Ψηφιακή Επεξεργασία & Ανάλυση Εικόνας

Εργαστηριακές Ασκήσεις - Μέρος Β'

Όνομα:Μάριος Νεκτάριος Σταματόπουλος ΑΜ:1059383

up1059383@upnet.gr

Θέμα 4

In [2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage
import math
from itertools import product
from random import shuffle,choice
```

In [3]:
```
def preProcessImg(image):
    img= np.array(image)
    img_color=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_gray= cv2.cvtColor(img_color, cv2.COLOR_RGB2GRAY)
    return img_color,img_gray
```

In [4]:
```
def getMaskHighestSobel(img):
    MAX_THRESH=300
    der=ndimage.sobel(img, 0)  # horizontal derivative

    max_sum=0
    sums=[]
    for row in der:
        athr=sum(row)
        sums.append(athr)

    row_thres=sums.index(max(sums))
    if(row_thres>MAX_THRESH):
        row_thres=MAX_THRESH

    #mask creation
    rows,cols=img.shape
    rows
    mask=[]
    mask[0:row_thres]=np.zeros( (row_thres,cols) )
    mask[row_thres:]=np.ones( (rows-row_thres,cols) )

    return mask
```

In [5]:
```
def limit(n, minn, maxn):
    return max(min(maxn, n), minn)
```

In [6]:
```
def apply_mask(img_color,mask):
    img_color_ROI=img_color.copy()
    for i in range(3):
        img_color_ROI[:,:,i]=img_color[:,:,i]*mask
    return img_color_ROI
```

In [7]:
```
def smoothing(arr):
    #smoothing sums
    win_size=8
    a=win_size//2
    for i in range(len(arr)):
        start=limit(i-a,0,len(arr)-1)
        end=limit(i+a,0,len(arr)-1)
        win=arr[start:end]
        arr[i]=sum(win)/win_size
    return arr
```

In [8]:
```
def write_video(frames,rows,cols,name="out"):
    fourcc = cv2.VideoWriter_fourcc('M','J','P','G')
    out = cv2.VideoWriter(name+'.avi', fourcc, 30, (cols,rows))

    # out = cv2.VideoWriter('mlkia.avi',cv2.VideoWriter_fourcc(*'DIVX'), 15, size)
    for i,f in enumerate(frames):
        f=cv2.cvtColor(f, cv2.COLOR_RGB2BGR)
```

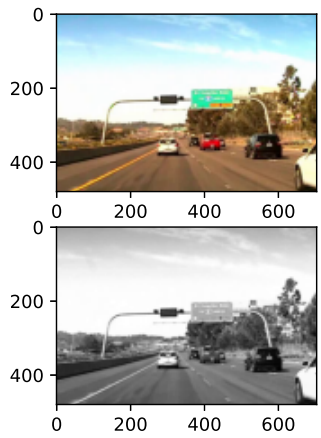
```
        cv2.putText(f,str(i), (300,50),  cv2.FONT_HERSHEY_SIMPLEX, 2, 255)
        out.write(f)

    out.release()
```

1η Μέθοδο

Η διαδικασια που χρησιμοποιηθηκε ειναι η εξης: Πρώτα η εικόνα μετατρέπετα σε ασπρόμαυρη

In [9]:
```
vidcap = cv2.VideoCapture('april21.avi')
success,image = vidcap.read()
count = 0

img_color,img=preProcessImg(image)

filter_ker=[[-1,0,1],
        [-1,0,1],
        [-1,0,1]]

plt.figure()
plt.subplot(211)
plt.imshow(img_color)

plt.subplot(212)
plt.imshow(img,cmap='gray')
# convolve2D(image,filter_ker )

