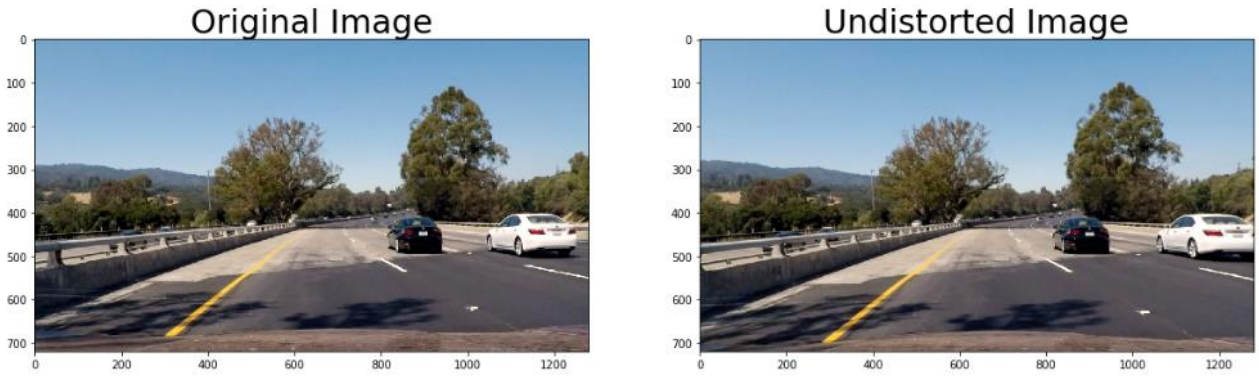
Writeup:

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

The code for this step is contained in the 2nd and 3rd code cell of the IPython notebook located in "./project.ipynb".

I start by preparing "object points", which will be the (x, y, z) coordinates of the chessboard corners in the world. Here I am assuming the chessboard is fixed on the (x, y) plane at z=0, such that the object points are the same for each calibration image. Thus, objp is just a replicated array of coordinates, and objpoints will be appended with a copy of it every time I successfully detect all chessboard corners in a test image. imgpoints will be appended with the (x, y) pixel position of each of the corners in the image plane with each successful chessboard detection.
I then used the output objpoints and imgpoints to compute the camera calibration and distortion coefficients using the cv2.calibrateCamera() function. I applied this distortion correction to the test image using the cv2.undistort() function and obtained this result:

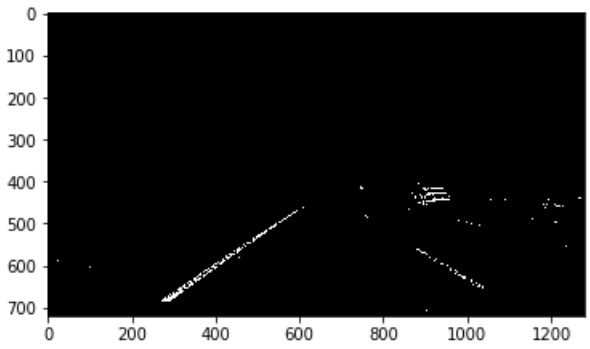


The effect of un-distortion can be seen on the white car above. The commented code saves the un-distorted versions of 6 test images in output images.

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.

I used a combination of color and gradient thresholds to generate a binary image (thresholding steps at code cells 6 and 7). Here's an example of my output for this step.

ORIGINAL IMAGE



COLOR/GRADIENT THRESHOLDED

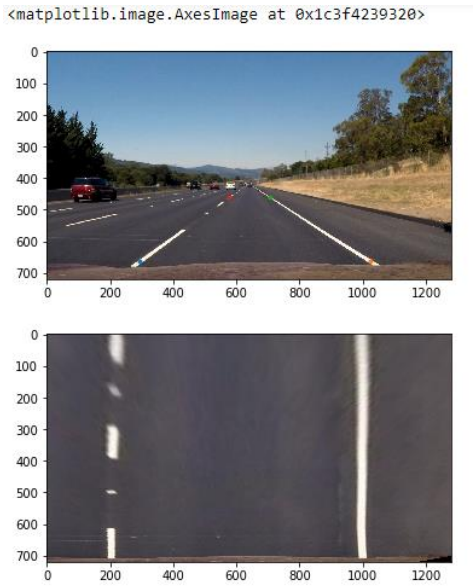
I came up with a final pipeline where I actually used a combination of sobelmag, S ,R and L. In my code (see cell 7), if any two of the four are activated, then I want the binary image to be activated. I was experimenting with different threshold values for these. The most impactful were S and L. I had seen L thresholds of (30,80) and (100,150) and (150,200) but I finally stuck with (100,150) as it handled shadows and bright spots perfectly in the video.

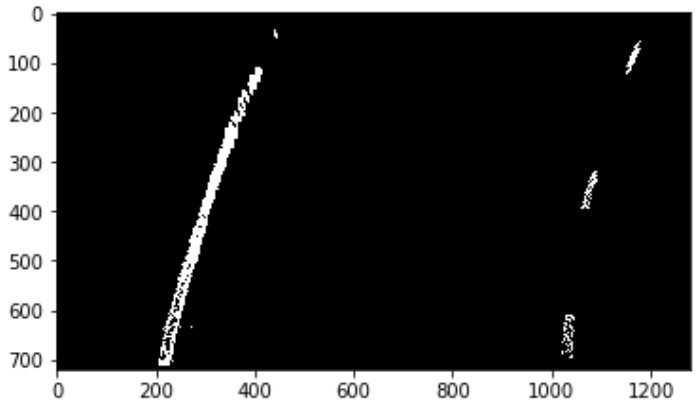
3. Describe how (and identify where in your code) you performed a perspective transform and provide an example of a transformed image.

