SDC Nanodegree Project 2

Alejandro Terrazas, PhD

May 7, 2019

Overview

The purpose of Project 2 is to teach the student about camera calibration, perspective transformations (specifically, the "Birdseye View" perspective), color edge and Sobel gradient edge detection, and to up the ante on the pipeline, introducing more of the steps that would occur in a production system. A series of notebooks are provided; however, all_together.ipynb has the entire pipeline n it. The directory all_together_output contains the video outputs

The intermediate outputs from each stage of the pipeline are stored in several directories, each ending with the suffix _test_images. For example, there is calibrated_camera_test_images which holds the provided .mp4 videos and .jpgs after camera calibration. Likewise, edge_test_images holds the results of the edge detections (mp4 and jpg files). The folder birdseye_view_test_images stores the mp4 and jpg files for raw image perspective transformation; however, these images were not used because transformation after edge detection appears to work best. Instead, birdseye_edge_test_images holds the transformed edge images that were used for testing the polynomial line estimates of the lane detection.

Because the pipeline was tested and developed in steps, several Jupyter notebooks are present (ending in .ipynb.) The important notebooks are named: calculate_camera_distortion, calibration_and_perspective, line_detection and polynomial fitting. The processes used in some of these steps are quite slow and, therefore, the pipeline was built in pieces using intermediate storing of results from each step.

Camera Calibration

All cameras have distortions due to their lenses and hardware implementation. Because the pipeline needs to produce real-world coordinates from these images, it is necessary to de-distort the images before making measurements. In most systems, the calibration matrix must be estimated from known objects placed in known locations relative to the camera. The most common used known object is the planar chessboard image.

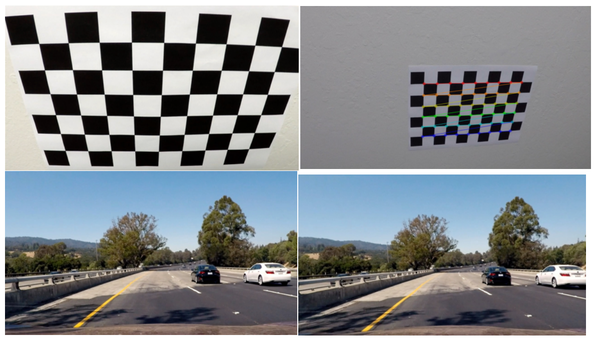


Figure 1: Chessboard images and de-distortion results. *A) Upper left: chessboard Prior to calibration. The image has noticeable bulging and distortion of the squares. B) Upper right: By detecting the edges of the central 6-rows by 8-column center part of the image, and using a grid of generated virtual world coordinates (roughly corresponding the real-world coordinates of the chessboard image), a transformation matrix can be obtained. In practice 10-20 or more varied views of the chessboard are used. Twenty images supplied with the course materials were used for Project 2. Lower left: corrected image, Once a matrix has been obtained, it*

can be tested on new images to see the correction. The matrix can be stored permanently until the camera needs re-calibration.

A series of chessboard images were used to compute an affine transformation to \\. The camera calibration is only run ***once per camera***. The code for camera calibration is stored in camera.pickle. Using pickle, the camera matrix ad the distortion matrix are stored in the main directory. This file is loaded once at the start of the pipeline.

The difference image shows where the distortion were corrected. Interestingly, there are several nice edges that could be used in future edge detection. The difference images are interesting to look at. Videos of the difference were stored under ./calibrated_test_images/difference/

Birds-eye View Perspective

In order to transform the images to birds eye view, a rhombus is selected in the source image (image to be transformed) . Early is the development of the pipeline, the camera images were converted to birds eye view; however, these transformed images were not useful in the edge detection step. The resultant images are of lower resolution (although they are interpolated into a full-size (720,1280) image. The edges of the transformed (after camera correction) raw images were poor, at best. Color segmentation would be useful on the transformed raw images; however, it was decided that transforming the edge images is a better approach (that is edge detection->transformation works better than transformation->edge detection). **Figure 2** shows a raw image converted to birds -eye view perspective of an edge image.

