HO CHI MINH UNIVERSITY OF SCIENCE

COMPUTER VISION - LAB03

March 17, 2024

Author

Nguyen Viet Kim - 21127333

Contents

1 Introduction

This report provides an overview of the algorithm, about the mathematics and the steps of the Object detection using Local Feature with SIFT (Scale-invariant feature transform) method in C++ using openCV. The report also consists of the program's result and the user guide at the end of the paper.
Self-assessment table:

No.	Tasks	Percentage	Note
1	SIFT function	100%	Used knnMatcher

2 File Management

```
21127333.zip
├─Data
├─Document
│ └─21127333_Report.pdf
└─Sources
  ├─.vscode
  │ ├─c_cpp_properties.json
  │ ├─launch.json
  │ └─tasks.json
  ├─bin
  │ └─21127333.exec
  ├─include
  │ ├─common.hpp
  │ └─sift.hpp
  └─src
    ├─main.cpp
    └─sift.cpp
```

- .vscode: Folder for configure the Visual Studio Code CC++ for running the program
- bin: Folder for saving execution files for the application
- include: Folder for storing required C++ header files
- src: Includes the C++ files.

3 Algorithm Description

The program detect the required object from the template image that is in the scene image using the SIFT algorithm and the Brute-Force matching, including the following steps:

3.1 Load images

The *templateImage* and *sceneImage* are loaded from files and converted to grayscale. These images represent the object (template) and the scene (where we want to find the object).

```
    // Load images
    Mat templateImage = imread(objectImagePath, IMREAD_GRAYSCALE);
    Mat sceneImage = imread(sceneImagePath, IMREAD_GRAYSCALE);
```

3.2 Define Detector and Descriptor

In this program, the object is detected using the Scale-invariant feature transform (SIFT) algorithm with Brute-Force matching for detecting and matching points in 2 images. We need initialize the SIFT detector and the Brute-Force matcher before using them.

```
    // Initialize feature detector and descriptor extractor
    Ptr<Feature2D> detector = SIFT::create();
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create("BruteForce");
```

3.3 Feature Detection and Description

The SIFT algorithm detects keypoints (distinctive points) in both images. For each keypoint, a descriptor is computed. Descriptors capture local information around keypoints, making them suitable for matching. The program use the built-in function from openCV to detect and compute the keypoints and store into vector $keypoints1$ and $keypoints2$.

```
    // Detect keypoints and compute descriptors
    vector<KeyPoint> keypoints1, keypoints2;
    Mat descriptors1, descriptors2;
    detector->detectAndCompute(templateImage, noArray(), keypoints1, descriptors1);
    detector->detectAndCompute(sceneImage, noArray(), keypoints2, descriptors2);
```

3.4 Matching Descriptors

The matcher (Brute-Force Matcher) compares the descriptors of keypoints from the template and scene images. It finds the best matches based on a distance metric (usually Euclidean distance or Hamming distance). The Brute-Force use the knnMatches to match the points

```
    // Match descriptors
    vector<vector<DMatch>> knnMatches;
    matcher->knnMatch(descriptors1, descriptors2, knnMatches, 2);
```

3.5 Filtering Match Points

To increase the accuracy of the point, the program use the Lowe's algorithm with the threshold ratio of 0.7f. the vector $goodMatches$ stores only the point with low distance from the threshold.

```
    // Filter matches using Lowe's ratio test
    const float ratio_thresh = 0.7f;
    vector<DMatch> goodMatches;
    for (size_t i = 0; i < knnMatches.size(); i++) {
        if (knnMatches[i][0].distance < ratio_thresh * knnMatches[i][1].distance) {
            goodMatches.push_back(knnMatches[i][0]);
        }
    }
```

3.6 Find Homography

The program finds transformation between the template and scene images which is the homography. Homography is a perspective transformation that maps points from one image to another. It is used for aligning the template image with the detected object in the scene.

```
    // Find homography
    vector<Point2f> obj;
    vector<Point2f> scene;
    for (size_t i = 0; i < goodMatches.size(); i++) {
        obj.push_back(keypoints1[goodMatches[i].queryIdx].pt);
        scene.push_back(keypoints2[goodMatches[i].trainIdx].pt);
    }
    Mat H = findHomography(obj, scene, RANSAC);
```

