Advanced Lane Finding Project

Udacity SDCND Term 1

Daniel Tobias

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.

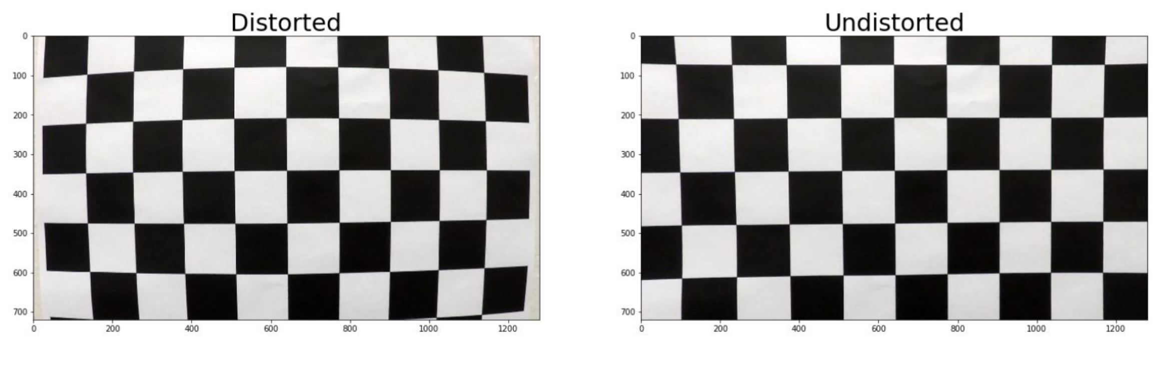
Camera Calibration

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 second code cell of the IPython notebook located in "./AdvanedLane.ipynb" with a title called **Camera Calibration**

The first thing I did was create an array of all combinations of values ny(0-8) and ny(0-5) this is the amount of corners. A corner is considered where 2 black squares and 2 white squares of the chessboard meet. It is assumed that there is no z value for each chessboard.. This array called `obj_pt` is the array of coordinates, and `obj_pts` will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. `img_pts` will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection. In all the I was able to successfully detect 18 out of 20 images sample images for calibration

I then used the output `obj_pts` and `img_pts` 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:



Advanced Lane Finding Project
Udacity SDCND Term 1
Daniel Tobias

Pipeline (single images)

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

In the second code block titled **Camera Calibration** on line 50,51,53 you can find where an example image was loaded with OpenCV and applied with the cv2.undistort function thena BGR to RGB operation so that matplotlib plt could plot the image in the notebook.

Same Image Distorted and Undistorted

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 reference code from the **fifth** code block labeled **Thresholding Pipeline**.*