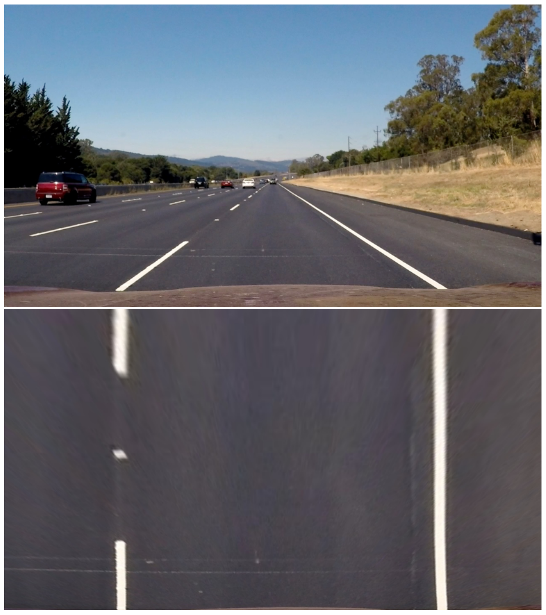


Figure 2. Birdseye view image after perspective transformation. *A trapezoidal area was determined in the original (top) image by successively cropping the image until clear lane elements fit cleanly. A second set of coordinates was determined or the target output. All images were first de-distorted with the camera calibration determined earlier in the process. The trapezoid coordinates were stored for later use on all images acquired with the camera (again, after distortion correction). The birds eye view on the lower panel matches nicely in the x direction with the origina,l untranformed image. The birds eye view makes certain processes easier, especially calculations involving curvature and measurement; however, edge detection is more difficult in the transformed space due to blurring of the edges. Color segmentation may work sufficiently in the transformed space.*

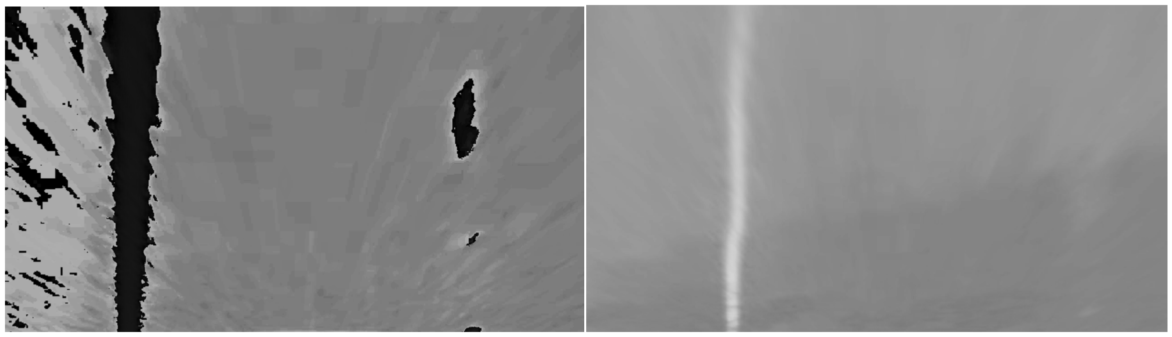


Figure 3. Interesting color channels. *The image on the left is the HLS H channel and was sensitive to everything the lane markings. The image on the right is the L channel from the LUV color space. A table was made of many different channels and it was determined that pure yellow and white color thresholding in the HSV space produced the best output. The success of different color channels was highly image dependent. Light conditions, shadows, surface type, all mattered greatly. The combination of edge type and color space channel, along with thresholding overcame many of the challenges.*

Canny and Sobel edge detection works well on many images; however, the use of different color spaces can overcome some of the challenges that occur in different lighting conditions. It is clear from the videos and test images supplied that real-world challenges are--not surprisingly—a major challenge.

To examine the different color channels, a script was written that output at least 10 different color channels from different color spaces. For example, it is well known that the luminance channel is strong for yellow (check this). Thresholding for a wide yellow band is an effective method for the test images. To that end, a color mask was also used in the "mix" of edge images.

