Readme

Project 4 - Advanced Lane Finding Project Denise R. James

Third Submission

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 threshold 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. The camera used to create the images and videos on the road were calibrated to chessboards with 9 columns and 6 rows. The camera calibration matrix and distortion coefficients were set to chessboards of 9 columns and 6 rows for this project. Twenty chessboard images were used to test the calibration of the camera used for the road images.

The code for this step is contained in the second code cell of the IPython notebook located at https://github.com/DeniseJames/P4/blob/master/camera_calibration.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 then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. "[1]

Of the twenty images tested with the new calibration three failed as expected. The images were not representative of having 9 rows and 6 columns. These failures are acceptable.

Below are the results of the camera coefficients:

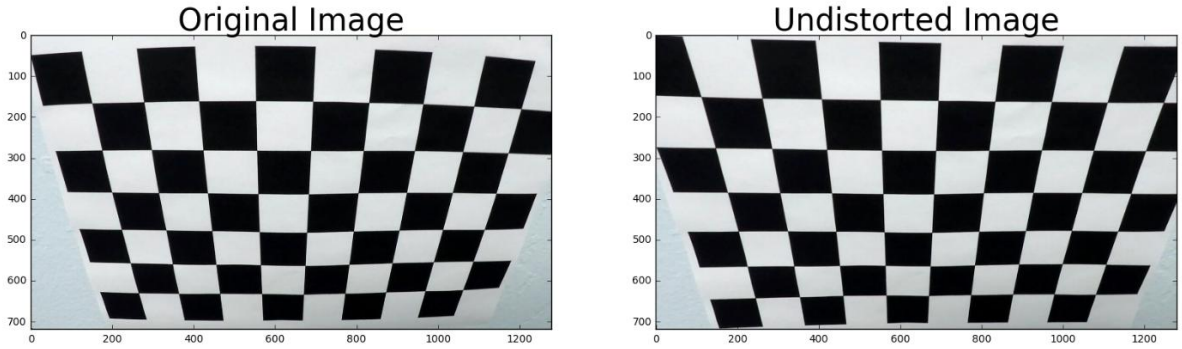
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration10.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration13.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration2.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration9.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration8.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration17.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration6.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration3.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration18.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration12.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration7.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration15.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration11.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration19.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration20.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration14.jpg
Picture passed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration16.jpg

Picture failed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration1.jpg
Picture failed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration5.jpg
Picture failed calibration: CarND-Advanced-Lane-Lines-master/camera_cal/calibration4.jpg

Apply a distortion correction to raw images

In sequential notebook, https://github.com/DeniseJames/P4/blob/master/P4-20170207.ipynb, I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result on correcting an image. The rest of the code for this project are included in the above url.

.



Below the corners_unwarp function uses the camera coefficients to return the actual image.

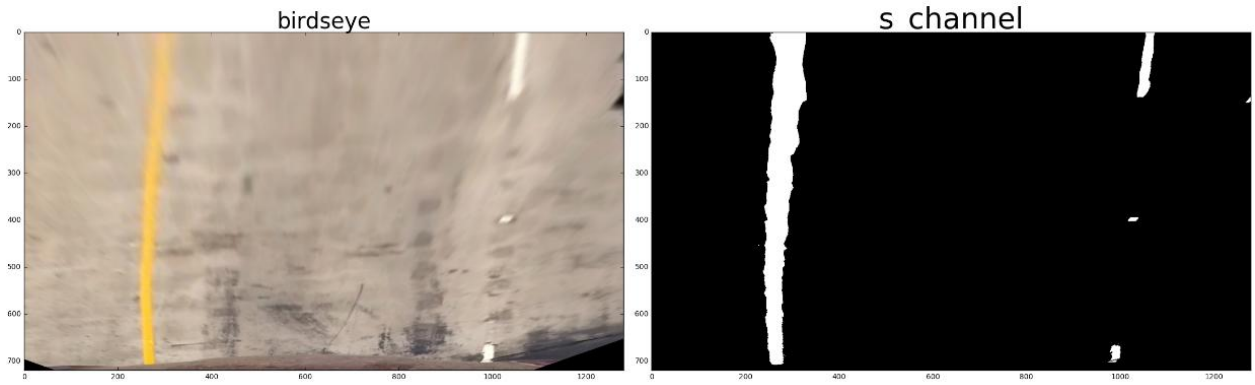
```
def corners_unwarp(img, mtx, dist):
    # Pass in the image into this fumction
    #undistort using mtx and dist
    undist = cv2.undistort(img, mtx, dist, None, mtx)
    return undist

undist = corners_unwarp(image, mtx, dist)
```

To demonstrate the results of the above code executed image and undist are shown.

