Advanced Lane Finding Project

***By: Subhendu Mishra** (Subhendu.mishra20@gmail.com)*

The goals / steps of this project are the following:

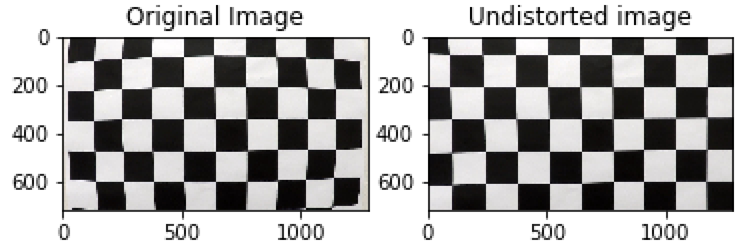
- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- 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

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

The code for this step is contained in the 2nd code cell of the IPython notebook located in "./Advanced_Lane_Lines.ipynb".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 use all the calibration images to create the `objpoints and imgpoints.`

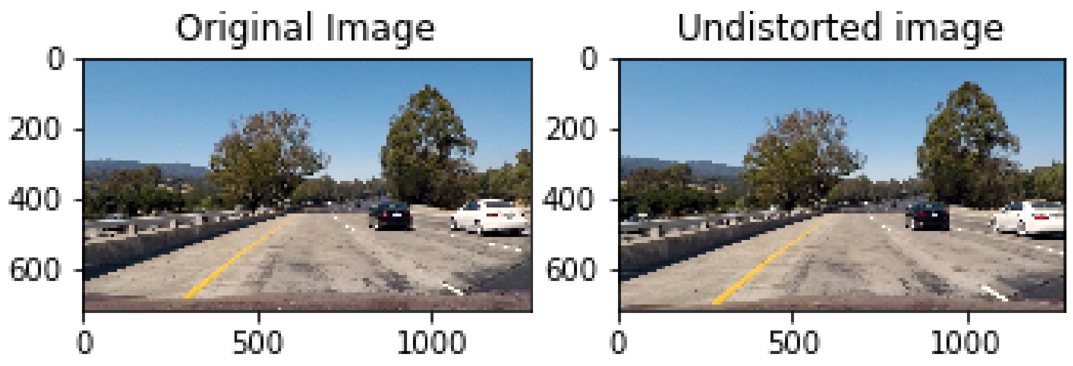
I then used the output `objpoints` and `imgpoints` to compute the camera calibration and distortion coefficients using the `cv2.calibrateCamera()` function. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:



Pipeline (single images)

1. Provide an example of a distortion-corrected image.

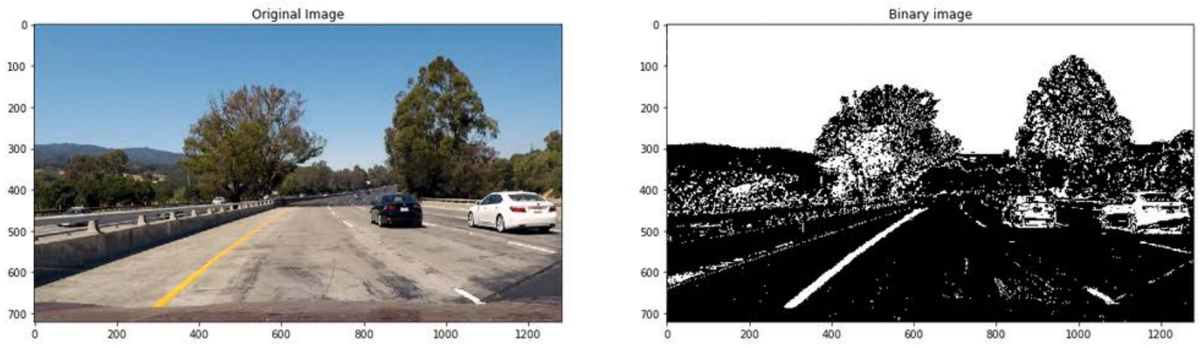
To demonstrate this step, I will describe how I apply the distortion correction to one of the test images (test1.jpg) :



I am using the cal_undistort() function in the code cell 3 to undistort each image using the function cv2.undistort() with the cv2.calibrateCamera() function which uses the `objpoints` and `imgpoints` as computed in code cell 2.

2. Describe how 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 (thresholding steps in code cells 6 and 7). Here's an example of my output for this step (code cell 8). Here I have used sobel() functions for calculating the gradient along the x and y direction. I have used threshold of binary images between 17 and 80 are identified with the lane lines. These are hard coded values and further improvements need to be made. The color space is used to distinguish the lane lines especially the yellow and white lines. For this the 'S' or the saturation parameter is used in the RGB2HLS transform. This helps us to identify the lane lines clearly even with varying lighting conditions.



