REPORT: # PATTERN DETECTION

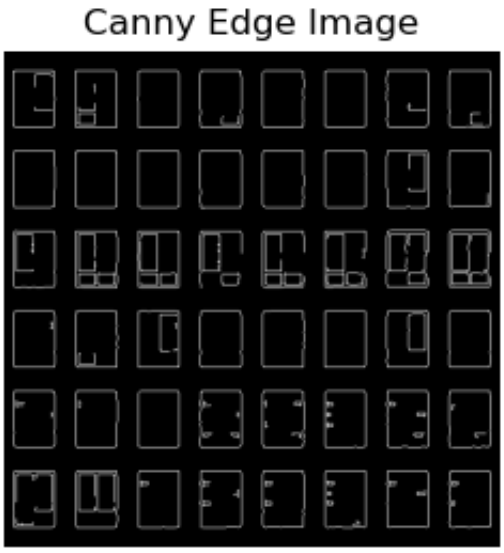
Atanda Abdullahi Adewale

METHODS

Pre-processing: Convert the image to Gray scale and apply any necessary filtering techniques to enhance edges, such as Gaussian blur or median filtering.

Edge detection: Use an edge detection algorithm, such as Canny edge detection, to extract edges from the image.

```
cv2.imshow('Original', img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_blur = cv2.GaussianBlur(gray, (5, 5), 0)
edges = cv2.Canny(gray_blur, 250, 450)
```



I have applied a Gaussian blur of 5x5 kernel size and increased the high and low threshold of Canny filter to get a better representation of window edges

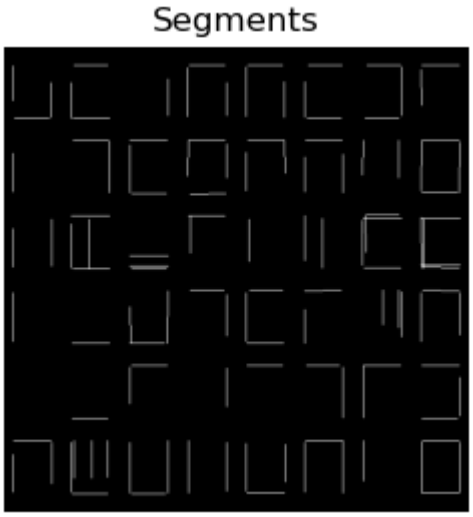
Extract segment primitives: simply obtain segments list for x and y directions where the intensity is not zero Or Use a line segment detection algorithm, such as the Hough transform, to extract a list of segment primitives from the edges

Associate discriminant information: For each segment primitive, calculate discriminant information such as the beginning of the segment, length, intensity, etc. This information will be used later to assemble and score segments.

```python
# Extract a list of segment primitives using Hough transform
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 9, minLineLength=35, maxLineGap=5)
segments = []
for line in lines:
    x1, y1, x2, y2 = line[0]
    segment = {'x1': x1, 'y1': y1, 'x2' : x2,'y2' : y2, 'length': np.sqrt((x2-x1)**2 + (y2-y1)**2)}
    if abs(x2-x1) > abs(y2-y1):
        segment['direction'] = 'horizontal'
    else:
        segment['direction'] = 'vertical'
    if segment['length']>=20:
        segments.append(segment)
#print(segments)
```

I have used cv2.HoughlinesP function to to generate the lines and extract the x and y direction from edge image representing Horizontal and Vertical segments, then set a minimum length of 35 as window width or Height.

Segment assembly: Design an algorithm to assemble the "closest" segments representing "approximately" a window. One approach is to find pairs of segments that are roughly perpendicular and of similar length. These pairs can then be combined into rectangular shapes that approximate a window.




```python
# Assemble and extract estimated windows
window_segments = []
window_coods = []

thickness = 2
threshold_gap = 10 # maximum gap between segments to be considered part of the same window
threshold_score = 1  # minimum score for a window to be considered valid
while len(segments) > 0:
    current_window = []
    current_window.append(segments.pop(0))
    list_temp=[]
    for i in range(len(segments)):
        if (segments[i]['direction'] != current_window[-1]['direction'] ) and\
          (((abs(segments[i]['x1'] - current_window[-1]['x1']) <= threshold_gap) and (abs(segments[i]['y1'] - current_window[-1]['y1']) <= threshold_gap)) or \
           ((abs(segments[i]['x2'] - current_window[-1]['x2']) <= threshold_gap) and (abs(segments[i]['y2'] - current_window[-1]['y2']) <= threshold_gap )) or \
           ((abs(segments[i]['x1'] - current_window[-1]['x2']) <= threshold_gap) and (abs(segments[i]['y1'] - current_window[-1]['y2']) <= threshold_gap )) or \
           ((abs(segments[i]['x2'] - current_window[-1]['x1']) <= threshold_gap) and (abs(segments[i]['y2'] - current_window[-1]['y1']) <= threshold_gap )) ):
           #current_window.append(segments.pop(i)) #current window
           list_temp.append(i)
           current_window.append(segments[i])
           break
```

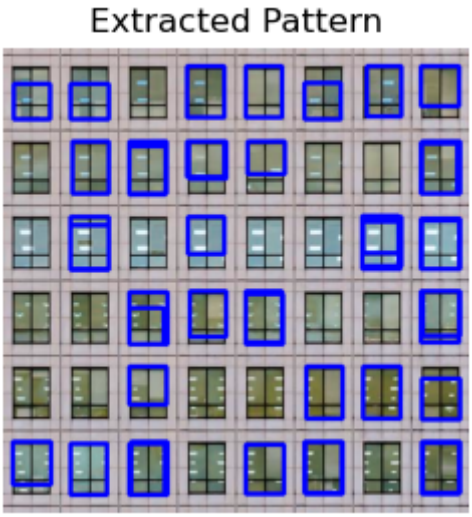
To assemble the segments, I set the gap: the distance between each segments coordinates to 10 for effective result and set minimum score of valid window to 1

Scoring: Calculate a score for each assembled segment based on its properties, such as aspect ratio, area, and alignment with neighboring segments. This score can be used to filter out non-window segments.

```
if len(current_window)>1:

    x1list=[segment_i['x1'] for segment_i in current_window ]
    x2list=[segment_i['x2'] for segment_i in current_window]
    xlist=x1list+x2list

    y1list=[segment_i['y1'] for segment_i in current_window ]
    y2list=[segment_i['y2'] for segment_i in current_window ]
    ylist=y1list+y2list

    x_left=min(xlist)
    x_right=max(xlist)
    y_up=min(ylist)
    y_down=max(ylist)

    cv2.line(img1, (x_left, y_up), (x_right, y_up), (0,0,255), 3)
    cv2.line(img1, (x_left, y_up), (x_left,y_down), (0,0,255), 3)
    cv2.line(img1, (x_left, y_down), (x_right,y_down), (0,0,255), 3)
    cv2.line(img1, (x_right, y_up), (x_right,y_down), (0,0,255), 3)
```

Repeat: Iterate through the remaining segments and repeat steps Segment Assembly and Scoring to find and score additional window segments.

Construct output image: Draw the segments lines containing only the estimated windows over the original image.



Conclusion:
Bounding boxes can not perfectly match the windows due to imperfect edge image by canny filter. Some perpendicular segments representing the window only in the X direction can't recover the full window rectangle.