CarnND Advance Lanes

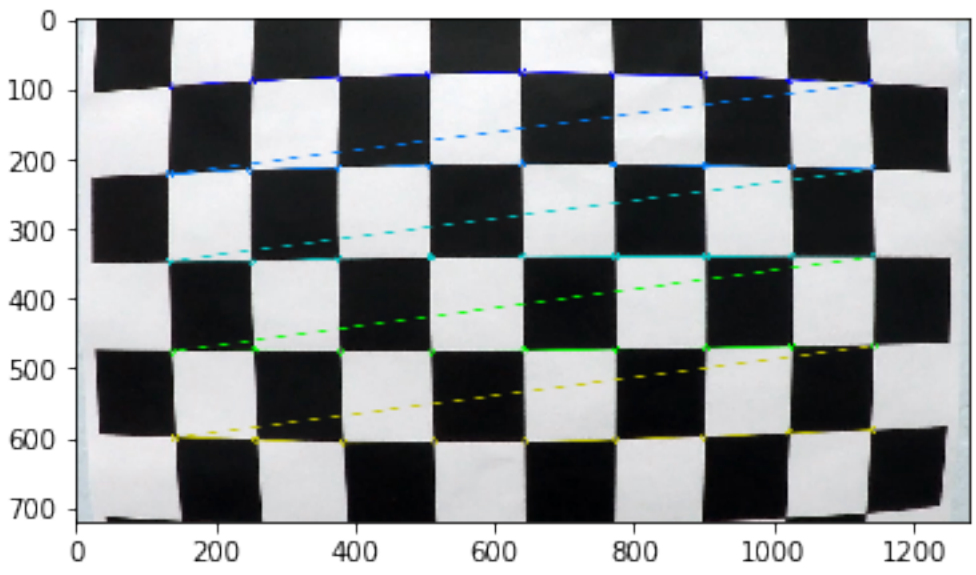
Author: Igor Passchier

Compute the camera calibration matrix and distortion coefficients

In this first step, the camera calibration matrix and distortion coefficient are calculated

In [1]:

```
%matplotlib inline

# import the libraries
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob

#first, go through the whole process once, to see the steps. Then, we will do
all images
# Number of checker bord corners in x and y
nx = 9
ny = 5

# Read 1 image
fname = 'camera_cal/calibration1.jpg'
img = cv2.imread(fname)

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find the chessboard corners
ret, imgpoints = cv2.findChessboardCorners(gray, (nx, ny), None)

# If found, draw corners
if ret == True:
    # Draw and display the corners
    cv2.drawChessboardCorners(img, (nx, ny), imgpoints, ret)
    plt.imshow(img)
else:
    print ('No checkerbord found. Wrong number of points?')
```



In [2]:

```
# Calculate distortion matrix, based on all images

images = glob.glob('camera_cal/calibration*.jpg')
objectPoints=[]
imagePoints=[]

nx=9
ny=6
objs=np.zeros([nx*ny,3],np.float32)
objs[:,:2]=np.mgrid[0:nx,0:ny].T.reshape(-1,2)

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (nx, ny), None)

    # If found, add object points, image points
    if ret == True:
        objectPoints.append(objs)
        imagePoints.append(corners)

    else:
        print ('No checkerbord found in '+fname)


#perform the calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints,
gray.shape, None, None)
###GLOBAL VARIABLES
```

```
No checkerbord found in camera_cal/calibration1.jpg
No checkerbord found in camera_cal/calibration4.jpg
No checkerbord found in camera_cal/calibration5.jpg
```

Helper functions for plotting

In [3]:

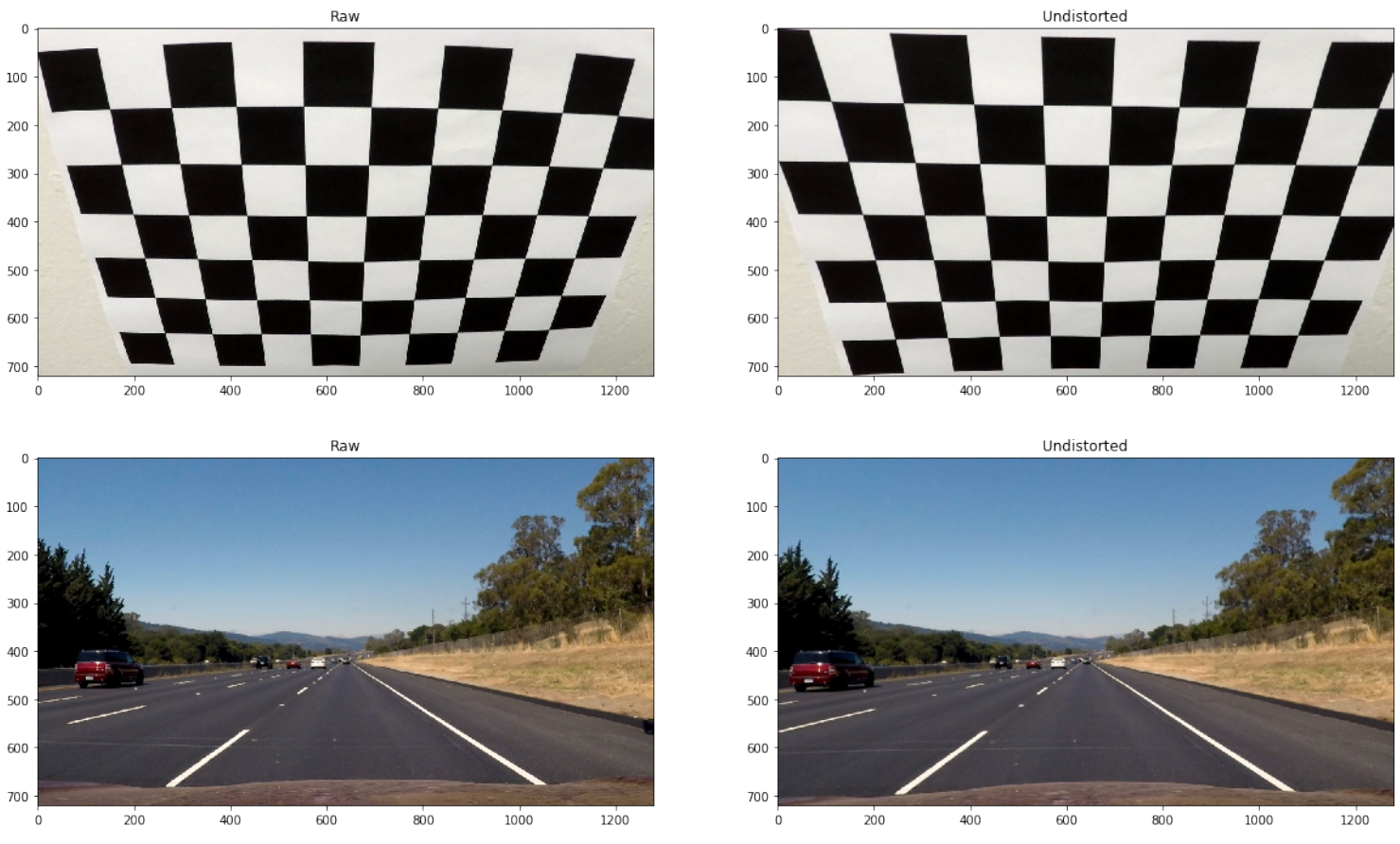
```
#Plot 2 images next to each other
def plot2(img1,img2,title1=None,title2=None,grey1=False,grey2=False):
    f, ax = plt.subplots(1,2, figsize=(20, 10))
    if (grey1):
        ax[0].imshow(img1,cmap='gray')
    else:
        ax[0].imshow(img1)
    if (title1 != None):
        ax[0].set_title(title1)
    if (grey2):
        ax[1].imshow(img2,cmap='gray')
    else:
        ax[1].imshow(img2)

    if (title2 != None):
        ax[1].set_title(title2)
    plt.show()
```

Apply a distortion correction to raw images.

In [4]:

```
def undistorted_sample(fname):
    f, ax = plt.subplots(1,2, figsize=(20, 10))
    ax[0].imshow(img)
    ax[0].set_title('Original')
    ax[1].imshow(dst)
    ax[1].set_title('Undistorted')
    plt.show()




# Read 1 image
#undistorted_sample('camera_cal/calibration2.jpg')
#undistorted_sample('test_images/straight_lines2.jpg')

#read an image, convert to RGB, and undistort
def read_image(fname):
    img = cv2.imread(fname)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img

def undistort(img):
    return cv2.undistort(img, mtx, dist, None, mtx)
```

```
raw=read_image('camera_cal/calibration2.jpg')
img=undistort(raw)
plot2(raw,img,'Raw','Undistorted')

raw=read_image('test_images/straight_lines2.jpg')
img=undistort(raw)
plot2(raw,img,'Raw','Undistorted')
```



In [5]:

```
#Read all test images for later use. They are stored by name in test_images

from os.path import basename,splitext

img_names = glob.glob('test_images/*')
###GLOBAL VARIABLE
test_images={}
for fname in img_names:
    img=undistort(read_image(fname))
    name=splitext(basename(fname))[0]
    test_images[name]=img

print("{} test images read from file".format(len(test_images.keys())))
```

```
9 test images read from file
```

Create a threshold binary image

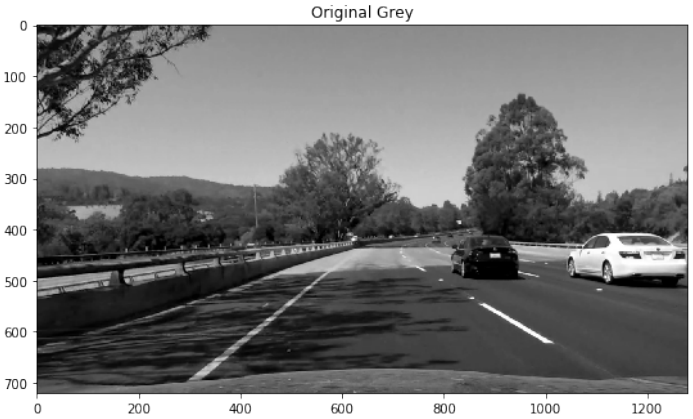
The final function to create the binary image with all the (possible) lane pixels, is `filter(img)`

In [6]:

