Udacity Self-Driving Car Nanodegree
Project 4: Advanced Lane Lines

Mark Helms, February 11, 2018

Goal/Summary

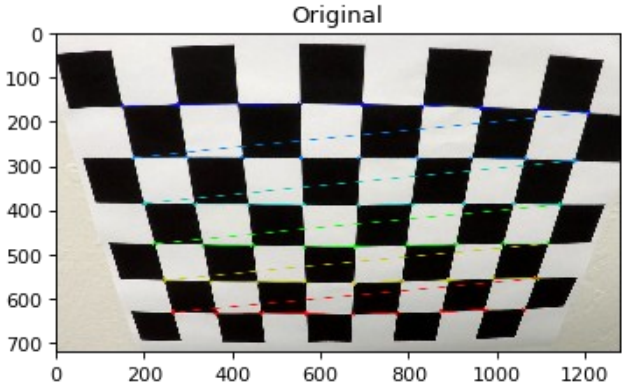
Identify lane lines in a video from a center-mounted, front-facing car camera using colorspaces and/or gradients and overlay the detected lane, information about the curvature of the road, and offset from lane center. Success is judged on whether or not all overlays are correctly performed, including under sub-optimal conditions, such as shadows. This has been achieved primarily through the use of camera calibration and two colorspace conversions, along with some mathematical formulas. In addition, the stand-out task of completing the challenge task was performed, however the "harder challenge" video was not successfully completed. This report summarizes the activity involved with the project.

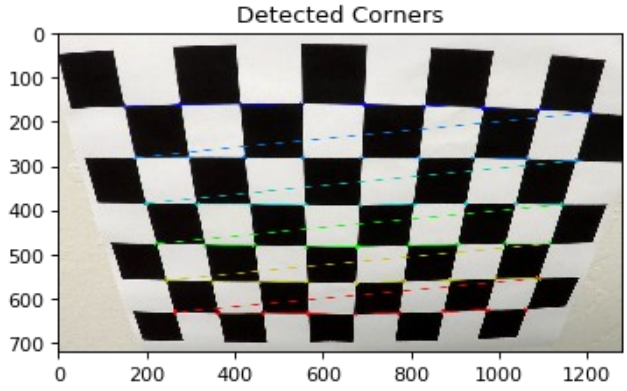
Approach

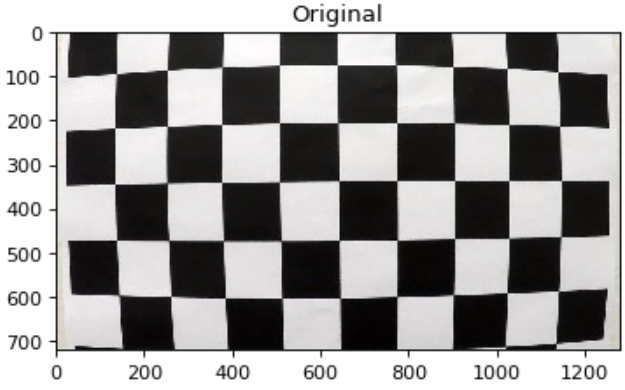
Step 1: Camera Lens Distortion Correction

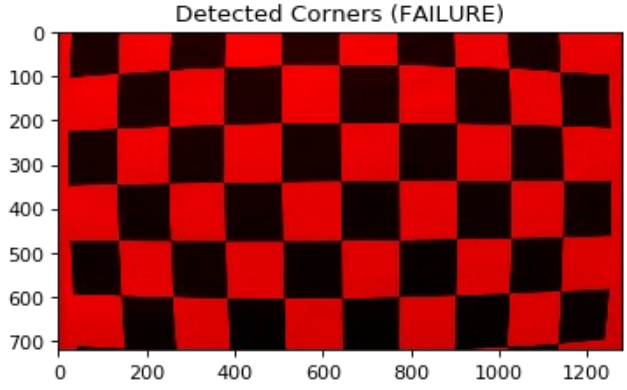
The point of this step is to correct for the distortion caused by the camera lens. The same camera is used to take videos of the road while driving and while it could be possible to identify lanes with or without distortion correction, the measurement of distances and angles would be inaccurate.

Multiple photos of a checkerboard pattern were supplied by Udacity as part of the dataset for the project. The checkerboard pattern contains 10x7 black and white squares, resulting in 9x6 corners. Using the OpenCV computer vision python module, the checkerboard corners were identified. In cases where the corners could not be found in the image, it is returned as a red image. In cases where the algorithm is successful, multi-color lines are overlaid running along the horizontal lines of the checkerboard pattern. Examples of a successful and unsuccessful checkerboard detection are shown below:



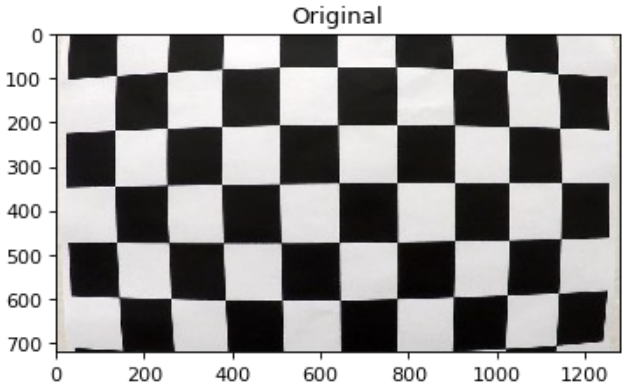


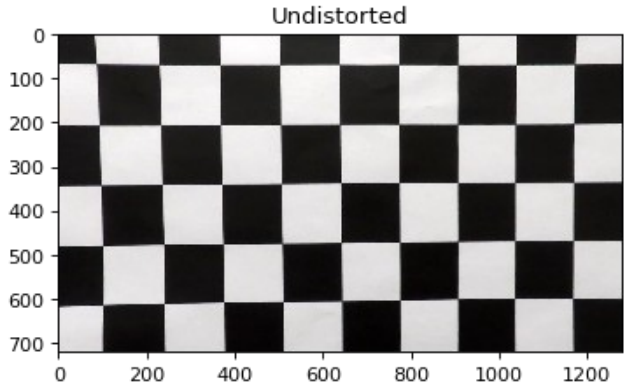




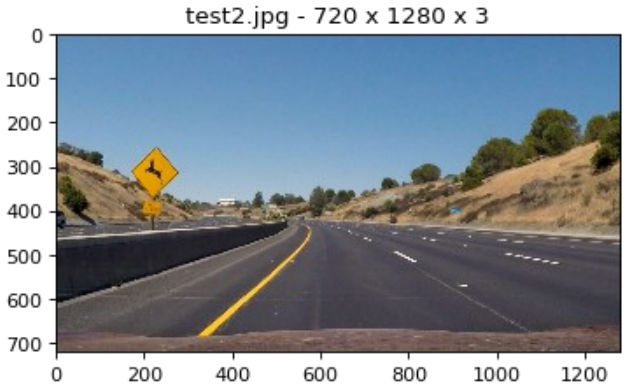
It's important to have many camera angles and zoom levels of the checkerboard pattern in order to accurately calculate a correction transformation. Udacity provided 20 total calibration images, of which 17 had identifiable checkerboard patterns.

Despite the failed checkerboard identification for the image above, it can still be undistorted based upon the other useful calibration images. The result is shown below. Note how the roundness of the lines is nearly completely gone. Some remains in the corners, which could mean that more calibration is required or that the checkerboard pattern being photographed was on a piece of paper that was slightly bent in the corners.





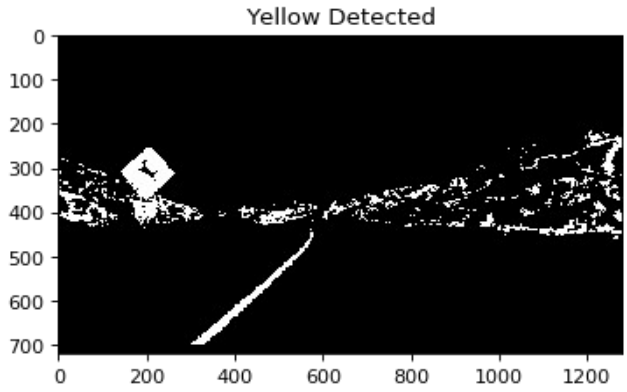
Test images for the road video were undistorted as well. A good example is shown here, where the deer sign has been made to appear more front-facing in the undistorted version.



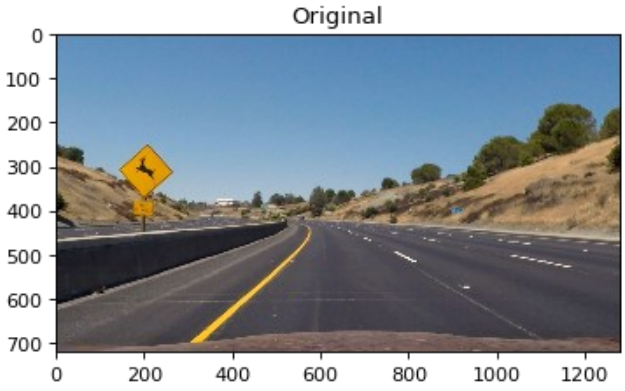
Step 2: Colorspace Conversion and Thresholding

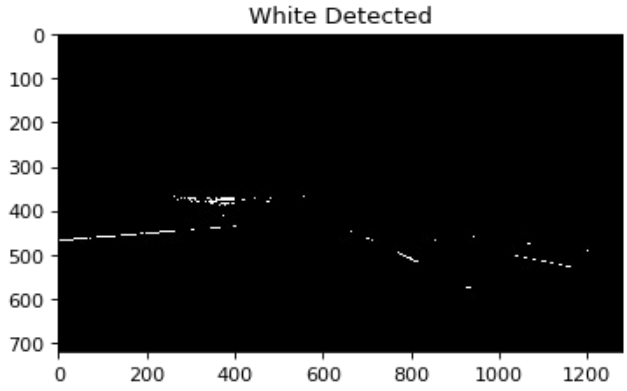
In order to identify locations of lane lines, two colorspace conversions were used: one to detect yellow lines and one to detect primarily white lines (although this one also detects some yellow lines). Colorspaces were chosen over a gradient-based approach, due to the rarity of yellow on the road and due to white lines being sufficiently whiter than the rest of the road in most cases. Successful application of colorspaces also results in many more points for polynomial fitting than gradient-based approaches and tend to suffer from shadowy areas less.

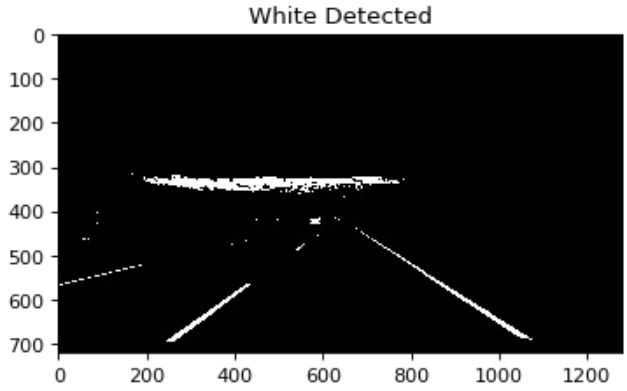
The method used to detect yellow lines is to convert the RGB colorspace into a HSV colorspace and imposing a lower and upper threshold on each channel. The most important channel was the hue (H), which was limited near the hue value for yellow (approximately 25), which was discovered through data analysis using the GIMP (Graphical Image Manipulation Program) informational windows. The image below shows the undistorted image above with yellow detected. Note how the sign and some of the dry grass on the hill are detected, but the lane line itself stands out nicely against the rest of the road.



The method used to detect primarily white lines is to convert the RGB colorspace into a HSV colorspace and imposing a threshold only on the saturation (S) channel. An adaptive algorithm is used to find the proper cutoff to use for the lower threshold, based on the average and standard deviation of the saturation channel. An example of the white detected is shown below for the same image and for another image, which has thicker lines visible.







The output from both the yellow line detector and white line detector are combined in order to produce both lane lines in all cases. This end result is a binary image where most 1's identify a line, at least in the lower half of the image, close to the center.

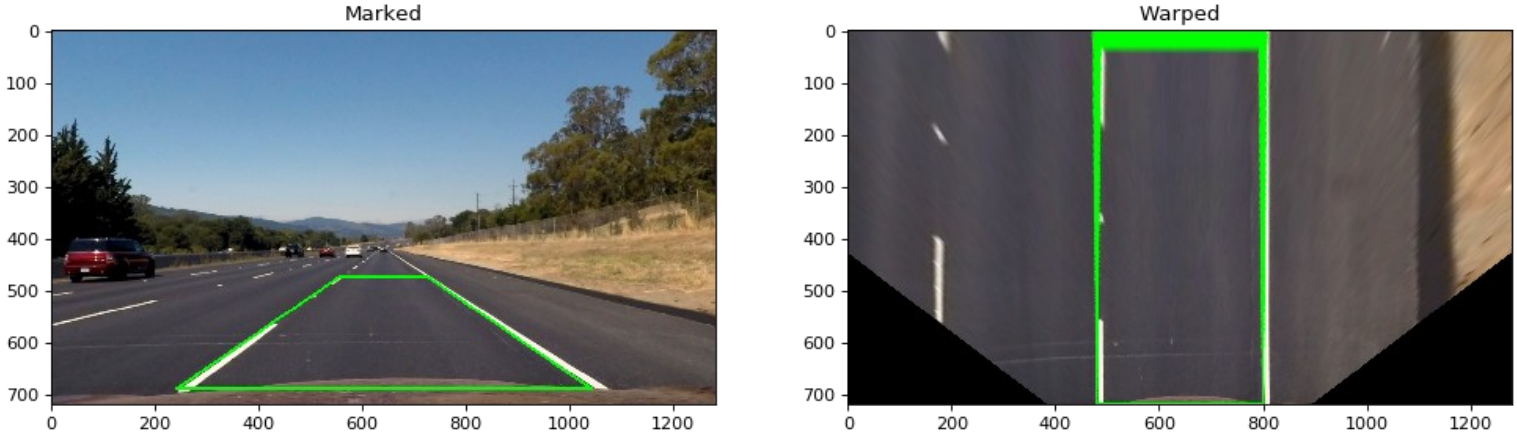
Step 3: Perspective Transformation

At this point, it's possible to declare lane lines as having been identified, however it's required to extract the curvature of the road and in order to do that, a perspective transform can be used to convert the horizon-facing images into a birds-eye view. In order to accomplish this, an example image was chosen that appeared to show a straight stretch of road. Points were then chosen until the converted image resulted in parallel, vertical lines. Care was taken to make sure that the points were equidistant from the center of the image, as it was assumed that the camera was mounted in the center of the vehicle. This is necessary in order to be able to calculate the offset from lane center.

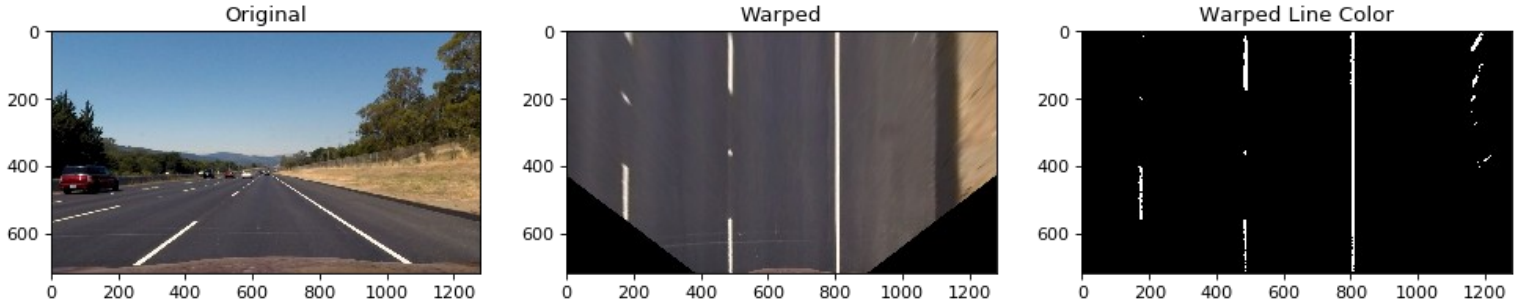
The source and destination points used for the perspective transform are shown in the table below. Note that the image dimensions are 1280x720. For the inverse transformation (unwarping the image), these destination points become the source and vice versa.

	Lower Left	Upper Left	Upper Right	Lower Right
Source	241,690	555,475	725,475	1041,690
Destination	480,719	480,0	800,0	800,719

An example of a perspective-transformed image is shown below. Green marking shows the points that were chosen in the original image and how they were transformed in the "warped" image.



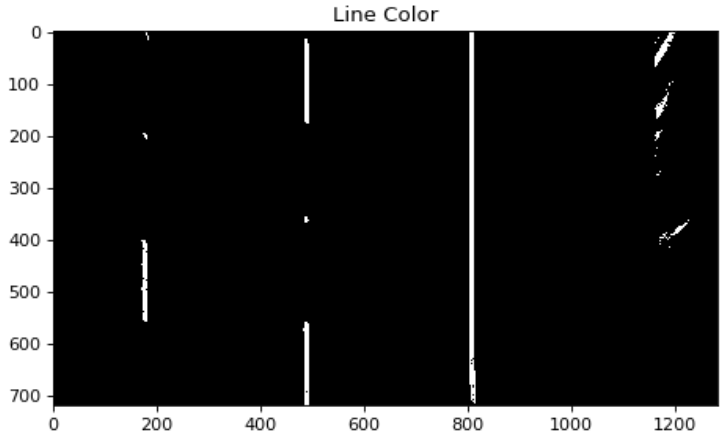
These images show the original, the warped (perspective transformed) version of the original and also the warped version of the line color detection image.

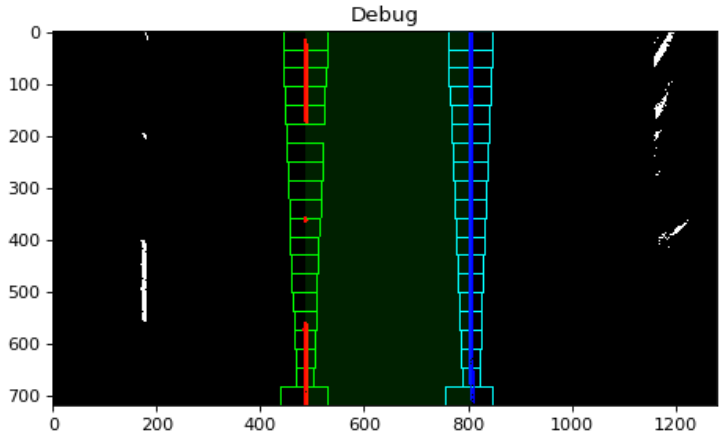


Step 4: Finding Lane Boundaries

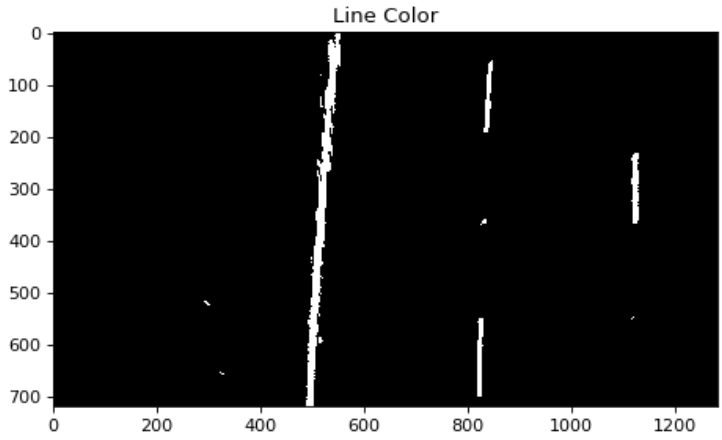
In order to find the lane boundaries, the pixels that are associated with each lane line need to be identified. After identifying the pixels belonging to each line, an equation for a polynomial can be fit to their positions using least squares. This functionality is provided by the NumPy python module with the polyfit(x,y,order) command. In this case, the order used was 2 for a quadratic fit.

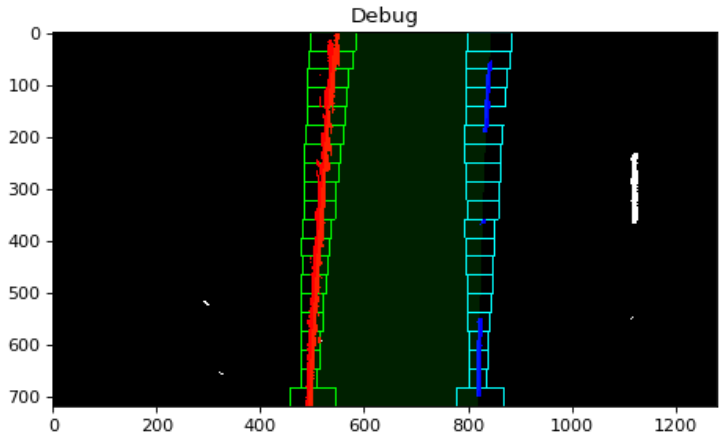
Finding the pixels involves using the technique represented in the lectures, namely searching for the beginning of the line (closest to the vehicle) and progressing upwards (away from the vehicle), recentering the search area as it progresses. An illustration of the implementation is shown below. The green rectangles indicate search regions for the left line and red points indicate discovered line points. The cyan rectangles and blue points correspond to the left line. The green-tinted area is the area between the two polynomials.





Below is another example, showing a slightly curved road.





Step 5: Calculating Curvature and Offset from Center of Lane

Once the polynomial has been fit to the lane lines in the warped image, the curvature can be calculated by finding the radius of curvature along the curve at the point closest to the vehicle.

As cited in the lectures, the radius of curvature for a quadratic polynomial can be defined as:

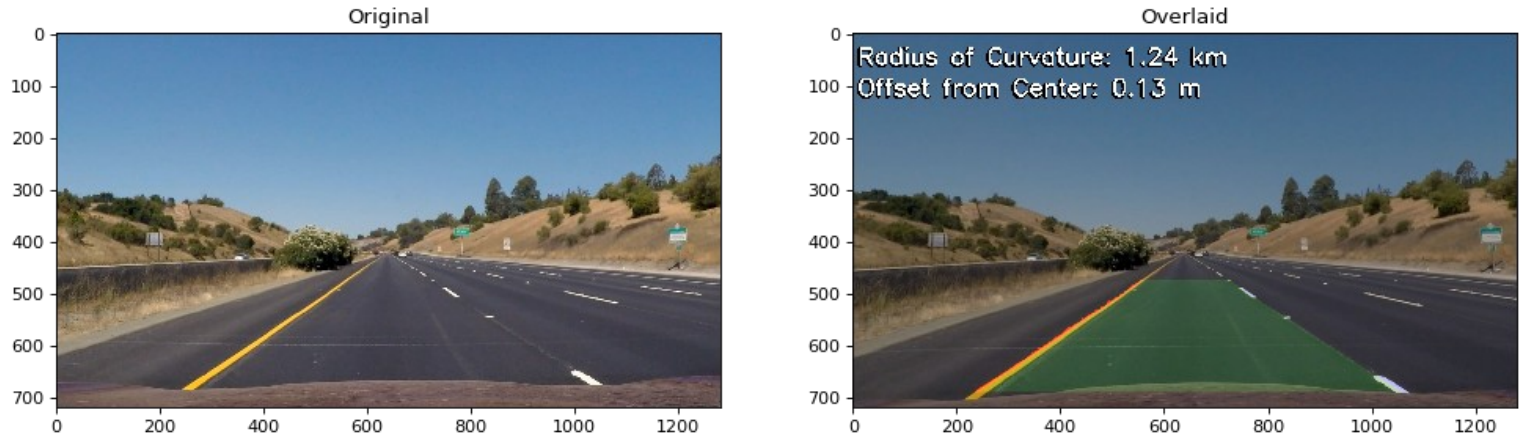
$$y(x)=Ax^2+Bx+C \quad , \quad \frac{dy}{dx}=2Ax+B \quad , \quad \frac{d^2y}{dx^2}=2A \quad , \quad R_{curve}=\frac{\left(1+\left(\frac{dy}{dx}\right)^2\right)^{\left(\frac{3}{2}\right)}}{\left|\frac{d^2y}{dx^2}\right|}=\frac{\left(1+(2Ax+B)^2\right)^{\left(\frac{3}{2}\right)}}{|2A|}$$

...although for our application, x and y will be switched, due to the rotation of the curve in the image.

In order to convert the pixel-value results that this produces to meter results, a ratio of meters per pixel needs to be applied. The conversion can be applied to the polynomial coefficients, by multiplying all x values by (meters/x pixel) and all y values by (meters/y pixel). To find these ratios, information about lane markings in the United States can be used, along with a warped line color image, such as the one shown above in Step 4. According to information provided in the lectures, lane strips are 3.0m long and lane width must be at least 3.7m. Using this information, it was determined that the meters per pixel in the x direction were **3.7m/320pix** and in the y-direction were **3.0m/150pix**.

Step 6: Overlaying Information and Unwarping

The final step in the processing pipeline is unwarping the lane markings and overlaying them on top of the original road-view. (Undistorting the lens is of course not performed.) This is accomplished easily using inverse perspective transform and the addWeighted function from OpenCV. Text is then added to convey the curvature and lane offset values calculated in Step 5. An example is shown below:



Step 7: Video Processing

The same processing steps can be used in the video processing pipeline as in the single image pipeline. Because the frames are in a time-series, the warm search mentioned above in Step 4 can be used following successful cold line searches. Additionally, new curve polynomial coefficients are combined with previous curve coefficients using an alpha mixing factor, which is adjustable, but defaulted to 0.5. The attached videos show the output from both the project video and the challenge video.

Discussion

As mentioned above, an area of improvement for camera lens distortion correction is to take more photos, although 17 does seem to be sufficient for good results. If any new camera (or lens) were to be used, this step would have to be repeated.

Lane line detection is probably one of the areas where improvement could be observed the most. One idea is to use a variety of line detection techniques and use a means of picking which one is best. This was attempted in the video processing pipeline, but the performance was not very reliable.

One of the side effects of performing a perspective transformation was the blurring of the pixel values in locations where the horizon is expanded, which can affect how saturated a line appears or its hue. Perhaps this expansion should be smaller.

Finding the lane boundaries is another great area of improvement for this pipeline. Specifically, techniques to determine whether the curvature is valid or close enough to a previous valid value were considered and implemented in limited fashion, but their performance isn't perhaps as powerful as they could be. This was due to their reliance on

certain assumptions, such as what the road may look like when a cold search needs to be performed. This made it difficult for the algorithm to perform well in the harder challenge video on tight turns.

Step 4: Performance

In order to test the performance, the drive.py script was run using the model generated after training the network. Performance on track one was excellent at the 9 mph default speed limit set in drive.py. Performance is mostly stable up to 20 mph. Around 30 mph, a resonance becomes apparent, as the vehicle tries hard to stay in the exact center, while often over-running its target. This could probably fixed one of three ways: 1) retraining with better training data, 2) using a dampening low-pass filter or 3) using a higher steering calculation rate. It's like that option 1 is the best, due to the fact that the dampening filter may hurt the performance during hard turns and a higher steering calculation rate may be too costly.

Step 5: Standing Out / Mountain Course

The same network design was trained using data collected and augmented in the same fashion as for the primary course. The vehicle was only driven on the right side of the road, due to the existence of a certain lane. The performance was surprisingly great. The car occasionally touches the lines with a tire, but not often, and it never leaves the road. The video was taken with the car operating at a target speed of 20 mph.