In [1]:
```
#AdvancedLaneFindingReport
```

Goal: To design a software pipeline to identify the lane boundaries from a front-facing camera on a car 1) Camera calibration 2) Color and gradient threshold 3) the Birds eye view 4) Lane detection and fit 5) Curvature of lanes and vehicle position with respect to center 6) Warp back and display information 7) Output visual display of the lane boundaries and numerical estimation of lane curvature and vehicle position. 8) Run the entire pipeline on a sample video recorded

image.png

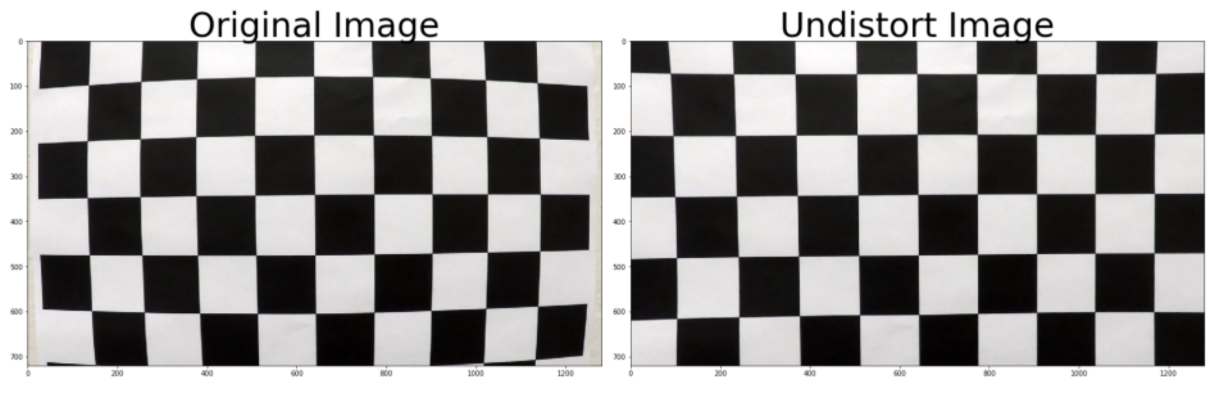
In [2]:
```
import glob
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
from    findLaneUtils import *
```

1) Camera calibration :

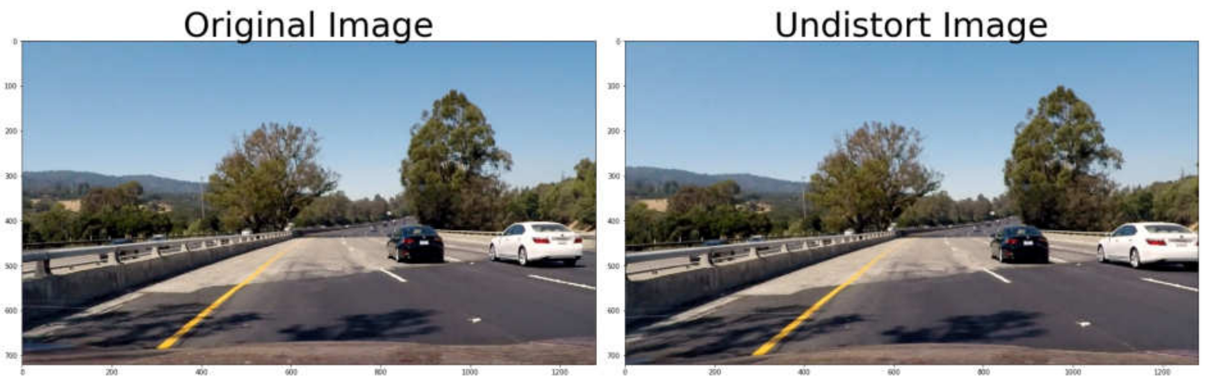
How do we compute : camera matrix and distortion coefficients. Show an example of a distortion corrected calibration image. We use images in ./calibationImages directory to do calibation. By calling CalculateCameraMatrix (calibration_dir) we get the matrix, distanceto undistort an image CalculateCameraMatrix() uses the following functiont accomplish its task: cv2.findChessboardCorners cv2.drawChessboardCorners cv2.calibrateCamera

In [4]:
```
calibration_dir = './calibrationImages/'
ret, CameraMat, distCoeff, rvecs, tvecs = CalculateCameraMatrix (calibratio
n_dir)
```