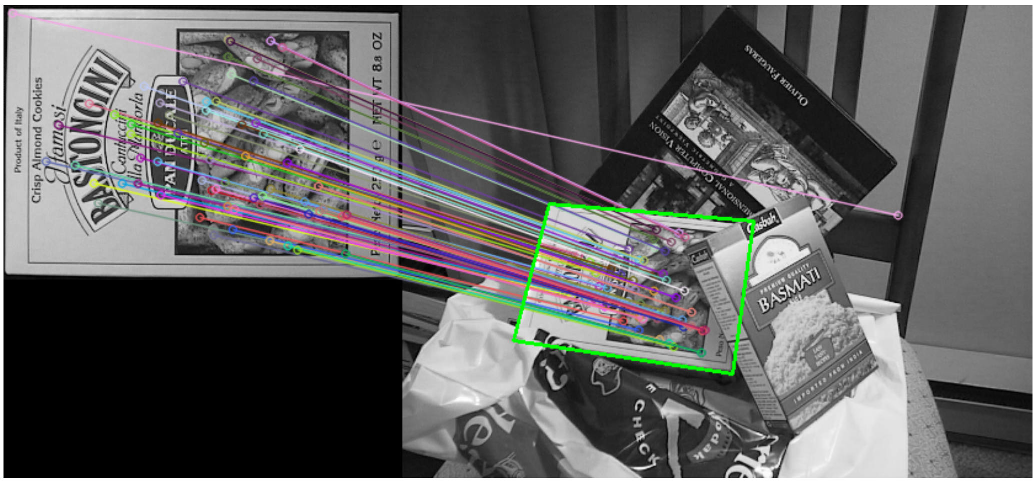
3.7 Draw Matches and Save Result

After all of the match point are computed, the program adjust the coordinate of the point corresponding to the images, matching corners and save the image via output file path given by the user

