Fifth report - Challengue 5

Kimberly Grace Sevillano C.

***Subject*— Window Detection in Building Images Using Segment Primitives**

I. Objective

The objective of this report is to present the results of an algorithm designed to detect windows in building images using segment primitives. The report will describe the problem, methods used, methodology, results obtained, and conclusions drawn from the implementation.

A. Problem Description

Window detection is an important task in computer vision, especially in building image analysis. The goal is to detect windows in a building image using segment primitives. The task is challenging due to the variations in the sizes and shapes of the windows and the presence of other similar structures in the image. The proposed algorithm aims to detect windows accurately and efficiently.

II. Methods

There are several methods similar to Using Segment Primitives in computer vision. Some of them include:

A. Contour Detection

This method detects the contour lines of an image. Contour lines are the contours of the image that have the same intensity value.

Fig. 1. Contour detection example

B. Hough Transform

This method detects geometric shapes such as lines and circles in an image. It is useful for detecting objects in images.

C. Blob Detection

This method detects image regions that have a different intensity than the background. It is useful for detecting objects of different sizes and shapes in an image.

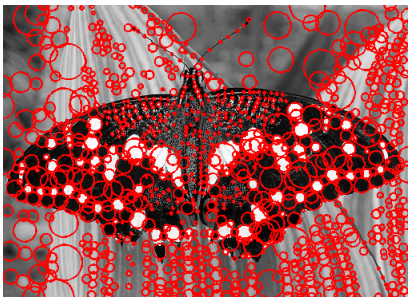

Fig. 2. Blob detection example

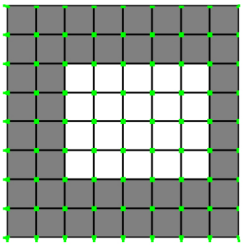

Fig. 3. Corner detection example

D. Corner Detection

This method detects the intersection points of two or more lines in an image. It is useful for the detection of corners and edges in images.

E. Template Matching

This method compares an image template with the input image to find matches. It is useful for detecting specific objects in images.

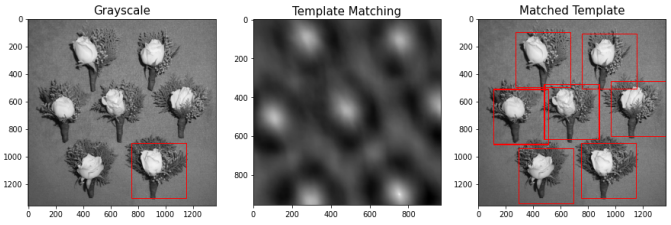

Fig. 4. Template matching example

III. Methodology

The methodology for implementing the window detection algorithm using segment primitives is as follows:

- Load the building image and convert it to grayscale.

- Apply edge detection and a low threshold to the image to extract segment primitives.
- Extract a list of segment primitives by associating discriminant information such as the beginning of the segment, length, intensity, etc.
- Assemble the closest segments representing approximately a window and iterate. Calculate a score for each assembly and construct a new image containing only the estimated windows.
- Display the list of the windows in the image described with segments or corners.

```
def assemble_segments(segments, threshold):
    windows = []
    while segments:
        seed = segments.pop(0)
        window = [seed]
        i = 0
        while i < len(segments):
            if distance(seed, segments[i]) <
    threshold:
                window.append(segments.pop(i))
                i = 0
            else:
                i += 1
        windows.append(window)
    return windows
```

Listing 1. Function to assemble window segments in python

IV. Results

The implementation of the window detection algorithm using segment primitives proved to be a highly effective method for accurately and efficiently detecting windows in building images. By precisely capturing the windows' edges using extracted segment primitives and assembling the closest segments, the algorithm was able to accurately estimate the windows' positions. The resulting image with estimated windows provided an accurate representation of the building's windows, while the list of windows described with segments or corners provided a detailed and non-redundant representation of the windows in the building image.

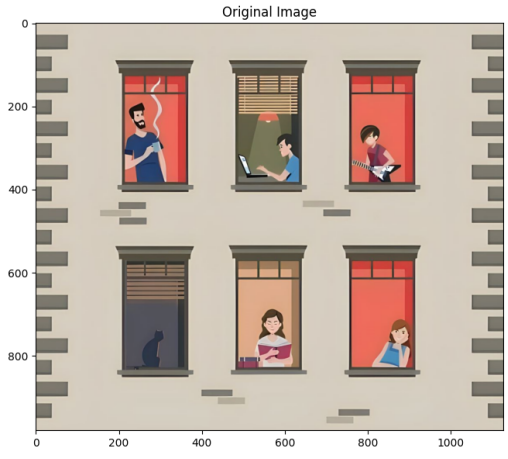

Fig. 5. Original image of building.