3. Describe how you performed a perspective transform and provide an example of a transformed image.

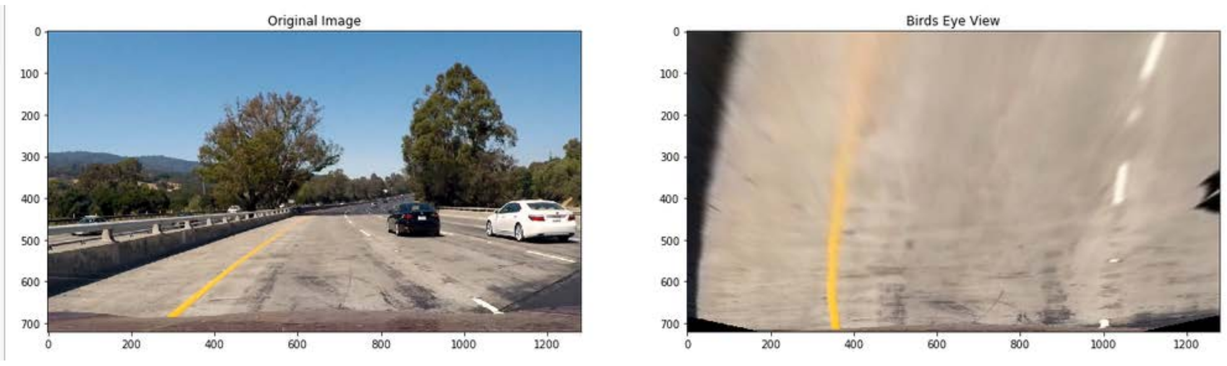
The code for my perspective transform includes a function called `corners_unwarp()`, which appears in the code cell 9 of the Jupyter notebook. The `corners_unwarp()` function takes as inputs an image (`img`) and does perspective transform using the `src and dst` points as described in the function. I chose to hardcode the source and destination points in the following manner:

```
src = np.float32([[200./1280*w,720./720*h],

                  [453./1280*w,547./720*h],

                  [835./1280*w,547./720*h],

                  [1100./1280*w,720./720*h]])

dst = np.float32([[(w-x)/2.,h],

                  [(w-x)/2.,0.82*h],

                  [(w+x)/2.,0.82*h],

                  [(w+x)/2.,h]])
```

This resulted in the following source and destination points:

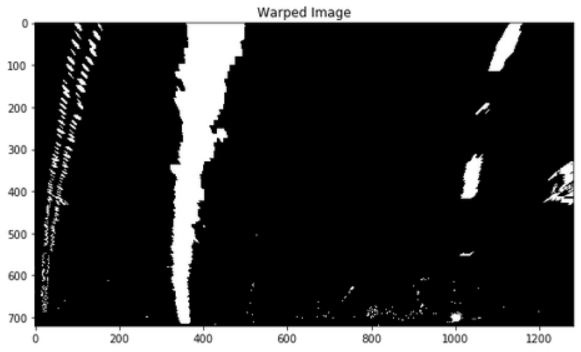
Source	Destination
200, 720	320, 720
453, 547	320, 590
835, 547	960, 590
1100, 720	960, 720

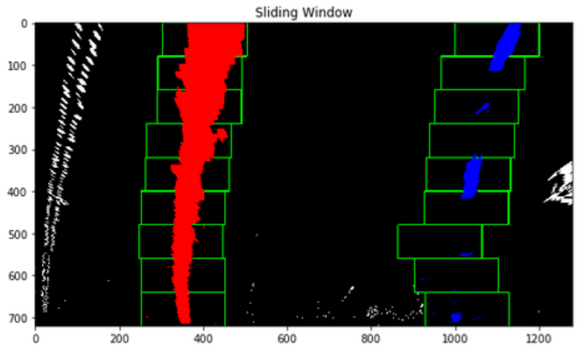
I verified that my perspective transform was working as expected by drawing the `src` and `dst` points onto a test image and its warped counterpart to verify that the lines appear parallel in the warped image.



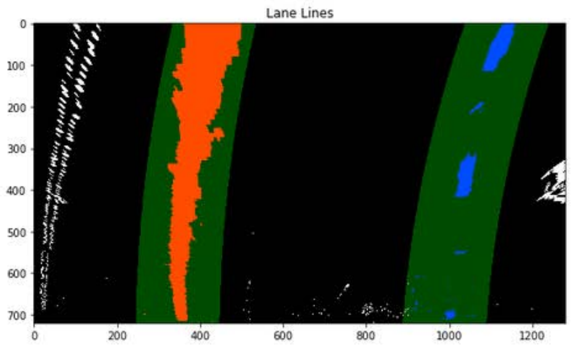
4. Describe how you identified lane-line pixels and fit their positions with a polynomial?

First I used the perspective transform to find the 'Birds Eye' view of the road ahead. Then I used the threshold functions to get the binary warped image as shown. Then I used the histogram to plot where in the image the lane lines are. Then I used the sliding window technique to identify the lane lines as seen in code cell 12.





Once the lane lines are identified, the function `sliding_window()` returns the left and right line polynomials. This is taken as input in the `lane_lines()`function in code cell 14. The sliding window and the lane lines detection are separated into two different functions so that once the lanes are scanned in the first frame, the lane lines for the following frames can be found near the already detected lanes.



