2020 Computer Vision
Assignment 1: Disparity estimation

1 Introduction

In this practical you will research, implement and test some image filtering operations for the purpose of estimating the disparity between two images. Image filtering is a fundamental step in many computer vision tasks, and you will find it useful to have a firm grasp of how it works. Furthermore, practical experience with image matching will give you insight into and appreciation of the challenges of matching between images, which is a fundamental precursor to a number of problems in computer vision. The main aims of the practical are:

- to understand how images are stored and processed in memory;
- to gain exposure to the challenges and limitations of finding correspondences between images;
- to test your intuition about image filtering by running some experiments; and
- to report your results in a clear and concise manner.

This assignment relates to the following ACS CBOK areas: abstraction, design, hardware and software, data and information, HCI and programming.

2 Setup

In this assignment, you will write a report and some code. While you are required to submit both, you will be graded based on your report only. There are many software libraries that can perform image filtering, but we recommend using either:

- Matlab: installed on University computers, and available on your home computer via ADAPT; or

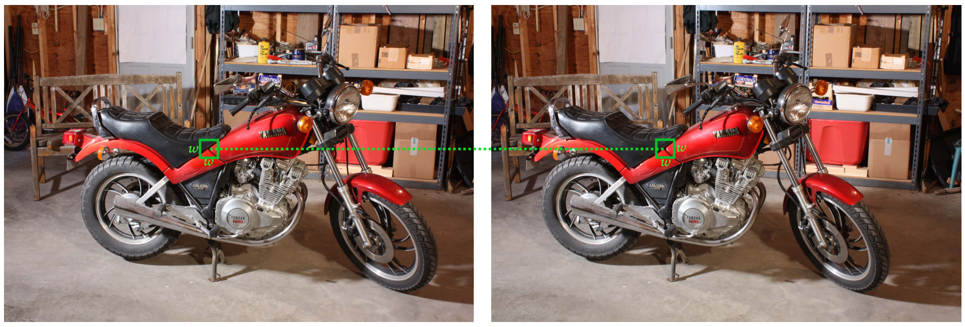

Figure 1: A pair of stereo images from the Middlebury Stereo dataset illustrating one patch correspondence; ω denotes the size of each patch that is used for per-pixel comparison between the two images.

- Python and the OpenCV library `https://opencv.org/`: which is free, but not pre-installed on the University computers.

Either option will serve you well, and both have inbuilt image processing functions which you are free to use. If you are not familiar with the software you will be using, allow yourself time to learn it! The assignments in this course require a relatively small amount of coding to complete, but you will need to understand what you are doing and the software you are using.

3 Patch match for disparity estimation

A fundamental task in computer vision involves finding corresponding points between two or more images. Correspondences identify the *same* scene-point that is visible in two (or more) images. This is illustrated in Figure 1 which depicts the connection of the same point on the motor-cycle between the two views. Correspondences are used in a number of contexts, including image interpolation and 3D reconstruction

Because the two images depict a similar scene from different view-points, the corresponding patches do not share the same x, y pixel co-ordinates. Instead, they can be represented as a *disparity map* $d : [x, y] \to \Re^2$ where $d(x, y) = [\delta_x, \delta_y]$ and the pixel $[x, y]$ in the first image is said to be in correspondence with the pixel $[x', y'] = [x + \delta_x, y + \delta_y]$ in the second. This representation is illustrated in Figure 2, where the two images from Figure 1 are composited to illustrate that corresponding scene-points do not share corresponding pixel co-ordinates. However, the correspondence between the scene point $\mathbf{p}$ can be represented by a displacement vector $d(\mathbf{p})$ to yield the corresponding pixel co-ordinates in the second image.

