May 10, 2019

Advanced Lane Finding

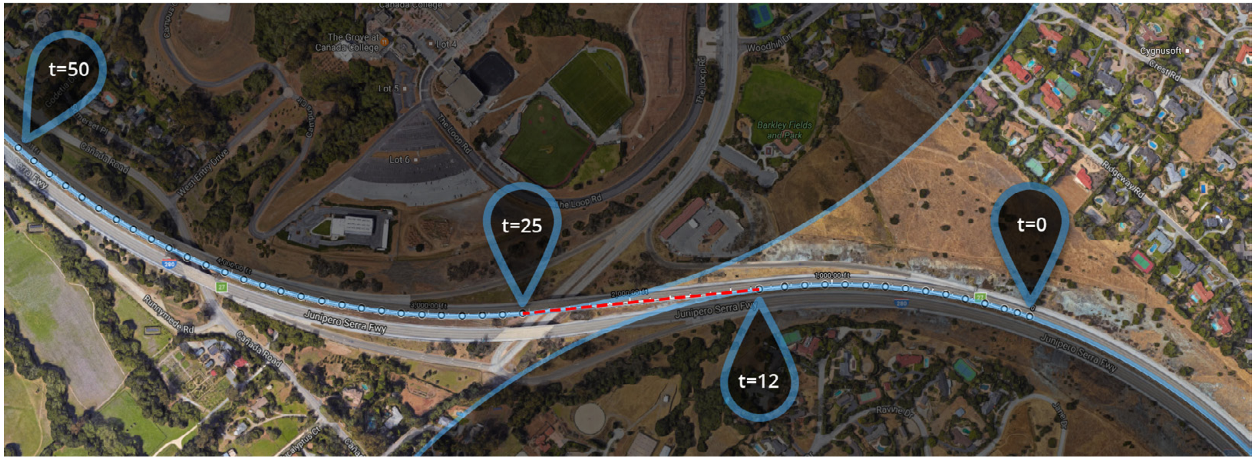
Proximity Algorithm research by James W. Dunn

Goal: *Calibrate a camera lens and annotate a video sequence with identified lane edges, curvature radius, and center drift.*

Contents

Analysis

The supplied project video contains 50 seconds of northbound footage through the S-curves along the Junipero Serra Freeway near Woodside Glens.



I-280 from approx GPS 37.443855, -122.255386 to GPS 37.446226, -122.272631, a traveled distance of about 1595.7m.

The video begins just before Exit 27, in a gentle left turn of 914 meter radius followed by a straight section of about 360 meters (indicated by the dashed red line above). The straight terminates at the end of a bridge (over Farm Hill Boulevard) followed by a slow right turn of 1037m radius. Just after the halfway point through this turn is another bridge (over Cañada Road). The video ends as the road straightens out of the turn. Average speed over this distance is 114km/h (or 71.4mph). Although lighting conditions appear to be ideal with high-contrast pavement markings, there are incongruous anomalies near the bridges. These include abrupt road-surface color shifts from dark- to light-gray, sporadic cast shadows from vegetation, and several vehicle/camera-pitch bounces caused by the uneven road surface at the approach slabs.

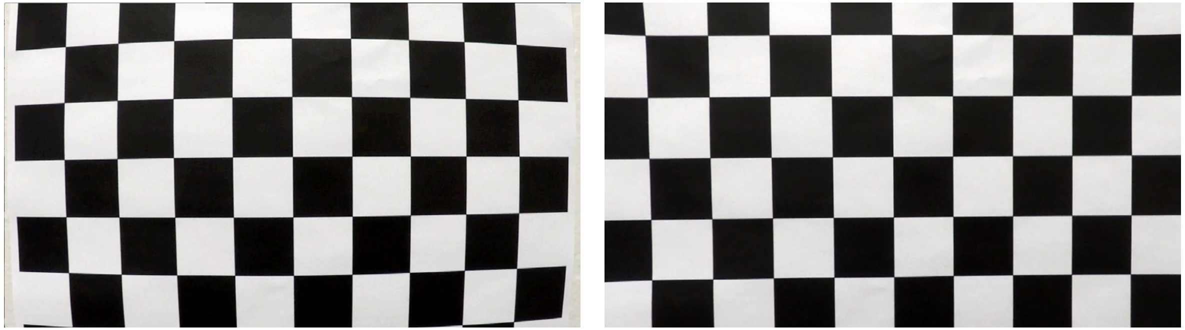
Two challenge videos are also provided. The first is located on route 85NB from the start of a detour onto the median and crossing under the Homestead Road overpass. Distance is approximately 497.3m in 16s.

