UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

COMPUTER VISION

Report Lab 03
Object Detection Using Local Feature

Lecturers Phạm Minh Hoàng
Nguyễn Trọng Việt
Võ Hoài Việt

Student Võ Nguyễn Hoàng Kim
21127090

CONTENT

A. SELF-ASSESSMENT FORM

Requirement	Level of completion
Object detection application using SIFT feature	100%

B. IMPLEMENTATION

I. SIFT

1. What is SIFT

- SIFT stands for Scale-Invariant Feature Transform, was invented by David Lower [1].
- It is a widely used algorithm in computer vision for detecting and describing local features in images.
- These features are points in the image that are invariant with scale, rotation, illumination, and viewpoint changes.

2. SIFT Algorithm

The following algorithm is derived and synthesized from the documents [2], [3]:

- **Scale-space extrema detection:** The algorithm starts by constructing a scale space, which is a series of blurred images obtained by convolving the original image with Gaussian kernels of different sizes. This process helps in identifying features at different scales. Then, it looks for local extrema maxima or minima) in the scale-space to find potential keypoint locations.
- **Keypoint Localization**: ensures accurate localization of keypoints and helps in discarding unstable keypoints.
- **Orientation Assignment:** After localizing keypoints, SIFT computes the orientation of each keypoint by considering the gradient magnitude and orientation of the pixels in its neighborhood. This step ensures that the descriptors are invariant to image rotation.
- **SIFT construction.**

II. Object detection

1. What is object detection

- Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos [4].
- In the scope of this program, object detection is stopped at the task of detecting an object with the template image within a scene image.

2. Problem statement

Input

- To implement the program, we use 2 images as inputs:
    - A template image: template of a known object.
    - A scene image: consisting of this object.

Output

- To provide an overview of the achieved results, the output will display 2 images:
    - An initial template image.
    - A result image with the detected object is highlighted by drawing lines around it.

Note: *In the object detection task, only a single result image is saved in the folder.*

Method [5]

- Load the template and scene image, convert them to Grayscale images.
- Create ORB (Oriented fast and Rotated Brief) with the underlying SIFT algorithm.
- Calculate the key points and descriptors for the template and scene image separately
- Create the Brute Force or kNN (K- nearest neighbor) matcher with the required parameters and then use the matches which yield the Matches based on the similarity distance.
    - Two matches as good if the distance between them is 75%.
- Use the good matches calculated in the above step to detect the given object in the scene.

3. Implement

To perform the task of object detection, we construct a function **objectDetection()** with the parameters being the template image and the scene image. Below, we will present the pseudo code and provide detailed explanations of the function.

a. Pseudo code

function **objectDetection**(templateImg, sceneImg):

```
// Create a result image like as initial image
resImg = clone(sceneImg)

// Convert images to grayscale
sceneGray = convertToGray(sceneImg)
templateGray = convertToGray(templateImg)

// Step 1: Initialize SIFT detector
sift = initializeSIFT()

// Step2: Detect keypoints and compute descriptors
keypoints_template, descriptor_template = sift→detectAndCompute(templateGray)
keypoints_scene, descriptor_scene = sift→detectAndCompute(sceneGray)

// Step 3: Match descriptors using kNN (k-nearest neighbors)
matches = knnMatch (descriptor_template, descriptor_scene, 2)

// Step 4: Filter matches based on distance ratio
goodMatches = filterMatches(matches)
```

```
    // Step 5: Extract matched keypoints
    matchedPoints1, matchedPoints2 = extractMatchedKeypoints(goodMatches, keypoints_template,
keypoints_scene)

    // Step 6: Find homography
    H = findHomography(matchedPoints1, matchedPoints2)

    // Step 7: Define corners of the template image
    templateCorners = defineTemplateCorners(templateImg)

    // Step 8: Map template corners to scene image using homography
    sceneCorners = mapTemplateCornersToScene(templateCorners, H)

    // Draw lines around the detected object
    drawLines(resImg, sceneCorners)
  return resImg
```

b. Explanation

After converting the input images to grayscale, we begin searching for local features using SIFT and utilize it to detect objects.

- Step 1: The pointer *sift* is initialized by using the **SIFT::create()** function of OpenCV.
- Step 2: Detect keypoints in both images and compute descriptors for each of the keypoints, this is done using the **detectAndCompute()** function provided by *sift*.
- Step 3: Find the closest matches between descriptors from the template image to the scene. In this program, we use k-nearest neighbors as a matcher.
    - With the matcher was initialized, **knnMatch()** function is used to performs k-nearest neighbor matching between the descriptors of keypoints in the template image and descriptors of keypoints in the scene image.
    - In this case, the function finds the top 2 nearest neighbors for each descriptor in the query set (template descriptors) from the train set (scene descriptors).
- Step 4: Use a threshold to filter and retain good matches.
    - For each element in the set of matches, we consider the distance between it and the next point: if the distance between two matches falls within 75%, that match will be considered good and added to the list of good matches.
- Step 5: extracts the matched keypoints from both the template and scene images and stores them in separate vectors.