In [5]:
```
img_file = calibration_dir+'/calibration1.jpg'
img1 = mpimg.imread(img_file)
dst = cv2.undistort(img1, CameraMat, distCoeff, None, CameraMat)
plot_2_images (img1, dst, "Original Image", "Undistort Image")
```



In [6]:
```
img_file = 'prj_img/bridge_shadow.jpg'
img1 = mpimg.imread(img_file)
undistorted_img = dst = cv2.undistort(img1, CameraMat, distCoeff, None, Cam
eraMat)
plot_2_images (img1, dst, "Original Image", "Undistort Image")
```

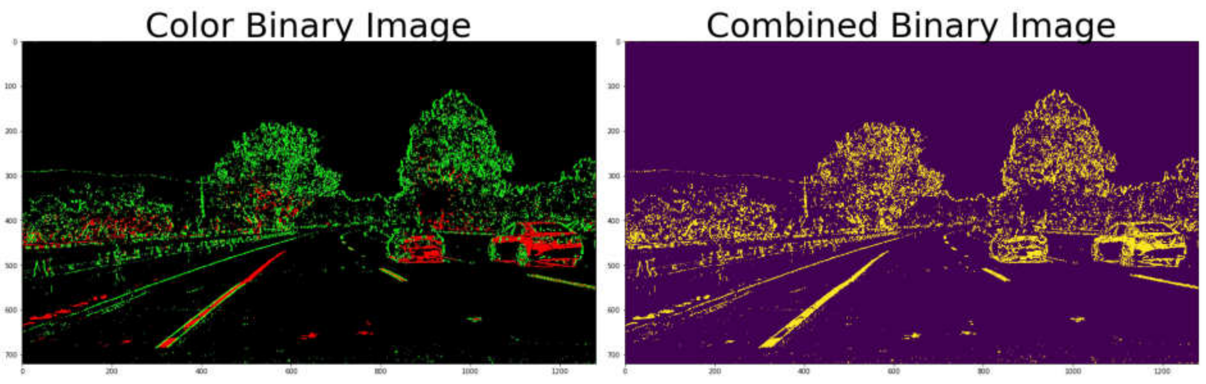


2) Color and gradient threshold.

How do we create a thresholded binary image by using color transforms, gradients Show a binary image result. The threshold_s_sx() is created by the following process Taking in an image Convert its color to gray scale Calculate value for sobelx(Gradient x) Calucalte absolute direction of gradient x compute Threshold x graident sxbinary Compute Thresthold color channel (S) Create color binary using what we calculate from sxbinary (threshold gradient x Create combined_binary from sxbinary (Threshold x gradient)

In [7]:
```
from    findLaneUtils import *
s_thresh = (170, 255)
sx_thresh= (20, 100)
img_file = 'prj_img/bridge_shadow.jpg'
img1 = mpimg.imread(img_file)
color_binary, combined_binary = threshold_s_sx(img1, s_thresh, sx_thresh)
color_binary2 = mpimg.imread('prj_img/color_binary2.jpg')
```

In [8]:
```
plot_2_images (color_binary2, combined_binary, "Color Binary Image", "Combi
ned Binary Image")
```



3) the Birds eye view

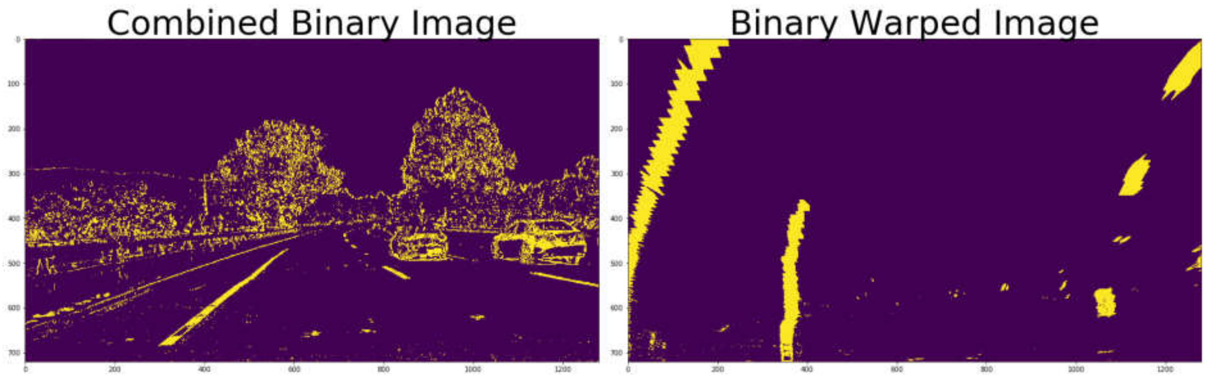
performed a perspective transform and provide an example of a transformed image. define perspective coordinates.(src, dst)