The advanced challenge video is located on route 84SWB (La Honda Road) from a point after the bend beyond the Fox Hill Rd turn-off, descending through a forest area to a 15mph hairpin bend, and up to a cement structure on the right. Distance is 548.7m in 47s.

Pavement marking measurements and specifications indicate: lanes are approximately 3.64m wide, white lines are about 4m long with 10.64m gaps, reflectors are positioned every 14.64m, and painted lines are 100mm wide. The challenge video lane width is about 3.05m.

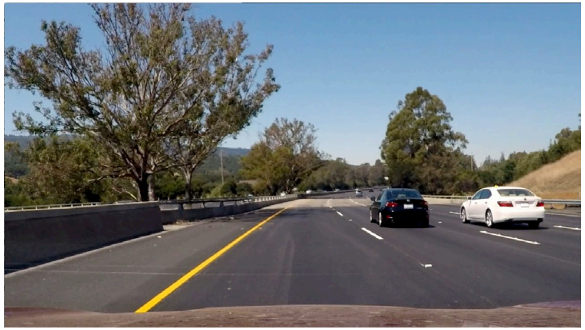
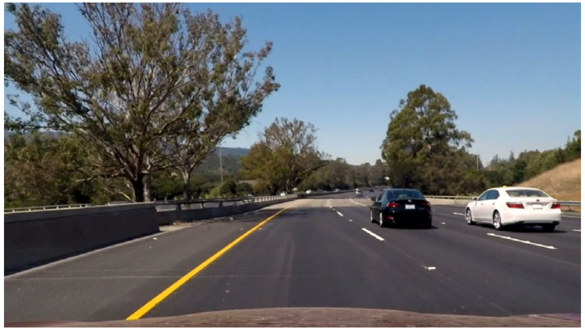
Camera calibration and distortion correction

OpenCV *findChessboardCorners*() is used to detect corners in the calibration images. Then *calibrateCamera*() is called to calculate the distortion coefficients and the camera matrix. This calibration data is saved as a serialized data file. Images 2, 3, 17, and 18 were sufficient for calibration. The code for calculating the distortion matrix of the camera can be found in calibrate.py at line 54. An example is provided below.


Distorted calibration image (left) and corrected image (right).

As there is no EXIF data provided, it must be assumed that the calibration images share the same lens settings as those in the video footage. (Confirmed with Ryan Keenan.)

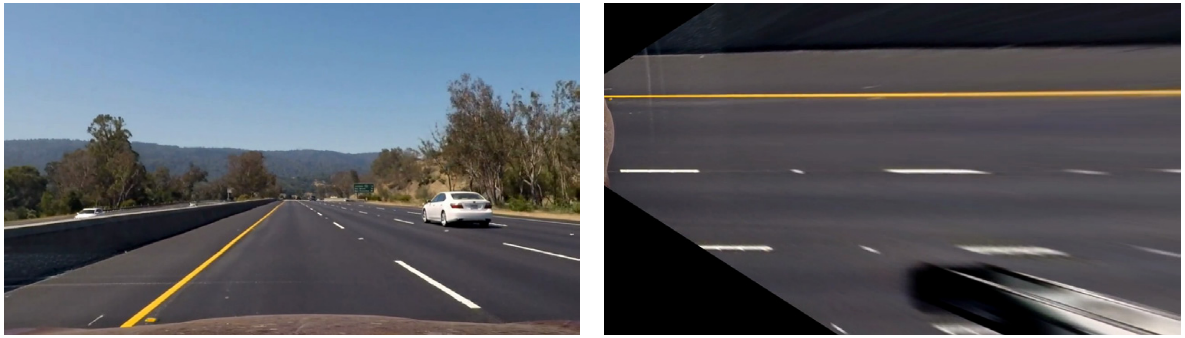
Indeed, to approximate the camera's field of view and focal length, if the corrected video is displayed full-screen on a 24" monitor and viewed closely from a distance of 10 inches, the vehicles in other lanes appear to be correctly shaped. This explains the apparent pincushion distortion when viewed from a greater distance, where the vehicle shapes begin to elongate. (Most notable in the white car in the "corrected" image below.)

