Project 2 – WriteUp

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.

Rubric Points

Camera Calibration

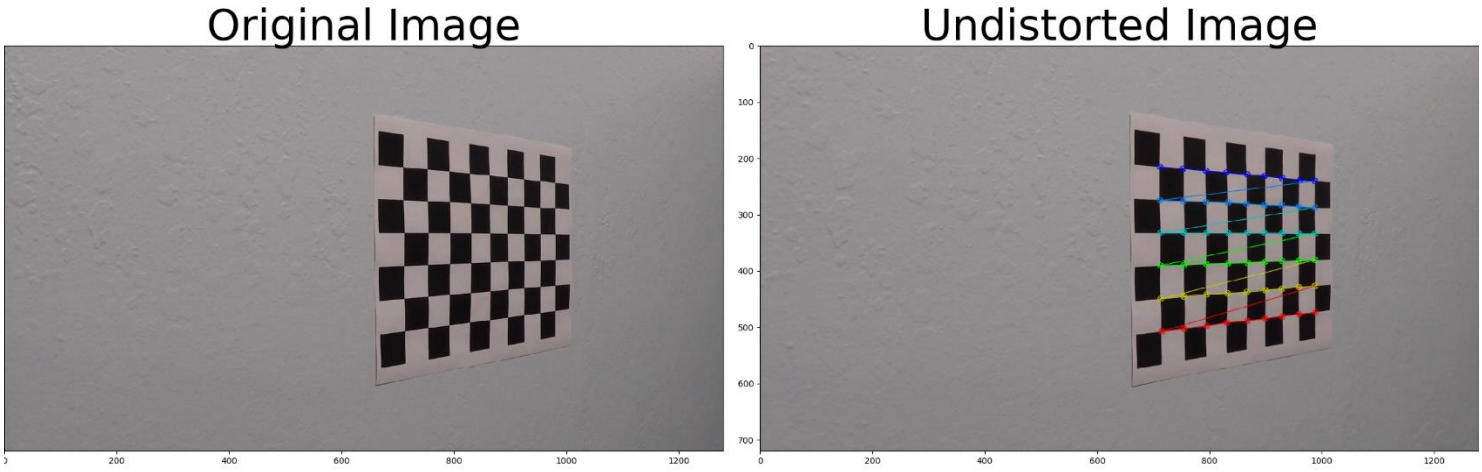
1. Have the camera matrix and distortion coefficients been computed correctly and checked on one of the calibration images as a test?

The code for this step is contained in lines 14 through 42 of the `Project2 AdvanceLaneFinding.py`

```
# Finding the chessboard corners
ret, corners = cv2.findChessboardCorners(gray, (nx,ny), None)
# adding object points, image points
if ret == True:
    objpoints.append(objp)
    imgpoints.append(corners)
```

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. I applied this distortion correction to the test image using the `cv2.undistort()` function and obtained this result:

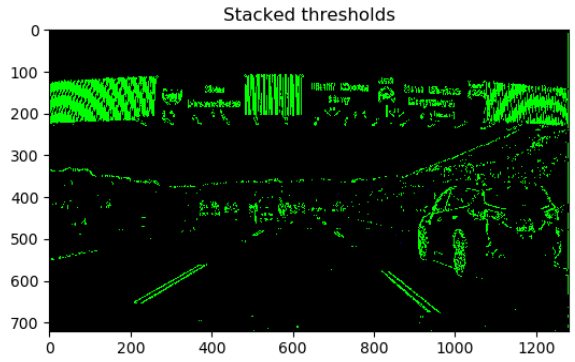


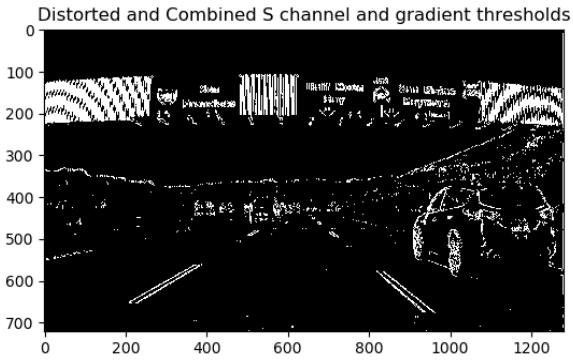
Pipeline (single images)

