Self-Driving Car Nano Degree

Advanced Lane Finding Project

Author: Bugra Turan, 9.3.2018

A rough overview of the pipeline for lane detection and evaluation:

- 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.

In general, most of the solution to this project was discussed in the online course so this was used as a starting point for single frame processing. In the following I will go through the whole process pipeline and describe the structure of the IPython Notebook.

Workflow

Camera Calibration

The principle of camera calibration was given in the lecture. Roughly the following steps are required:
a) An object and image point list to store the detected corners from the chessboard images.
b) A set of chessboard images taken from different viewpoints and distances.
c) Finding the corners of the chessboard structure using the OpenCV2 function `findChessboardCorners`
d) A function that calculates the transform matrices. Here, we use the cv2 function called `calibrateCamera`

The helper function `calib_camera()` does the calibration and returns the required matrices for undistortion of the images.

Distortion Correction

The distortion correction algorithm from the course was implemented. A picture of the proper undistortion can be seen below:

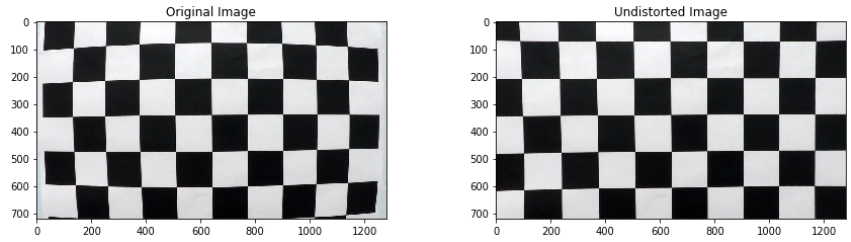
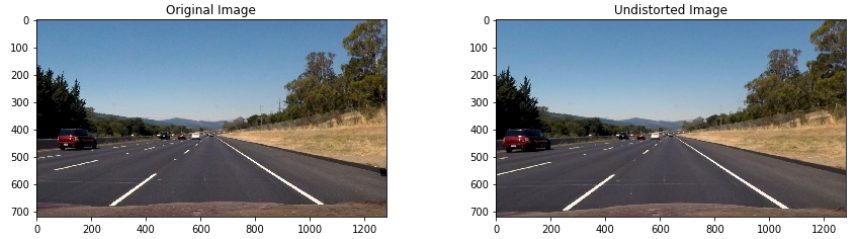
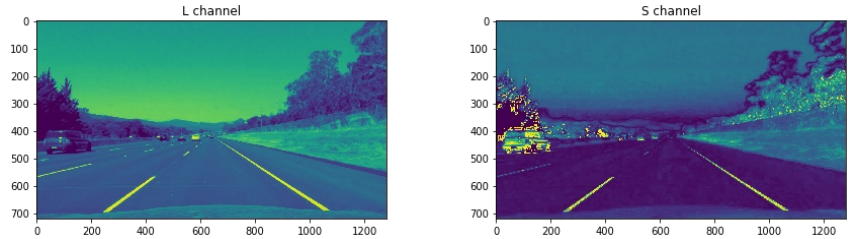


Image Gradient, Threshold detection

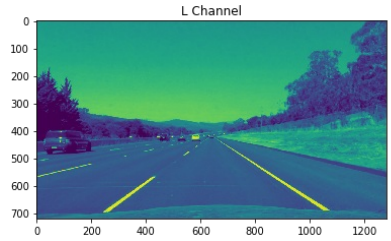
Various methods were used for the image processing for the binary lane identification. I will explain the process step by step shown for one of the test images. First, the image was undistorted:

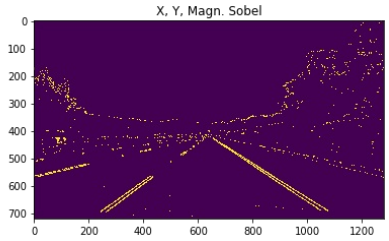


One can see the corrected position of the red car nicely. This step is pretty straight forward.
Next, the image was converted into the Hue, Light, Saturation colorspace since, according to the course features can be identified more easily. A picture of the Saturation and Light channel is shown below. Note that the Hue channel is not used further on.

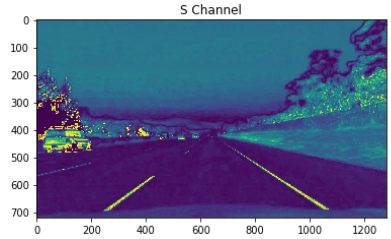


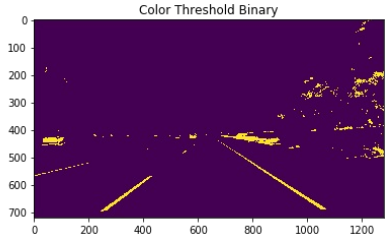
Afterwards a Sobel filter is applied to the Light channel in the x and y direction, respectively. Also the overall gradient is calculated is a measure of the direction as well as the magnitude. A threshold condition is applied to all three features and combined logically. The resulting picture can be seen below:



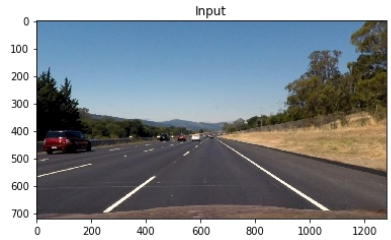


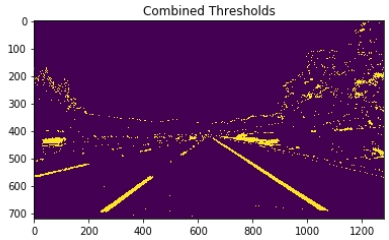
Furthermore, a color threshold is applied to two, L and S channels and these conditions were combined as well. The pictures show the applied threshold on the S channel.