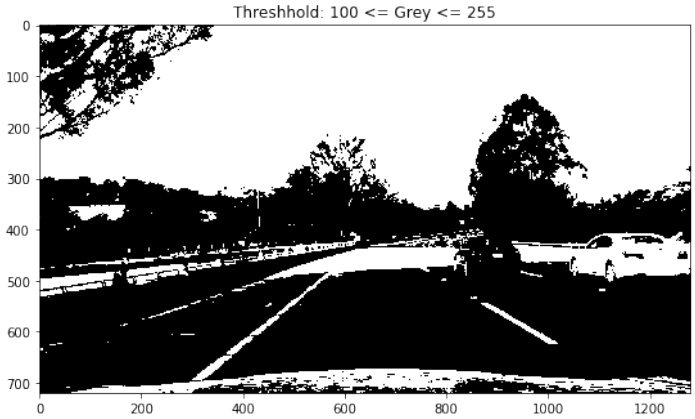
```
#Helper function to threshold range to greyscale image. If a name is given,
#then the original and filtered images are also plotted
def apply_threshold(img,tmin,tmax,name=None,plot=False):
    res=np.zeros_like(img)
    res[(img>=tmin)&(img<=tmax)]=1
    if (plot):
        f, ax = plt.subplots(1,2, figsize=(20, 10))
        ax[0].imshow(img,cmap='gray')
        ax[0].set_title('Original '+name)
        ax[1].imshow(res,cmap='gray')
        ax[1].set_title('Threshhold: {} <= {} <= {}'.format(tmin,name, tmax))
        plt.show()
    return res

# Function to calculate sobel values. Depending on the value of orient, sobelx
, sobely, gradient or absolute value
# is provided
def sobel(grey,orient='abs',ksize=3):
    sobelx = cv2.Sobel(grey, cv2.CV_64F, 1, 0,ksize=ksize)
##    sobely = cv2.Sobel(grey, cv2.CV_64F, 0, 1,ksize=ksize)
##    sobel= np.sqrt(np.square(sobelx)+np.square(sobely))
##    absgraddir = np.arctan2(np.absolute(sobely), np.absolute(sobelx))
##    sobel=np.uint8(255*sobel/np.max(sobel))
    sobelx=np.uint8(255*np.abs(sobelx)/np.max(sobelx))
##    sobely=np.uint8(255*np.abs(sobely)/np.max(sobely))
##    absgraddir=np.uint8(255*2*absgraddir/np.pi)

    if (orient=='x'):
        return sobelx
    if (orient=='y'):
        return sobely
    if (orient=='grad'):
```

```
    if (orient== 'grad'):
        return absgraddir
    if (orient=='abs'):
        return sobel
    return sobel

#overall function to filter for lanemarker pixels
#return an binary image of the same size
def filter(img,plot=False,debug=False):
    #Grey threshold
    grey = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    greybin = apply_threshold(grey,100,255,'Grey',plot)

    #Sobel threshold
    sx=sobel(grey,'x',7)
    sxbin=apply_threshold(sx,40,255,'Sobel x',debug)
##    sy=sobel(grey,'y',7)
##    sybin=apply_threshold(sy,70,255,'Sobel y',debug)
##    sa=sobel(grey,'abs',7)
##    sabin=apply_threshold(sa,40,170,'Sobel',debug)
##    sgrad=sobel(grey,'grad',9)
##    sgradbin=apply_threshold(sgrad,130,170,'Direction',debug)
    sobelbin=sxbin
    apply_threshold(sobelbin,0.5,1.5,'Sobel',plot)


    #HSL threshold
    hsl=cv2.cvtColor(img,cv2.COLOR_RGB2HLS)
    h=apply_threshold(hsl[:,:,0],10,60,'H',debug)
    s=apply_threshold(hsl[:,:,1],90,255,'S',debug)
    l=apply_threshold(hsl[:,:,2],90,255,'L',debug)
    hslbin=np.zeros_like(h)
    hslbin[(h>0) & (s>0) & (l>0)]=1
    apply_threshold(hslbin,0.5,1.5,'hsl',plot)

    #RGB Trheshold
    r=apply_threshold(img[:,:,0],120,255,'R',debug)
##    g=apply_threshold(img[:,:,1],120,255,'G',debug)
##    b=apply_threshold(img[:,:,2],120,255,'B',debug)
    rgbbin=r
    apply_threshold(rgbbin,0.5,1.5,'rgb',plot)


    linbin=np.zeros_like(rgbbin)
    linbin[((greybin>0) & (rgbbin>0) & (hslbin>0)) | (sobelbin>0)]=1
    apply_threshold(linbin,0.5,1.5,'final',plot)
    return linbin

#Read the test image
img = test_images['extra1']
bin=filter(img,True,True)
plot2(img,bin,'Original','Final filter',False,True)
```

