CarnND Advance Lanes

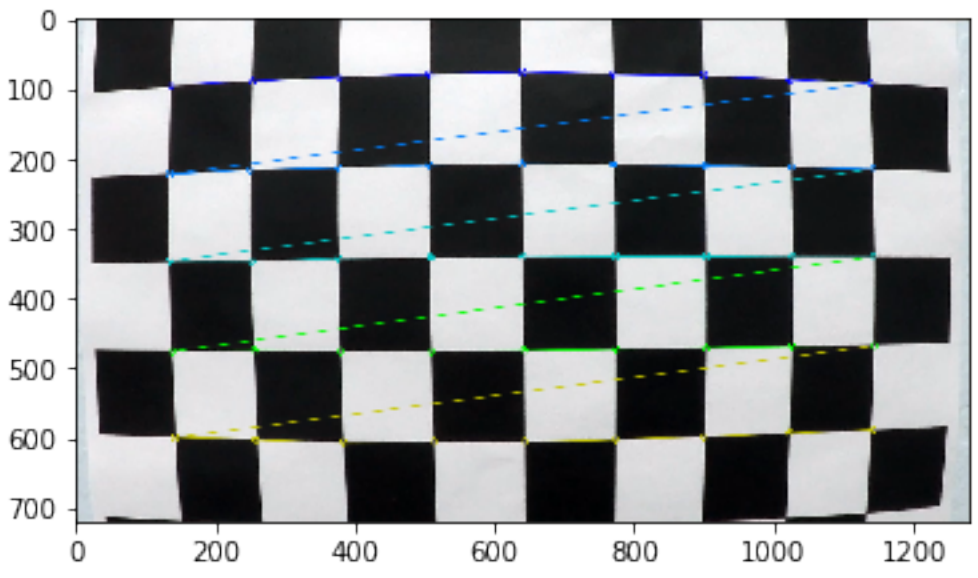
Author: Igor Passchier

Compute the camera calibration matrix and distortion coefficients

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

In [2]:

```
%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 [3]:

```
# 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 [4]:

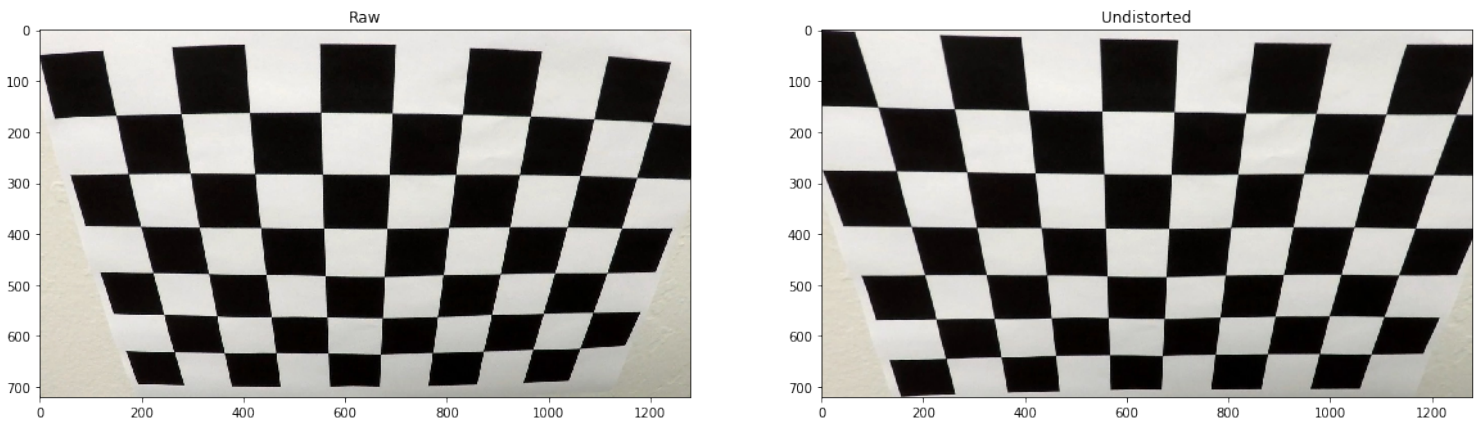
```
#Plot 2 images next to each other
def plot2(img1,img2,title1=None,title2=None,grey1=False,grey2=False):
    f, ax = plt.subplots(1,2, figsize=(20, 10))
    if (grey1):
        ax[0].imshow(img1,cmap='gray')
    else:
        ax[0].imshow(img1)
    if (title1 != None):
        ax[0].set_title(title1)
    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 [5]:

```
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 [6]:

```
#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

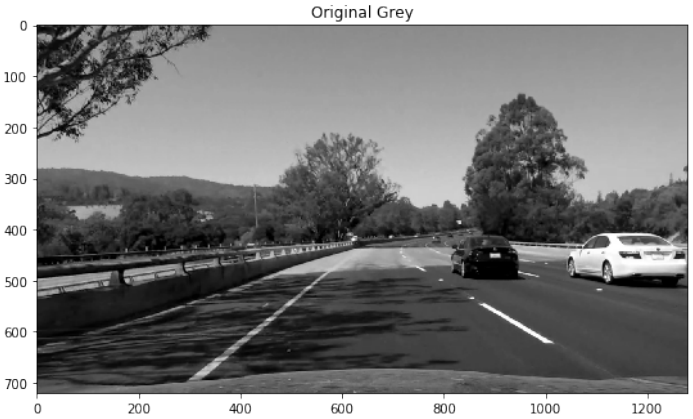
The final function to create the binary image with all the (possible) lane pixels, is `filter(img)`

In [7]:

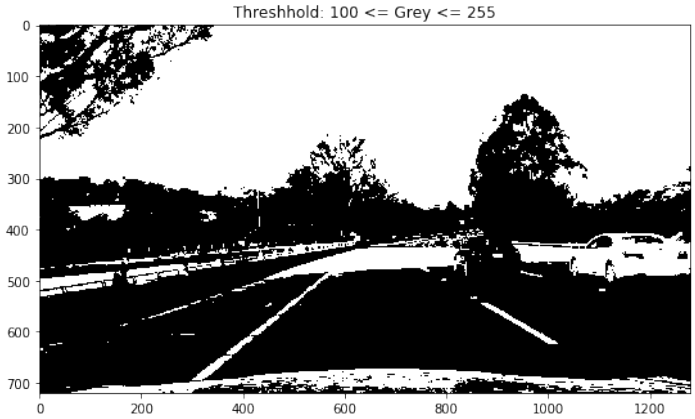
```
#Helper function to threshold range to greyscale image. If a name is given,
#then the original and filtered images are also plotted
def apply_threshold(img,tmin,tmax,name=None,plot=False):
    res=np.zeros_like(img)
    res[(img>=tmin)&(img<=tmax)]=1
    if (plot):
        f, ax = plt.subplots(1,2, 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)
```

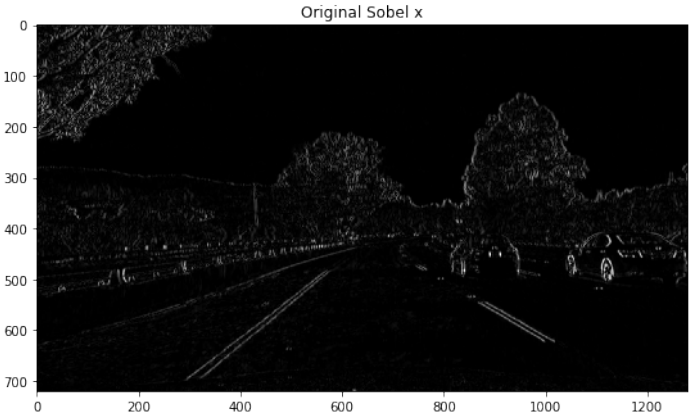
Original Grey



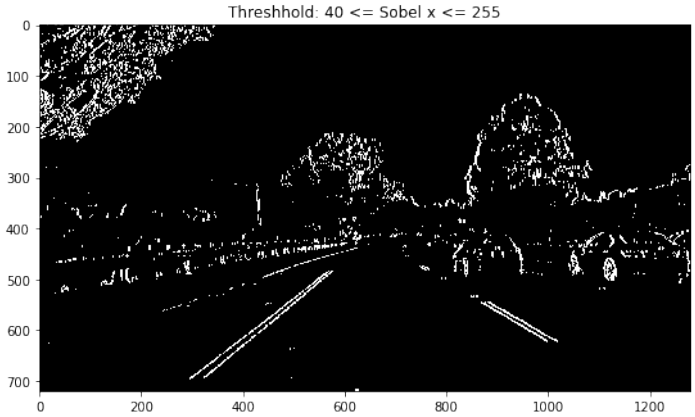
Threshhold: 100 <= Grey <= 255



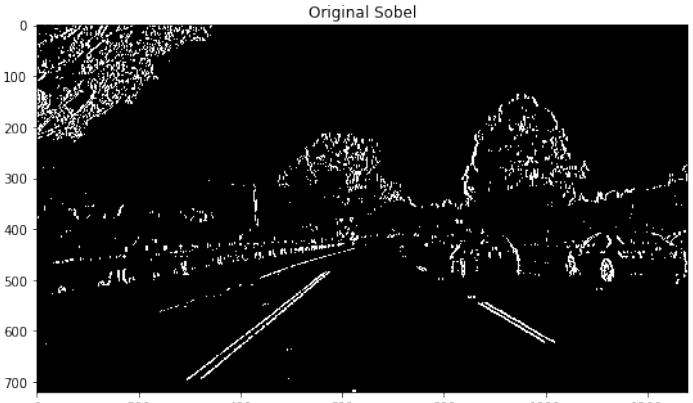
Original Sobel x



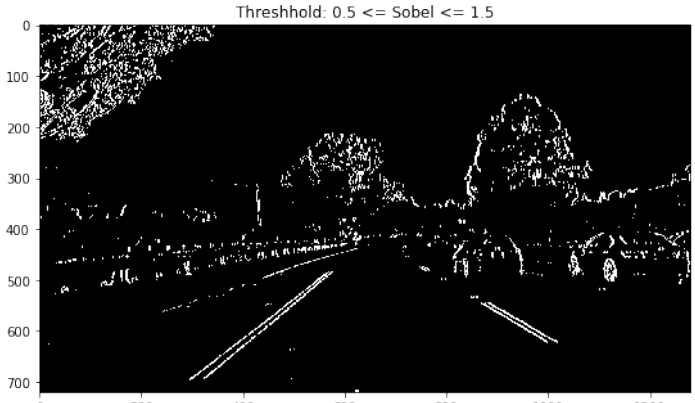
Threshhold: 40 <= Sobel x <= 255



Original Sobel

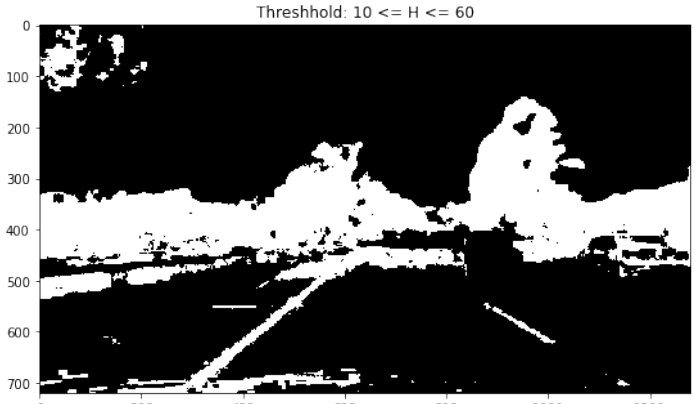


Threshhold: 0.5 <= Sobel <= 1.5



Original H

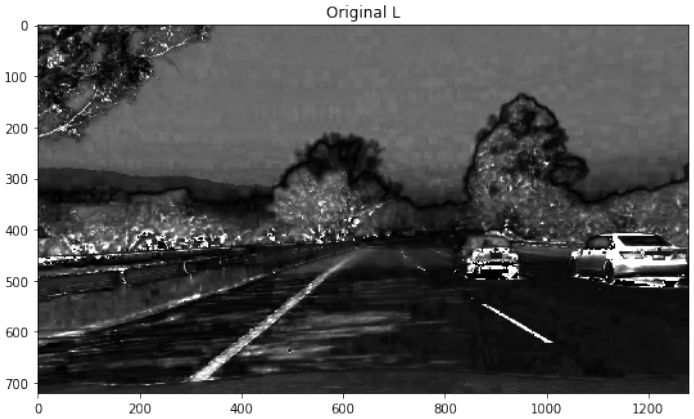
Threshhold: 10 <= H <= 60



Original S

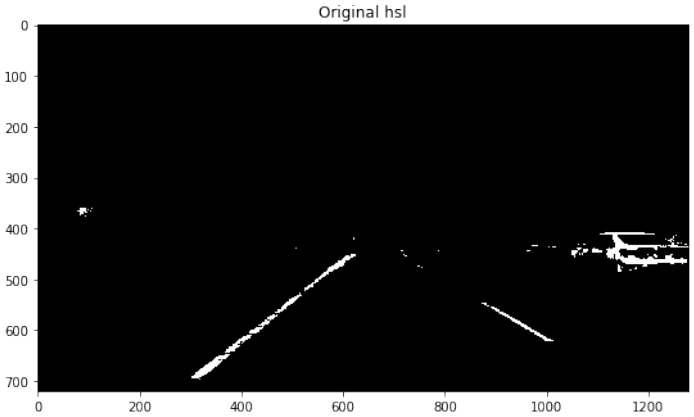
Threshhold: 90 <= S <= 255

Original L

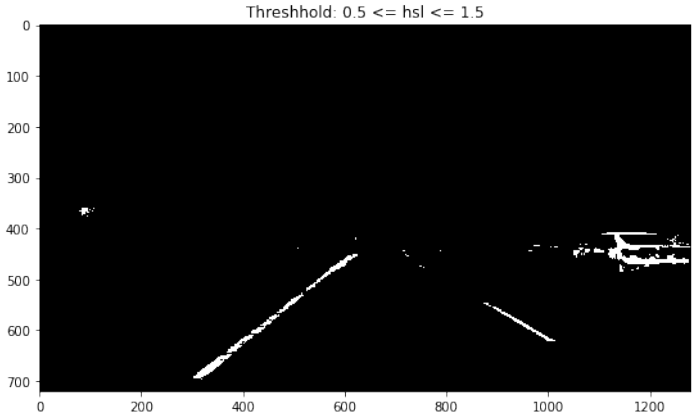


Threshhold: 90 <= L <= 255

Original hsl

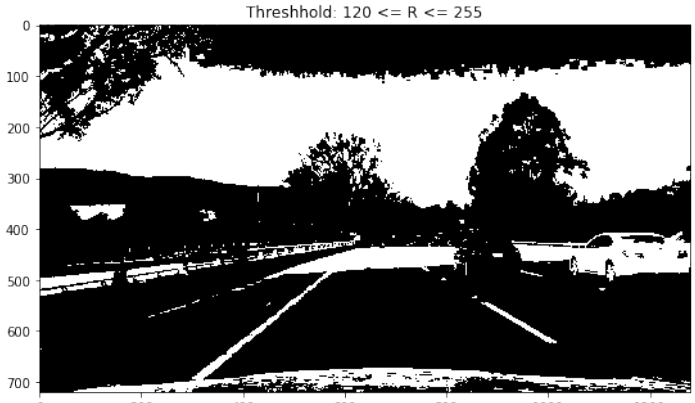


Threshhold: 0.5 <= hsl <= 1.5



Original R

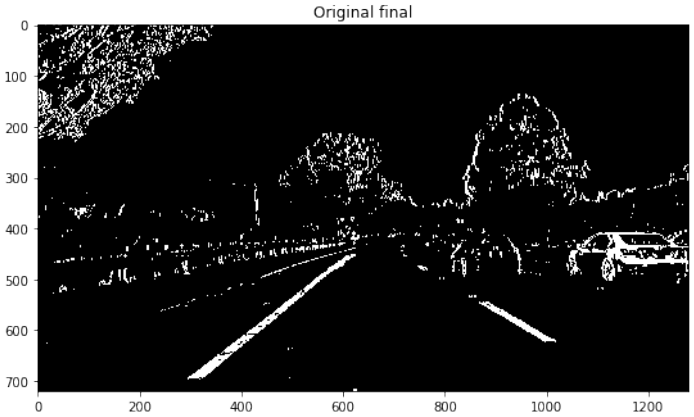
Threshhold: 120 <= R <= 255



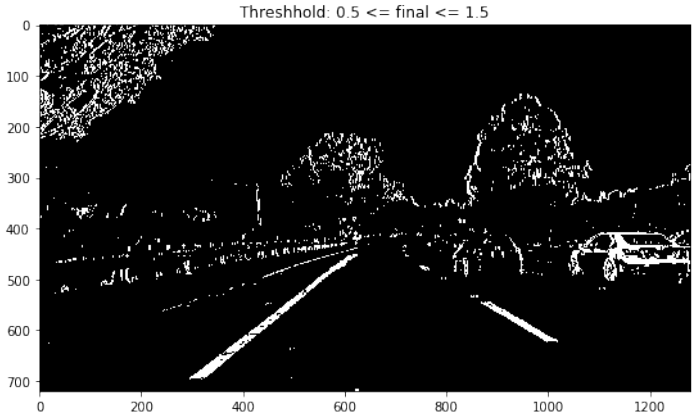
Original rgb

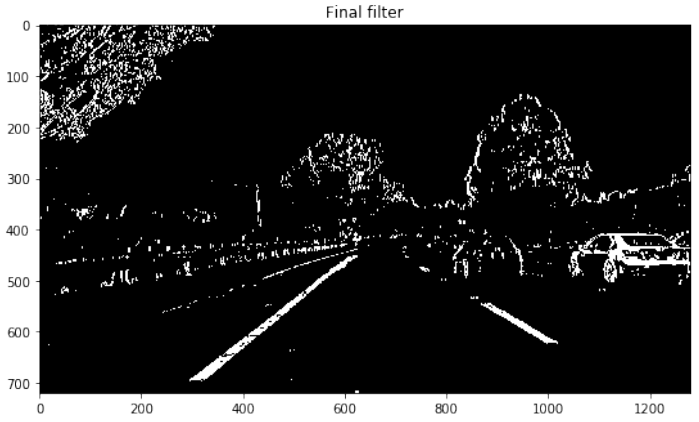
Threshhold: 0.5 <= rgb <= 1.5

Original final



Threshhold: 0.5 <= final <= 1.5



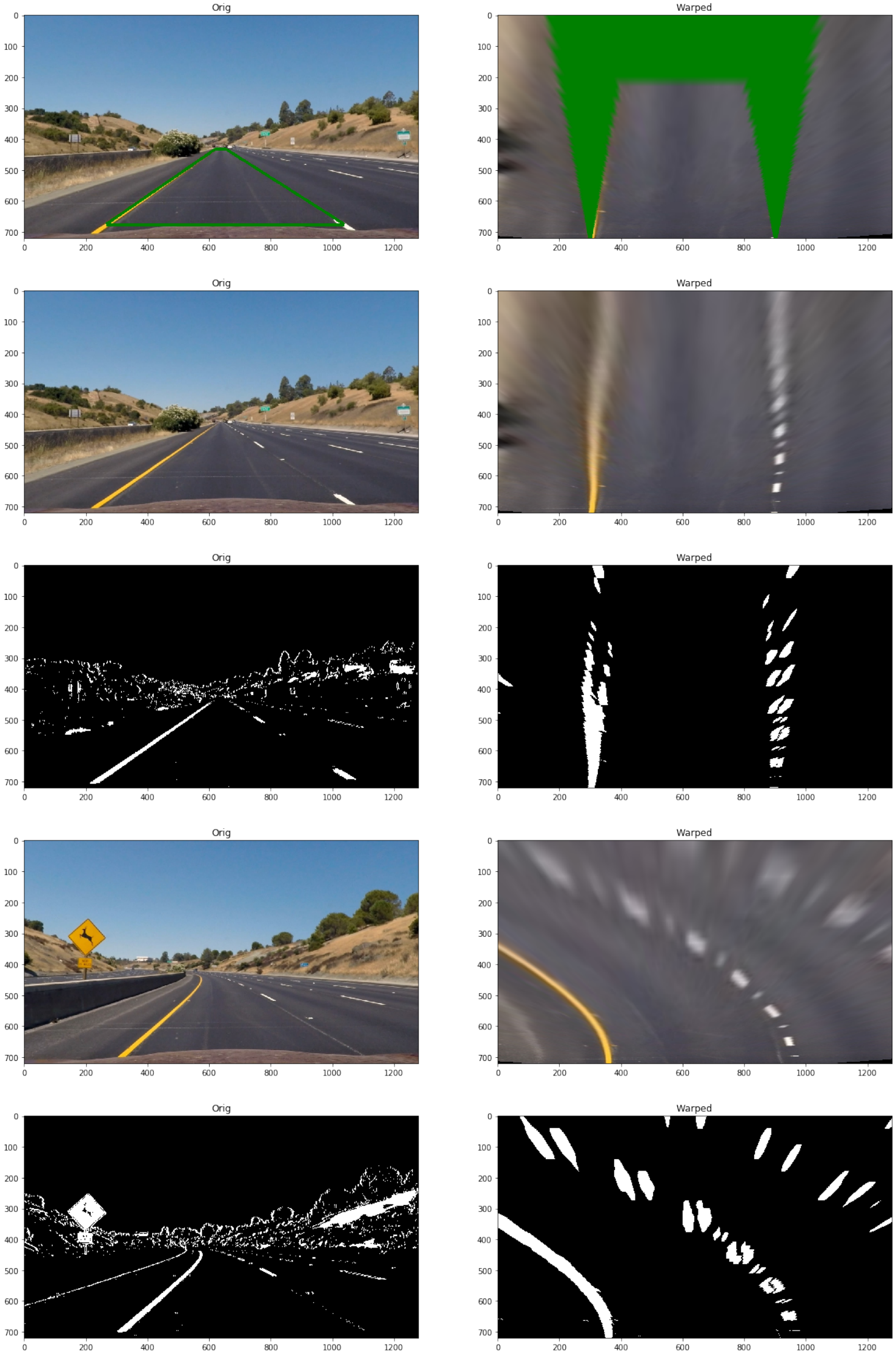


Birds-eye view

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 [8]:

