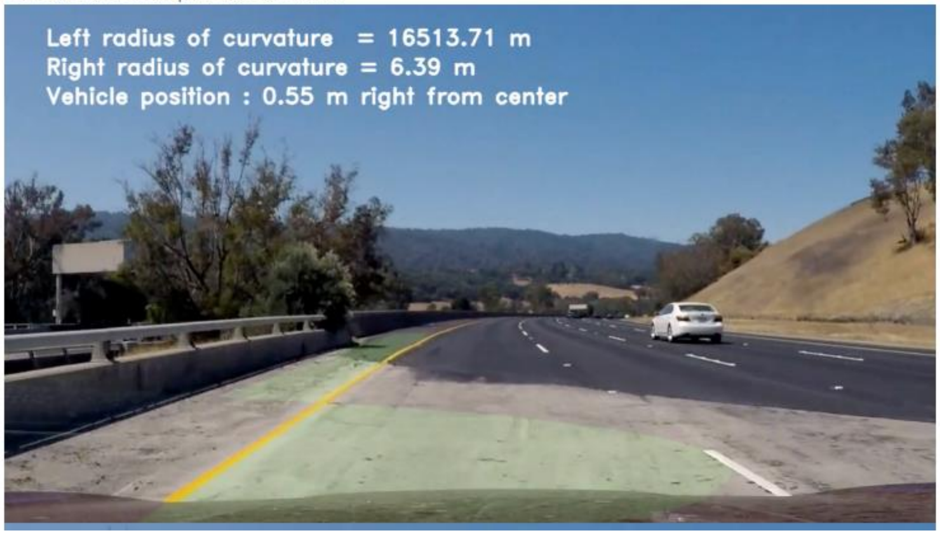
1. Thank you for pointing out some problems in the video. For example,

Here are some examples from the video:



```
Left radius of curvature  = 16513.71 m
Right radius of curvature = 6.39 m
Vehicle position : 0.55 m right from center
```

After checking the code again, I adjusted a few threshold values in HSV transform, and also fixed a bug in “add_polygon” function between left line and right line. This case has disappeared in the new “project_out_new.mp4”.

2. More thoughts about the scenarios that the pipeline will struggle/fail.

The pipeline now is not robust. Some cases that may cause failure include:

1) road condition: work zone/accident

2) light/weather condition: day/night, clear/rain/snow

3) positions of the vehicle, middle lane/left lane

4) traffic conditions: no vehicle ahead/congestion

The pipeline needs to be improved by enriching the training dataset. As many as possible scenarios have to be considered.

Advanced Lane Finding Project

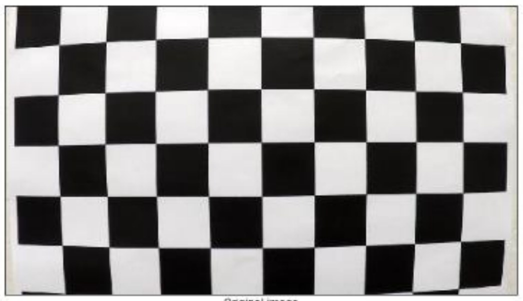
The goals / steps of this project are the following:

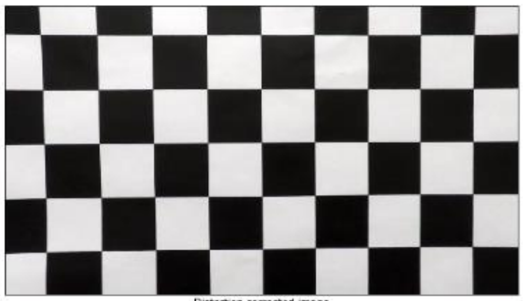
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.

- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

1. Camera Calibration

The calibration of camera is realized through an image of chessboard corners. In this project, the chessboard has nx = 9 and ny = 6 corners. The functions in openCV "cv2.findChessboardCorners" and "cv2.calibrateCamera" are mainly used. For example, the "calibration1.jpg" images before and after the distortion correction are as follows:


Original image


Distortion corrected image

2. Apply a distortion correction to raw images.

Another example of applying the distortion correction coefficients learned in camera calibration on testing image is as follows:

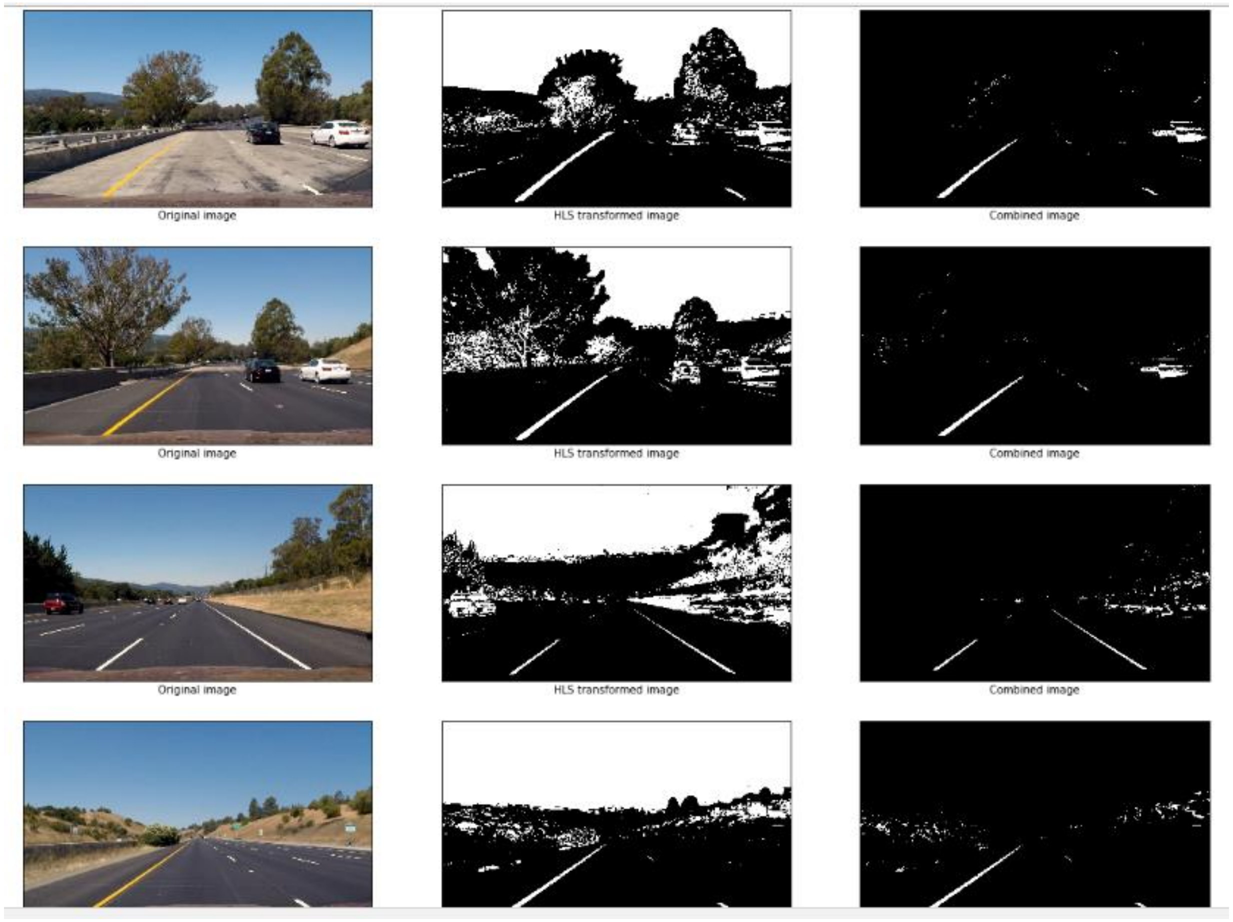
Original image

Distortion corrected image

3. Use color transforms, gradients, etc., to create a thresholded binary image.

In this project, I applied “HLS transform”, “HSV transform”, “gradient transform”, and “sobel transform”. Different threshold values were tested. Finally, the HLS and HSV transforms were combined. For “HLS transform”, only S channel was kept; and for “HSV transform”, thresholds for yellow and white channels were applied.

Several thresholded binary images after processing are as follows:



4. Apply a perspective transform to rectify binary image ("birds-eye view").

The main point here is to decide four locations for both the source image and destination image. After a few tests, the four points were selected as follows:

For source image (“straight_lines1.jpg”):

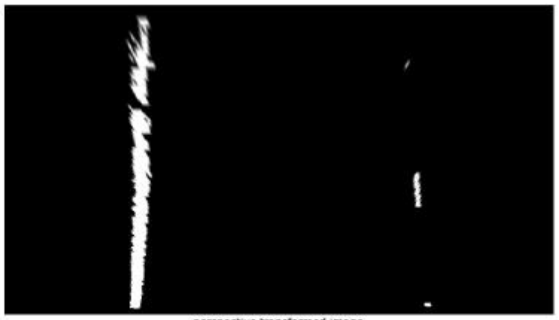
```
src = np.float32(
    [(200, 720),   # bot left
     (585, 455),   # top left
     (705, 455),   # top right
     (1120, 720)   # bot right
    ])
```