The disparity map can be estimated by finding, for every point in the first image, the point in the second image with the greatest similarity. Given two points $[x, y]$ and $[x', y']$ for the first and second images respectively, one way

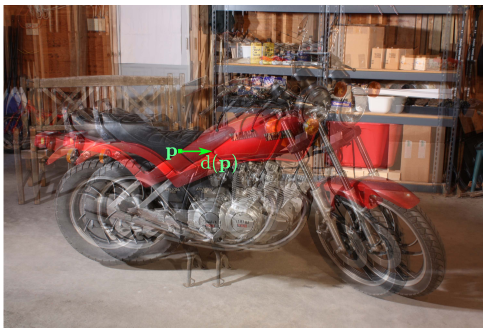

Figure 2: Correspondences can be represented by a displacement vector for every pixel in one image. Here, the disparity $d(\mathbf{p})$ for pixel $\mathbf{p}$ in the left image is added to the co-ordinates of $\mathbf{p}$ to give the pixel co-ordinates in the right image of Figure 1.

to measure their image similarity is given by the sum of squared differences:

$$s(\mathbf{p}, \mathbf{p}') = \sum_{i=-\omega}^{\omega} \sum_{j=-\omega}^{\omega} \big(\mathcal{I}(\mathbf{p} + [i, j]) - \mathcal{I}'(\mathbf{p}' + [i, j])\big)^2 \tag{1}$$

for some particular patch size ω. Note that this score is 0 if every corresponding pixel in the two patches are identical, and therefore maximising similarity corresponds, in this instance, to minimising $s(\cdot)$:

$$d(\mathbf{p}) = \arg\min_{\mathbf{p}'} \big(s(\mathbf{p}, \mathbf{p}')\big) - \mathbf{p}. \tag{2}$$

Exhaustively searching every pair of pixels between two images is extremely time-consuming, and a number of approaches have been described to quickly find approximate solutions.

The 'Patch-Match Algorithm'[1] is one such approach that exploits spatial coherence by iteratively propagating and then refining hypothesised correspondences. An overview of the algorithm is as follows:

1. initialise $d(\cdot)$ with random displacements
2. for every pixel $\mathbf{p}$:
   (a) $d(p) \leftarrow$ offset with maximum similarity selected from those in the neighbourhood of $\mathbf{p}$
   (b) update $d(\mathbf{p})$ by searching a sequence of randomly chosen points from a diminishing window around $d(\mathbf{p})$

[1]PatchMatch: a randomized correspondence algorithm for structural image editing, C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, in ACM SIGGRAPH, pages 1–11, July 2009

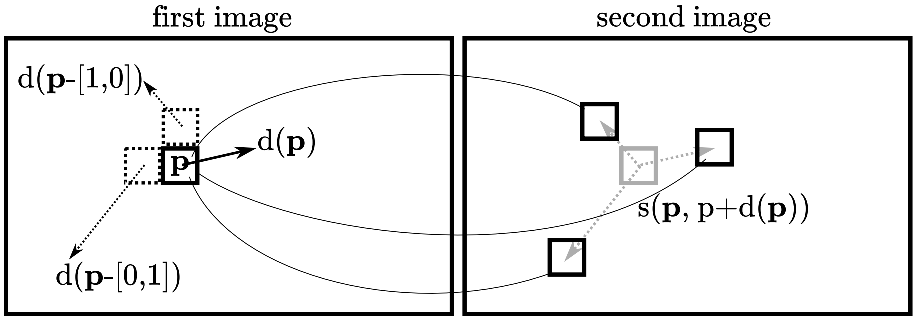

Figure 3: Displacement vectors are propagated by considering neighbourhood offsets. Note that only the pixels above and to the left of **p** are considered because the sweep direction begins, in this instance, from the top left corner.

3. repeat (2) until convergence

There are implementation details in how the offsets are chosen from the neighbourhood around a given pixel, related to the order in which the pixels are updated, and how the random search updates the displacement; these details will be described below.

PatchMatch begins with a randomly initialised displacement field $d(\cdot)$ which assigns a random correspondence for every pixel in the first image. The algorithm improves this estimate by processing each pixel in turn and, under the assumption that a pixel's displacements are likely to be similar, considers the displacements of its neighbours to determine if their displacement is superior. For a given pixel **p**, the displacement is propagated by

$$d(\mathbf{p}) = \arg\min_{d' \in \mathcal{D}} s(\mathbf{p}, \mathbf{p} + d') \tag{3}$$

where $\mathcal{D} = \{d(\mathbf{p}), d(\mathbf{p} - [1,0]), d(\mathbf{p} - [0,1])\}$ include the displacements above and to the left of pixel **p**. This process is illustrated in Figure 4.

