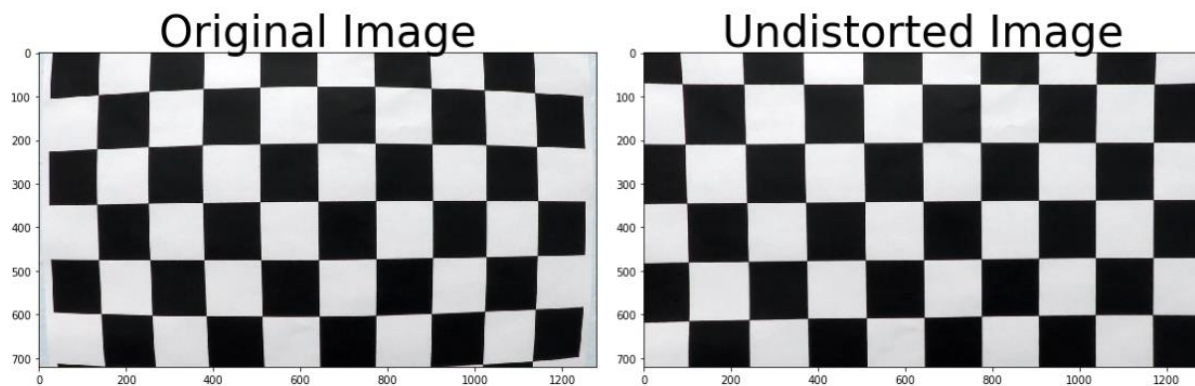


This is the write-up report for project4 Advanced Lane Finding.

## Camera Calibration

Camera calibration is done using all images located in 'camera\_cal/' folder. The code for camera calibration is in first cell of ipython notebook, "P4.ipynb." This section is pretty straight forward, all same steps like in lectures.

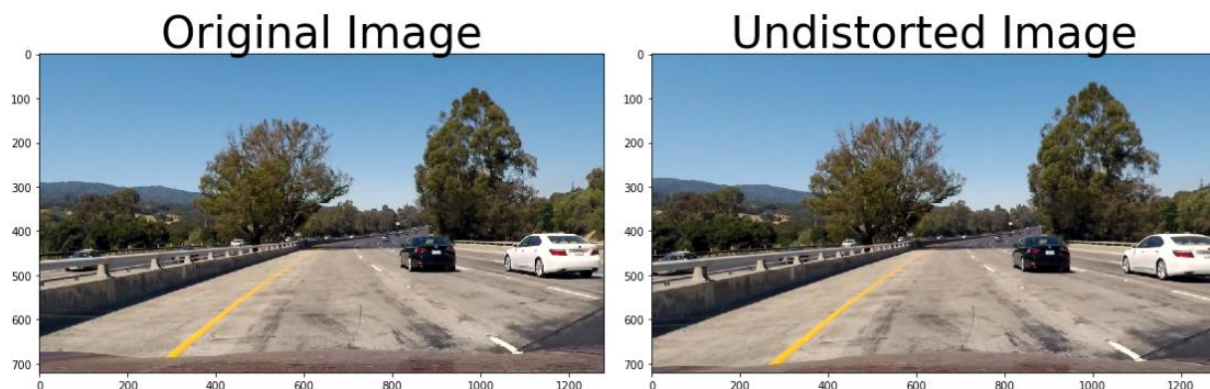
"Corners" points are obtained from `cv2.findChessboardCorners` to create "objpoints" and "imgpoints." These corner points are fed into `cv2.calibrateCamera` to get "mtx" and "dist" that needed to undistort image with `cv2.undistort`.



This camera calibration step is actually included in `lane_coloring()`, not in `pipeline()` due to the pipeline structure. This is because I started off from this function, and stuck with this naming convention. It is easier to be understood that `pipeline()` is actually a filtering function for image thresholds explained in next section.