```
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

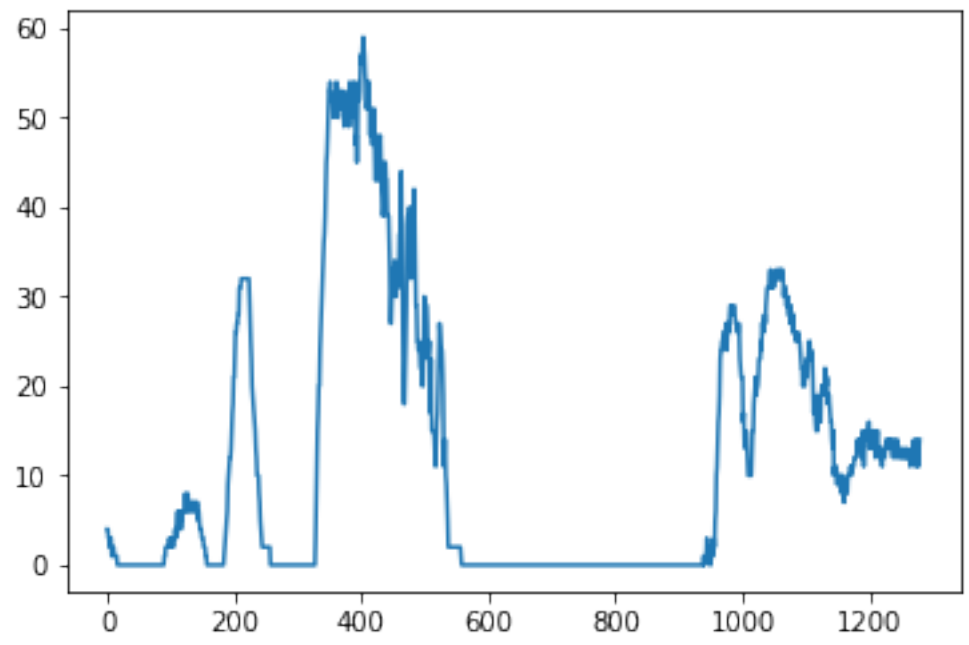
In [88]:

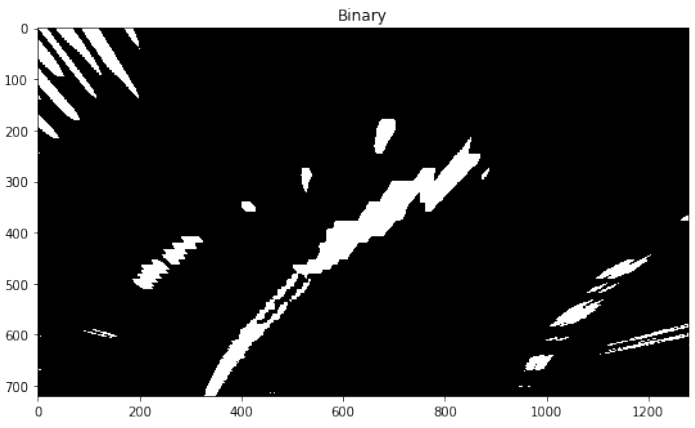
```
def calculate_windows(shape,xlcenter,xrcenter,margin):
```

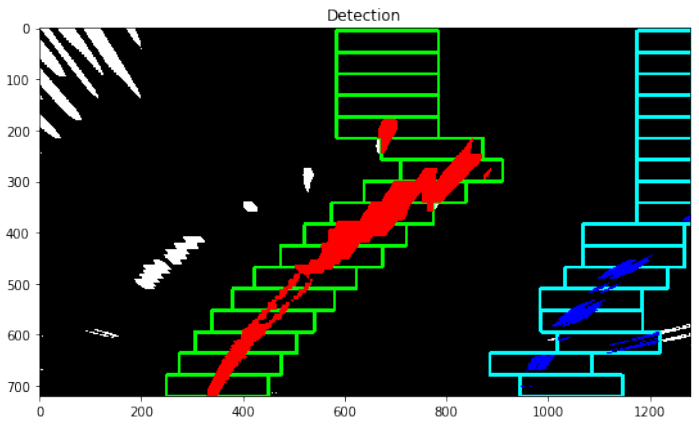
```
    xllow = max(min(xlcenter - margin,shape[1]),0)

    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)
```





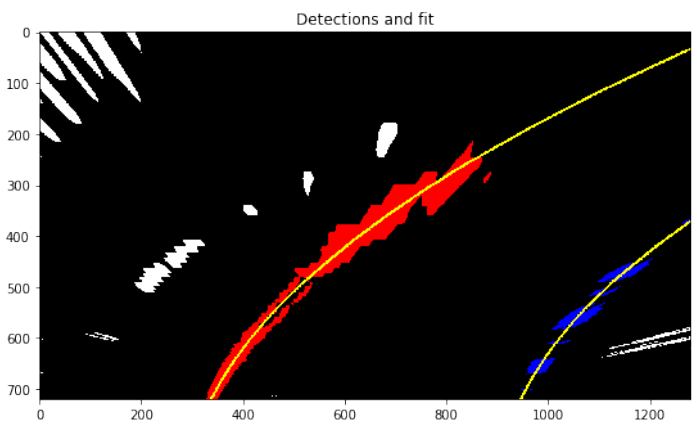


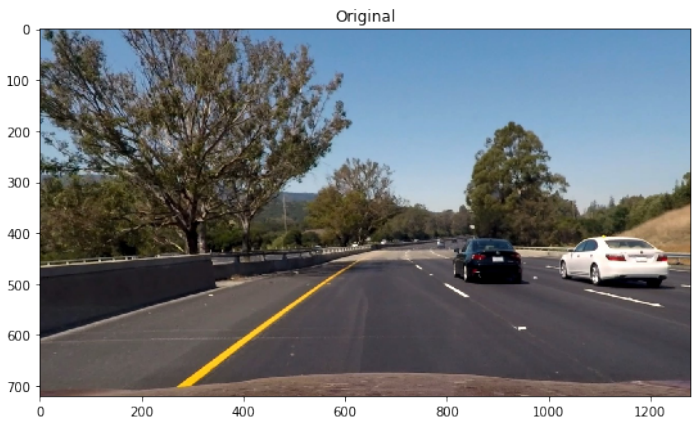
Determine the curvature of the lane and vehicle position with respect to center.

