Find Advanced Lane Lines using OpenCV

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.

Camera Calibration

. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

First, I'll compute the camera calibration using chessboard images in camera_cal folder.

In order to compute the camera matrix and distortion coefficients, CameraCalibration uses OpenCV's cv2.findChessboardCorners function on grayscale version of each input image in turn, to generate a set of image points. I then use numpy.mgrid in order to create a matrix of (x, y, z) object points. Once I have the image points and the matching object points for the entire calibration dataset, I then use cv2.calibrateCamera to create the camera matrix and distortion coefficients. Then cv2.undistort, returns an undistorted version of the user's input image. Code is implemented in camera_Calibraton and undistort functions.

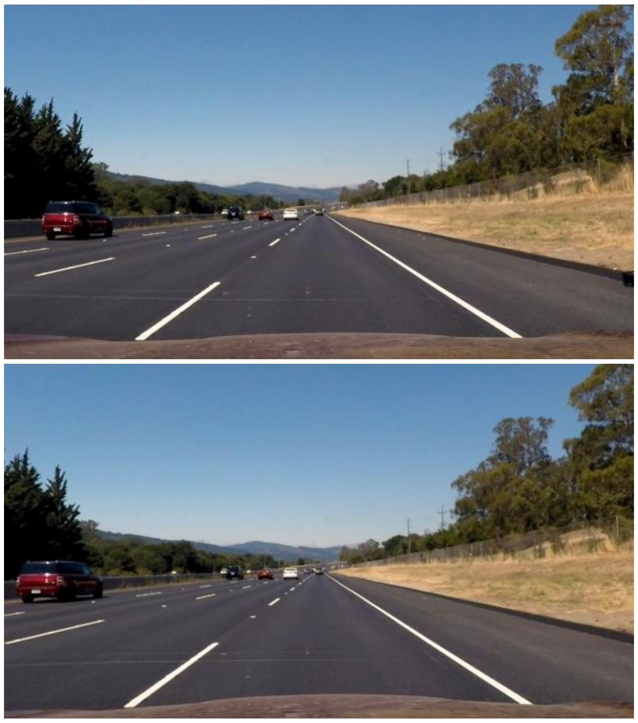
Pipeline (single images)

Upper image is original and below is undistorted image.

First I calibrated camera and then called below method for one image.

```
def undistort(image, mtx, dist):
    image = cv2.undistort(image, mtx, dist, None, mtx)
    return image
```

Provide an example of a distortion-corrected image.



Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used a combination of color and gradient thresholds to generate a binary image

Created following separate methods for each gradient, threshold, magnitude and direction

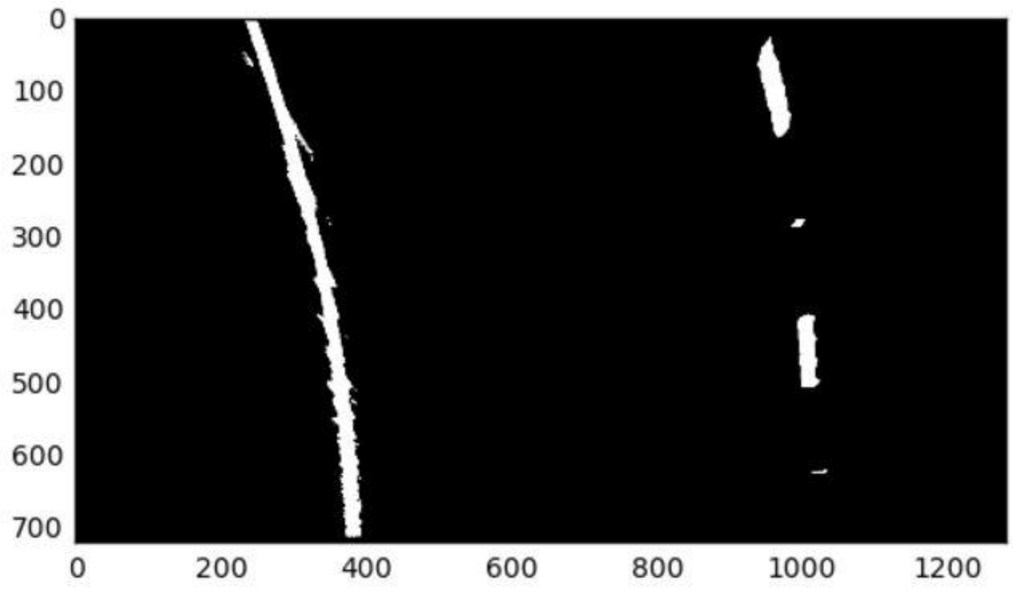
```
abs_sobel_thresh
mag_thresh
dir_threshold
```

Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

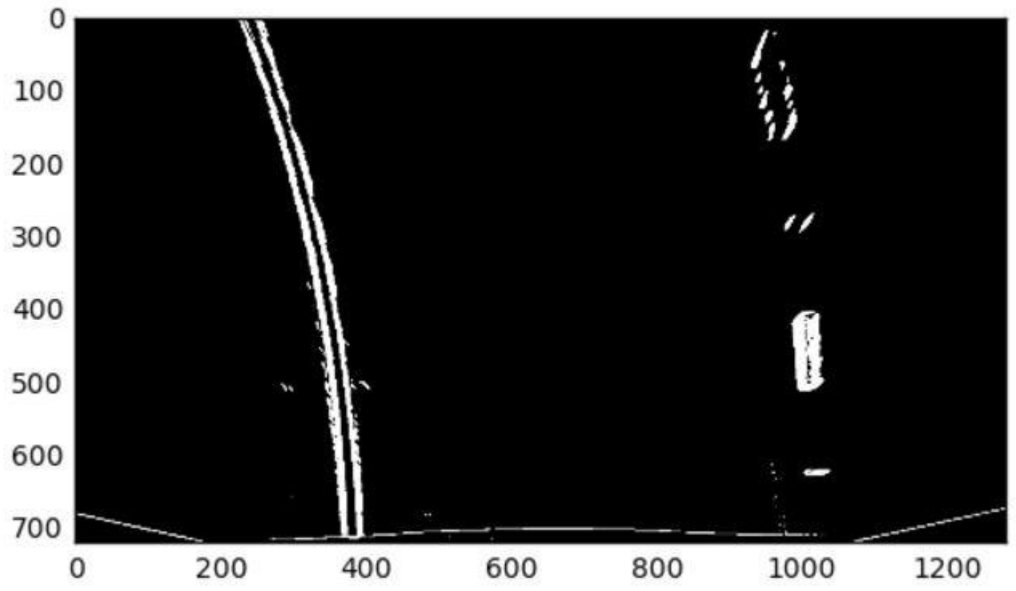
Perspective transformation gives us bird's eye view of the road, this makes further processing easier as any irrelevant information about background is removed from the warped image. Code for this can be found in transform_perspective function. Here I am defining source and destination points. So here I am using cv2.getPerspectiveTransform and cv2.warpPerspective functions from OpenCV.

Binary Images:

After we obtain the perspective transform, we apply colour masks to identify yellow and white pixels in the image. Colour masks are applied after converting the image from RGB to HSV space. We identify yellow colour as the pixels whose HSV-transformed intensities are between ([0, 100, 100]) and ([50, 255, 255]), and white colour as the pixels with intensities between ([20, 0, 180]) and ([255, 80, 255]).



So above image is generated when we apply color mask and the code can be found in color_mask function. In addition to the color masks, we apply sobel filters to detect edges. We apply sobel filters on L and S channels of image, as these were found to be robust to color and lighting variations. After trial and error, we decided to use the magnitude of gradient along x- and y- directions with thresholds of 50 to 200 to identify the edges. Code for this can be found in apply_sobel_filter function.



Then I combine both the binaries and get this as below

Also I have used some Outlier removal methods like if the change in coefficient is above 0.0010, the lanes are discarded, If any lane was found with less than 5 pixels, we use the previous line fit coefficients as the coefficients for the current one. All this was done by trial and error. Code can be seen in pipeline function.