Distorted test image (left) and corrected image (right).

Perspective transform

To determine the perspective transform into an overhead view, four corners of a lane were pinpointed on a straight segment of the road. A distance of 29.28m (twice the distance between two reflectors) was selected for the lane depth. Source trapezoidal points (x,y) are (557,460) (557+166,460), (84+1112,670), and (84,670). Corresponding rectangular destination points for the perspective translation and rotation are (1273,156), (1273,393), (25,393), and (25,156).



Calibrated video frame number 409 (left) and corresponding overhead view after perspective transform (right).

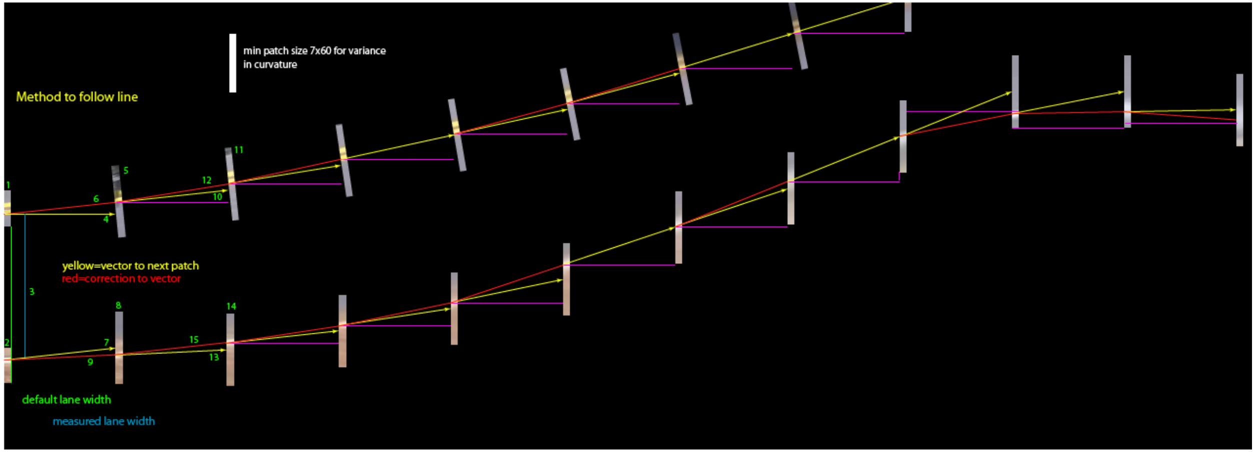
To maintain lane aspect ratio and an equal x and y scale, a “rotated” perspective transform is applied which orients the overhead viewed lanes along the screen x-axis (ascending to the right). The y-axis therefore measures the distance between lane lines.

Operating at a scale of 160 pixels to 3.6576m, the resulting virtual lane measures 160px (rows) by 1248px (cols) which is convenient to work in a standard x-y screen coordinate system and to display horizontally within a 1280x720 landscape image. The function that performs the perspective transform is named *makeOverhead*() and can be found in pipeline.py at line 150.

Method to identify lane lines

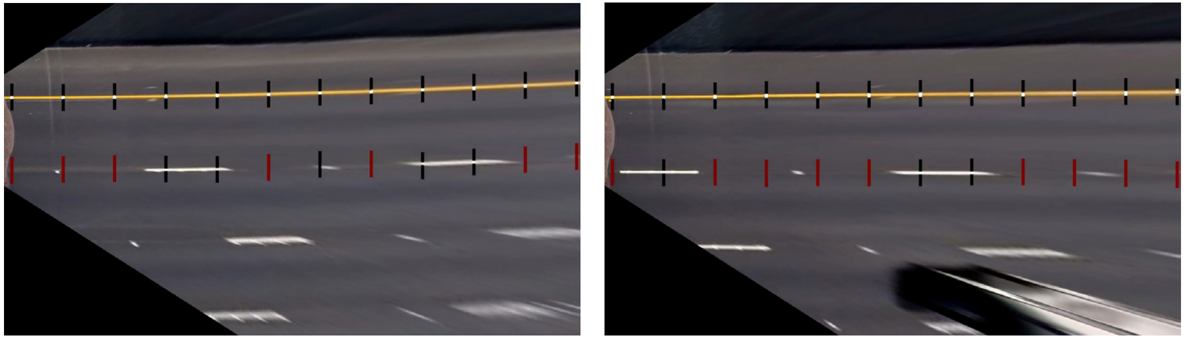
An optimized filter array model is constructed to search for and locate pavement markings. There are 12 filters per lane-line, spaced 114 pixels apart. Each filter measures 58x7 pixels. These filter regions ride the lines from frame to frame like rails.