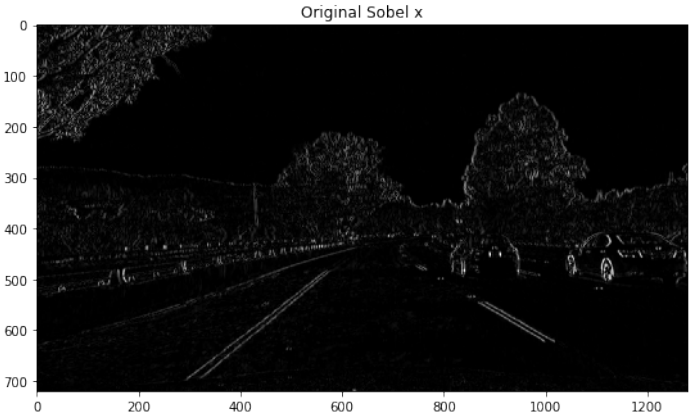
Original Grey



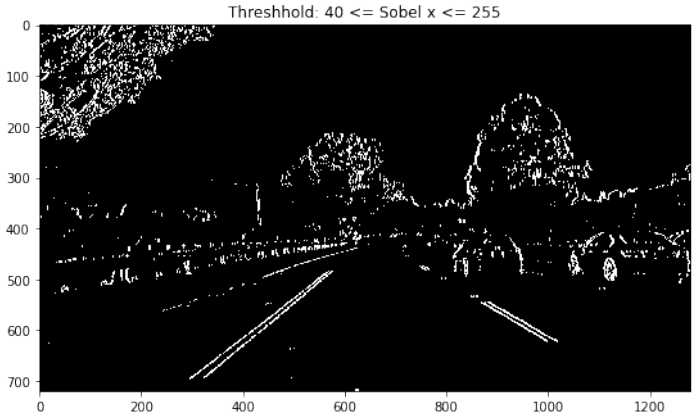
Threshhold: 100 <= Grey <= 255



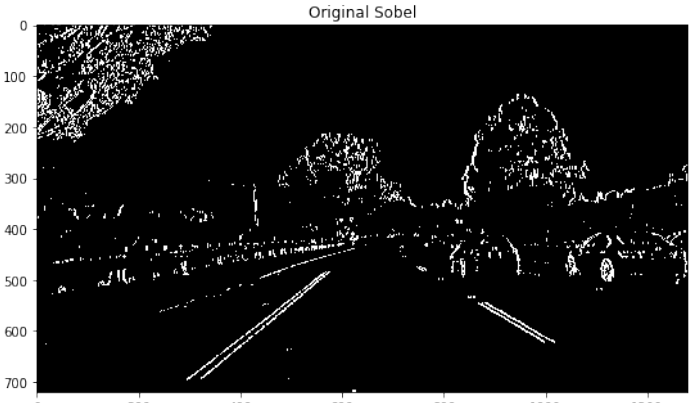
Original Sobel x



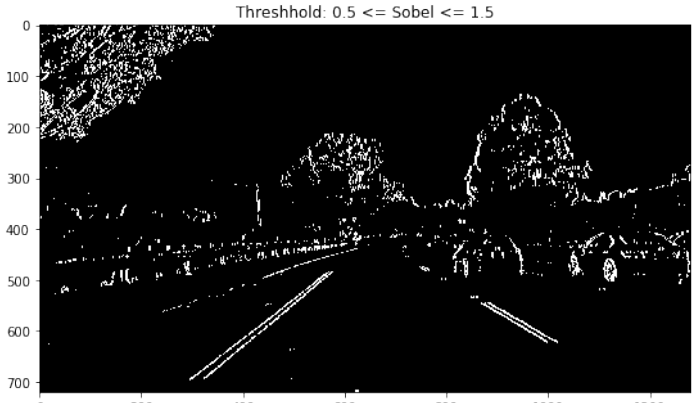
Threshhold: 40 <= Sobel x <= 255



Original Sobel

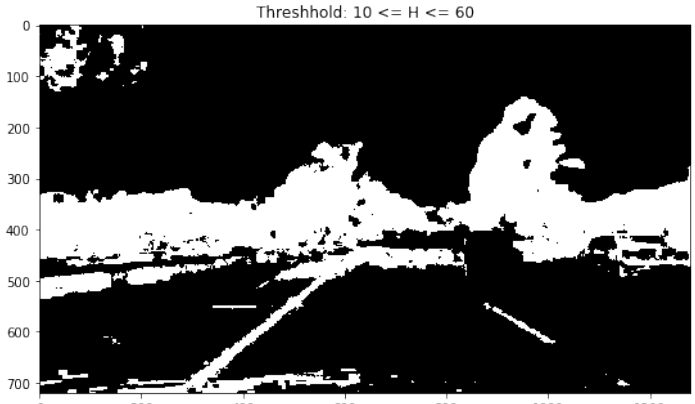


Threshhold: 0.5 <= Sobel <= 1.5



Original H

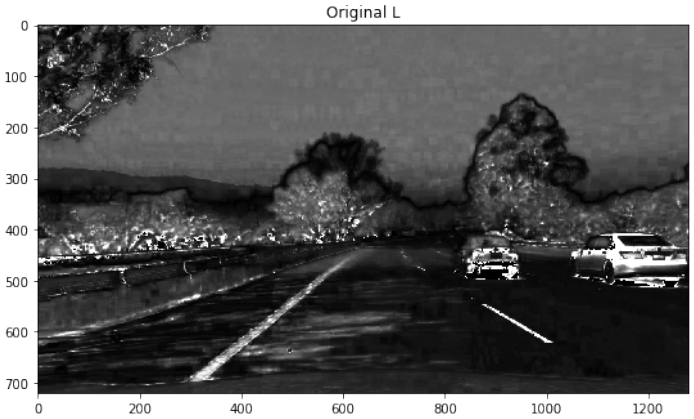
Threshhold: 10 <= H <= 60



Original S

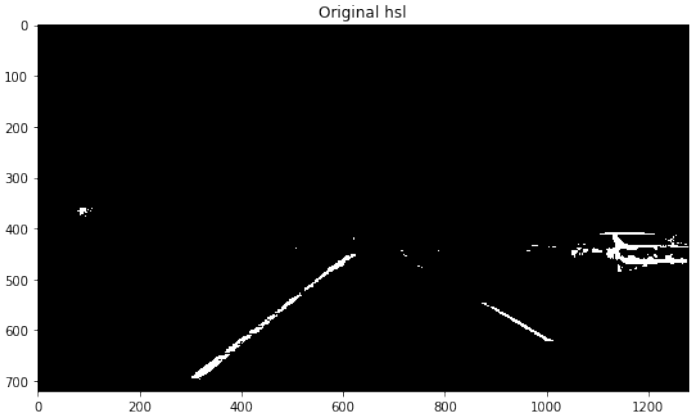
Threshhold: 90 <= S <= 255

Original L

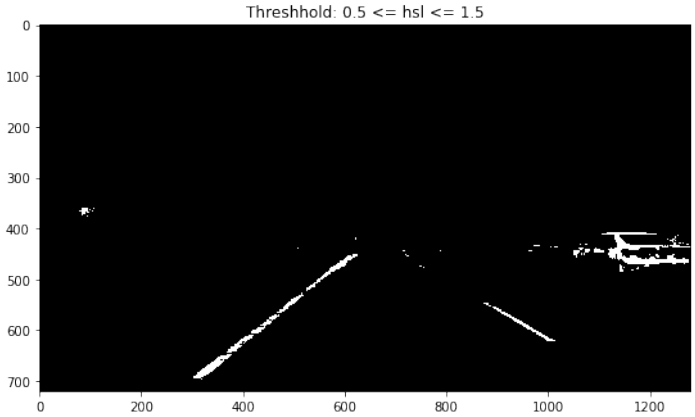


Threshhold: 90 <= L <= 255

Original hsl

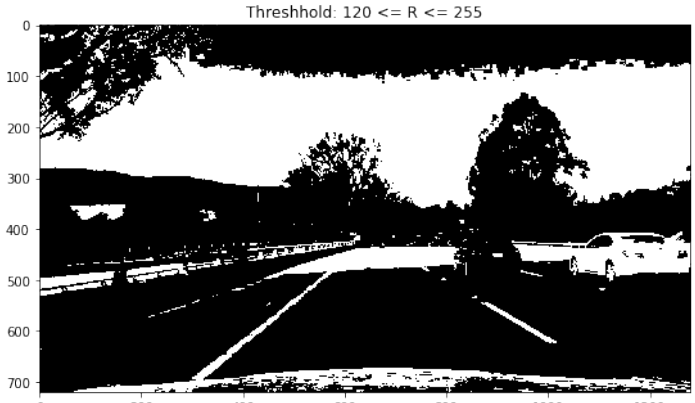


Threshhold: 0.5 <= hsl <= 1.5



Original R

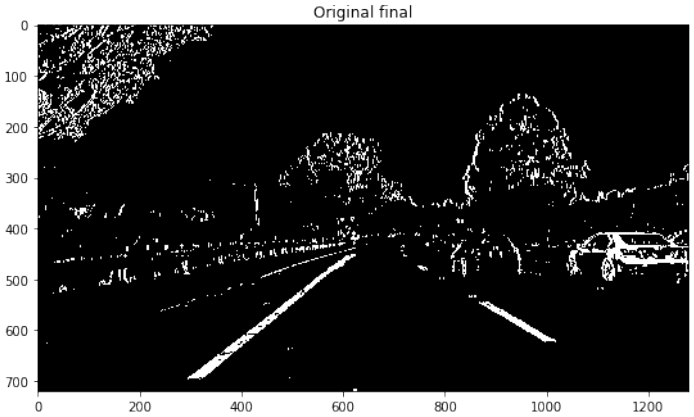
Threshhold: 120 <= R <= 255



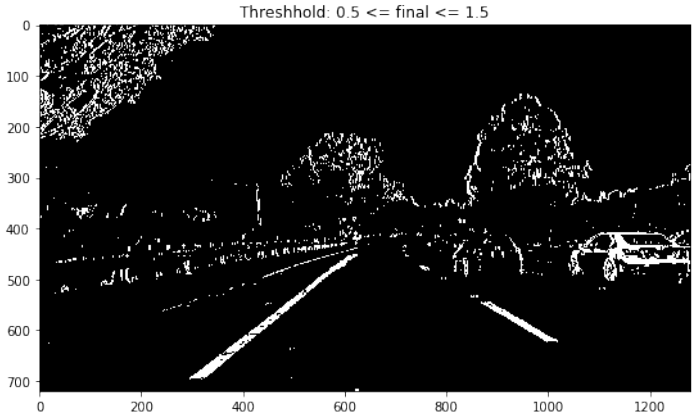
Original rgb

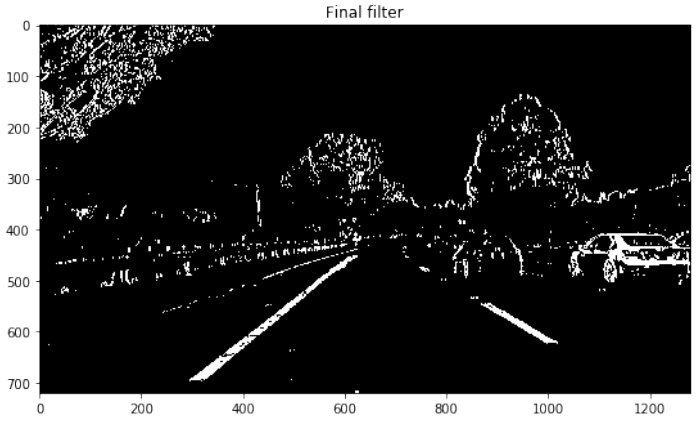
Threshhold: 0.5 <= rgb <= 1.5

Original final



Threshhold: 0.5 <= final <= 1.5



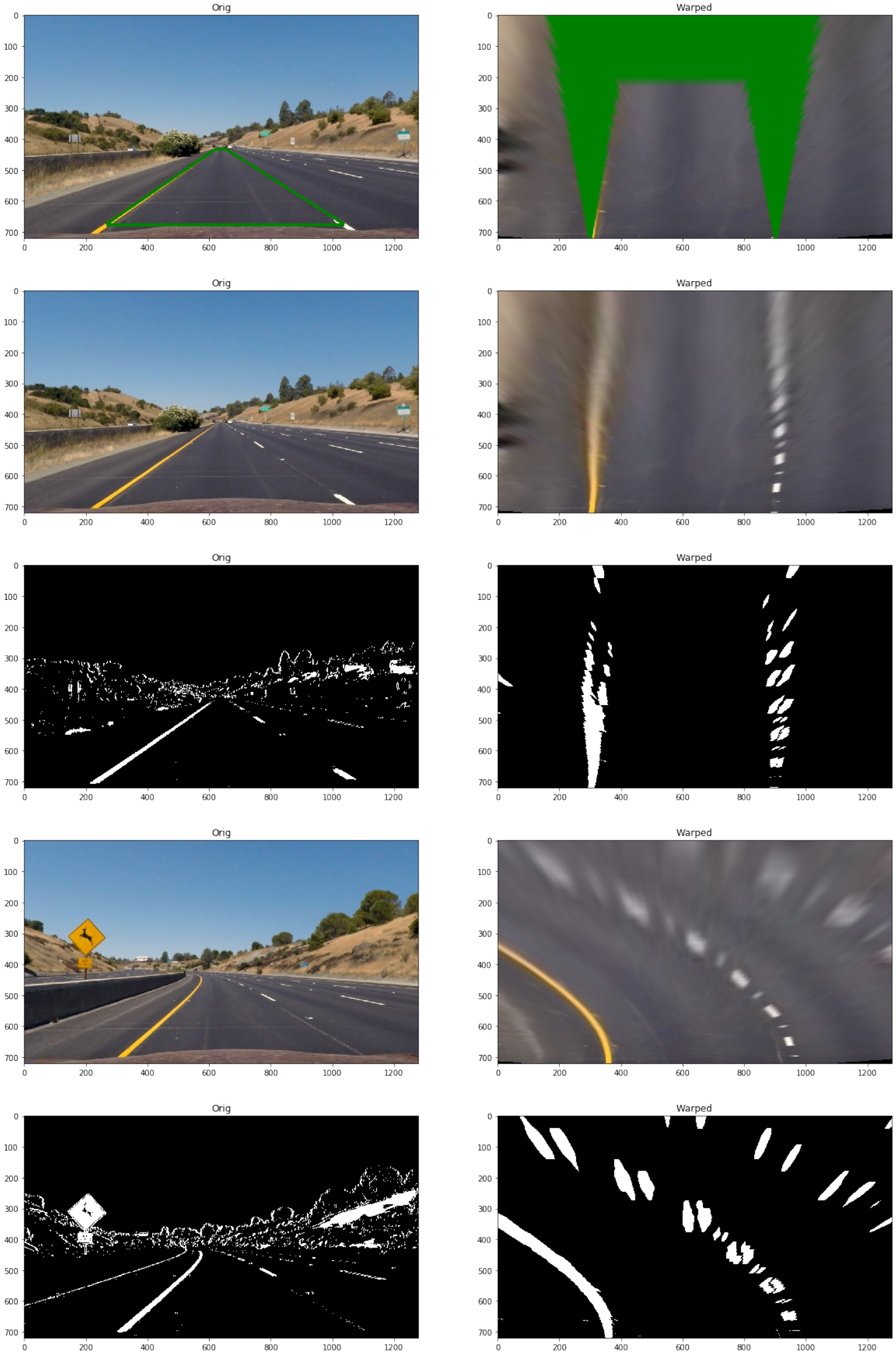


Birds-eye view

The final function to perform the calculation is `calculate_perspective_warp()`. This function returns `M` and `Minv`. These can be used in `warp(img,M)` to perform the actual warp or inverse warp.

