Project 4: Software pipeline to identify lane boundaries with front camera

Advanced Lane Finding

The goals / steps of this project are the following:

- Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- Apply a distortion correction to raw images.
- Use color transforms, gradients, etc., to create a thresholded binary image.
- Apply a perspective transform to rectify binary image ("birds-eye view").
- Detect lane pixels and fit to find the lane boundary.
- Determine the curvature of the lane and vehicle position with respect to center.
- Warp the detected lane boundaries back onto the original image.
- Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position.

Overview

The goal of this project is to understand and implement pipeline for advanced lane finding solution for the recorded video as input. To start with understand the calibration, distortion coefficients using a chessboard image, and using cv2 chess board functions to get an undistorted image. Also do the required perspective image transformation to visualize correctly.

Then use the various threshold methods to identify the correct and gradient scale of the image. Using the threshold and perspective transformation of the image, identify the interested region. Identify the lane that needs to fit, also calculate the curvature/radius of the lines. Using the lane area and perform the inverse perspective transform and combine the processed image with the original image.

Files Submitted & Code Quality

Required Files:

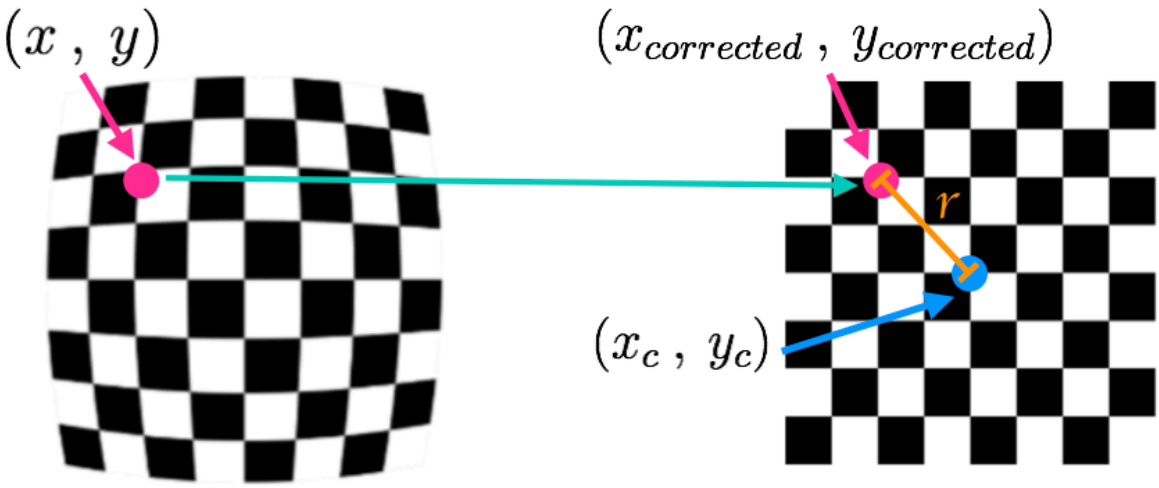
My project includes the following files:

- AdvancedLaneFinding.pynb containing the code for the calibration, image pipeline and video creation
- AdvanceLaneFinding.html file with the test images
- writeup_report.pdf summarizing the results
- project_video_out.mp4 is the recorded video of the final image processing using the pipeline
- Test images are provided part of AdvancedLaneFinding.pynb as well as in writeup_report.pdf for various stages of the pipeline

Camera Calibration

Distortion Coefficients and Correction

In our video/images we see mostly Radial and Tangential distortion. Below is the chess board provided which depicts the distorted and undistorted chess boards i mages. It shows the (x,y) are the coefficients seen in the distorted image, (x-corr ected, y-corrected) are the corrected points in the undistorted image.