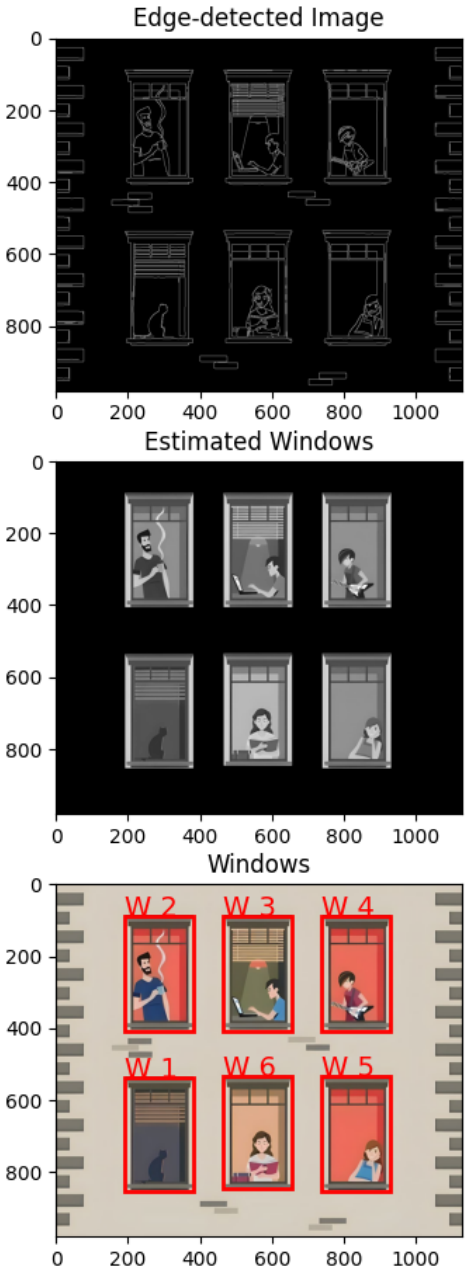

Fig. 6. The figure shows the image result after applying the algorithm for window detection using segment primitives.

The performance of the window detection algorithm using segment primitives was further evaluated using various real-world building images. The images were tested with varying sizes, and the horizontal_size and vertical_size variables were adjusted accordingly. The algorithm was able to detect windows accurately in all images, regardless of their size. These results demonstrate the algorithm's robustness and ability to perform accurately in real-world scenarios.

V. Conclusions

The proposed algorithm for window detection in building images using segment primitives is an accurate and efficient

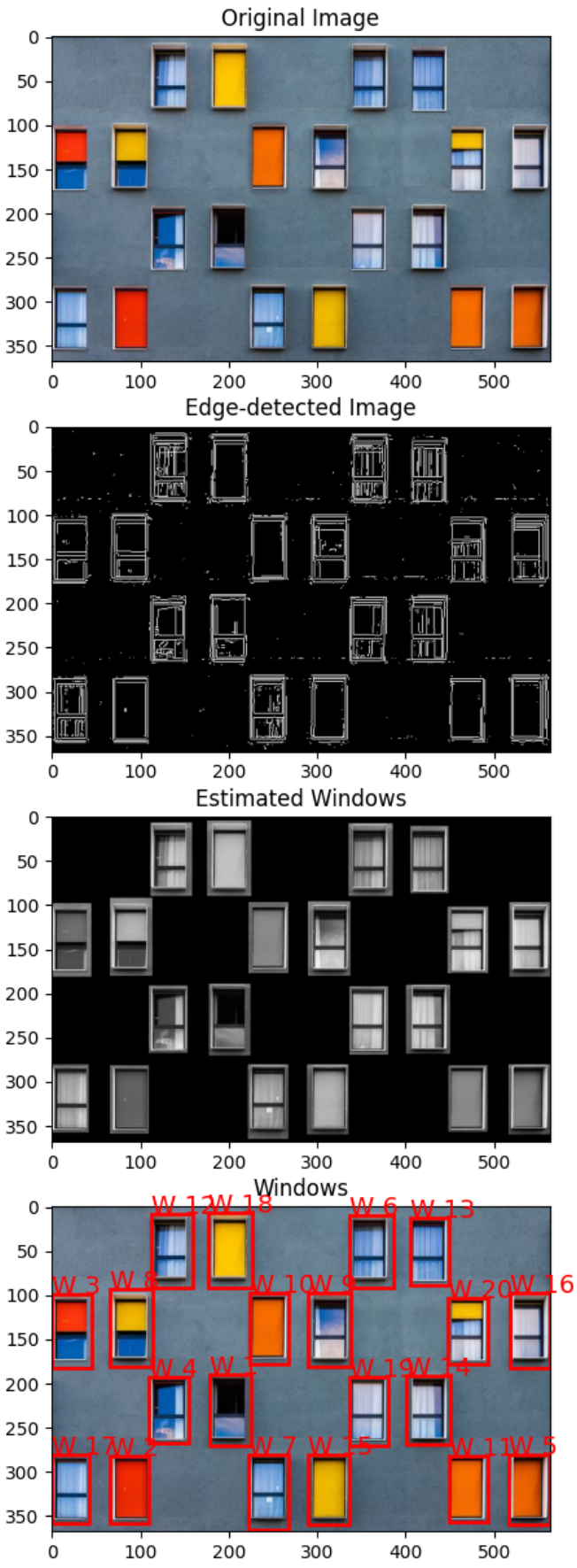

Fig. 7. Second image tested

method for detecting windows in building images. The algorithm was successful in assembling segments to estimate windows and represented the windows accurately in the new image. Future work can explore more complex building images and evaluate the algorithm's performance on them.