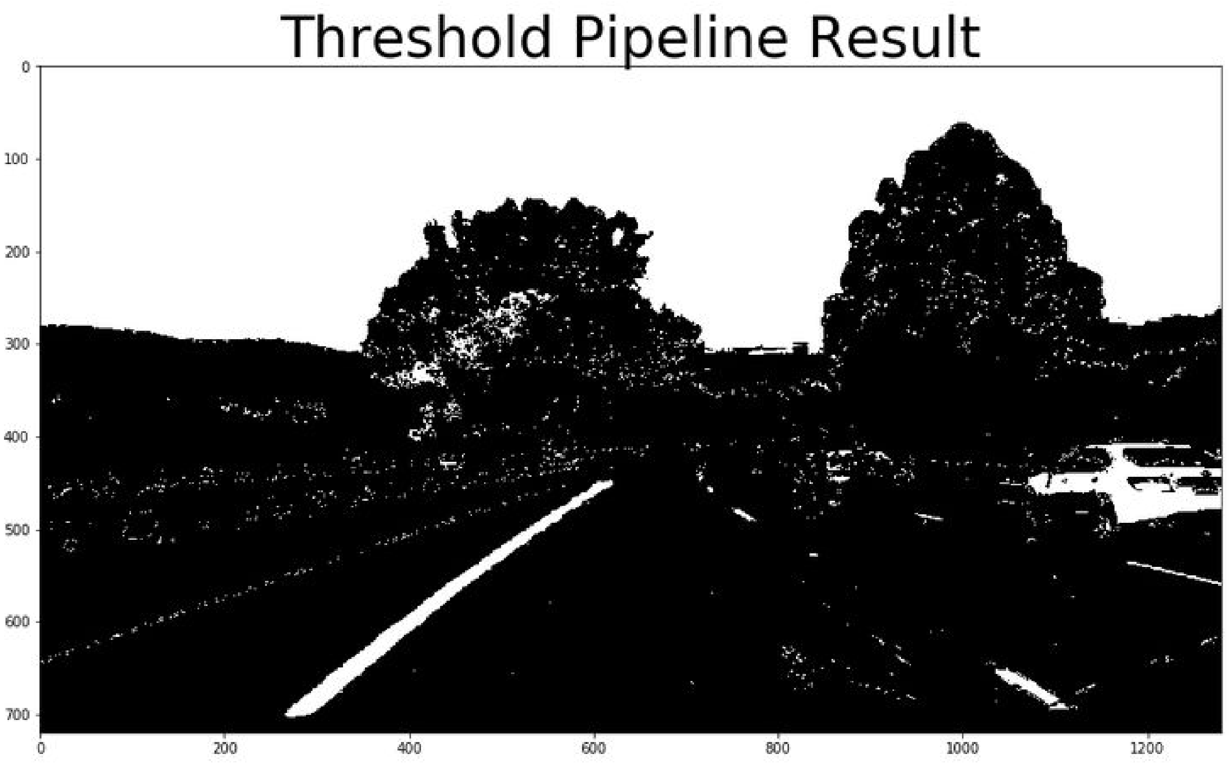
I tried a couple of combinations of the functions that the lesson covered over. A good combination that worked was to convert a copy of the image to HSV and separate the saturation and lightness channels. For the saturation channel the operation looked for a range of values from 100 to 255 and placed a 1 in a similar sized array, you can see this operation in code block 5 @ line 10,11. A similar operation happened with the l_channel @ line 14,15 and placed a 1 in a different similar array. Then in line 18,19 I logical **AND** the elements of each binary representation of the values that fell in a certain threshold.

For the second part(line 22-30) of the thresholding pipeline I used the **dir_threshold** function from code block 4 and apply it with the parameters (0.8,1.2) to create binary thresh of the directional derivative of the image. I then used the **mag_thresh** from code block 4 on the image which returns a binary threshed version. The results from the dir_threshold and mag_thresh are anded together to create a combined_mag_dir. Then the binary version from the saturation and lightness thresholding and the direction and magnitude threshing are OR together to create the threshold pipeline

Advanced Lane Finding Project

Udacity SDCND Term 1

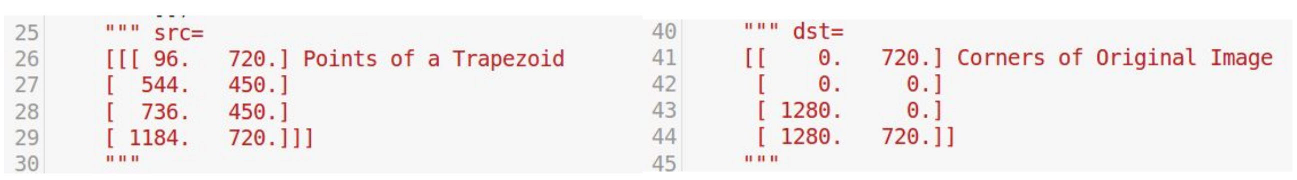
Daniel Tobias



Here is the resulting binary image of the thresholding pipeline from cell block **5**

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 `warp()`, which appears in lines 7th code cell in the in the jupyter notebook AdvancedLane. The `warp()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose a region slightly larger than needed



```
    """ src=
    [[[ 96.   720.] Points of a Trapezoid
    [  544.   450.]
    [  736.   450.]
    [ 1184.   720.]]]
    """
```