1. Has the distortion correction been correctly applied to each image?

The code for this step is contained in lines 46 of the `Project2 AdvanceLaneFinding.py`

2. Has a binary image been created using color transforms, gradients or other methods?






```python
def combine_color_thresholds(img):
    #Returns a binary thresholded image produced retaining only white and yellow elements on the picture
    #The provided image should be in RGB format
    hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
    # Select saturation
    l_channel = hls[ : , : , 1]
    # Threshold s chnnel
    l_binary = np.zeros_like(l_channel)
    l_binary[(l_channel >= 110) & (l_channel <= 255)] = 1
    # Select saturation
    s_channel = hls[ : , : , 2]
    # Threshold s chnnel
    s_binary = np.zeros_like(s_channel)
    s_binary[(s_channel >= 120) & (s_channel <= 255)] = 1
    # Combine thresholds
    color_thresh = np.zeros_like(s_binary)
    # Used the OR operation
    color_thresh[(s_binary == 1) & (l_binary == 1)] = 1
    return color_thresh

def combine_thresholds(img, ksize = 7):
    # X, Y gradiets, magnitude of gradient and direction of gradient thresholds
    gradx = abs_sobel_thresh(img, orient='x', sobel_kernel = 15, thresh_min = 20, thresh_max = 120) # was 20-100
    grady = abs_sobel_thresh(img, orient='y', sobel_kernel = 15, thresh_min = 20, thresh_max = 120)
    mag_binary = mag_thresh(img, sobel_kernel = 15, mag_thresh = (80, 200)) # was 30-100
    dir_binary = dir_threshold(img, sobel_kernel = ksize, thresh=(0, np.pi/2))

    # Combine thresholds
    #combined = np.zeros_like(gradx)
    combined = np.zeros_like(dir_binary)
    combined[(gradx == 1) & ((grady == 1) | (mag_binary == 1) & (dir_binary == 1))] = 1
    #combined[((gradx == 1)) | ((mag_binary == 1) & (dir_binary == 1))] = 1
    #combined[((gradx == 1))] = 1

    kernel = np.ones((3, 3), np.uint8)
    combined = cv2.morphologyEx(combined.astype(np.uint8), cv2.MORPH_OPEN, kernel)

    return combined

def thresh_binaryImage(image,ret,mtx,dist):
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    hls_thresh = combine_color_thresholds(img)
    undist_img_gray = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)[:,:,0]
    grad_thresh = combine_thresholds(undist_img_gray)
    # Combine thresholds
    combined = np.zeros_like(hls_thresh)

    # Combine with OR
    combined[(hls_thresh == 1) | (grad_thresh == 1)] = 1

    return combined
```

