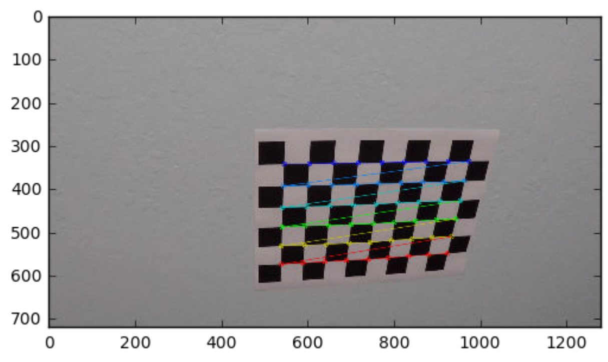
Project 4 - Advanced Lane Finding Project

The goals / steps of this project are the following:

- •Compute the camera calibration matrix and distortion coefficients given a set of chessboard images.
- •Apply a distortion correction to raw images.
- •Use color transforms, gradients, etc., to create a thresholded binary image.
- •Apply a perspective transform to rectify binary image ("birds-eye view").
- •Detect lane pixels and fit to find the lane boundary.
- •Determine the curvature of the lane and vehicle position with respect to center.
- •Warp the detected lane boundaries back onto the original image.
- •Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position

Step 1 and Step 2 in provided **Python Notebook** will **import libraries** needed for this project and defining the **Class Line** with functions including storing radius of curvature, various cordinates values, Left fits and right fits.

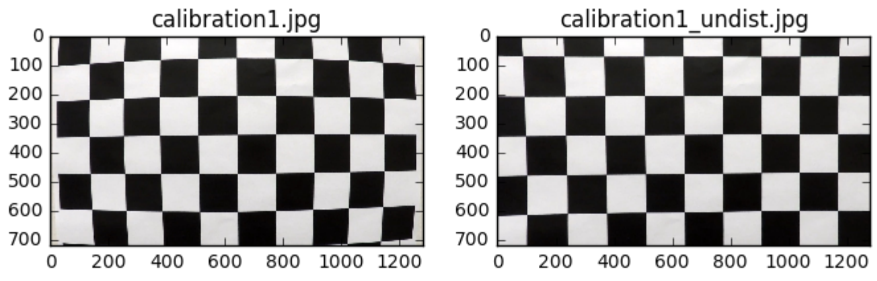
Step 3 in the python notebook is about **finding and drawing chess board corners** using functions from cv2. One such result shown below

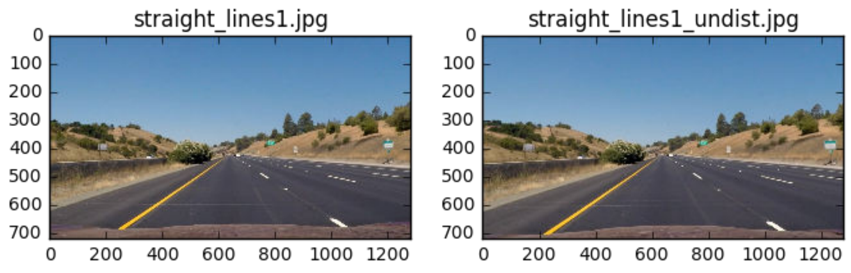


Above is followed by performing **camera calibration and undistortion** on test images using below functions from cv2-

```
# calibrate camera

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, img_size, None,None)

 # undistort image

dst = cv2.undistort(img, mtx, dist, None, mtx)
```

Result of images are stored in **output_images folder** which is provided with the project. Also I am showing few results here -



