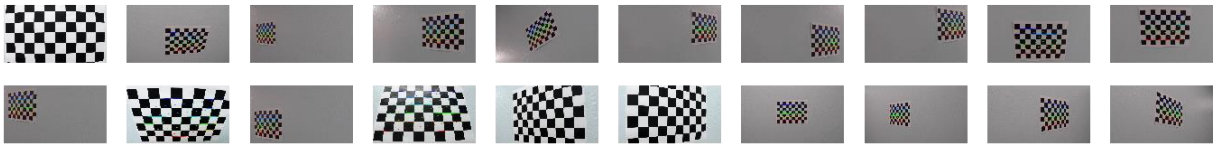
This problem started out by trying to improve the Lane line detection from the first project. So the first technique that needed to be employed was that of calibrating the camera. This was done by using the chessboard images included with this project. The images were first converted into grayscale and then used to a function of open cv to find the corners of the squares on the chessboard. See images in the jupyter notebook.

CODE:

```
images = glob.glob("camera_cal/calibration*.jpg")

assert len(images) == 20

plt.figure(figsize=(24, 3))

for i in range(len(images)):
    img = cv2.imread(images[i])
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    retval, corners = cv2.findChessboardCorners(gray, (nx, ny), None)
    if retval:
        cv2.drawChessboardCorners(img, (nx, ny), corners, retval)
        image_points_list.append(corners)
        object_points_list.append(object_points)
    plt.subplot(2, 10, i + 1)
    plt.axis('off')
    plt.imshow(img)
```

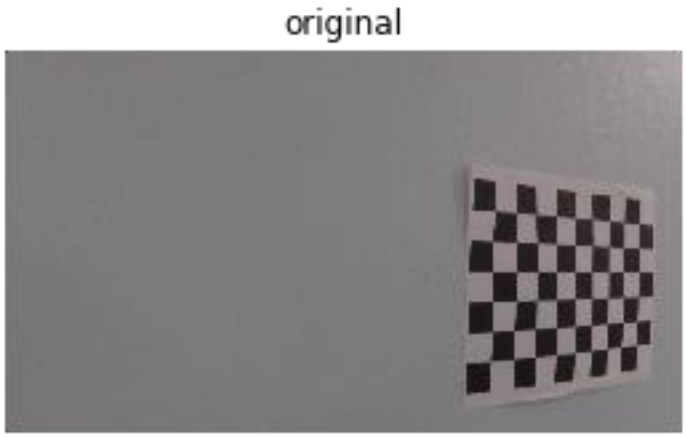


Then the camera matrix needed to be determined. After the camera matrix was determined it could then be used to undistort the images since we all know that images taken from a camera can be slightly distorted. This way images that were taken from a slightly sideways view could be adjusted to be seen as a straight facing view. See images in notebook labeled original and undistorted.

CODE:

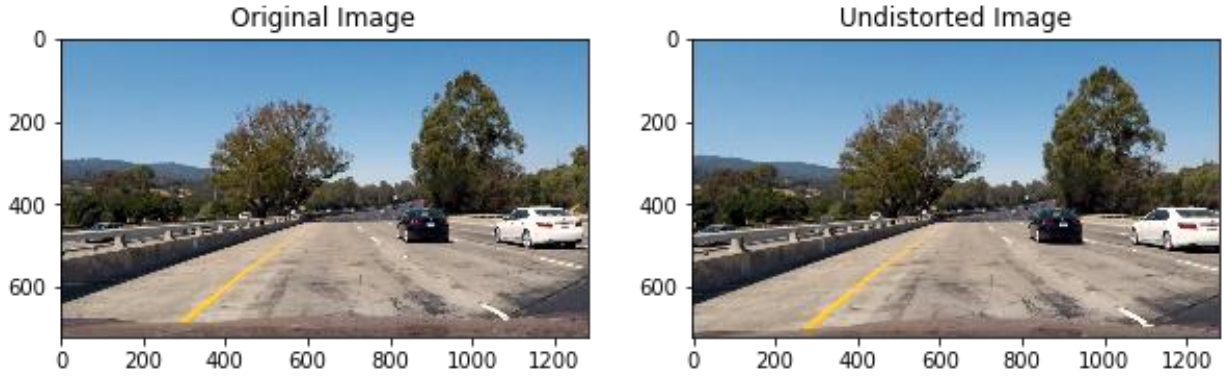
```
plt.figure(figsize=(10, 10))

img = cv2.imread(images[6])

plt.subplot(1, 2, 1)
```

```
plt.title('original')

plt.axis('off')

plt.imshow(img)
```




```
img_size = (img.shape[1], img.shape[0])

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(object_points_list, image_points_list, img_size, None, None)

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


plt.subplot(1, 2, 2)

plt.title('undistorted')

plt.axis('off')

plt.imshow(img);
```

Next another function of warping an image was used to get a perspective image of the road. See images titled undistorted and warped.

