Johannesburg,
South Africa
+27.060.503.2734

Jean-Paul Wilson

GitHub
LinkedIn
jeanpaulwilson@gmail.com

1 Abstract

The following project demonstrates the implementation of a computer vision approach to lane line detection that is intended to be robust under varying lighting, lane color and road surface conditions.

The result is video footage with the detected lanes superimposed onto the footage, along with vehicle positioning information.

2 Methodology

The solution generated allows for the prediction of a vehicle's position relative to the lane lines, and the amount of curvature of the lanes on the road ahead. The process for carrying this out can be divided into the following main steps:

1. Camera Lense Calibration and Distortion Correction.
2. Edge Detection with Color and Gradient Thresholding.
3. Perspective Transformation.
4. Determination of Lane Lines.
5. Measure road curvature and Vehicle Location.

Assumptions: Although the techniques implemented in this notebook may be useful for autonomous driving and driver assistance programs, it would not suffice on its own. One of the reasons for this is that several simplifying assumptions have been made, which are not applicable in the real world:

- The road ahead of the vehicle is assumed to be flat, which is often not the case.
- The location of the video recorder (dashcam) is assumed to be in the centre of the windshield of the car.
- The height of the video recorder is assumed to be the same for all vehicles that would make use of it.

2.1 Calibration and Distortion Correction

Camera lenses distort the images (radially because of their curvature on the edges, and tangentially when the plane of recording is not parallel to the image) warping their dimensions in the recorded image and video. This is potentially disastrous for autonomous driving, where an accurate interpretation of the vehicle's surroundings is paramount to safe operation. Therefore, in order to correct for the distortions, camera calibration is essential. Chessboard images are the most useful for this because it is simple to locate the corners in grayscaled images of the chessboards. Several cv2 methods are useful for the calibration process. These methods (with their inputs and outputs where applicable) include:

- ret, corners = **cv2.findChessboardCorners**(img, pattern_size)
- if ret; draw corners, else red circle: **cv2.drawChessboardCorners**(img, pattern_size, corners, ret)
- ret, mtx, dist, rvecs, tvecs = **cv2.calibrateCamera**(objpoints, imgpoints, img_shape, None, None)
- undist = **cv2.undistort**(img, mtx, dist, None, mtx)

The calibration steps are as follows:¶

1. Create an array of object points (which will be the same for each of the images used in the calibration process).
2. Create an array of image points, from each of the images used in the calibration process.
3. Read in the images, finding their corners, and appending them to the object point and image point arrays.
4. Calibrate the images using the arrays from 3 above.
5. Transform (undistort) and then return the required image(s) for the next step in the lane finding process.

2.2 Edge Detection

Two types of thresholding are done for the purposes of locating lane lines; color and gradient. For best results in varying lighting conditions and for different color lane lines, the two thresholding techniques are combined.

2.2.1 Gradient Thresholding

The Sobel operator is a method of edge detection which takes the derivative of a matrix of an image's grayscaled pixel values in either the x or y direction.

- The sobel operator is a square matrix of odd dimensions (3, 5, 7, etc). Taking the Sobel of an image with uniform pixel values returns a value of zero.
- Sobel in the x direction detects predominantly vertical edges while conversely, Sobel - y detects horizontal images.