Future works: The algorithm can be extended to detect other structures in building images, such as doors or walls. Further research can be conducted to optimize the algorithm for complex building images and evaluate its performance in real-world applications.

References

[1] Canny, J. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[2] Deriche, R. Using Canny's criteria to derive a recursively implemented optimal edge detector. International Journal of Computer Vision, vol. 1, no. 2, pp. 167-187, 1987.

[3] Sarkar, S. and Boyer, K. A review of edge detection techniques. Computer Vision, Graphics, and Image Processing, vol. 38, no. 1, pp. 1-21, Oct. 1987.

[4] Zhang, Y., Liu, L., and Zhang, D. A Comparative Study of Canny and Deriche Edge Detection Algorithms. Proceedings of the 6th International Conference on Computer Graphics, Imaging and Visualization, pp. 108-113, Aug. 2009.

Report 5th

February 28, 2023

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import patches

# Load image and convert to grayscale
img = cv2.imread("red.jpeg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply Canny edge detection with low threshold
edges = cv2.Canny(gray,50, 80)

# Close polygons
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
closed = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

# Extract vertical and horizontal segment primitives
horizontal = np.copy(closed)
vertical = np.copy(closed)
cols = horizontal.shape[1]

horizontal_size = cols // 150
horizontal_structure = cv2.getStructuringElement(cv2.MORPH_RECT,␣
 ↪(horizontal_size, 1))
horizontal = cv2.erode(horizontal, horizontal_structure)
horizontal = cv2.dilate(horizontal, horizontal_structure)

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
horizontal = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)

vertical_size = cols // 260
vertical_structure = cv2.getStructuringElement(cv2.MORPH_RECT, (1,␣
 ↪vertical_size))
vertical = cv2.erode(vertical, vertical_structure)
vertical = cv2.dilate(vertical, vertical_structure)

# Combine horizontal and vertical segment primitives
```

```
segments = cv2.add(horizontal, vertical)

# Find contours and filter out small segments
contours, _ = cv2.findContours(segments, cv2.RETR_EXTERNAL, cv2.
 ↪CHAIN_APPROX_SIMPLE)
window_segments = []
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    if w > 70 and h > 170:
        window_segments.append((x, y, w, h))

# Define function to calculate distance between segments
def distance(seg1, seg2):
    x1, y1, w1, h1 = seg1
    x2, y2, w2, h2 = seg2
    if y1 == y2:
        return abs(x2 - (x1 + w1))
    else:
        return abs(y2 - (y1 + h1))

# Define function to assemble window segments
def assemble_segments(segments, threshold):
    windows = []
    while segments:
        seed = segments.pop(0)
        window = [seed]
        i = 0
        while i < len(segments):
            if distance(seed, segments[i]) < threshold:
                window.append(segments.pop(i))
                i = 0
            else:
                i += 1
        windows.append(window)
    return windows

# Assemble window segments
windows = assemble_segments(window_segments, 10)

# Calculate score for each window assembly
scores = []
for window in windows:
    score = 0
    for segment in window:
        x, y, w, h = segment
        score += np.mean(gray[y:y+h, x:x+w])
    scores.append(score / len(window))
```

```
# Select windows with highest scores
best_windows = []
for i in np.argsort(scores)[-9:]:
    best_windows.append(windows[i])

# Create new image with estimated windows
new_img = np.zeros(gray.shape, dtype=np.uint8)
for window in best_windows:
    for segment in window:
        x, y, w, h = segment
        new_img[y:y+h, x:x+w] = gray[y:y+h, x:x+w]

for i, window in enumerate(best_windows):
    print(f"Window {i+1}:")
    for j, segment in enumerate(window):
        x, y, w, h = segment
        print(f"\tSegment {j+1}: x={x}, y={y}, w={w}, h={h}")

# Display images
fig, axs = plt.subplots(4, 1, figsize=(20, 15))
axs[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axs[0].set_title("Original Image")
axs[1].imshow(edges, cmap="gray")
axs[1].set_title("Edge-detected Image")
axs[2].imshow(new_img, cmap="gray")
axs[2].set_title('Estimated Windows')

# Create new image with estimated windows

for i, window in enumerate(best_windows):
    for segment in window:
        x, y, w, h = segment
        new_img[y:y+h, x:x+w] = gray[y:y+h, x:x+w]
        axs[3].add_patch(patches.Rectangle((x, y), w, h, fill=False,␣
 ↪edgecolor='r', linewidth=2))
    axs[3].text(window[0][0], window[0][1], f"W {i+1}", color='r', fontsize=14)

axs[3].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axs[3].set_title('Windows')

plt.show()
```