Four custom Python classes were created to queue data points, manage lane edges, model the lane structure, and provide execution control. The filter array is initialized at a given lane width with respect to camera center (under the assumption of a straight road). As the first frame is processed, rules are applied which “snap” the filter construct onto the left line.



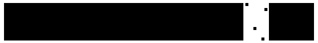
Early planning stage for the filter array construct - overlaying a frame from the advanced challenge video.

In subsequent frames, the search locus of the left lane line is initialized with the findings from the previous frame. The right lane line is initialized at a distance equal to that of the current frame’s left line from the averaged virtual center line. A threshold is applied to each filter with a fallback threshold if no line pixels are detected.

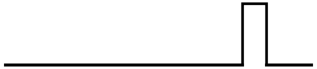


Video frames 98 and 409 with 24 threshold filters applied to create binary images containing likely lane pixels. Red regions indicate a zero-result filter (no line found).

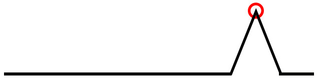
A histogram of each filter's binary image is convolved with a 5-pixel window to create a product area with peak values near the center of the line. [Wikipedia: Convolution]



Filter binary image

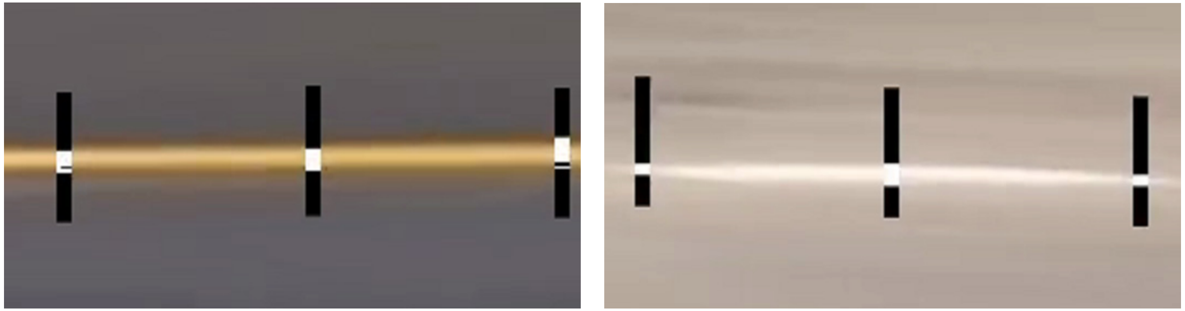


Histogram



Convolution

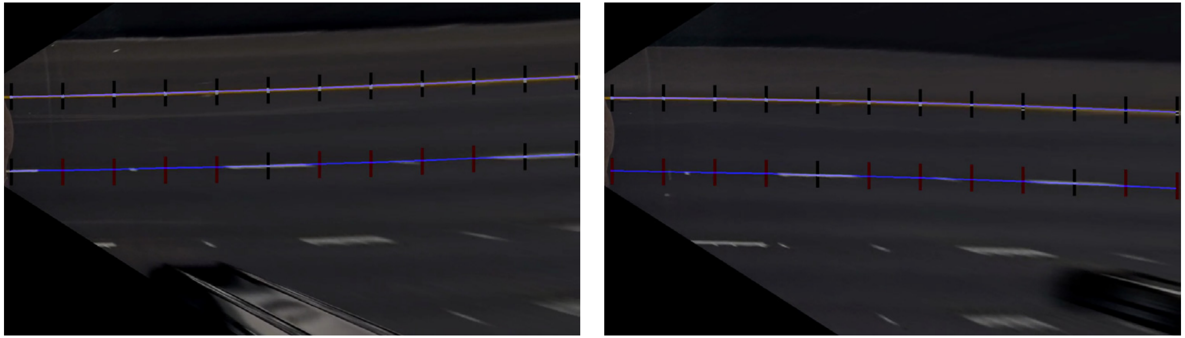
The NumPy *argmax*() function is applied to locate a centroid candidate for the y-value. Lane line pixels are identified by the *locateLines*() function in pipeline.py at line 200, assisted by *processFilter*() at line 185.