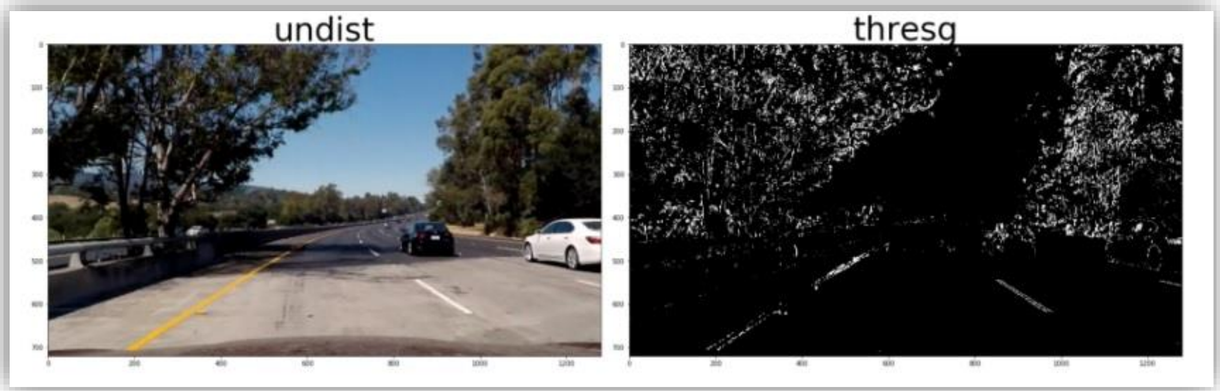
- As lane lines are generally oriented in 45 degree orientation, a combination of both Sobel operators is necessary.
- Three methods of detection are implemented and combined, with tuning of the thresholds required for optimum results:
    - Absolute value thresholding of Sobel x or Sobel y
    - Gradient Magnitude
    - Gradient Direction

2.2.2 Color Thresholding

There are several ways to categorize colors and represent them in digital images. These representations are called color spaces. Three of the most common color spaces are:

- RGB: Red, Green, Blue
- HSV: Hue, Saturation, Value
- HLS: Hue, Lightness, Saturation

After experimenting with the three color spaces, separating their channels and testing on different images of roads with the lane lines in different lighting conditions and of different colors, it was found that **S**aturation in the HLS color space performs best overall for color thresholding, even though it is not the best in every situation. (For example, the Red channel of RGB still performs best for white lines, under constant, bright lighting (no shadows).