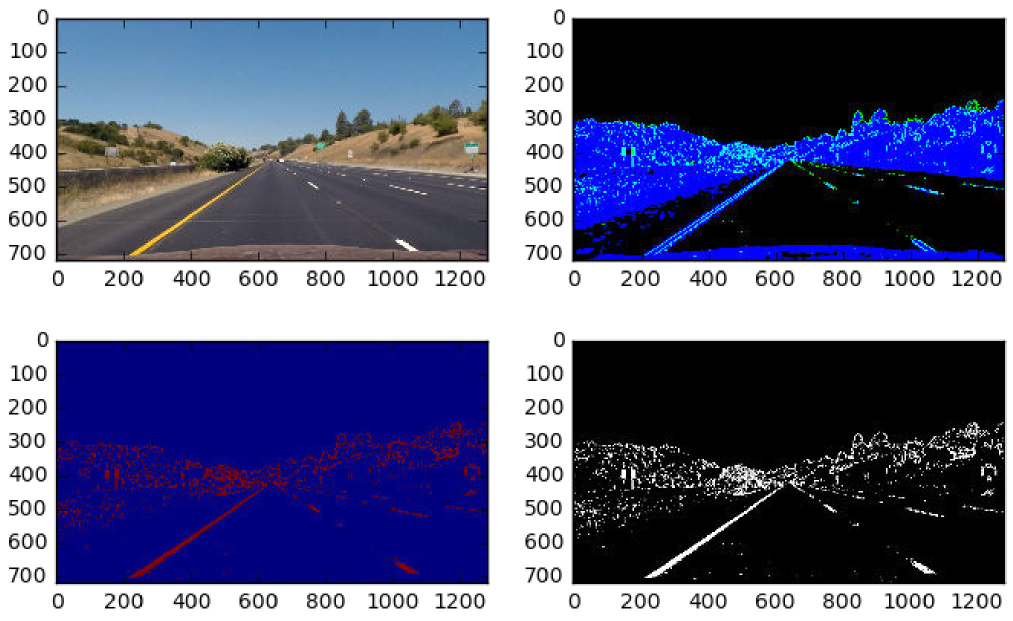


Camera matrix and distortion coefficients are also saved into pickle file using following commands-

```
# Save the camera calibration result for later use (we won't worry about rvecs / tvecs)

dist_pickle = {}

dist_pickle["mtx"] = mtx

dist_pickle["dist"] = dist

pickle.dump( dist_pickle, open( "project4_pickle.p", "wb" ) )
```

Step 4- I used a combination of color and gradient thresholds to generate a binary image (`explained in step 4 in python notebook`). RGB image is converted into HLS as S channel in HLS image is performing better during different lighting conditions. Sobelx transfrom is used to obtained vertical lines. Then both these transfroms are combined , and together they are showing better results.

After lot of trial and error, **S channel threshold is taken in the range (150,255) and Sobelx range is chosen as (20,255)**. Some output results are shown below-