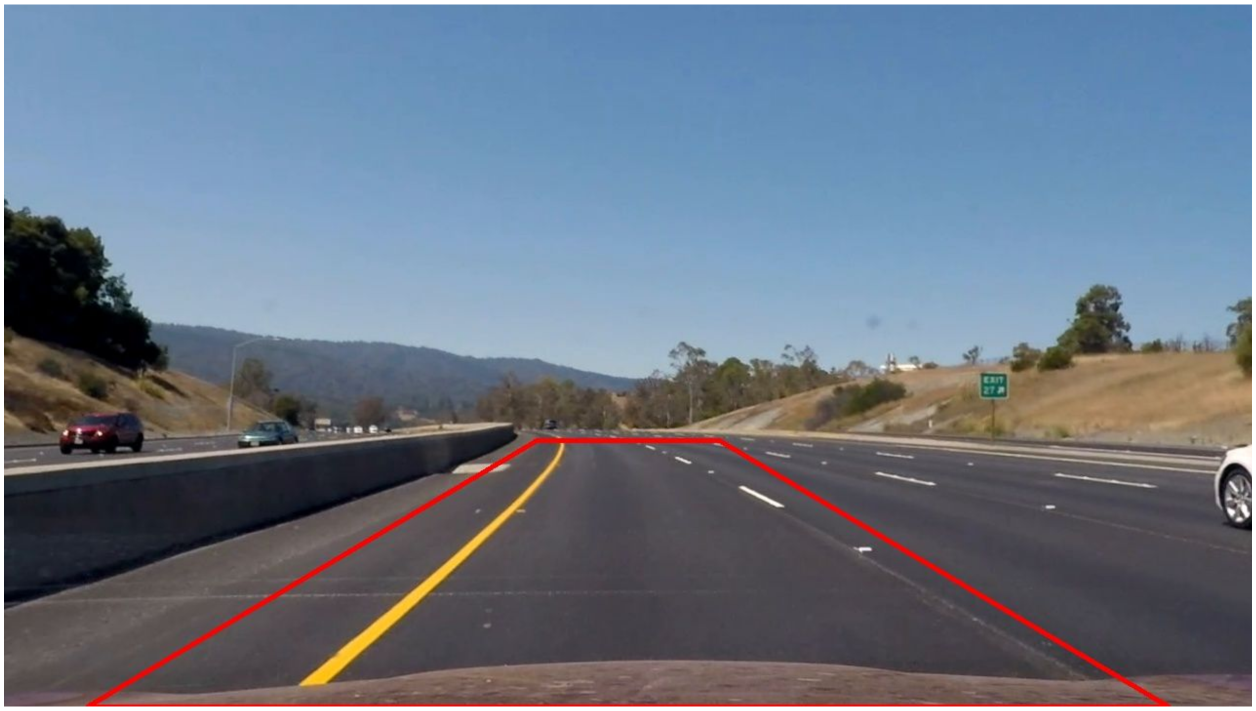
```
    """ dst=
    [[    0.   720.] Corners of Original Image
     [    0.     0.]
     [ 1280.     0.]
     [ 1280.   720.]]
    """
```

This resulted in the following source and destination points:

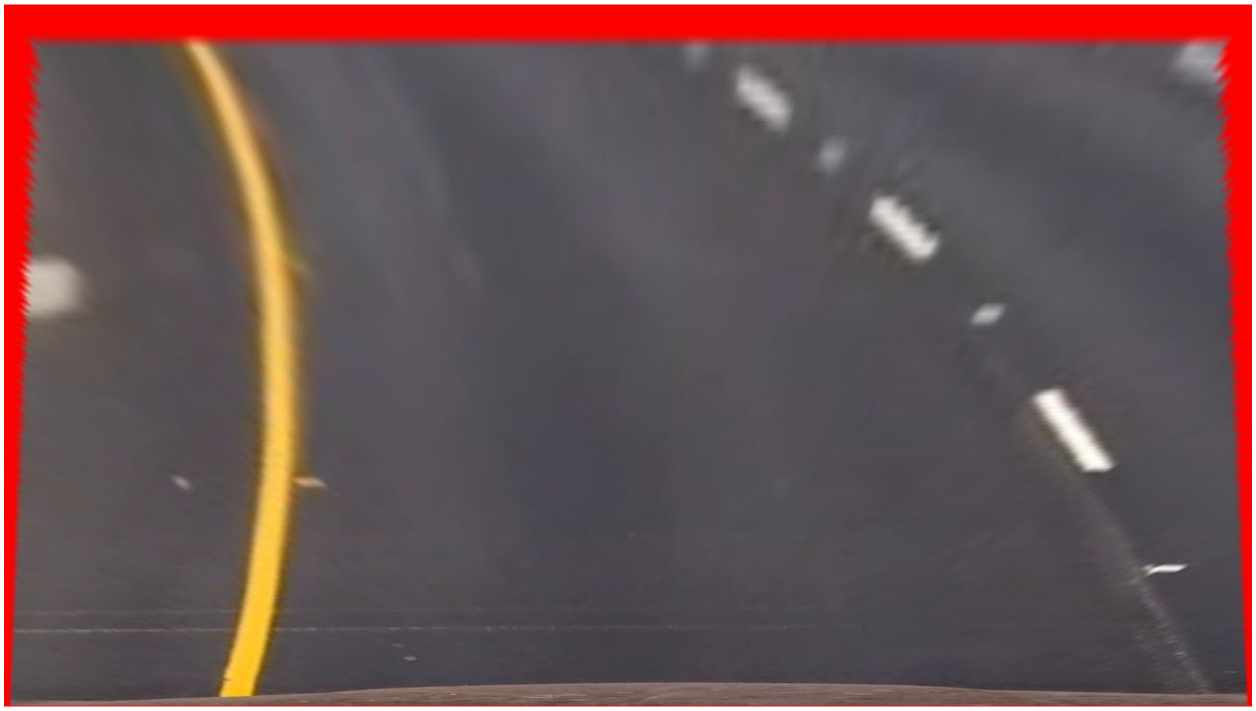
Advanced Lane Finding Project

Udacity SDCND Term 1

Daniel Tobias



The image above is the region that is used to perform the perspective transform.

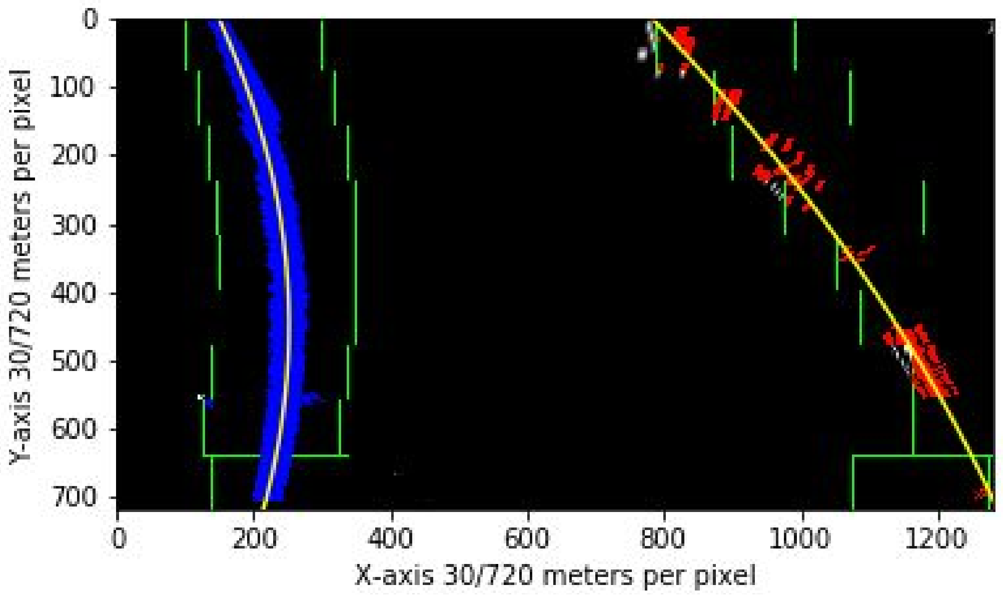


The image above is the result of using the perspective transform operation.

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

#Code referenced here on out comes from inside the function pipeline() in ***code block 8***

The threshed version of the warped lane was used to obtain the information for detecting the lane lines. Histograms were used on the lower half of the image(code block 8, line 36,37) to find appropriate X axis values using the peak values of the histogram on each side. The peak values of the histogram on each side were use to get a strong X axis values that that could be interpreted as where the lane started. Sliding windows were created around these values to tolerate the change in the lane lines and the movement of the car so that the search area was not too rigid. Using the area enclosed in the window, (code block 8, line 93,94) second order polynomial was used to fit the left and right lane pixels separately.