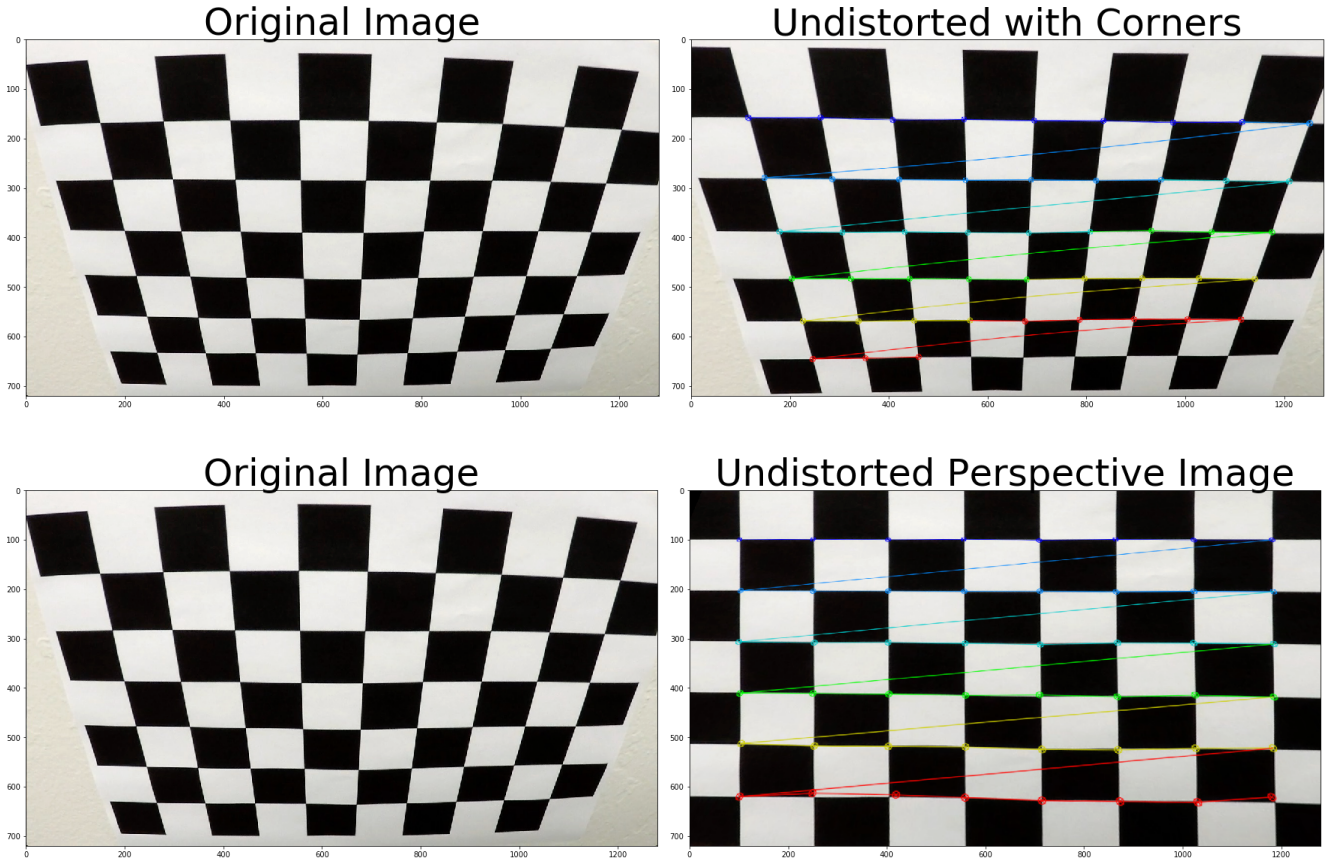
Visualization:

OpenCV functions are used to calculate the correct camera matrix and distortion coefficients using the Calibration chess board images provided part of repository.

Here I have used the image camera_cal/calibration2.jpg from the repository.

The first step to get the object points from the distorted chess board image, then convert the image to the Gray scale using cv2.cvtColor function using cv2.COLOR_BG2GRAY flag. Get the corners of the chess board using the cv2.findChessboardCorners function.

Used the cv2 caliberateCamera function to get the matrix and distortion coefficients for the gray scaled image.



The detected corners are used to visualize the perspective image of the chess board as show above, using the cv2.warpPerspective image.