Summing and Thresholding Edge Images

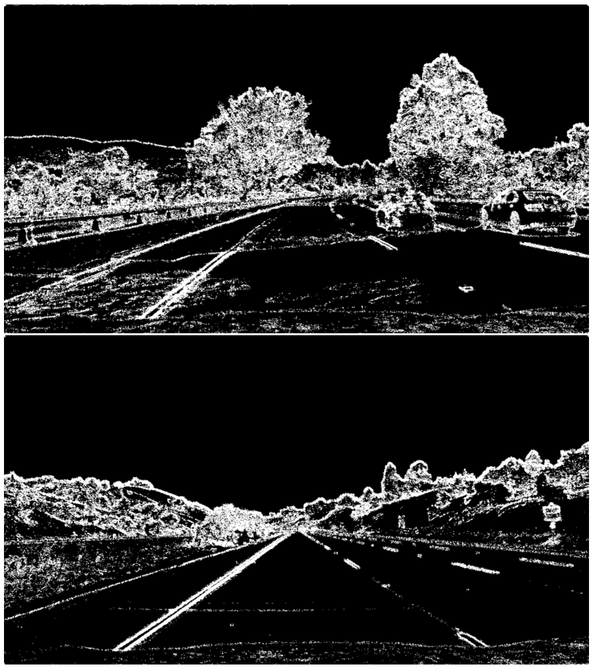
While the lesson encouraged a combination of strategies for or' ing and and' ing the edge images, in this case a simple summing of edge images was used. For each edge detection method, a variety of thresholds and other parameters were used based on experience from the previous project. The best settings and color channels were determined through examination of a large number of output images. It was determined that 5-10 individual color space channels were useful.

In the videos, the segmentation and edges required different settings than for the test images. Because each step was carried out in sequence, it is possible that the saving of the edge images resulted in loss of information or contraction of the range available. Regardless, the edge detection worked especially well on the videos. In addition to the color edge detection, some morphological operations (i.e., dilation and erosion) improve the resultant edges. Finally, because the edge were summed, threshold settings can effect what edges "get through" to the final edge image. Experimentation revealed that the threshold giving less than 10% activated pixels was best.

Figure 4, below, shows some example still images. These edge images were then transformed to the birds eye perspective for line fitting and estimation (see **Figure 5**.)

Among the most useful colorspaces were: XYZ, HLS, HSV, gray, and YUC. All provided good edges in several channels (e.g, Hue in HLS). These color bands were then output as gray scale images (and mp4 files, in the case of video).

Figure 4, Example edge images after summing multiple edge types over five color channels. *Different edge computations were used including Sobel magnitude, Sobel directional gradients, and Canny edges. Each of these edge computations were performed on 5-10 different color channels (HSV channel H, YUC channel Y, etc.) and color mask (tuned to yellow, white). The HLS H channel was active everywhere the lane wasn't (i.e., it is a negative indicator) and was, thus, inverted. After summing, the imaged was thresholded and ½ maximum value to obtain the edge images. While the results were sometimes impressive, the computational costs are high. More research may indicate a more efficient way. For example, adaptive color thresholding.*