For destination image,

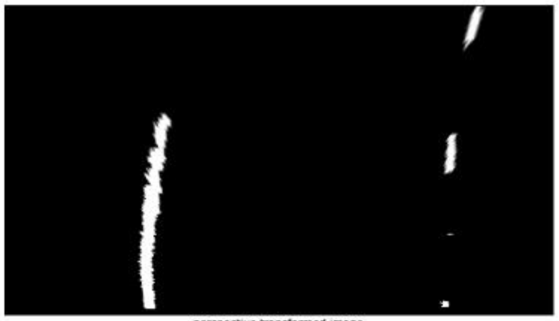
```
des = np.float32(
    [(300, 720), # bot left
    (300, 0), # top left
    (1000, 0), # top right
    (1000, 720), # bot right
    ])
```

The results of applying the perspective transform are shown here:

Original image


perspective transformed image

Original image


perspective transformed image

5. Detect lane pixels and fit to find the lane boundary.

Similar process is used as given in the lectures.

First, the base positions of the left and right lanes were identified through the second-half histogram diagram.

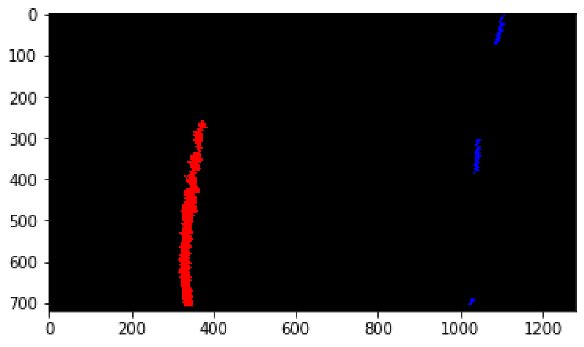
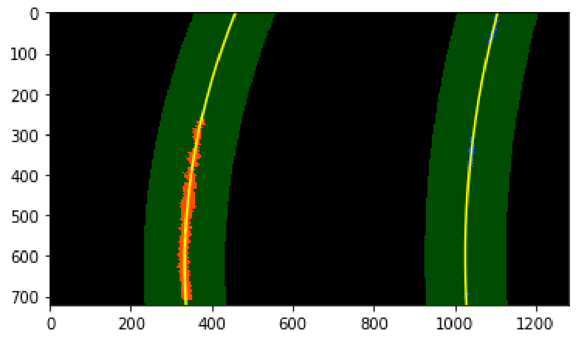
Second, the image is split into 9 small windows. From bottom to the top, for each window, based on the current base position and user-defined margin size, the "good" nonzero left and right pixels are identified.

Third, if the total nonzero pixels are greater than a threshold, recalculate the current base positions for the left and right lanes.

Fourth, fit a polynomial line for each of left and right lanes.

Fifth, draw the polynomial lines on the image.

One example is shown as following:

6. Determine the curvature of the lane and vehicle position with respect to center.

Two functions "cal_conversions" and "cal_dist_to_center" are built to calculate the curvature of the lane and vehicle position. The calculation equations from the lecture is applied.

$$f(y) = Ay^2 + By + C$$

and

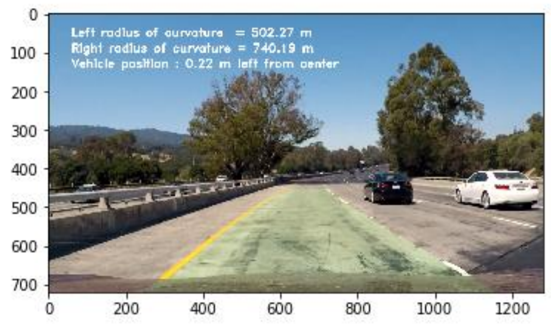
$$R_{curve} = \frac{(1+(2Ay+B)^2)^{3/2}}{|2A|}$$

The offset to the center is the difference between the midpoint of the left and right lanes to the center of the image.

7. Warp the detected lane boundaries back onto the original image.

The binary image was transformed back and added to the original image using "cv2.warpPerspective" and "cv2.addWeighted"

One example is shown as following:


Left radius of curvature = 502.27 m
Right radius of curvature = 740.19 m
Vehicle position : 0.22 m left from center

8. Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

The relevant information is printed on the image as shown above.

9. Discussion

a) The threshold values need to be carefully tuned. They are very important to decide whether that step works or not.

b) The order of the four points in perspective transform step need to be consistent for source and destination images.

10. Video Processing

The same pipeline is applied to the video data. The output is saved in "project.mp4"