In [7]:

```
from PIL import Image, ImageDraw

def calculate_perspective_warp(target_left=300,target_right=900):
    img=test_images['straight_lines1']
    #src and destination points
    #dst choosen to put the lines at x=300 and x=980

    src=np.array([[266,675],[1038,675],[655,433],[619,433],[266,675]],np.float
32)
    dst=np.array([[target_left,719],[target_right,719],[target_right,10],[targ
et_left,10],[target_right,719]],np.float32)
    M = cv2.getPerspectiveTransform(src[0:4], dst[0:4])
    Minv = cv2.getPerspectiveTransform(dst[0:4], src[0:4])



    shape=(img.shape[1],img.shape[0])
    warped = cv2.warpPerspective(img, M, shape, flags=cv2.INTER_LINEAR)
    #binw=cv2.warpPerspective(bin, M, shape, flags=cv2.INTER_LINEAR)

    pic = Image.fromarray(img)
    draw=ImageDraw.Draw(pic)
    draw.line(src,fill='green',width=10)
    orig=np.array(pic.getdata(),
                  np.uint8).reshape(pic.size[1], pic.size[0], 3)
    warped = cv2.warpPerspective(orig, M, shape, flags=cv2.INTER_LINEAR)
    plot2(orig,warped,'Orig','Warped')



    return M,Minv

(M,Minv)=calculate_perspective_warp()


def warp(img,M,plot=False):
    shape=(img.shape[1],img.shape[0])
    warped = cv2.warpPerspective(img, M, shape, flags=cv2.INTER_LINEAR)
    if (plot):
            plot2(img,warped,'Orig','Warped',True,True)

    return warped

warped=warp(test_images['straight_lines1'],M,True)
warped=warp(filter(test_images['straight_lines1']),M,True)
warped=warp(test_images['test2'],M,True)
warped=warp(filter(test_images['test2']),M,True)
```



Detect lane pixels and fit to find the lane boundary

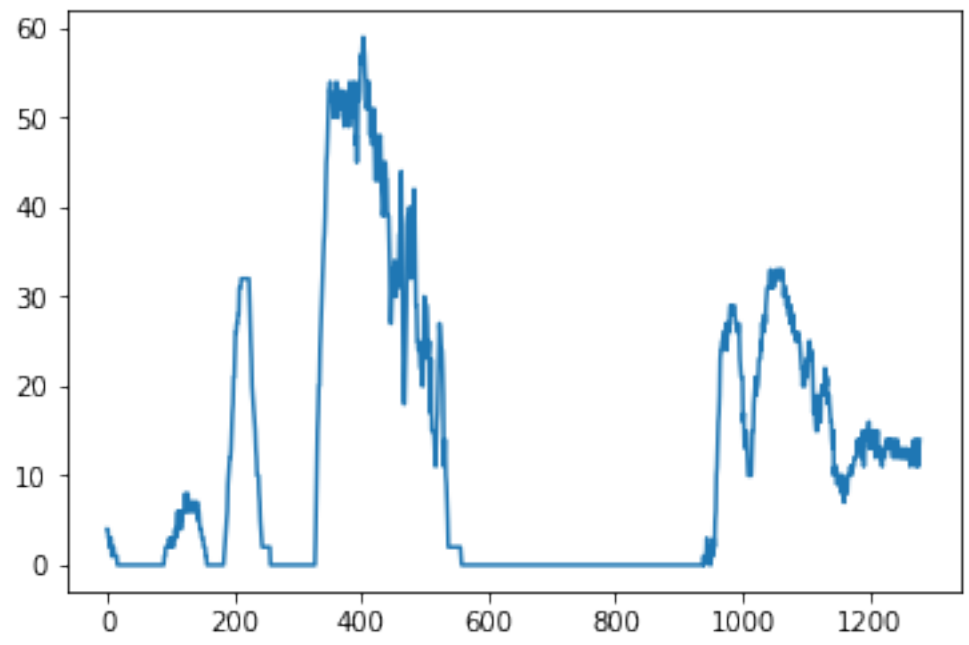
In [8]:

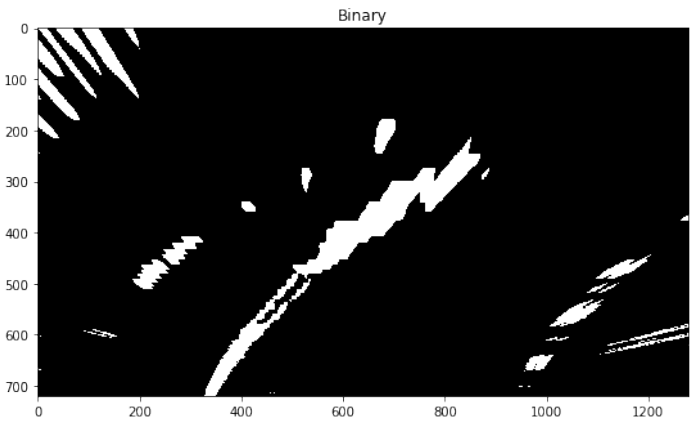
```
def calculate_windows(shape,xlcenter,xrcenter,margin):
```

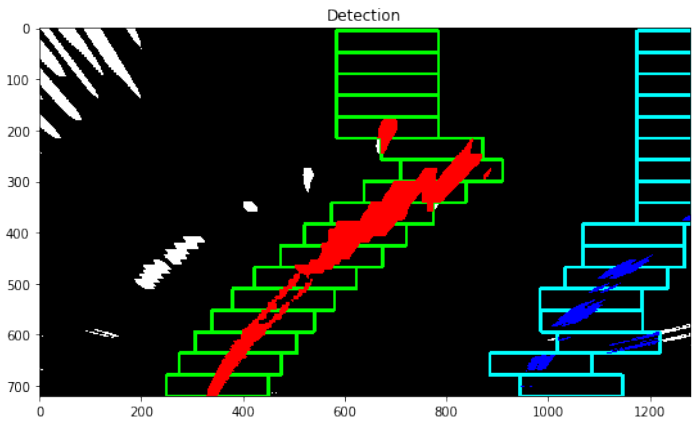
```
    xllow = max(min(xlcenter - margin,shape[1]),0)

    xrlow = max(min(xrcenter - margin,shape[1]),0)
    xlhigh = max(min(xlcenter + margin,shape[1]),0)
    xrhigh = max(min(xrcenter + margin,shape[1]),0)
    return xllow,xrlow,xlhigh,xrhigh

def calc_fit(fit,y):
    x = (fit[0]*y**2 + fit[1]*y + fit[2])
    return x


#helper function to create the horizontal range of a window, and to select the
activated pixels
def get_nonzero_pixels(nonzero,ylow,yhigh,xlow,xhigh):
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    # Identify the nonzero pixels in x and y within the window
    good_inds = ((nonzeroy >= ylow) & (nonzeroy < yhigh)
                      & (nonzerox >= xlow) & (nonzerox < xhigh)).nonzero()[0]

    return good_inds


def detect_lines(binary_warped,plot=False, nwindows=17,margin = 200, redo_marg
in=75, minpix=100, left_fit=None, right_fit=None,redoWithFoundPixels=True,draw
_windows=False ):
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])

    use_fit=not ((left_fit is None) | (right_fit is None))
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))*255

    # Create empty lists to receive left and right lane pixel indices
    left_lane_inds = []
    right_lane_inds = []

    if (not use_fit):
        #only take lower 1/3th for histogram
        start=np.int(2*binary_warped.shape[0]/3)
        histogram = np.sum(binary_warped[start:,:], axis=0)
        if (plot):
            plt.plot(histogram)
            plt.show()


        midpoint = np.int(histogram.shape[0]/2)
        leftx_base = np.argmax(histogram[:midpoint])
        rightx_base = np.argmax(histogram[midpoint:]) + midpoint

        # Current positions to be updated for each window
        leftx_current = leftx_base
        rightx_current = rightx_base
    else:
        nwindows=nwindows*10
```

```
    # Step through the windows one by one
    for window in range(nwindows):
        window_height=np.int(binary_warped.shape[0]/nwindows)
        ylow = binary_warped.shape[0] - (window+1)*window_height
        yhigh = ylow + window_height
        if (use_fit):
            leftx_current=int(calc_fit(left_fit,(ylow+yhigh)/2))
            rightx_current=int(calc_fit(right_fit,(ylow+yhigh)/2))
            (xllow,xrlow,xlhigh,xrhigh)=calculate_windows(binary_warped.shape,
leftx_current,rightx_current,redo_margin)


        else:
            (xllow,xrlow,xlhigh,xrhigh)=calculate_windows(binary_warped.shape,
leftx_current,rightx_current,margin)

        good_left_inds=get_nonzero_pixels(nonzero,ylow,yhigh,xllow,xlhigh)
        good_right_inds=get_nonzero_pixels(nonzero,ylow,yhigh,xrlow,xrhigh)


        # Identify the nonzero pixels in x and y within the window
        #print('Pixels found Left: {}, Right:{}'.format(good_left_inds.size,go
od_right_inds.size))

        if ((not use_fit) & redoWithFoundPixels):
            if len(good_left_inds) > minpix:
                leftx_redo = np.int(np.mean(nonzerox[good_left_inds]))
            else:
                leftx_redo=leftx_current
            if len(good_right_inds) > minpix:
                rightx_redo = np.int(np.mean(nonzerox[good_right_inds]))
            else:
                rightx_redo=rightx_current
            (xllow,xrlow,xlhigh,xrhigh)=calculate_windows(binary_warped.shape,
leftx_redo,rightx_redo,redo_margin)
            good_left_inds=get_nonzero_pixels(nonzero,ylow,yhigh,xllow,xlhigh)
            good_right_inds=get_nonzero_pixels(nonzero,ylow,yhigh,xrlow,xrhigh
)


        #print('Pixels found in redo: Left: {}, Right:{}'.format(good_left_ind
s.size,good_right_inds.size))


        # Draw the windows on the visualization image
        if (draw_windows &(nwindows<50)):
            cv2.rectangle(out_img,(xllow,ylow),(xlhigh,yhigh),(0,255,0), 5)
            cv2.rectangle(out_img,(xrlow,ylow),(xrhigh,yhigh),(0,255,255), 5)


        # Append these indices to the lists
        left_lane_inds.append(good_left_inds)
        right_lane_inds.append(good_right_inds)
        # If you found > minpix pixels, recenter next window on their mean pos
ition
        if len(good_left_inds) > minpix:
            leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
```

```
        if len(good_right_inds) > minpix:
            rightx_current = np.int(np.mean(nonzerox[good_right_inds]))

    # Concatenate the arrays of indices
    left_lane_inds = np.concatenate(left_lane_inds)
    right_lane_inds = np.concatenate(right_lane_inds)

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]


    out_img[nonzeroy[left_lane_inds], nonzerox[left_lane_inds]] = [255, 0, 0]
    out_img[nonzeroy[right_lane_inds], nonzerox[right_lane_inds]] = [0, 0, 255
]

    if (plot):
        plot2(binary_warped,out_img,'Binary','Detection',True,False)

    return out_img,leftx,lefty,rightx,righty

img=test_images['test6']
binary_warped=warp(filter(img),M)

(detections,leftx,lefty,rightx,righty)=detect_lines(binary_warped,redo_margin=
100,plot=True,draw_windows=True)
```





