Lab 2: Feature detection

MCHA4400

Semester 2 2023

Introduction

In this lab, you will become familiar with navigating the OpenCV API documentation (https://docs.opencv.org/4.8.0/), and use it to employ the following methods for finding features in images:

1. Harris corner detector [1],
2. Shi & Tomasi corner detector [2],
3. ORB feature detector [3],
4. ArUco marker detector [4].

You will create a C++ terminal application with a command-line interface that can open a video or image file to find and display image features. To achieve this, you will use OpenCV to do the following:

- Load images data from image and video files with appropriate input checks,
- Modify images with marker and text overlays,
- Display images and associated features, and export them to files.

The lab should be completed within 4 hours. The assessment will be done within your enrolled lab session(s). Once you complete the tasks, call the lab demonstrator to start your assessment.

The lab is worth 5% of your course grade and is graded from 0–5 marks.

LLM Tip

You are strongly encouraged to explore the use of Large Language Models (LLMs), such as GPT, PaLM or LLaMa, to assist in completing this activity. Bots with some of these LLMs are available to use via the UoN Mechatronics Slack team, which you can find a link to from Canvas. If you are are unsure how to make the best use of these tools, or are not getting good results, please ask your lab demonstrator for advice.

1 Command line interface (1 mark)

For the tasks specified in Sections 2–5, a command line interface is used. For this task, the OpenCV command line parser `cv::CommandLineParser` is used. The command line interface enables the user to do the following:

1. Access useful *help* documentation for the application input arguments.
2. Specify the maximum number of features to search for using the `-N` argument.
3. Specify the path to the image or video that the application will process, that can be read using `cv::imread` or `cv::VideoCapture`, respectively.
4. Specify to display the processed images on screen using `cv::imshow` or export the processed image or video using the `-e` or `--export` flags using `cv::imwrite` or `cv::VideoWriter`, respectively. Each output file must be written to the output directory with path format `out/<input-file-stem>_<detector>.<input-file-extension>`.

5. Specify the detection algorithm to be used (`harris`, `shi`, `orb`, `aruco`) using the `-d` or `--detector` argument.

The interface should print relevant information to the console such as image dimensions, the number of features that were detected, and any other information specified in the following tasks.

To begin, prepare the build directory for this lab and change to it as follows:

Terminal

```
nerd@basement:~/MCHA4400/lab2$ cmake -G Ninja -B build && cd build
[cmake-ing intensifies]
nerd@basement:~/MCHA4400/lab2/build$
```

2 Harris corner detector (1 mark)

Use the OpenCV implementation of the Harris [1] corner detector, `cv::cornerHarris`, to find corners of interest in the image. Store all pixels and their associated Harris scores, which are above a chosen threshold, inside a vector[1]. Adjust algorithm parameters to ensure a usable result.

1. Plot all the features that are above some texture based threshold using `cv::circle`.
2. Print a sorted list of the texture values and indices of the `N` most highly textured features.
3. Plot the indices of the `N` most highly textured features using `cv::putText`.

Build and run the `lab2` program with the Harris corner detector to generate Figure 1 as follows:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ninja && ./lab2 ../data/box.jpg -d=harris -N=4
[Ninja-ing intensifies]
Using harris feature detector
Image width: 1280
Image height: 1024
Features requested: 4
Features detected: 143
    idx: 0    at point:  (356,422)    Harris Score: 0.00187296 NOTE: Your scores may be different
    idx: 1    at point:  (356,423)    Harris Score: 0.0017795
    idx: 2    at point:  (520,203)    Harris Score: 0.00147312
    idx: 3    at point:  (910,362)    Harris Score: 0.00147124
```

Experiment with the detector using the files included in the `data` directory and then run the following to export some of the results to the `out` directory:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=harris -N=15 -e ../data/lab.jpg
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=harris -N=15 -e ../data/descriptor0.JPG
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=harris -N=15 -e ../data/outdoor.MOV
```

[1] Consider using a vector of structures. The structure could comprise of elements `x`, `y`, `score` which contain the pixel coordinates x, y and the score respectively.

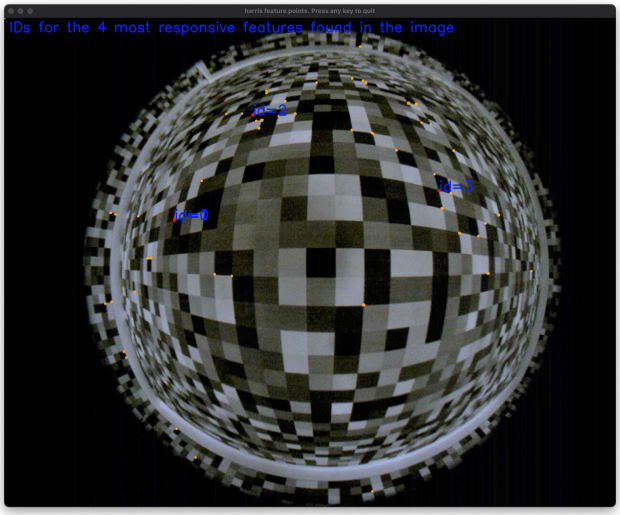

Figure 1: Output from running `./lab2 ../data/box.jpg -d=harris -N=4`

3 Shi & Tomasi corner detector (1 mark)

Implement the Shi & Tomasi [2] corner detector using `cv::cornerMinEigenVal` to find corners of interest in the image. Do **not** use `cv::goodFeaturesToTrack`, as the keypoint scores cannot be extracted to sort features according to their texture. Store all pixels and their associated eigenvalues, which are above a chosen threshold, inside a vector. Adjust thresholds and other algorithm parameters to ensure a usable result.