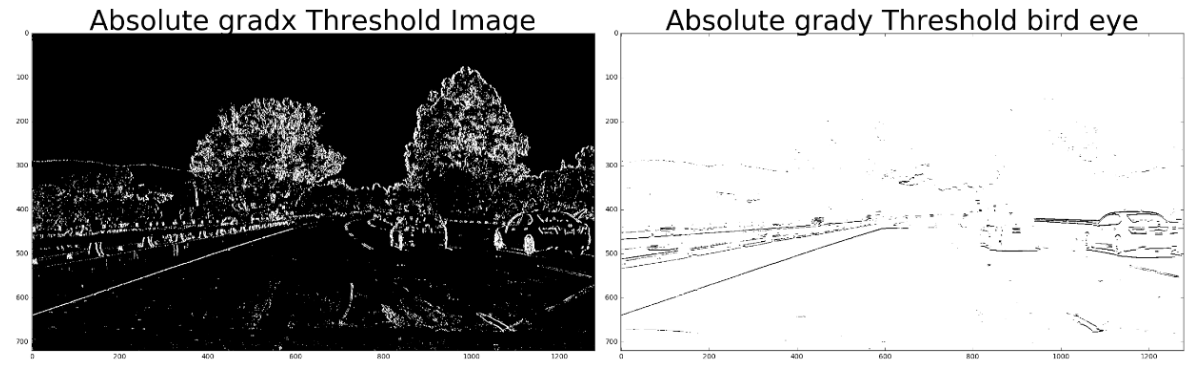
2.

A method or combination of methods (i.e., color transforms, gradients) has been used to create a binary image containing likely lane pixels. There is no "ground truth" here, just visual verification that the pixels identified as part of the lane lines are, in fact, part of the lines. Example binary images should be included in the writeup (or saved to a folder) and submitted with the project.

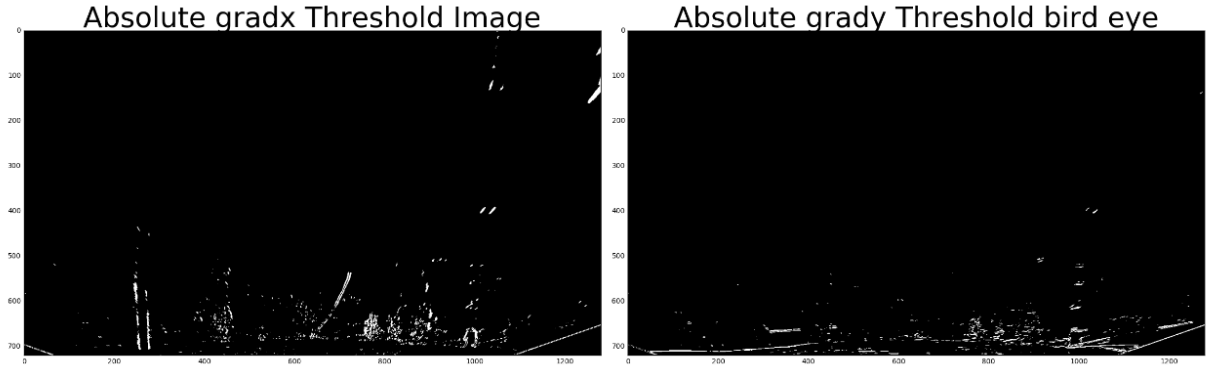
I used color transformation, S-channel of the HLS format. Gradients did not help the mask so they were not used in detecting the lanes. Below is a bird's eye transformation next to a binary HLS, s-channel.