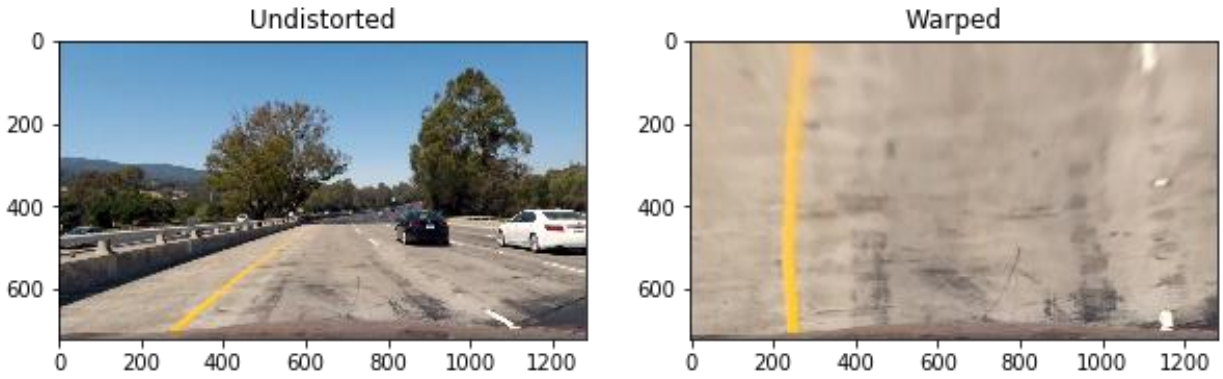



The next step need was to use the warped image to determine the lane lines.

CODE:

```
def perspective_transform(img, display=True, read = True):
    if read:
        undist = undistort(img, show = False)
    else:
        undist = undistort(img, show = False, read=False)
    img_size = (undist.shape[1], undist.shape[0])


    src = np.float32([[492, 485],[805, 485],
                      [1245, 720],[42, 720]])
    dst = np.float32([[0, 0], [1280, 0],
                      [1250, 720],[40, 720]])
    M = cv2.getPerspectiveTransform(src, dst)
    warped = cv2.warpPerspective(undist, M, img_size)
    if display:
        plot_2((undist, warped), ('Undistorted', 'Warped'))
    else:
        return warped, M
    for image in glob.glob('test_images/test*.jpg'):
```

perspective_transform(image)

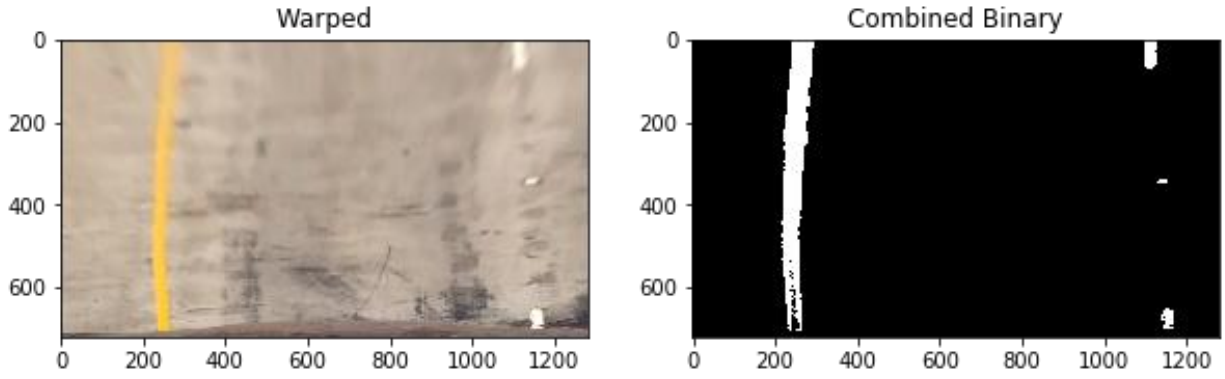



This was done by looking at the different color intensity using a threshold to determine if it was a lane line or not. This was all combined and defined as a combined binary image. See notebook examples Warped and Combined Binary.