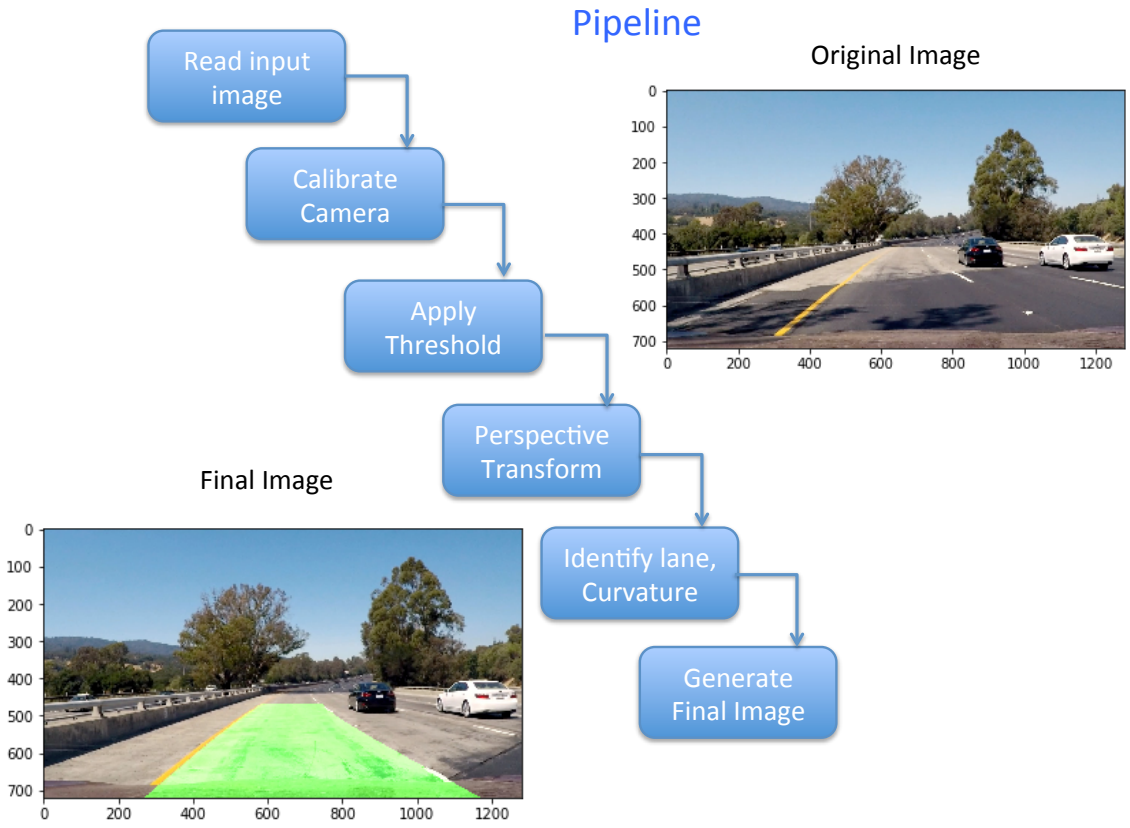
Created a wrapper function corners_unwarp_chess, which takes the calibrated image and produces the warped perspective image using the cv2 functions.

Pipeline (Single Image)

In the advanced lane finding pipeline, we take each clip/image from the video and perform the various stages of processing in the pipeline.

Used the matplotlib image library to read the image from the test image repository. Here is the main function process_image which takes the original image as input and produces the result which is the final image with lane detected and plotted.

```
def process_image(img):

    warped, M, Minv, binary_warped, undist = pipeline3(img)

    ploty, left_fitx, right_fitx, out_img = lanefit(binary_warped, img)

    result = finalImage(binary_warped, img, ploty, left_fitx, right_fitx, Minv)

    return result
```



Distortion Correction:

The original image provided has distortion, since the camera is fixed at the center of the car. There are both Radial and Tangential distortion part of the images. In order to processing correctly the image need to be undistorted as needed.

Code snippet from my "def corners_unwarp(img, mtx, dist):"

```
# Used the OpenCV undistort() function to remove distortion
undist = cv2.undistort(img, mtx, dist, None, mtx)
plt.imshow(undist)
```

The above image is the original and Undistorted image, using the cv2.undistort

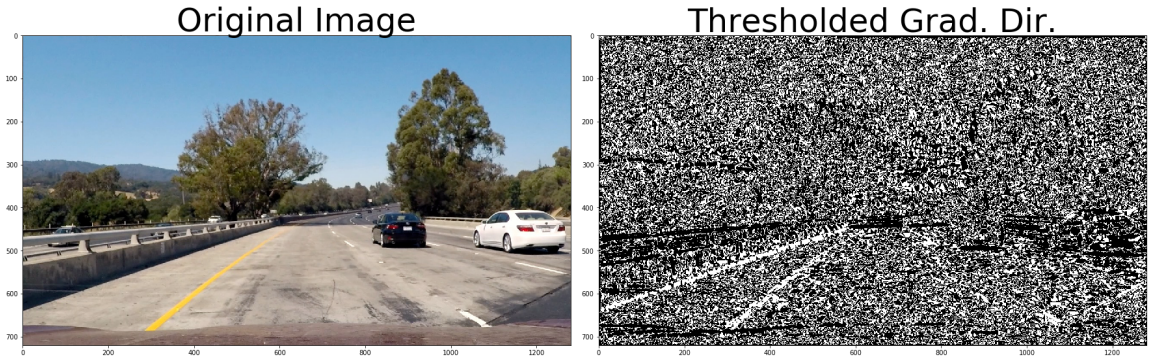
Thresholded Binary Image:

On the undistorted image, applied the various threshold functions to identify the pixels where the gradient of an image falls with a specified threshold range.

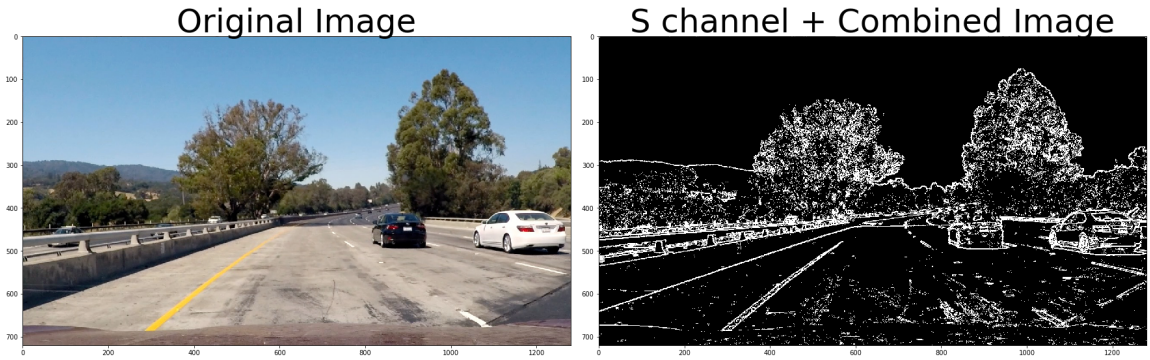
Code snippet from my "def pipeline(img):

```
# Apply each of the thresholding functions
    gradx = abs_sobel_thresh(img, orient='x', sobel_kernel=ksize, thresh=(40, 200))
    grady = abs_sobel_thresh(img, orient='y', sobel_kernel=ksize, thresh=(40, 200))
    mag_binary = mag_thresh(img, sobel_kernel=ksize, mag_thresh=(40, 200))
    #plt.imshow(mag_binary)
    dir_binary = dir_threshold(img, sobel_kernel=ksize, thresh=(0, np.pi/2))

    combined = np.zeros_like(dir_binary)
  combined[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1))] = 1
```

Below is the image with using the directional threshold, which clearly depicts the lanes. In my code dir_threshold uses the cv2 functions arctan2 and sobel functions.



As we know image is represented by RBG, but yellow pixels is not correctly in the gradient image. So, we use the HLS intuitions in determine the correct color pixels' gradients in processing the image. Here in the below example used the S –Channel with the Thresholded image in order to identify both yellow and white pixels.



Code Snippet from def pipeline3(img):

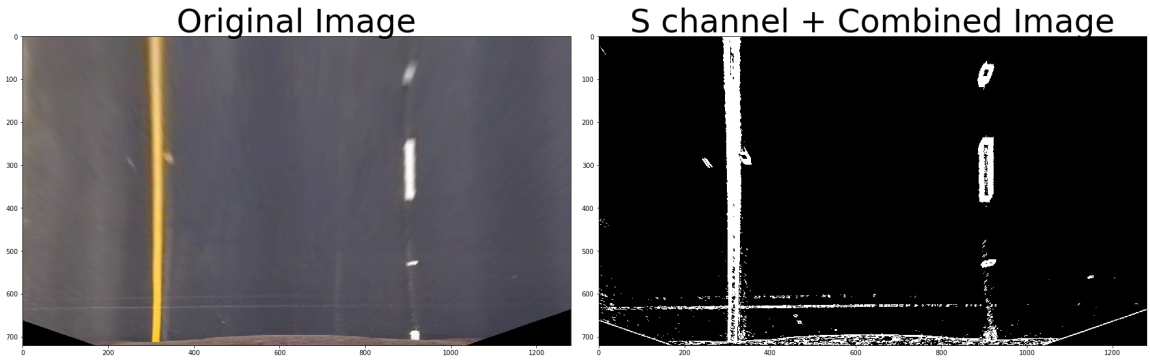
```
    warped, M, Minv, undist = corners_unwarp(img, mtx, dist)
hls = cv2.cvtColor(warped, cv2.COLOR_RGB2HLS
s = hls[:,:,2]

…

s_binary = np.zeros_like(combined)
s_binary[(s > 170) & (s < 255)] = 1
color_binary = np.zeros_like(combined)
color_binary[(s_binary > 0) | (combined > 0)] = 1
```