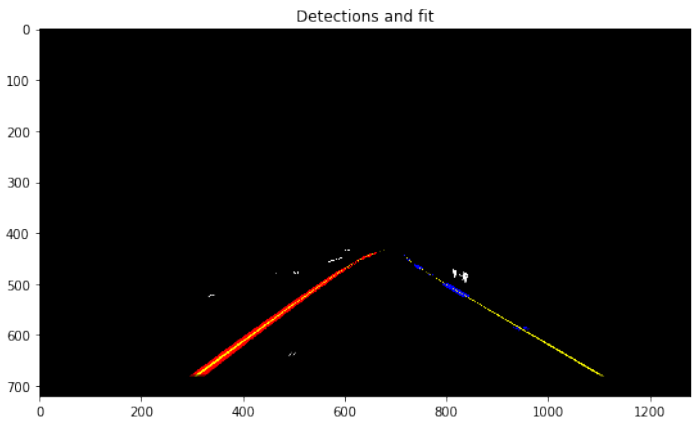
In [92]:

```
#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 [39]:

```
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)
```

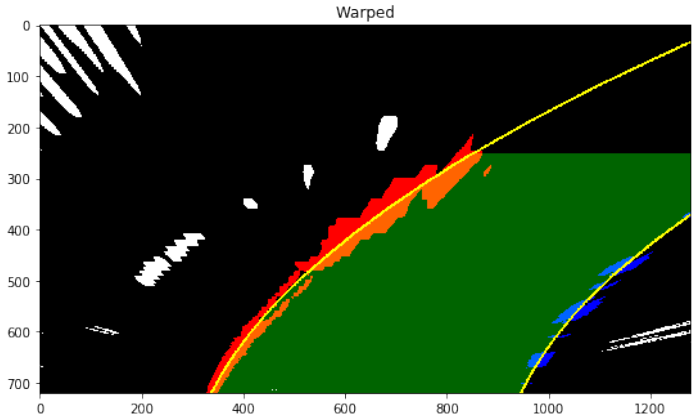
```
Curvatures 1/pix:  608.409118574 463.617922448
Off center pix  :  2.59165493217
Curvatures 1/m:  246.208729101 205.390335256
Off center m  :  0.0159818720817
```

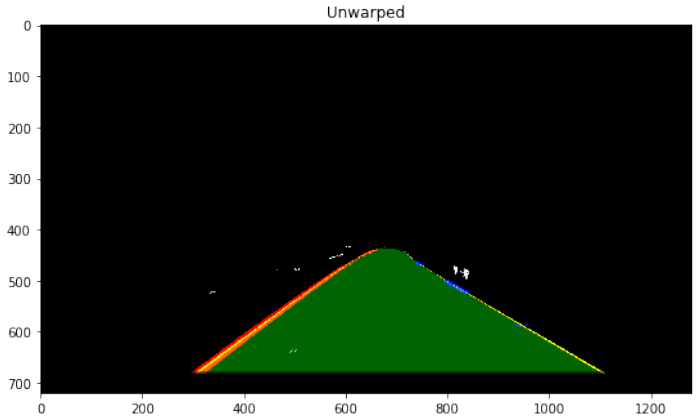
Warp back to original image

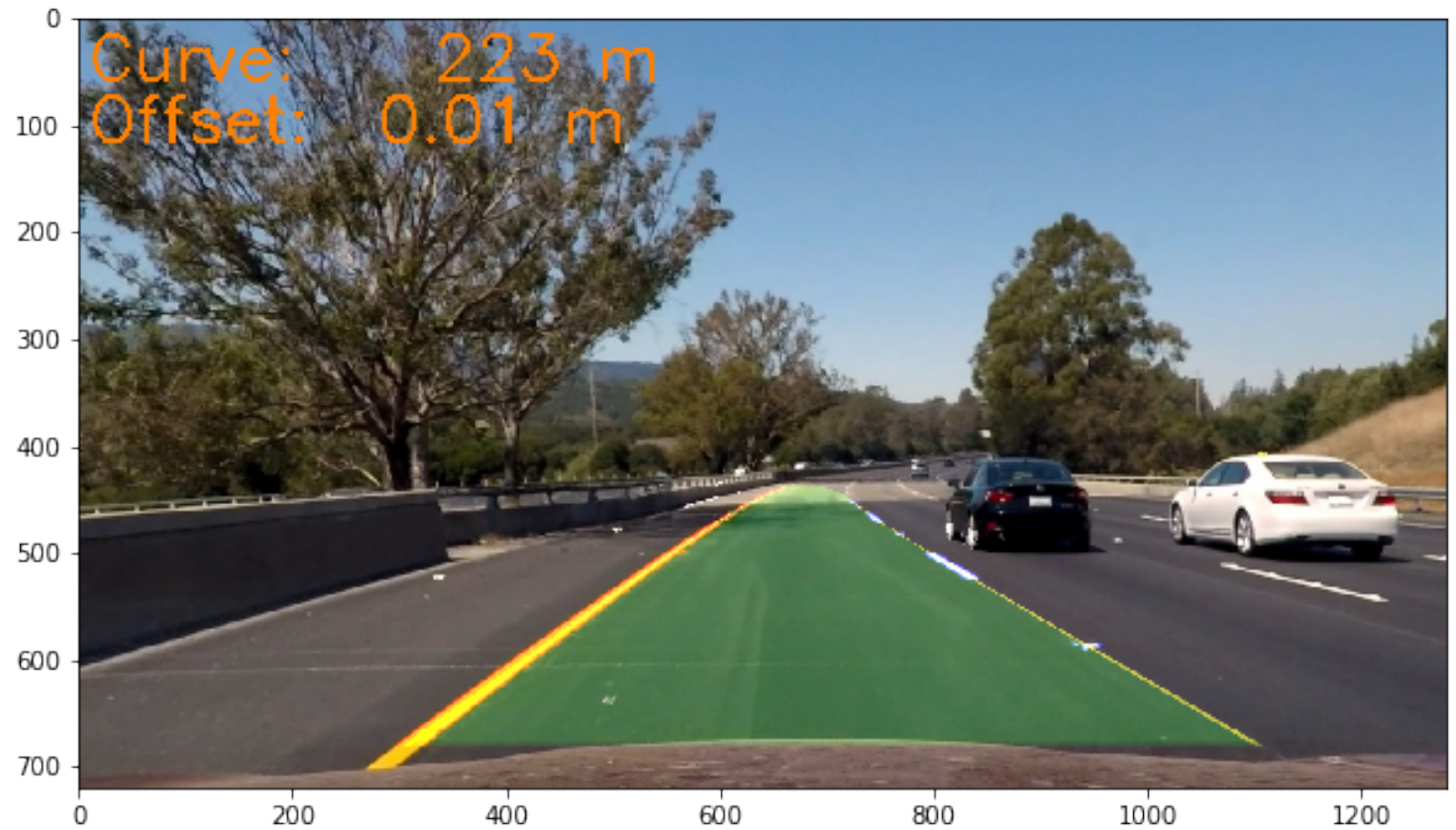
In [93]:

```
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

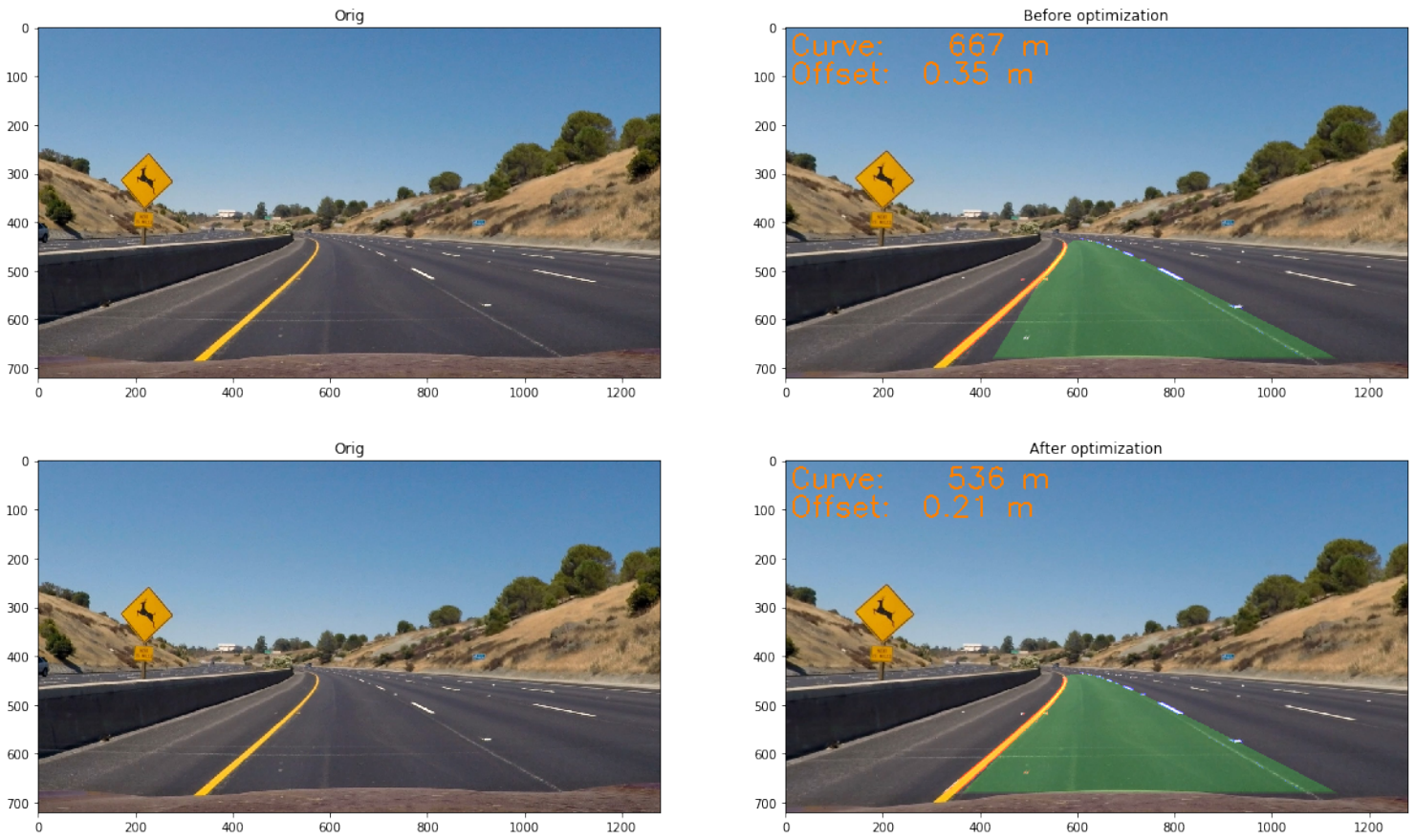
Input is 1 image Output is an overlayed image, the left and right lane curverture, and the distance from center

In [81]:

```
#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')
```

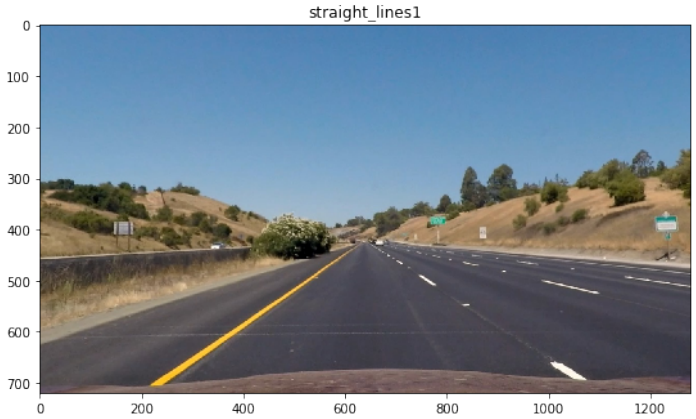


Process all test images with complete pipeline

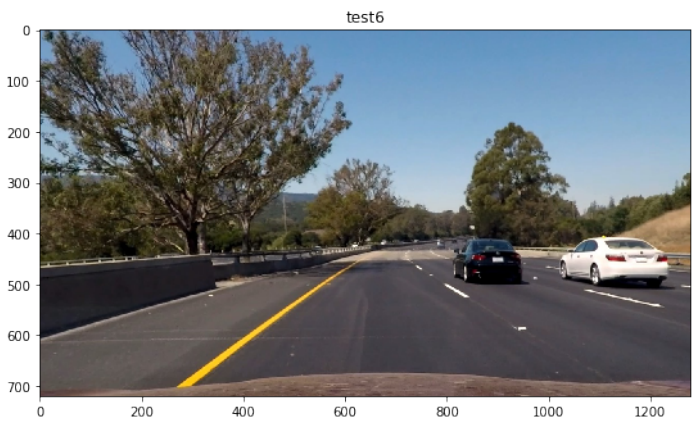
All images are processed twice: the first time to find the fit lanes, the second time to go through the optimized pipeline