In [9]:
```
img_size = (combined_binary.shape[1], combined_binary.shape[0])
width, height = img_size
offset = 200
src = np.float32([
    [  588,   446 ],
    [  691,   446 ],
    [ 1126,   673 ],
    [  153 ,   673 ]])
```

In [10]:
```
dst = np.float32([[offset, 0],
    [(img_size[0] - offset), 0],
    [(img_size[0] - offset), img_size[1]],
    [offset, img_size[1]]])
```

In [11]:
```
M    = cv2.getPerspectiveTransform(src,dst)
Minv = cv2.getPerspectiveTransform(dst, src)
#binary_warped = cv2.warpPerspective(b_thresholded,M, (width, height))
img_size = (combined_binary.shape[1], combined_binary.shape[0])
binary_warped = cv2.warpPerspective(combined_binary,M, img_size)
```

In [12]:
```
plot_2_images (combined_binary, binary_warped,  "Combined Binary Image", "B
inary Warped Image")
```



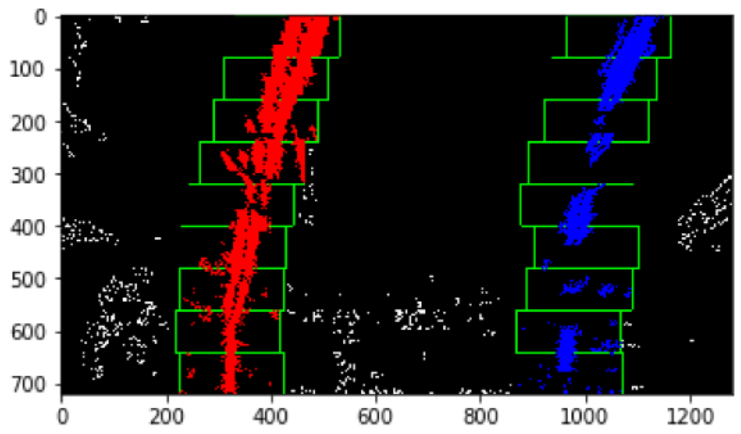
In [13]:
```
pickle_file = "AdvLaneFinding_camera_info.p"
save_camera_info (pickle_file, CameraMat, distCoeff, M, Minv)
pickle_file = "AdvLaneFinding_camera_info.p"
CameraMat, distCoeff, M, Minv = load_camera_info (pickle_file)
```

4) Lane detection and fit.

identified lane-line pixels and fit their positions with a polynomial

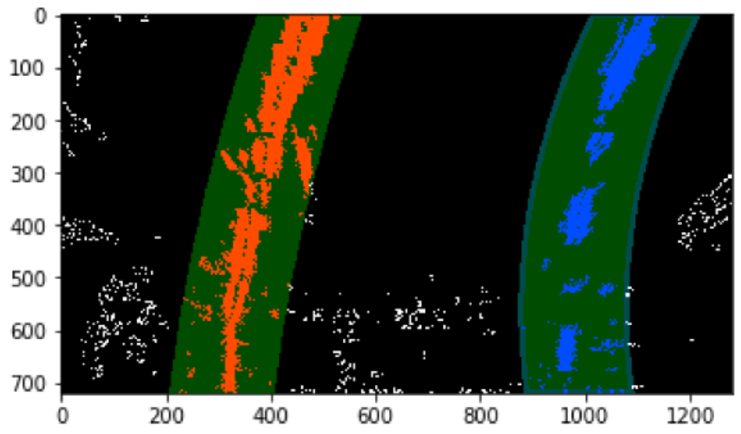
```
In [14]: %matplotlib inline
         from    findLaneUtils import *
         binary_warped = mpimg.imread('warped-example.jpg')
         out_img, left_fit, right_fit = laneDetection (binary_warped)
         plt.imshow(out_img)

Out[14]: <matplotlib.image.AxesImage at 0x7f3b8ec26908>
```



```
In [15]: out_img2,  leftx, lefty, rightx, righty, ploty = laneDetectionNext (binary_
         warped, left_fit, right_fit)
         plt.imshow(out_img2)

Out[15]: <matplotlib.image.AxesImage at 0x7f3bc5862dd8>
```