- Step 6: Find the homography transformation between two sets of points:
    - To compute the homography (transformation matrix) *H* that maps points from the template image to their corresponding points in the scene image.
    - This transformation is estimated based on the matched keypoints using the RANSAC algorithm, which helps in robustly estimating the transformation even in the presence of outliers or incorrect matches.
- Step 7: To determine the corners of the template image, we rely on its dimensions (number of rows, number of columns) to determine the coordinates of these points.
- Step 8: Map template corners to scene image using homography H. This task using perspectiveTransform() function to implement.
- Finally, we draw the bounding lines around the object found in the scene image using the **line()** function.

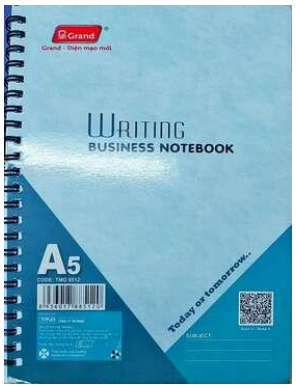
4. Result

- This section will present the results (including input and output) of some example images:

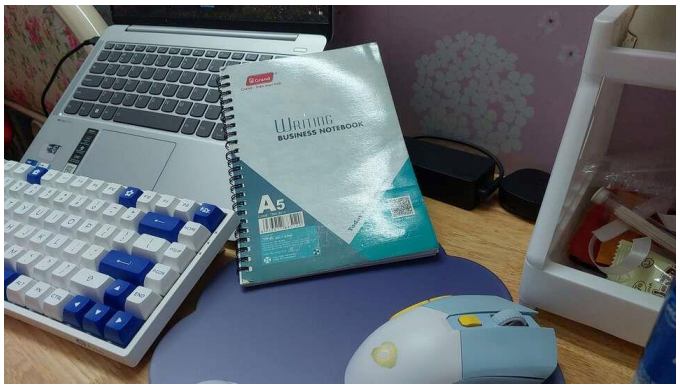
a. Example 1

Input

Template image

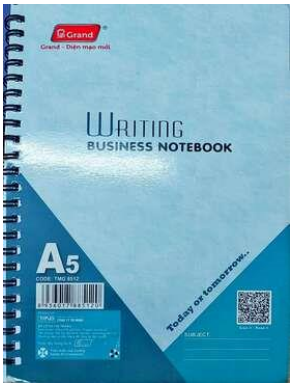


Scene Image

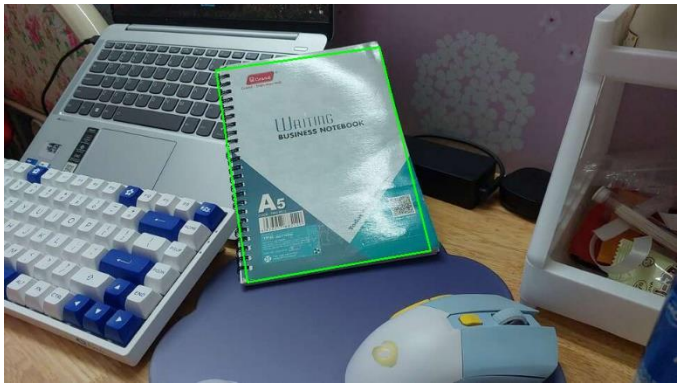


Output

Template image



Output



b. Example 2

Input

Template image

Scene image

Output

Template image

Output

C. USER GUIDE

I. Folder Structure

- There are 4 folders:
  - doc: Contains the report.
  - data: Contains the 2 sub-folders: Input and Output, where input images and output images in.
  - exe: Contains executable file. The executable file is named **21127090.exe**
  - source: Contains the source code.

II. How to run the source code

*The source code is implemented in **Visual Studio 2022** on the Windows operating system.*

- Run the source code, follow these steps:
  - Step 1: Open the terminal of your browser.
  - Step 2: Enter the path to the directory containing the source code.

- Step 3: Enter the command line based on the following structure:

  21127090.exe *-sift* <*TemplateImagePath*> <*SceneImagePath*> <*OutputFilePath*>

  Notation:

  - **21127090.exe:** the executable file.
  - **-sift:** the name of command.
  - **TemplateImagePath:** the path of templae image.
    - Example: ..\Data\Input\template.jpg
  - **SceneImagePath:** the path of scene image.
    - Example: ..\Data\Input\scene.jpg
  - **OutputImagePath:** the path of output image. (we must assign the name for the output image)
    - Example: ..\Data\Output\result.jpg

D. REFERENCES

[1] "Scale-invariant feature transform," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform.

[2] 21TGMT, "Scale-Invariant Feature Transform".

[3] D. Tyagi, "Introduction to SIFT(Scale Invariant Feature Transform)," Medium, 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40.

[4] Wikipedia, "Object detection," Wikipedia. [Online].

[5] D. Prasad, "OpenCV Feature Matching — SIFT Algorithm (Scale Invariant Feature Transform)," Medium, [Online]. Available: https://medium.com/analytics-vidhya/opencv-feature-matching-sift-algorithm-scale-invariant-feature-transform-16672eafb253.