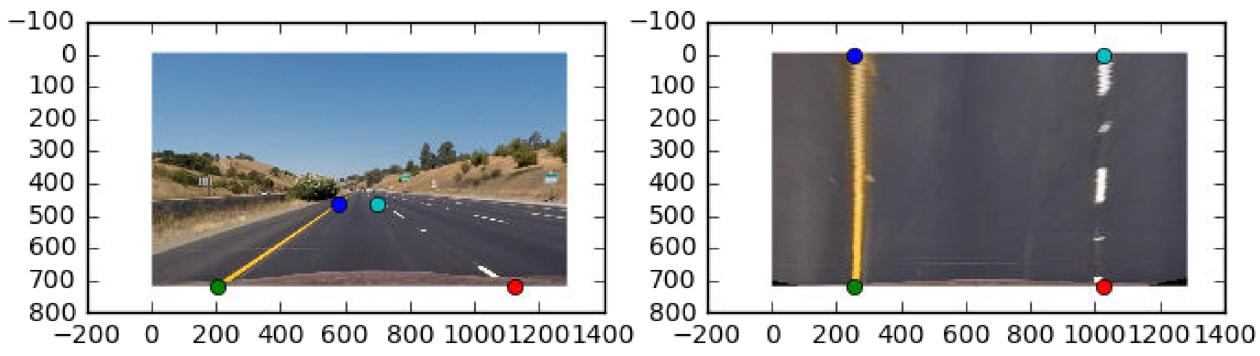


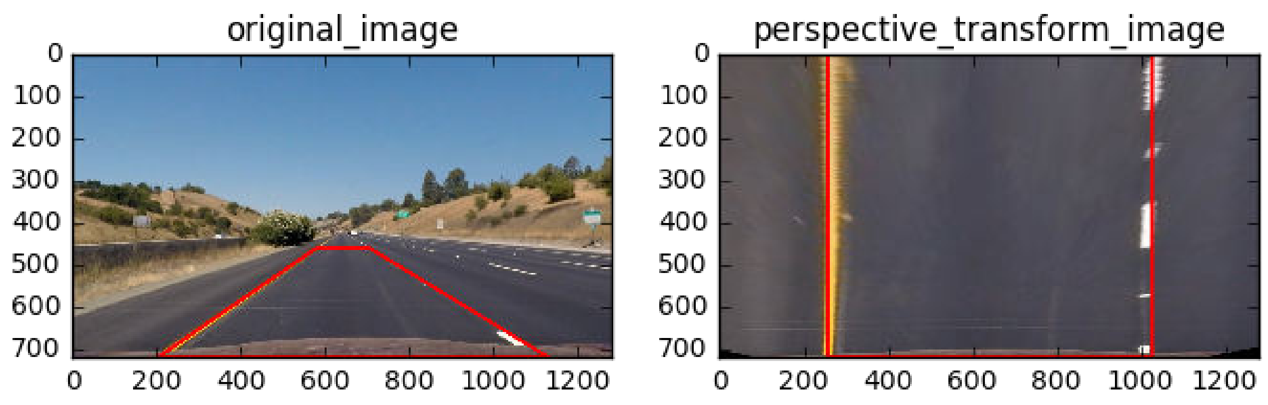
Top left image is original image, **Top right** is color binary image obtained after preforming color and gradient transfrom and then combining them. **Bottom left** in binary image of top right image. **Last, bottom right** is gray binary image. Code for obtaining above outputs are provided in step 4 in python notebook

Step 5- The code for perspective transform includes a function called `warper()`. The `warper()` function takes as inputs an image (`img`), as well as source (`src`) and destination (`dst`) points. I chose to hardcode the source and destination points in the following manner:

```
######    for simple video and all other test images    ############
src = np.float32(
    [[(img_size[0] / 2) - 60, 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 + 60), img_size[1] / 2 + 100]])
dst = np.float32(
    [[(img_size[0] / 5), 0],
    [(img_size[0] / 5), img_size[1]],
    [(img_size[0] * 4 / 5), img_size[1]],
    [(img_size[0] * 4 / 5), 0]])
```

One such result is shown below. For rest of reults for other images, see provided python notebook.





Step 6- Its about using the function to perform **sliding window** concept to identify lane-line pixels and fit their positions with a polynomial. Before that, we have to perform three type of sanity checks. First, to verify the curvature difference between both lines. Second to verify lines are parallel or not, third to verify the distance between both lanes. If all sanity checks pass, then for next frame of video we don't need to perform sliding concept, in fact we can employ **skip sliding window concept**. Code for both are given in step 6 of python notebook.

Step 7 - Calculated the radius of curvature of the lane in both pixels and meters both. Code is -