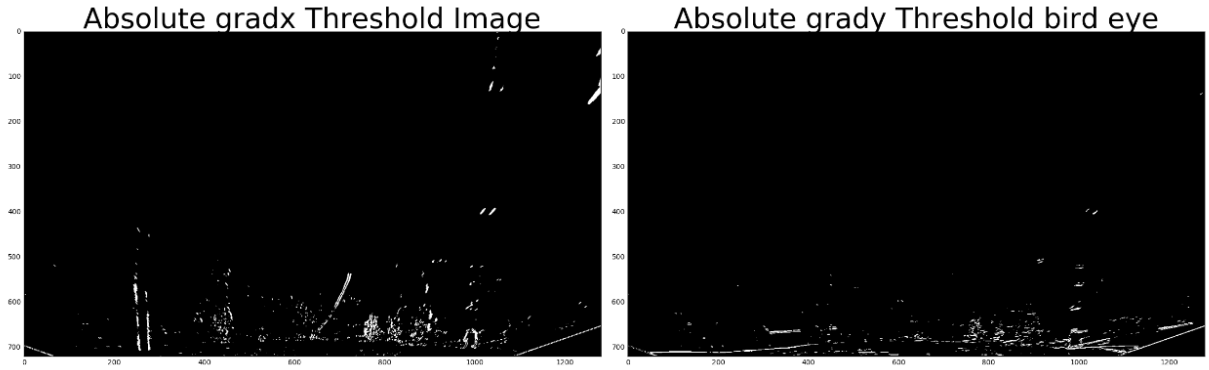


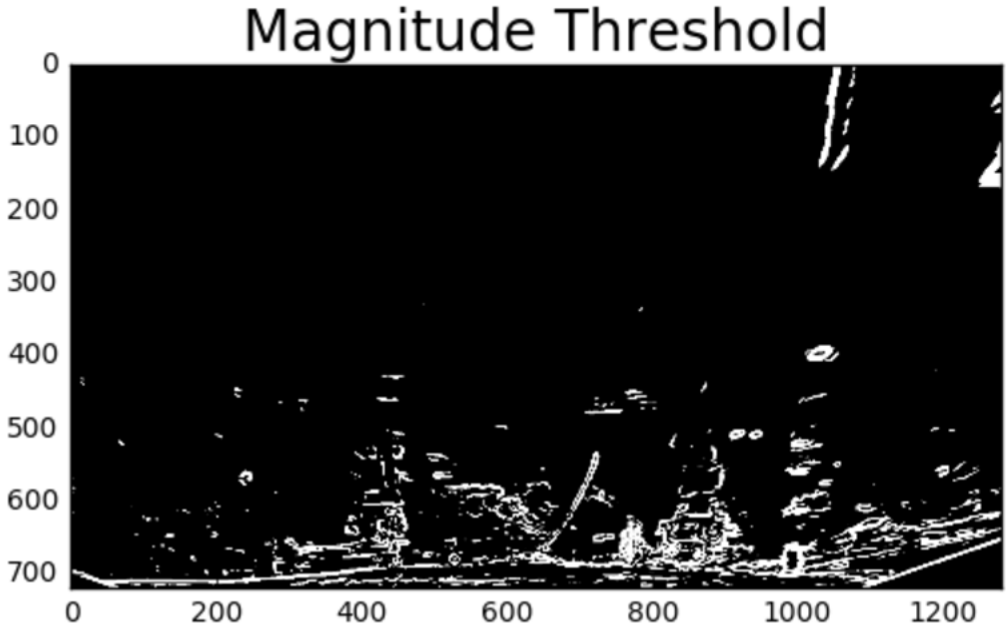
Here are some the gradients I found not helpful to the mask.