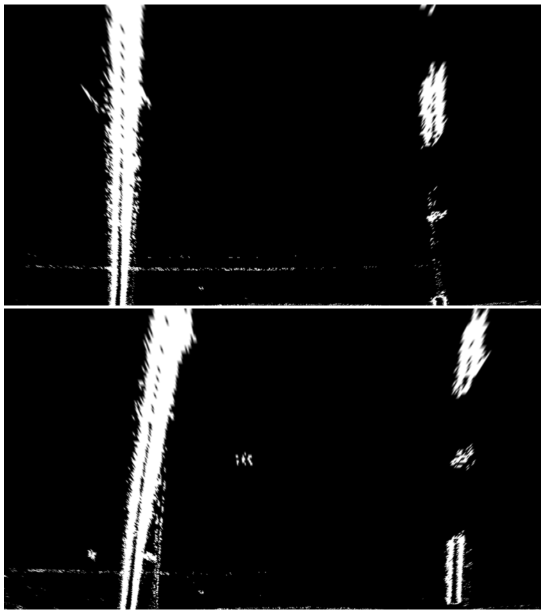


Figure 5. Edge images after color segmentation and perspective transformation to the birdseye view. *The top image shows two straight lines which in the untransformed image converged at the horizon. The bottom image was from a curved segment of road. Edges become distorted slightly after transformation; however, these data are sufficient to extract and fit lines. The right and left lines appear to have the similar curvature, which can be used to smooth outliers.*

Finding the Lanes Using Histograms

Instead of using a fixed-sized square that moves up along the image as suggested in the lesson, a divide and conquer approach was attempted. The birds-eye-view perspective edge image (after camera calibration) was segmented in stripes across the image in sequential steps. First, a global histogram of edges was created summing across the vertical dimension of the image (e.g., a 720 high by 1280 width image produced a 1280 element histogram where each element represents the sum of that column. On the second iteration, two stripes were used (one for the top half and one for the bottom half). This yielded three total stripes, each with a different center, thereby yielding 3 points for creating a polynomial line (see below, Polynomial Fitting). The points in this example (assuming at 720 pixel-high image) are 360 for iteration 1 and, 540 and 180 for iteration 2. Likewise, a third iteration of 3 or 4 was used.

Using Peak Parameters to Create a Stopping Rule

Although, not yet identified, a stopping rule would produce better results and eliminate unnecessary fitting of the polygon. Various information measures were used to see when the distribution of pixels changed within a stripe. For example, if the distribution of a stripe doesn't match well with the overall distribution observed for the image as a whole, low confidence could be factored into that stripes imipact on the lane detection as a whole. The idea is to introduce information theory to the lane detection. Cross entropy (Shannon) and the Earth Movers Distance (EMD or Wassertstein Distance) were used, There was not enough time to do a comprehensive study; however, the method has promise. Determining where to scrap a sample due to high entropy may eliminate unusable edge data.

Fitting the Polynomial

As noted, three points for the left side and three points for the right side were generated for each stripe. Each group of three points represents a tuple of size three containing the FWHM point before the peak and the FWHM point after the peak and the peak itself. The best center line was achieved by averaging these three values. The average of the three values was used along with the y coordinate (the center point of the strip) as a single x, y coordinate for line estimation.

In the case of the global mean peak, there was only a single y value (the center of the image) and x value (mean peak); therefore, the polynomial was not useful. However, the polynomial was fit after the 2 iterations corresponding the 3 x,y pairs (the first single global pair along with coordinate pairs determined in the upper and lower halves of the image). By the 3rd iteration (7 x,y pairs), the line converged well with most of the test images. More constraints on the slopes of the lines should help. For example, averaging the left- and right- side line fits should reduce the effects of outliers on one side or the other.

Stripe Histograms and Peak Picking

The 1280 element histogram was smoothed with a Savgol filter to produce an example like Figure 3. This can be seen the notebook titled xx. The smoothed 1280 element histogram was then split into right and left halves.

The overall peak of the smoother histogram was obtained separately for right and left halves. The peak was then used to then used to compute the full-width at half maximum (FWHM) was determined. The nearest FWHM prior to the peak and after the peak was then determined. This produced left three points for each stripe. Figure 6 shows example of the process.