Perspective Transform:

Perspective transform maps the points in given image to different, desired, points with a new perspective. The perspective transform will provide a birds-eye view transform that lets us to view a lane from above.

Below is the original and undistorted image:

The below image provides the perspective view of the lanes. The Left side image is the perspective view of the original image, the right-side image is the HLS, Thresholded image birds eye view.

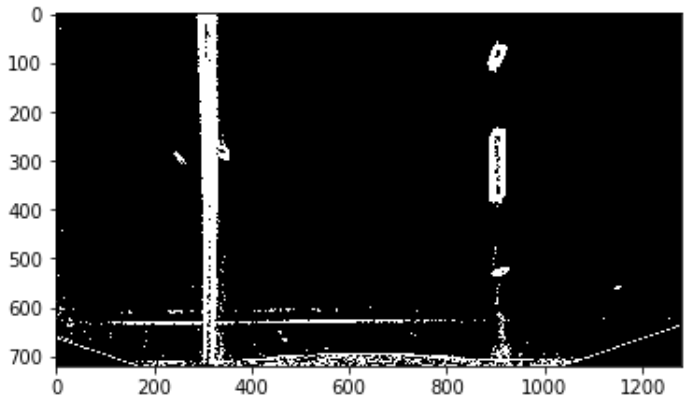


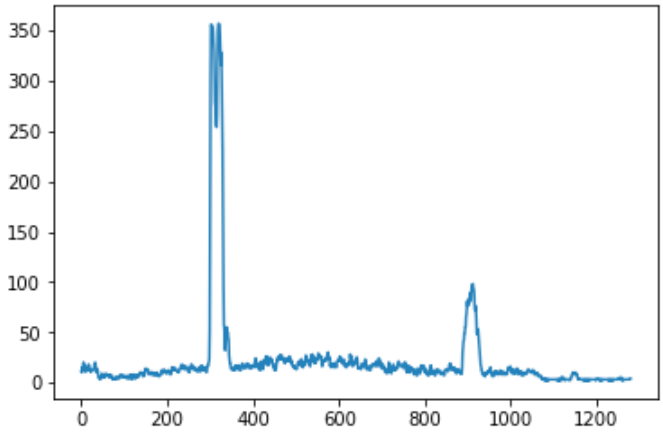
Code snippet from "def pipeline3(img):"

```
src = np.float32(corners)
dst = np.float32(corners_dst)
# Given src and dst points, calculate the perspective transform matrix
M = cv2.getPerspectiveTransform(src, dst)
MInv = cv2.getPerspectiveTransform(dst, src)
# Warp the image using OpenCV warpPerspective()
warped = cv2.warpPerspective(undist, M, img_size)
```

Lane Pixels and Poly fit:

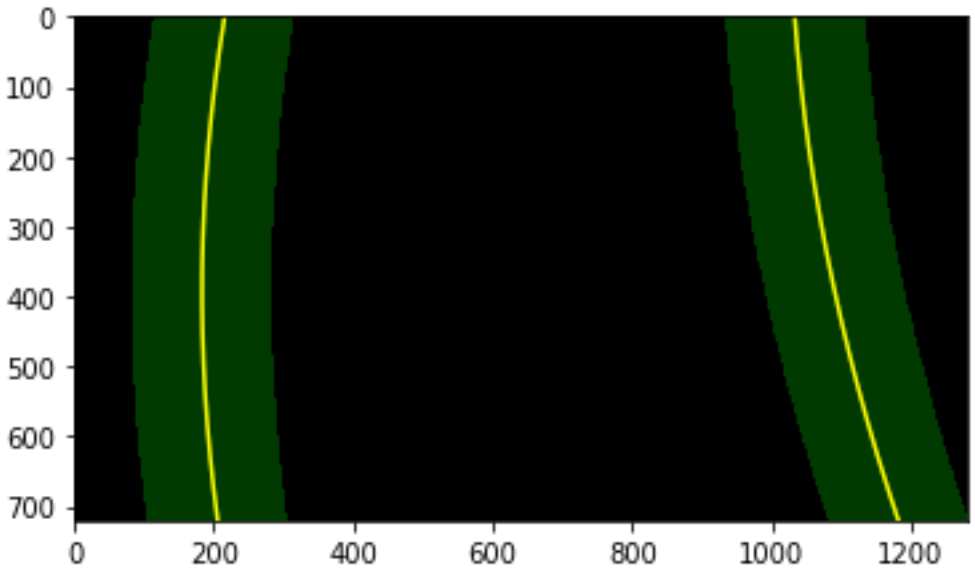
Used the peaks in a histogram line finding method, in order to determine the lane lines that standout clearly. Finally used the sliding windows and fit a polynomial method to find the hot pixels that are associated with the lane lines.