The code for my perspective transform the 4th and 5th code cell of the IPython notebook. I chose the hardcode the source and destination points in the following manner:

```
src = np.float32([[293,663] , [1024,663] , [705, 460] , [578, 460] ])
dst = np.float32([[200,700] , [1000,700] , [1000,100] , [200,100]])
```

I verified that my perspective transform was working as expected by drawing the src and dst points onto a test image with straight lines and its warped counterpart to verify that the lines appear parallel in the warped image. The 4 colored points in the first image have been mapped to a rectangle in the second image to set the perspective for the camera.

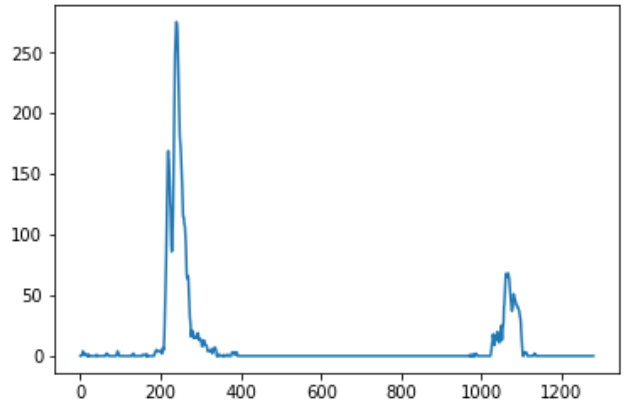




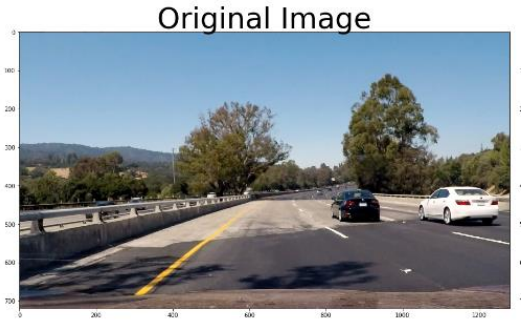
PERSPECTIVE TRANSFORM OF BINARY IN PREVIOUS STEP

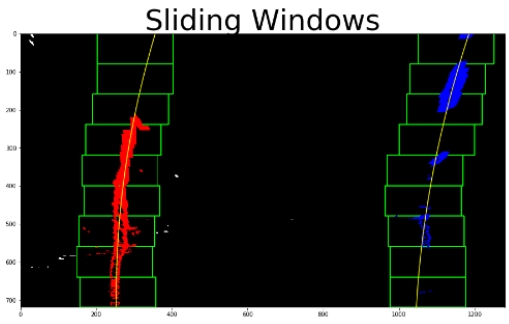
4. Describe how (and identify where in your code) you identified lane-line pixels and fit their positions with a polynomial?

The code for this section can be found primarily in code cell 9. I am also keeping of some important information about each line by using python classes in code cell 8. The next step was to use a histogram based on where the binary activations occur across the x-axis, as the high points in a histogram are the most likely locations of the base points of lane lines.



Now that we have a histogram, we can search based off the midpoint of the histogram for two lanes bottom points. Once the function 'first_lines()' has these values, it will use sliding windows (the number of which has been kept to 9) to determine where the line most likely goes from the bottom to the top of the image.





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 also is in the code cell 9 in the first_lines() function. The radius of the curvature would be low if there is a curve and would be high if the lane is straight. Theoretically for straight lane the radius should be close to infinity. It has been calculated similar to how was taught in the classroom modules. For offset calculation I calculate the polynomial fit in that space and used the average of the two lines as my lane curvature. Then calculated the difference in the image center which I assumed to be the car's center and these statistics are being printed on the image using cv2.putText(). The conversion from pixels to meters was done beforehand.

Resulting image looks like the following:

The commented code has saved all the resultant images for test images in the output_images folder too.

Pipeline (video) can be found at https://github.com/gireek/CarND-Advanced-Lane-Lines/blob/master/reg_vid.mp4

Discussion:

Challenges:

The gradient and color threshold part was the trickiest. Even with lots of changes in thresholds of R , S and L channels the lane detection was shaking in the video when shadows or brightness came all of a sudden. I came up with a final pipeline where I actually used a combination of sobelmag, S ,R and L. In my code (see cell 7), if any two of the four are activated, then I want the binary image to be activated. I was experimenting with different threshold values for these. The most impactful were S and L. I had seen L thresholds of (30,80) and (100,150) and (150,200) but I finally stuck with (100,150) as it handled shadows and bright spots perfectly in the video. The code became very hard to maintain in the radius of curvature and offset part thus I ended up including it entirely in the first_lines() function.

Improvements:

The code structure and usage of classes can be done in a better way. Masking of region can also be done to improve result on challenge videos but I skipped it because it seems to be borderline hard coding to me. I think deep learning can help more with learning where to highlight because when some pavements come in middle or the lane itself is colored differently in half like the challenge video the crude pipeline which I have currently gives an overlap of right and left lane prediction which tend to look like a sine curve! Also other cars detection will be an added advantage which I hope to learn in the next project.

Conclusion: I am pretty content with the video here and learnt a lot in this project.