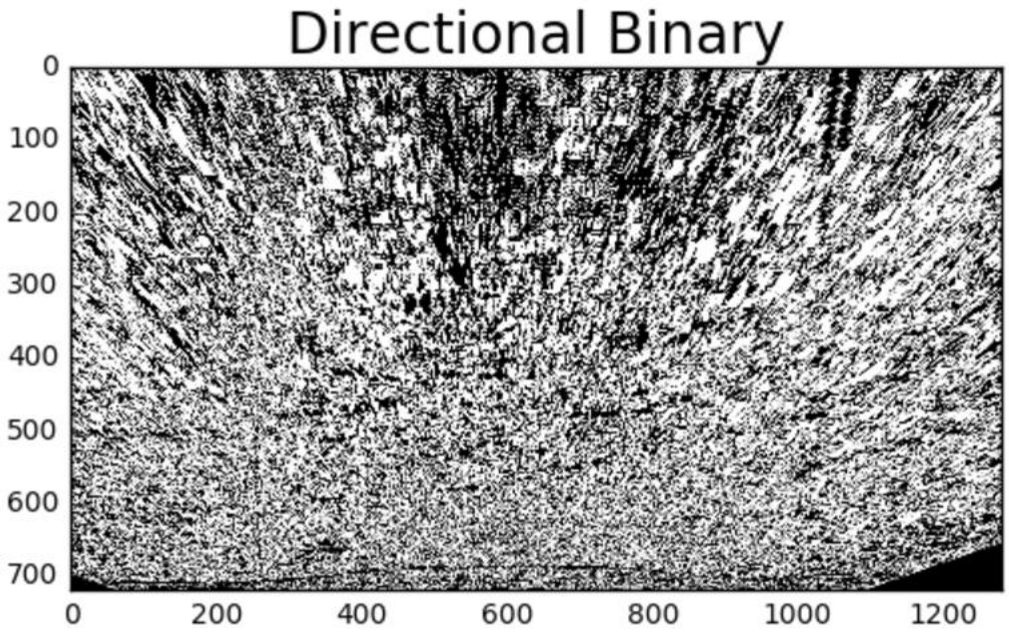


Following are bird's eye binary images.


Absolute gradx Threshold Image
Absolute grady Threshold bird eye


Absolute gradx Threshold Image
Absolute grady Threshold bird eye

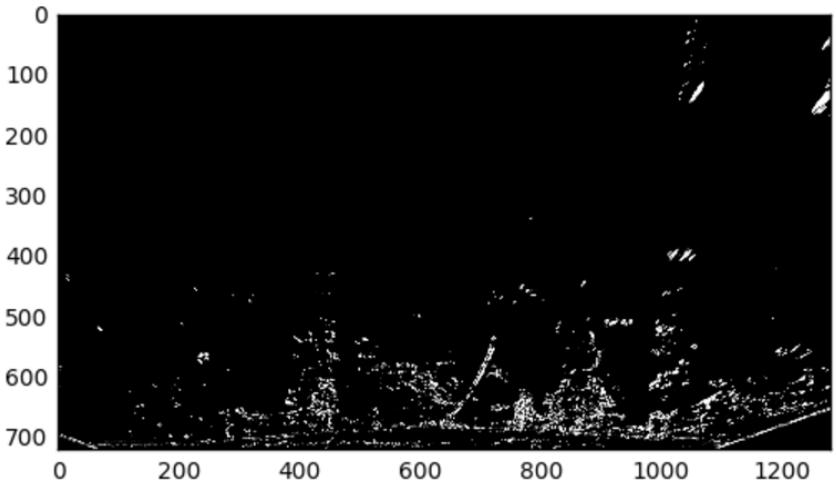

Magnitude Threshold
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



The combination of the above images as demonstrated in the lecture notes is:

```
In [18]: combined = np.zeros_like(dir_binary)
         combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))] = 1
         plt.imshow(combined, cmap='gray')
```

```
Out[18]: <matplotlib.image.AxesImage at 0x7f2e9b979978>
```



The combination image would harm the s-channel mask. I decided to use the s-channel alone.

I was not successful in tuning the gradient parameters.

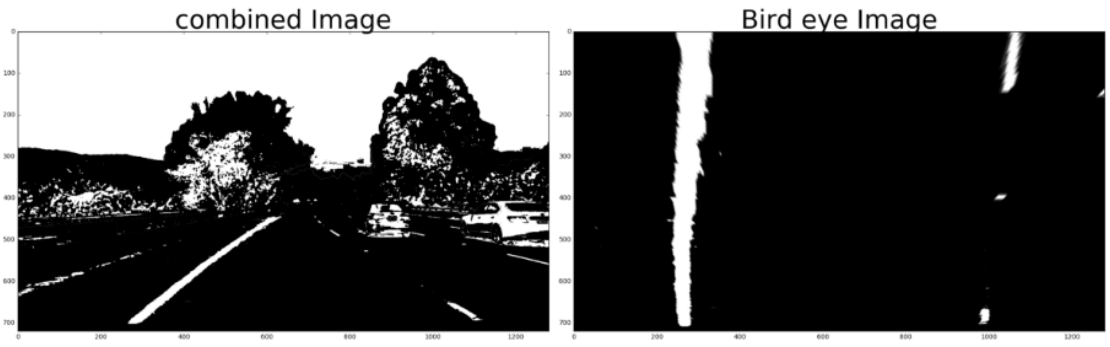
UPDATE DUE TO REJECTED PROJECT IN THIS ITEM.

