CM50248. Visual Understanding 1
— Coursework Specification —

Dr Christian Richardt

2017/2018

1 Overview

There is no examination for this unit; it is assessed on coursework alone.

The coursework for Visual Understanding 1 comprises a total of six exercises, which are split into two parts of three exercises each. The weight of individual exercises varies slightly. The coursework also includes the submission of a detailed report at the end of the semester.

The first part of the coursework covers the topics:

1. Convolution (Section 2)
2. Features (Section 3)
3. Matching (Section 4)

The second part of the coursework covers the topics:

4. Camera calibration (Section 5)
5. Fundamental matrix estimation (Section 6)
6. Triangulation of 3D points (Section 7)

1.1 Deadlines

Your first deadline is on **Monday 13 November 2017 at 8pm**. This will be a live demo of the first three exercises, which is worth 25% of the overall mark. No report is required at this stage, just demonstrate that your code is running. If each of the three exercises is implemented properly, as specified in this document, you will get full marks. Incomplete work will get partial marks. The goal is to make you start your coursework early, and to give you some feedback before the Christmas break.

Your final report about **all** six exercises in this document is due on **Friday 19 January 2018 at noon** – the final course deadline. This report is worth 75% of the overall mark: 25% for your report on the first three exercises (already demoed at the first deadline), and 50% for the last three exercises with supporting code. We will provide summative feedback within one to three weeks.

1.2 Final report

You should hand in a single write-up that gives an account of the entire coursework you have undertaken. Consider this your chance to practise writing, an integral and important part of research as results must be communicated.

The report, written in English, should describe your solutions to all six exercises. Your report should provide:

- An overview of the problem – and solution – that is designed for non-mathematical readers.
- A theoretical account using an appropriate mathematical depth. Proofs are not necessary, instead show you can describe (some aspect of) the world using mathematics.
- A limited literature review. Organise this to support your work; a list of papers and books is often not informative. This is especially important for the more open parts of the coursework.
- A brief account of code you have written. You may include snippets or pseudocode in the document.
- An account of experiments you have conducted. Clearly motivate each one, in particular state the hypothesis the experiment is designed to test. Present your results, discuss them, and draw conclusions. A conclusion should be more than a summary – it should say what you have learned. Experiments should be reproducible by readers.

Write with regard to your reader. Try to keep the arguments flowing in a natural way. Place details into separate appendices.

I recommend you start writing with an introduction, sketched results, and an overall conclusion. Next, add descriptions for each exercise and the literature review. Then go through adding details as necessary.

A highly polished document is not required. You should reserve about 20 to 30 hours to write up, including drawing diagrams, compiling a bibliography, making sure you have included all salient points, and making sure relatively unimportant details are given little weight.

Working code should be submitted too. Provide a 'readme' file to say how to run it.

Submit your report, documented code and any supplementary material you think is relevant on Moodle by **Friday 19 January 2018 at noon**.

The University of Bath rules of plagiarism[1] apply.

1.3 Feedback

You are encouraged to seek feedback throughout the lifetime of the coursework. If you have technical problems, please approach your lab tutor in the first instance. If you want a defined section of coursework marking – or a mark estimate – then see the unit convenor.

[1] http://www.bath.ac.uk/library/help/infoguides/plagiarism.html

2 Convolution (15%)

This exercise is designed to help you use MATLAB by implementing image convolution of greyscale images – a basic image filtering function that is implemented in many computer vision systems (e.g. for edge detection) and most image editing programs such as Photoshop (e.g. for image sharpening).

2.1 Basic convolution [20%]

Implement basic convolution as a MATLAB function

```
fun result = basic_convolution(image, kernel)
```

that takes as input a greyscale `image` (2D matrix) and a filtering `kernel` (2D matrix), and returns the convolved image result as a greyscale image with the same size and datatype as the input image.

Compare the output of your function to the output of MATLAB's built-in function '`conv2`'. Do so qualitatively (that is, compare outputs by eye), and quantitatively by (say) the sum of squared differences.

2.2 Border handling [20%]

Improve your implementation from the previous exercise (name it "`extended_convolution`") by first centring the filtered image (so that the content of the input and filtered images is not shifted between them), and then fill in the border regions by extending/replicating the edge pixels of the image ('clamp-to-edge'). A perfect result will match MATLAB's built-in function "`imfilter`" (with options 'replicate' and 'conv') exactly, i.e. with a sum of squared differences (SSD) of zero.

2.3 Image filtering [10%]

Design and demonstrate 3×3 convolution kernels for:

- computing horizontal, vertical and (any) diagonal image gradients, and
- sharpening an image using unsharp masking.

Write code that computes Gaussian kernels with different standard deviations. Demonstrate and compare your filtering results.

2.4 Exploiting the convolution theorem [50%]

Demonstrate that the convolution theorem holds by implementing convolution using the Fourier transform, and comparing the resulting images to the standard convolution implemented in '`extended_convolution`' above. You can use MATLAB's functions for 2D Fast Fourier Transform (FFT), "`fft2`" and its inverse "`ifft2`".

Write code that measures the CPU run time of convolution, both directly and via the fast Fourier transform:

```
tic; % start clock
% operation to time
t = toc; % stop clock, measure elapsed time
```

Using square kernels, plot the elapsed time as a function of kernel area for each case. Discuss your results and draw a conclusion.

