####1. Briefly state how you computed the camera matrix and distortion coefficients. Provide an example of a distortion corrected calibration image.

As shown in cell 3 I use the opencv built in camera calibration routines to undistort the image. First I calculate the chessboard corners and use them to calculate the matrix and distortion coefficients. An undistorted image sample is also shown.

###Pipeline (single images)

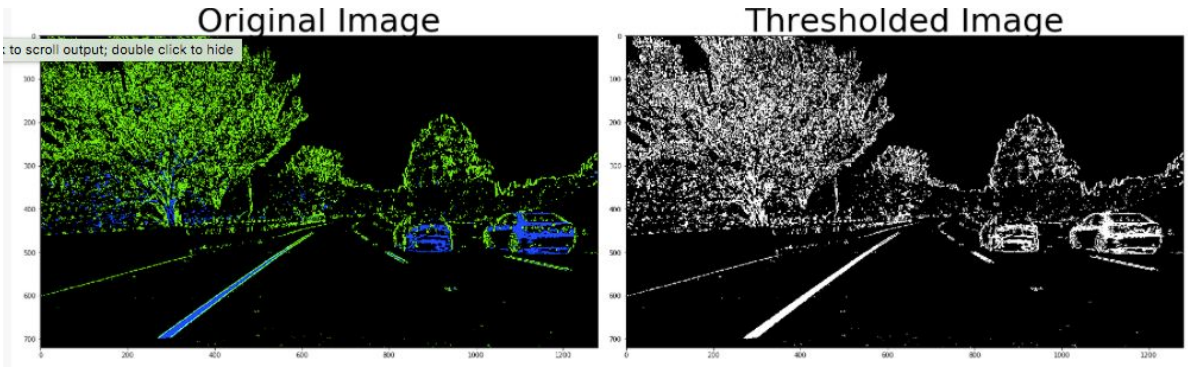
####1. Provide an example of a distortion-corrected image.

To demonstrate this step, I have applied the above function to the test image. Test6.jpeg. The result is in cell 15

####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.

The steps to this are in the respective function:

Abs_sobel_thresh which is in cell 10. I have shown both a color binary and a combined thresholded binary of the same test image test6.jpeg



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

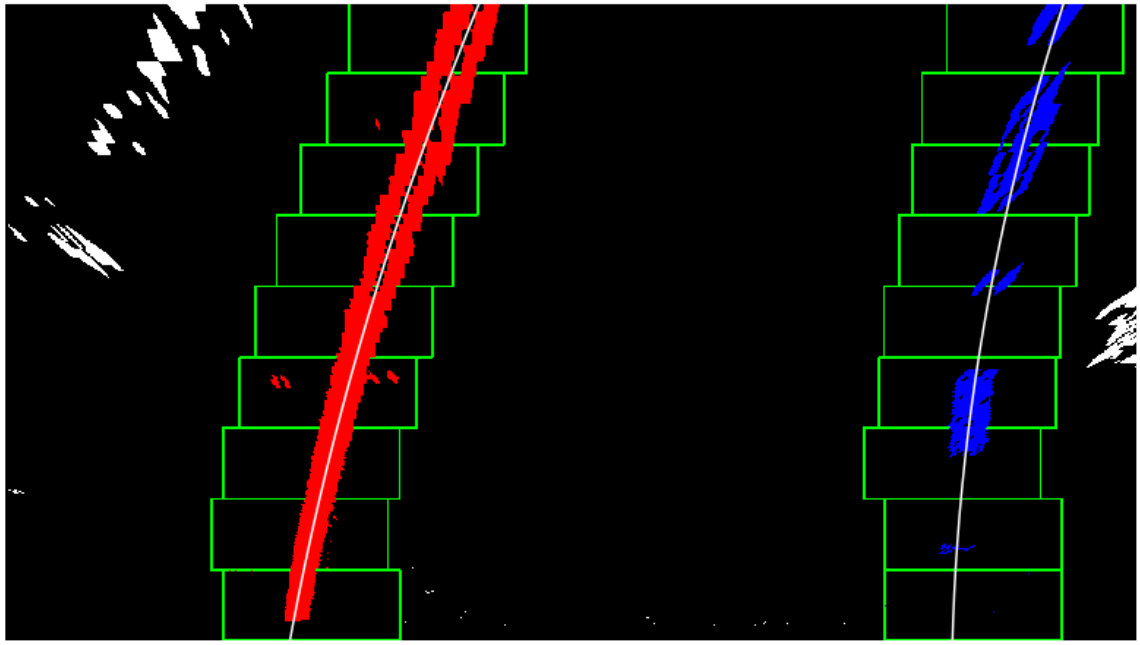
FIrst I obtained the source and destination points as shown in the function. I tested this function with the straight_lines1.jpg image. The results are shown in Cell 12.

I just used hard-coded values for src by just figuring out the points on the image by trial and error. Drawing the image helped in this regard. For the destination points I used some variation of the formula provided in the README guidelines

I used these in the warp2_image function as shown in the cell 54. Using the opencv built in routines to get the warped image.

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

Fitting the line to the polynomial is accomplished in the find_lane_lines function. I've fit the lines to a second order polynomial using the numpy polyfit function. I use the polyfit function twice for the left and right co-ordinates. Then I use the numpy linespace function to get the range of points for plotting. This is shown in the output at cell 61



####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.

This is accomplished in the function get_radius_of_curvature.

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

This is shown in the function draw_lanes_on_image. I've combined the radius of curvature using the find_lane_lines

###Pipeline (video)

####1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (wobbly lines are ok but no catastrophic failures that would cause the car to drive off the road!).

The video is in the output_images directory.

In my first submission, I had not stored the previous frame values in a temp class and as a result, lane detection was not correct in some of the frames. I have now corrected the same.

###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?

This was the most difficult assignment for me. There was a lot of trial and error which in finding the perspective points, which is why it took me such a long time to figure things out.

Also getting the polyfit and figuring out a second order polynomial wasn't easy. This is primarily because I've forgotten most of the math.

Shortcomings of the current approach - The lane detection, particularly fitting a polynomial is subject to the underlying road conditions as it was evident in one of the

steps. Even with the improvement of storing the previous frame values in a class, the lane finding is highly dependent on the presence of a uniform lane surface. For instance if the surface is ridden with potholes, the lane finding might not work.

I also wonder if there is a better alternative than the sliding window protocol. It’s a linear search through the image with overlaps. Is there a possibility for a random walk like approach instead of sliding windows. I am not certain yet.