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

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

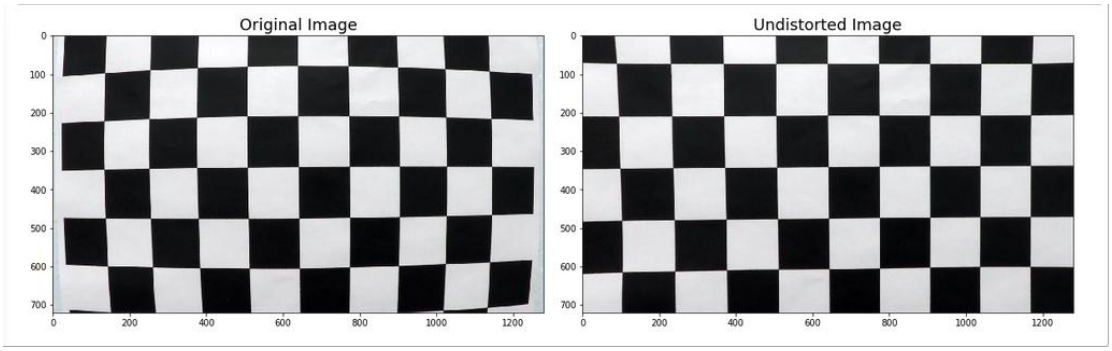
The code for this step is contained in the 1th code cell of the IPython notebook located in "./pipeline_project.ipynb"(also see pipeline_project_video.py from line 30 to 67)

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. Next I save the mtx and dist as 'calib_dist_pickle.p'. It is because once the frame needs be undistorted, it only read 'calib_dist_pickle.p' and find mtx and dist, it don' t need to operate the whole process of camera calibration again.

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



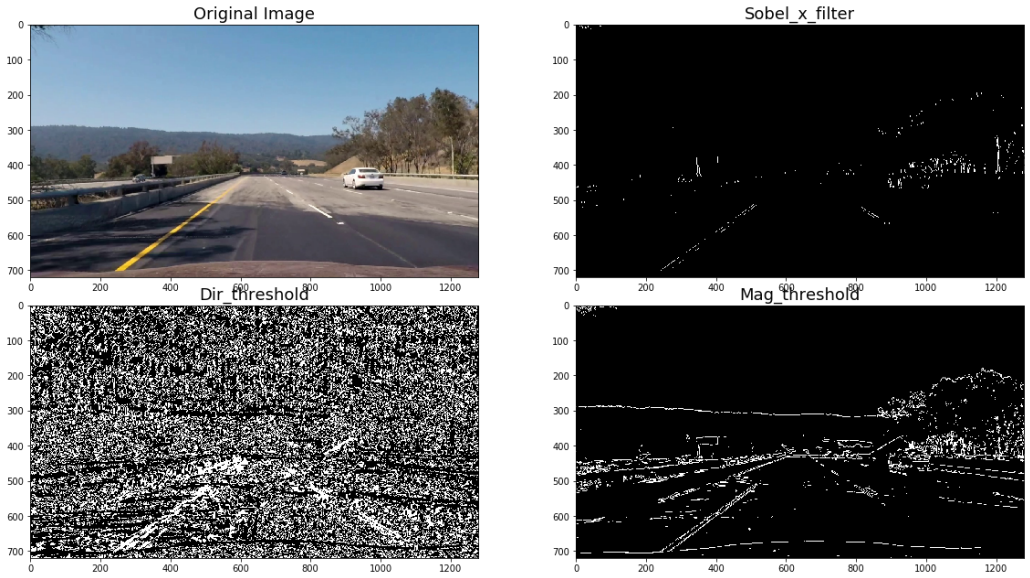
Pipeline (single images)

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 that has strong light and tree shadow like this one:

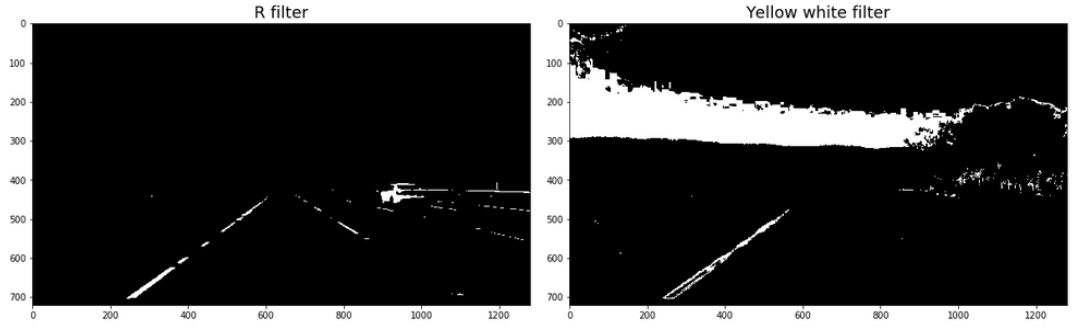
2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

I used gradient: sobel thresholds, Magnitude threshold and direction threshold(The code for this step is contained in the 3th and 4th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 70 to 115). Here's an example of my output for this step.



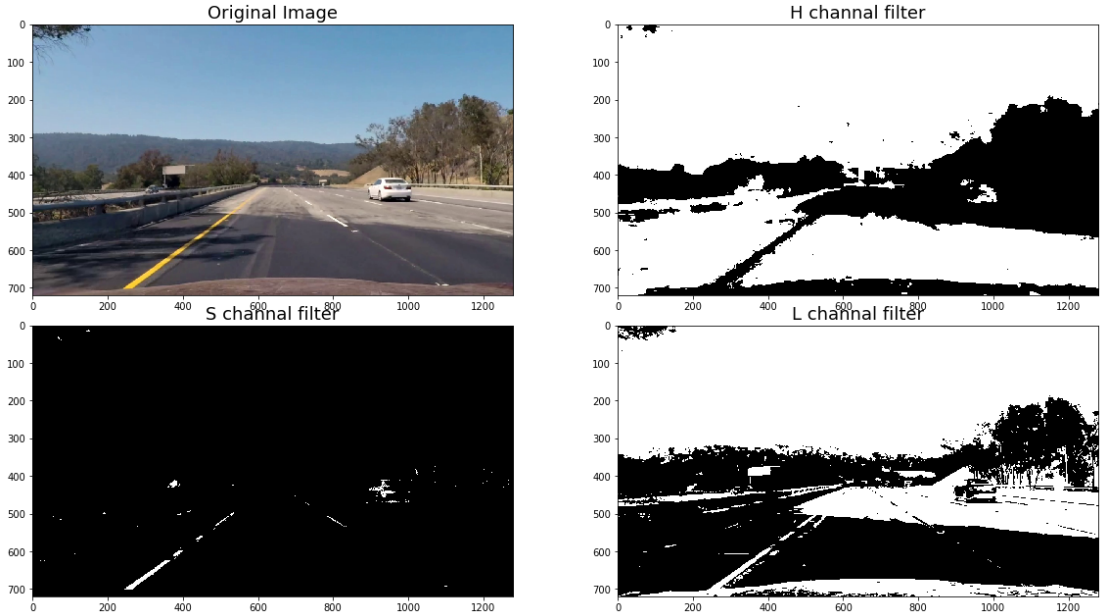
From picture, I can find sobel_x_filter is better than others.

Then I used r_select and hsv color space thresholds(The code for this step is contained in the 5th code cell of the IPython notebook located in "./pipeline_project.ipynb ", and also see pipeline_project_video.py from line 117 to 143). Here's an example of my output for this step.



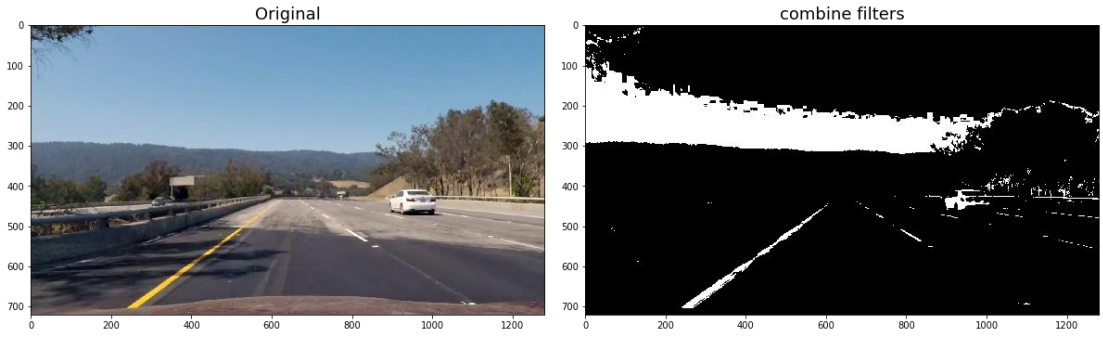
From picture, for right pipeline, I can find R_filter is better than the second filter. However, for left pipline, Yellow and white filter is better than R filter. Of course, it depends on their parameters. I will discuss it in the combination of color and gradient thresholds later.

