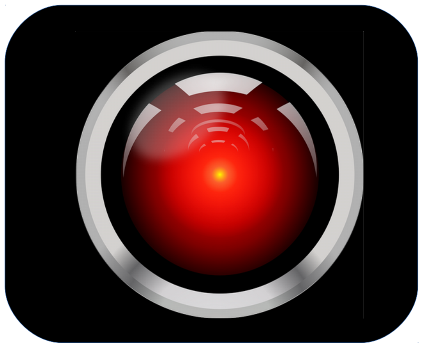
Advanced Lane Finding Project

Camera Calibration

1. The camera matrix and distortion coefficients

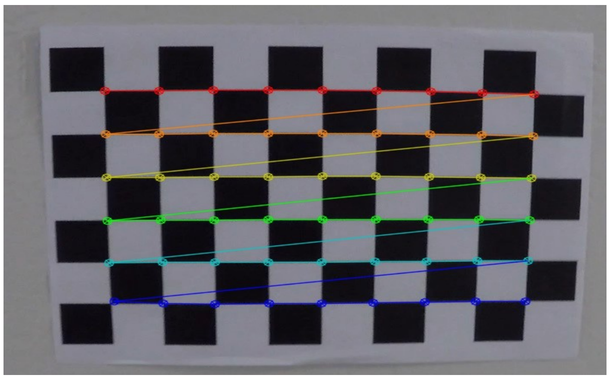
Professionals do this calibration step in front of their pipeline to improve the results. SDC use normally wide angle or fish eye lenses for cameras. Here you can see one of the famous fish eye in history: the Nikon 8mm full format cameras lens - more know under the name “HAL9000”:



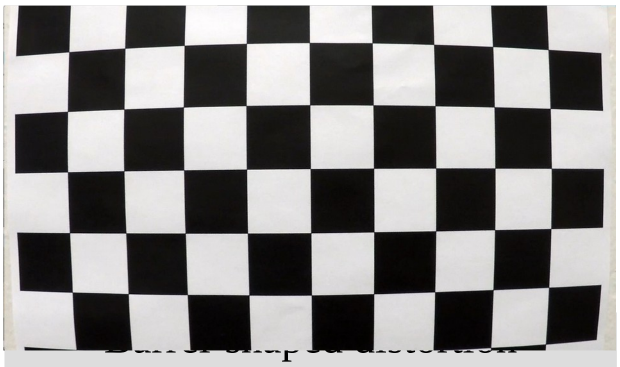
However this kind of lenses produce typically a barrel-shaped distortion for optical reasons. But with some math this distorted images can be transformed back to undistorted images.

First you need a minimum 20 different images of chessboards like show beside.

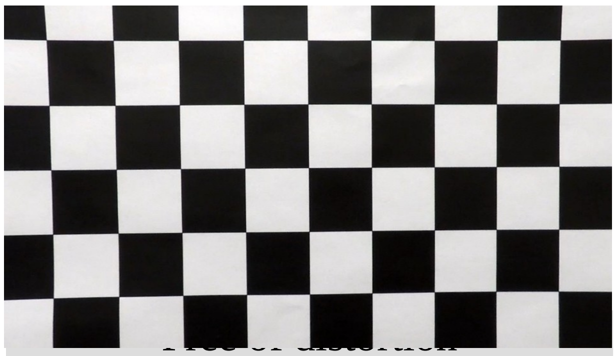
For this step you need the python code *camera_calibration.py*. First you have to count the inner corners of the chessboard and prepare the object points here 6*9 in the code (line 8,9) and the main function *cv2.findChessboardCorners* (line 24) and *cv2.drawChessboardCorners* (line 32). Here you can see how that looks like:



With that point and the gray image the main function, *cv2.calibrateCamera()* can be used. It returns the camera matrix, distortion coefficients, rotation and translation vectors etc. I applied this distortion correction to the test image using the *cv2.undistort()* function and obtained this result:



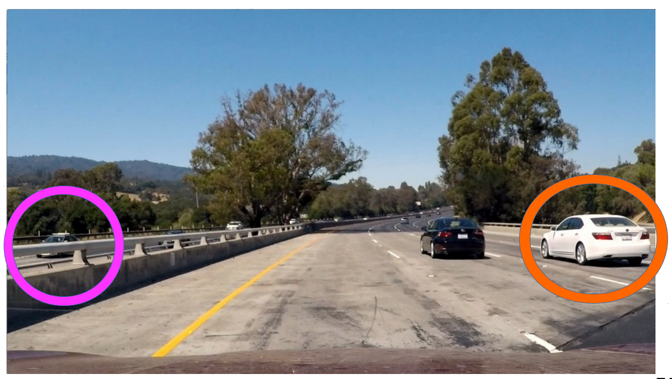
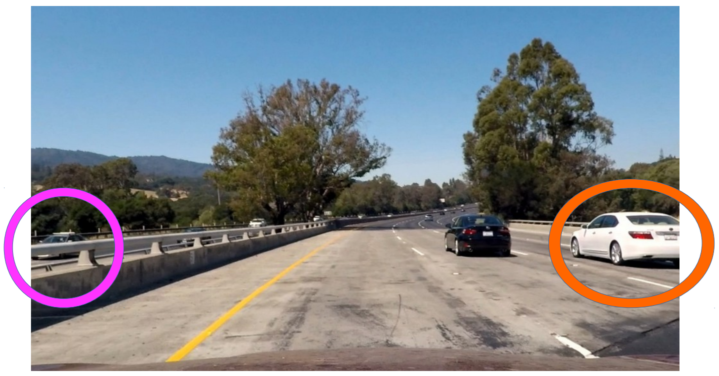
Barrel-shaped distortion



Free of distortion

Pipeline (single images)

1. Applied the distortion correction to each image

Here you can see the small differents on the left and right images. For this transformation I use the *cv2.undistort()* and the values from the chessbord above.

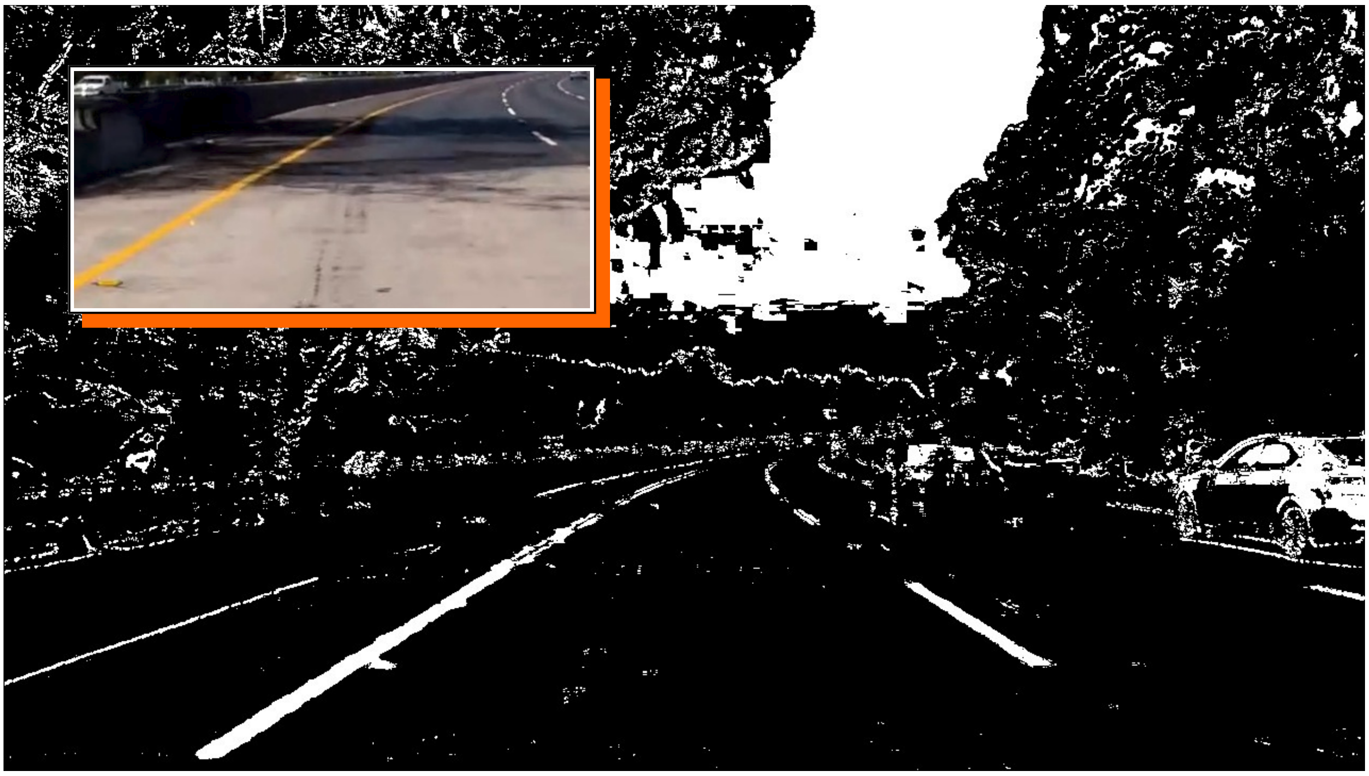
2. Create a binary image by using color transforms and gradients methods