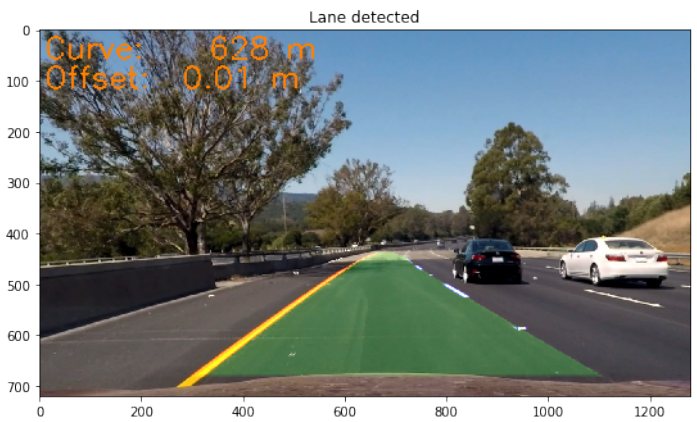
In [82]:

```
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

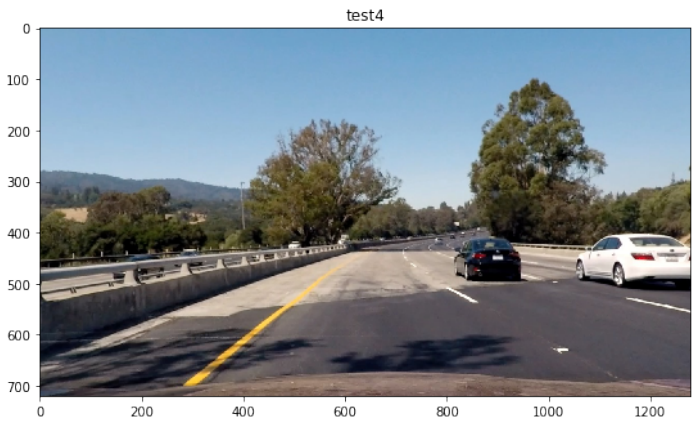

Lane detected
Curve: 14615 m
Offset: −0.23 m


test6

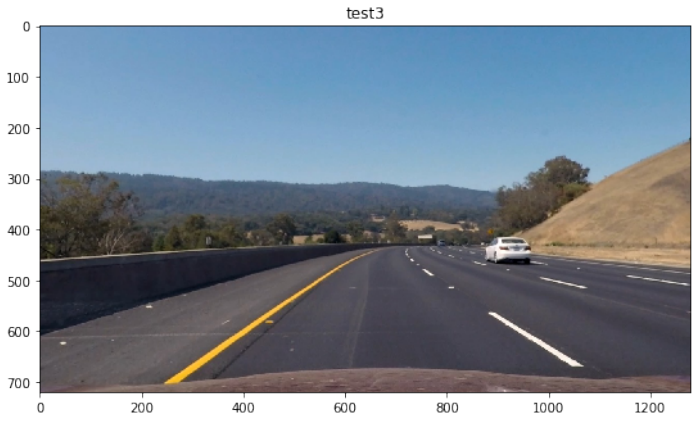

Lane detected
Curve: 628 m
Offset: 0.01 m

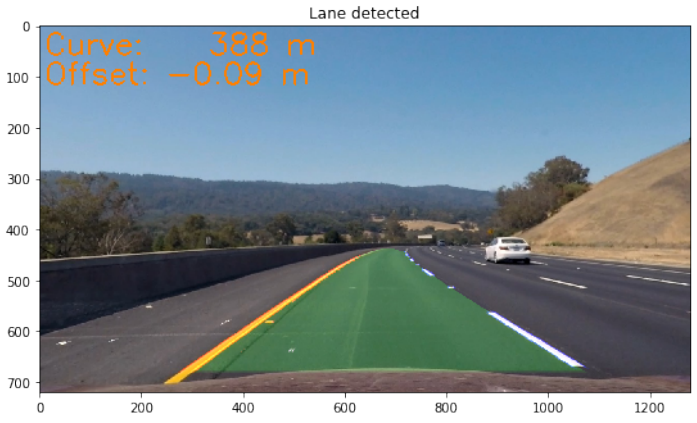

extra1


Lane detected
Curve: 460 m
Offset: 0.00 m


test4


Lane detected
Curve: 424 m
Offset: 0.07 m


test3


Lane detected
Curve: 388 m
Offset: −0.09 m


test5


Lane detected
Curve: 340 m
Offset: −0.25 m


straight_lines2


Lane detected
Curve: 6592 m
Offset: −0.25 m


test1


Lane detected
Curve: 813 m
Offset: −0.11 m

In [83]:

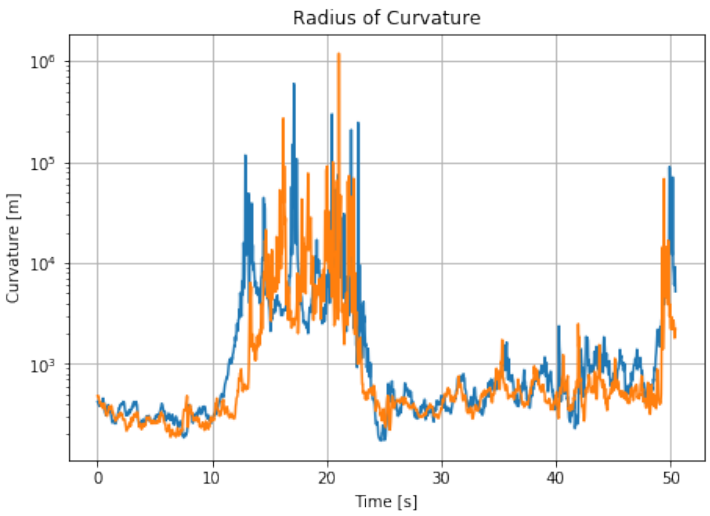
```
#plot the curve of position results. Assume 25 frames/sec
def plot_curvatures():
    x=np.linspace(0, (len(left_curve_values))/25, len(left_curve_values) )

    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()
```

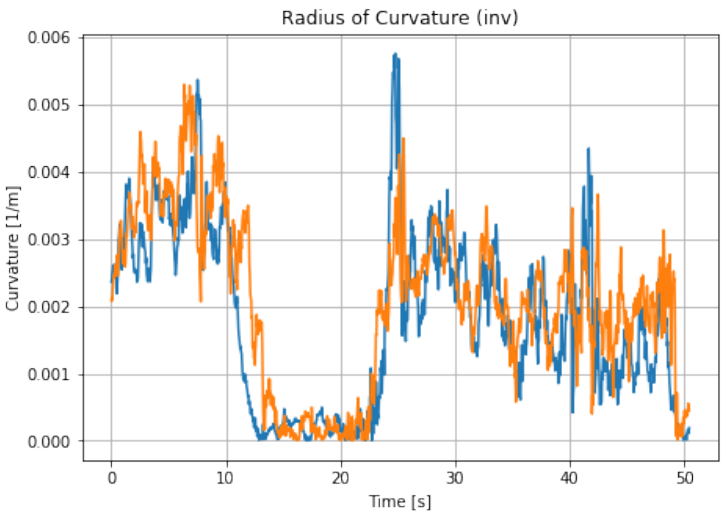
Process the project video

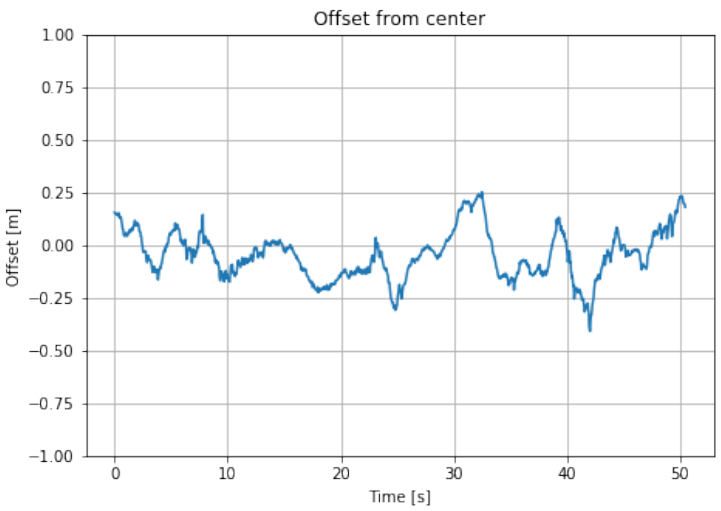
In [84]:

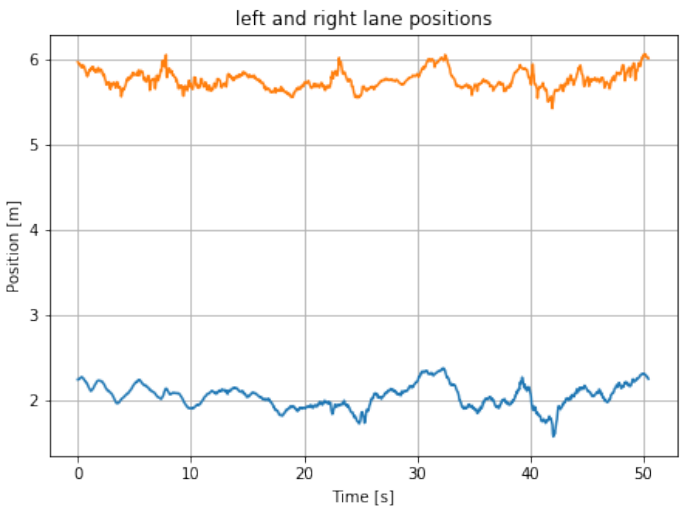
```
import imageio
imageio.plugins.ffmpeg.download()
from moviepy.editor import VideoFileClip

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()
```

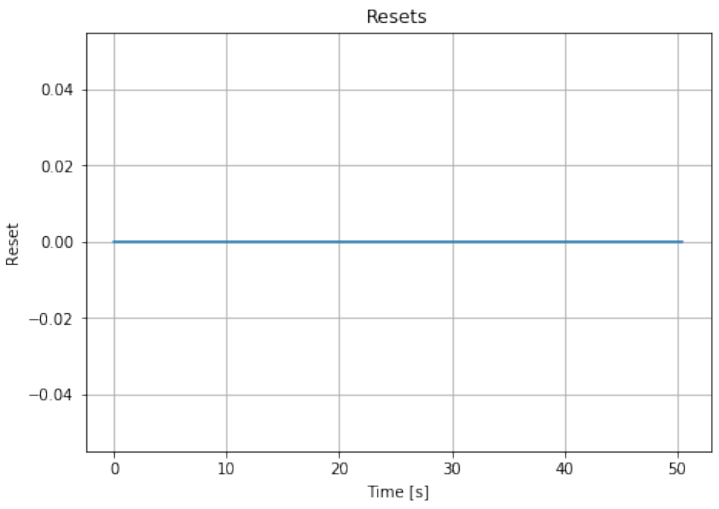
```
[MoviePy] >>>> Building video result.mp4
[MoviePy] Writing video result.mp4

100%|██████████| 1260/1261 [03:28<00:00,  5.00it/s]