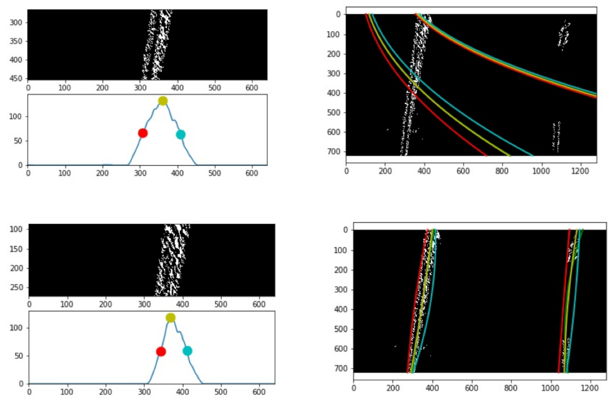


Figure 6. Examples of histogram line finding. *Upper Left side: First iteration with the stripe covering top half of the image. The image on top shows the birds-eye perspective edge image (described in a) for the left side lane. Immediately below the image is the histogram created by summing the pixels I each column of the image. The histogram is smoothed in order to facilitate peak identification. The yellow dot represents the peak. The red and cyan dots correspond to the full width at half max (FWHM) values before and after the peak. The points closely align with the lane. The plot on the lower left a different stripe from the lower half. This stripe is from the second iteration and covers one quarter of the full image height. Again, clear peaks were obtained in this admittedly easy example. Note how the width of the peak is wider in the upper plot compared to the lower plot. The spread of this peak indicates both the amount of noise in the edge image (in this case, low) but also some information about the type of line (in this case, sloping).*

Using a Convex Hull Mask to Reduce Rendering and "Jumpiness" of the Line Fits

In the spirit of "the least expensive pixel to render is the one you don't render at all.", an effort was made to eliminate rendering and fitting for invalid data. One novel idea was to create a mask from the polylines for each lane. This mask was compared to the incoming edge image and if greater than 80% of the activated edge pixels fall under the mask, to simply skip the frame. This results in far smoother line estimates. Some tricks were required to keep the system going if it stops sampling due to the mask. If no line has been drawn in 40 frames, the mask is set to zeros and the lines are reset.

Using Historical Line Averaging to Further Improve Line Estimation

A buffer of length 20 was used for averaging the polyfit line parameters. This improve line performance significant and prevent the lines from "chasing" outliers.

Keeping Track of Lane Properties Using OOP

It soon became clear that tracking the results of all of these objects created unreadable code; therefore, some object-oriented programming elements were used. Specifically, the Lane Class was introduced. This Class produced an encapsulated object that could be queried for the main elements (mean peak, FWHM prior to peak, FWHM post peak and the peak value itself) that are needed to draw the lanes. Other parameters contained within the class are the histogram and offset (used mainly for the right lane).

Meaning of Curvature: Do the Numbers Make Sense

The function measure_curvature_pixels returns number in the range of [1000:60,000], (500:2000] for a relatively straight line. An intuitive understanding of the meaning of these numbers requires converting the pixels to real world coordinates. 720 pixels is the height of the image. With a perfectly straight line, 720 pixels would the farthest distance represented in the birds eye view. As the lines become curved, the number of pixels is increased indicating that more travel is required to complete the line segment. A tight circle would run from 0,0 to 720, 720 and would return values of 1,600,000 pixels. Converting pixels to real world value requires understanding the real-world distance of the image plane; however, assuming 120 meters results in 1.3k.

How to Improve of the Pipeline

The edge detection process is unwieldy and depends on too many image channels and edge detection. Of course, consistent edge detection in real world images is challenging and hyperparameters tuned on one set of images rarely work as well as they do on the training images. Histogram equalization was one idea for improving the edge detections; however, only a few channels were improved by the technique and yet some of the others deteriorated. There are some hyperparameters that can be tuned to improve edge detection. Further study can decide which channels benefit the most from histogram equalization. Moreover, more sophisticated approaches are used in practice.

One of the more promising avenues tried in this project is the adaptive horizontal binning based on entropy and earth mover distance. Some lines are very straight and bold, and, thus, can be solved easily with a few large stripes and a low order polynomial. There histograms line up nicely and are sharper than curved images. Thus, the initial few stripes can be used to adapt the sample to throw out uncertain peaks.

Performance on the Test Videos

The line detection worked quite well on the project video and the first challenge video. The harder challenge video was indeed harder but much of the challenges were from missing edge data. A wider field of view might improve performance.