Writeup Readme

Project 4 - Advanced Lane Finding Project

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.

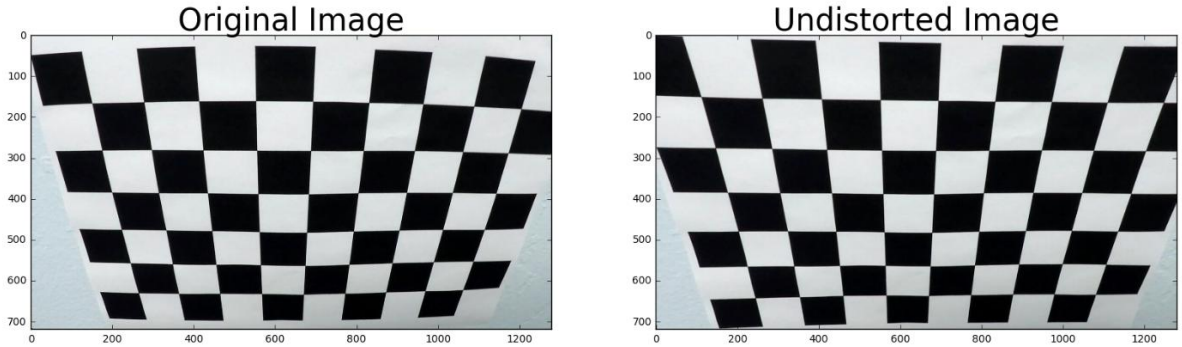
Below are the results of the camera coefficients:

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.



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

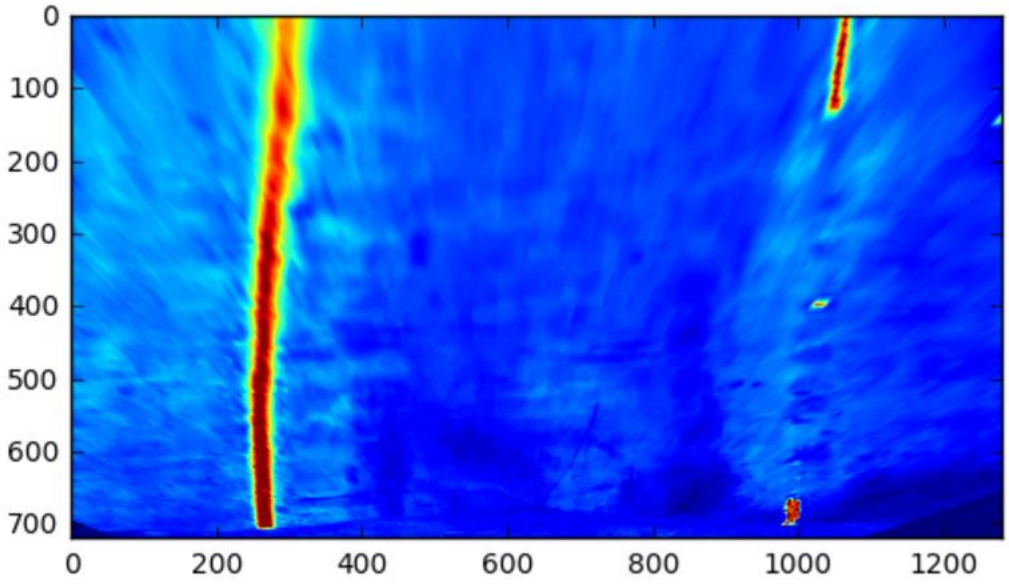
```
def corners_unwarp(img, mtx, dist):
    # Pass in the image into this fumction
    #undistort using mtx and dist
    undist = cv2.undistort(img, mtx, dist, None, mtx)
    return undist

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

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

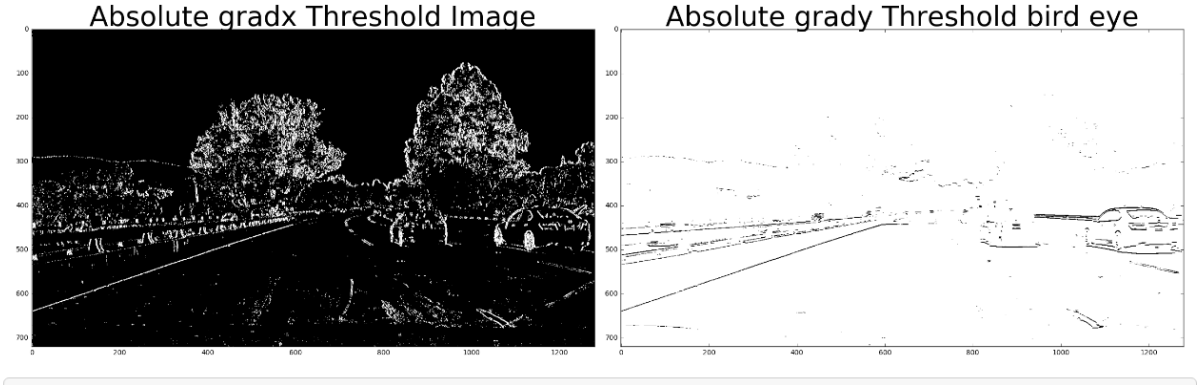
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I initially choose HSV color scheme but switched to HSL as the lecture notes use. I also uses magnitude sobel thresholds. This code is in the for the mask I use is in cell 23 in

https://github.com/DeniseJames/P4/blob/master/P4-20170207.ipynb Here is the HLS and sobel mask.



Below is a absolute gradient of the road image:



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

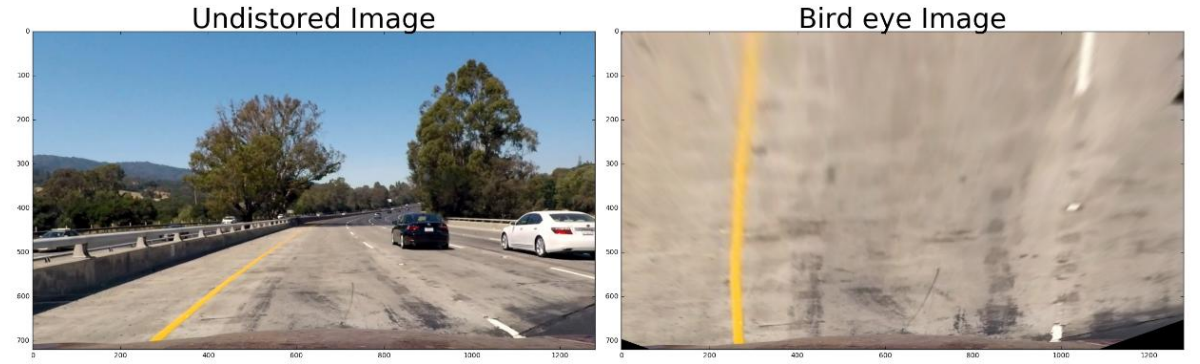
The code for my perspective transform is in code block 8, the first 8 lines as posted below:

```
# 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)
```

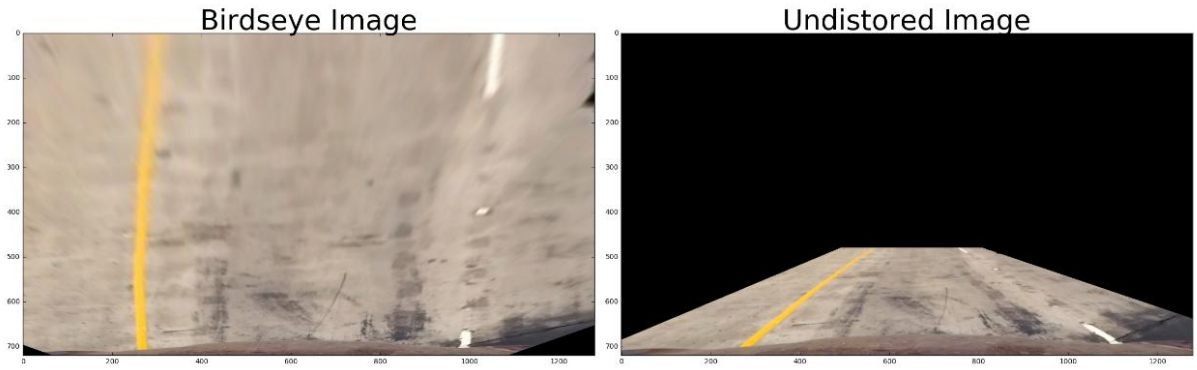
This resulted in the following source and destination points:

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.

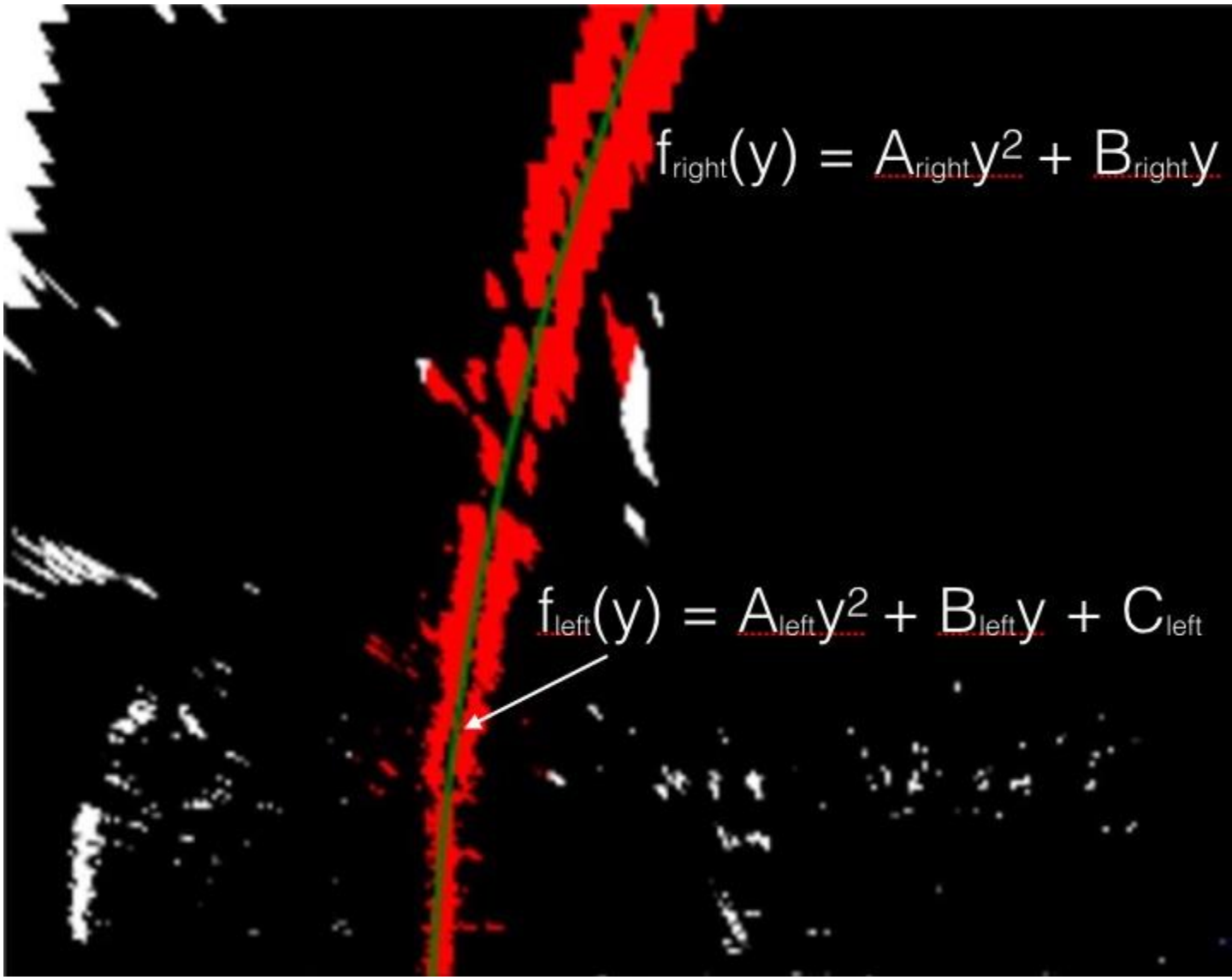


Below is the inverse transformation:



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

Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:



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 did this in lines # through # in my code in `my_other_file.py`

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 lines # through # in my code in `yet_another_file.py` in the function `map_lane()`. Here is an example of my result on a test image:

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'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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.