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

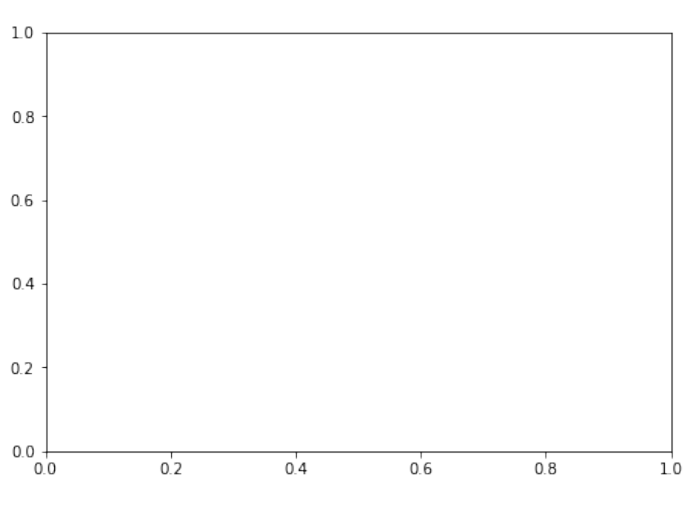












Process the challenge videos

In [86]:

```
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()
```

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

  0%|          | 0/300 [00:00<?, ?it/s]
  0%|          | 1/300 [00:00<01:06,  4.51it/s]
  1%|          | 2/300 [00:00<01:05,  4.57it/s]
  1%|          | 3/300 [00:00<01:04,  4.62it/s]
  1%|▏         | 4/300 [00:00<01:06,  4.44it/s]
  2%|▏         | 5/300 [00:01<01:07,  4.40it/s]
  2%|▏         | 6/300 [00:01<01:12,  4.08it/s]
  2%|▏         | 7/300 [00:01<01:12,  4.02it/s]
  3%|▎         | 8/300 [00:01<01:12,  4.04it/s]
  3%|▎         | 9/300 [00:02<01:09,  4.21it/s]
  3%|▎         | 10/300 [00:02<01:07,  4.28it/s]
  4%|▎         | 11/300 [00:02<01:09,  4.14it/s]
  4%|▍         | 12/300 [00:02<01:08,  4.22it/s]
  4%|▍         | 13/300 [00:03<01:06,  4.31it/s]
  5%|▍         | 14/300 [00:03<01:07,  4.24it/s]
  5%|▌         | 15/300 [00:03<01:06,  4.26it/s]
  5%|▌         | 16/300 [00:03<01:04,  4.37it/s]
  6%|▌         | 17/300 [00:03<01:03,  4.45it/s]
  6%|▌         | 18/300 [00:04<01:02,  4.51it/s]
  6%|▋         | 19/300 [00:04<01:03,  4.40it/s]
  7%|▋         | 20/300 [00:04<01:09,  4.06it/s]
  7%|▋         | 21/300 [00:04<01:11,  3.90it/s]
  7%|▋         | 22/300 [00:05<01:13,  3.80it/s]
  8%|▊         | 23/300 [00:05<01:16,  3.62it/s]
  8%|▊         | 24/300 [00:05<01:17,  3.56it/s]
  8%|▊         | 25/300 [00:06<01:20,  3.43it/s]
  9%|▊         | 26/300 [00:06<01:18,  3.48it/s]
  9%|▉         | 27/300 [00:06<01:17,  3.51it/s]
  9%|▉         | 28/300 [00:07<01:16,  3.55it/s]
 10%|▉         | 29/300 [00:07<01:29,  3.03it/s]
 10%|▉         | 30/300 [00:07<01:37,  2.78it/s]
 10%|█         | 31/300 [00:08<01:35,  2.81it/s]
 11%|█         | 32/300 [00:08<01:28,  3.04it/s]
 11%|█         | 33/300 [00:08<01:24,  3.15it/s]
 11%|█▏        | 34/300 [00:09<01:20,  3.32it/s]
 12%|█▏        | 35/300 [00:09<01:21,  3.27it/s]
 12%|█▏        | 36/300 [00:09<01:22,  3.19it/s]
```

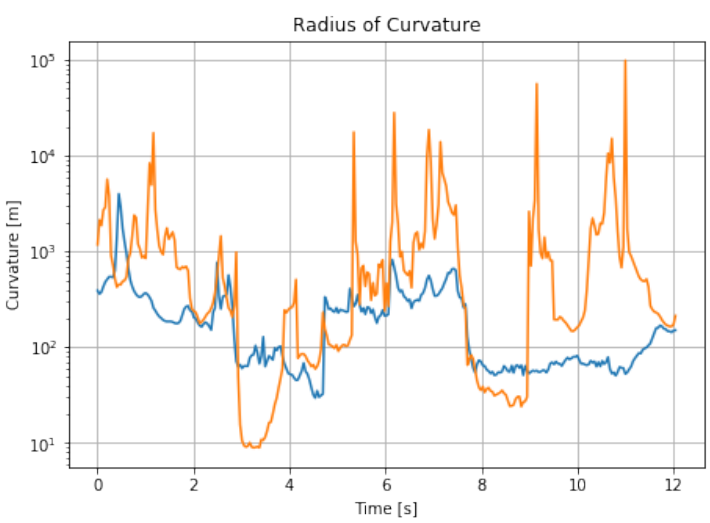
12%|▉ | 37/300 [00:09<01:20, 3.27it/s]
13%|▉ | 38/300 [00:10<01:14, 3.53it/s]
13%|▉ | 39/300 [00:10<01:12, 3.59it/s]
13%|▉ | 40/300 [00:10<01:11, 3.65it/s]
14%|▉ | 41/300 [00:10<01:07, 3.83it/s]
14%|▉ | 42/300 [00:11<01:05, 3.95it/s]
14%|▉ | 43/300 [00:11<01:03, 4.05it/s]
15%|▉ | 44/300 [00:11<01:02, 4.09it/s]
15%|▉ | 45/300 [00:11<01:00, 4.21it/s]
15%|▉ | 46/300 [00:12<01:04, 3.94it/s]
16%|▉ | 47/300 [00:12<01:05, 3.87it/s]
16%|▉ | 48/300 [00:12<01:02, 4.06it/s]
16%|▉ | 49/300 [00:12<00:59, 4.21it/s]
17%|▉ | 50/300 [00:13<00:55, 4.50it/s]
17%|▉ | 51/300 [00:13<00:57, 4.32it/s]
17%|▉ | 52/300 [00:13<00:56, 4.36it/s]
18%|▉ | 53/300 [00:13<00:54, 4.51it/s]
18%|▉ | 54/300 [00:13<00:52, 4.69it/s]
18%|▉ | 55/300 [00:14<00:50, 4.88it/s]
19%|▉ | 56/300 [00:14<00:46, 5.30it/s]
19%|▉ | 57/300 [00:14<00:43, 5.55it/s]
19%|▉ | 58/300 [00:14<00:41, 5.79it/s]
20%|▉ | 59/300 [00:14<00:41, 5.87it/s]
20%|▉ | 60/300 [00:14<00:40, 5.98it/s]
20%|▉ | 61/300 [00:15<00:39, 6.09it/s]
21%|▉ | 62/300 [00:15<00:38, 6.21it/s]
21%|▉ | 63/300 [00:15<00:40, 5.86it/s]
21%|▉ | 64/300 [00:15<00:42, 5.51it/s]
22%|▉ | 65/300 [00:15<00:42, 5.47it/s]
22%|▉ | 66/300 [00:16<00:44, 5.20it/s]
22%|▉ | 67/300 [00:16<00:46, 4.96it/s]
23%|▉ | 68/300 [00:16<00:46, 4.98it/s]
23%|▉ | 69/300 [00:16<00:47, 4.91it/s]
23%|▉ | 70/300 [00:16<00:46, 4.90it/s]
24%|▉ | 71/300 [00:17<00:48, 4.73it/s]
24%|▉ | 72/300 [00:17<00:47, 4.77it/s]
24%|▉ | 73/300 [00:17<00:46, 4.87it/s]
25%|▉ | 74/300 [00:17<00:46, 4.89it/s]
25%|▉ | 75/300 [00:17<00:49, 4.53it/s]
25%|▉ | 76/300 [00:18<00:48, 4.61it/s]
26%|▉ | 77/300 [00:18<00:49, 4.55it/s]
26%|▉ | 78/300 [00:18<00:47, 4.69it/s]
26%|▉ | 79/300 [00:18<00:49, 4.43it/s]
27%|▉ | 80/300 [00:19<00:48, 4.51it/s]
27%|▉ | 81/300 [00:19<00:47, 4.59it/s]
27%|▉ | 82/300 [00:19<00:44, 4.88it/s]
28%|▉ | 83/300 [00:19<00:42, 5.07it/s]
28%|▉ | 84/300 [00:19<00:43, 4.99it/s]
28%|▉ | 85/300 [00:20<00:41, 5.13it/s]
29%|▉ | 86/300 [00:20<00:41, 5.22it/s]
29%|▉ | 87/300 [00:20<00:38, 5.47it/s]
29%|▉ | 88/300 [00:20<00:38, 5.53it/s]
30%|▉ | 89/300 [00:20<00:40, 5.25it/s]
30%|▉ | 90/300 [00:20<00:39, 5.31it/s]
30%|▉ | 91/300 [00:21<00:39, 5.33it/s]
31%|▉ | 92/300 [00:21<00:38, 5.43it/s]
31%|▉ | 93/300 [00:21<00:37, 5.52it/s]

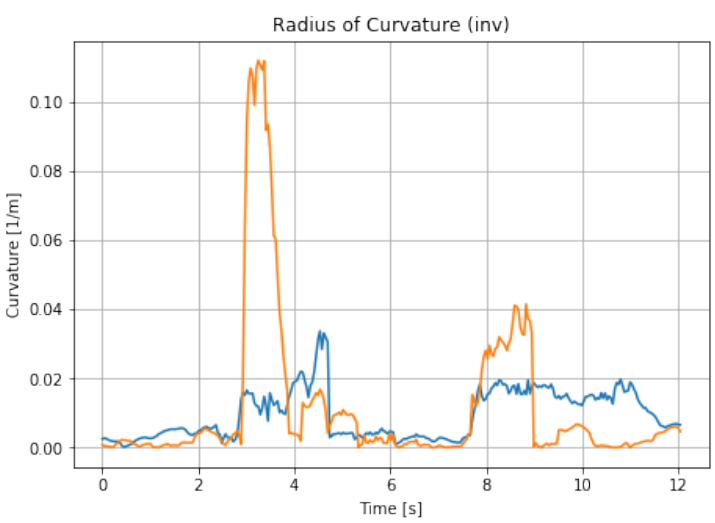
31%|███▏ | 94/300 [00:21<00:38, 5.29it/s]
32%|███▏ | 95/300 [00:21<00:39, 5.23it/s]
32%|███▏ | 96/300 [00:22<00:40, 5.02it/s]
32%|███▏ | 97/300 [00:22<00:42, 4.83it/s]
33%|███▎ | 98/300 [00:22<00:41, 4.84it/s]
33%|███▎ | 99/300 [00:22<00:46, 4.35it/s]
33%|███▎ | 100/300 [00:23<00:45, 4.42it/s]
34%|███▎ | 101/300 [00:23<00:43, 4.55it/s]
34%|███▍ | 102/300 [00:23<00:40, 4.88it/s]
34%|███▍ | 103/300 [00:23<00:37, 5.25it/s]
35%|███▍ | 104/300 [00:23<00:34, 5.60it/s]
35%|███▌ | 105/300 [00:23<00:34, 5.70it/s]
35%|███▌ | 106/300 [00:24<00:32, 5.93it/s]
36%|███▌ | 107/300 [00:24<00:31, 6.07it/s]
36%|███▌ | 108/300 [00:24<00:31, 6.12it/s]
36%|███▋ | 109/300 [00:24<00:31, 6.16it/s]
37%|███▋ | 110/300 [00:24<00:30, 6.27it/s]
37%|███▋ | 111/300 [00:24<00:30, 6.30it/s]
37%|███▋ | 112/300 [00:25<00:31, 5.92it/s]
38%|███▊ | 113/300 [00:25<00:31, 5.85it/s]
38%|███▊ | 114/300 [00:25<00:31, 5.84it/s]
38%|███▊ | 115/300 [00:25<00:31, 5.85it/s]
39%|███▊ | 116/300 [00:25<00:29, 6.18it/s]
39%|███▉ | 117/300 [00:25<00:28, 6.48it/s]
39%|███▉ | 118/300 [00:25<00:26, 6.95it/s]
40%|███▉ | 119/300 [00:26<00:27, 6.54it/s]
40%|████ | 120/300 [00:26<00:27, 6.48it/s]
40%|████ | 121/300 [00:26<00:26, 6.73it/s]
41%|████ | 122/300 [00:26<00:25, 6.95it/s]
41%|████ | 123/300 [00:26<00:24, 7.15it/s]
41%|████▏ | 124/300 [00:26<00:23, 7.57it/s]
42%|████▏ | 125/300 [00:26<00:22, 7.90it/s]
42%|████▏ | 126/300 [00:27<00:21, 8.14it/s]
42%|████▏ | 127/300 [00:27<00:20, 8.24it/s]
43%|████▎ | 128/300 [00:27<00:21, 8.06it/s]
43%|████▎ | 129/300 [00:27<00:23, 7.25it/s]
43%|████▎ | 130/300 [00:27<00:23, 7.31it/s]
44%|████▎ | 131/300 [00:27<00:22, 7.55it/s]
44%|████▍ | 132/300 [00:27<00:22, 7.50it/s]
44%|████▍ | 133/300 [00:27<00:22, 7.47it/s]
45%|████▍ | 134/300 [00:28<00:21, 7.57it/s]
45%|████▌ | 135/300 [00:28<00:21, 7.68it/s]
45%|████▌ | 136/300 [00:28<00:21, 7.71it/s]
46%|████▌ | 137/300 [00:28<00:22, 7.20it/s]
46%|████▌ | 138/300 [00:28<00:23, 6.84it/s]
46%|████▋ | 139/300 [00:28<00:24, 6.66it/s]
47%|████▋ | 140/300 [00:28<00:24, 6.52it/s]
47%|████▋ | 141/300 [00:29<00:24, 6.53it/s]
47%|████▋ | 142/300 [00:29<00:23, 6.62it/s]
48%|████▊ | 143/300 [00:29<00:23, 6.77it/s]
48%|████▊ | 144/300 [00:29<00:22, 6.95it/s]
48%|████▊ | 145/300 [00:29<00:22, 6.84it/s]
49%|████▊ | 146/300 [00:29<00:22, 6.92it/s]
49%|████▉ | 147/300 [00:29<00:21, 6.98it/s]
49%|████▉ | 148/300 [00:30<00:21, 7.02it/s]
50%|████▉ | 149/300 [00:30<00:21, 6.86it/s]
50%|█████ | 150/300 [00:30<00:21, 7.00it/s]

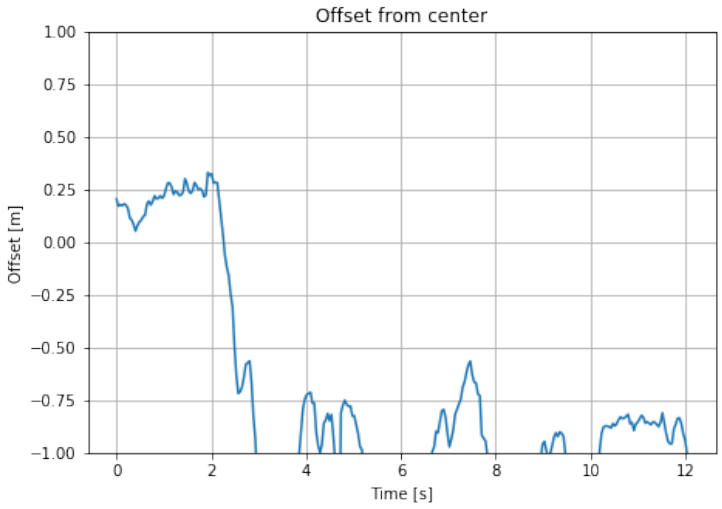
50%|██████ | 151/300 [00:30<00:21, 6.85it/s]
51%|██████ | 152/300 [00:30<00:21, 6.79it/s]
51%|██████ | 153/300 [00:30<00:21, 6.92it/s]
51%|██████ | 154/300 [00:31<00:21, 6.85it/s]
52%|██████ | 155/300 [00:31<00:20, 6.91it/s]