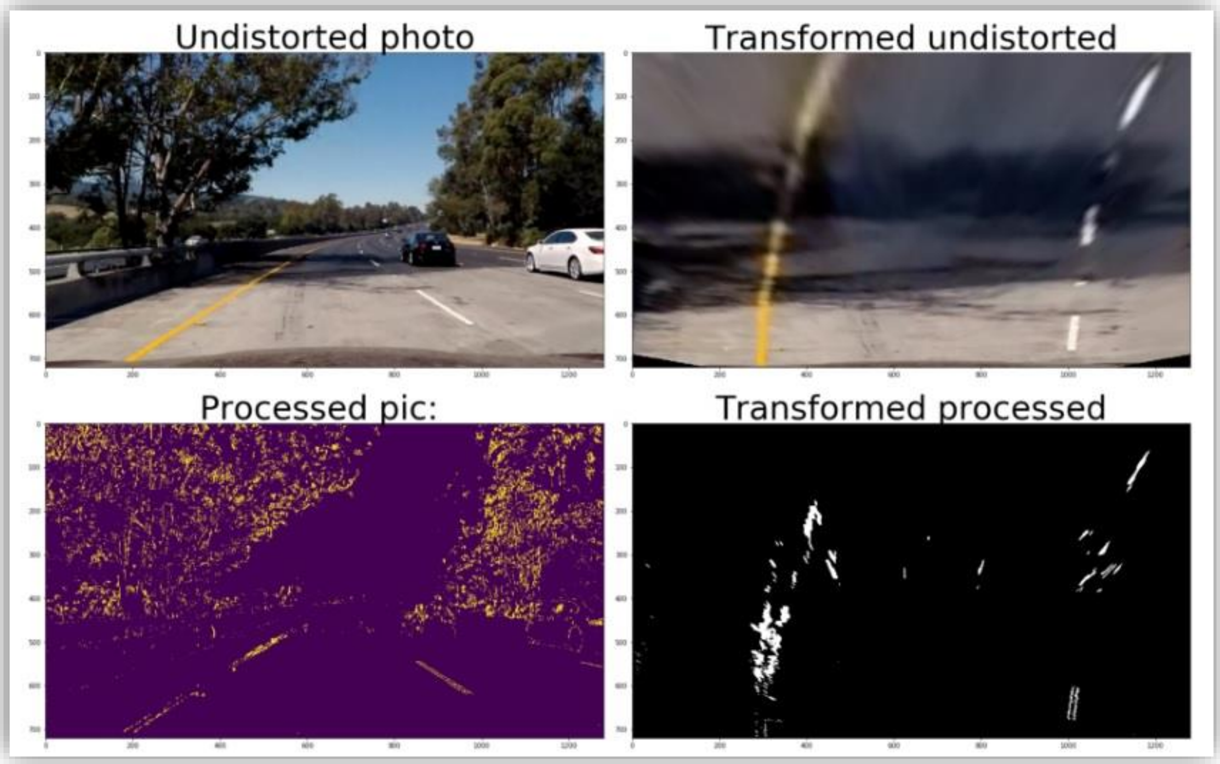
2.3 Perspective Transform

Once the lines have been located to a suitable level of accuracy (in this case by combining gradient thresholding with color thresholding), the image requires a tranformation of perspective in order to detect the **actual curvature** of the road. The following describes how to change from the perspective of the front dash of the car looking out towards the road ahead to a top-down, or bird's eye view perspective.

Johannesburg,
South Africa
+27.060.503.2734

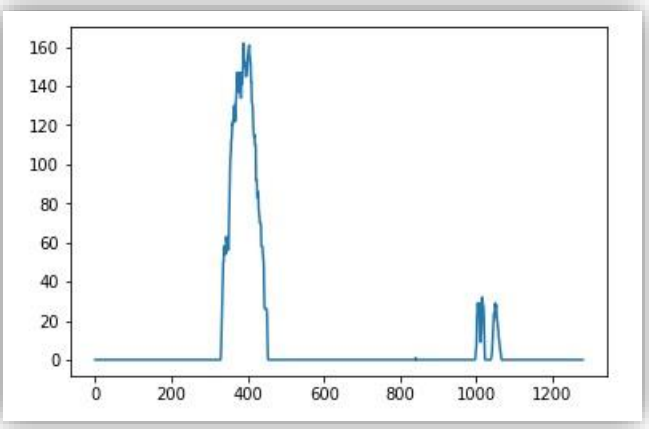
Jean-Paul Wilson

GitHub
LinkedIn
jeanpaulwilson@gmail.com



2.4 Lane Detection and Projection

Now that the pixels for the lane lines are in a useful format, the final processing steps can take place. After a histogram shows the starting point of the lane lines at the bottom of the image, a 'sliding window' approach is employed to locate the area in the frame when the lane line pixels are concentrated. Once this has been located, the pixels can be colored to highlight the left and right lines, and a trendline can be plotted.

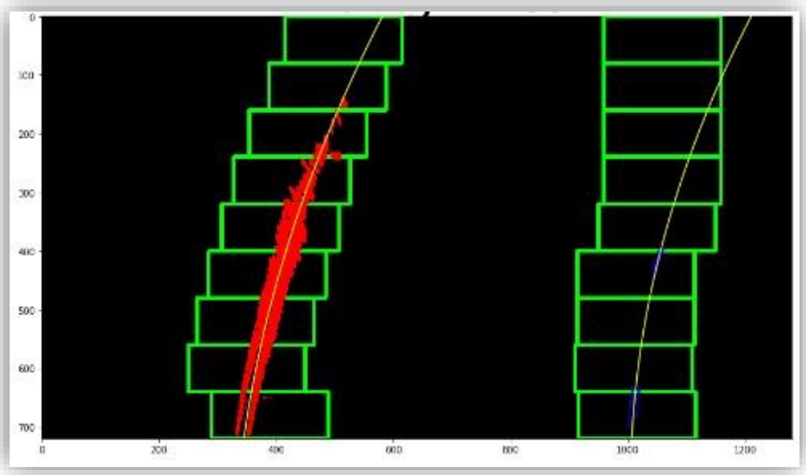


These trendlines then have their curvature calculated in order to determine the curvature of the road. The location of the vehicle with respect to the center of the lane is also calculated.

