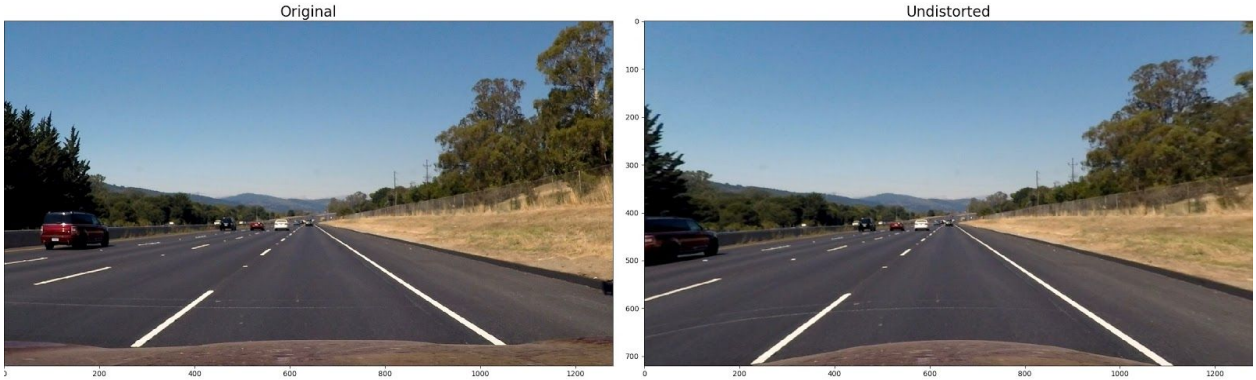
The project implements finding lanes on a road, measuring its curvature and finding the offset of the vehicle from the lane center.
Final video - [*output_images/submission_video.mp4*]

CAMERA CALIBRATION

File dealing with calibration and undistortion - ***calib.py***

1. First we find the chessboard corners from the calibration images using opencv
2. Then we create the 3D points corresponding to these corners as we know they are evenly distributed in space.
3. Then we use the opencv function "*cv2.calibrateCamera*" to get the camera matrix and distortion coefficients
4. This is then used to undistort the image

Sample Output :



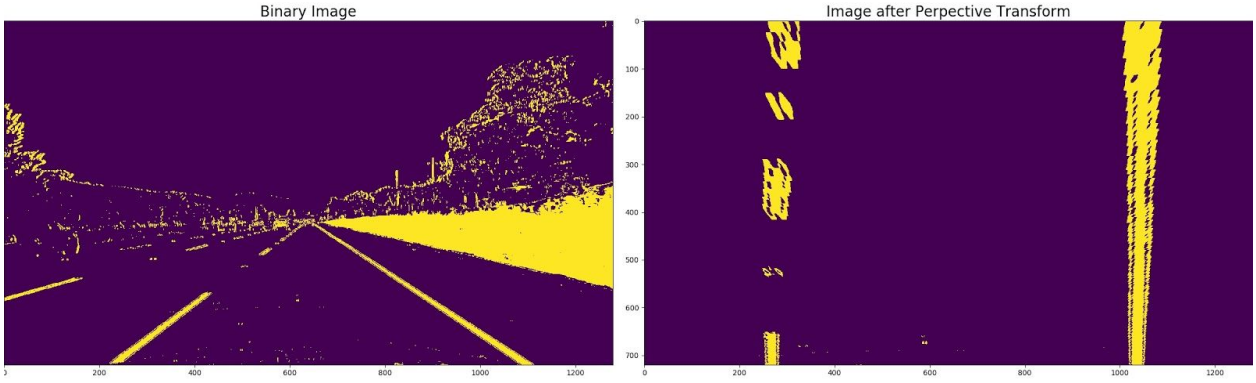
PIPELINE

1) **Distortion Correction** -
   After calibrating the camera we can use the camera matrix and distortion coefficients to undistort the image. I have provided a sample output in the section above
2) **Binarization** -
   File dealing with binarization - ***binary.py***
   This step involves using color and gradient thresholds to get a sense of where the lane lines are.
   a) I converted the image into the HLS color space and used appropriate thresholds to segment out yellow and white patches
   b) Then a threshold on the sobel gradient is used to find high gradients in the image.
   c) Combining both the results gives us a rough estimate of where the lane lines lie.