Then I used HLS color space thresholds(The code for this step is contained in the 6th code cell of the IPython notebook located in "./pipeline_project.ipynb ", and also see pipeline_project_video.py from line 146 to 162)



From pictures, we can find using s_channel filter is better than others. The other two channels seem be affected by external factors such as lights and tree shadows.

Now I used a combination of color and gradient thresholds to generate a binary image (The code for this step is contained in the 7th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 164 to 174). Here's an example of my output for this step.



From picture, I can see the picture that using combine filter can find left and right pipeline but not affected by external factor. My combine code is combined_lsx[((s_binary==1)&(l_binary ==1)|(yw_binary ==1)|(r_binary == 1))] = 1 and other parameters is following:

l_binary = hls_select(img, channel='L', thresh=(100, 200))

s_binary = hls_select(img, channel='S', thresh=(180, 255))

As my lowest threshold value is 180, I can reduce external factors. Of course, as the high lowest threshold cause fewer fitpoints. As a result, I need other filters assist to find more fitpoints. Then I used yw_binary and r_binary. But the reason why I do not use the sobel filter is that sobel_x filter can find pipeline with the strong light and tree shadow but its effect is worse that combine filter with (yw_binary ==1)|(r_binary == s1). This two parameter is following:

r_select(img, thresh=(200, 255)

yellow_hsv_low = np.array([0, 100, 100])

yellow_hsv_high = np.array([80, 180, 255])

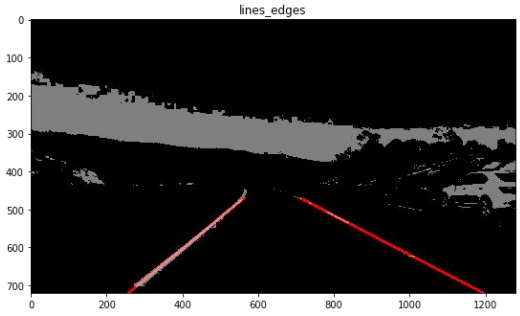
white_hsv_low = np.array([90, 0, 170])

white_hsv_high = np.array([160, 80, 255])

So this kind of combine filter is optimal combine that has few effected by external in this project video frame after testing.

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

The code that for this step is contained in the 8th code cell of the IPython notebook located in "./pipeline_project.ipynb ", and also see pipeline_project_video.py from line 466 to 606) for my project using region_thresholds that limit the region to and HoughLinesP function find fitline, See this picture.



From this picture, we can find the 4 source (`src_corners `) points (a, b, c, d) with two straight lane lines. Two points (c, d) from the bottom of two lines and Two points(a, b) from the top of two lines .

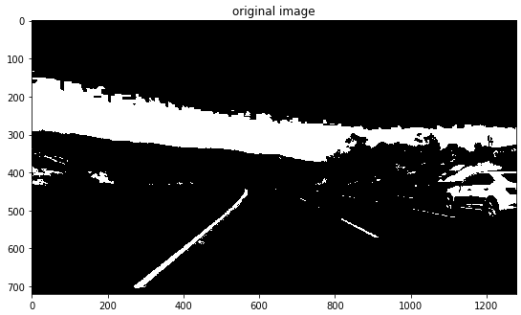
Then the code (This step is contained in the 9th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 610 to 621) for my perspective transforms well as source (`src_corners `) and destination (`dst_corners `) points.