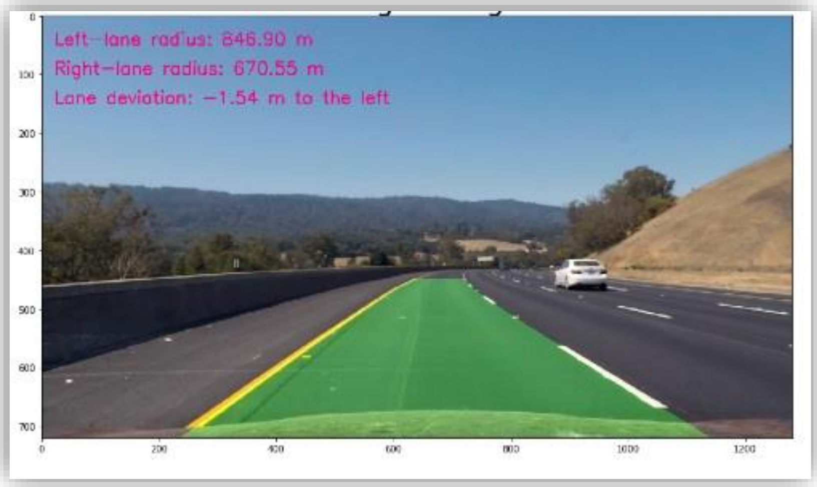
Johannesburg,
South Africa
+27.060.503.2734

Jean-Paul Wilson

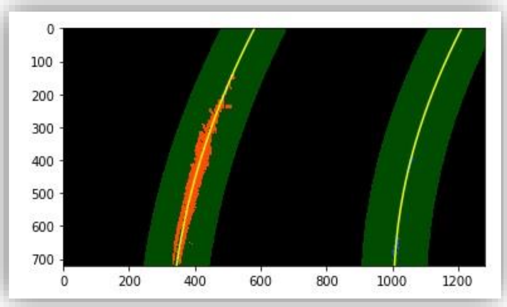
GitHub
LinkedIn
jeanpaulwilson@gmail.com



Once the lane lines are located, the lines are transformed back to the original perspective, and they, along with a polygon shading the area inside the lane lines, and the vehicle position and lane curvature information, is superimposed on the original image frame.


Left-lane radius: 846.90 m
Right-lane radius: 670.55 m
Lane deviation: -1.54 m to the left

In order to save on computing power, once the lane lines have been located, it is no longer necessary to begin the pixel search from scratch using the sliding technique. This is because the location of lines will be similar to that of the previous frames, and so the search can simply be done in the same region.



