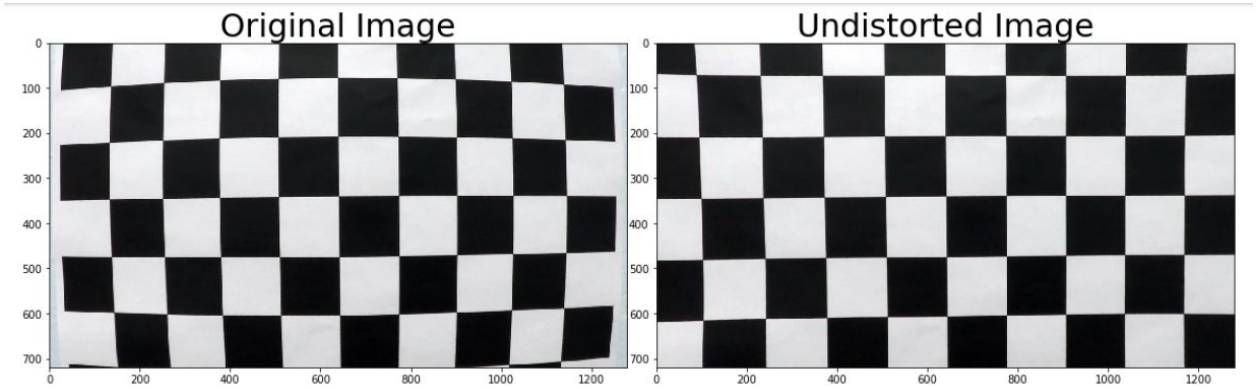
Camera Calibration

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 to 3rd cells of the IPython notebook located

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:



Pipeline (single images)

2. Provide an example of a distortion-corrected image.

To demonstrate this step, I will describe how I apply the distortion correction to one of the test images like this one:

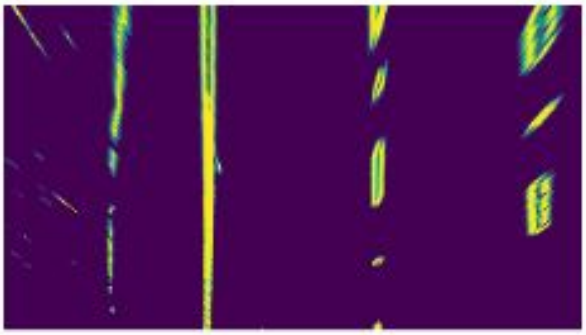
I converted this image to the following picture using several thresholding functions: sobel x, sobel y and magnitude and direction of gradients. Also, I convert this image to S space and threshold the color channel, the details code can be found in the cell 9th.

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

Then I unwarp the image using cv2.undistort function cv2.getPersepctiveTransform and cv2.getPerspectiveTransform function. The parameters of dist and src I chosen are from the codes below:

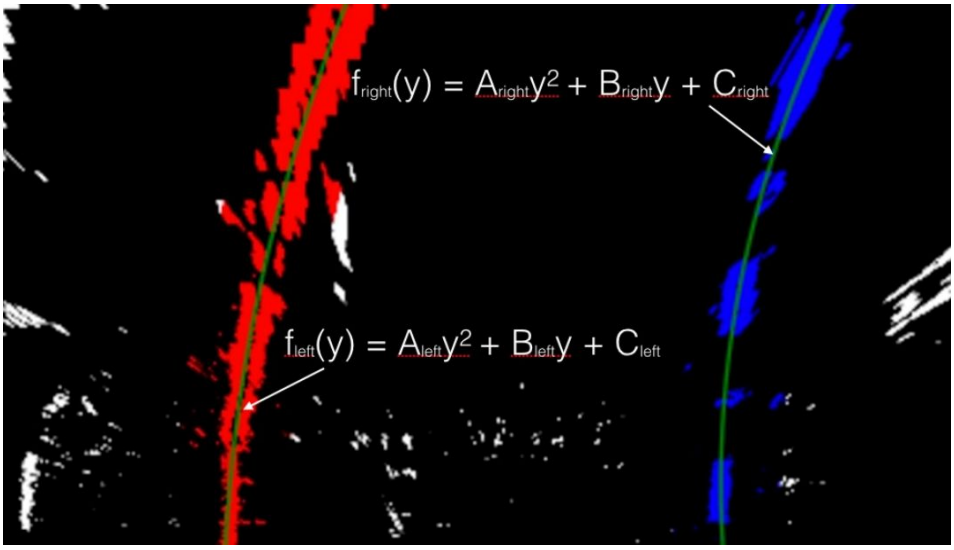
```
h, w = undst_img.shape[:2]
src = np.float32([(575, 464),(707, 464), (258, 682), (1049, 682)])
dst = np.float32([[450,0], [w-450, 0], [450, h], [w-450, h]])
```

The image being unwarped is below:



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

Then I did some other stuff and fit my lane lines with a 2nd order polynomial kinda like this:



I did this with the codes below:

```
def Find_Curvature(left_fit, right_fit, y):

    # Define conversions in x and y from pixels space to meters

    ym_per_pix = 30/720 # meters per pixel in y dimension

    xm_per_pix = 3.7/700 # meters per pixel in x dimension
```

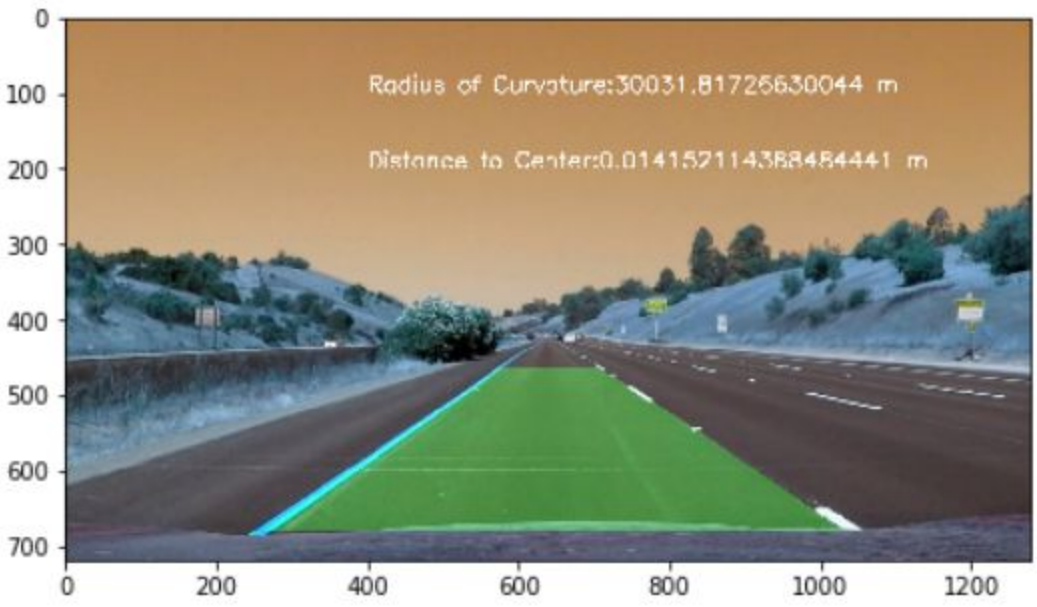
```
    ymal = np.max(y)

    left_Rad = ((1 + (2*left_fit[0]*ymal*ym_per_pix +
left_fit[1])**2)**(3.0/2))/(2*np.absolute(left_fit[0]))

    return left_Rad
```

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

I implemented this step in the function draw_lines. I use function cv2.putText to put the curvature radius to the original image. Here is an example of my result on a test image:



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!).

I uploaded the video in the zip folder

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?

The problems I faced in this project is the robustness of detecting the lines. Because the right lanes are dash lines, so the Pipeline would sometimes fail to detect a decent line.

In that case, I apply some sanity checks. For example, I constrained the difference of the current fit and the last fit in certain ranges that it would enforce the program to drop the bad fit that is not consistent. I also apply parallel checks and left to right lane distance check.

At the sametime, I would average the most current fits to make the lines more smooth.

With the checks I mentioned above, the fit to the lanes would be more smooth and more consistent throughout the video.