5) Curvature of lanes and vehicle position with respect to center.

calculated the radius of curvature of the lane and the position of the vehicle with respect to center.

```
In [16]: left_curverad, right_curverad = discoverCurvature (ploty, leftx, lefty, rig
         htx, righty )
         print("left curve:", left_curverad, 'm', "right curve:", right_curverad,
         'm')

         left curve: 1046.03640972 m right curve: 377.992604342 m
```

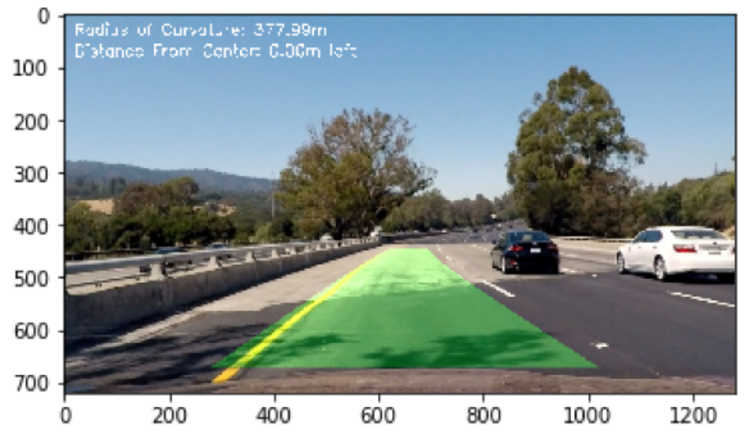
6) Warp back and

7) Display information

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

```
In [17]: #new_image = warpback_display (undistorted_img, binary_warped, left_fit, ri
         ght_fit, Minv, ploty, right_curverad )
         new_image = warpback_display (undistorted_img, binary_warped, left_fit, rig
         ht_fit, Minv,  right_curverad )
         plt.imshow(new_image)
```

```
Out[17]: <matplotlib.image.AxesImage at 0x7f3bc592e860>
```


Radius of Curvature: 377.99m
Distance From Center: 0.00m left
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

8) Video for the Driving Pipeline

```
In [1]: from findLaneUtils import *
        from proj_lane import *
        from proj_pipeline import *
        from moviepy.editor import VideoFileClip
        from IPython.display import HTML
```

```
In [2]: pickle_file = "AdvLaneFinding_camera_info.p"
        CameraMat, distCoeff, M, Minv = load_camera_info (pickle_file)
```

```
In [3]: e = DrivingPipeLine(CameraMat, distCoeff, M, Minv)
```

```
In [4]: video_output = 'project_video_output.mp4'
        video_input  = "project_video.mp4"
```

In [5]:
```
clip         = VideoFileClip(video_input)
video_clip   = clip.fl_image(e.pipeline)
%time video_clip.write_videofile(video_output, audio=False)

HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(video_output))
```

```
[MoviePy] >>>> Building video project_video_output5.mp4
[MoviePy] Writing video project_video_output5.mp4

100%|██████████| 1260/1261 [13:28<00:00,  1.52it/s]

[MoviePy] Done.
[MoviePy] >>>> Video ready: project_video_output5.mp4

CPU times: user 12min 9s, sys: 2min 21s, total: 14min 30s
Wall time: 13min 35s
```

Out[5]:

In [6]:
```
video_output = 'project_challenge_video_output.mp4'
video_input  = 'challenge_video.mp4'
d = DrivingPipeLine(CameraMat, distCoeff, M, Minv)
```

In [7]:
```
clip         = VideoFileClip(video_input)
video_clip   = clip.fl_image(d.pipeline)
%time video_clip.write_videofile(video_output, audio=False)

HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(video_output))
```