In Line 93 in *video_gen.py* you can see how the binary was calculate with “and” and “or” from three different binaries. It takes x- “and” y- Sobel (def line 29) which you can imagine as a +-45° Sobel. This will be overlay with an logic *or* of the color thresholding (def line 71). This function use a logic *and* of the s-channel of the HLS color space and the s-channel of the HSV color space. This results in a very good binary for the video. Here you can see the binary of a very difficult time point in the project video for binarize:

- difficult tree shadow

- changing dirty road surface

- curve situation



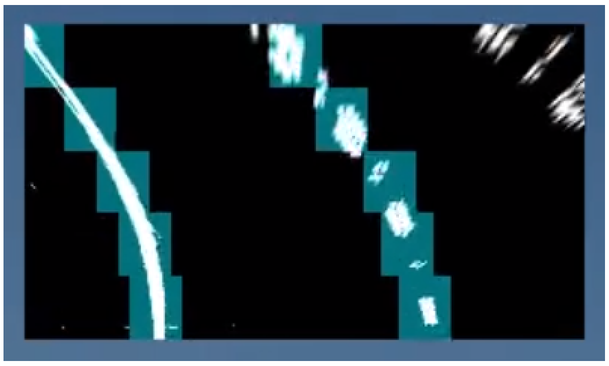
3. The perspective transform of the image

The code for my perspective transform includes a function called warper(), which appears in lines 216 in the main file *video_gen.py*. The *warper()* function takes as inputs an image (img). First the Function calculates the source (src) and destination (dst) points with it's hardcode four trapizoid and rectangle points line (97-106). Of course this has to change for the challenge for example, in order to get the bird view map.

You can see the result of this transformation the bird view in the Video in the top right.
Please have an attention of the width of the road. In case of a strait road **the lanes are in parallel**.

4. The tracker object

This is the heart of the project. First the object *curve_centers* (tracker.py line 269) must be defined as global and initialized (line 274) outside the image loop of the video (line 276). Than it will be called in line 219 and return all the points of the left and right lanes including some statistics information.



The number of the returned points are defined by the rounded down quotient of the *window_height* (line 272) and the vertical size of the warped image. I choose 144 pixel for five convolution boxes. The *window_width* (line 60) give the margin which limits the trackers lane search in horizontal direction, after he found the first one. For the given video 60 is well, also for the sharpest curves. In the image right you see that the up left box is corner to corner connected to its neighbor. This show us that the margin is fully used and fits well here.

Very important is the smooth factor, which controls the amount of averaging the lanes to make them smooth. For averaging all time point choose 0, but don't expect too much from the result by this or hight numbers. 10 has a well damping effect on the moving lanes. Choose 1 for no damping for the lanes.

Here is a short description what is going on in the object itself. The initialization defines the variables of the object - all are constant without the array ***recent_centers =[]*** (tracker.py line7).
which stores all the found left and right point. The points will search in the main and only function find_window_centroids of the object. The magic will done in line 34 and 36 with **convolve**, which compute hight numbers from thick lines in every horizontal slice (size=w*indow_height)*. One for the left and one for the right side of the bottom slice of the image. After that the level loop (line 42) search the next point in the limit of the given margin and store the results in *window_centroids.*

The radius of curvature of both lanes were calculate in the lines 73 through 76 and 101 in *tracker.py.*
The position of the vehicle with respect to center was calculate in lines 257 and 258 in *video_gen.py.*

Now the Sanity Check will apply. This needs a closer look and we discuss that in the next chapter.

This data will store in the object itself (line 99) and return the average of the last results given by the smooth factor. Also the average of the left and right curve radius and then lengths of the object. This is useful to count the imageframes.