3. Has a perspective transform been applied to rectify the image?

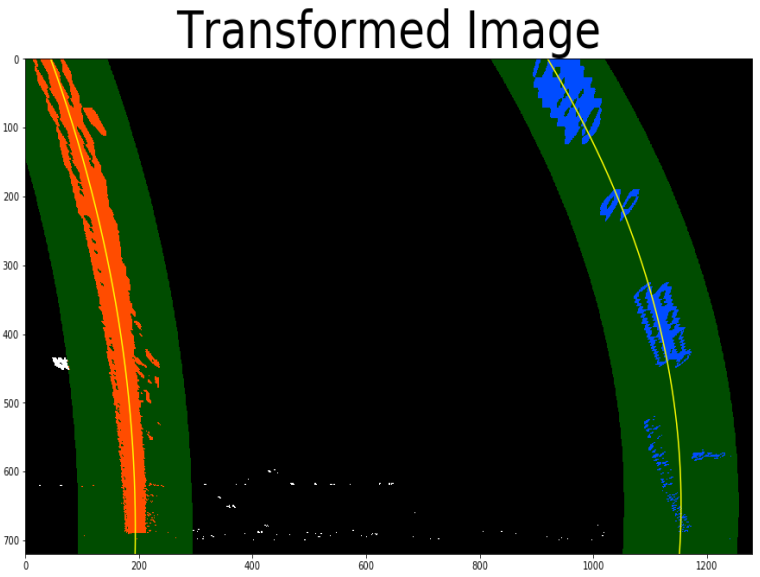
Perspective Transform is done in function Perspective_Transform() line 84 through 95. The source points and destination points are hardcoded as follows.

```
src = np.float32([[567,465], [745,465] , [1103,665] , [251,665]])
offset = 50
dest = np.float32([[offset, offset],
          [img size[0]-offset, offset],
          [img size[0]-offset, img size[1]-offset],
          [offset, img_size[1]-offset]])
```

4. Have lane line pixels been identified in the rectified image and fit with a polynomial?

The Lane Pixels are identified and plotted using fit_polynomial(warped) and search_around_poly(warped,left_fit,right_fit). Using Slide window and finding out the occupied pixels with the equation fund using that.

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 and also that the Lane Pixels were successfully plotted.



5. Having identified the lane lines, has the radius of curvature of the road been estimated? And the position of the vehicle with respect to center in the lane?

Yep, The Curvature and Position of the vehicle using measure_curvature_real(left_fitx,right_fitx,img_size, left_fit,right_fit) function. The Position of the vehicleis determined assuming the camera is mounted on the center of the car.

Pipeline (video)

1. Does the pipeline established with the test images work to process the video? It

sure does! Here's a link to my video result

Below is the pipeline function which converts the video to image processes it and stitches back the video.

```
ret,mtx,dist=camera_calibration()
def process_image(image):
    undist = np.copy(image)
    img_size = (image.shape[1], image.shape[0])
    combined_binary=thresh_binaryImage(image,ret,mtx,dist)
    M,Minv,warped=Perspective_Transform(img_size,combined_binary)
    left_fit,right_fit=fit_polynomial(warped)
    left_fitx,right_fitx,ploty=search_around_poly(warped,left_fit,right_fit)
    left_curverad, right_curverad,center_dist=measure_curvature_real(left_fitx,right_fitx,img_size,left_fit,right_fit)
    curvature = ((left_curverad+right_curverad)/2)
    # Create an image to draw the lines on
    warp_zero = np.zeros_like(warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))

    # Warp the blank back to original image space using inverse perspective matrix (Minv)
    newwarp = cv2.warpPerspective(color_warp, Minv, (image.shape[1], image.shape[0]))
    # Combine the result with the original image
    result = cv2.addWeighted(undist, 1, newwarp, 0.3, 0)
    text='Curvature of lane = '+str(format(curvature, '.2f'))+'m'
    cv2.putText(result, text,(50,50),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2,cv2.LINE_AA)
    if center_dist > 0:
        text = 'Vehicule position: '+str(format(center_dist, '.2f'))+ 'm left of center'
    else:
        text = 'Vehicule position: '+str(format(center_dist, '.2f'))+ 'm right of center'
    cv2.putText(result, text, (50, 160), cv2.FONT_HERSHEY_SIMPLEX, 2, (255,255,255), 2, cv2.LINE_AA)
    return result
```

Sample Outputs:

README

1. Has a README file been included that describes in detail the steps taken to construct the pipeline, techniques used, areas where improvements could be made?

You're reading it!

Dynamically determining the src and dest points for persepective transform would improve for real world problem. The pipeline works very well for the project video, however fails for the challenge video , will keep on working to improve the pipeline.