print(img.shape)
```

```
(480, 704)
```

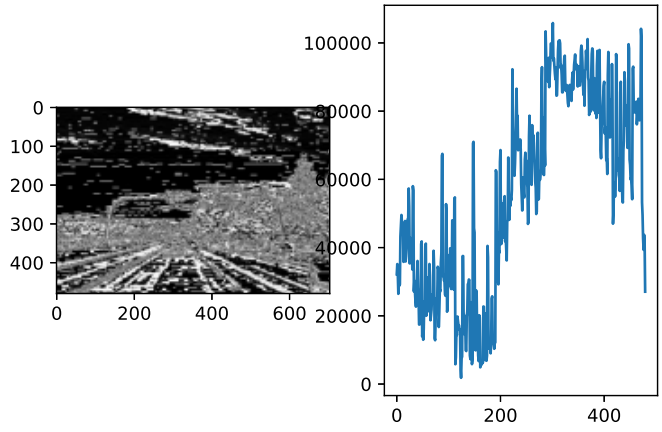


Έπειτα υπολογίζεται το άθροισμα των τιμών της κάθε γραμμής και αυτη που έχει το μεγαλύτερο άθροισμα επιλέγεται ως το σημειο που απο εκει και κάτω θα ορίζεται η περιοχή ενδιαφέροντος

In [10]:
```
der=ndimage.sobel(img, 0)  # horizontal derivative

max_sum=0
sums=[]
for row in der:
    athr=sum(row)
    sums.append(athr)

row_thres=sums.index(max(sums))
print("row threshold:",row_thres)

plt.figure()
plt.subplot(121)
plt.imshow(der,cmap='gray',vmin=0,vmax=255)
plt.subplot(122)
plt.plot(sums)
```

```
row threshold: 301
```

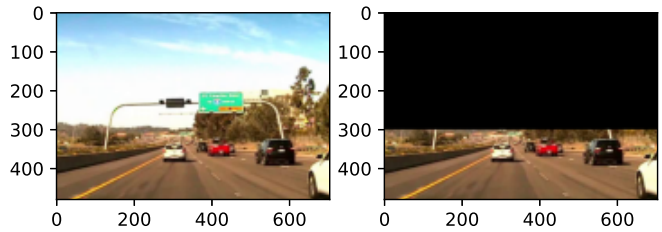
Out[10]:
```
[<matplotlib.lines.Line2D at 0x279701159a0>]
```



In [11]:
```
plt.figure()
plt.subplot(121)
plt.imshow(img_color)

mask=getMaskHighestSobel(img)
img_color_ROI=apply_mask(img_color,mask)
print(img_color_ROI.shape)

plt.subplot(122)
plt.imshow(img_color_ROI)
```

```
(480, 704, 3)
```

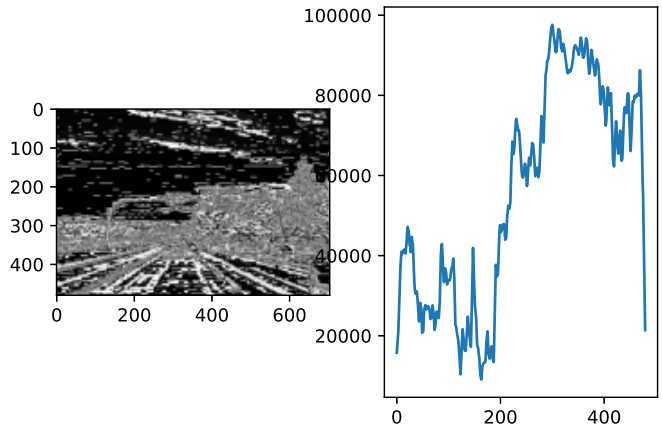
Out[11]: <matplotlib.image.AxesImage at 0x27970151070>



Η δευτερη μέθοδος που υλοποιήθηκε ειναι μια βελτίωση της προηγούμενης.Συγκεκριμένα αποτέλεσμα φιλτράρεται ώστε να φυγει ο θόρυβος παίρνοντας το αποτέλεσμα που φαίνεται παρακάτω

In [12]:
```
sums=smoothing(sums)
plt.figure()
plt.subplot(121)
plt.imshow(der,cmap='gray',vmin=0,vmax=255)
plt.subplot(122)
plt.plot(sums)
```

Out[12]: [<matplotlib.lines.Line2D at 0x2797140ee50>]



2η Μέθοδος

Επίσης μεταξύ των frames είναι λογικό η περιοχή ενδιαφέροντος να μην αλλάζει πολύ.Συνεπώς γίνεται μια τοπική αναζήτηση κάθε φορά παίροντας ως αναφορά την αχή της περιοχής ενδιαφέροντος του προηγούμενου frame.Ωστόσο έχει ορίστει ενα REFRESH_INTERVAL το οποίο δηλώνει ανα πόσα frames θα γίνεται εκ νέου αναζήτηση σε όλη την εικόνα,ώστε να μην υπάρχουν θέματα με τοπικά μέγιστα.Η υλοποίηση της 2ης μεθόδου φαίνεται παρακάτω:

In [20]:
```
def getMaskHighestSobel2(img,print_details):
```

```
    MAX_THRESH=350
    MAX_DX=50
    REFRESH_INTERVAL=10 #number of frames that algo refreshes local search
    der=ndimage.sobel(img, 0)  # horizontal derivative
    #limiting to MAX_THRESH
    der=der[:MAX_THRESH]

    #making variable names smaller
    prev_threshold=getMaskHighestSobel2.prev_threshold
    counter=getMaskHighestSobel2.counter

    local_search_reset=counter>REFRESH_INTERVAL or  prev_threshold==0# local search reset condition

    # local_search_reset=True

    #get row_thres

    if (not local_search_reset):
        arr_to_search=der[limit(prev_threshold-MAX_DX,0,800):prev_threshold+MAX_DX]
        start_ind=limit(prev_threshold-MAX_DX,0,800)
    else:
        arr_to_search=der
        start_ind=0

    sums=[]
    for row in arr_to_search:
        athr=sum(row)
        sums.append(athr)

    sums=smoothing(sums)

    row_thres=start_ind+sums.index(max(sums))

    #if not first time executed  or local search is not to be reset
    if (not prev_threshold==0 and (not local_search_reset) ):
        row_thres=limit(row_thres,prev_threshold-MAX_DX,prev_threshold+MAX_DX)



    getMaskHighestSobel2.prev_threshold=row_thres

    if(print_details):

        plt.subplot(221)
        plt.plot(sums)
        print("start_ind:",start_ind)
        print("row_thres:",row_thres)

        sums=[]
        for row in der:
            athr=sum(row)
            sums.append(athr)

        sums=smoothing(sums)
        plt.subplot(223)
        plt.plot(sums)

        plt.subplot(222)
        plt.imshow(img)

        plt.subplot(224)
        plt.imshow(der,cmap='gray',vmin=0,vmax=255)

        row_thres=sums.index(max(sums))
        print("row_thres whole window:",row_thres)
        plt.show()
        input()

    # resetting local search
    if (local_search_reset):
        getMaskHighestSobel2.counter=0
    else:
        getMaskHighestSobel2.counter=counter+1
    # print(row_thres)
    #mask creation
    rows,cols=img.shape
    rows
    mask=[]
    mask[0:row_thres]=np.zeros( (row_thres,cols) )
    mask[row_thres:]=np.ones( (rows-row_thres,cols) )

    return mask

getMaskHighestSobel2.prev_threshold=0
getMaskHighestSobel2.counter=0
```

Προσθήκη και φιλτράρισμα θορύβου

Οι παρακατω συναρτήσεις χρησιμοποιηθηκαν για την δημιουργία και την εξάλειψη του θορύβου

In [14]:
```
def addGaussianNoise(img):
    x = np.copy(img) + np.random.normal(scale=np.mean(img)/(10**1.5), size=img.shape)
    return x
```

In [15]:
```
def addSaltPepperNoise(img):
    pct=0.6
    x = np.copy(img)
    coords = list(  product(  range(img.shape[0]), range(img.shape[1]) )   )
    shuffle(coords)
    for coord in coords[:int(pct*img.shape[0]*img.shape[1])]:
        x[coord] = 0 if choice([True, False]) else 255

    return x
```

In [16]:
```
def moving_avg_filter(img, size=3):
    if size % 2 != 1: return

    pad = size//2

    res = np.zeros(img.shape)
    img = np.pad(img, pad, constant_values=128)

    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i, j] = np.mean(img[i:i+2*pad, j:j+2*pad])

    return res
```

In [17]:
```
def median_filter(img, size=3):
    if size % 2 != 1: return

    pad = size//2

    res = np.zeros(img.shape)
    img = np.pad(img, pad, constant_values=128)

    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i, j] = np.median(img[i:i+2*pad, j:j+2*pad])

    return res
```

In [21]:
```
ADD_NOISE=0

frames=[]
plt.figure()
vidcap = cv2.VideoCapture('april21.avi')
success=True
count=0
while vidcap.isOpened():

    success,image = vidcap.read()
    if(not success):
        break

    img_color,img_gray=preProcessImg(image)
    if (ADD_NOISE):
        img_gray=addSaltPepperNoise(img_gray)
        img_gray=addGaussianNoise(img_gray)
        img_gray=median_filter(img_gray)
        img_gray=moving_avg_filter(img_gray)
    # print_details=True if (count==100 or count in range(140,150)) else False
    print_details=False
    mask=getMaskHighestSobel2(img_gray,print_details)
    # mask=getMaskHighestSobel(img_gray)
    masked_img=apply_mask(img_color,mask)
    frames.append(masked_img)

    count += 1
    # print("count:",count)
    # if(count%30==0):
    #     print(count/30)


#write video
r,c=img_gray.shape
write_video(frames,r,c,name="Method 2 without noise")
```

```
<Figure size 432x288 with 0 Axes>
```

In [357…
```
write_video(frames,r,c,name="Method 2 with noise")
```

Σύγκριση αποτελεσμάτων μετά την εισαγωγή θορύβου

Μέθοδος 1

Παρατηρείται ότι ο θόρυβος δεν εξαλείφεται πλήρως και έχει ως αποτέλεσμα να μην επιλέγεται η σωστή περιοχή ενδιαφέροντος αλλά μια αρκετά μεγαλύτερη καθ'όλη την διάρκεια του video.Επίσης παρατηρούνται απότομες εναλλαγες μεταξυ των καρέ λόγω της απουσίας tracking μεταξύ τους.

Μέθοδος 2

Παρατηρείται ότι ο θόρυβος δεν εξαλείφεται πλήρως και έχει ως αποτέλεσμα να μην επιλέγεται η σωστή περιοχή ενδιαφέροντος αλλά μια αρκετά μεγαλύτερη για το πρώτο μισό του video.Αυτό έχει ως αποτέλεσμα την αύξηση του υπολογιστικού κόστους της μετέπειτα επεξεργασίας.Έπειτα εξομαλύνεται αλλά συνεχίζουν να υπάρχουν απότομες εναλλάγες

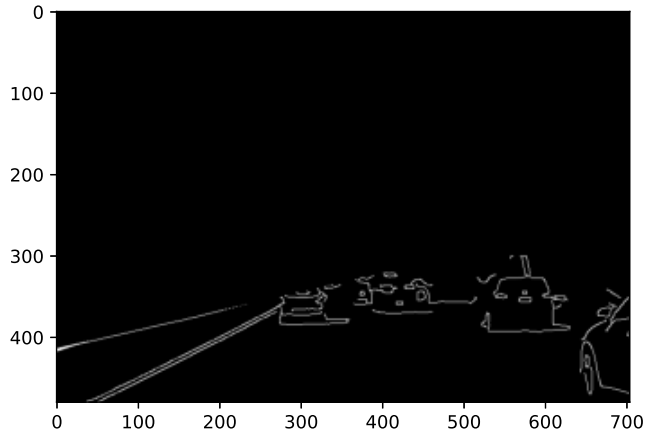
Μάσκα κατάλληλη για την ανίχνευση του οχήματος.

Για να βρεεθεί η τελική μάσκα ανίχνευσης των αυτοκινήτων χρημοποιείτια η συνάρτηση findLines στην οποία η κάθε εικόνα περνιέται απο φίλτρο Gaussin Blur,Canny edge detection ώστε το αποτέλεσμα αυτών των 2 να δωθεί στον Μετασχηματισμό Hough.Επίσης υπολογίζονται οι ακραίες ευθείες που αντιστοιχούν στα άκρα του σρόμου και φτιάχνεται μια μάσκα που περικλείειε όσα βρίσκονται μεταξύ τους και αππορίπτει τα εξωτερικά όπως φαίνεται στην παρακάτω εικόνα:

In [343…

```
line_mask,line_image=find_lanes(frames[0])
plt.figure()
plt.imshow(line_mask*line_image,cmap="gray")

img_gray=line_mask*line_image
```

```
<ipython-input-343-618a5226e23c>:1: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(frames[0])
<ipython-input-343-618a5226e23c>:1: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(frames[0])
Coords:
0 413
343 300
```



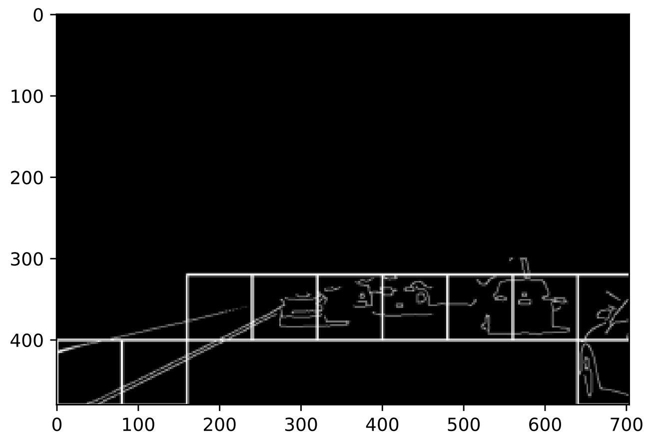
Έπειτα το αποτέλεσμα δίνετια στην συνάρτηση find_vehicles στην οποία μέσω ενος κυλιόμενου παραθύρου υπολογίζεται η πυκνότητα κατα σημεία ώστε να ανιχνευθούν τα αυτοκίνητα.Στο τέλος ο αριθμός των πραθύρων που επιστρέφονται είνα ιδυναμικος ανάλογα με το γράφημα που φαίνεται παρακάτω.Ειδικότερα επιλέγεται εκείνο το ελάχιστο της 1ης παραγώγου της πυκνότητας των block το οποιο αντιστοιχεί στον μεγαλύτερο αριθμό.Στο συγκεκριμένο γράφημα αυτό ειναι το 10.

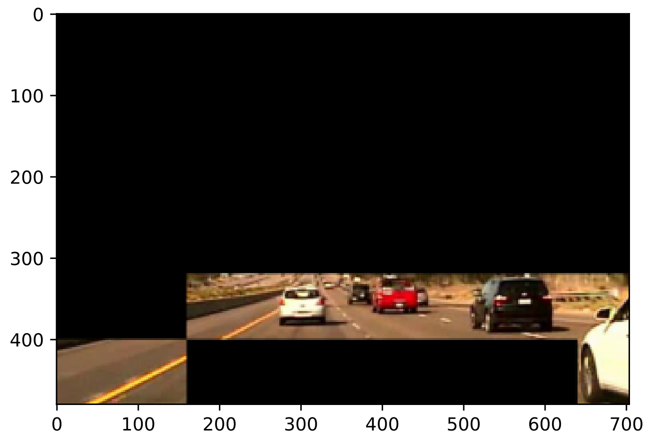
In [346…

```
rects,out=find_vehicles(img_gray,threshold=2000.0,numberOfRects=10,win_size=80)
plt.figure()
plt.imshow(rects,cmap='gray')
plt.figure()
plt.imshow(apply_mask(frames[0],out))
```

```
320 320
320 560
320 400
320 240
400 640
320 640
320 480
400 80
400 0
320 160
```

Out[346… `<matplotlib.image.AxesImage at 0x17d26172df0>`





In [351…

```
frames_final=[]
for f in frames:
    try:
        line_mask,line_image=find_lanes(f)
        img_gray=line_mask*line_image
        # plt.imshow(img_gray)
        # # apply binary thresholding
        # ret, thresh = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY)
        # # detect the contours on the binary image using cv2.CHAIN_APPROX_NONE
        # contours, hierarchy = cv2.findContours(image=thresh.astype(np.uint8), mode=cv2.RETR_TREE, method=cv2.CHAIN_APPROX_NONE)
        # # draw contours on the original image
        # image_copy = f.copy()
        # cv2.drawContours(image=image_copy, contours=contours, contourIdx=-1, color=(0, 255, 0), thickness=2, lineType=cv2.LINE_AA)
        # frames_final.append(image_copy)

        xxx,out=find_vehicles(img_gray,threshold=2000.0,numberOfRects=10,win_size=80)