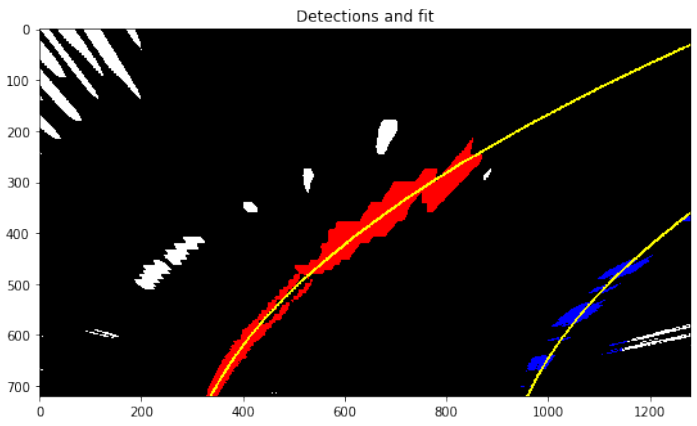


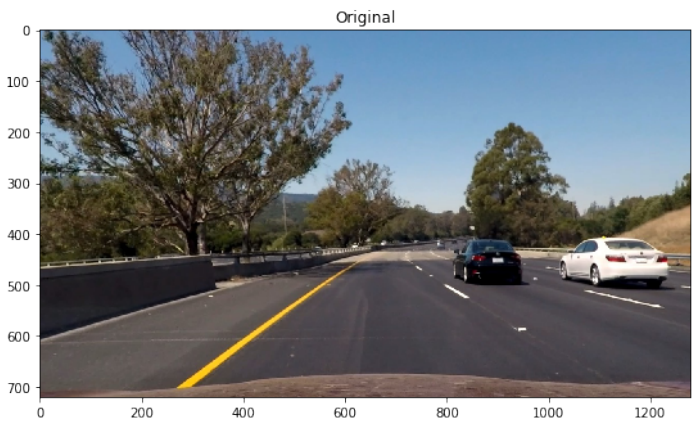
Determine the curvature of the lane and vehicle position with respect to center.

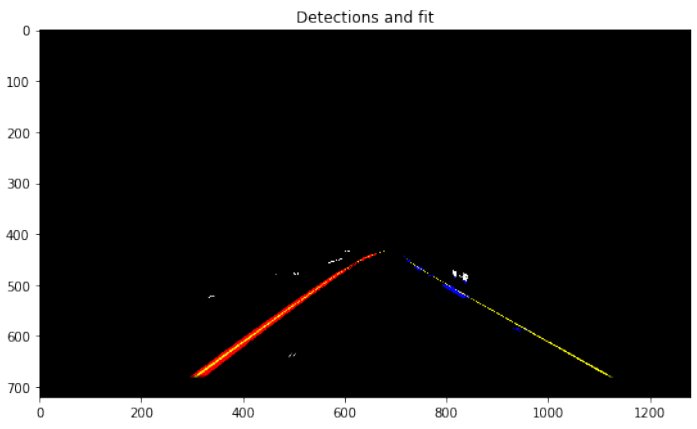
In [9]:

```
#vertical: 50 pixels is 1 dash line is 3 m
#horizontal: distance between left and right lane in the transformation define
d to be 600 = 3.7m

def fit_curve(img,x,y,plot=False,yscale=3./50.,xscale=3.7/600.):
    # Fit a second order polynomial to each. Use the y value also as weight, s
o that closer by is more important
    w=y
    y=y*yscale
    x=x*xscale

    fit = np.polyfit(y, x, 2,w=w)

    if (plot):
        # Generate x and y values for plotting
        ploty = np.linspace(0, img.shape[0]-1, img.shape[0])
        yfit=ploty*yscale

        xfit = (fit[0]*yfit**2 + fit[1]*yfit + fit[2])

        plt.subplots(1,1,figsize=(10, 10))
        plt.imshow(img)
        plt.plot(xfit/xscale, yfit/yscale, color='yellow')

        plt.xlim(0, img.shape[1])
        plt.ylim(img.shape[0], 0)
        plt.show()

    return fit

def overlay_fit(img,fit,xscale,yscale,color=(255,255,0),width=3):
    out=np.array(img, copy=True)
        #only use lower 65%
    offset=int(img.shape[0]*0.35)
    offset=0
    size=img.shape[0]-offset
    #20 points is enough. for as many points as pixels, use size as last param
eter
    ploty = np.linspace(offset, offset+size-1, 20 )
    y=ploty*yscale

    fitx = (fit[0]*y**2 + fit[1]*y + fit[2])/xscale

    data=list(zip(fitx, ploty))
    pts=np.asarray(data,np.int32).reshape((-1,1,2))
    cv2.polylines(out,[pts],False,color,width)
    return out

yscale=1.
xscale=1.
```

```
left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=xsca

le)
right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xscale=x
scale)

(detections,leftx,lefty,rightx,righty)=detect_lines(binary_warped,redo_margin=
100,left_fit=left_fit,right_fit=right_fit,plot=False)



yscale=3./50.
xscale=3.7/600.
left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=xsca
le)
right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xscale=x
scale)

fitted=overlay_fit(detections,left_fit,xscale,yscale)
fitted=overlay_fit(fitted,right_fit,xscale,yscale)
plot2(warp(img,M),fitted,'Warped','Detections and fit',True,False)
plot2(img,warp(fitted,Minv),'Original','Detections and fit',True,False)
```







In [10]:

```
def curve_and_center(left_fit,right_fit,y,width):
    left_curverad = ((1 + (2*left_fit[0]*y + left_fit[1])**2)**1.5) / np.absol
ute(2*left_fit[0])
    right_curverad = ((1 + (2*right_fit[0]*y + right_fit[1])**2)**1.5) / np.ab
solute(2*right_fit[0])

    xl = left_fit[0]*y**2 + left_fit[1]*y + left_fit[2]
    xr = right_fit[0]*y**2 + right_fit[1]*y + right_fit[2]

    xcenter=(xl+xr)/2
    offcenter=xcenter-width/2

    return left_curverad,right_curverad,offcenter,xl,xr



yscale=1.
xscale=1.
left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=xsca
le)
right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xscale=x
scale)

yeval=(binary_warped.shape[0]-1)*yscale
width=binary_warped.shape[1]*xscale
(left_curve,right_curve,offcenter,xl,xr)=curve_and_center(left_fit,right_fit,y
eval,width)
print('Curvatures 1/pix: ', left_curve, right_curve)
print('Off center pix  : ',offcenter)




yscale=3./50.
xscale=3.7/600.
left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=xsca
le)
right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xscale=x
scale)

yeval=(binary_warped.shape[0]-1)*yscale
width=binary_warped.shape[1]*xscale

(left_curve,right_curve,offcenter,xl,xr)=curve_and_center(left_fit,right_fit,y
eval,width)
print('Curvatures 1/m: ', left_curve, right_curve)
print('Off center m  : ',offcenter)
```

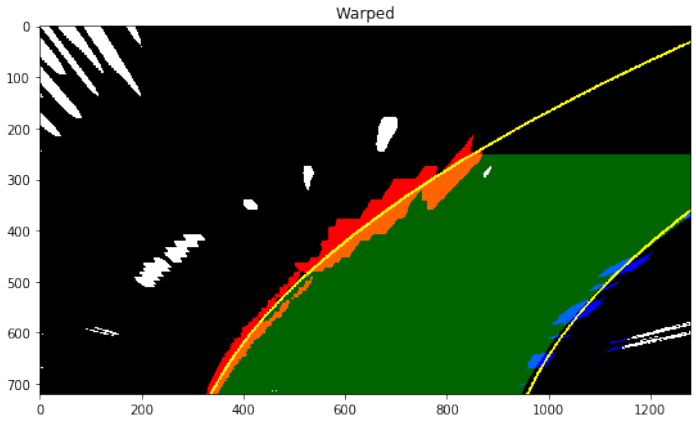
```
Curvatures 1/pix:  575.379132835 556.616106452
Off center pix  :  7.46691818649
Curvatures 1/m:  236.050981504 243.347997213
Off center m  :  0.0460459954834
```