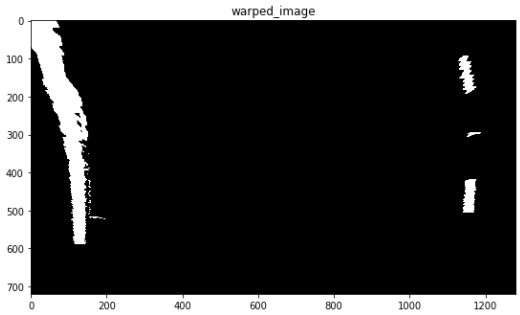
```
src_corners = np.float32([(ax, ay), (bx, by), (cx, cy), (dx, dy)])

dst_corners = np.float32([[offset, offset], [img_size[0]-offset, offset],

                          [img_size[0]-offset, img_size[1]-offset], [offset, img_size[1]-offset]])
```

This resulted in the following source and destination points:

Source	Destination
563, 470	100, 100
710, 470	1180, 100
1193, 720	1180, 620
256, 720	100, 620

I verified that my perspective transform was working as expected by drawing the ` src_corners ` and ` dst_corners ` points on original image and its warped counterpart to verify that the lines appear parallel in the warped image.

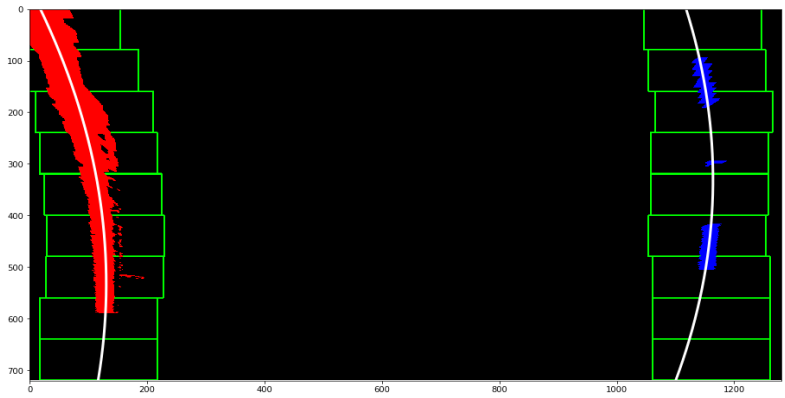




Also in the video, I smooth over the last frames to obtain a good result(the code for this step includes pipeline_project_video.py from line 628 to 638)

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

Then I did Implement Sliding Windows and Fit a Polynomial line from udacity course (The code for this step is contained in the 9th code cell of the IPython notebook located in "./pipeline_project.ipynb ", and also see pipeline_project_video.py from line 239 to 315) and the result is following:



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 calculate the radius of curvature that is contained in the 11th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 336 to 356)and the result is following:

Left_curverad: 373.21 m

Right_curverad:330.33m

It can also can see in the code (pipeline_project_video.py from line 651 to 678). This means that the radius of curvature is very special when radius of curvature of some road is very big .

As some roads in the project_video.mp4 is straight, radius of curvature of some road is very big so that it more than 10k m. So I set the radius of curvature of straight lane line as 0.,

When the radius of curvature of pipeline is more than 1200, judge whether the line is straight with similar slops (less than 0.2). If it neither the radius of curvature of pipeline less than 1200 nor line is straight, the fitline of this frame is very bad. Then delete it.

Moreover, as the right line of road is easy to appear bad radius of curvature but the left line of road is good radius of curvature, I set radius of curvature of right line is equal to radius of curvature of right line. But I verify the video that this situation appear less than 10 frames .

Then if the difference between right line and left line of road is big (more than 500), I calculate the average curvature between right line(0.2 weight) and left line(0.8 weight) because the fitline of right line is easy to appear error. Theoretically, their radiuses of curvature are same.

I calculate the distance from the position of the vehicle with respect to center that is contained in the 11th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 358 to 380 and the result is following:

Diff_center: 0.23 m

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 the 10th code cell of the IPython notebook located in "./pipeline_project.ipynb ", also see pipeline_project_video.py from line 318 to 334. 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](project_video_output.avi)

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.

1. When the pipeline is straight, I set the radius of curvature as 0. But once it has a slight change on pipeline, the result of radius of curvature is nonzero. However, the real pipeline is straight. So the video always appears.

2. The change of radius of curvature is not stable in the lower part video.

For these two problems or failures, focus on the calculation of radius of curvature. I need to try to use other methods to calculate radius of curvature.

3. The right line of road always appear many bad polynomial curvature. So in the future, I may use other methods to achieve polynomial curvature