Báo cáo giữa kỳ môn phân tích và xử lý hình ảnh

1. Tổng quan:

Mục tiêu: Phát hiện và định vị đối tượng trong ảnh.

Triển khai: Dùng OpenCV, Matplotlib, NumPy, và imutils để thực hiện.

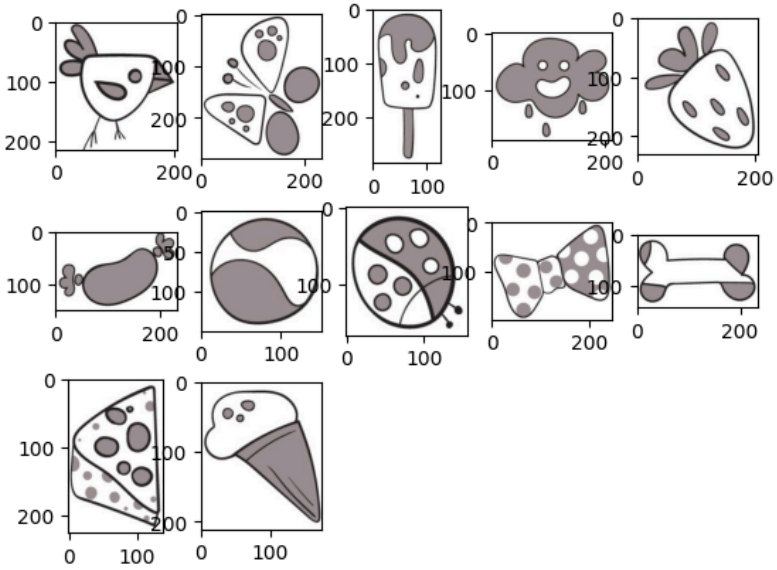
2. Phân tích code và vấn đề:

a. Khai báo thư viện.

```
import cv2
import matplotlib.pyplot as plt
import os
import numpy as np
```

b. Đọc và hiển thị ảnh để kiểm tra ảnh đã được đọc đúng hay chưa.

```
def read_image(folder_path):
  template = []
  for file in os.listdir(folder_path):
    img_bgr = cv2.imread(os.path.join(folder_path, file))
    template.append(img_bgr[:,:,::-1])
  return template
templates = read_image("/home/asus/Code/image-processing/finding/object 2")


for idx in range(len(templates)):
    plt.subplot(3, 5, idx + 1)
    plt.imshow(templates[idx])
```

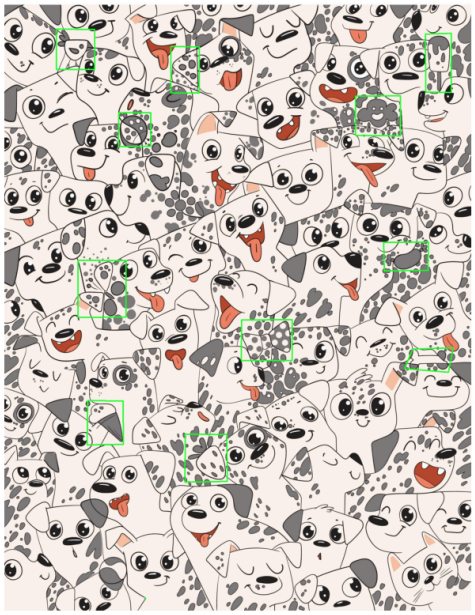


Đọc ảnh BGR rồi chuyển sang RGB. Sau đó cho vào mảng templates.

c. Phương pháp 1: Sử dụng SIFT và Homography.

```
output_image = image[:,:,::-1].copy()
output_templates = templates.copy()
```

```
image_gray = cv2.cvtColor(output_image, cv2.COLOR_BGR2GRAY)

for idx in range(len(output_templates)):
    template_gray = cv2.cvtColor(output_templates[idx], cv2.COLOR_BGR2GRAY)

    sift = cv2.SIFT_create()

    kp1, des1 = sift.detectAndCompute(template_gray, None)
    kp2, des2 = sift.detectAndCompute(image_gray, None)

    index_params = dict(algorithm=1, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    matches = flann.knnMatch(des1, des2, k=2)

    good_matches = [m for m, n in matches if m.distance < 0.85 * n.distance]

    if len(good_matches) > 10:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
        h, w = template_gray.shape
        pts = np.float32([[0, 0], [w, 0], [w, h], [0, h]]).reshape(-1, 1, 2)
        dst = cv2.perspectiveTransform(pts, M)

        image_boxed = cv2.polylines(output_image, [np.int32(dst)], True, (0, 255, 0), 3)

plt.figure(figsize=(15,10))
plt.imshow(image_boxed[:,:,::-1])
plt.axis('off')
plt.show()
```

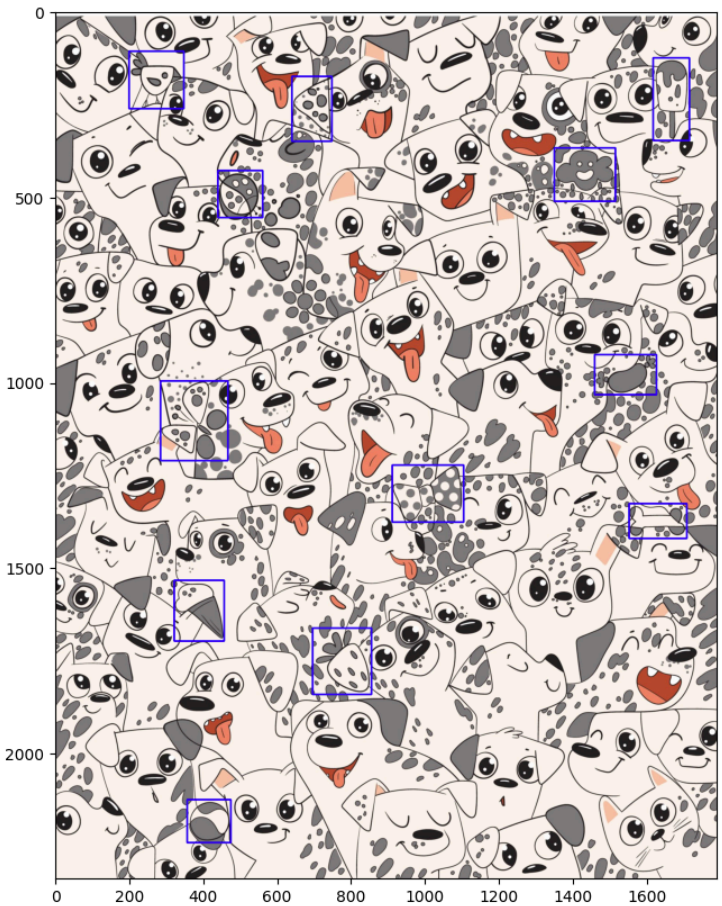


- ➢ Sử dụng thuật toán SIFT (Scale-Invariant Feature Transform) để phát hiện và định vị các đối tượng mẫu trong hình ảnh lớn, sau đó vẽ một hình chữ nhật bao quanh khu vực khớp.
- ➢ Phương pháp này tập trung vào việc phát hiện đối tượng dựa trên đặc trưng hình học, phù hợp với các đối tượng có thể bị xoay, thay đổi tỷ lệ hoặc biến dạng phối cảnh.
- ➢ Các bước:
    - Chuẩn bị: Chuyển ảnh lớn và hình ảnh mẫu sang ảnh xám bằng cv2.cvtColor.
    - Sử dụng SIFT để tạo đối tượng.
    - Sử dụng FlannBasedMatcher để tìm các cặp đặc trưng giữa ảnh mẫu và ảnh lớn.
    - Tính homography nếu đủ cặp khớp tốt, tính ma trận homography rồi dùng M để biến đổi tọa độ 4 góc của ảnh sang tọa độ tương ứng trong ảnh lớn.
    - Rồi vẽ kết quả bằng matplotlib.

d. Phương pháp templates matching với thay đổi tỉ lệ.

```
output_image = image[:,:,::-1].copy()
output_templates = templates.copy()
```

```python
import imutils
scales = np.linspace(0.5, 1, 10)

(tH, tW) = output_image.shape[:2]

for idx in range(len(templates)):
    output_template = output_templates[idx]
    best_match = None
    max_val = -1
    best_loc = None
    best_scale = None
    best_w, best_h = None, None
    for scale in scales:
        resized = imutils.resize(output_template, width=int(output_template.shape[1] * scale))
        w, h = resized.shape[1], resized.shape[0]

        if h > tH or w > tW:
            break

        result = cv2.matchTemplate(output_image, resized, cv2.TM_CCOEFF_NORMED)
        (_, maxVal, _, maxLoc) = cv2.minMaxLoc(result)

        if maxVal > max_val:
            best_match = maxLoc
            max_val = maxVal
            best_loc = maxLoc
            best_scale = scale
            best_w, best_h = w, h

    if best_loc is not None:
        top_left = best_loc
        bottom_right = (top_left[0] + best_w, top_left[1] + best_h)
        cv2.rectangle(output_image, top_left, bottom_right, (0, 0, 255), 3)

plt.figure(figsize=(10,10))
plt.imshow(output_image)
plt.show()
```



- ➢ Sử dụng Template Matching để tìm vị trí của ảnh mẫu trong ảnh lớn, với khả năng thay đổi tỷ lệ (scale) để tăng độ chính xác.
- ➢ Phương pháp này đơn giản hơn SIFT, dựa trên so sánh trực tiếp giữa ảnh mẫu và ảnh lớn, phù hợp với các đối tượng không bị xoay hoặc biến dạng quá nhiều, nhưng có thể thay đổi kích thước.
- ➢ Các bước:
    - Tạo mảng scales từ 0.5 đến 1(10 mức tỷ lệ) để thử nghiệm các kích thước khác nhau của mẫu ảnh.
    - Lấy chiều cao và chiều rộng của ảnh lớn.
    - Thay đổi tỉ lệ và khớp mẫu: Với mỗi ảnh mẫu đổi kích thước thành imutils.resize theo từng tỉ lệ trong scales.
    - Dùng cv2.matchTemplate với phương pháp TM_CCOEFF_NORMED để tìm khu vực tốt nhất trong ảnh lớn.
    - Lưu lại giá trị khớp nhất max_val, vị trí best_loc và kích thước tương ứng (best_w, best_h).
    - Nếu có vị trí khớp tốt thì hình chữ nhật màu bao quanh khu vực ảnh lớn.
    - Hiển thị kết quả bằng matplotlib.

3. So sánh hai phương pháp:

a. SIFT + Homography:
    - Ưu điểm: Xử lý tốt các trường hợp đối tượng bị xoay, thay đổi tỷ lệ, hoặc biến dạng phối cảnh.
    - Nhược điểm: Phức tạp hơn, yêu cầu đủ điểm đặc trưng khớp, có thể không hoạt động tốt nếu ảnh mẫu quá đơn giản hoặc thiếu đặc trưng.

b. Template Matching:
    - Ưu điểm: Đơn giản, nhanh, hiệu quả với các đối tượng có hình dạng cố định và không bị xoay.
    - Nhược điểm: Không xử lý tốt xoay hoặc biến dạng phối cảnh, chỉ dựa trên so sánh pixel.