Created a lanefit function takes the binary warped and combined image in determine the sliding window. By default the windows has been set to 9, and steps through the windows one by one in plots the rectangle. Finally add the left lane and right lane indicies to left_lane_inds and right_lane_inds.

The polyfit function is used to get polynomial with the non zero left and right lane x and y pixel positions.



Radius of curvature of lane:

Used the below lane curvature function in determining the curvature of lanes after the histogram sliding window line detection.

```
def lanecurvature(leftx, lefty, rightx, righty) :

    # Fit new polynomials to x,y in space

    left_fit_cr = np.polyfit(ploty*ym_per_pix, leftx*xm_per_pix, 2)

    right_fit_cr = np.polyfit(ploty*ym_per_pix, rightx*xm_per_pix, 2)

    # Calculate the new radii of curvature

    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix +
left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])

    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix +
right_fit_cr[1])**2)**1.5) / np.absolute(2*right_fit_cr[0])

    # Now our radius of curvature is in meters

    print(left_curverad, 'm', right_curverad, 'm')

    # Example values: 632.1 m    626.2 m

    return left_curverad, right_curverad
```

Final Image with lane detection:

Created a function finalImage takes the warped image and the fit lines, and uses the inverse transpose from the original image and transform the perspective image. And doing the inverse, the cv2.fillpoly is used to fill the marked lanes with the Green color as shown below to get the final Image.


```
def finalImage(warped, img, ploty, left_fitx, right_fitx, Minv):
    warp_zero = np.zeros_like(warped).astype(np.uint8)
    #plt.imshow(warped)
    warp_zero = np.zeros_like(warped).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))
    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))
        # Draw the lane onto the warped blank image
    cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))
    # Warp the blank back to original image space using inverse perspective
matrix (Minv)
    newwarp = cv2.warpPerspective(color_warp, Minv, (img.shape[1],
img.shape[0]))
    # Combine the result with the original image
   result = cv2.addWeighted(img, 1.0, newwarp, 0.6, 0)
    return result
```

Pipeline (Video)

Final Video:

Used the moviepy.editor function VideoFileClip to read the given image project_video.mp4 and used the fl_image function to read each images in the video file and used the *process_image* function to do the pipeline processing.

The processed image is stored in mp4 format and stored in the same location as project_video_out.mp4 .

clip = VideoFileClip("project_video.mp4")
video_out = "project_video_out.mp4"

video_cap = clip.fl_image(process_image)
%time video_cap.write_videofile(video_out, audio=False)

return result

```
clip = VideoFileClip("project_video.mp4")
video_out = "project_video_out.mp4"

video_cap = clip.fl_image(process_image)
%time video_cap.write_videofile(video_out, audio=False)
```

```
Image shape: (720, 1280, 3)
[MoviePy] >>>> Building video project_video_out.mp4
[MoviePy] Writing video project_video_out.mp4
  0%|          | 0/1261 [00:00<?, ?it/s]
Image shape: (720, 1280, 3)
  0%|          | 1/1261 [00:00<06:31,  3.22it/s]
Image shape: (720, 1280, 3)
  0%|          | 2/1261 [00:00<06:28,  3.24it/s]
Image shape: (720, 1280, 3)
```

Conclusion, Future Work

- Was able to the lane detection using the example code provided in the various exercises. The various exercises really helped in understanding the concepts like calibration, threshold, HLS, curvature, perspective view.
- As an improvement to the code, will try the python class based code implementation.
- Will also try with the different videos in identifying the lanes, and see how this code scales.