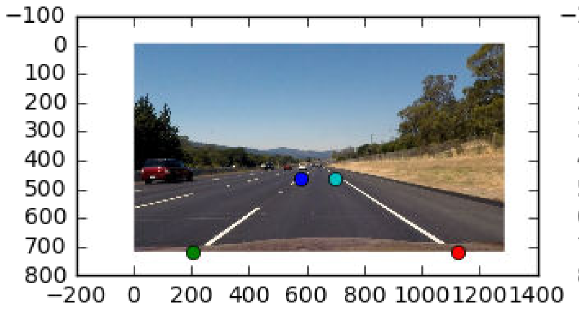
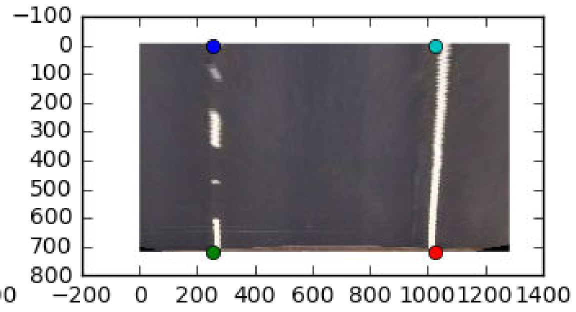
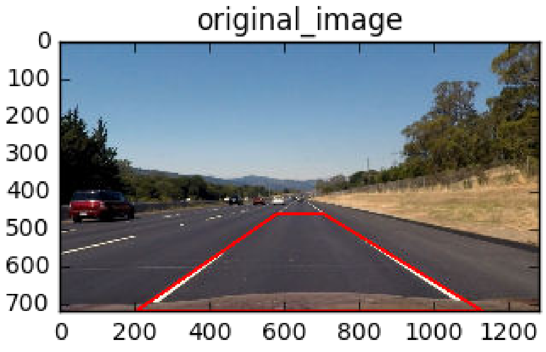
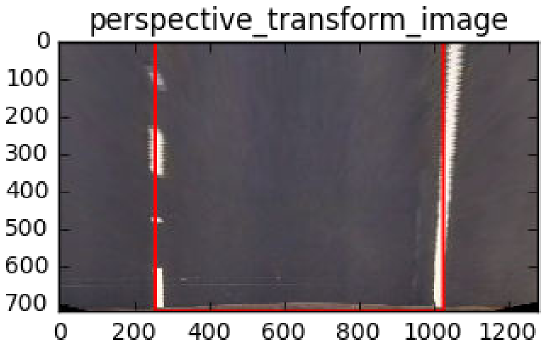
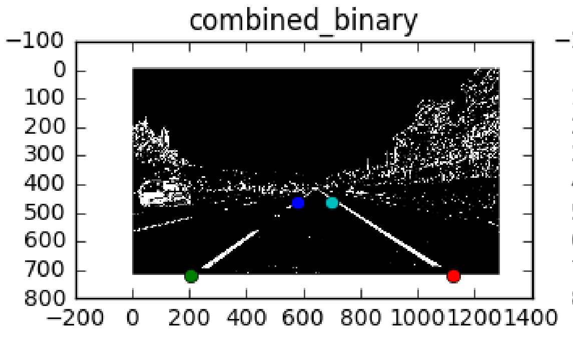
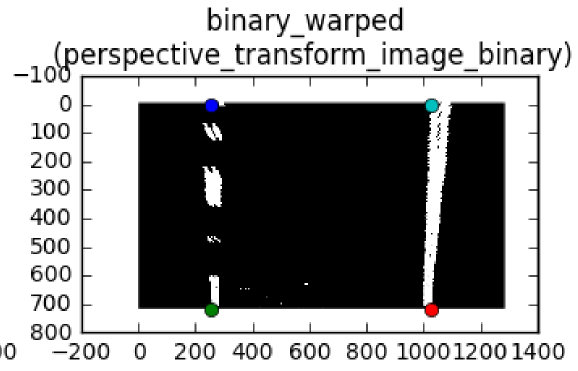
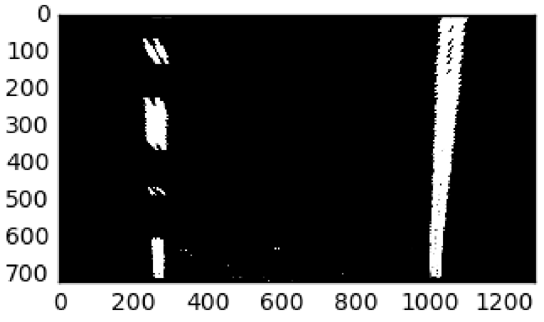
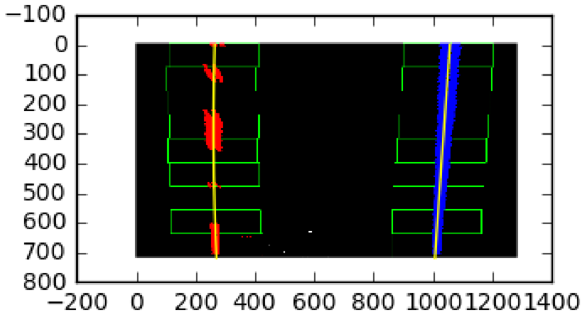
```
# Calculate the new radii of curvature
    left_curverad = ((1 + (2*left_fit_cr[0]*y_eval*ym_per_pix + left_fit_cr[1])**2)**1.5) / np.absolute(2*left_fit_cr[0])
    right_curverad = ((1 + (2*right_fit_cr[0]*y_eval*ym_per_pix + right_fit_cr[1])**2)**1.5) /
np.absolute(2*right_fit_cr[0])
    # Now our radius of curvature is in meters
    print(left_curverad, 'm', right_curverad, 'm')
    # Example values: 632.1 m    626.2 m
```