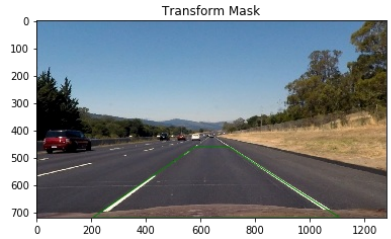
In a last and final step both threshold images are combined with an bitwise OR function and scaled to 0 and 255 for further processing. The final mask can be seen on the right with the original undistorted image:

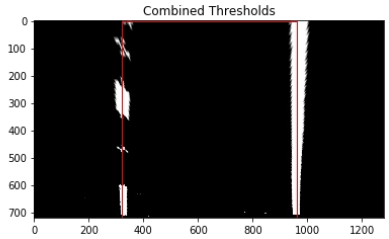




Perspective Transform

For a better view of the lanes and especially their curvature the image is transformed or warped to a Bird's view. For this an image where the lane lines are very straight is used to define four so-called source points in the image the represent the region that is going to be transformed. Respectively, another set of four points was defined as a destination frame. The images below show the image before and after transformation as well as the sets of points that define the transform.

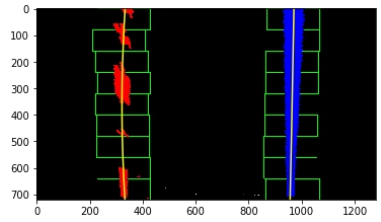




The source points are overlayed in green, while the destination points can be seen in red on the right.

Lane Detection

Now that we have the correct perspective and the mask with mostly only the points from the lane lines we need to interpolate our function. I will describe only the difference to the approach from the course. The windows search and polyfit afterwards was more or less the same as in the course. The result of the detection is ploted in the following:

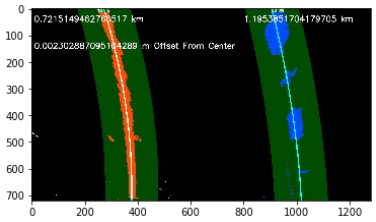


Please note that also for sequences the fast method for fitting subsequent frames was implemented but not used. The result of the fitting are two polynomial functions of the second order with three parameters each.
`[ 8.85740264e-05 -6.47568281e-02 3.33829009e+02] [-2.57766104e-06 -1.80493177e-02 9.70771532e+02]`
The corresponding equation is:
`A*x^2+B*x+C`

Radius of Curvature and Position Offset

With this equation we can calculate the curvature of the lane as well as the offset of the car from the center of the lane. The curvature calculation was taken

from the lecture and the positional offset takes the positions of the lane lines directly at the car and sets them into relation with the center of the car which is in the middle of the y-axis of the camera.



The polyfit is overlayed as well as the curvature, offset and colored points in red and blue for the left and right lane, respectively.

Back-Transformation

For better visualization the image is transformed back from the bird's view. A polygon that is spanned by the polyfits is overlayed onto the image along with all relevant infos.

In the following the complete pipeline as described here was integrated into one function for the processing of video sequences.

Pipeline (Single Images)

The pipeline was described above on a functional level. For better processing of multiple images the whole pipeline was defined in one function. I will briefly describe what the sub-functions do:

1) `undistorted, combined_thre=lane_detect(img, mtx, dist)`
lane_detect() does the preprocessing of the image and returns an undistorted image for the overlay of the results as well as the threshold mask with the lane line points. The raw image is passed as input as well the camera correction parameters.

2) `binary_warped=warp_image(combined_thre)`
This function does the transformation of the pixel mask into the Bird's view.

3) `left_fit, right_fit, nonzerox, nonzeroy, left_lane_inds, right_lane_inds = fit_lanes(binary_warped, bTracking)`
The function fit_lanes() does the lane line fitting and returns the fits as output. Furthermore it takes an bool for activate tracking of the fits when a video sequence is processed. This tracking manges a list of 20 fits and uses the median of all fits from this list. The behavior is basically a low-pass filter of the fit so a certain dynamic properties are lost, which can be seen in certain frames, but in overall the detection is much more stable.

4)
`result, left_fitx, right_fitx, ploty=plot_fit(binary_warped, left_fit, right_fit, nonzerox, nonzeroy, left_lane_inds, right_lane_inds)`
The function plot_fit() is used for plotting of the fitted lines.

5) `result=unwarp_image(binary_warped,undistorted, left_fitx, right_fitx, ploty)`
If the flag back_transform is set to true the function unwarp_image() transforms the image back from the bird's view.

6) `result=overlay_results(result, left_fitx, right_fitx, ploty)`
Lastly, the function overlay_results() puts all the extracted information into the images or videos.

Certain flags are used to set the pipeline into the correct mode. First all test_images were processed and saved into the output folder in bird's view. Afterwards the test_images were processed again but this time with back transform of the image.

Pipeline (Video Sequence)

To process the videos the same approach was done using MoviePy as it has been already done in the first project.

Here's a link to my video result
For the video pipeline it is possible to use the fit tracking by setting the flag bTracking to true. A lower value for the number of list elements could improve the dynamic behavior.

Discussion

There was quite some solutions given in the lecture to tackle the task. I had some problems to determine the correct thresholds in the preprocessing since the tested values worked reasonable well on the test images but failed in certain situations in the videos. Furthermore, the tracking of the fits and conditional appending was not working so well as I expected. Nevertheless I think this is a good approach that I will try out further on. Right now every fit is added to the list. A conditional appending will help massively.

Lastly, especially on the challenge videos it could be seen how difficult it is to really cover all situations with changiing light conditions. I guess it would be really hard to tackle these situations without any automatic threshold adjustments in the image processing.