Johannesburg,
South Africa
+27.060.503.2734

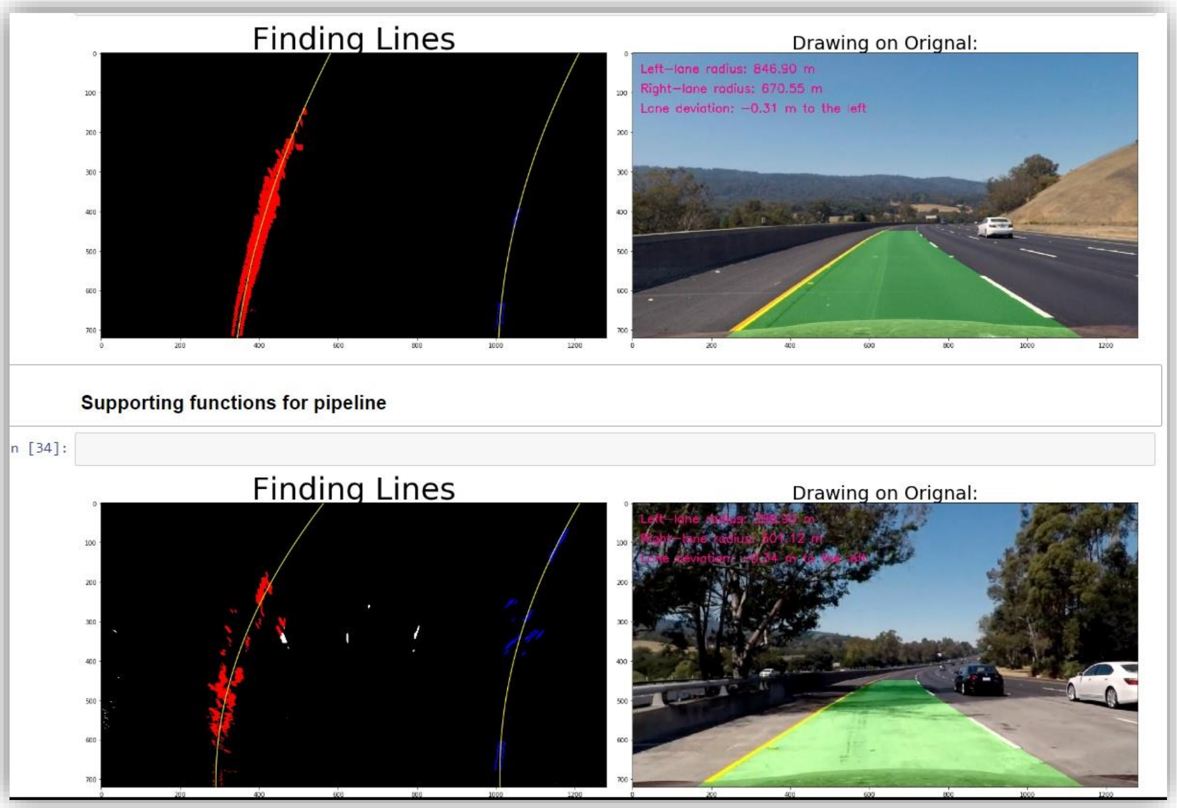
Jean-Paul Wilson

GitHub
LinkedIn
jeanpaulwilson@gmail.com

Now that each step can be done correctly, the code is organised as a pipeline in order to convert a video stream into one that detects lane lines as described above.

It is important to set up a way to smooth the video. That is, in some frames the lane lines might not be detected. There, a way to call on similar, previous lane detection coordinates to superimpose onto these frames is required. A double ended queue (deq) was implemented to store these coordinates.

The output from the pipeline worked as expected, when tested on a range of different frames. A video of this implementation can be found at this link.



```
Left-lane radius: 846.90 m
Right-lane radius: 670.55 m
Lane deviation: -0.31 m to the left
```

```
Left-lane radius: [illegible] m
Right-lane radius: 501.12 m
Lane deviation: -0.34 m to the left
```

3 Discussion

Although the implementation provided is able to detect lane lines, there are several major challenges that I had in completing this project, and many improvements that I plan to implement in the near future.

Challenges

- The detection of the lane lines took a very long time. I experimented with various combinations of sobel kernel sizes, trying to draw the line between eliminating noise and accurately detecting lane lines. However, every aspect of the thresholding I found difficult to master, especially when in combination with the others. However, at this point I feel far more experienced and will not face such a steep learning curve when applying these techniques on later projects that I undertake.

Johannesburg,
South Africa
+27.060.503.2734

Jean-Paul Wilson

GitHub
LinkedIn
jeanpaulwilson@gmail.com

- The pipeline for video was the largest challenge for me and took 2 weeks longer than I had initially estimated. I was not certain about whether the deq I had implemented was working, and whether the co-ordinates I was extracting were correctly being done so.

Improvements

- Further smoothing of the pipeline will be useful
- Refinement of the edge detection will help a lot
- I will record my own video and test this implementation on it to see how generalized my approach has been.