HO CHI MINH UNIVERSITY OF SCIENCE

COMPUTER VISION - LAB02

March 11, 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 Harris Edge Detection implementation in C++ using openCV. The report also consists of the program's result and the user guide at the end of the paper.

2 File Management

The source code include these following folders:

```
21127333.zip
├── .vscode
│   ├── c_cpp_properties.json
│   ├── launch.json
│   └── tasks.json
└── Sources
    ├── bin
    │   └── 21127333.exec
    ├── data
    │   ├── Test.jpeg
    │   └── Harris.png
    ├── include
    │   ├── common.hpp
    │   └── harris.hpp
    └── src
        ├── harris.cpp
        └── main.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 Harris corner detection algorithm is based on the sum of squared difference (SSD function) with the following formula:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x-y)]^2$$

Where:

- $w(x,y)$: Window function
- $I(x+u,y+v)$: Shifted Intensity
- $I(x-y)$: Intensity

The Harris corner detection algorithm consists of the following steps:

3.1 Convert the image to grayscale

Since corner detection relies on changes in intensity, converting the image to grayscale simplifies the process.

$$G_{x,y} = 0.2126 \cdot R_{x,y} + 0.7152 \cdot G_{x,y} + 0.0722 \cdot B_{x,y}$$

In the code implementation, the program use the built in function that convert the image from RGB to Grayscale.

3.2 Apply Gaussian noise elimination

Gaussian blur is applied to the grayscale image to eliminate any noise or distortion that could affect corner detection.
Given a grayscale image G with pixel values represented as $G_{x,y}$, applying a Gaussian filter involves convolving the image with a Gaussian kernel K of size$n \times n$ and standard deviation :

$$G'_{x,y} = (G * K)_{x,y} = \sum_{i=1}^{n} \sum_{j=1}^{n} G_{x+i-1,y+j-1} \cdot K_{i,j}$$

where $G'_{x,y}$ represents the filtered pixel value at location (x, y) after convolution, and $K_{i,j}$ denotes the kernel coefficient at position (i, j).

$$K_{i,j} = \frac{1}{2\pi\sigma^2} e^{-\frac{(i-\frac{n+1}{2})^2+(j-\frac{n+1}{2})^2}{2\sigma^2}}$$

The implementation of the program use the built in function to apply the Gaussian filter to the image with openCV.

3.3 Compute the gradient of the image

Sobel operators are used to compute the gradient of the image in both the x and y directions. This gives us the rate of change of intensity in each direction.
The gradient of the image is computed using the Sobel operator:

$$I_x = \frac{\partial I}{\partial x} \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I$$

$$I_y = \frac{\partial I}{\partial y} \approx \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

3.4 Compute the Gaussian window

A Gaussian window is generated to be used in the subsequent steps of the algorithm. This window helps to weight the contributions of neighboring pixels.
A 2D Gaussian window is generated with size $N \times N$ and standard deviation σ:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

3.5 Compute the structure tensor

Using the gradient information and the Gaussian window, the structure tensor components M_{xx}, M_{yy}, and M_{xy} are computed.
The elements of the structure tensor M are computed:

$$M_{xx} = I_x^2$$
$$M_{yy} = I_y^2$$
$$M_{xy} = I_x \cdot I_y$$

3.6 Compute the determinant and trace

The determinant and trace of the structure tensor are computed, which are used to calculate the corner response. The determinant and trace of the structure tensor are calculated:

$$\det(M) = M_{xx} \cdot M_{yy} - M_{xy}^2$$
$$\text{trace}(M) = M_{xx} + M_{yy}$$

3.7 Compute the Harris corner response

Using the determinant, trace, and a user-defined parameter k, the Harris corner response is computed for each pixel.
The Harris corner response R is computed using the determinant and trace along with a constant k:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

3.8 Thresholding and non-maximum suppression

The corner response is thresholded to identify potential corner points. Non-maximum suppression is then applied to eliminate multiple responses for the same corner.

$$R'_{x,y} = \begin{cases} 255 & \text{if } R_{x,y} > \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

4 Program Result

Original image

Apply Harris corner detection

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> <file_input> <file_output>
```

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:

- **-harris <input_file> <output_file>**: Apply Harris edge detection to the input image.