52%|██████ | 156/300 [00:31<00:20, 7.04it/s]
52%|██████ | 157/300 [00:31<00:20, 7.05it/s]
53%|██████ | 158/300 [00:31<00:20, 7.02it/s]
53%|██████ | 159/300 [00:31<00:20, 6.95it/s]
53%|██████ | 160/300 [00:31<00:20, 6.96it/s]
54%|██████ | 161/300 [00:32<00:19, 7.01it/s]
54%|██████ | 162/300 [00:32<00:19, 6.96it/s]
54%|██████ | 163/300 [00:32<00:20, 6.83it/s]
55%|██████ | 164/300 [00:32<00:20, 6.78it/s]
55%|██████ | 165/300 [00:32<00:23, 5.72it/s]
55%|██████ | 166/300 [00:32<00:25, 5.16it/s]
56%|██████ | 167/300 [00:33<00:26, 4.99it/s]
56%|██████ | 168/300 [00:33<00:27, 4.78it/s]
56%|██████ | 169/300 [00:33<00:28, 4.68it/s]
57%|██████ | 170/300 [00:33<00:27, 4.67it/s]
57%|██████ | 171/300 [00:34<00:30, 4.29it/s]
57%|██████ | 172/300 [00:34<00:30, 4.25it/s]
58%|██████ | 173/300 [00:34<00:28, 4.38it/s]
58%|██████ | 174/300 [00:34<00:27, 4.51it/s]
58%|██████ | 175/300 [00:34<00:28, 4.42it/s]
59%|██████ | 176/300 [00:35<00:28, 4.40it/s]
59%|██████ | 177/300 [00:35<00:30, 4.04it/s]
59%|██████ | 178/300 [00:35<00:31, 3.92it/s]
60%|██████ | 179/300 [00:36<00:30, 4.00it/s]
60%|██████ | 180/300 [00:36<00:29, 4.13it/s]
60%|██████ | 181/300 [00:36<00:27, 4.37it/s]
61%|██████ | 182/300 [00:36<00:26, 4.48it/s]
61%|██████ | 183/300 [00:36<00:26, 4.34it/s]
61%|██████ | 184/300 [00:37<00:25, 4.52it/s]
62%|██████ | 185/300 [00:37<00:24, 4.78it/s]
62%|██████ | 186/300 [00:37<00:22, 5.06it/s]
62%|██████ | 187/300 [00:37<00:21, 5.32it/s]
63%|██████ | 188/300 [00:37<00:20, 5.56it/s]
63%|██████ | 189/300 [00:37<00:20, 5.55it/s]
63%|██████ | 190/300 [00:38<00:19, 5.60it/s]
64%|██████ | 191/300 [00:38<00:19, 5.68it/s]
64%|██████ | 192/300 [00:38<00:18, 5.88it/s]
64%|██████ | 193/300 [00:38<00:17, 6.01it/s]
65%|██████ | 194/300 [00:38<00:17, 6.06it/s]
65%|██████ | 195/300 [00:38<00:17, 6.06it/s]
65%|██████ | 196/300 [00:39<00:17, 5.93it/s]
66%|██████ | 197/300 [00:39<00:17, 5.80it/s]
66%|██████ | 198/300 [00:39<00:19, 5.26it/s]
66%|██████ | 199/300 [00:39<00:19, 5.13it/s]
67%|██████ | 200/300 [00:39<00:19, 5.07it/s]
67%|██████ | 201/300 [00:40<00:19, 5.19it/s]
67%|██████ | 202/300 [00:40<00:18, 5.33it/s]
68%|██████ | 203/300 [00:40<00:18, 5.27it/s]
68%|██████ | 204/300 [00:40<00:19, 5.05it/s]
68%|██████ | 205/300 [00:40<00:20, 4.56it/s]
69%|██████ | 206/300 [00:41<00:20, 4.49it/s]
69%|██████ | 207/300 [00:41<00:21, 4.28it/s]

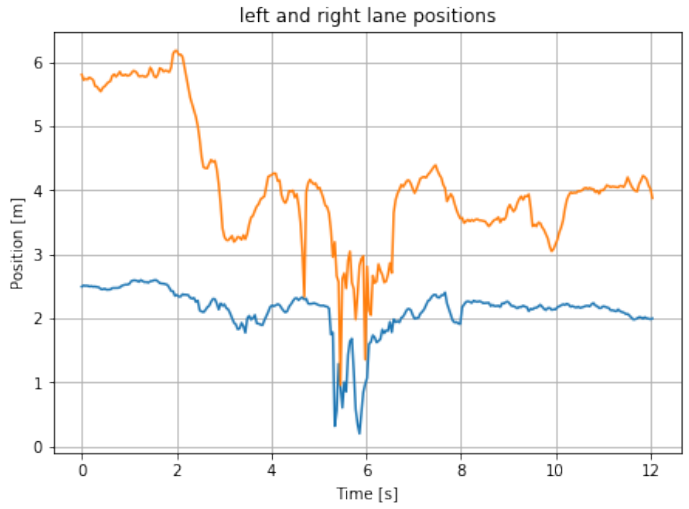
```
 69%|██████▉   | 208/300 [00:41<00:21,  4.19it/s]
 70%|██████▉   | 209/300 [00:41<00:20,  4.34it/s]
 70%|███████   | 210/300 [00:42<00:21,  4.25it/s]
 70%|███████   | 211/300 [00:42<00:23,  3.85it/s]
 71%|███████   | 212/300 [00:42<00:22,  3.88it/s]
 71%|███████   | 213/300 [00:42<00:21,  3.97it/s]
 71%|███████▏  | 214/300 [00:43<00:21,  4.09it/s]
 72%|███████▏  | 215/300 [00:43<00:21,  3.97it/s]
 72%|███████▏  | 216/300 [00:43<00:22,  3.69it/s]
 72%|███████▏  | 217/300 [00:44<00:21,  3.80it/s]
 73%|███████▎  | 218/300 [00:44<00:20,  3.96it/s]
 73%|███████▎  | 219/300 [00:44<00:19,  4.21it/s]
 73%|███████▎  | 220/300 [00:44<00:20,  3.89it/s]
 74%|███████▎  | 221/300 [00:45<00:20,  3.89it/s]
 74%|███████▍  | 222/300 [00:45<00:19,  4.03it/s]
 74%|███████▍  | 223/300 [00:45<00:18,  4.17it/s]
 75%|███████▍  | 224/300 [00:45<00:17,  4.31it/s]
 75%|███████▌  | 225/300 [00:46<00:18,  3.95it/s]
 75%|███████▌  | 226/300 [00:46<00:19,  3.80it/s]
 76%|███████▌  | 227/300 [00:46<00:18,  3.90it/s]
 76%|███████▌  | 228/300 [00:46<00:19,  3.75it/s]
 76%|███████▋  | 229/300 [00:47<00:20,  3.49it/s]
 77%|███████▋  | 230/300 [00:47<00:20,  3.41it/s]
 77%|███████▋  | 231/300 [00:47<00:19,  3.55it/s]
 77%|███████▋  | 232/300 [00:48<00:19,  3.50it/s]
 78%|███████▊  | 233/300 [00:48<00:19,  3.51it/s]
 78%|███████▊  | 234/300 [00:48<00:17,  3.72it/s]
 78%|███████▊  | 235/300 [00:48<00:17,  3.66it/s]
 79%|███████▊  | 236/300 [00:49<00:17,  3.75it/s]
 79%|███████▉  | 237/300 [00:49<00:17,  3.70it/s]
 79%|███████▉  | 238/300 [00:49<00:17,  3.52it/s]
 80%|███████▉  | 239/300 [00:49<00:18,  3.31it/s]
 80%|████████  | 240/300 [00:50<00:18,  3.24it/s]
 80%|████████  | 241/300 [00:50<00:19,  3.02it/s]
 81%|████████  | 242/300 [00:51<00:18,  3.08it/s]
 81%|████████  | 243/300 [00:51<00:17,  3.28it/s]
 81%|████████▏ | 244/300 [00:51<00:17,  3.21it/s]
 82%|████████▏ | 245/300 [00:51<00:16,  3.31it/s]
 82%|████████▏ | 246/300 [00:52<00:15,  3.49it/s]
 82%|████████▏ | 247/300 [00:52<00:14,  3.54it/s]
 83%|████████▎ | 248/300 [00:52<00:14,  3.48it/s]
 83%|████████▎ | 249/300 [00:52<00:14,  3.46it/s]
 83%|████████▎ | 250/300 [00:53<00:13,  3.68it/s]
 84%|████████▎ | 251/300 [00:53<00:13,  3.74it/s]
 84%|████████▍ | 252/300 [00:53<00:13,  3.54it/s]
 84%|████████▍ | 253/300 [00:54<00:13,  3.56it/s]
 85%|████████▍ | 254/300 [00:54<00:12,  3.72it/s]
 85%|████████▌ | 255/300 [00:54<00:12,  3.54it/s]
 85%|████████▌ | 256/300 [00:54<00:11,  3.75it/s]
 86%|████████▌ | 257/300 [00:55<00:11,  3.87it/s]
 86%|████████▌ | 258/300 [00:55<00:11,  3.78it/s]
 86%|████████▋ | 259/300 [00:55<00:11,  3.54it/s]
 87%|████████▋ | 260/300 [00:55<00:10,  3.70it/s]
 87%|████████▋ | 261/300 [00:56<00:10,  3.82it/s]
 87%|████████▋ | 262/300 [00:56<00:09,  4.00it/s]
 88%|████████▊ | 263/300 [00:56<00:09,  3.74it/s]
 88%|████████▊ | 264/300 [00:56<00:09,  3.77it/s]
```

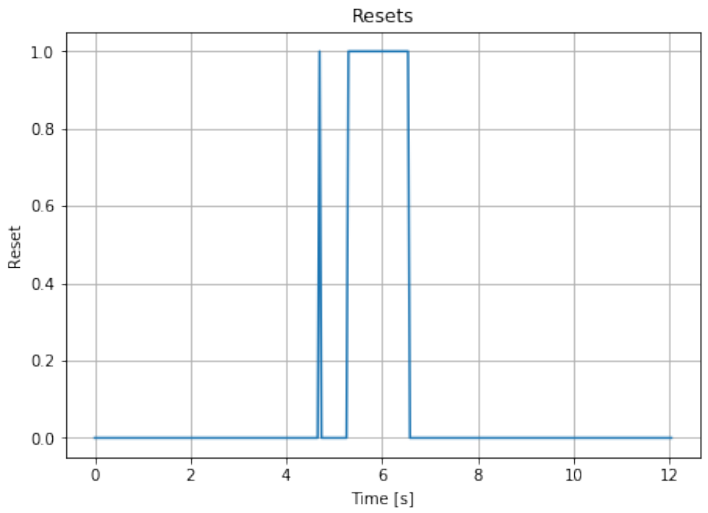
```
 88%|████████▊  | 265/300 [00:57<00:09,  3.84it/s]
 89%|████████▉  | 266/300 [00:57<00:08,  3.95it/s]
 89%|████████▉  | 267/300 [00:57<00:08,  3.78it/s]
 89%|████████▉  | 268/300 [00:58<00:08,  3.89it/s]
 90%|█████████  | 269/300 [00:58<00:07,  4.07it/s]
 90%|█████████  | 270/300 [00:58<00:06,  4.35it/s]
 90%|█████████  | 271/300 [00:58<00:06,  4.46it/s]
 91%|█████████  | 272/300 [00:58<00:06,  4.16it/s]
 91%|█████████  | 273/300 [00:59<00:06,  4.15it/s]
 91%|█████████▏ | 274/300 [00:59<00:06,  4.16it/s]
 92%|█████████▏ | 275/300 [00:59<00:05,  4.30it/s]
 92%|█████████▏ | 276/300 [00:59<00:06,  3.97it/s]
 92%|█████████▏ | 277/300 [01:00<00:06,  3.64it/s]
 93%|█████████▎ | 278/300 [01:00<00:06,  3.54it/s]
 93%|█████████▎ | 279/300 [01:00<00:05,  3.60it/s]
 93%|█████████▎ | 280/300 [01:01<00:05,  3.41it/s]
 94%|█████████▎ | 281/300 [01:01<00:05,  3.30it/s]
 94%|█████████▍ | 282/300 [01:01<00:05,  3.36it/s]
 94%|█████████▍ | 283/300 [01:01<00:04,  3.61it/s]
 95%|█████████▍ | 284/300 [01:02<00:04,  3.96it/s]
 95%|█████████▌ | 285/300 [01:02<00:03,  3.81it/s]
 95%|█████████▌ | 286/300 [01:02<00:03,  3.62it/s]
 96%|█████████▌ | 287/300 [01:03<00:03,  3.71it/s]
 96%|█████████▌ | 288/300 [01:03<00:03,  3.88it/s]
 96%|█████████▋ | 289/300 [01:03<00:02,  3.97it/s]
 97%|█████████▋ | 290/300 [01:03<00:02,  3.78it/s]
 97%|█████████▋ | 291/300 [01:04<00:02,  3.70it/s]
 97%|█████████▋ | 292/300 [01:04<00:02,  3.72it/s]
 98%|█████████▊ | 293/300 [01:04<00:01,  3.74it/s]
 98%|█████████▊ | 294/300 [01:04<00:01,  3.89it/s]
 98%|█████████▊ | 295/300 [01:05<00:01,  3.96it/s]
 99%|█████████▊ | 296/300 [01:05<00:01,  3.87it/s]
 99%|█████████▉ | 297/300 [01:05<00:00,  3.77it/s]
 99%|█████████▉ | 298/300 [01:05<00:00,  3.67it/s]
100%|█████████▉ | 299/300 [01:06<00:00,  3.69it/s]
100%|██████████| 300/300 [01:06<00:00,  3.66it/s]