I learned I did all of my gradient thresholds on the warped image. In this reiteration I performed gradient and color thresholds on the image before transformation. With help from John Chen's open source P4 notebook, this is the new binary mask statements:

```
gradx = abs_sobel_thresh(undist, orient='x', thresh=(25, 100))
    grady = abs_sobel_thresh(undist, orient='y', thresh=(50, 150))
    magch = mag_threshold(undist, sobel_kernel=9, mag_thresh=(50, 250))
    dirch = dir_threshold(undist, sobel_kernel=15, thresh=(0.7, 1.3))
    # 100, 255 s-channel
    sch = hls_s(undist, thresh=(88, 190))
    hch = hls_h(undist, thresh=(50, 100))
    shadow = np.zeros_like(dirch).astype(np.uint8)
    shadow[(sch > 0) & (hch > 0)] = 128
    rEdgeDetect = ((undist[:,:,1]/4)).astype(np.uint8)
    rEdgeDetect = 255-rEdgeDetect
    rEdgeDetect[(rEdgeDetect>210)] = 0
    combined = np.zeros_like(dirch).astype(np.uint8)
    combined[((gradx > 0) | (grady > 0) | ((magch > 0) & (dirch > 0)) | (sch > 0)) &
         (shadow == 0) & (rEdgeDetect>0)] = 35
    combined = np.maximum(combined,sch )
```

This is the combined binary mask.

```
Out[17]: <matplotlib.text.Text at 0x7f47761f07b8>
```



As I said the last resubmission the gradients did not give me a better mask than s-channel.

I believe the new movie will allow me to pass this project four.

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

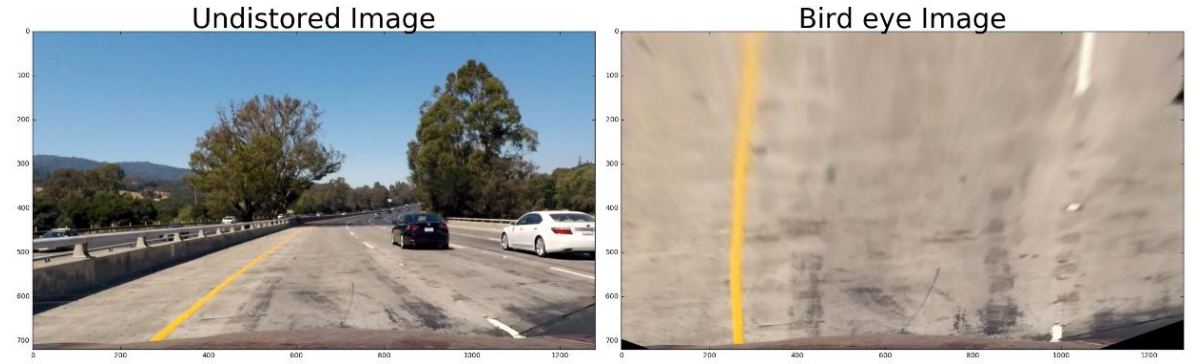
The code for my perspective transform is in code block 8, the first 8 lines as posted below:

```
# define 4 source points for perspective transformation
src = np.float32([[220,719],[1220,719],[750,480],[550,480]])
# define 4 destination points for perspective transformation
dst = np.float32([[240,719],[1040,719],[1040,0],[240,0]])


M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)
# Warp an image using the perspective transform, M:
birdseye = cv2.warpPerspective(undist, M, (1280, 720), flags=cv2.INTER_LINEAR)
```

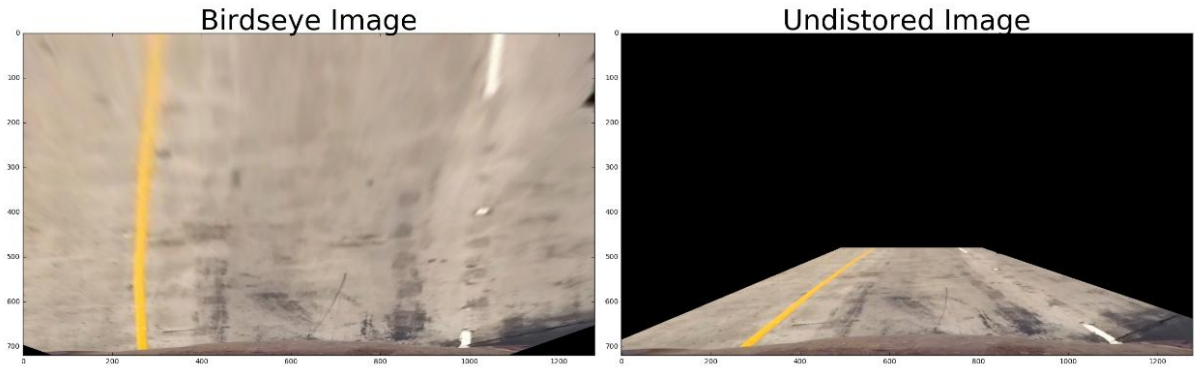
This resulted in the following source and destination points:

Source	Destination
220, 719	240, 0
1220,719	1040, 719
750,480	1040, 0
550,480	240,0

I verified that my perspective transform worked 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.