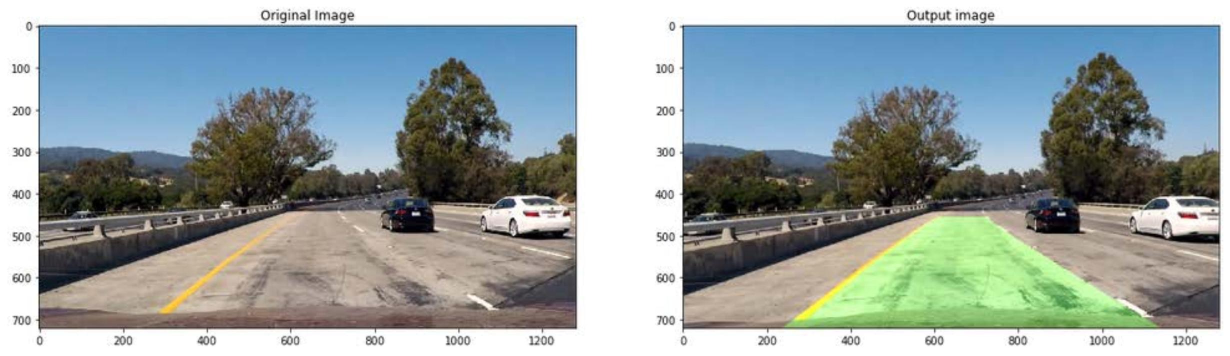
5. Describe how you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I did this in code cell 16 of the Jupyter notebook. The function `curvature(image, ploty, leftx, rightx )` takes in the image and the array of the fitted left and right lane lines. Then it uses standard US road lane dimensions to find the radius, position and direction of the vehicle. Here it is taken that the lane is 30 m long and 3.7 m wide. The center of the image is subtracted from the center of the detected lane area to find whether the car is left or right of the center. For the test1 image the radius, position and direction is as given below:

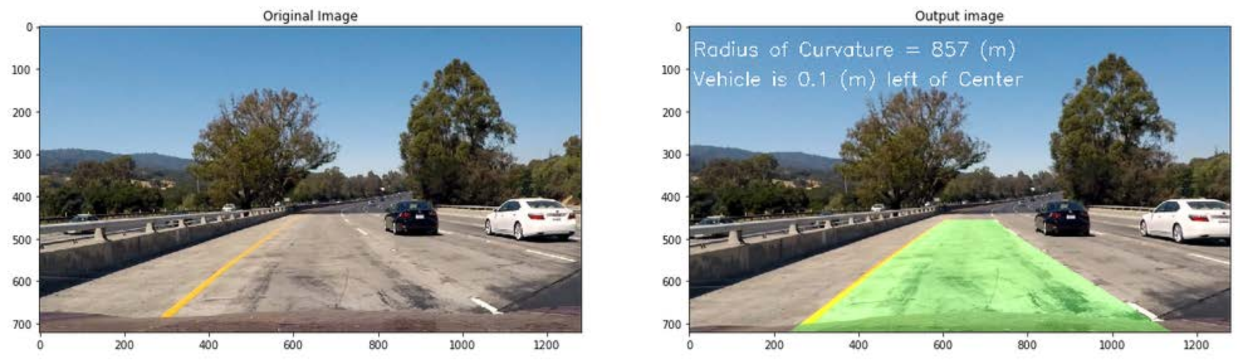
Radius: 857 (m)	Position: 0.1 (m) left of the center

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

I implemented this step in code cell 18 of the jupyter notebook `Advanced_Lane_Lines.ipynb`. Here the `corners_warp()` function (code cell #10) is used to warp the area of the lane back to the original image. Here is an example of my result on a test image:



Finally in the `pipeline()` function (code cell # 20) I write the radius, position and direction of the vehicle back onto the image.



Radius of Curvature = 857 (m)
Vehicle is 0.1 (m) left of Center

Pipeline (video)

1. Provide a link to your final video output.

Here's a [link to my video result]

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?

I think the output produced is reasonable. Although the conditions were quite perfect in the project video. First I take the video and undistort the images. For this the camera is calibrated beforehand. Then perspective transform is used on each undistorted image to get a birds-eye view of the road ahead. This image is processed to get the binary image using gradient analysis using the HLS color space and sobel operator. Here we have assumed to have threshold values specific to our requirements. It is not recommended to have a hardcoded range in which we want to identify the lane lines. This is necessary because sometimes the climate can change drastically or lighting conditions can become very poor. This further needs consideration, however the limits taken in the project work well with the project video.

This gives us binary image of the lane lines. We then plot the right and left lane lines using sliding window technique to identify two polynomial lines. Here again check is needed to make sure the lane lines are parallel and the radius of curvature is within limits. If they are not, then we need to re-evaluate the binary image. Once the lane curves are identified the radius of curvature and position of the vehicle with respect to the center can be found. Here I have used the minimum of the two radii as the curvature radius. This may be where the pipeline fails. Again the sanity check is needed to verify the results. The position of the vehicle is determined according to the standard lane dimensions. Here again the pipeline can fail, if the vehicle is travelling on a narrower road or unmarked lane lines. The position states whether the car should go to the left or the right and by how much is the vehicle off the center. The algorithm used here works but can be further improved.