## Pipeline (single images)

### 1. Provide an example of a distortion-corrected image.



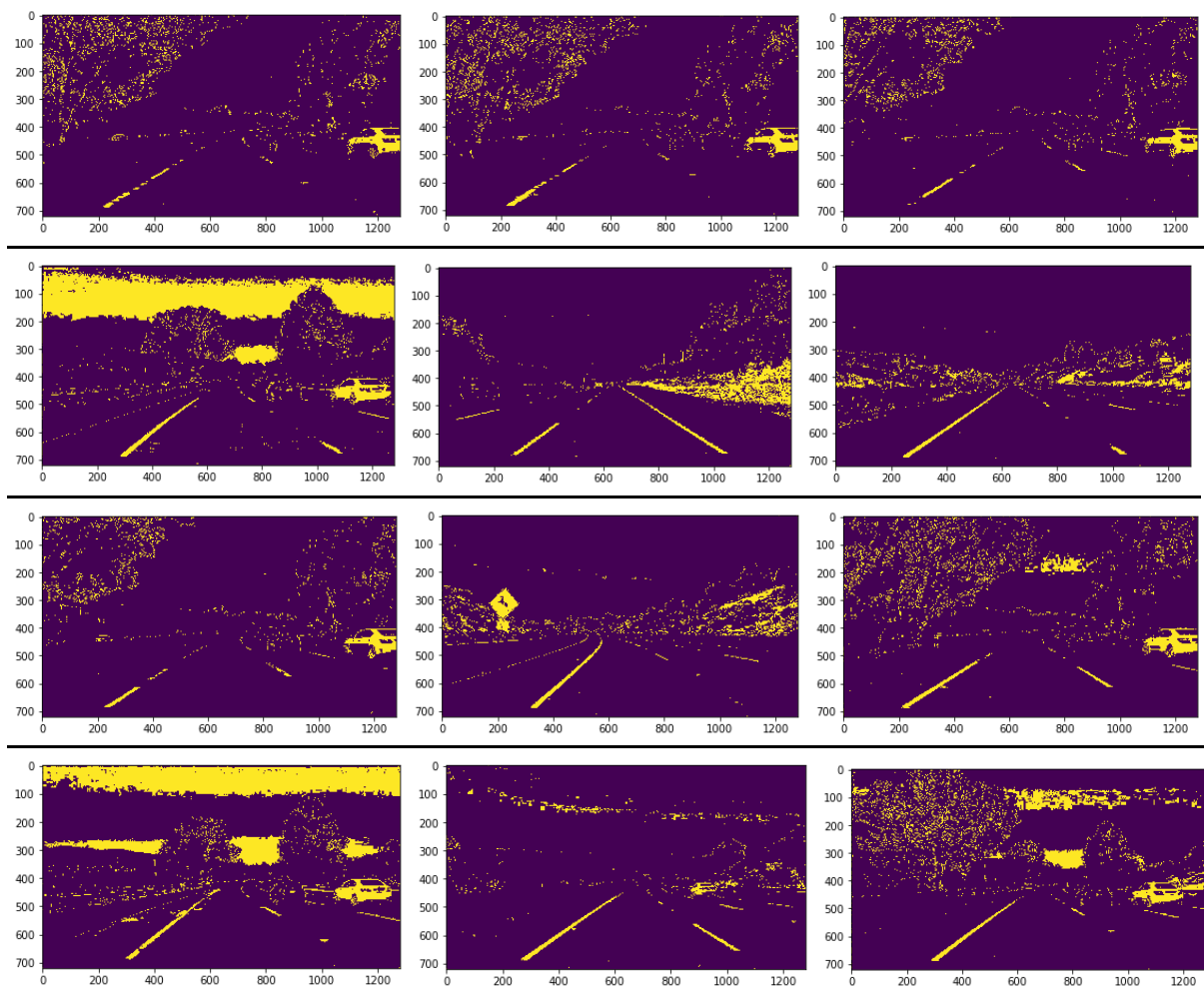
Above images are the outcome of camera distortion correction.

## 2. Describe how (and identify where in your code) you used color transforms, gradients or other methods to create a thresholded binary image. Provide an example of a binary image result.

There are several color transform methods are mixed together.

- L-channel from HLS color channels
- HSV filter recycled from Project1; Grayscale image from `filter()` >150
- Simple RGB filtering condition with  $R > 150$ ,  $G > 100$ ,  $B < 100$ .

Usage of s-channel and sobel x gradient was also considered, but dropped eventually because I found out it gives bad result on some section of project video – where the shadow from tree and road color changes drastically. Following is the list of output images from combination of above methods. Test images include all images in “test\_images” folder and some frame captured from 40~42<sup>nd</sup> sec.



```

test_images_output/frame1038.jpg
test_images_output/frame1039.jpg
test_images_output/frame1040.jpg
test_images_output/frame1041.jpg
test_images_output/frame1042.jpg
test_images_output/frame1043.jpg
test_images_output/straight_lines1.jpg
test_images_output/straight_lines2.jpg
test_images_output/test1.jpg
test_images_output/test2.jpg
test_images_output/test3.jpg
test_images_output/test4.jpg
test_images_output/test5.jpg
test_images_output/test6.jpg

```

### 3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

```

imshape = img.shape
x1 = int(imshape[1]/2 - 85)
x2 = int(imshape[1]/2 + 85)
y1 = int(imshape[0]/2 + 100)
y2 = int(imshape[0])

if img is not None:
    src = np.float32([[100,y2-40],[x1, y1],[x2, y1],[imshape[1]-100,y2-40]])
    dst = np.float32([[300,670],[300,50],[980,50],[980,670]])

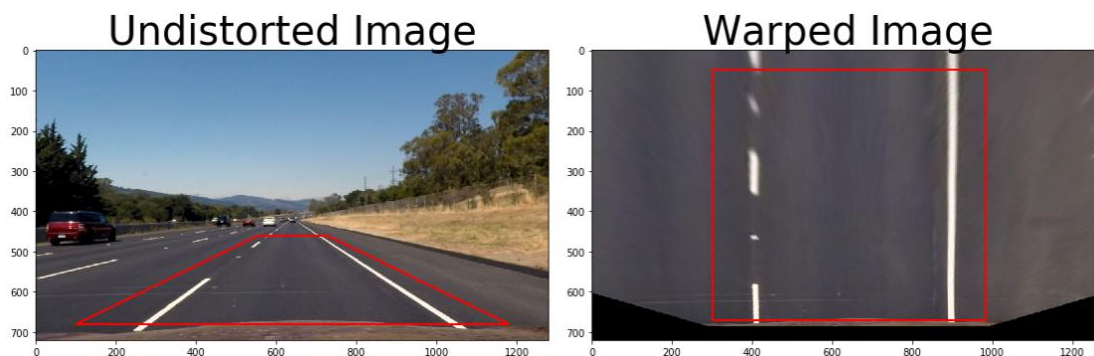
```

Most coordinates of perspective transform source (src) are calculated based on size of image (Bottom 40 pixels are removed due to the hood of the vehicle), and most of destination (dst) are hardcoded on the other hand.

To summarize, following table is the source and destination coordinates.

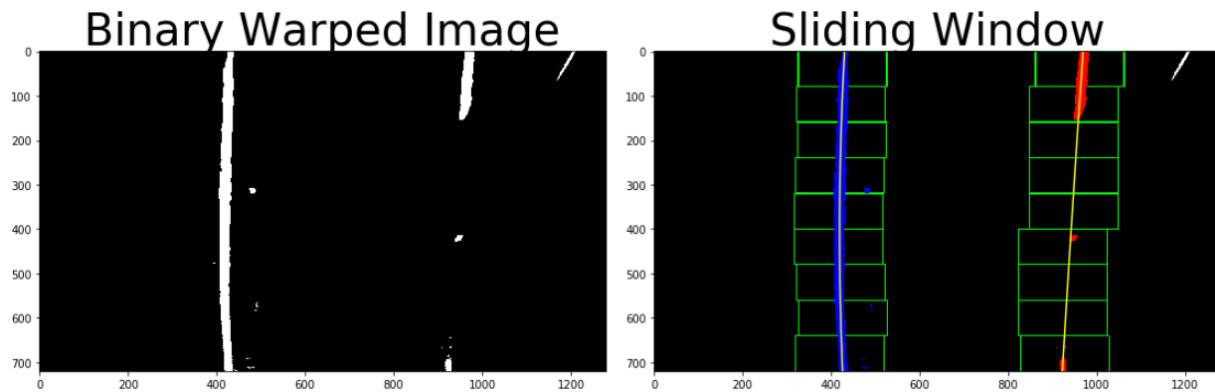
Source	Destination
100, 680	300, 670
555, 460	300, 50
725, 460	980, 50
1180, 680	980, 670

And following is the result of perspective transform.

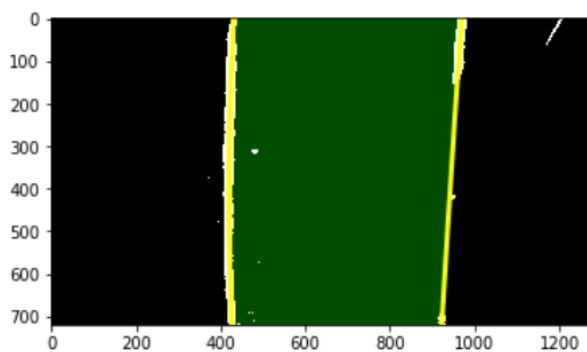





#### 4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

Similar to the lectures, I performed sliding window method and polynomial to fit the lane lines.