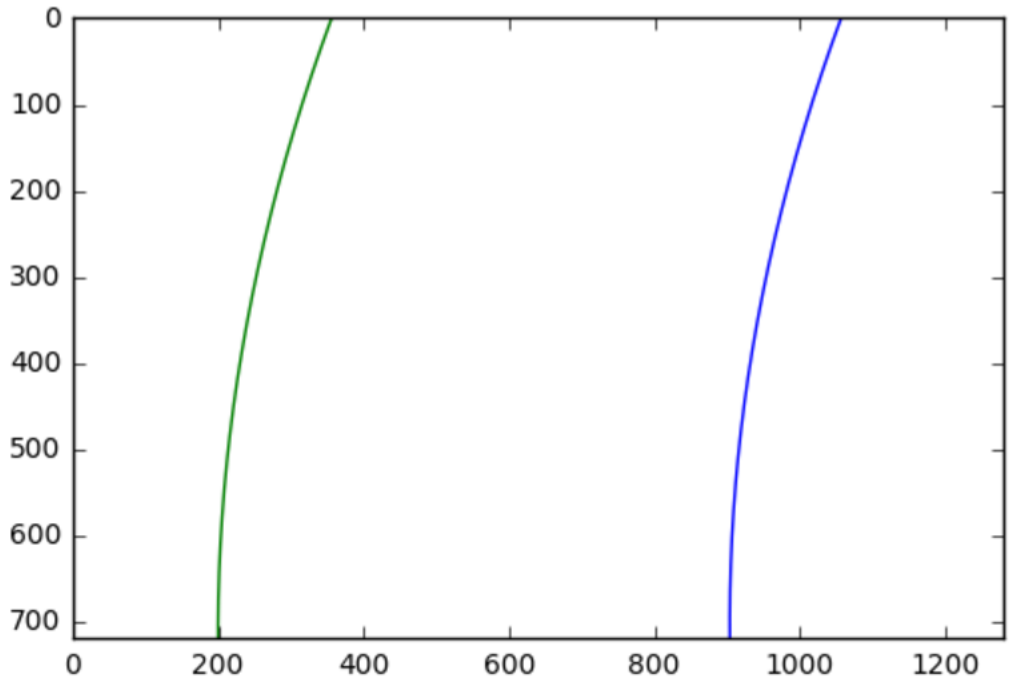
Below is the inverse transformation:



4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

I identified the lane-line pixels by using the s-channel mask from the image. For an initial condition, previous found lines I used the approximate data from the lecture and shown in cell 23.

Below is a plot of the initial assumed first polynomials.



Plot of left line polynomial from pixels show in green, right line polynomial is in blue.

Now that I have an initial condition I can start analyzing frame 0 in the video. If no lines are found, use the previous found lines from the approximate data.

In the process_image function in cell 37 a new polynomial is calculated for each frame.

5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I calculated the radius of curvature using the lecture notes show in cell 37.

```
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval + left_fit_cr[1])**2)**1.5) \
                                 /np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval + right_fit_cr[1])**2)**1.5) \
                                    /np.absolute(2*right_fit_cr[0])
```

The lecture notes provides an awesome tutorial on the radius of curvature here

The vehicle offset to center is discussed in lecture 12 video.

```
# define center

    left_line_baseX = left_fit[0]*720**2 + left_fit[1]*720 + left_fit[2]
    right_line_baseX = right_fit[0]*720**2 + right_fit[1]*720 + right_fit[2]
    scale = 3.7/np.abs(left_line_baseX - right_line_baseX)
    midpoint = np.mean([left_line_baseX, right_line_baseX])
    offset = (1280/2 - midpoint) * scale
```

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

In cell 37 several frames are plotted. Here is one example.

CarND-Advanced-Lane-Lines-master/SomeFrames/frame0013.jpg

Pipeline (video)

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

Here is a link to my video.

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?

My challenges were printing out frames from the video correctly but the video would not. I was not able to debug that issue. I took a new approach on writing my code. I wish I could have gotten gradient help. I have been assured from other cohorts they did not use gradient either.

To make it more robust I would use averaging the frames for less jittering.

Project 4 has been by far my most challenging project. Parameter tuning is difficult at the stage of my experience in computer vision learning. Next Term, I will be better.