Close-up views of 3 binary threshold filters: yellow line on asphalt (left) and white line on cement bridge (right).

Rules are applied to maintain tracking of lines. For the project video, a maximum rate of curvature indicates a deviation of no more than 7 pixels between filters. The cascade effect of this rule adjusts the subsequent filter's y-value into a targeted search region of the next section of the line. While tracking the yellow line, if an HLS threshold yields a zero value, then a color range filter is attempted. Similarly, if a range threshold on the white line fails, the threshold is relaxed slightly. If a filter detects a line candidate beyond a specified distance, then it is considered an outlier. Because there is a 3-to-1 ratio of expected gaps in the white line, it is assumed to be an average distance from the yellow line. Confidence in line detection is determined as a percentage of pixels found on the convolved histogram: line candidates falling below 60% are ignored.

Polynomial fitting of lines

Left and right line data is collected from the filters into a circular queue over three frames. The center line is the midpoint between left and right and is queued for a length of 15 frames. Averages are used to fit second-order polynomials using the NumPy function *polyfit*().



Examples of lane line pixels identified and overplotted with a fitted curved functional form.

The determined coefficients are stored in the **Line** class where a *lineF*() method retrieves y-values given an x-value. Polynomials are fitted by the *drawLane*() function at line 253 in pipeline.py with the assistance of the class method *fitToPoly*() defined at line 80.

Curvature and lane drift

Two methods of curvature calculation were explored: A. the three-point exact solution (the intersection of the perpendicular bisectors of two edges) and B. the second order equation for approximation of radius of curvature:

$$R_{curve} = \frac{(1+(2ax+b)^2\ )^{3/2}}{|2a|}$$

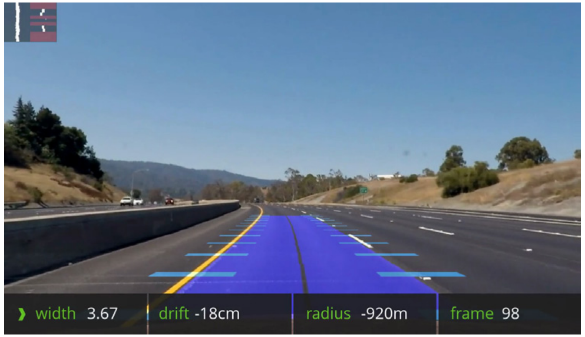
The OpenCV routine *minEnclosingCircle*() implements method A and is given three points along the average center line; the first being 640px behind the vehicle in overhead space, the second at the zero point (representing the present vehicle location), the third at 640 pixels ahead of the vehicle. The two methods were compared in an A/B test; the difference in results for the complete set of video frames was within 2 meters.

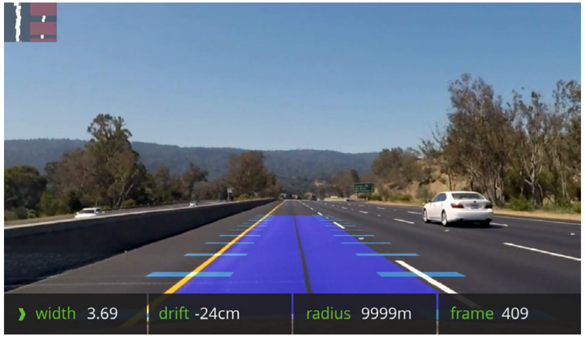
Calculation of the radius is performed using method A in *drawData*() at line 355 in pipeline.py with assistance by function *radius*() defined at line 313.

Lane drift is the difference from the center line y-value at x=0 to camera center (which is a fixed y-value of 275). This calculation can be found at line 367.

Annotation

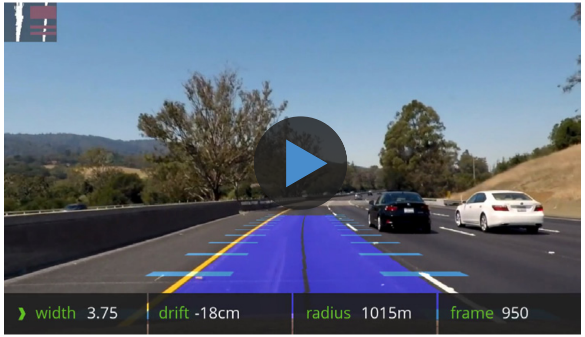
The undistorted video is annotated with the fitted lane displayed in deep blue with a 4 pixel gap representing the center line. The filtered regions are represented along the lane lines with azure blue rectangles. Curvature radius is updated every 20 frames and displayed in meters; left curvature is a negative value. Lane drift is displayed in centimeters; left of center is a negative value. Lane width is calculated frame by frame. A filter diagnostics panel is displayed in the upper left corner; red indicates the filter did not find a line; white indicates pixels which passed the filter threshold and move on to the convolution stage.





Annotated frames 98 (left) and 409 (right) indicating lane boundaries, curvature, and position from center.

The final annotated video can be found in the *output_images* directory or on Vimeo through the link below. Additional videos in the directory include overhead projection, filter tracking, line fitting, and lane painting.



http://vimeo.com/204613721

Discussion

Approach

Several days were invested in the analysis portion of this project to fully understand the geometry of what lay before the camera. Adobe Illustrator was helpful in marking up sample frames and measuring proportions. Of great help was Google Maps for scouting and surveying the driven route. Microsoft Excel was useful for recording measurements and determining scale conversions. Iterative filter exploration came next and eventually a rules-based method was discovered to localize the filtering to a highly-targeted array of regions along the expected path of a road line with a tolerance to accommodate curvature.

Performance

Due to the filtering on narrow regions of interest, the pipeline processes the 50 second video in about 3 minutes on a Xeon W3520 (equivalent to i7 920 first generation) at 2.8GHz with 12GB memory. Further performance improvements could be made by serializing each warped video frame such that subsequent executions can avoid the perspective transform.

Challenges

Shadow areas and low-contrast lines (like those in the challenge video) cause filter noise or zero-results. Additional threshold levels should be identified along with confidence ranking to facilitate cascading fall-through or conditional pre-selection.

High-curvature roads such as those in the advanced challenge require an alternate method of guidance to avoid derailment of the filter structure. The improved method would 1) follow the lane lines using dynamically determined intervals, 2) orient the filters perpendicular to the vector found from the prior region, and 3) optionally integrate GPS map data.

Further Research

A convolutional network model can be trained to identify lines using the filter regions as feature input. The existing threshold system can provide line location values as supervised target data. The trained model could then operate in place of the threshold system.

It may help to track pavement features through multiple frames. This could provide additional data such as particle direction, velocity, line lock, shape continuity, and center pavement signage.

References

Analytical tools

- Microsoft Excel
- Adobe Illustrator
- Google Maps

Curvature

- intmath.com/applications-differentiation/8-radius-curvature.php (see method 1 for an approximation and method 3 for exact)
- en.wikipedia.org/wiki/Smallest-circle_problem

Pavement marking specifications

- dot.ca.gov/trafficops/camutcd/docs/TMChapter6.pdf
- mutcd.fhwa.dot.gov/htm/2003r1/part3/part3a.htm
- What are road markings: youtu.be/_KaHbbVxJWE

Lane Detection

- Lane Detection and Tracking with MATLAB: youtu.be/SFqAAseL_1g
- Lane Detection Vanishing Point Tracking: youtu.be/__g2gppGtnQ

Color detection

- pyimagesearch.com/2014/08/04/opencv-python-color-detection

Convolution

- en.wikipedia.org/wiki/Convolution

Open Source Computer Vision Library

- opencv.org

NumPy Library

- numpy.org