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

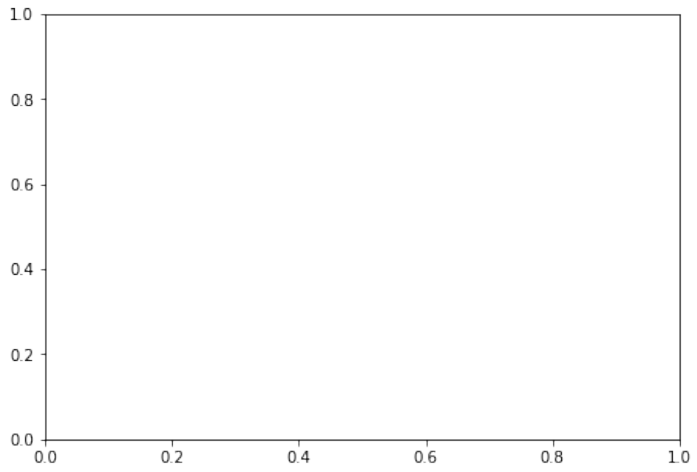












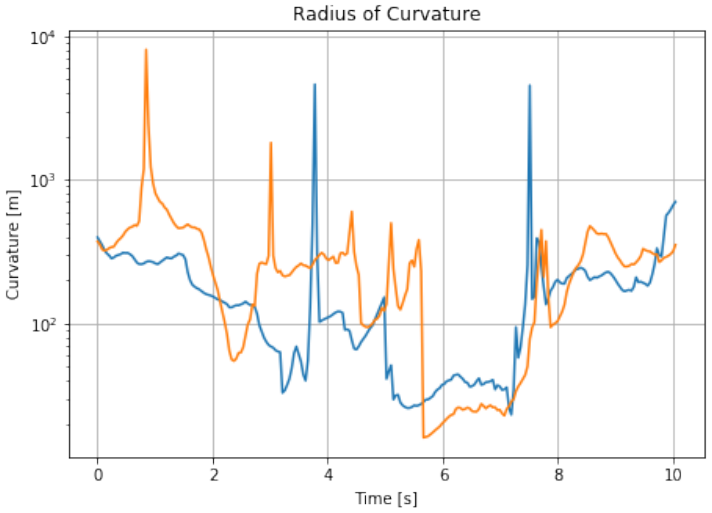
```
[MoviePy] >>>> Building video harder_challenge_result.mp4
[MoviePy] Writing video harder_challenge_result.mp4

  0%|          | 0/251 [00:00<?, ?it/s]
  0%|          | 1/251 [00:00<02:27,  1.70it/s]
  1%|          | 2/251 [00:01<02:34,  1.61it/s]
  1%|          | 3/251 [00:01<02:38,  1.56it/s]
  2%|▏         | 4/251 [00:02<02:38,  1.56it/s]
  2%|▏         | 5/251 [00:03<02:46,  1.48it/s]
  2%|▏         | 6/251 [00:04<02:45,  1.48it/s]
  3%|▎         | 7/251 [00:04<02:46,  1.46it/s]
  3%|▎         | 8/251 [00:05<02:36,  1.55it/s]
  4%|▎         | 9/251 [00:05<02:26,  1.66it/s]
  4%|▍         | 10/251 [00:06<02:13,  1.81it/s]
  4%|▍         | 11/251 [00:06<02:10,  1.84it/s]
  5%|▍         | 12/251 [00:07<02:02,  1.95it/s]
  5%|▌         | 13/251 [00:07<01:57,  2.02it/s]
  6%|▌         | 14/251 [00:08<01:49,  2.16it/s]
  6%|▌         | 15/251 [00:08<01:41,  2.33it/s]
  6%|▋         | 16/251 [00:08<01:33,  2.51it/s]
  7%|▋         | 17/251 [00:09<01:26,  2.71it/s]
  7%|▋         | 18/251 [00:09<01:22,  2.81it/s]
  8%|▊         | 19/251 [00:09<01:22,  2.82it/s]
  8%|▊         | 20/251 [00:10<01:21,  2.84it/s]
  8%|▊         | 21/251 [00:10<01:19,  2.90it/s]
  9%|▉         | 22/251 [00:10<01:16,  2.98it/s]
  9%|▉         | 23/251 [00:10<01:15,  3.04it/s]
 10%|▉         | 24/251 [00:11<01:14,  3.03it/s]
 10%|▉         | 25/251 [00:11<01:18,  2.88it/s]
 10%|█         | 26/251 [00:12<01:18,  2.86it/s]
 11%|█         | 27/251 [00:12<01:21,  2.76it/s]
 11%|█         | 28/251 [00:12<01:24,  2.63it/s]
 12%|█▏        | 29/251 [00:13<01:29,  2.48it/s]
```

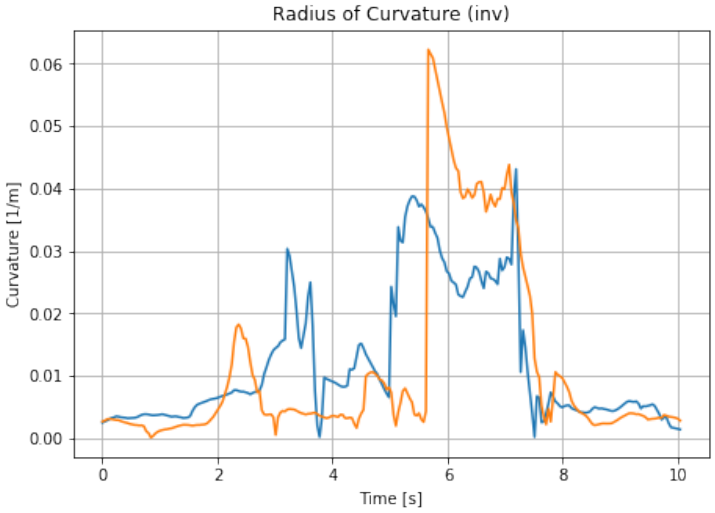
12%|▉ | 30/251 [00:13<01:34, 2.34it/s]
12%|▉ | 31/251 [00:14<01:38, 2.23it/s]
13%|▉ | 32/251 [00:14<01:45, 2.07it/s]
13%|▉ | 33/251 [00:15<01:55, 1.88it/s]
14%|▉ | 34/251 [00:16<01:55, 1.88it/s]
14%|▉ | 35/251 [00:16<01:53, 1.90it/s]
14%|▉ | 36/251 [00:17<01:51, 1.92it/s]
15%|▉ | 37/251 [00:17<02:03, 1.74it/s]
15%|▉ | 38/251 [00:18<02:10, 1.63it/s]
16%|▉ | 39/251 [00:19<02:15, 1.57it/s]
16%|▉ | 40/251 [00:20<02:30, 1.40it/s]
16%|▉ | 41/251 [00:20<02:38, 1.32it/s]
17%|▉ | 42/251 [00:21<02:44, 1.27it/s]
17%|▉ | 43/251 [00:22<02:43, 1.28it/s]
18%|▉ | 44/251 [00:23<02:39, 1.30it/s]
18%|▉ | 45/251 [00:24<02:37, 1.31it/s]
18%|▉ | 46/251 [00:24<02:42, 1.26it/s]
19%|▉ | 47/251 [00:25<02:38, 1.29it/s]
19%|▉ | 48/251 [00:26<02:30, 1.35it/s]
20%|▉ | 49/251 [00:27<02:27, 1.37it/s]
20%|▉ | 50/251 [00:27<02:24, 1.39it/s]
20%|▉ | 51/251 [00:28<02:22, 1.41it/s]
21%|▉ | 52/251 [00:29<02:18, 1.43it/s]
21%|▉ | 53/251 [00:29<02:15, 1.46it/s]
22%|▉ | 54/251 [00:30<02:07, 1.54it/s]
22%|▉ | 55/251 [00:30<01:58, 1.65it/s]
22%|▉ | 56/251 [00:31<01:53, 1.72it/s]
23%|▉ | 57/251 [00:31<01:47, 1.81it/s]
23%|▉ | 58/251 [00:32<01:43, 1.87it/s]
24%|▉ | 59/251 [00:32<01:39, 1.93it/s]
24%|▉ | 60/251 [00:33<01:37, 1.96it/s]
24%|▉ | 61/251 [00:33<01:38, 1.94it/s]
25%|▉ | 62/251 [00:34<01:34, 1.99it/s]
25%|▉ | 63/251 [00:34<01:30, 2.07it/s]
25%|▉ | 64/251 [00:35<01:28, 2.10it/s]
26%|▉ | 65/251 [00:35<01:25, 2.18it/s]
26%|▉ | 66/251 [00:35<01:21, 2.27it/s]
27%|▉ | 67/251 [00:36<01:16, 2.41it/s]
27%|▉ | 68/251 [00:36<01:11, 2.55it/s]
27%|▉ | 69/251 [00:37<01:10, 2.57it/s]
28%|▉ | 70/251 [00:37<01:09, 2.61it/s]
28%|▉ | 71/251 [00:37<01:07, 2.68it/s]
29%|▉ | 72/251 [00:38<01:05, 2.71it/s]
29%|▉ | 73/251 [00:38<01:03, 2.79it/s]
29%|▉ | 74/251 [00:38<01:04, 2.75it/s]
30%|▉ | 75/251 [00:39<01:04, 2.72it/s]
30%|▉ | 76/251 [00:39<01:03, 2.75it/s]
31%|▉ | 77/251 [00:39<01:01, 2.82it/s]
31%|▉ | 78/251 [00:40<01:01, 2.80it/s]
31%|▉ | 79/251 [00:40<01:01, 2.81it/s]
32%|▉ | 80/251 [00:40<01:00, 2.85it/s]
32%|▉ | 81/251 [00:41<00:57, 2.97it/s]
33%|▉ | 82/251 [00:41<00:56, 2.99it/s]
33%|▉ | 83/251 [00:41<00:57, 2.94it/s]
33%|▉ | 84/251 [00:42<00:57, 2.89it/s]
34%|▉ | 85/251 [00:42<00:53, 3.08it/s]
34%|▉ | 86/251 [00:42<00:52, 3.12it/s]

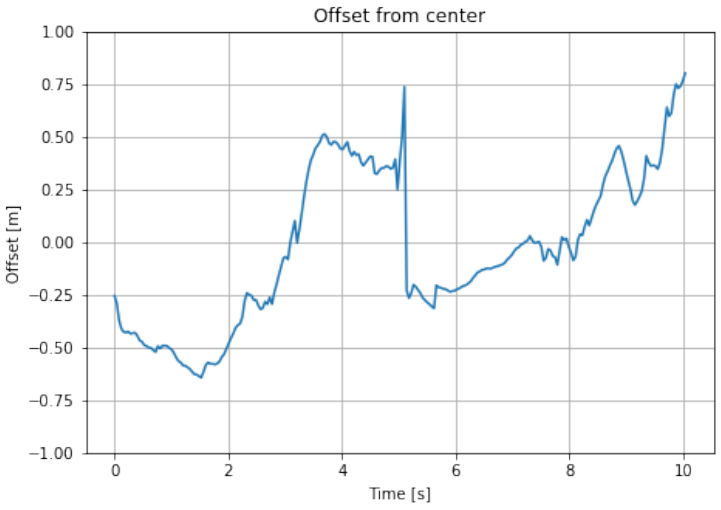
35%|███▌ | 87/251 [00:43<00:53, 3.05it/s]
35%|███▌ | 88/251 [00:43<00:53, 3.02it/s]
35%|███▌ | 89/251 [00:43<00:49, 3.26it/s]
36%|███▌ | 90/251 [00:44<00:49, 3.27it/s]
36%|███▌ | 91/251 [00:44<00:48, 3.33it/s]
37%|███▋ | 92/251 [00:44<00:49, 3.24it/s]
37%|███▋ | 93/251 [00:45<00:50, 3.15it/s]
37%|███▋ | 94/251 [00:45<00:51, 3.07it/s]
38%|███▊ | 95/251 [00:45<00:48, 3.24it/s]
38%|███▊ | 96/251 [00:46<00:48, 3.21it/s]
39%|███▊ | 97/251 [00:46<00:45, 3.39it/s]
39%|███▉ | 98/251 [00:46<00:47, 3.25it/s]
39%|███▉ | 99/251 [00:46<00:46, 3.26it/s]
40%|███▉ | 100/251 [00:47<00:46, 3.23it/s]
40%|████ | 101/251 [00:47<00:47, 3.17it/s]
41%|████ | 102/251 [00:47<00:48, 3.05it/s]
41%|████ | 103/251 [00:48<00:46, 3.16it/s]
41%|████ | 104/251 [00:48<00:47, 3.10it/s]
42%|████▏ | 105/251 [00:48<00:44, 3.26it/s]
42%|████▏ | 106/251 [00:49<00:42, 3.45it/s]
43%|████▎ | 107/251 [00:49<00:40, 3.54it/s]
43%|████▎ | 108/251 [00:49<00:37, 3.77it/s]
43%|████▎ | 109/251 [00:49<00:35, 3.95it/s]
44%|████▍ | 110/251 [00:50<00:34, 4.03it/s]
44%|████▍ | 111/251 [00:50<00:35, 3.94it/s]
45%|████▍ | 112/251 [00:50<00:37, 3.75it/s]
45%|████▌ | 113/251 [00:50<00:38, 3.61it/s]
45%|████▌ | 114/251 [00:51<00:37, 3.64it/s]
46%|████▌ | 115/251 [00:51<00:40, 3.32it/s]
46%|████▋ | 116/251 [00:51<00:46, 2.92it/s]
47%|████▋ | 117/251 [00:52<00:47, 2.82it/s]
47%|████▋ | 118/251 [00:52<00:49, 2.68it/s]
47%|████▋ | 119/251 [00:53<00:49, 2.68it/s]
48%|████▊ | 120/251 [00:53<00:50, 2.58it/s]
48%|████▊ | 121/251 [00:53<00:48, 2.67it/s]
49%|████▊ | 122/251 [00:54<00:48, 2.67it/s]
49%|████▉ | 123/251 [00:54<00:47, 2.69it/s]
49%|████▉ | 124/251 [00:54<00:43, 2.93it/s]
50%|████▉ | 125/251 [00:55<00:40, 3.12it/s]
50%|█████ | 126/251 [00:55<00:36, 3.43it/s]
51%|█████ | 127/251 [00:55<00:35, 3.54it/s]
51%|█████ | 128/251 [00:55<00:30, 4.04it/s]
51%|█████▏ | 129/251 [00:56<00:29, 4.20it/s]
52%|█████▏ | 130/251 [00:56<00:28, 4.23it/s]
52%|█████▏ | 131/251 [00:56<00:27, 4.36it/s]
53%|█████▎ | 132/251 [00:56<00:26, 4.41it/s]
53%|█████▎ | 133/251 [00:56<00:26, 4.48it/s]
53%|█████▎ | 134/251 [00:57<00:25, 4.67it/s]
54%|█████▍ | 135/251 [00:57<00:25, 4.56it/s]
54%|█████▍ | 136/251 [00:57<00:24, 4.76it/s]
55%|█████▍ | 137/251 [00:57<00:23, 4.80it/s]
55%|█████▌ | 138/251 [00:57<00:23, 4.82it/s]
55%|█████▌ | 139/251 [00:58<00:23, 4.86it/s]
56%|█████▌ | 140/251 [00:58<00:22, 4.98it/s]
56%|█████▌ | 141/251 [00:58<00:22, 4.84it/s]
57%|█████▋ | 142/251 [00:58<00:21, 4.99it/s]
57%|█████▋ | 143/251 [00:58<00:21, 5.03it/s]