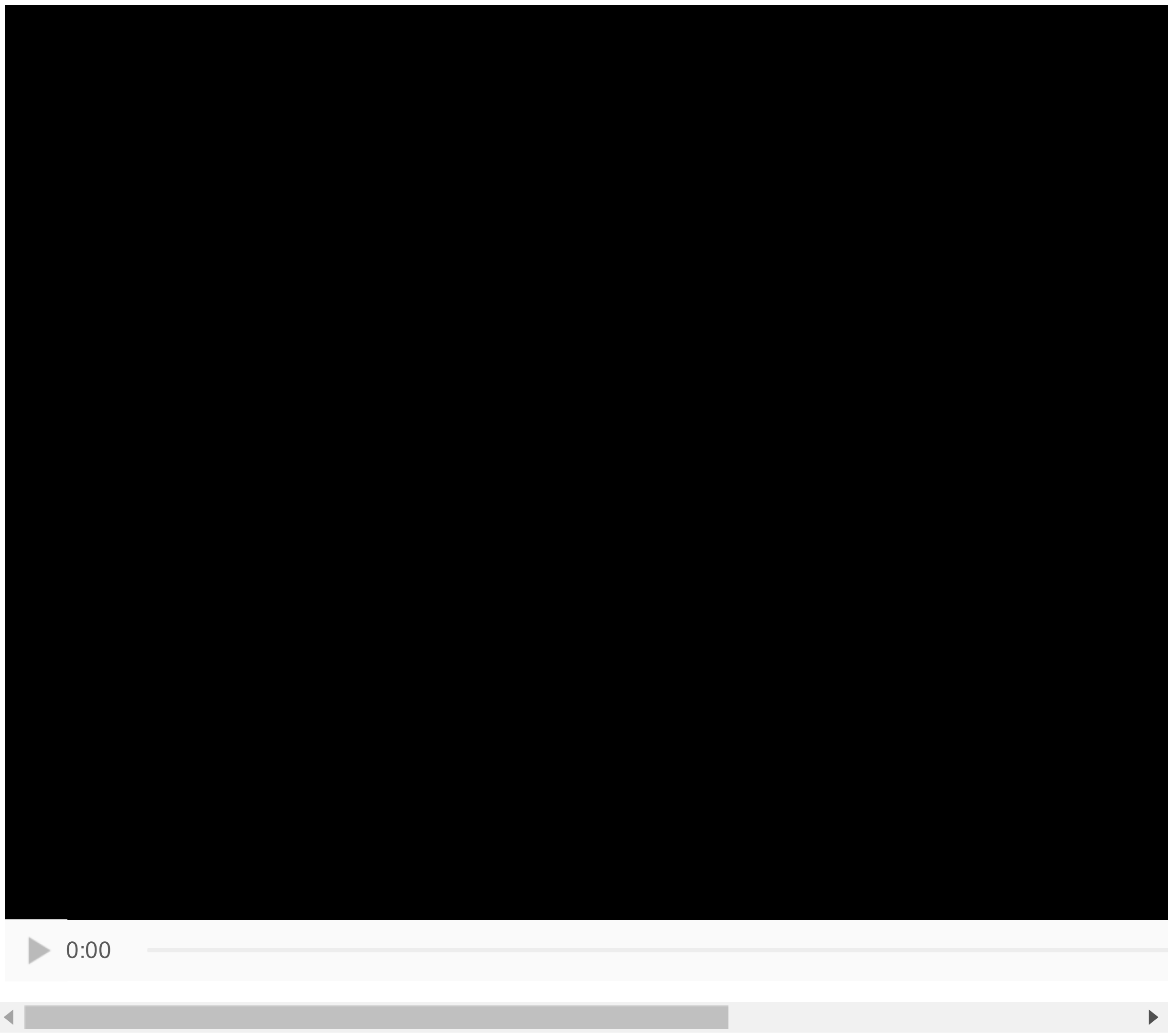
```
[MoviePy] >>>> Building video project_challenge_video_output5.mp4
[MoviePy] Writing video project_challenge_video_output5.mp4

100%|██████████| 485/485 [04:58<00:00,  1.65it/s]

[MoviePy] Done.
[MoviePy] >>>> Video ready: project_challenge_video_output5.mp4

CPU times: user 4min 28s, sys: 54.1 s, total: 5min 22s
Wall time: 5min 4s
```

Out[7]:



8) Discusssion

This pipeline design needs the following precomputed values :(CameraMat, distCoeff, M, Minv) For each image receiving from video stream: 1) Create binary_warped from (CameraMat, distCoeff, M, Minv) 2) Create undistorted image from (CameraMat, distCoeff, M, Minv) 3) The first image, we use binary-warp and try to find lane line. the end product are left_fit, right_fit Adn we update the curvature for the lane 4) For other images we calculate: eftx, lefty, rightx, righty from previous info from (left_fit, right_fit) we again update curvature for the new lane 5) We create warpback image and display lane curvature The pipeline is not very efficient because it did not have the sanity check design in the lecture. In other place in the world where we do not have lane lines at all. This pipeline will not work.

```
In [ ]:
```