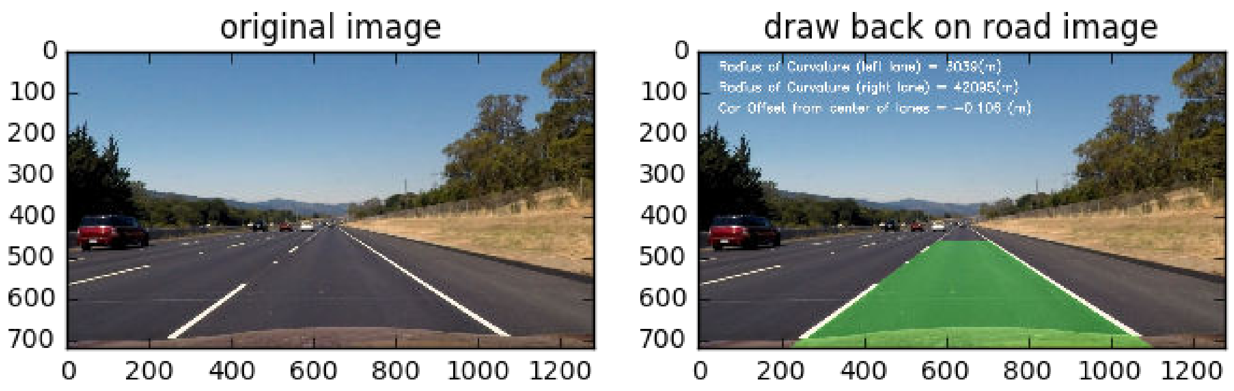
Step 8- Drawing Colored polygon enclosing road lane

In this part of code, once all the functions are performed on images, like sanity checks, obtaining polygon dimensions for lanes on road,etc, then resultant polygon is drawn on original road image

Step 9 & Step 10 - In this step, I have written code displaying step by step images obtained after various stages from starting to the end. I am also showing one set of examples below. For rest all, check provided python notebook output at step 9. Also, step 10 is used to obtain **car offset from center of lanes,** result of which is shown on top on image.

`output_images/straight_lines2_undist.jpg`




original_image

perspective_transform_image

combined_binary

binary_warped
(perspective_transform_image_binary)




original image
draw back on road image
Radius of Curvature (left lane) = 3039(m)
Radius of Curvature (right lane) = 42095(m)
Car Offset from center of lanes = -0.106 (m)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