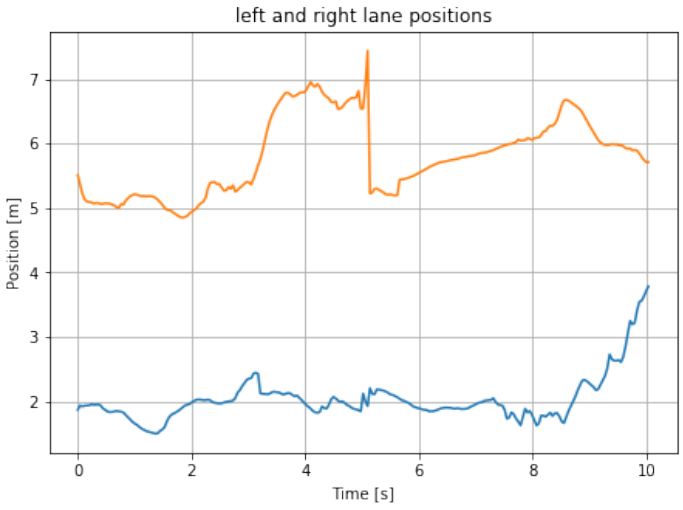
```
 57%|| 144/251 [00:59<00:21,  4.88it/s]
 58%|| 145/251 [00:59<00:21,  5.01it/s]
 58%|| 146/251 [00:59<00:20,  5.08it/s]
 59%|| 147/251 [00:59<00:21,  4.95it/s]
 59%|| 148/251 [00:59<00:20,  4.98it/s]
 59%|| 149/251 [01:00<00:20,  5.03it/s]
 60%|| 150/251 [01:00<00:20,  4.87it/s]
 60%|| 151/251 [01:00<00:20,  4.93it/s]
 61%|| 152/251 [01:00<00:19,  5.05it/s]
 61%|| 153/251 [01:00<00:19,  5.00it/s]
 61%|| 154/251 [01:01<00:19,  5.06it/s]
 62%|| 155/251 [01:01<00:18,  5.11it/s]
 62%|| 156/251 [01:01<00:18,  5.06it/s]
 63%|| 157/251 [01:01<00:18,  5.07it/s]
 63%|| 158/251 [01:01<00:17,  5.20it/s]
 63%|| 159/251 [01:02<00:17,  5.15it/s]
 64%|| 160/251 [01:02<00:17,  5.14it/s]
 64%|| 161/251 [01:02<00:17,  5.26it/s]
 65%|| 162/251 [01:02<00:18,  4.74it/s]
 65%|| 163/251 [01:02<00:18,  4.70it/s]
 65%|| 164/251 [01:03<00:17,  4.97it/s]
 66%|| 165/251 [01:03<00:16,  5.17it/s]
 66%|| 166/251 [01:03<00:16,  5.28it/s]
 67%|| 167/251 [01:03<00:16,  5.09it/s]
 67%|| 168/251 [01:03<00:16,  5.06it/s]
 67%|| 169/251 [01:04<00:16,  5.00it/s]
 68%|| 170/251 [01:04<00:15,  5.19it/s]
 68%|| 171/251 [01:04<00:16,  4.97it/s]
 69%|| 172/251 [01:04<00:16,  4.89it/s]
 69%|| 173/251 [01:04<00:16,  4.79it/s]
 69%|| 174/251 [01:05<00:16,  4.79it/s]
 70%|| 175/251 [01:05<00:15,  4.82it/s]
 70%|| 176/251 [01:05<00:14,  5.22it/s]
 71%|| 177/251 [01:05<00:12,  5.87it/s]
 71%|| 178/251 [01:05<00:11,  6.44it/s]
 71%|| 179/251 [01:05<00:10,  6.92it/s]
 72%|| 180/251 [01:05<00:09,  7.32it/s]
 72%|| 181/251 [01:06<00:09,  7.43it/s]
 73%|| 182/251 [01:06<00:09,  7.50it/s]
 73%|| 183/251 [01:06<00:09,  7.32it/s]
 73%|| 184/251 [01:06<00:10,  6.62it/s]
 74%|| 185/251 [01:06<00:10,  6.59it/s]
 74%|| 186/251 [01:06<00:09,  6.81it/s]
 75%|| 187/251 [01:07<00:09,  6.83it/s]
 75%|| 188/251 [01:07<00:09,  6.75it/s]
 75%|| 189/251 [01:07<00:09,  6.54it/s]
 76%|| 190/251 [01:07<00:09,  6.14it/s]
 76%|| 191/251 [01:07<00:10,  5.56it/s]
 76%|| 192/251 [01:07<00:11,  5.17it/s]
 77%|| 193/251 [01:08<00:12,  4.76it/s]
 77%|| 194/251 [01:08<00:12,  4.59it/s]
 78%|| 195/251 [01:08<00:12,  4.42it/s]
 78%|| 196/251 [01:08<00:13,  4.05it/s]
 78%|| 197/251 [01:09<00:14,  3.73it/s]
 79%|| 198/251 [01:09<00:16,  3.14it/s]
 79%|| 199/251 [01:10<00:18,  2.85it/s]
 80%|| 200/251 [01:10<00:18,  2.81it/s]
```

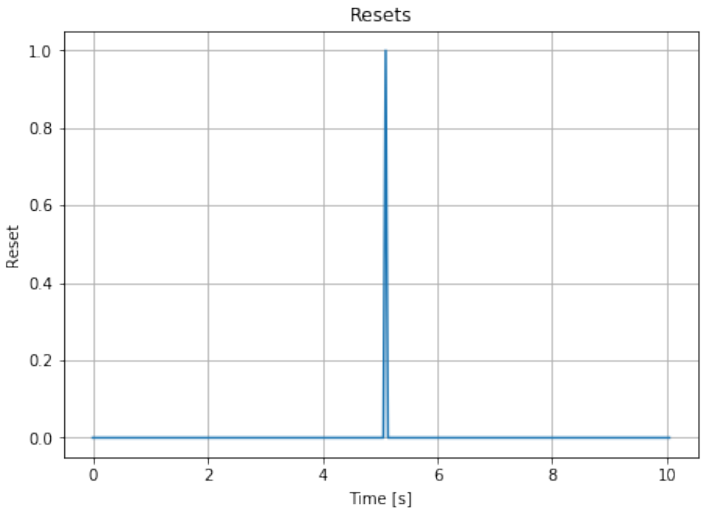
```
 80%|████████▊   | 201/251 [01:10<00:18,  2.68it/s]
 80%|████████▊   | 202/251 [01:11<00:20,  2.42it/s]
 81%|████████▊   | 203/251 [01:11<00:19,  2.51it/s]
 81%|████████▉   | 204/251 [01:12<00:19,  2.41it/s]
 82%|████████▉   | 205/251 [01:12<00:20,  2.23it/s]
 82%|████████▉   | 206/251 [01:13<00:19,  2.26it/s]
 82%|████████▉   | 207/251 [01:13<00:22,  1.94it/s]
 83%|████████▉   | 208/251 [01:14<00:25,  1.71it/s]
 83%|█████████   | 209/251 [01:15<00:25,  1.66it/s]
 84%|█████████   | 210/251 [01:16<00:26,  1.52it/s]
 84%|█████████   | 211/251 [01:16<00:28,  1.40it/s]
 84%|█████████   | 212/251 [01:17<00:28,  1.37it/s]
 85%|█████████   | 213/251 [01:18<00:30,  1.26it/s]
 85%|█████████   | 214/251 [01:19<00:30,  1.23it/s]
 86%|█████████   | 215/251 [01:20<00:29,  1.23it/s]
 86%|█████████▏  | 216/251 [01:21<00:27,  1.25it/s]
 86%|█████████▏  | 217/251 [01:21<00:26,  1.26it/s]
 87%|█████████▏  | 218/251 [01:22<00:26,  1.27it/s]
 87%|█████████▏  | 219/251 [01:23<00:24,  1.29it/s]
 88%|█████████▏  | 220/251 [01:24<00:23,  1.33it/s]
 88%|█████████▏  | 221/251 [01:24<00:22,  1.31it/s]
 88%|█████████▎  | 222/251 [01:25<00:21,  1.33it/s]
 89%|█████████▎  | 223/251 [01:26<00:21,  1.30it/s]
 89%|█████████▎  | 224/251 [01:27<00:20,  1.31it/s]
 90%|█████████▎  | 225/251 [01:27<00:20,  1.29it/s]
 90%|█████████▍  | 226/251 [01:28<00:19,  1.30it/s]
 90%|█████████▍  | 227/251 [01:29<00:18,  1.32it/s]
 91%|█████████▍  | 228/251 [01:30<00:16,  1.39it/s]
 91%|█████████▍  | 229/251 [01:30<00:15,  1.43it/s]
 92%|█████████▍  | 230/251 [01:31<00:14,  1.45it/s]
 92%|█████████▌  | 231/251 [01:32<00:13,  1.48it/s]
 92%|█████████▌  | 232/251 [01:32<00:11,  1.60it/s]
 93%|█████████▌  | 233/251 [01:33<00:10,  1.64it/s]
 93%|█████████▌  | 234/251 [01:33<00:10,  1.68it/s]
 94%|█████████▌  | 235/251 [01:34<00:09,  1.72it/s]
 94%|█████████▋  | 236/251 [01:34<00:08,  1.76it/s]
 94%|█████████▋  | 237/251 [01:35<00:07,  1.79it/s]
 95%|█████████▋  | 238/251 [01:35<00:07,  1.83it/s]
 95%|█████████▋  | 239/251 [01:36<00:06,  1.83it/s]
 96%|█████████▋  | 240/251 [01:36<00:05,  1.86it/s]
 96%|█████████▊  | 241/251 [01:37<00:05,  1.83it/s]
 96%|█████████▊  | 242/251 [01:37<00:04,  1.89it/s]
 97%|█████████▊  | 243/251 [01:38<00:04,  1.79it/s]
 97%|█████████▊  | 244/251 [01:39<00:04,  1.66it/s]
 98%|█████████▊  | 245/251 [01:39<00:03,  1.62it/s]
 98%|█████████▊  | 246/251 [01:40<00:03,  1.48it/s]
 98%|█████████▉  | 247/251 [01:41<00:02,  1.41it/s]
 99%|█████████▉  | 248/251 [01:42<00:02,  1.36it/s]
 99%|█████████▉  | 249/251 [01:43<00:01,  1.24it/s]
100%|█████████▉  | 250/251 [01:43<00:00,  1.28it/s]

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

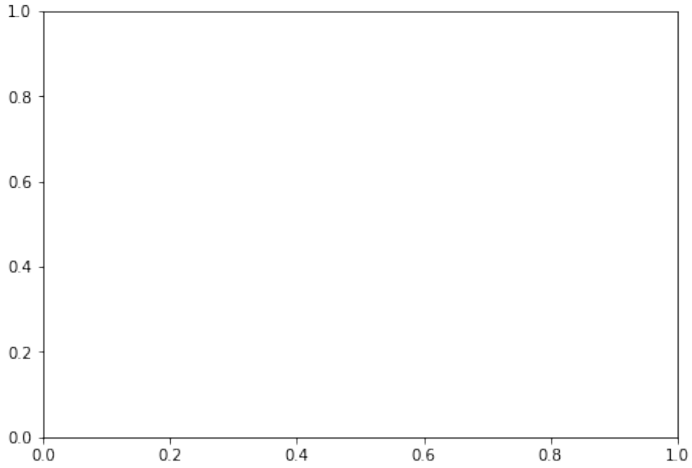

Radius of Curvature
Curvature [m]
Time [s]


Radius of Curvature (inv)
Curvature [1/m]
Time [s]


Offset from center
Offset [m]
Time [s]


left and right lane positions
Position [m]
Time [s]


Resets
Reset
Time [s]



In [ ]: