UJWAL D

INTERNSHIP REPORT

Detection of Solar Panels From Drone Footages

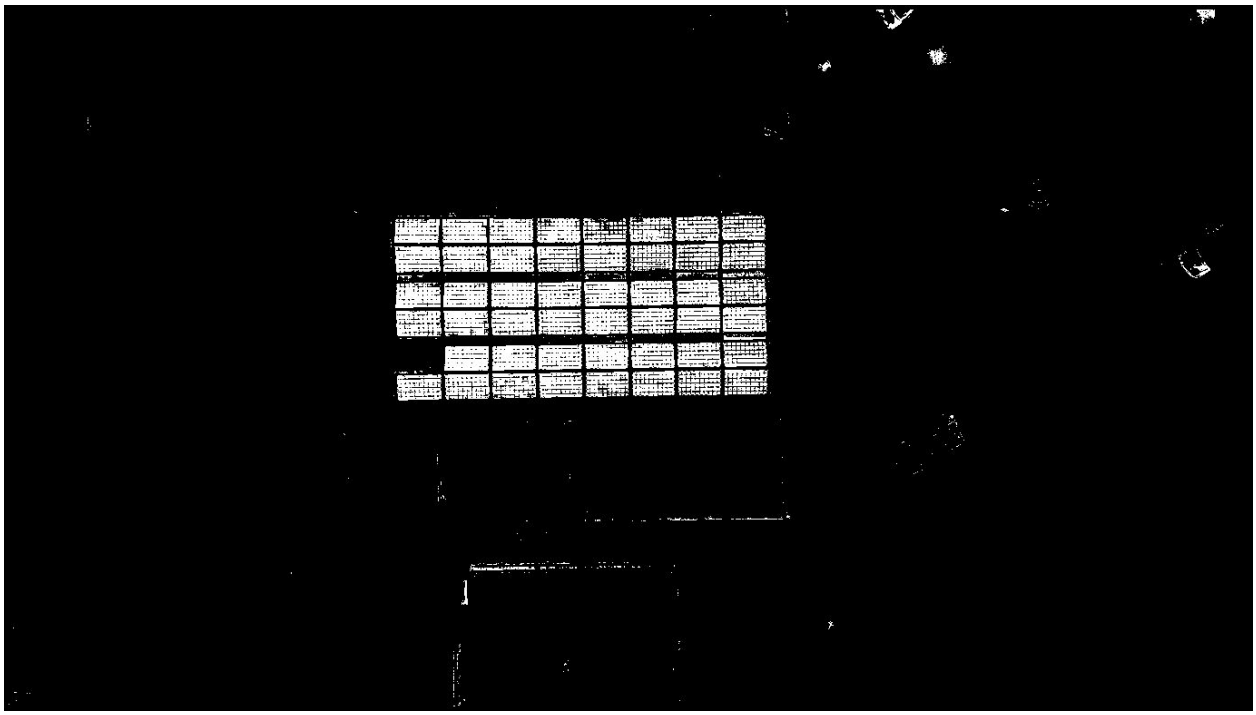
Introduction

This report explains a basic image processing approach to detect the location of the solar panels. A typical sample input image is as follows:

Processing Scheme:

First Stage

Go through each pixel and mark only the pixels with B value greater than 1.5 times the R value as Regions of interest.Below Image is the result of this operation.



Second Stage

To eliminate some of the noise from the previous stage run a window slider through the image(Not the entire image,just at locations which were classified as region of interests) and if the ratio of pixels of interest to total number of pixels in the window is above a certain threshold mark all the pixels inside the window as pixels of interest.

Here we've introduced two parameters that can be adjusted-window size and threshold

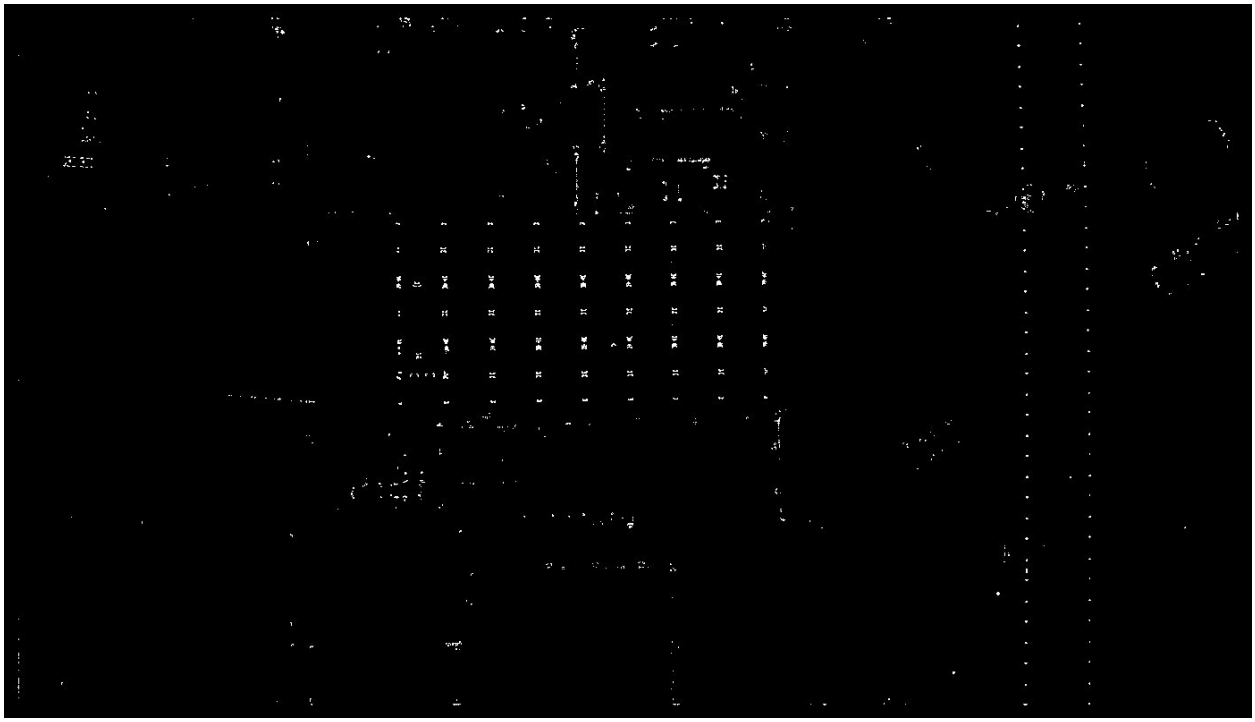
The bigger the window size,the better the noise elimination, but more computationally expensive the code is. So striking a balance, a window size of 15x15 pixels performed well

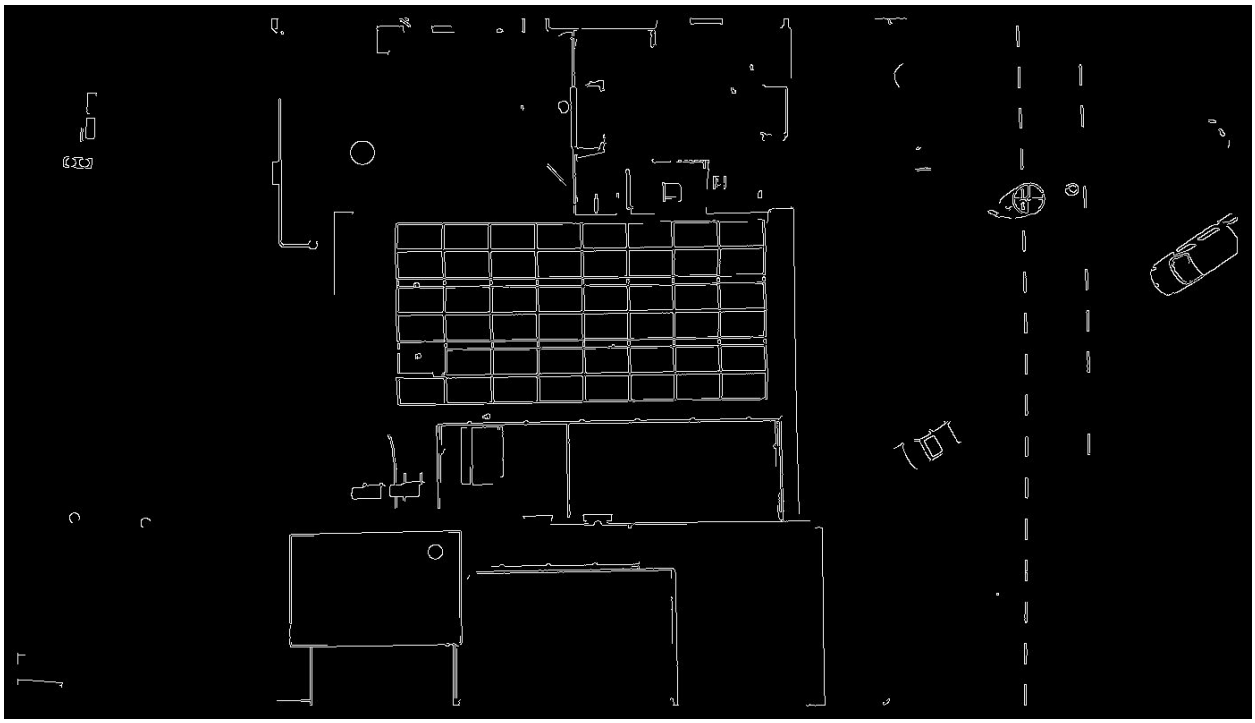
along with a threshold of 0.4

Now that a region of interest is formed, further processing can be done.

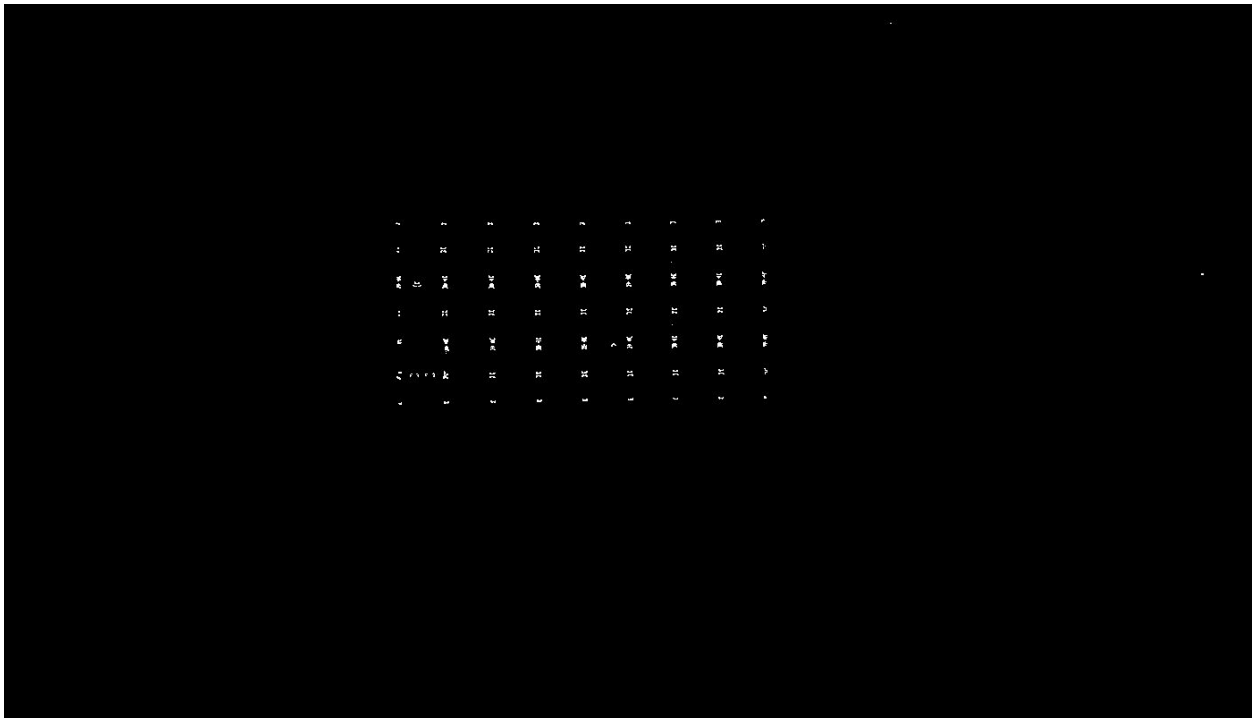
Third Stage

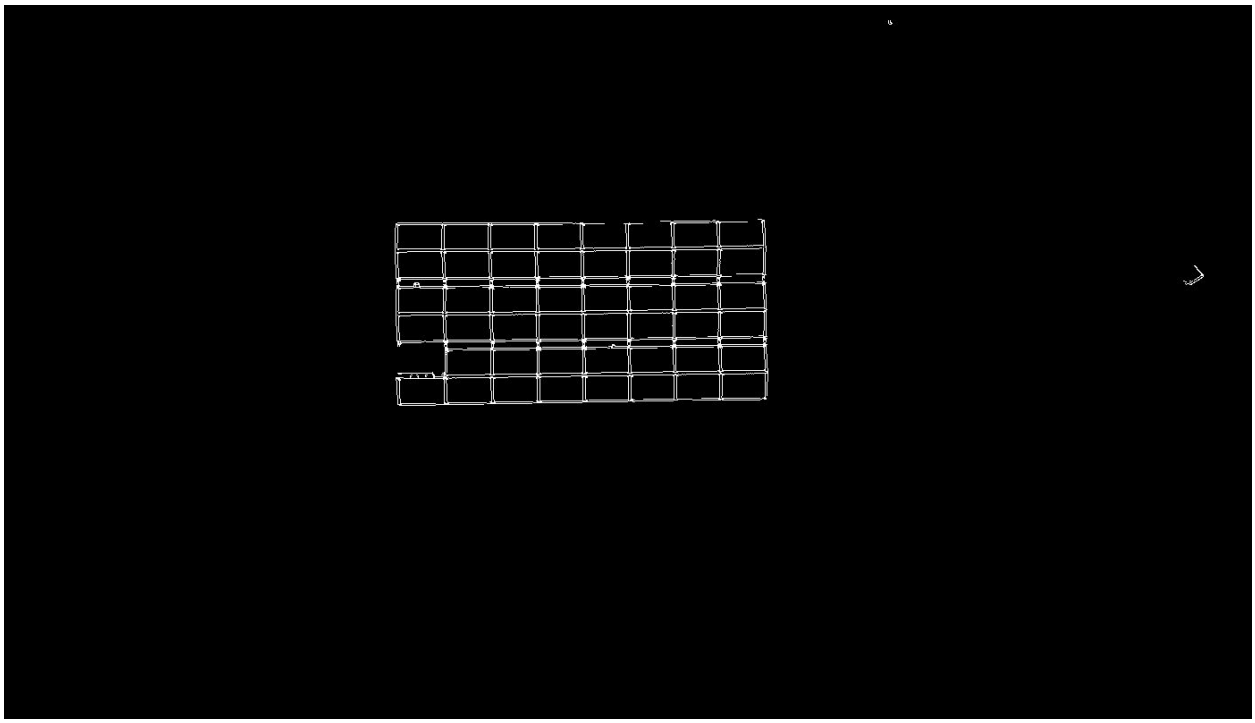
OpenCV has built in functions for corner(Corner Harris) and edge(Canny) detection. Below are the corner and edge detected versions of the input image.Using Canny edge detection introduces two parameters to be set that determines the threshold level for classifying a certain region as an edge.





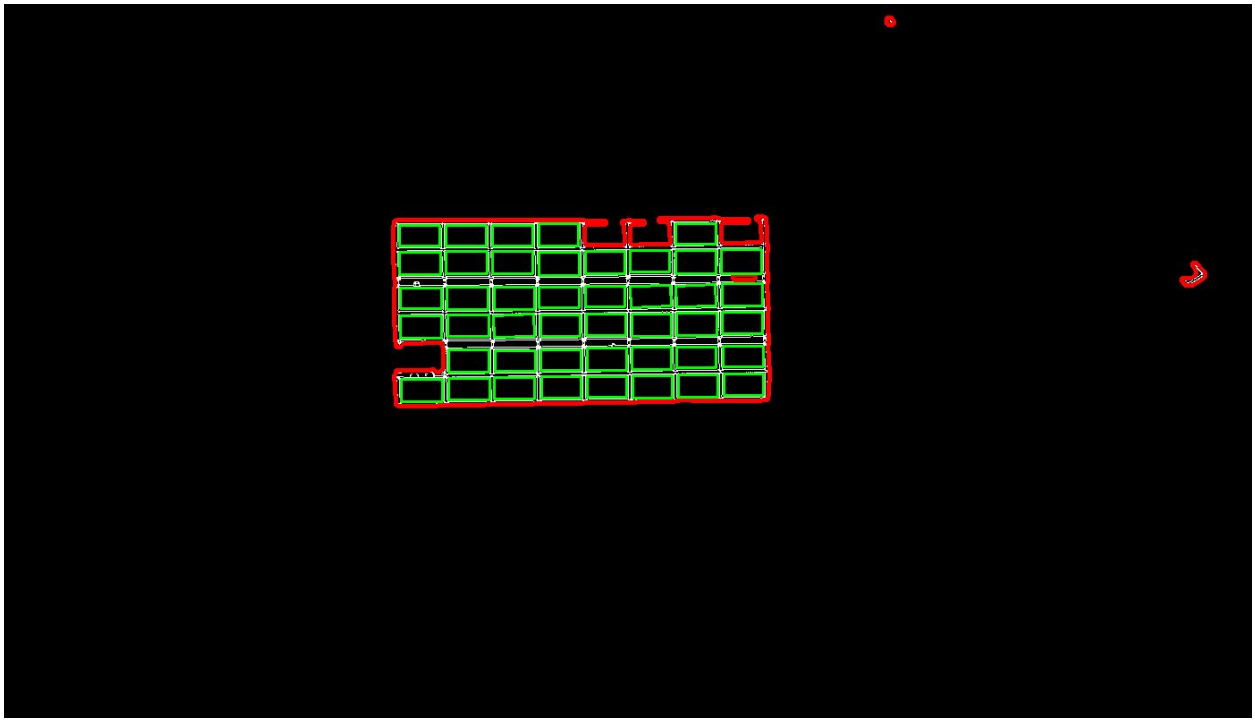
Now, ANDing these with region of interest we get the following result





Fourth Stage

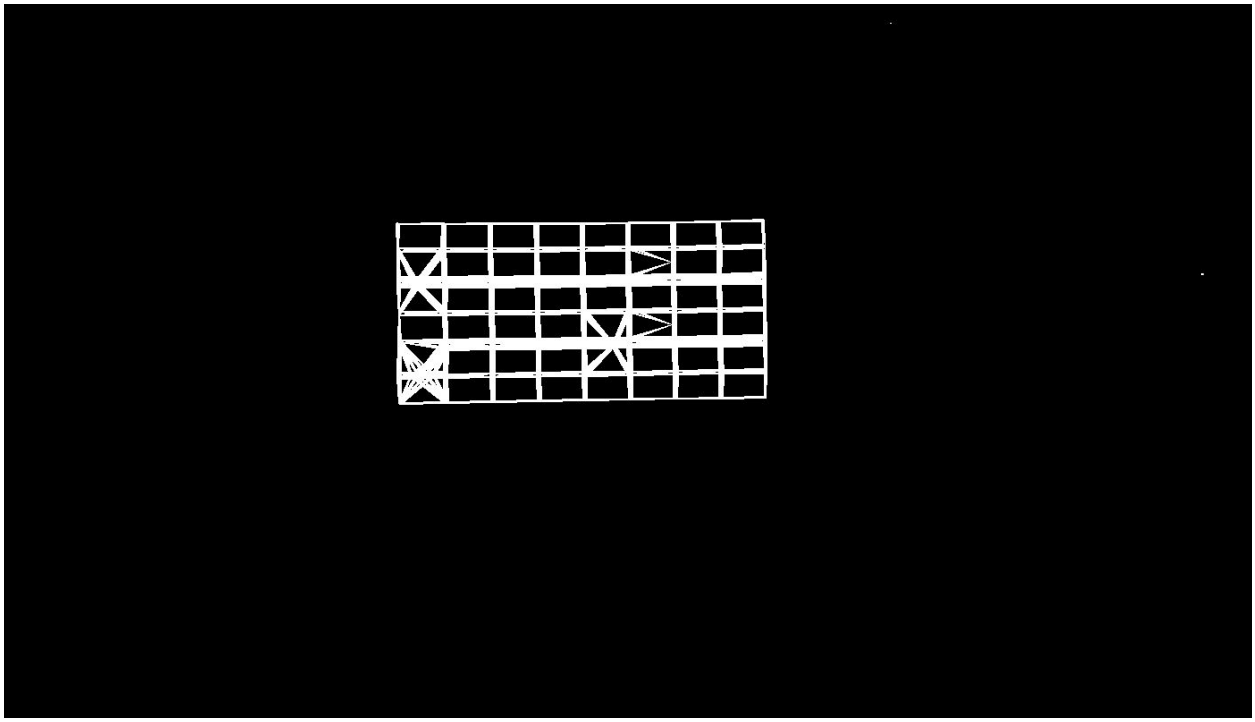
Now we could perform contour matching on the image from the edge detected image from previous stage.We take a template contour which is rectangular and compare each contour in the image to the template using the function “matchShapes()” present in OpenCV.This returns a value which indicates the how close the contour is to the template contour.The smaller the return value the closer it resembles the template contour with a return value of 0.00 being an ideal match. Again a threshold parameter is to be determined which separates the solar panel contours from the non solar panel contours(2.00 worked well in our case).But the shape wasn’t sufficient. There existed a few noise contours in the image which were close to the template shape but were either a lot bigger or a lot smaller than the panel contours. Thus also checking the contour area to determine if it is a panel or not yielded better results.However panels towards the edges would often go undetected as shown(Green contours are the ones classified as a panel and the red ones are non panels)



So to get better results, we combine the information present in the corner detected image.

We draw lines connecting two corner points provided that the line is smaller than a particular threshold(Another parameter to be tweaked is introduced).With a threshold of

50 we get the below result

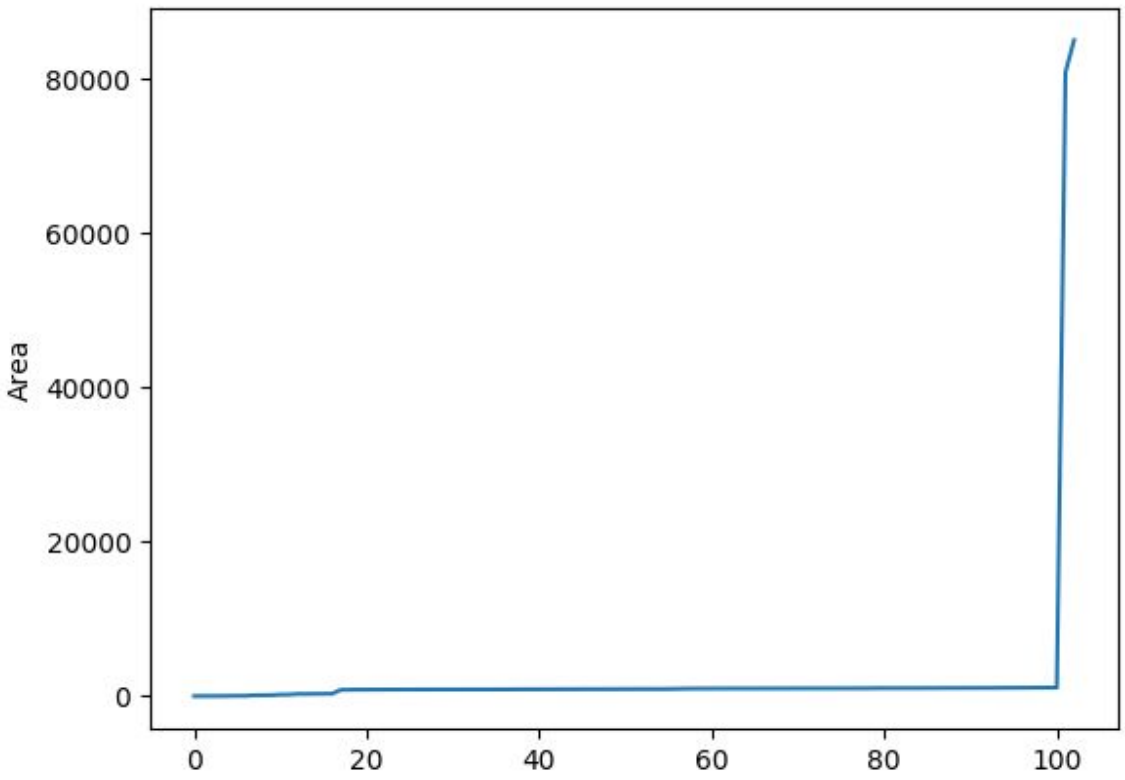


And performing the same contour matching scheme on the above image made it possible to detect the panels at the edges.

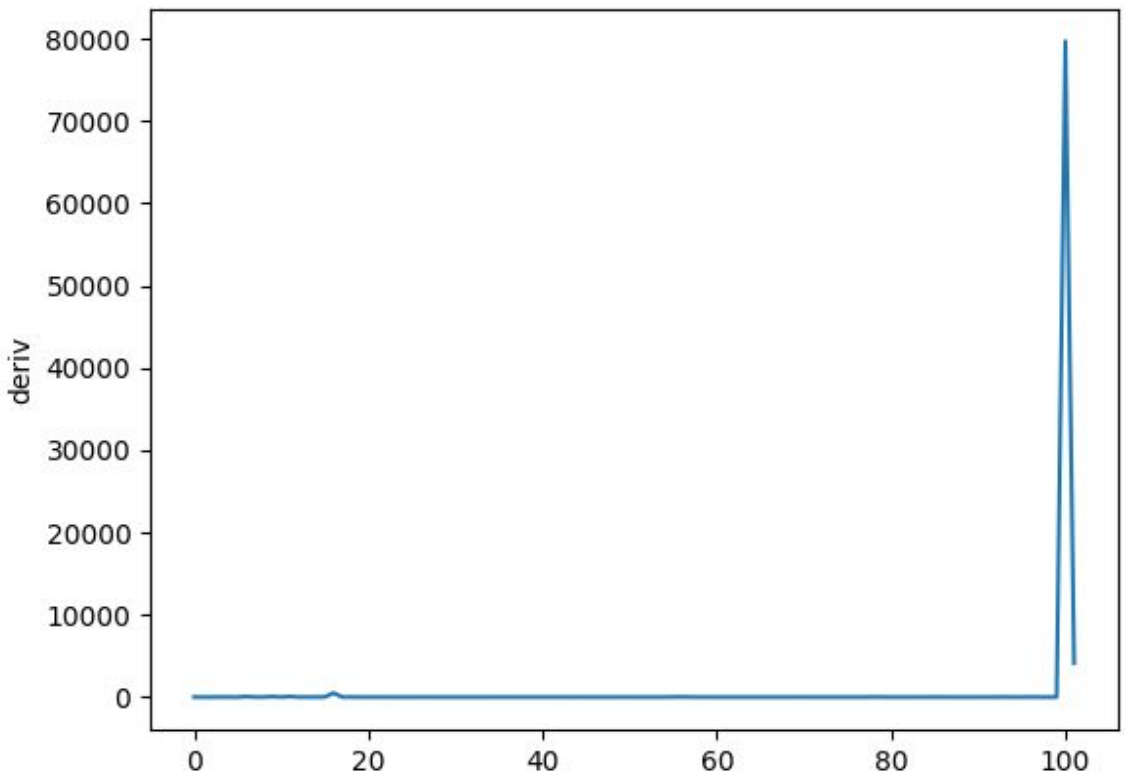
Also note that ‘closing’,’dilating’ and again ‘closing’ (Morphological transforms) the images before the contour matching step gave better results.

However for the algorithm to be applicable to other images as well, the contour area parameter and the line length threshold parameter has to be determined by some information in the image and cannot be set by the programmer.

A way to isolate the contour area could parameter could be thought of by looking at the distribution of all contour areas which passed the shape matching threshold.

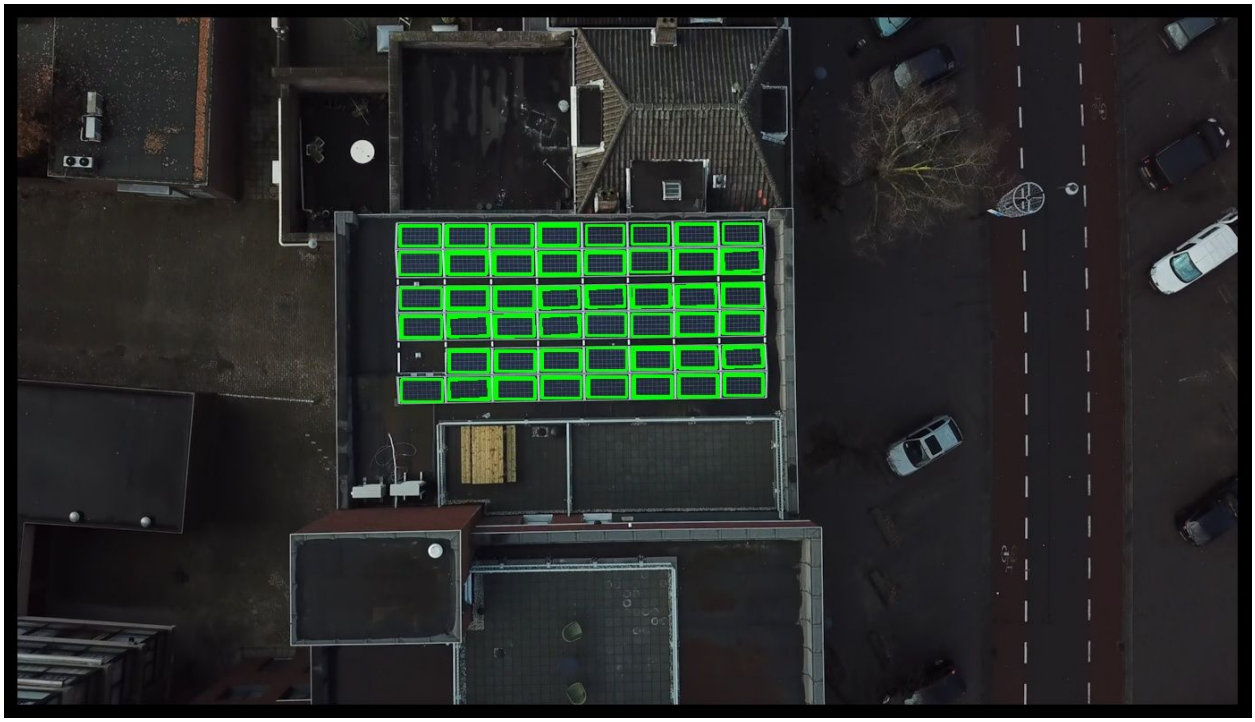


The areas that belong to solar panel contours are usually surrounded by noise much smaller or larger.So taking the derivative and isolating the areas corresponding to the points between the two largest spikes in the plot of the derivative would do the trick.



So, now overlaying the resulting contours with the original image gives us the following

result:



A way to isolate the line length threshold required to make use of the corner information could be by finding a way to isolate the solar panel contour areas in the distribution of the edge detected image alone and then taking the square root of the mean of the areas.(This I haven't been able to implement because I couldn't come up with a way to isolate the panel contour areas in the edge detected image alone)

Code:

```
#!/usr/bin/env python3
import numpy as np
import cv2
import os
from matplotlib import pyplot as plt

# constants
IMAGE_LOCATION="/home/ujwal/Desktop/"

def retarea(blah):
return blah[1]
def splice(in_imageROI,flag='b'):
in_image=np.copy(in_imageROI)
if flag=='b':
        in_image[:,:,1]=0;
        in_image[:,:,2]=0;
if flag=='g':
        in_image[:,:,0]=0;
        in_image[:,:,2]=0;
if flag=='r':
        in_image[:,:,0]=0;
        in_image[:,:,1]=0;
return in_image

def convert(in_image):
b,g,r=cv2.split(in_image)
imageROI=cv2.merge([r,g,b])
return imageROI

def getimage(filename,flag=1):
image=cv2.imread(os.path.join(IMAGE_LOCATION,filename),flag)
return image

def main():
x=15
image=getimage("8.png")
image=cv2.copyMakeBorder(image,x,x,x,x,cv2.BORDER_CONSTANT,value=0)
imageROIdisc=np.zeros(image.shape)
imageROI,disc,disc1=cv2.split(imageROIdisc)
disc2=np.copy(disc1)
imageROIfinal=np.copy(imageROI)


edge=cv2.Canny(image,300,600)
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
gray=np.float32(gray)
```

```
dst = cv2.cornerHarris(gray,2,3,0.04)
disc1[(dst>0.01*dst.max())]=255

todo=[(0,0),(0,1)]
todraw=[]
threshold=0.4
tc=0
for i in range(0,image.shape[0]):
        for j in range(0,image.shape[1]):
                if  ((image[i][j][0]>image[i][j][1]) and (image[i][j][0]>(1.5*image[i][j][2])) and
(image[i][j][1] in range(30,120)) ):
                        imageROI[i][j]=255
                        todo.append((i,j))
                        print(str(i)+','+str(j))

for i in range(len(todo)):
                tc=0
                for k in range(0,x):
                        for l in range(0,x):
                                if imageROI[todo[i][0]+k-(int(x/2))][todo[i][1]+l-(int(x/2))]==255:
                                        tc=tc+1
                print(tc)
                if (tc/(x*x))>threshold:
                        for k in range(0,x):
                                for l in range(0,x):

imageROIfinal[todo[i][0]+k-(int(x/2))][todo[i][1]+l-(int(x/2))]=255

for ila in range(0,image.shape[0]):
        for jila in range(0,image.shape[1]):
                if (imageROIfinal[ila][jila]==255):
                        disc[ila][jila]=edge[ila][jila]
                        disc2[ila][jila]=disc1[ila][jila]
                        if disc1[ila][jila]==255:
                                todraw.append((jila,ila))

disc[disc2[:]>0]=255

threshold=50

for index in todraw:
        for jndex in todraw:
                if np.linalg.norm(np.array(index)-np.array(jndex))< threshold:
                        cv2.line(disc2,index,jndex,255,1)

cv2.imwrite('rest.png',disc2)
cv2.imwrite('reste.png',disc)
discnew=getimage('rest.png')
discnew[disc2[:]>0]=[0,0,255]
template = getimage('cntTemp.png',0)
img_rgb = getimage('rest.png')
img_gray = getimage('rest.png',0)

kernel=np.ones((5,5),np.uint8)
```

```
img_gray = cv2.morphologyEx(img_gray, cv2.MORPH_CLOSE, kernel)
img_gray=cv2.dilate(img_gray,kernel)
img_gray = cv2.morphologyEx(img_gray, cv2.MORPH_CLOSE, kernel)
i=0
canderiuuuu=[]
candidate=[]
candidatelist=[]
print("*****************************************")
im2,contours,hierarchy = cv2.findContours(template,2,1)
im2q,contoursq,hierarchyq = cv2.findContours(img_gray,2,1)


print(cv2.contourArea(contours[0]))
for cnt in contoursq:
        ret = cv2.matchShapes(cnt,contours[0],1,0.0)
        #print(cnt)
        if (ret<2 ):
                print("Match")
                candidatelist.append((cnt,cv2.contourArea(cnt)))
                canderiuuuu.append(cv2.contourArea(cnt))




img_rgb = getimage('reste.png')
img_gray = getimage('reste.png',0)
#container=np.zeros(img_rgb.shape)
kernel=np.ones((5,5),np.uint8)
img_gray = cv2.morphologyEx(img_gray, cv2.MORPH_CLOSE, kernel)
img_gray=cv2.dilate(img_gray,kernel)
img_gray = cv2.morphologyEx(img_gray, cv2.MORPH_CLOSE, kernel)
i=0

print("*****************************************")

im2,contours,hierarchy = cv2.findContours(template,2,1)
im2q,contoursq,hierarchyq = cv2.findContours(img_gray,2,1)
print(cv2.contourArea(contours[0]))
for cnt in contoursq:
        ret = cv2.matchShapes(cnt,contours[0],1,0.0)


        if (ret<2 ):
                print("Match")
                candidatelist.append((cnt,cv2.contourArea(cnt)))
                canderiuuuu.append(cv2.contourArea(cnt))


canderiv=[]
candidate=sorted(candidatelist,key=retarea)
for i in range(0,(len(candidate)-1)):
```

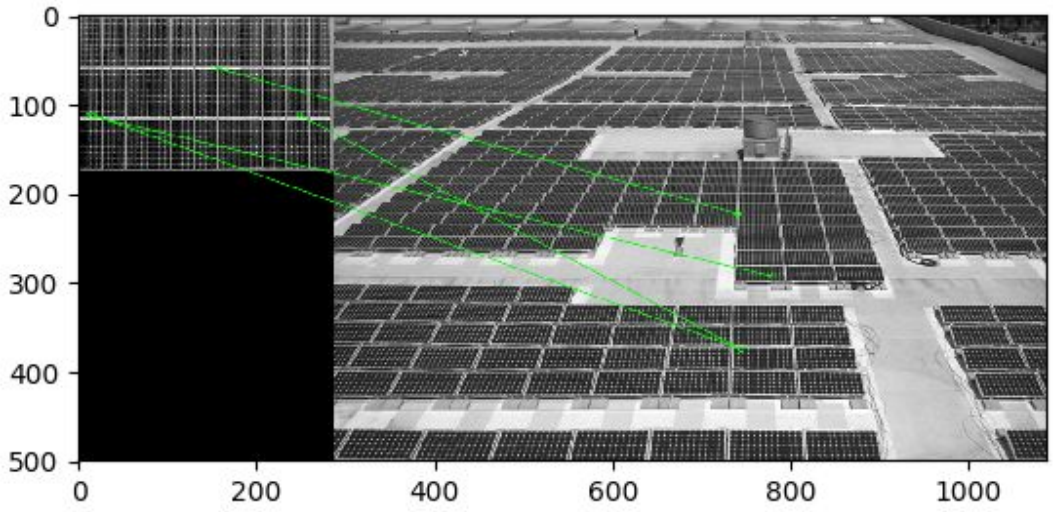
```
        canderiv.append(candidate[i+1][1]-candidate[i][1])
canderiuuuu.sort()
plt.plot(canderiuuuu)
plt.ylabel('Area')
plt.show()
upper=canderiv.index(max(canderiv))
del canderiv[upper:]
lower=canderiv.index(max(canderiv))
candidate1=candidate[(lower+1):(upper+1)]



for cnt in candidate1:
        rect=cv2.minAreaRect(cnt[0])
        box =cv2.boxPoints(rect)
        box =np.int0(box)
        cv2.drawContours(img_rgb,[box],0,(0,255,0),2)
        cv2.drawContours(image,[box],0,(0,255,0),2)

cv2.imshow('TaDa',image)
cv2.imwrite('fres.jpg',image)
cv2.waitKey(0)
cv2.imwrite('res.png',disc2)

print("*****************************************")

print(upper)
print(candidate[:][1])
print(candidate1[:][1])
print(canderiv)
if __name__ == '__main__':
main()
```

Other approaches that didn’t quite work out:

1.)Direct template matching

The above images are the result of template matching with threshold of 0.2,0.4 and 0.5 respectively

2.)SIFT and SURF algorithms



The solar panel wasn't unique enough to generate a good set of unique features that is typically needed for SIFT implementation(SIFT works well if you are looking for objects that are unique such as a particular billboard ad in an image of many billboard signs)