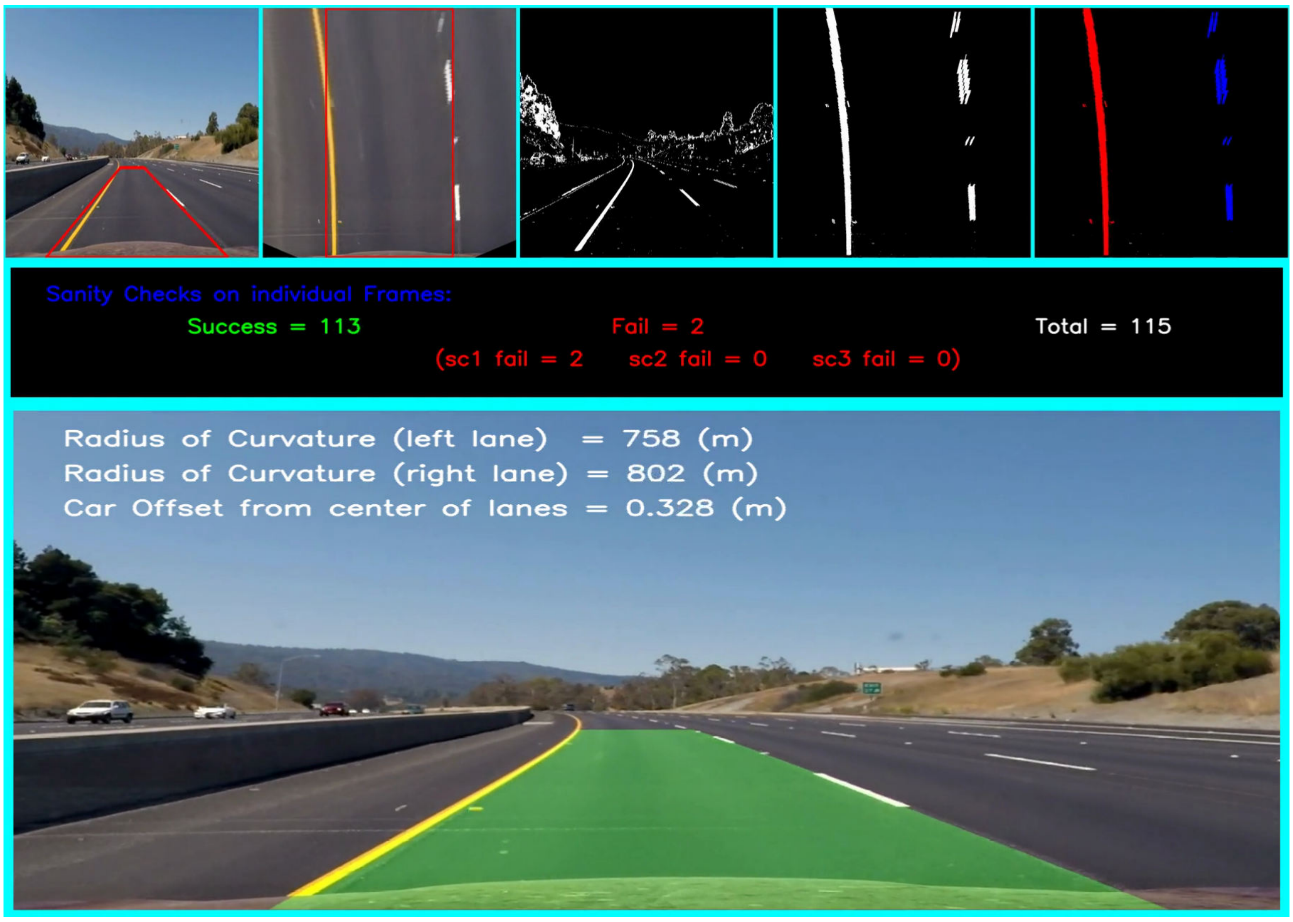
Step 11- Various type of sanity checks performed on road images

This step is about defining code for three type of sanity checks performed on images. Code is shown in step 11 and is self explanatory.

Sanity_checks_all function is used to combine all the above three sanity checks boolean results.

Step 12- Function to combine many images in one Frame

In this, a function is defined so as to combine many images on single image to see the output of various stages simultaneously. It helped a lot in optimizing the paremeters quicky. You will see the results of this images from this function in video frames. One such frame is shown below-


Sanity Checks on individual Frames:
Success = 113
Fail = 2
Total = 115
(sc1 fail = 2
sc2 fail = 0
sc3 fail = 0)
Radius of Curvature (left lane) = 758 (m)
Radius of Curvature (right lane) = 802 (m)
Car Offset from center of lanes = 0.328 (m)

Step 13 and Step 14- In these steps various videos are imported which were provided by UDACITY and final function (combining all other functions) is performed to obtain final resultant video with annotated advanced lane lines. Most of the part of importing videos if from code of project 1 itself, hence self explanatory. Result of images and videos are uploaded with the project.

Also **youtube links of videos** are provided below-

Project Video Solution-
https://www.youtube.com/watch?v=o926N4CAUSs

Challenge Video Solution-
https://www.youtube.com/watch?v=EFyJRL8__YY

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 were faced by me in project video and challenge video frames where lightning was less, due to which lane detection becomes tough. HLS and Sobel transfrom parameters were optimised by trail and error, which helped a lot in improving the implentation.

The most which helped was to perform sanity checks ignoring the few frames where pipeline was not able to detect the lines efficiently.

Also plotting many images on single frame helped in observing various stages of pipeline simulateously, and made the optimising quick and easy.

The pipeline is more likely to fail where there are extreme sharp curves like in harder challenge video. I am exploring various other color spaces and some other methods to get improved versions of this.