        frames_final.append(apply_mask(f,out))

    except Exception as e:
        print("ERROR : "+str(e))

write_video(frames_final,r,c,name="final mask")
```

```
d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
```

```
<ipython-input-351-898ef1662d9a>:4: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(f)
```

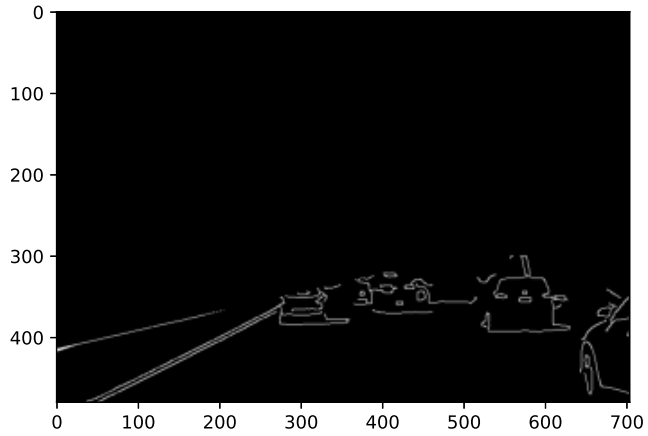
In [349…

```
def find_lanes(img):
    """
    INPUT  :grayscale image
    OUTPUT :mask with lines
    """
    for i in range( len(img) ):
        if ( not img[i][0][0]==0):
            row_thres=i
            break

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    kernel_size = 9
    blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)
    low_threshold = 250/3
    high_threshold = 250
    edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
    rho = 1  # distance resolution in pixels of the Hough grid

    theta = np.pi / 180  # angular resolution in radians of the Hough grid
    threshold = 15  # minimum number of votes (intersections in Hough grid cell)
    min_line_length = 100  # minimum number of pixels making up a line
    # min_line_length = 80  # minimum number of pixels making up a line
    max_line_gap = 20  # maximum gap in pixels between connectable line segments
    line_image = np.ones((gray.shape[0],gray.shape[1]) )  # creating a blank to draw lines on
    # plt.figure()
    # plt.imshow(edges)
    # Run Hough on edge detected image
    # Output "lines" is an array containing endpoints of detected line segments
    lines = cv2.HoughLinesP(edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)
    crucial_lines=[]
    for line in lines:
        for x1,y1,x2,y2 in line:
            x=[x1,x2]
            y=[y1,y2]
            # Calculate the coefficients. This line answers the initial question.
            coefficients = np.polyfit(x, y, 1)
            coefficientsInv = np.polyfit(y, x, 1)
            polynomial = np.poly1d(coefficients)
            polynomialInv = np.poly1d(coefficientsInv)

            dy=y2-y1
            dx=x2-x1
            angle=math.atan2(dy,dx)
            angle=math.degrees(angle)

            if (abs(angle)>12 and abs(angle)<70):
                crucial_lines.append([polynomial,int( polynomial(0) )])
                # cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),5)


    leftMost_line_poly=min(crucial_lines,key=lambda x:x[1])[0]
    cv2.line(edges,(0,int( leftMost_line_poly(0) )),(int( leftMost_line_poly(row_thres) ),row_thres),(255,0,0),5)
    # print("Coords:")
    # print(0,int( leftMost_line_poly(0) ) )
    # print( int( leftMost_line_poly(row_thres) ),row_thres )
    line_image[:row_thres]=np.zeros( (row_thres,len(img[0])) )

    for i in range(row_thres,len(img)):
        for j in range( len(img[0]) ):
            Point=(j,i)
            # print("CONDITION")
            # print(Point)
            # print(isPointAboveLIne(leftMost_line_poly,Point))
            if (isPointAboveLIne(leftMost_line_poly,Point) ):
                line_image[i,j]=0
            else:
                break
    #plt.imshow(line_image,cmap='gray',vmin=0,vmax=1)
    return edges,line_image

line_mask,line_image=find_lanes(frames[0])
plt.figure()
plt.imshow(line_mask*line_image,cmap="gray")
```