1. Plot all the features that are above some texture based threshold using `cv::circle`.
2. Print a sorted list of the texture values and indices of the `N` most highly textured features.
3. Plot the indices of the `N` most highly textured features using `cv::putText`.

Experiment with the detector using the files included in the `data` directory and then run the following to export some of the results to the `out` directory:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=shi -N=15 -e ../data/lab.jpg
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=shi -N=15 -e ../data/descriptor0.JPG
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=shi -N=15 -e ../data/outdoor.MOV
```

4 ORB feature detector (1 mark)

Use the OpenCV implementation of the ORB [3] feature detector library to detect and describe features in the image. Create an ORB object using `cv::ORB::create`[2] and then use the `detect` and `compute` member functions on that object to detect features in the image and to compute the ORB descriptors for the set of keypoints, respectively. Adjust algorithm parameters to ensure a usable result.

1. Print ORB descriptor information to the console.

[2]Note that this function returns an OpenCV smart pointer to a new ORB object, i.e., of type `cv::Ptr<cv::ORB>`.

2. Plot the keypoints using `cv::drawKeypoints` with `flags` set to `cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS`.

Build and run the `lab2` program with the ORB feature detector as follows:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ninja && ./lab2 ../data/uonengineering.png -d=orb -N=3
[Ninja-ing intensifies]
Using orb feature detector
Image width: 1290
Image height: 676
Descriptor Width: 32
Descriptor Height: 3
KeyPoint 0    descriptor: [ 45,  45, 208, 195, 205, 121, 206, 207,  67, 178, 188,  10,  11, 105,  95, 175,
    ↪ 89, 202, 194, 122,  94,  98, 110, 184,  47, 221, 133,  63, 247, 255, 218, 134] Note: Your descriptors
    ↪ may be different
KeyPoint 1    descriptor: [ 47, 177, 184, 116, 173, 119, 215, 219,  83, 255, 189,  15,  74, 253, 127, 174,
    ↪ 217,  94, 177,  17, 157, 107, 250, 130,  91, 223,  85,  63, 210, 127, 220, 246]
KeyPoint 2    descriptor: [ 19, 189, 158, 188, 121, 132, 158, 231, 187, 200,  88, 109,  44, 254,  36,  18,
    ↪ 174, 123, 245,   5, 101, 235, 154, 239, 185, 118,  95, 139,  72,  36,  43, 159]
```

Experiment with the detector using the files included in the `data` directory and then run the following to export some of the results to the `out` directory:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=orb -N=15 -e ../data/lab.jpg
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=orb -N=15 -e ../data/descriptor0.JPG
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=orb -N=15 -e ../data/outdoor.MOV
```

5 ArUco marker detector (1 mark)

Use the OpenCV implementation of the ArUco [4] library to find markers with `cv::aruco::ArucoDetector::detectMarkers` in the image using the `cv::aruco::DICT_6X6_250` dictionary.

1. Print a sorted list of the marker corner locations, sorted by the marker identifiers.

2. Plot ArUco markers using `cv::aruco::drawDetectedMarkers`.

LLM Tip

The API for the ArUco module changed significantly with the release of OpenCV 4.7 on 29th Dec 2022. These changes may be more recent than the training cut-off for most available LLMs. The ArUco OpenCV tutorial may be a good reference instead.

Build and run the `lab2` program with the ArUco feature detector as follows:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ninja && ./lab2 ../data/singlemarkersoriginal.jpg -d=aruco
[Ninja-ing intensifies]
Using aruco feature detector
Image width: 640
Image height: 480
    ID: 23  with corners: (298,185) (334,186) (335,212) (297,211)
    ID: 40  with corners: (359,310) (404,310) (409,351) (362,350)
    ID: 62  with corners: (233,273) (190,273) (196,241) (237,241)
    ID: 98  with corners: (427,255) (469,256) (477,289) (434,288)
    ID: 124 with corners: (425,163) (430,186) (394,186) (390,162)
    ID: 203 with corners: (195,155) (230,155) (227,178) (190,178)
```

Experiment with the detector using the files included in the `data` directory and then run the following to export some of the results to the `out` directory:

Terminal

```
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=aruco -e ../data/negativeAdrian.png
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=aruco -e ../data/singlemarkersoriginal.jpg
nerd@basement:~/MCHA4400/lab2/build$ ./lab2 -d=aruco -e ../data/tags.png
```

References

[1] Christopher Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[2] Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[3] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.

[4] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2014.01.005. URL https://www.sciencedirect.com/science/article/pii/S0031320314000235.