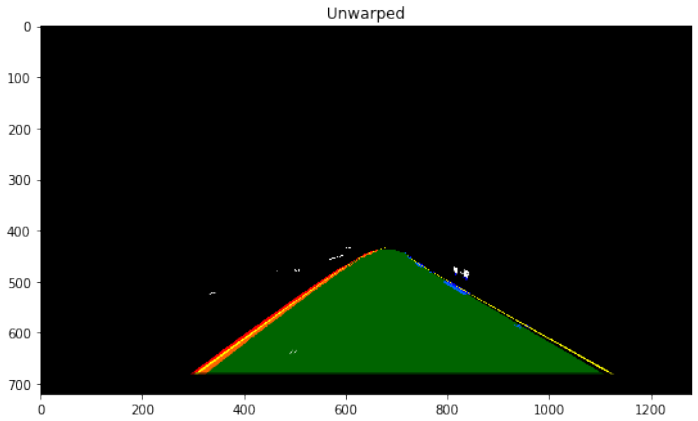
Warp back to original image

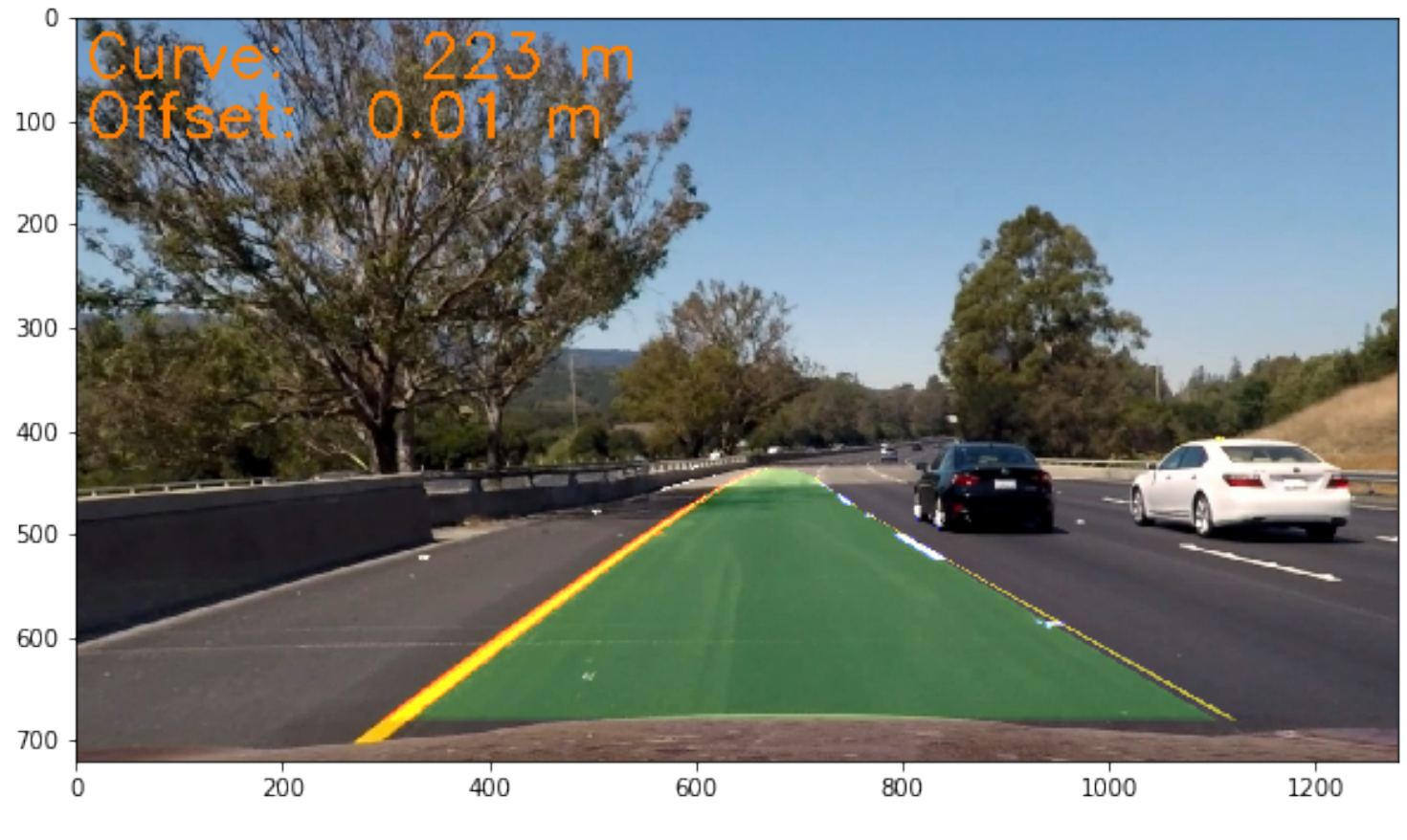
In [11]:

```
import time


def warp_back(img,detections,left_fit,right_fit,xscale=1.,yscale=1.,plot=False
,curvature=None,offcenter=None):

    #out_img=np.zeros_like(img)
    out=np.zeros_like(detections)
    #out_img = np.dstack((binary_warped, binary_warped, binary_warped))*0
    #pic = Image.fromarray(out_img)
    #draw=ImageDraw.Draw(pic)

    #only use lower 65%
    offset=int(img.shape[0]*0.35)
    size=img.shape[0]-offset
    #20 points is enough. for as many points as pixels, use size as last param
eter
    ploty = np.linspace(offset, offset+size-1, 20 )
    y=ploty*yscale

    left_fitx = (left_fit[0]*y**2 + left_fit[1]*y + left_fit[2])/xscale
    right_fitx = (right_fit[0]*y**2 + right_fit[1]*y + right_fit[2])/xscale

    datal=list(zip(left_fitx, ploty))
    datar=list(zip(right_fitx, ploty))

    pts=np.asarray(datal,np.int32).reshape((-1,1,2))
    #cv2.polylines(out,[pts],False,(200,200,0),25)

    pts=np.asarray(datar,np.int32).reshape((-1,1,2))
    #cv2.polylines(out,[pts],False,(200,200,0),25)

    pts=np.asarray(datar+list(reversed(datal)),np.int32).reshape((-1,1,2))
    cv2.fillPoly(out,[pts],(0,200,0))
    out=cv2.addWeighted(detections, 1, out, 0.5, 0.)



    unwarped = cv2.warpPerspective(out, Minv, (img.shape[1],img.shape[0]), fla
gs=cv2.INTER_LINEAR)
    if (plot):
        plot2(out,unwarped,'Warped','Unwarped',True,False)

    out=cv2.addWeighted(img, 1, unwarped, 0.5, 0.)

    if (not (curvature is None)):
        cv2.putText(out, 'Curve: {:6.0f} m'.format(curvature), (int(10), int(o
ut.shape[0]*0.08)), cv2.FONT_HERSHEY_SIMPLEX, 2.,[255,128,0],3)

    if (not (offcenter is None)):
        cv2.putText(out, 'Offset: {:5.2f} m'.format(offcenter), (int(10), int(
```

```
out.shape[0]*0.16)), cv2.FONT_HERSHEY_SIMPLEX, 2.,[255,128,0],3)

    if (plot):
        plt.subplots(1,1,figsize=(10, 10))
        plt.imshow(out)
        plt.show()

    return out

yscale=1.
xscale=1.
left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=xsca
le)
right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xscale=x
scale)

(detections,leftx,lefty,rightx,righty)=detect_lines(binary_warped,left_fit=lef
t_fit,right_fit=right_fit,plot=False,draw_windows=True)

yscale=3./50.
xscale=3.7/600.
left_fit=fit_curve(fitted,leftx,lefty,plot=False,yscale=yscale,xscale=xscale)
right_fit=fit_curve(fitted,rightx,righty,plot=False,yscale=yscale,xscale=xscal
e)
(left_curve,right_curve,offcenter,xl,xr)=curve_and_center(left_fit,right_fit,y
eval,width)
unwarped=warp_back(img,fitted,left_fit,right_fit,xscale,yscale,True,(left_curv
e+right_curve)/2, offcenter)
```







Complete pipeline

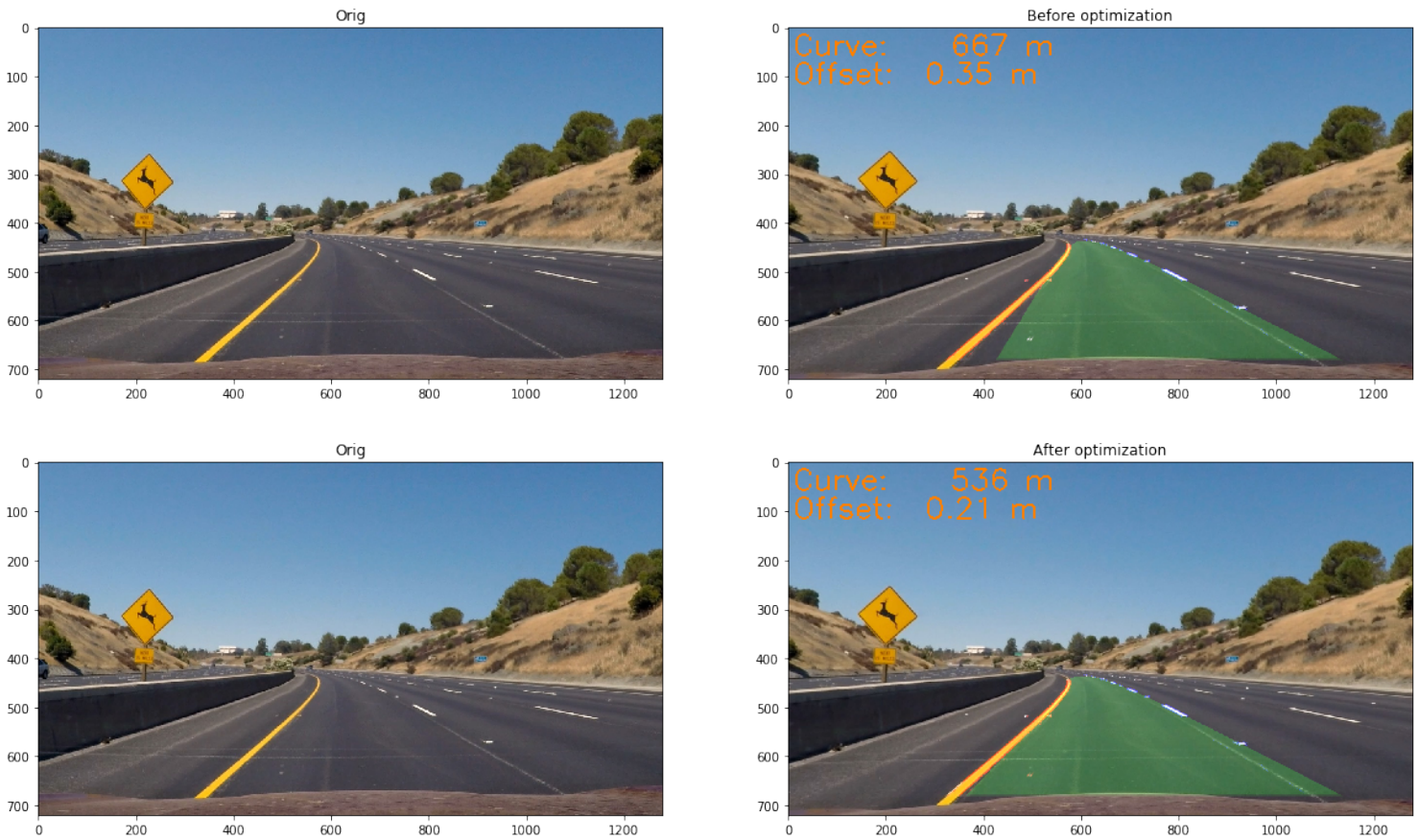
Input is 1 image Output is an overlayed image, the left and right lane curverture, and the distance from center