CODE:

```
def combined_binary(image, show=True):
    img, M = perspective_transform(image, display = False)

    s_chan = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)[:,:,2]
    l_chan = cv2.cvtColor(img, cv2.COLOR_BGR2LUV)[:,:,0]
    b_chan = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)[:,:,2]

    # Threshold color channel
    s_min = 181
    s_max = 255
    s_bin = np.zeros_like(s_chan)
    s_bin[(s_chan >= s_min) &
          (s_chan <= s_max)] = 1

    b_min = 153
    b_max = 202
```

```
    b_bin = np.zeros_like(b_chan)
    b_bin[(b_chan >= b_min) &
          (b_chan <= b_max)] = 1

    l_min = 220
    l_max = 255
    l_bin = np.zeros_like(l_chan)
    l_bin[(l_chan >= l_min) &
          (l_chan <= l_max)] = 1

    combined_bin = np.zeros_like(s_bin)
    combined_bin[(l_bin == 1) | (b_bin == 1)] = 1

    if show == True:
        f, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))
        ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        ax1.set_title('Warped')
        ax2.imshow(combined_bin, cmap='gray')
        ax2.set_title('Combined Binary')
    else:
        return combined_bin

for image in glob.glob('test_images/test*.jpg'):
    combined_binary(image)
```




Where the color of the road has been remove since this is a grayscale image and checked for color intensity. The next step was to fit a polygon to where the predicted lane lines were this will help to determine the calculations, by knowing the size of the polygon the program can then calculate where it is in relation to the image and with that information it can then calculate the curvature of the road. In order to get an accurate value of the curvature of the road it is displayed on the screen every four frames, also a sliding search is used to make sure the lane lines are accurately being determined.

CODE:

```
def fit_polynomial(image):
    combined_bin = combined_binary(image, show=False)


    rightx = []
    righty = []
    leftx = []
    lefty = []


    x, y = np.nonzero(np.transpose(combined_bin))
    i = 720
    j = 630
    while j >= 0:
        histogram = np.sum(combined_bin[j:i,:], axis=0)
        left_peak = np.argmax(histogram[:640])
        x_idx = np.where((((left_peak - 25) < x)&(x < (left_peak + 25))&((y > j) & (y < i))))
        x_window, y_window = x[x_idx], y[x_idx]
        if np.sum(x_window) != 0:
```