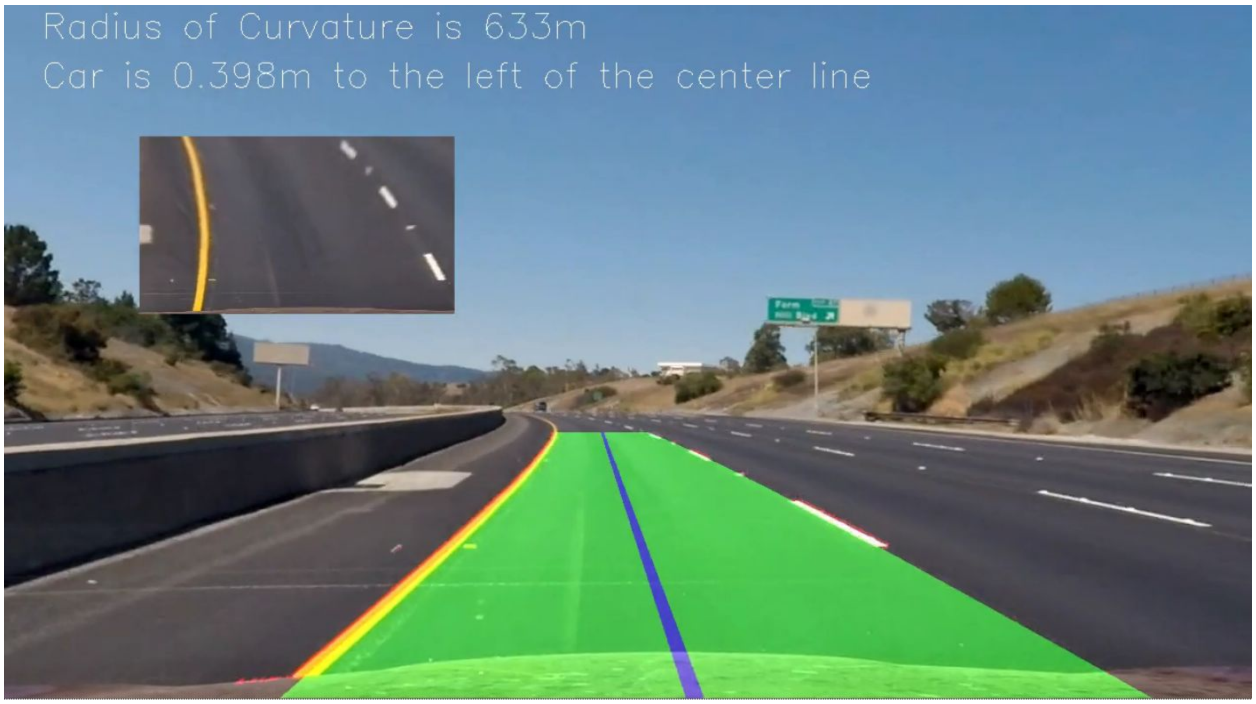


5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

I calculate the curvature with respect to pixels from code block 8 line 108,109 and the curvature in meters at line 129,130. I used a rolling mean to attempt to smooth the micro jerkiness of the video, another approach I tried was exponential smoothing which worked as well. Line 125,126 are where the position of the vehicle relative to the center of the lane was calculated. I illustrate this using a **blue line** that connects the base of the image and the top most value of the center of the lane which you can see in the image below.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.

Lines 139 through 158 in the eighth code block contain the perspective transformation and helping code in the function pipeline. Specifically the Inversed warpPerspective() functions happens on line 156. I also inserted a bird's-eye view of the lane to show the perspective transform.


Radius of Curvature is 633m
Car is 0.398m to the left of the center line

Bottom of blue line is tied to midpoint of video, top is tied to topmost center point.

Pipeline (video)

1. Provide a link to your final video output.

Video of binary threshold bird's-eye view of lane region
https://youtu.be/K33zpAOEDCQ

Final Video of lane detection
https://youtu.be/nYm-f4Y7jx4

Advanced Lane Finding Project
Udacity SDCND Term 1
Daniel Tobias

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 didn’t encounter that many problems with this project, most of problems were related to matplotlib not working as intended. Most of the challenge was also writing detail to the image and inserting images within an image by pixel values got tricky.

This pipeline will most likely fail in different lighting conditions, and weather conditions. For example the glare of headlights from the car that is driving or cars from the opposite direction, may appear as significant lines and may get pass the threshold pipeline. Any change of camera angle would require the warp region to be recalibrated. If a car pulled in front of the car with the camera it would be interesting to see how the pipeline would behave. Hitting bumps at high speeds would significantly alter the desired effects of the pipeline.

A calibration step could be added before the pipeline started and periodically to adjust for the sun, which would take into factors like amount of daylight/brightness. Also chopping off the top third of the image would not harm the pipeline and would save valuable computation time, instead of thresholding the entire image.