The displacement is then refined by searching a sequence of random offsets δ, greedily updating the base displacement if the similarity improves:

$$d(\mathbf{p}) \leftarrow \begin{cases} d(\mathbf{p}) + \delta_i & \text{if } s(\mathbf{p}, \mathbf{p} + d(\mathbf{p}) + \delta_i) < s(\mathbf{p}, \mathbf{p} + d(\mathbf{p})) \\ d(\mathbf{p}) & \text{otherwise.} \end{cases} \tag{4}$$

The displacement δ_i is given by $\delta_i = \kappa\alpha^i\mathcal{R}_i$ where $\mathcal{R}$ is a random variable from the uniform distribution in $[-1, +1] \times [+1, -1]$, $\alpha = 0.5$ to control the speed with which the search area decreases, and κ is a suitably large window size. The process of updating the search window size and location is illustrated in Figure 4.

The algorithm proceeds by sweeping the image, propagating and searching pixel disparities as it goes, until convergence. In practice, the sweep

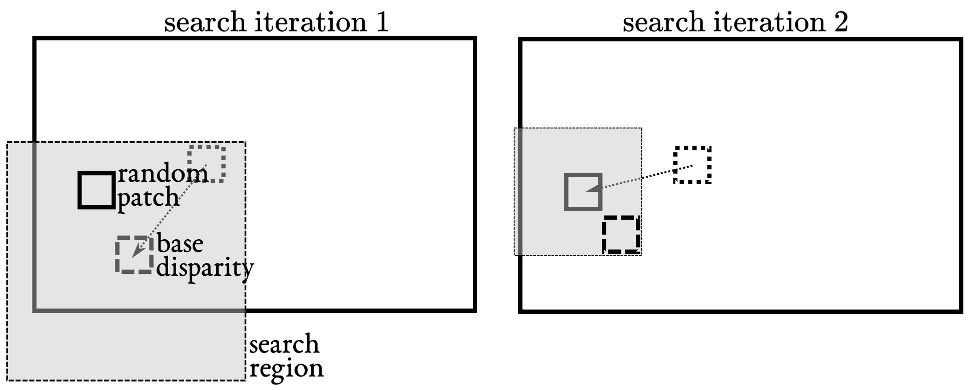

Figure 4: Displacements are updated by randomly searching within a diminishing window around the current best displacement.

direction is alternated in each pass. The set $\mathcal{D}$ in (3) considers disparities above and to the left of a given pixel when the search direction is from the top-left to the bottom-right of the image; but on alternate passes—when the search begins from the bottom-right—the set $\mathcal{D}$ is defined by the disparities to the right and below the given pixel.

This practical will ask you to implement and experiment with the Patchmatch algorithm to understand the challenges of finding correspondences between two images. Please note that the programming aspect is not particularly onerous, and the aim of this assignment is to run experiments to test your hypothesis about the performance of patch match against a variety of different types of image and how the underlying parameters affects performance. Accordingly, we expect you to implement *this* particular variant described above rather than use any existing PatchMatch implementation to ensure that you are able to perform the required experiments and analysis with respect to the algorithm described here.

The key to the report is to test your hypothesis about the behaviour of the system and to support your conclusions with evidence. The aim of the practical is to understand the challenges with estimating correspondences, rather than simply implement the PatchMatch algorithm. When conducting your experiments, clearly articulate what hypothesis you are testing by explaining why a particular patch or stereo pair was chosen and try to give some insight into why a particular outcome was observed. Finally, please be assured that computing correspondences is a challenging problem, and *you should anticipate that your results to be incorrect for some cases*, so please do not be discouraged if your results are not perfect. Discovering where it does and does not work is the point of the practical!

3.1 Task 1

First, read through the rest of the instructions for this assignment, and then return to this section. Create or download several a minimum of three pairs of test images that you will experiment with. The Middlebury stereo dataset `http://vision.middlebury.edu/stereo/data` has a large number of stereo pairs. (The images from Figure 1 is from this dataset.) This dataset includes image pairs with ground truth disparity which may be useful in understanding the problem and to determine if your implementation is working. Many of the images from the Middlebury dataset are high resolution and therefore will take a non-trivial amount of time to evaluate, so you might want to consider down-sampling the images to reasonable resolution before conducting your experiments.

