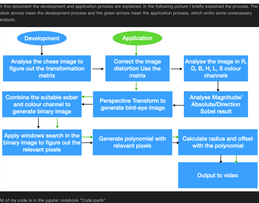
In this document the development and application process are explained. In the following picture I briefly explained the process. The black arrows mean the development process and the green arrows mean the application process, which omits some unnecessary analysis.



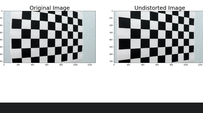
All of my code is in the jupyter notebook "Code.ipynb"

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 first code cell of the IPython notebook, the involved functions are called camera_calibration(),remake distortion coefficient() and undistortion_conduct to do the distortion correction.

The following picture is an comparison:



Pipeline (single images)

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

To apply distortion-correction we use the following example:

The steps are explained as following:

- use the chess image and the function cv2.findChessboardCorners to creat objectpoints and imgpoints
- use function cv2.calibrateCamera and the objectpoints and imgpoints to generate distortion coefficients
- use the distortion coefficients to correct the chess images
  The corrected chess images are located in the folder output_images, with postfix: _undistort.jpg.
  Imagepair for comparisons have the postfix: _compar_on_undist.jpg

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 a combination of color and gradient thresholds to generate a binary image (function abs_sobel_thresh applies the absolute sobel threshold, the function mix_select is used to apply the threshold). Here's an example of my output for this step. (note: this is not actually from one of the test images)

The steps are following:

- firstly, I did a overview for each colour channel i.e. R G B H L S. The results are following:

With the observation of the different color channel, R, B, G. And the investigation of H, L, S, we can find the line color is distinct in R channel and S channel.

secondly, I applied threshhold for the separate channel. So we gain more knowledge about the performance.

Thought the detailed investigation, I confirmed that the Threashhold in S and R channel can distinguish the lane line in the near field. And threshold of absolute gradient in x give a presentation of lane line in the far field.

So I combined the Threashhold in S and gradien x for the further investigation

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 includes a function called `cbreak_warp()`.
The `cbreak()` function takes as inputs an undistorted image (`undist`),
Inside of the function I defined source (`src`) and destination (`dst`) points manually putted in the following manner:

```
    src = np.float32([
                      [[(img_size[0] / 2) - 61, img_size[1] / 2 + 100],
                       [[(img_size[0] / 6) - 20], img_size[1]],
                       [[(img_size[0] * 5 / 6) + 60], img_size[1]],
                       [[(img_size[0] / 2 + 61), img_size[1] / 2 + 100]])
    dst = np.float32([
                      [[(img_size[0] / 4), 0],
                       [[(img_size[0] / 4), img_size[1]],
                       [[(img_size[0] * 3 / 4), img_size[1]],
                       [[img_size[0] * 3 / 4), 0]])
```

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

Then I fingured out the pixels with sliding windows in the binary image (function: find_lane_pixels3() and fit my lane lines with a 2nd order polynomial in the function fit_polynomial() like this:

```
#python
left_fitx = left_fit[0]*ploty**2 + left_fit[1]*ploty + left_fit[2]
right_fitx = right_fit[0]*ploty**2 + right_fit[1]*ploty + right_fit[2]
```

this following image as example:

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 the function with the name measure_curvature_offset(). It takes the pixels and transform them to the real world unit. Then again used the np.polyfit to fit out a 2-nd polynomial. Then I choosed the middle y point to calculate the radius.

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 function finding_lanes_pipeline(). It takes a image or a video from to draw the lane area on. 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?

I took a relative long time to figure out the threshold. If we can use deep learning algor to figure out them, it would be better.

I tried the pipeline in the challenge video. But it works not so well. Because the road edge is so close to the lane line. The separation well builds a corner. May be we can solve it, if we find a threshold to filter it out.

Furthermore, one issue I faced was the unstable prediction under the shadow. As like the following picture.

Then I added a sanity check to solve the problem.
Only if the condition 1/2 < left_curvature / right_curvature < 0.01 and the condition 2) (|scalar(right_lane_position-left_lane_position) -4.5| are fulfilled simultaneously. The found lines are considered as valid result.