Lane Curvature:

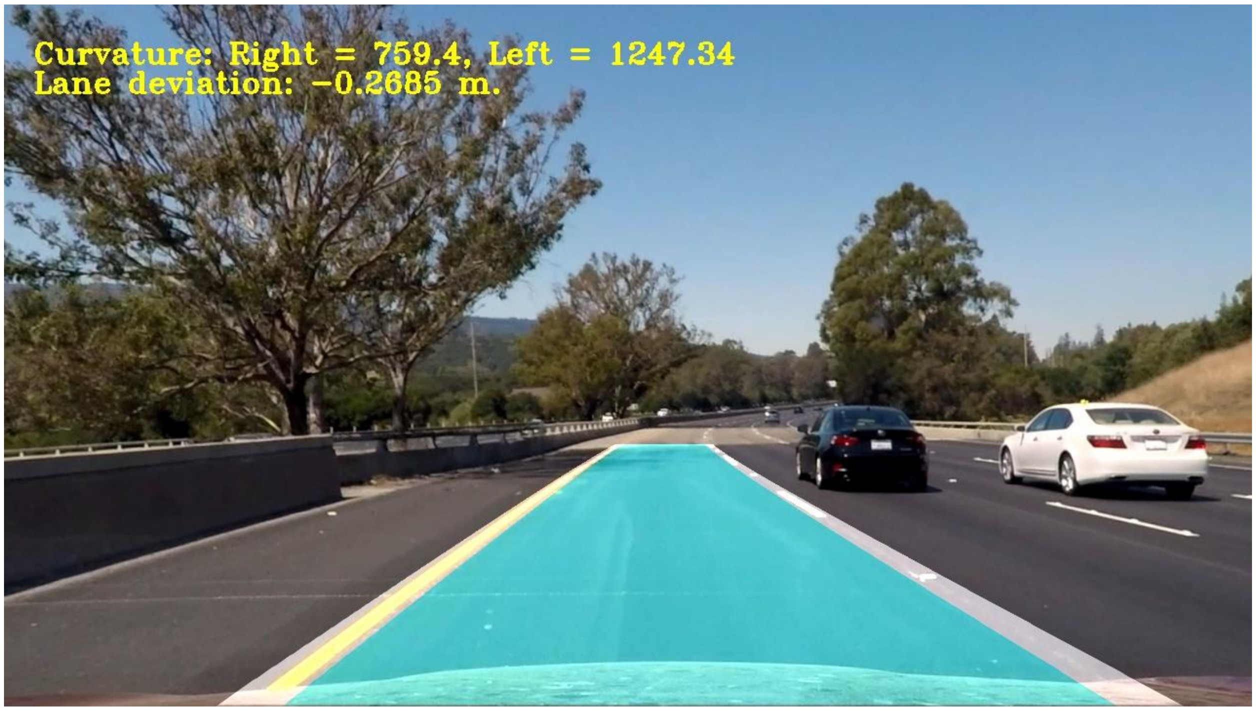
In order to calculate the lane curvature radius, I scaled the x and y coordinates of my lane pixels and then fit a new polynomial to the real-world-sized unit. Using this polynomial, I could then use the radius of curvature formula below to calculate the curve radius in metres at the base of the image: Code for this can be found in get_curvature function.

$$R = \left| \frac{\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{3/2}}{\frac{d^2 y}{dx^2}} \right|$$

Offset from lane center:

To calculate the position of the car, I used the polyfit function to find the bottom coordinates of right and left lanes. Assuming the lane has width of 3.7 metres I calculated distance between centre of image and centre of lane. This is our offset.

Final Result:


Curvature: Right = 759.4, Left = 1247.34
Lane deviation: -0.2685 m.

All the processed images are saved in output_images folder. Project Video output project_output.mp4 is also in output_images folder.

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

The pipeline works well for project output and fails for challenge videos. A lot of manual tuning is done in this project due to lack of time. Also in order to make lane detection more good we can normalize the data.