Sample output :

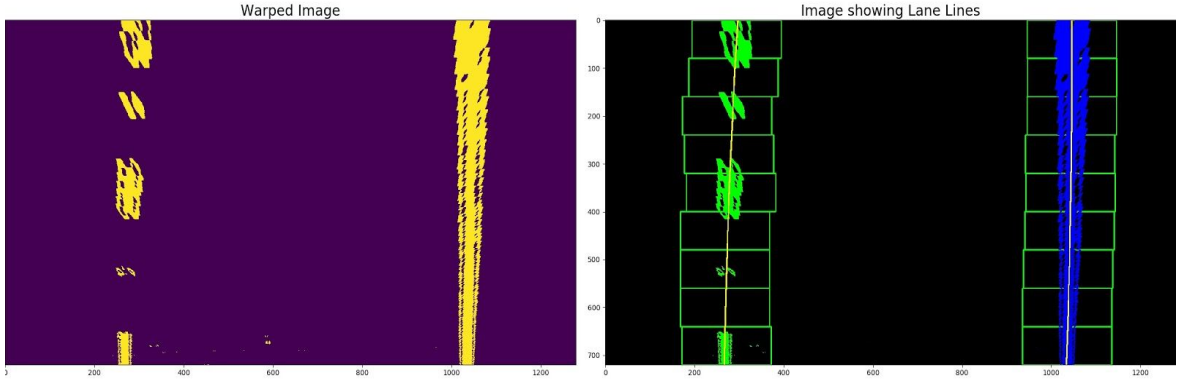
3) **Perspective Transform -**
File dealing with perspective transform - ***perspective.py***
The function ***warp_top_view()*** returns a birds eye view of the road.
We choose 4 points in the image and map it to 4 points in the warped image.
If we have chosen the points correctly for a bird's eye view, lane lines for the example image chosen, should appear parallel in the warped image.

Sample output :

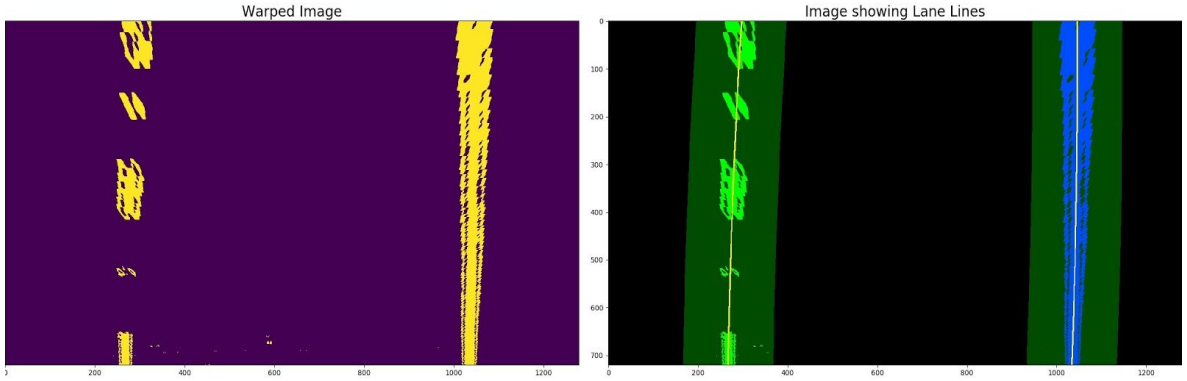


4) **Identify Lane Lines -**
File dealing with finding lane lines - ***lane.py***
The lane lines could be found via an exhaustive sliding window search or using the lane lines found in the previous frame.
A moving average of the lane line coefficients is maintained to avoid errors due to bad frames.
    a) **Exhaustive sliding window** :
    The image is divided into two halves along the X axis for the left and right lane

lines. In each half of the image, we define small windows along the Y direction to look for lane lines.
Peaks in the histogram of the binary warped image indicate lane lines and the starting point for the sliding windows.
These windows are shifted along the X direction along the curvature of the line depending on the number of lane pixels found in the sliding window.
The pixels indicating lanes in these windows is then used to fit a two degree polynomial representing the lane line.
Sample Output :



b) **Searching along previous lane line :**
Instead of defining discrete sliding windows to search for pixels indicating lanes, we use the previous lane line and search for the lane pixels around it.
Sample output :

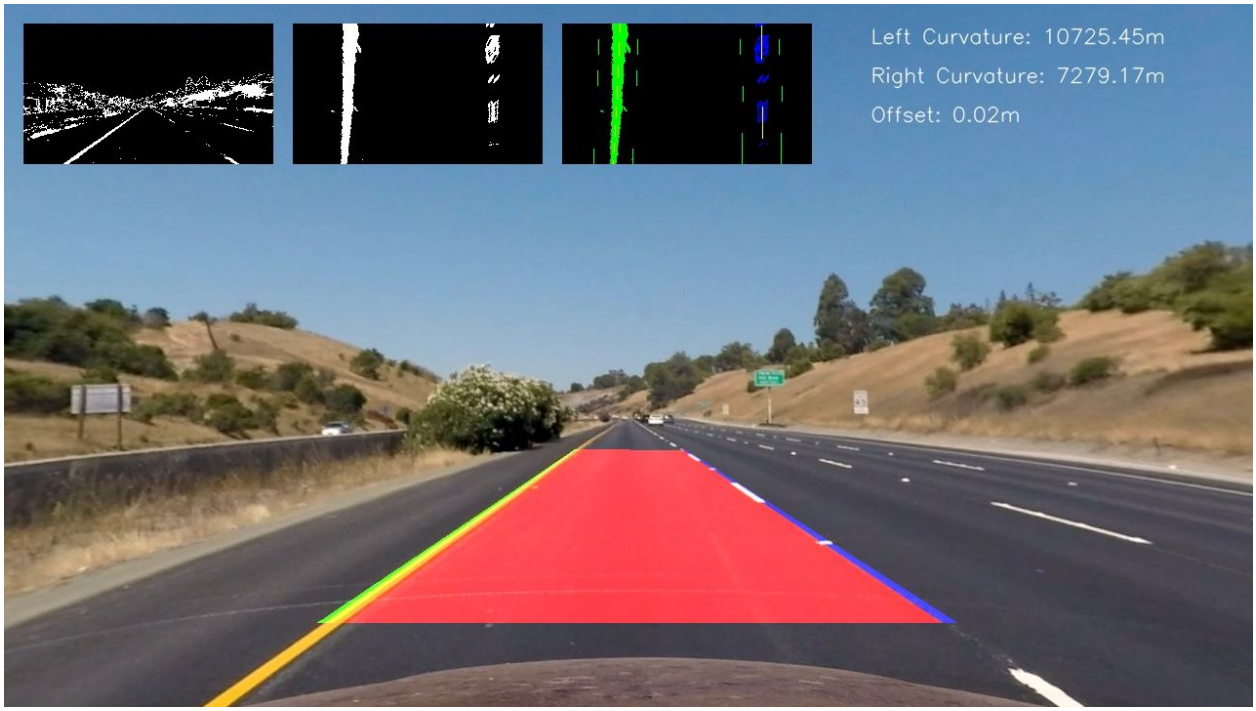


5) **Radius of curvature and offset -**
The radius of curvature is calculated using the formula mentioned in the lectures.
It is implemented as a function in the Curve class in ***curve.py***
As for the offset we can assume that the camera is mounted on the centre of the car.

Under this assumption, the offset of the image center from the lane center, along the X axis, becomes the offset of the car from the lane center.
We then multiply this with the scaling factor to convert from pixel to metre space.
The scaling factor is calculated as -
(width of the warping boundary in object space) / (pixel width in the warped image)
This is implemented as the function ***get_offset_from_center()*** in ***lane.py***

6) **Plot lane back on image -**
All the parts of the pipeline are then stitched into one image to be displayed.
The function ***stitch_all_results()*** in ***main.py*** implements this

Sample output :



The whole pipeline can be run from the file ***main.py*** with a parameter which decides whether to run on images or the video.
Sample -
python main.py --mode img
python main.py --mode video

SHORTCOMINGS / IMPROVEMENTS

- Since the detection pipeline is heavily dependent on gradients, any high gradient affects the detection pipeline. Eg - gradients due shadows of other moving vehicles on the lane, or trees on the sides
- With the wide use of Fully Convolutional Networks for semantic segmentation, they are the obvious improvement to segment out lane lines with high accuracy