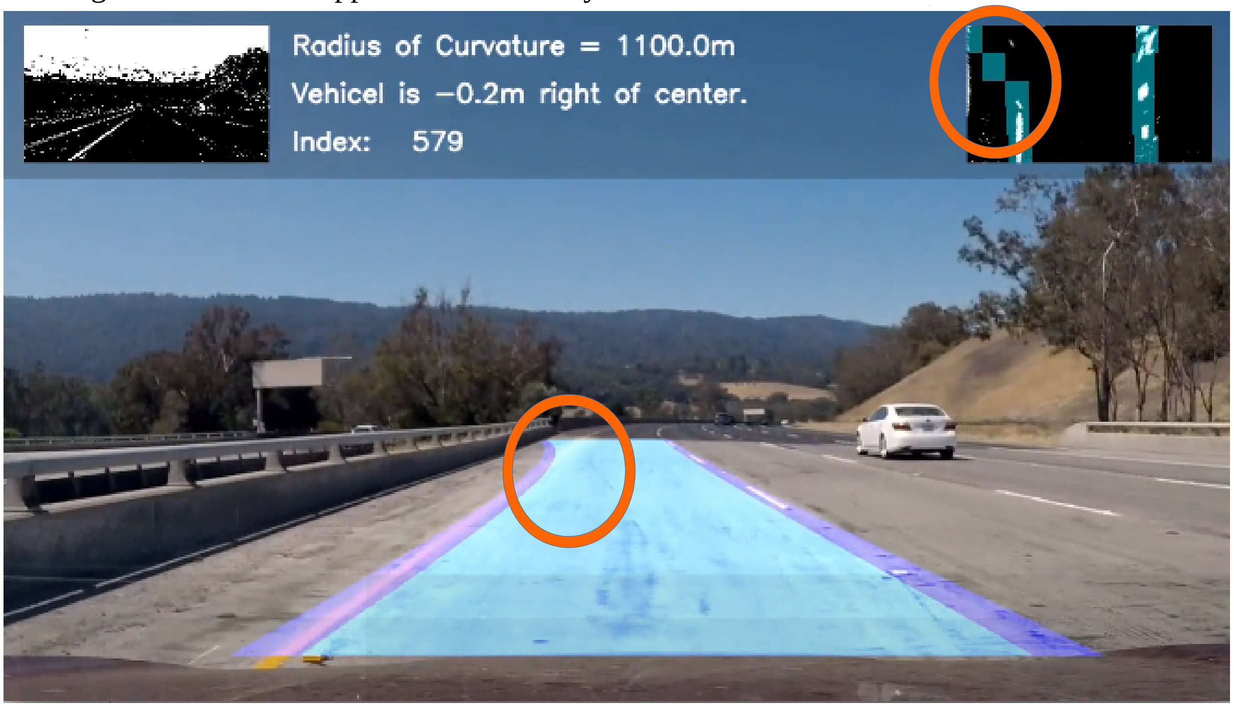
All together the tracker give us the left and right lane point from the given image. Then I fit my lane lines with a 2nd order polynomial kinda like shown in the next chapter.