```
<ipython-input-349-4bbdf681551b>:72: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(frames[0])
<ipython-input-349-4bbdf681551b>:72: RankWarning: Polyfit may be poorly conditioned
  line_mask,line_image=find_lanes(frames[0])
```

Out[349…

```
<matplotlib.image.AxesImage at 0x17d1ebaf1f0>
```



In [350…

```
def find_vehicles(image,threshold,numberOfRects=5,win_size=150):
    img=image.copy()
    out=np.zeros(img.shape)
    arr=[]
    for i in range(0,image.shape[0],win_size):
        for j in range(0,image.shape[1],win_size):
            area=np.sum(image[i:i+win_size, j: j+win_size], )
            # print(area,i,j)
            if area>threshold:
                # represents the top left corner of rectangle
                start_point = (j, i)
                arr.append([area,start_point])
    arr=sorted(arr, key=lambda x: x[0],reverse=True)
    plot=[i[0] for i in arr]
    # plt.figure()
    # plt.subplot(311)
    # plt.plot(plot)
    # plt.subplot(312)
    # plt.plot(np.diff(plot) )
    # plt.subplot(313)
    # plt.plot(np.diff(np.diff(plot)) )

    a=np.diff(plot)
    a=np.r_[True, a[1:] < a[:-1]] & np.r_[a[:-1] < a[1:], True]#find local minimums
    a=np.delete(a, -1)
    ind=np.where(a==True)
    numberOfRects=max(ind[0])
    numberOfRects=max(5,numberOfRects)
    # print("numberOfRects:",numberOfRects)
    for i in arr[:numberOfRects]:
        # represents the top left corner of rectangle
        start_point = i[1]
        # Ending coordinate, here (220, 220)
        # represents the bottom right corner of rectangle
        end_point = (start_point[0]+win_size, start_point[1]+win_size)
        # Blue color in BGR
        color = (255, 0, 0)
        # Line thickness of 2 px
        thickness =2
        # Using cv2.rectangle() method
        # Draw a rectangle with blue line borders of thickness of 2 px
        img = cv2.rectangle(img, start_point, end_point, color, thickness)
        # print(start_point[1],start_point[0])
        try:
            for i in range(min(win_size, img.shape[0]-start_point[1] )):
                for j in range(min(win_size, img.shape[1]-start_point[0] )):
                    out[start_point[1]+i][start_point[0]+j]=1
        except Exception as e:
            print("ERROR : "+str(e))
    return img,out
```

In [347…

```
def isPointAboveLIne(line_poly,PointCoords):
    x,y=PointCoords
    x1,y1=0,line_poly(0)

    x2,y2=1,line_poly(1)

    return ((x1 - x2)*(y - y2) - (y1 - y2)*(x - x2)) > 0
```

In [ ]:

```
def addGausianNoise(img):
    x = np.copy(img) + np.random.normal(scale=np.mean(img)/(10**1.5), size=img.shape)
    return x
```

In [ ]:

```
def addSaltPepperNoise(img):
    pct=0.6
    x = np.copy(img)
```

```
    coords = list(  product(  range(img.shape[0]), range(img.shape[1]) )   )
    shuffle(coords)
    for coord in coords[:int(pct*img.shape[0]*img.shape[1])]:
        x[coord] = 0 if choice([True, False]) else 255

    return x
```

In [ ]:

```
def addSaltPepperNoise(img):
    pct=0.6
    x = np.copy(img)
    coords = list(  product(  range(img.shape[0]), range(img.shape[1]) )   )
    shuffle(coords)
    for coord in coords[:int(pct*img.shape[0]*img.shape[1])]:
        x[coord] = 0 if choice([True, False]) else 255

    return x

def moving_avg_filter(img, size=3):
    if size % 2 != 1: return

    pad = size//2

    res = np.zeros(img.shape)
    img = np.pad(img, pad, constant_values=128)

    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i, j] = np.mean(img[i:i+2*pad, j:j+2*pad])

    return res
def median_filter(img, size=3):
    if size % 2 != 1: return

    pad = size//2

    res = np.zeros(img.shape)
    img = np.pad(img, pad, constant_values=128)

    for i in range(res.shape[0]):
        for j in range(res.shape[1]):
            res[i, j] = np.median(img[i:i+2*pad, j:j+2*pad])

    return res
```