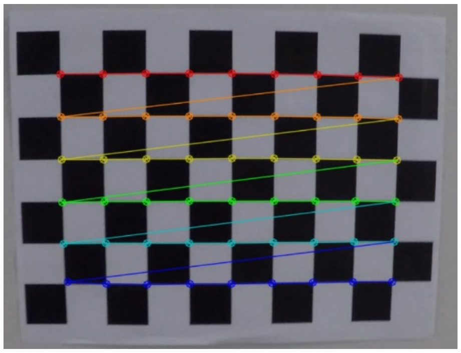
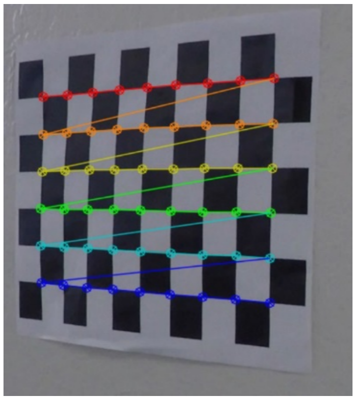
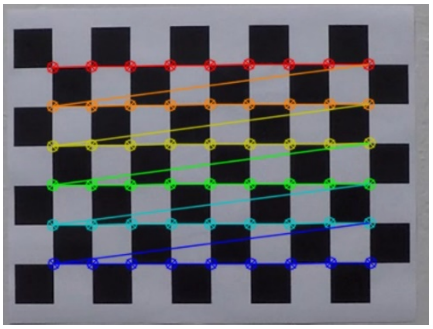
In this project, I started with writing codes in **segment 1** for finding the corners in images in camera_cal folder. I transferred the images in gray and used cv2.findChessboardCorners(gray, (9,6),None) and cv2.drawChessboardCorners(img, (9,6), corners, ret). Then I saved them in same folder by name of corners_found#.

Below are the results for some of them (corners_found10 , 11 and 17):



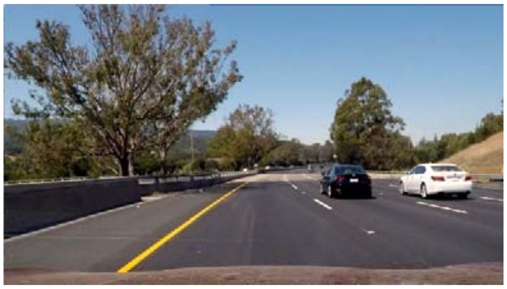




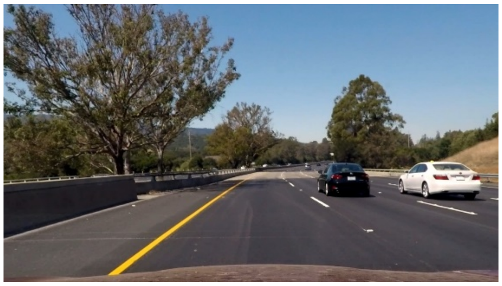
I also used cv2.calibrateCamera(objpoints, imgpoints, img_size , None, None) and then built a file calibration_pickle.b (and stored in camera_cal folder) for using in next stage for calibrating the images

Then I started **segment 2** of my project. Here I used the 6 test images in test_imagess folder (test1 to test6), and did several functions on them:

I first read the mtx and dist by loading from the pickle file I built earlier. Then I used cv2.undistort (img, mtx, dist, None, mtx) to calibrate the test1 to test6 images and stored them in same folder (in test_imagess folder) by name calibrated1 to 6. For instance, for test6 image:



test6 image before calibration



test6 calibrated image (calibrated6)

(Note: In my local computer, test6 is correlated with clibrated6. However on Udacity machine, test3 equals clibrated6 and any other image name ending with 6). Then I developed a function named abs_sobel_thresh() and Sobel operators for x and y with the following thresholds in this function:

gradx = abs_sobel_thresh(img, orient='x', thresh_min=12, thresh_max=255)

grady = abs_sobel_thresh(img, orient='y', thresh_min=25, thresh_max=255)

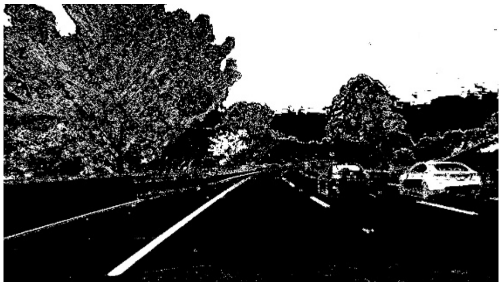
I also developed another function for color transform with following thresholds on S (from HLS) and V (from HSV) transforms:

```
 c_binary = color_threshold(img, sthresh=(100, 255), vthresh=(50, 255))
```

Inside above function, it combines S and V and returns the output.

Then I combined the results of Sobel and color thresholds into one image:

```
    processImage[((gradx ==1) & (grady ==1) | (c_binary ==1))] = 255

    result1 = processImage
```



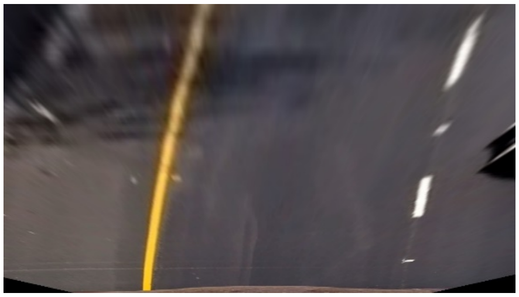
Binary6, result of sobel (x and y) and color(channels s in hls and v in hsv) transforms combined

I stored them as binary1 to binary6 in test_imagess folder.

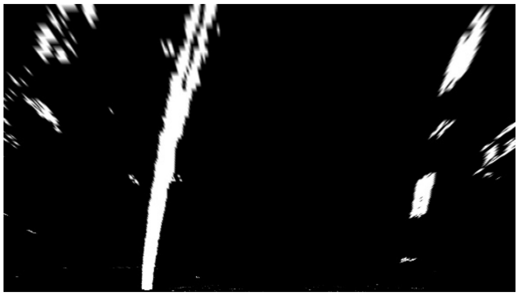
I then used cv2.getPerspectiveTransform(src, dst) and cv2.warpPerspective(img, M, img_size, flags = cv2.INTER_LINEAR) to build warped images. I used those points for src and dst:

```
src = np.float32([[(img_size[0] / 2) - 55, img_size[1] / 2 + 100],[((img_size[0] / 6) - 10), img_size[1]],

    [(img_size[0] * 5 / 6) + 60, img_size[1]],[(img_size[0] / 2 + 55), img_size[1] / 2 + 100]])

dst = np.float32([[(img_size[0] / 4), 0],[(img_size[0] / 4), img_size[1]],[(img_size[0] * 3 / 4), img_size[1]],

    [(img_size[0] * 3 / 4), 0]])
```

I did this two times for all 6 test images. One time I did to the color images(calibrated 1 to 6) after calibration(named as color_warped1 to 6), and the other time on binary image after calibration(binary_warped1 to 6). For instance for test6 image:

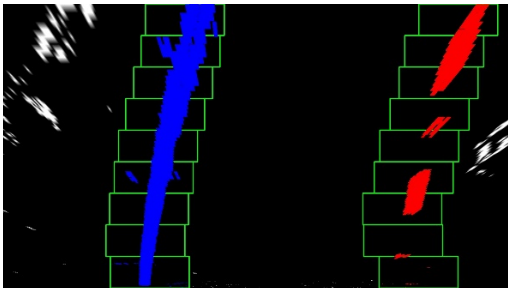


color_warped6

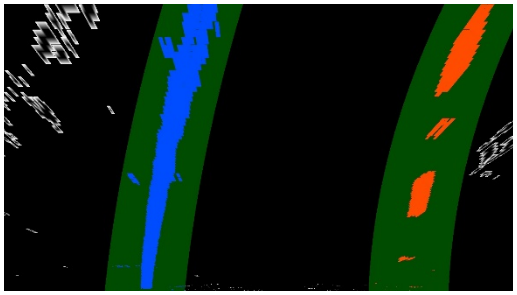


binary_warped6

Then I used the sliding method to find the white lines and mark them (named binary_warped_marked1 to 6). Then I used fit_poly techniques to fit poly in the white lines in left and right and named the files binary_warped_fitpoly1 to 6(I used functions such as find_lane_pixels, fit_polynomial, search_around_poly in order to do those). For instance, results for test6 image:

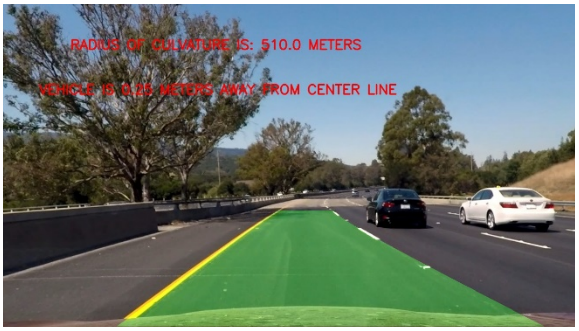


binary_warped_marked6



Binary_warped_fitpoly6

I then wrote codes to calculate the left, right, and center curvature radius, and the deviation between vehicle and the center line. I also wrote codes to transfer the images into real world (by using Minv) and mark the area between the two lines green (after finding the white lanes and fit poly in left and right lines mentioned earlier above). I also put texts about curvature radius and vehicle deviation from the center line on images. I named them as calibrated_marked_texted1 to 6. For instance, for test6 image:



RADIUS OF CULVATURE IS: 510.0 METERS

VEHICLE IS 0.25 METERS AWAY FROM CENTER LINE

calibrated_marked_texted6

Finally, I wrote codes to develop a function named Process_image(codes written in **segment 3**) containing several functions I used and explained above, and ran the project_video in test_video folder through it and got the result, then saved it in test_video_output folder (codes written in **segment 4**)

It looks that it is working nicely and gently. :-)