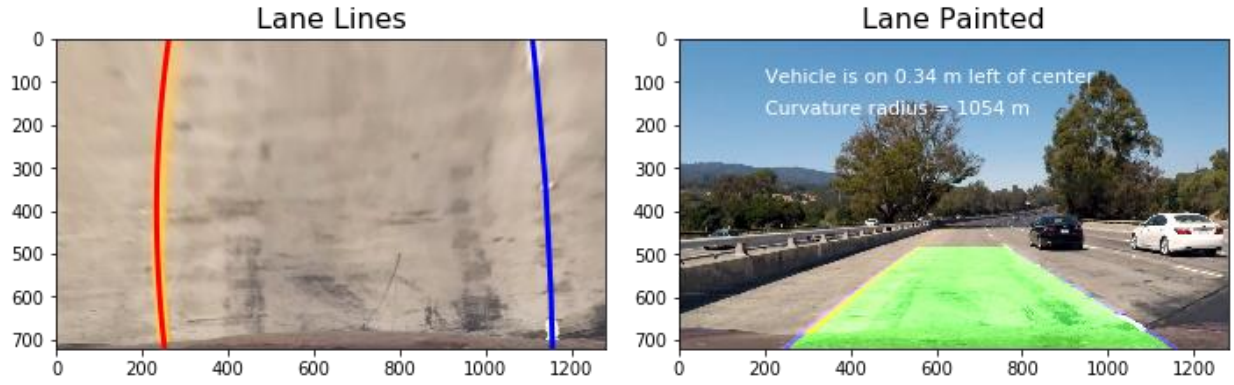
```
        leftx.extend(x_window.tolist())
        lefty.extend(y_window.tolist())

    right_peak = np.argmax(histogram[640:]) + 640
    x_idx = np.where((((right_peak - 25) < x)&(x < (right_peak + 25))&((y > j) & (y < i))))
    x_window, y_window = x[x_idx], y[x_idx]
    if np.sum(x_window) != 0:
        rightx.extend(x_window.tolist())
        righty.extend(y_window.tolist())
    i -= 90
    j -= 90

lefty = np.array(lefty).astype(np.float32)
leftx = np.array(leftx).astype(np.float32)

righty = np.array(righty).astype(np.float32)
rightx = np.array(rightx).astype(np.float32)

left_fit = np.polyfit(lefty, leftx, 2)
left_fitx = left_fit[0]*lefty**2 + left_fit[1]*lefty + left_fit[2]

right_fit = np.polyfit(righty, rightx, 2)
right_fitx = right_fit[0]*righty**2 + right_fit[1]*righty + right_fit[2]

rightx_int = right_fit[0]*720**2 + right_fit[1]*720 + right_fit[2]
rightx = np.append(rightx,rightx_int)
righty = np.append(righty, 720)

rightx = np.append(rightx,right_fit[0]*0**2 + right_fit[1]*0 + right_fit[2])
```

```
righty = np.append(righty, 0)
leftx_int = left_fit[0]*720**2 + left_fit[1]*720 + left_fit[2]

leftx = np.append(leftx, leftx_int)
lefty = np.append(lefty, 720)

leftx = np.append(leftx,left_fit[0]*0**2 + left_fit[1]*0 + left_fit[2])
lefty = np.append(lefty, 0)

lsort = np.argsort(lefty)
rsort = np.argsort(righty)

lefty = lefty[lsort]
leftx = leftx[lsort]

righty = righty[rsort]
rightx = rightx[rsort]

left_fit = np.polyfit(lefty, leftx, 2)
left_fitx = left_fit[0]*lefty**2 + left_fit[1]*lefty + left_fit[2]

right_fit = np.polyfit(righty, rightx, 2)
right_fitx = right_fit[0]*righty**2 + right_fit[1]*righty + right_fit[2]

# Radius calculation
ym_per_pix = 30./720 # meters per pixel in y dimension
xm_per_pix = 3.7/200 # meteres per pixel in x dimension

left_fit_cr = np.polyfit(lefty*ym_per_pix, leftx*xm_per_pix, 2)
```

```
right_fit_cr = np.polyfit(righty*ym_per_pix, rightx*xm_per_pix, 2)

left_curverad = ((1 + (2*left_fit_cr[0]*np.max(lefty)/2 + left_fit_cr[1])**2)**1.5) \
                    /np.absolute(2*left_fit_cr[0])
right_curverad = ((1 + (2*right_fit_cr[0]*np.max(lefty)/2 + right_fit_cr[1])**2)**1.5) \
                     /np.absolute(2*right_fit_cr[0])



# Offset from center
center = abs(640 - ((rightx_int+leftx_int)/2))
offset = 0
img_size = (combined_bin.shape[1], combined_bin.shape[0])

src = np.float32([[492, 485],[805, 485],
              [1245, 720],[42, 720]])
dst = np.float32([[0, 0], [1280, 0],
             [1250, 720],[40, 720]])
Minv = cv2.getPerspectiveTransform(dst, src)

warp_zero = np.zeros_like(combined_bin).astype(np.uint8)
color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

pts_left = np.array([np.flipud(np.transpose(np.vstack([left_fitx, lefty])))])
pts_right = np.array([np.transpose(np.vstack([right_fitx, righty]))])
pts = np.hstack((pts_left, pts_right))

cv2.polylines(color_warp, np.int_([pts]), isClosed=False, color=(0,0,255), thickness = 40)
cv2.fillPoly(color_warp, np.int_([pts]), (0,255, 0))
```

```
    warp_again = cv2.warpPerspective(color_warp, Minv, (combined_bin.shape[1],
combined_bin.shape[0]))
    finale = cv2.addWeighted(mpimg.imread(image), 1, warp_again, 0.5, 0)

    f, (ax1, ax2) = plt.subplots(1,2, figsize=(10, 6))
    f.tight_layout()

    ax1.imshow(cv2.cvtColor((perspective_transform(image, display=False)[0]), cv2.COLOR_BGR2RGB))
    ax1.set_xlim(0, 1280)
    ax1.set_ylim(0, 720)

    ax1.plot(left_fitx, lefty, color='red', linewidth=3)
    ax1.plot(right_fitx, righty, color='blue', linewidth=3)

    ax1.set_title('Lane Lines', fontsize=16)
    ax1.invert_yaxis() # to visualize images

    ax2.imshow(finale)
    ax2.set_title('Lane Painted', fontsize=16)

    if center < 640:
        ax2.text(200, 100, 'Vehicle is on {:.2f} m left of center'.
                 format(center*3.7/700),
                 color='white', fontsize=11)
    else:
        ax2.text(200, 100, 'Vehicle is on {:.2f} m right of center'.
                 format(center*3.7/700),
                 color='white', fontsize=11)
    ax2.text(200, 175, 'Curvature radius = {} m'.
```

```
        format(int((left_curverad + right_curverad)/2)),
        color='white', fontsize=11)
for image in glob.glob('test_images/test*.jpg'):
    fit_polynomial(image)
```




Although the processing of the calculations seem to be well within time I wonder if the car is going faster than the speed limit would the calculations be able to process fast enough without a high end gpu in the car. That is the only problem I can foresee in the future. Other problems could be road conditions for the program to identity the road accurately when there are weather conditions and if there are missing road lines. So one way to train the car for these problems is to use images with no lane lines and also in different weather conditions to predict where the lane lines might be. Since a car has a specific size in the video location it can be predicted where the lane lines should be. There are other information on the road images that could be used to determine the potential lane lines.