In [12]:

```
#The pipeline requires global variables to be present for distortion correctio
n, and perspective warp
# and left_fit and right_fit. curvature and offset are stored in global parame
ters as well

def init_pipeline():
    global left_fit,right_fit, left_curve_values,right_curve_values,offcenter_
values,xl_values,xr_values,reset_values
    left_fit=None
    right_fit=None
    left_curve_values=np.array([],np.float32)
    right_curve_values=np.array([],np.float32)
    offcenter_values=np.array([],np.float32)
    xl_values=np.array([],np.float32)
    xr_values=np.array([],np.float32)
    reset_values=np.array([],np.float32)
```

```
def pipeline(img, distorted=True):

    global left_fit,right_fit, left_curve_values,right_curve_values,offcenter_
values,xl_values,xr_values,reset_values
    if(distorted):
        img=undistort(img)
    bin=filter(img)

    bin_warped=warp(bin,M)

    (detections,leftx,lefty,rightx,righty)=detect_lines(bin_warped,plot=False,
left_fit=left_fit,right_fit=right_fit)

    yscale=3./30.
    xscale=3.7/600.
    left_fit_cr=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xsca
le=xscale)
    right_fit_cr=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,x
scale=xscale)

    yeval=(bin_warped.shape[0]-1)*yscale
    width=bin_warped.shape[1]*xscale

    (left_curve,right_curve,offcenter,xl,xr)=curve_and_center(left_fit_cr,righ
t_fit_cr,yeval,width)


    xscale=1.
    yscale=1.
    left_fit=fit_curve(detections,leftx,lefty,plot=False,yscale=yscale,xscale=
xscale)
    right_fit=fit_curve(detections,rightx,righty,plot=False,yscale=yscale,xsca
le=xscale)
    unwarped=warp_back(img,detections,left_fit,right_fit,xscale,yscale,curvatu
re=(left_curve+right_curve)/2,offcenter=offcenter)


    left_curve_values=np.append(left_curve_values,left_curve)
    right_curve_values=np.append(right_curve_values,right_curve)
    offcenter_values=np.append(offcenter_values,offcenter)
    xl_values=np.append(xl_values,xl)
    xr_values=np.append(xr_values,xr)
    #determine whether the values are good. If not, reset and start without fi
t result next time
    if((offcenter<-1.5)|(offcenter>1.5)|(xl<1)|(xr>7)):
        left_fit=None
        right_fit=None
        reset_values=np.append(reset_values,1.)
    else:
        reset_values=np.append(reset_values,0.)


    return unwarped


init_pipeline()
img=read_image('test_images/test2.jpg')
unwarped=pipeline(img,True)
```

```
plot2(img,unwarped,'Orig','Before optimization')

unwarped=pipeline(img,True)
plot2(img,unwarped,'Orig','After optimization')
```

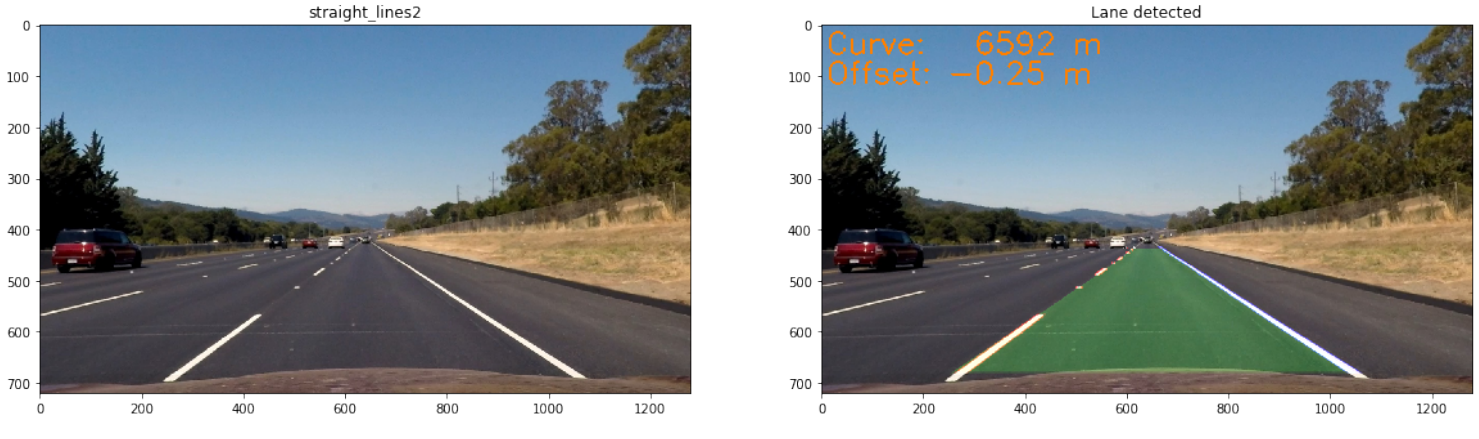


Process all test images with complete pipeline

All images are processed twice: the first time to find the fit lanes, the second time to go through the optimized pipeline

In [13]:

```
init_pipeline()
for im in test_images:
    img=test_images[im]
    init_pipeline()
    unwarped=pipeline(img,False)
    unwarped=pipeline(img,False)


    plot2(img,unwarped,im,'Lane detected')
```

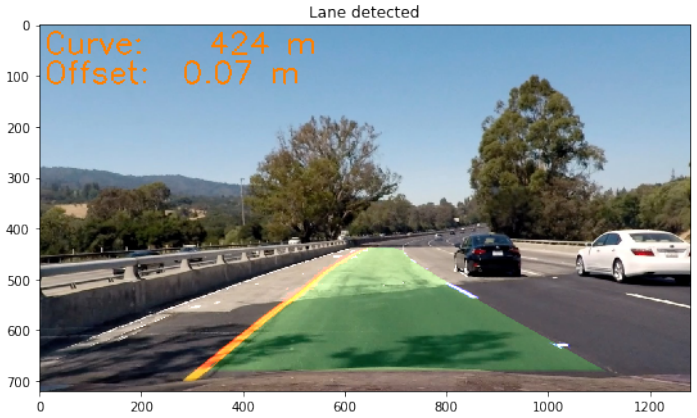


straight_lines1

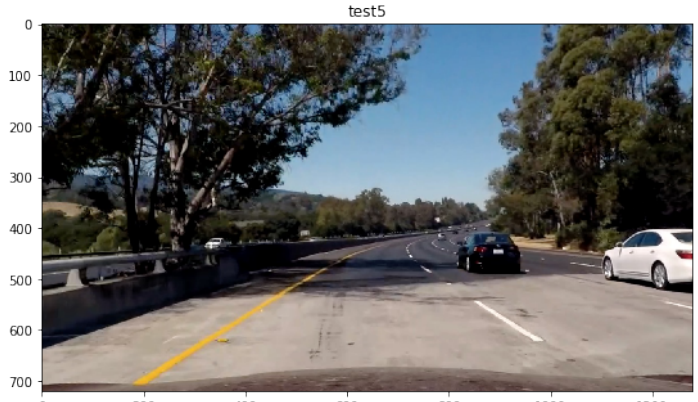
Lane detected

test4

Lane detected



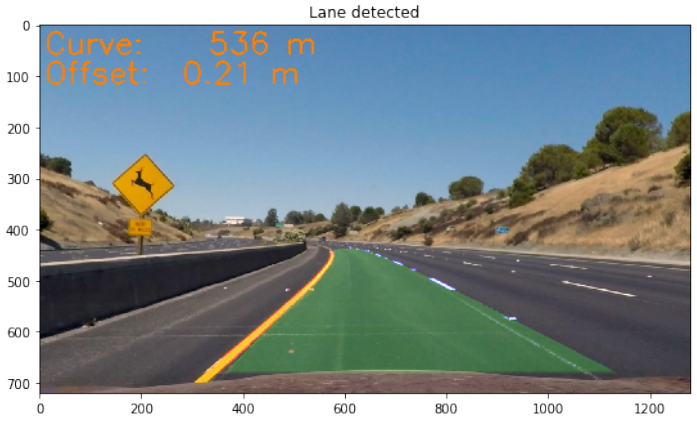
test5



Lane detected

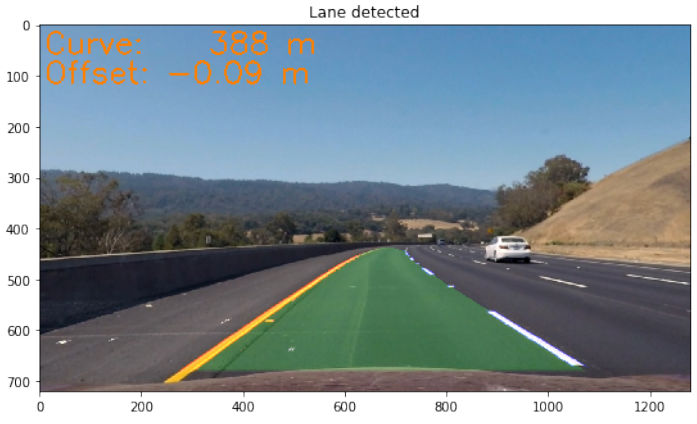
test2

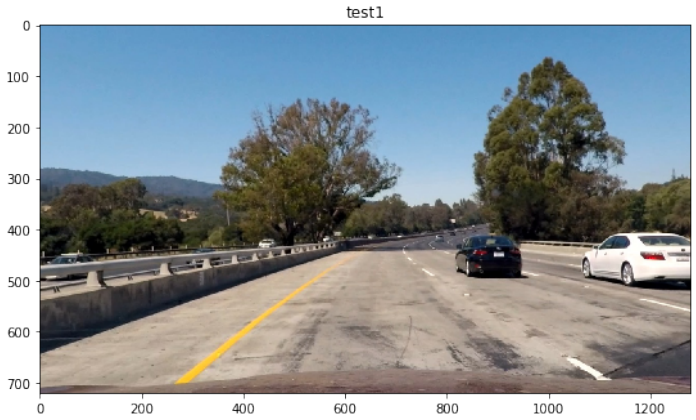
Lane detected

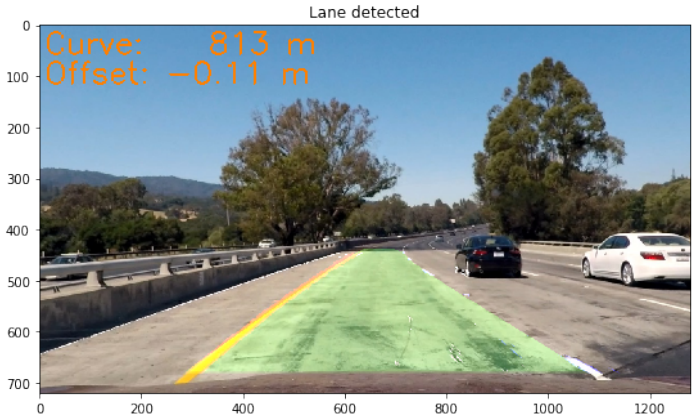


test3

Lane detected




test1


Lane detected
Curve: 813 m
Offset: −0.11 m


extra1


Lane detected
Curve: 460 m
Offset: 0.00 m


test6


Lane detected
Curve: 628 m
Offset: 0.01 m

In [14]:

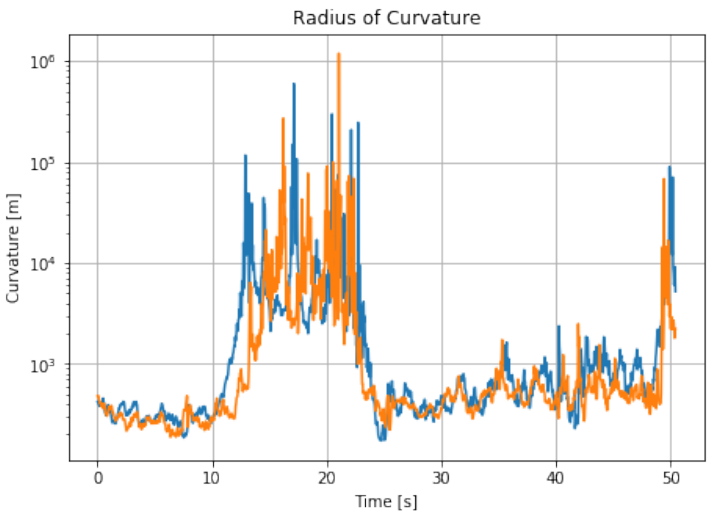
```
#plot the curve of position results. Assume 25 frames/sec
def plot_curvatures():
    x=np.linspace(0, (len(left_curve_values))/25, len(left_curve_values) )

    f, ax = plt.subplots(1,2, figsize=(16, 5))
    ax[0].plot(x,left_curve_values)
    ax[0].plot(x,right_curve_values)
    ax[0].set_yscale('log')

    #ax[0].set_ylim([0,5000])
    ax[0].set_title('Radius of Curvature')
    ax[0].set_ylabel('Curvature [m]')
    ax[0].set_xlabel('Time [s]')
    ax[0].grid(True)

    ax[1].plot(x,1./left_curve_values)
    ax[1].plot(x,1./right_curve_values)
    #ax[1].set_ylim([0,5000])
    ax[1].set_title('Radius of Curvature (inv)')
    ax[1].set_ylabel('Curvature [1/m]')
    ax[1].set_xlabel('Time [s]')
    ax[1].grid(True)
    plt.show()

    f, ax = plt.subplots(1,2, figsize=(16, 5))
    ax[0].plot(x,offcenter_values)
    ax[0].set_ylim([-1,1])
    ax[0].set_title('Offset from center')
    ax[0].set_ylabel('Offset [m]')
    ax[0].set_xlabel('Time [s]')
    ax[0].grid(True)

    ax[1].plot(x,xl_values)
    ax[1].plot(x,xr_values)
#    ax[1].set_ylim([-5,5])
    ax[1].set_title('left and right lane positions')
    ax[1].set_ylabel('Position [m]')
    ax[1].set_xlabel('Time [s]')
    ax[1].grid(True)

    f, ax = plt.subplots(1,2, figsize=(16, 5))
    ax[0].plot(x,reset_values)
#    ax[0].set_ylim([-5,5])
    ax[0].set_title('Resets')
    ax[0].set_ylabel('Reset')
    ax[0].set_xlabel('Time [s]')
    ax[0].grid(True)
    plt.show()
```

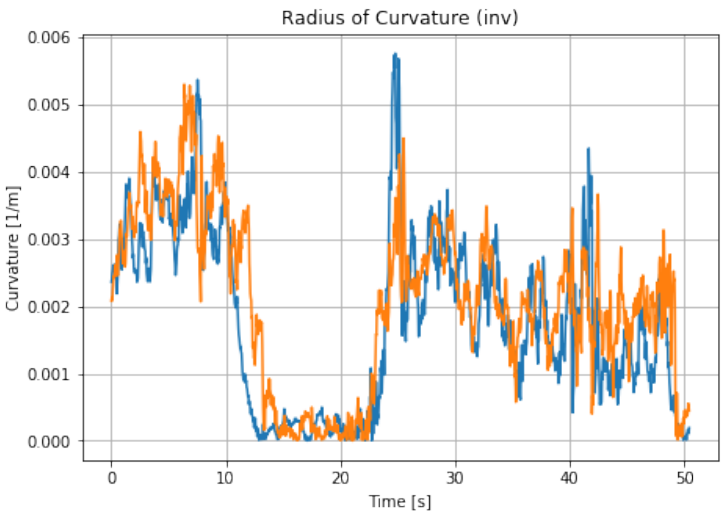
Process the project video

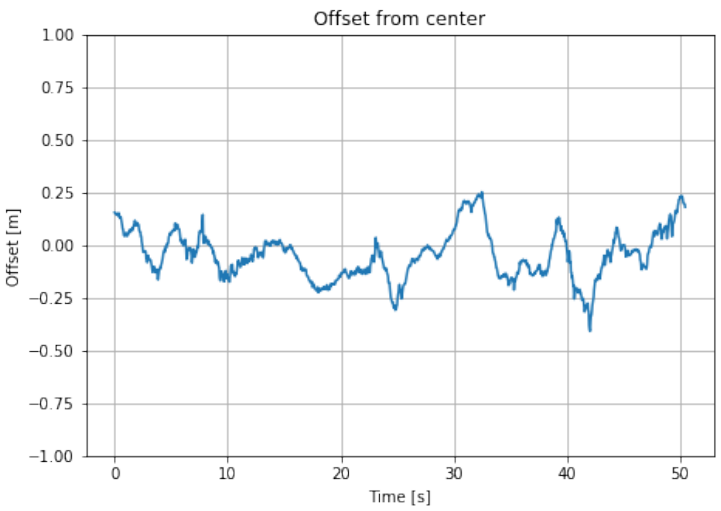
In [15]:

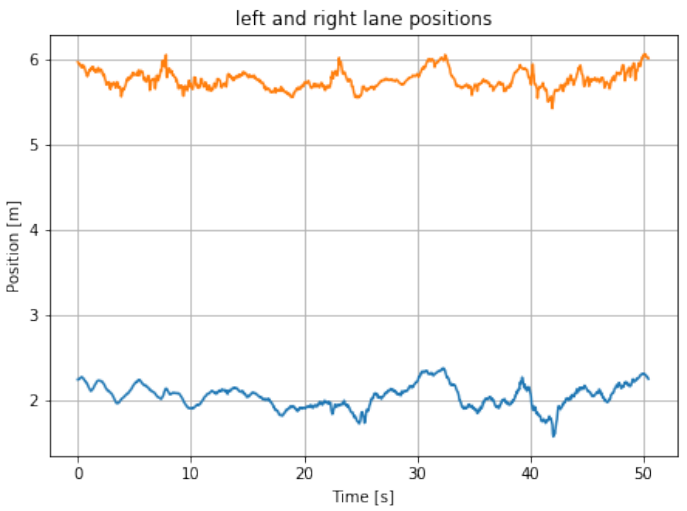
```
import imageio
imageio.plugins.ffmpeg.download()
from moviepy.editor import VideoFileClip

init_pipeline()
left_curve_values=np.array([],np.float32)
right_curve_values=np.array([],np.float32)
offcenter_values=np.array([],np.float32)

clip1 = VideoFileClip('project_video.mp4')
out = clip1.fl_image(pipeline)
out.write_videofile('result.mp4', audio=False)
plot_curvatures()
```

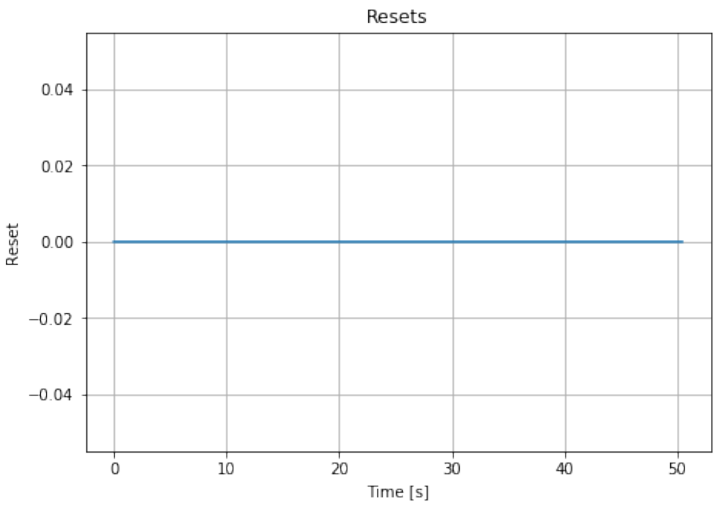
```
[MoviePy] >>>> Building video result.mp4
[MoviePy] Writing video result.mp4

100%|████████████| 1260/1261 [03:20<00:00,  4.90it/s]

[MoviePy] Done.
[MoviePy] >>>> Video ready: result.mp4
```

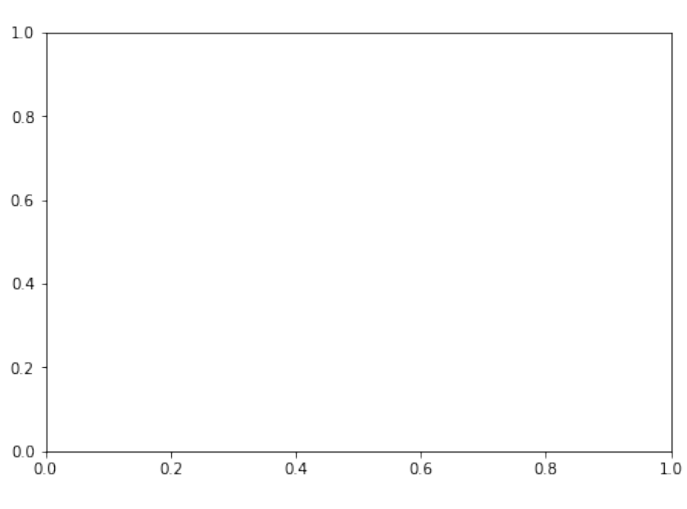












Process the challenge videos

In [16]:

```
init_pipeline()

#clip1 = VideoFileClip('challenge_video.mp4').subclip(0,10)
#out = clip1.fl_image(pipeline)
#out.write_videofile('challenge_result.mp4', audio=False)

#plot_curvatures()

#init_pipeline()
#clip1 = VideoFileClip('harder_challenge_video.mp4').subclip(0,10)
#out = clip1.fl_image(pipeline)
#out.write_videofile('harder_challenge_result.mp4', audio=False)

#plot_curvatures()
```

In [ ]:

```

```