Then, I filled the regions in between of lanes with `cv2.fillPoly`, converted this binary back with unwarp process by performing perspective transform with src and dst swapped.



```
# Unwarp to transform coloring back to camera perspective
M1 = cv2.getPerspectiveTransform(dst, src)
unwarp = cv2.warpPerspective(weighted, M1, img_size, flags=cv2.INTER_LINEAR)
img = cv2.cvtColor(unwarp, cv2.COLOR_BGR2RGB) #Uncomment for images
colored = cv2.addWeighted, 1, unwarp, 0.8, 0)
```

#### 5. Describe how (and identify where in your code) you calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

Lane curvature calculation is in the middle of `lane_coloring()` pipeline from line 68~100, and position of vehicle is from lane 116~130.

Position is simply calculated base on the average of bottom values (closest to the vehicle, `yvalue = img.shape[0]`) of two curvatures, assuming the camera is perfectly centered. Thus, if this value is greater than mid point (`x:640`), then it's off to right side with deviation distance `dist = abs((ctr_p - 640)*xm_per_pix)` where `xm_per_pix` is given as 3.7m/700 pixels.

6. Provide an example image of your result plotted back down onto the road such that the lane area is identified clearly.



Image test results can be populated with following,

```
# Testing lane_coloring() on images
test_dir = './test_images/'
output_dir = './test_images_output/'
files = os.listdir(test_dir)

for file in files:
    image = mpimg.imread(test_dir + file)
    output = lane_coloring(image)
    filename = output_dir + file

    os.makedirs(os.path.dirname(filename), exist_ok=True)
    cv2.imwrite(filename, output)
```

And video with following code cell, both included in P4.ipynb

```
from moviepy.editor import VideoFileClip
from IPython.display import HTML
import os

output = 'test_videos_output/project_output.mp4'

clip = VideoFileClip("project_video.mp4")#.subclip(40,42)
processed_clip = clip.fl_image(lane_coloring)
```

## Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

A few things I noticed was that the curvature/vehicle position calculation is very sensitive base on values of perspective transform and color thresholds.

Also, there is a room for further improvement with adaptive thresholding although I managed to complete it with a combination of static thresholds. The pipeline might still likely fail if there was any yellowish/white vehicle nearby within the camera range.

## Additional Discussions for Resubmission

### Camera Calibration

#### Required

There are two very similar problems in the camera calibration. Though the calculations are correct the problem is how many times the camera is calibrated.

In `cal_undistort`, camera calibration is done. Each call to `cal_undistort` would require to calculate the camera calibration. In the `lane_coloring()`, camera calibration is done for each frame. Camera calibration can be done in the beginning of project and these variables `ret, mtx, dist, rvecs, tvecs` can be reused.

I have tested this claim with following code inside of my `lane_coloring()` pipeline without camera calibration.

```
global objpoints, imgpoints, src, dst, mtx, dist

# Video frame undistort
# ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img.shape[:2], None, None)
undist = cv2.undistort(img, mtx, dist, None, None)
```

However, this yields following effect on the edge of result image.



Simple uncommenting `calibrateCamera` fixed this problem right away.



Base on this test, I think the camera calibration has to be done with different `mtx` and `dist` on each frame, not reused. Of course, `objpoints` and `imgpoints` from `ChessboardCorners` could be reused for sure though.

## Pipeline (test images)

### Required

Radius of curvature values are not accurate as the appropriate pixel to meter conversion values are not based on the perspective transform. For example, when converting to real world space, `xm_per_pix = 3.7/700` is used which means that the distance between bottom of two lanes is 700 pixels. Because of the chosen source and destination points, appropriate parameter values should be selected. We are expecting radius of curvature values in hundreds for the turning part of the road and in thousands for the straight part. You can read about it at the end of this lesson in the classroom: [Link](#)

If you see in the images below from the lesson, it can be seen that bottom of left lane is at 200 and bottom of right lane is 900. The difference is 700 pixels which is used in `xm_per_pix`.

This is fixed by replacing following values.

```
ym_per_pix = 30/620  
xm_per_pix = 3.7/500
```

Note that `ym_per_pix` is also updated, because I transformed perspective in y-value, too.

Source	Destination
100, 680	300, 670
555, 460	300, 50
725, 460	980, 50
1180, 680	980, 670





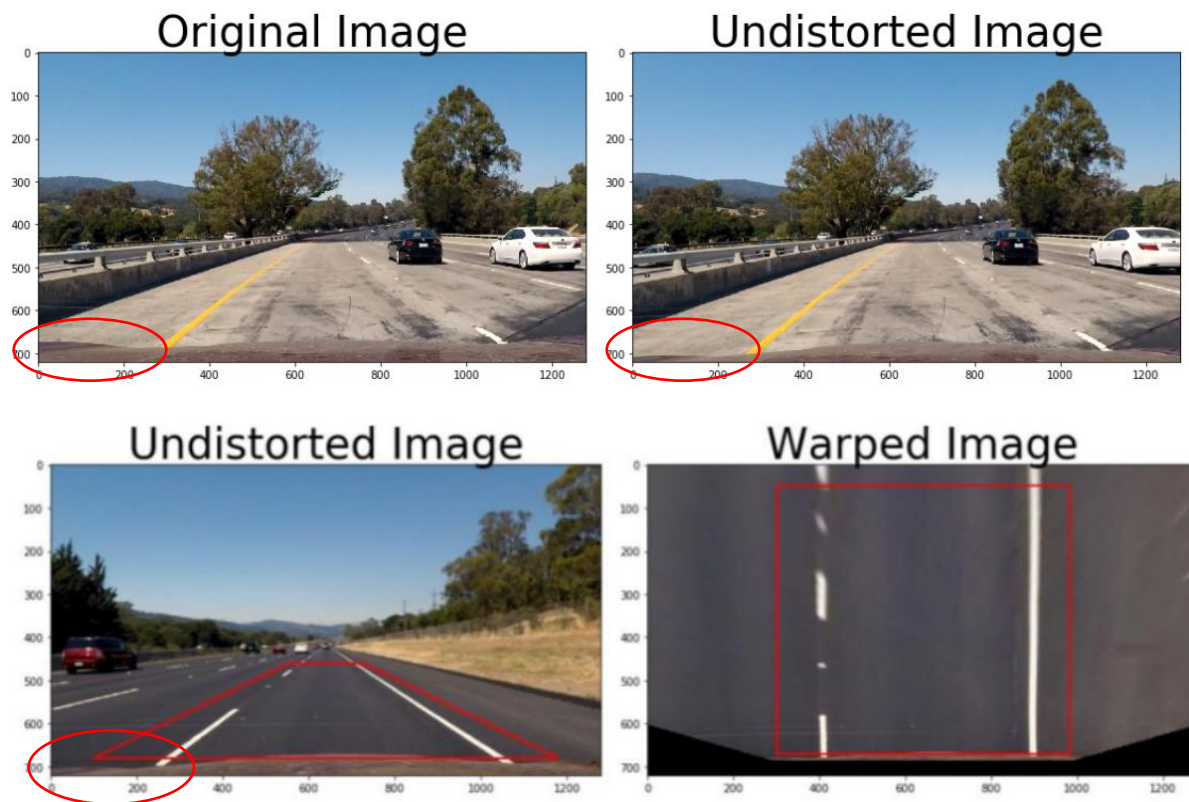
The fit from the rectified image has been warped back onto the original image and plotted to identify the lane boundaries. This should demonstrate that the lane boundaries were correctly identified. An example image with lanes, curvature, and position from center should be included in the writeup (or saved to a folder) and submitted with the project.

## Required

Lanes are drawn on the distorted image instead of undistorted one. It is obvious by looking at the car front hood. Please, make sure to undistort the image in the beginning of the pipeline, use that image for further processing and at the end lanes should also be drawn on the undistorted image not the original image.

There were some spaghetti code where I mixed up some variables, but the picture I attached in the report was actually correctly undistorted one.

One major distinction between original distorted image and undistorted image is the hood shape at the corners.



This is probably because this undistortion image is not produced from the `lane_coloring()` pipeline.





The image processing pipeline that was established to find the lane lines in images successfully processes the video. The output here should be a new video where the lanes are identified in every frame, and outputs are generated regarding the radius of curvature of the lane and vehicle position within the lane. The pipeline should correctly map out curved lines and not fail when shadows or pavement color changes are present. The output video should be linked to in the writeup and/or saved and submitted with the project.

---

## Awesome

The result video looks great! The pipeline is able to accurately map out the true locations of the lane lines throughout the entire video, and doesn't fail in the presence of shadows and pavement color changes. This is a very impressive result!

## Required

As mentioned above, lanes are drawn on the distorted frame instead of undistorted one. It is obvious by looking at the car front hood. Please, make sure to undistort the frame in the beginning of the pipeline, use that frame for further processing and at the end lanes should also be drawn on the undistorted frame not the original frame. Also, radius of curvature values are to be corrected for the video.

Once again, I fixed radius of curvature values and some variable mix up fixed as I mentioned earlier. In addition, I changed the picture of section 6 of this write-up report to update the change but nothing else.

However, I believe some of the comments regarding camera calibration issue was a reviewer's mistake.