5. The Sanity Check

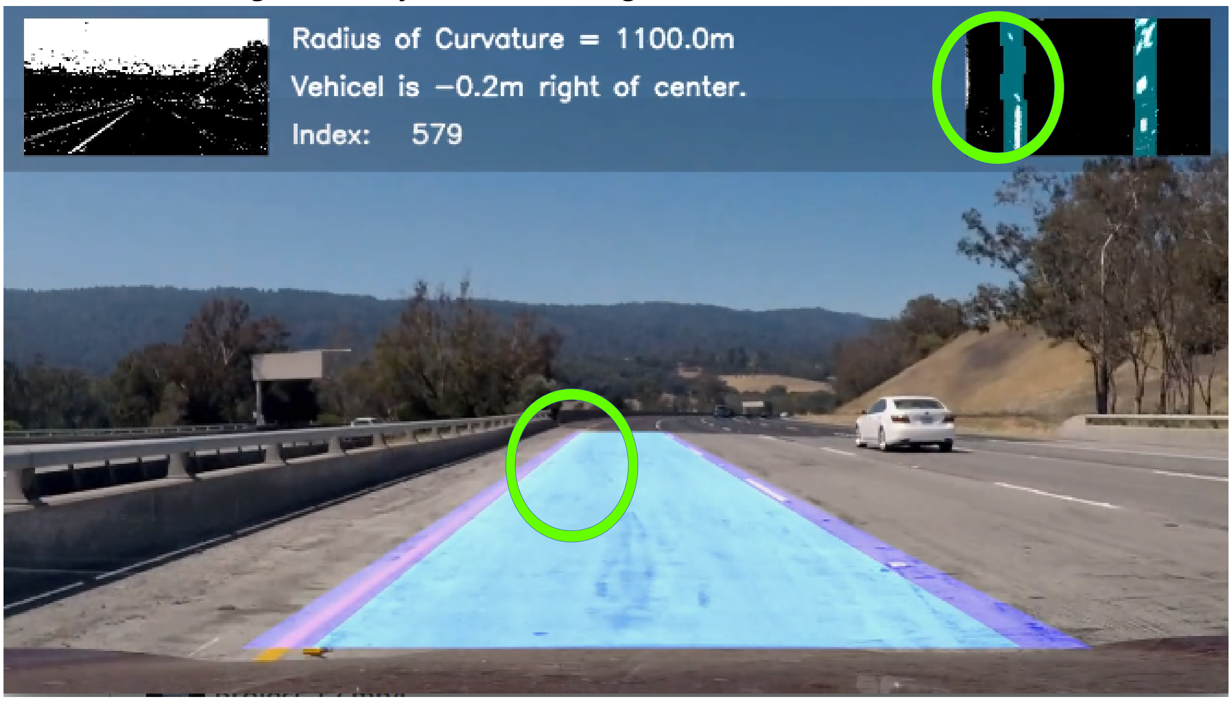
The main idea is to identify wrong results an fix them in the best manner.
The first thing is the check the different of the left and right curve radius. If thous radius differ more than a the given *curve_faktor,* the check routine will continue *(line 65,79).*

After that the mid centers and the overall half road wide will be calculated from the recent centers (line 81-84) we calculate the left half road wide from the mid centers to the new left center point of each level. If this wide is outside the defined tolerance (line 67) of the expected half road wide, we have to fix this issue. The wrong left result will be replaced by right result of this level minus two times the overall half road wide plus or minus the tolerance to over-fit this fixing process a bit.
For the right side we do the same thing in mirror image. What sounds so difficult is easy show in this images from the video:

Here you can see that the curve radius of the left and right calculated lane differs. Also the road is wider at the top. This is because the binary don't catch the lane and the convolution window moves left due nothing to detect. This happens without sanity check.


Radius of Curvature = 1100.0m
Vehicel is -0.2m right of center.
Index: 579

Now the same thing with sanity check and fixing - nice:


Radius of Curvature = 1100.0m
Vehicel is -0.2m right of center.
Index: 579

Pipeline (video)

1 *The nice output*

For debugging reasons and of course for your eyes there is an additional function to bring all this into the video frame. The name of this function is *nice_output* and will be called in line 260.

It will take the there images and the variables for the info screen. The function (122-169) put a rectangle mask on the video image and put in the left corner the binary and the debugging image with the convolution. Between that and there lines of information with curve radius, of center information and the video frame index.

Discussion

1. Improvments

The result is pretty well, because the binary is very well. Only one situation from the sanity has no left lane. Here is an improvement possible to detect the yellow left lane differ from the bright right one for better results. But I don’t go this way. Never the less here are better solution and tuning possible, but this is time consuming. Therefor I add a basic sanity check. The sanity check also has space for fix every problem of any challenge. For example :
If the left and right results has to be fixed than calculate from the overall half road wide the left and right center points.

I like to write a program to detect the speed of the car. Therefor a point must be follow by a new vertical tracker in the bird view. With the distance of a moving point and the time between each frame we can calculate the speed.

But in general a **CNN must learn to do all this improving jobs of tracking lanes**. It is so clear CNN has convolution layer and can detect edges better than any pipeline. Also years of work on this sanity check will not be able to do a CNN job. Maybe this will happen in the next Project P5… How knows?

Acknowledge and Source

Thanks to for the very good intro of P4 I got from the Q&A linke from the classroom.

https://www.youtube.com/watch?v=vWY8YUayf9Q&feature=youtu.be

Also for the idea of the nice output. This I got adapted from here:

https://github.com/ndrplz/self-driving-car/tree/master/project_4_advanced_lane_finding