Note that the Middlebury dataset only has 'narrow-baseline' stereo pairs where the camera has not moved significantly between each frame. You should consider experimenting with two images that are not of identical scenes, e.g. a stereo pair where the camera has moved significantly, the lighting or exposure has changed, or the scene changes between frames. One excellent source of wide-baseline image pairs is carefully selected frames of an image sequence taken by a moving camera! Experimenting with different classes of image pairs will help gain insight into the problems of computing correspondences, and help enrich your report.

Your report should:

1. include the pairs of images and;
2. explain why that pair was chosen: ie. what hypothesis in the following two tasks do you think it will help illustrate?

3.2 Task 2

To explore how the similarity score evaluates candidate points, manually select a point in one image and exhaustively compute (1) for every pixel in the second image. You can visualise this as a colour-coded image where the colour (or intensity) at pixel $[x, y]$ corresponds to the similarity between the patch around $[x, y]$ in the second image against your selected patch in the first image. For each score distribution, identify the point that minimises (1) and compare it to the scene-point that you expect to match.

Experiment with a number of different stereo pairs–particularly of scenes where the exposure/lighting changes between frames and a variety of different scene points. The Middlebury dataset is an excellent resource because you can compare how the score distribution is affected (or not?) by the change in exposure. In performing your experiments, consider both the sum of squared differences as described in (1) and a variant where *both* images are transformed by subtracting their mean and dividing by the standard deviation for each channel of each image independently

$$\hat{\mathcal{I}} = \frac{\mathcal{I} - \bar{\mathcal{I}}}{\sqrt{\sum(\mathcal{I} - \bar{\mathcal{I}})^2}} \tag{5}$$

where $\bar{\mathcal{I}}$ is the mean of image $\mathcal{I}$.

While the easiest way to select a source point is to use an image editor (e.g. GIMP `www.gimp.org`) to identify the pixel co-ordinates, be aware that different image processing libraries may use the different co-ordinate systems: we recommend you carefully check that the source patch used is the one you intended.

Consider the following questions in your report:

1. Describe how the score distribution is affected by the choice of scene-point. Do all scene points have the same characteristic? If not, why not; it so, why did you expect that to be the case?

2. Describe how the score distribution is affected by change in patch size. Is there a single best patch size for all scene-points? Are there cases where a smaller or larger patch size have different advantages or disadvantages?

3. Did the image transform in (5) affect the results in any of the experiments? Why, or why not?

3.3 Task 3

Implement the Patch Match algorithm described above to compute dense correspondences. Visualise the disparity as a grey-scale image, where the intensity corresponds to the magnitude of the estimated disparity. Finally, reconstruct the source image using the disparity map and *reverse mapping* pixels from the second image. Reverse mapping involves assigning the colour at pixel $\mathbf{p}$ from the pixel at $\mathbf{p} + d(\mathbf{p})$ in the second image (although there are other approaches to this, too).

Optionally, you can visualise the correspondences between the two images by plotting them side-by-side and drawing a *sparse* set of lines between correspondences, similar to the illustration in Figure 1. This might give you some greater insight into whether PatchMatch is correctly identifying corresponding pixels.

In your report, consider the following questions:

1. Did the propagation and random search improve, or otherwise affect, the estimated correspondence for the points you experimented with in Task 2? Why do you think that the results were or were not different?

2. What relationship—if any—can you identify between the disparity magnitude and the scene structure? What discrepancies can you identify in the disparity image and how can they be explained?

3. Compare your reconstructed image results to the original input. Is the source image exactly recreated correctly, or have errors been introduced: and if so, why?

3.4 Task 4 (for Masters' students only)

Run a median filter on the disparity image and observe how the disparity is affected by varying the window size. In your report, consider

1. What are the advantages of the median filter, and in what cases does it adversely affect the results?

4 Assessment

Hand in your report and supporting code/images via the MyUni page. Upload two files:

1. report.pdf, a PDF version of your report. The report should answer each task in a separate section. Make sure you answer each question listed at the end of each task, and include results (images or figures) to back up your answers.

2. code.zip, a zip file containing your code and test images. Make sure you include all the code you have written and your test images. Your mark will be based on your report - the code will just be used to check your work.

The prac is worth 10% of your overall mark for the course. It is due on **Monday March 30 at 11.59pm.**

John Bastian
3rd March 2020