When you are measuring run-time performance, make sure you make reasonable assumptions. Repeat experiments and take averages. You could also use the in-built MATLAB profiler, or the ‘`timeit`’ function.

2.5 Assessment

Demo: Show convolution results (and SSD) with various kernels for your three different convolution functions.

Final report: Submit code and a brief section in your report (2–3 pages) containing your write up. A good report will contain a brief explanation of convolution and the convolution theorem, with citations to texts. Use of LaTeX is encouraged.

3 Features (20%)

This exercise is designed to give practical experience in describing images using features. The goal is to implement detection of Canny edges [Canny, 1986], Harris corners [Harris and Stephens, 1988], and difference-of-Gaussian blobs [Lowe, 2004].

3.1 Canny edge detection [30%]

Implement Canny edge detection [1986] for greyscale images.

Evaluate and discuss the influence of the different parameters of the algorithm.

3.2 Harris corner detection [20%]

Implement Harris corner detection [1988] for greyscale images.

Evaluate and discuss the influence of the different parameters of the algorithm.

3.3 Difference-of-Gaussian interest point detection [50%]

Write a function that constructs a scale space representation of an input greyscale image using difference-of-Gaussians (DoG). Parameters to the function will govern the number and location of levels (octaves). In the DoG scale space, detect local extrema, and refine their locations [Sections 3 and 4 of Lowe, 2004].

DoG interest points are characterised by dark surrounded by light, or vice versa. Draw up some test images that are designed to demonstrate this. For example, make an image with centre-surround objects at a variety of scales. Show your interest point detector detects the objects at the appropriate scale.

3.4 Assessment

Demo: Demonstrate and visualise your results for various parameter values.

Final report: Hand in code and a write-up, as in the previous task.

4 Matching (15%)

In this exercise, you will match features to find pairwise correspondences, and use these correspondences to estimate the homograpy between two images. First, the homography will estimated from the minimal amount of input, which is four pairs of corresponding points. Then, the homography estimation will be robustified using RANSAC (RANdom SAmple Consensus), which can cope with a large amount of incorrect correspondences (i.e. outliers).

4.1 Bruce force matching [20%]

Start by implementing a brute-force feature matcher that finds corresponding pairs of features by comparing descriptor distances. For this, you can use MATLAB's built-in feature detection and description functions, e.g. for SURF.

Discard correspondences that are not mutually the best match, and apply the ratio test to filter out non-unique features.

Visualise the correspondences, for example by showing both input images side-by-side and drawing lines between corresponding points.

4.2 Homography estimation [20%]

Implement a function to estimate a homography from four pairs of corresponding image points. The input correspondences can come from your matcher from the previous section, you can define them manually, or use any of the feature detection functions in MATLAB.

4.3 Visualising the homography [30%]

Use the homography you estimate between two images to transfer image points from one image to the other.

Next, use the homogragraphy for warping one image to align with the other, using backward mapping with bilinear interpolation.

4.4 Homography estimation with RANSAC [30%]

Implement the RANSAC model fitting procedure [Fischler and Bolles, 1981] to robustly estimate a homography between two images from noisy input correspondences.

Design ground-truth images to test your matcher; these should include some distractors. Use the ground-truth images in experiments designed to measure the reliability of the matcher.

4.5 Assessment

Demo: Demonstrate and visualise your results.

Final report: Hand in code and a write-up, as in the previous task.

5 Camera calibration (15%)

Using a target image, estimate the intrinsic parameters (lens distortion and intrinsic camera matrix) of a camera, for example using Zhang's method [2000].

Consider how to show that the intrinsic parameters are consistently estimated.

5.1 Assessment

Write down your method and explain your reasoning.

Hand in code and a lab report, no demo required.

The report should be sufficient to allow another student to reproduce your work.

The code is supporting evidence that you have completed a practical assignment.

The same applies for the next two tasks.

6 Fundamental matrix estimation (20%)

6.1 Fundamental matrix [30%]

Estimate the fundamental matrix from point correspondences.

When writing up, explain why estimating $\mathbf{F}$ can be difficult, and describe methods to address the problems.

6.2 RANSAC [30%]

Refine your algorithm using RANSAC [Fischler and Bolles, 1981].

Evaluate how this improves on your first implementation.

6.3 Essential matrix decomposition [40%]

Derive the essential matrix from the estimated fundamental matrix, and decompose it to obtain the extrinsic calibration between two cameras.

Visualise and evaluate the accuracy of your solution.

7 Triangulation of 3D points (15%)

Implement a suitable algorithm for triangulating the 3D position of a point from its projection in two images.

Apply your triangulation to all corresponding points between two images to obtain a sparse point cloud reconstruction of the scene.

Appraise the reconstruction method you use, preferably using empirically gathered evidence.

8 Bibliography

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. ISSN 0001-0782. doi: 10.1145/358669.358692.

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 147–151, 1988. doi: 10.5244/C.2.23.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004. doi: 10.1023/B:VISI.0000029664.99615.94.

Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000. ISSN 0162-8828. doi: 10.1109/34.888718. URL `http://research.microsoft.com/~zhang/Calib/`.