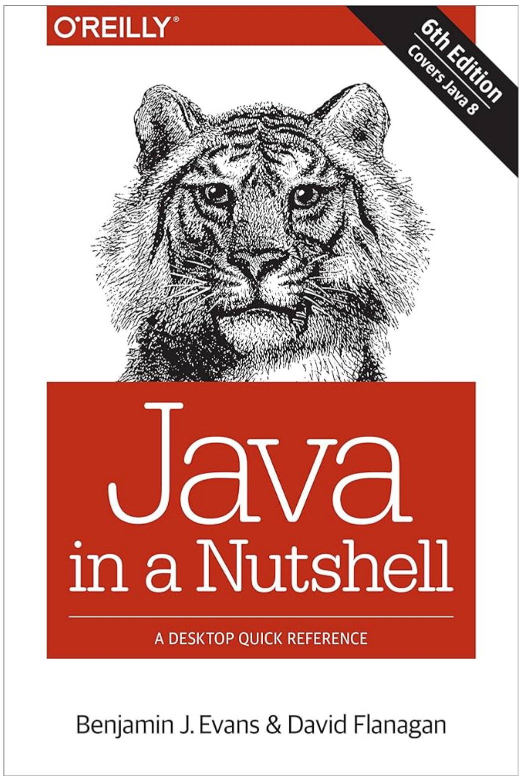
4 Program Result

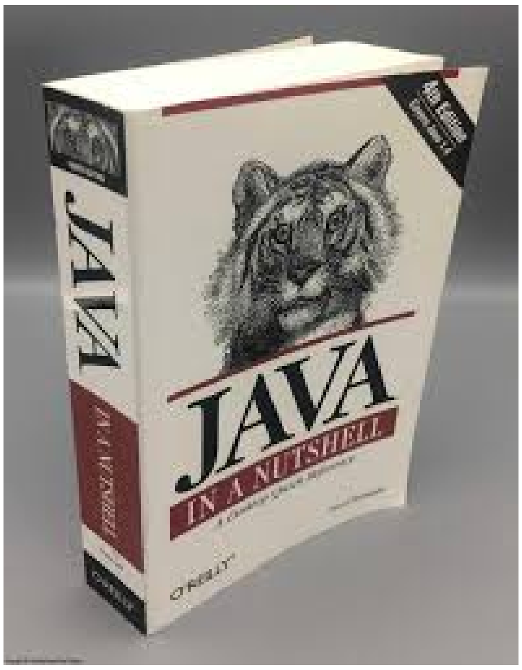
Object image

Scene image



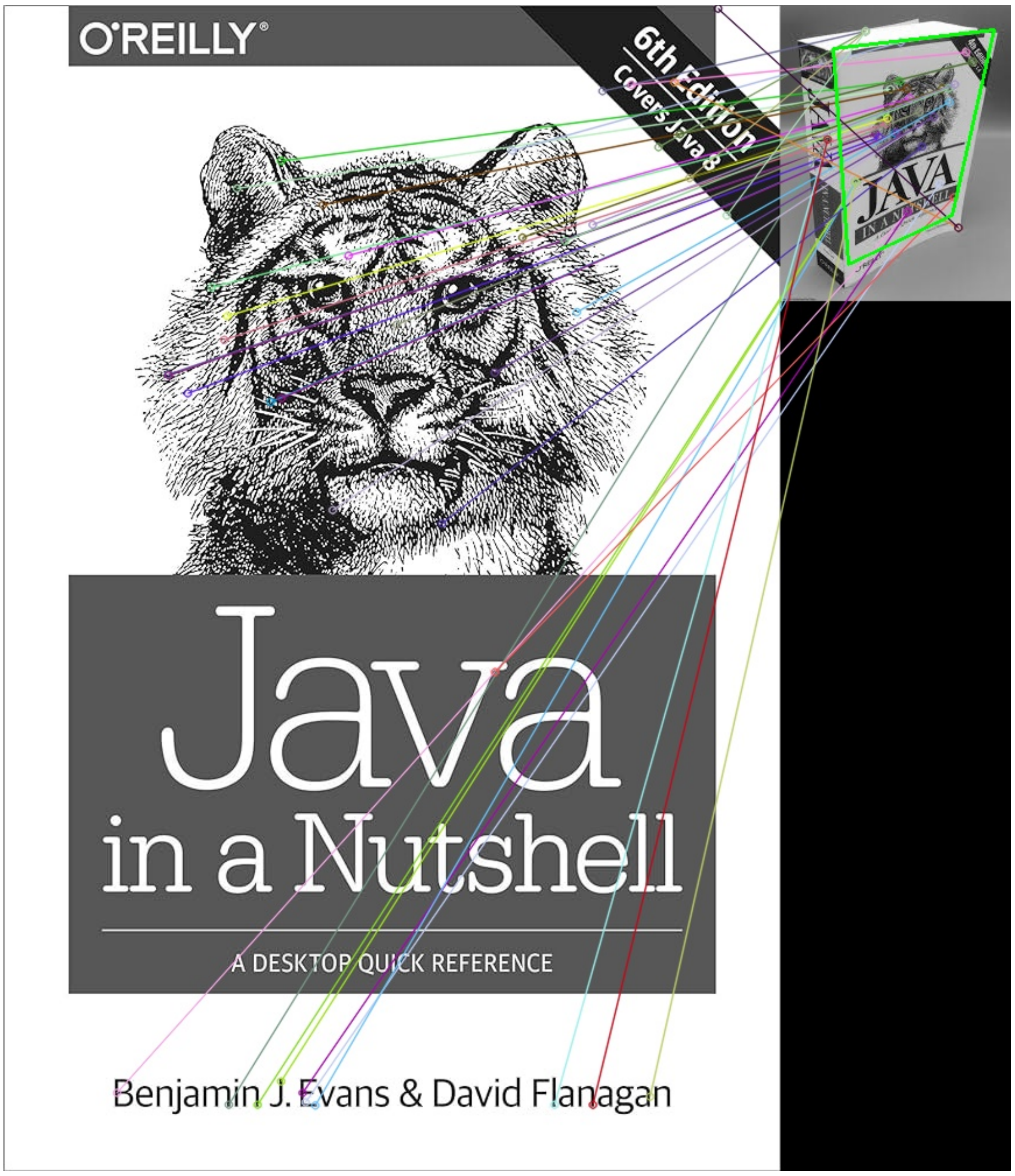
Result image





Object image

Scene image



Result image

5 User guide

Unzip the file 21127333.zip. Once ready, open the folder in the Visual Studio Code. Press *Ctrl* + ' to open the terminal from the VSCode. We need to locate to the bin directory and execute the program by running these commands:

```
$ cd /path/to/directory/bin
$ ./21127333 <mode> <template_file_path> <scene_file_path> <
    output_file_path>
```

User needs to select option for image processing mode. User can select the image for image processing and the output file name for image after the changes.
The program accepts command-line arguments to specify the mode of operation and input/output files. Below are the supported modes along with their usage:

- **-sift <template_image_path> <scene_image_path> <output_image_path**: Apply object detection using Local Feature.